



# 脚本语法和图表函数

---

Qlik Sense®

2.1.1

Copyright © 1993-2015 QlikTech International AB. 保留所有权利。



版权所有 © 1993-2015 QlikTech International AB。保留所有权利。

Qlik®、QlikTech®、Qlik Sense®、QlikView®、Sense® 和 Qlik 徽标是在多个国家/地区注册的商标，或另供 QlikTech International AB 用作商标。本文引用的其他商标是其各自所有者的商标。

<b>1 什么是 Qlik Sense?</b>	<b>18</b>
1.1 在 Qlik Sense 中可以做什么?	18
1.2 Qlik Sense 如何工作?	18
应用模式	18
关联体验	18
协作性和可移动性	18
1.3 如何部署 Qlik Sense?	18
Qlik Sense Desktop	18
Qlik Sense Server	19
1.4 如何管理 Qlik Sense 站点	19
1.5 扩展 Qlik Sense 并根据自己的目的进行调整	19
构建扩展程序和插件	19
构建客户端	19
构建服务器工具	19
连接到其他数据源	19
<b>2 脚本语法</b>	<b>20</b>
2.1 脚本语法简介	20
2.2 什么是 Backus-Naur 形式?	20
2.3 脚本语句和关键字	21
脚本控制语句	21
脚本控制语句概述	21
Call	23
Do..loop	24
Exit script	25
For..next	25
For each..next	26
If..then..elseif..else..end if	29
Sub..end sub	30
Switch..case..default..end switch	31
脚本前缀	32
脚本前缀概述	32
Add	35
Buffer	36
Bundle	37
Concatenate	38
Crosstable	39
First	39
Generic	40
Hierarchy	40
HierarchyBelongsTo	41
Image_Size	42
Info	42
Inner	43
IntervalMatch	44
Join	46

Keep .....	47
Left .....	47
Mapping .....	49
NoConcatenate .....	50
Outer .....	50
Replace .....	51
Right .....	52
Sample .....	53
Semantic .....	54
Unless .....	54
When .....	55
脚本常规语句 .....	56
脚本常规语句概述 .....	56
Alias .....	61
Binary .....	62
Comment field .....	63
Comment table .....	63
Connect .....	64
Declare .....	66
设置新的字段定义 .....	66
重复使用现有的字段定义 .....	67
Derive .....	68
Direct Query .....	69
Direct Discovery 字段列表 .....	71
Directory .....	73
Disconnect .....	74
Drop field .....	74
Drop table .....	74
Execute .....	75
FlushLog .....	76
Force .....	76
Load .....	77
格式规格项目 .....	84
字符集 .....	84
表格格式 .....	85
Delimiter .....	85
No eof .....	86
Labels .....	87
Header is .....	87
Record is .....	88
Quotes .....	88
XML .....	89
KML .....	89
Let .....	90
Map .....	90
NullAsNull .....	91

NullAsValue .....	91
Qualify .....	92
Rem .....	93
Rename field .....	93
Rename table .....	94
Search .....	95
Section .....	96
Select .....	96
Set .....	98
Sleep .....	99
SQL .....	99
SQLColumns .....	100
SQLTables .....	101
SQLTypes .....	101
Star .....	102
Store .....	103
Tag .....	104
Trace .....	105
Unmap .....	105
Unqualify .....	106
Untag .....	106
工作目录 .....	107
Qlik Sense Desktop 工作目录 .....	107
Qlik Sense 工作目录 .....	107
2.4 在数据加载编辑器中使用变量 .....	107
删除变量 .....	108
将变量值加载为字段值 .....	108
变量计算 .....	109
系统变量 .....	109
系统变量概述 .....	110
HidePrefix .....	112
HideSuffix .....	112
Include .....	112
OpenUrlTimeout .....	113
StripComments .....	113
Verbatim .....	114
值处理变量 .....	114
值处理变量概述 .....	114
NullDisplay .....	115
NullInterpret .....	115
NullValue .....	115
OtherSymbol .....	115
数字解释变量 .....	116
数字解释变量概述 .....	116
货币格式 .....	116
数字格式 .....	117

时间格式 .....	117
BrokenWeeks .....	118
DateFormat .....	119
DayNames .....	119
DecimalSep .....	119
FirstWeekDay .....	119
LongDayNames .....	120
LongMonthNames .....	120
MoneyDecimalSep .....	120
MoneyFormat .....	120
MoneyThousandSep .....	121
MonthNames .....	121
ReferenceDay .....	121
ThousandSep .....	121
TimeFormat .....	122
TimestampFormat .....	122
Direct Discovery 变量 .....	122
Direct Discovery 系统变量 .....	122
Teradata 查询分级变量 .....	123
Direct Discovery 字符变量 .....	124
Direct Discovery 数字解释变量 .....	125
错误变量 .....	125
错误变量概述 .....	126
ErrorMode .....	126
ScriptError .....	127
ScriptErrorCount .....	128
ScriptErrorList .....	128
2.5 脚本表达式 .....	128
<b>3 可视化表达式 .....</b>	<b>130</b>
3.1 定义聚合范围 .....	130
3.2 集合的语法 .....	132
3.3 集合修饰符 .....	132
基于另一个字段 .....	132
基于元素集(修饰符中的字段值列表) .....	133
强制排除 .....	133
集合修饰符和集合运算符 .....	134
集合修饰符使用赋值和默认集合运算符 .....	134
集合修饰符和高级搜索 .....	135
集合修饰符和货币符号扩展 .....	135
集合修饰符和默认字段值定义 .....	135
3.4 可视化表达式和聚合语法 .....	136
图表表达式的一般语法 .....	136
聚合的一般语法 .....	136
<b>4 运算符 .....</b>	<b>138</b>
4.1 位运算符 .....	138

4.2 逻辑运算符 .....	139
4.3 数字运算符 .....	139
4.4 关系运算符 .....	139
4.5 字符串运算符 .....	140
<b>5 脚本和图表表达式中的函数 .....</b>	<b>142</b>
5.1 聚合函数 .....	142
在数据加载脚本中使用聚合函数 .....	142
在图表表达式中使用聚合函数 .....	142
Aggr - 图表函数 .....	142
基本聚合函数 .....	144
基本聚合函数概述 .....	144
数据加载脚本中的基本聚合函数 .....	144
图表表达式中的基本聚合函数 .....	145
FirstSortedValue .....	146
FirstSortedValue - 图表函数 .....	148
Max .....	149
Max - 图表函数 .....	150
Min .....	152
Min - 图表函数 .....	153
Mode .....	155
Mode - 图表函数 .....	156
Only .....	158
Only - 图表函数 .....	159
Sum .....	160
Sum - 图表函数 .....	161
计数器聚合函数 .....	163
数据加载脚本中的计数器聚合函数 .....	163
图表表达式中的计数器聚合函数 .....	163
Count .....	164
Count - 图表函数 .....	165
MissingCount .....	167
MissingCount - 图表函数 .....	168
NullCount .....	170
NullCount - 图表函数 .....	170
NumericCount .....	171
NumericCount - 图表函数 .....	173
TextCount .....	174
TextCount - 图表函数 .....	175
财务聚合函数 .....	177
数据加载脚本中的财务聚合函数 .....	177
图表表达式中的财务聚合函数 .....	177
IRR .....	178
IRR - 图表函数 .....	179
NPV .....	180
NPV - 图表函数 .....	181

XIRR .....	183
XIRR - 图表函数 .....	183
XNPV .....	185
XNPV - 图表函数 .....	186
统计聚合函数 .....	187
数据加载脚本中的统计聚合函数 .....	187
图表表达式中的统计聚合函数 .....	189
Avg .....	192
Avg - 图表函数 .....	193
Correl .....	195
Correl - 图表函数 .....	196
Fractile .....	198
Fractile - 图表函数 .....	199
Kurtosis .....	201
Kurtosis - 图表函数 .....	202
LINEST_B .....	204
LINEST_B - 图表函数 .....	205
LINEST_DF .....	206
LINEST_DF - 图表函数 .....	206
LINEST_F .....	207
LINEST_F - 图表函数 .....	208
LINEST_M .....	209
LINEST_M - 图表函数 .....	210
LINEST_R2 .....	211
LINEST_R2 - 图表函数 .....	212
LINEST_SEB .....	213
LINEST_SEB - 图表函数 .....	213
LINEST_SEM .....	214
LINEST_SEM - 图表函数 .....	215
LINEST_SEY .....	216
LINEST_SEY - 图表函数 .....	217
LINEST_SSREG .....	218
LINEST_SSREG - 图表函数 .....	219
LINEST_SSRESID .....	220
LINEST_SSRESID - 图表函数 .....	220
Median .....	221
Median - 图表函数 .....	222
Skew .....	224
Skew - 图表函数 .....	225
Stdev .....	227
Stdev - 图表函数 .....	228
Sterr .....	230
Sterr - 图表函数 .....	231
STEYX .....	233
STEYX - 图表函数 .....	234
如何使用 linest 函数的示例 .....	236



加载样本数据 .....	236
显示数据加载脚本计算的结果 .....	237
创建 linest 图表函数可视化内容 .....	237
统计检验函数 .....	238
卡方检验函数 .....	238
T 检验函数 .....	238
Z 检验函数 .....	238
卡方检验函数 .....	239
Chi2Test_chi2 .....	239
Chi2Test_df .....	240
Chi2Test_p - 图表函数 .....	241
T 检验函数 .....	242
TTest_conf .....	245
TTest_df .....	246
TTest_dif .....	247
TTest_lower .....	248
TTest_sig .....	249
TTest_sterr .....	250
TTest_t .....	250
TTest_upper .....	251
TTestw_conf .....	252
TTestw_df .....	253
TTestw_dif .....	254
TTestw_lower .....	255
TTestw_sig .....	256
TTestw_sterr .....	257
TTestw_t .....	258
TTestw_upper .....	259
TTest1_conf .....	259
TTest1_df .....	260
TTest1_dif .....	261
TTest1_lower .....	262
TTest1_sig .....	262
TTest1_sterr .....	263
TTest1_t .....	264
TTest1_upper .....	265
TTest1w_conf .....	265
TTest1w_df .....	266
TTest1w_dif .....	267
TTest1w_lower .....	268
TTest1w_sig .....	268
TTest1w_sterr .....	269
TTest1w_t .....	270
TTest1w_upper .....	271
Z 检验函数 .....	272
ZTest_z .....	273

ZTest_sig .....	274
ZTest_dif .....	275
ZTest_sterr .....	275
ZTest_conf .....	276
ZTest_lower .....	277
ZTest_upper .....	278
ZTestw_z .....	279
ZTestw_sig .....	279
ZTestw_dif .....	280
ZTestw_sterr .....	281
ZTestw_conf .....	282
ZTestw_lower .....	283
ZTestw_upper .....	284
统计检验函数示例 .....	285
如何在图表中使用 chi2-test 函数的示例 .....	285
如何在数据加载脚本中使用 chi2-test 函数的示例 .....	287
创建典型的 t-test 报告 .....	289
如何使用 z-test 函数的示例 .....	292
字符串聚合函数 .....	293
数据加载脚本中的字符串聚合函数 .....	294
图表中的字符串聚合函数 .....	294
Concat .....	295
Concat - 图表函数 .....	296
FirstValue .....	298
LastValue .....	299
MaxString .....	300
MaxString - 图表函数 .....	301
MinString .....	302
MinString - 图表函数 .....	303
组合维度函数 .....	305
ValueList - 图表函数 .....	305
ValueLoop - 图表函数 .....	307
嵌套聚合函数 .....	307
带 TOTAL 限定符的嵌套聚合函数 .....	308
5.2 颜色函数 .....	308
预定义颜色函数 .....	310
ARGB .....	311
RGB .....	311
HSL .....	312
5.3 条件函数 .....	312
条件函数概述 .....	312
alt .....	313
class .....	314
if .....	315
match .....	316

mixmatch .....	316
pick .....	316
wildmatch .....	317
5.4 计数函数 .....	317
计算函数概述 .....	317
autonumber .....	318
autonumberhash128 .....	321
autonumberhash256 .....	323
IterNo .....	325
RecNo .....	325
RowNo .....	327
RowNo - 图表函数 .....	328
5.5 日期和时间函数 .....	330
日期和时间函数概述 .....	330
时间的整数表达式 .....	330
Timestamp 函数 .....	331
Make 函数 .....	331
其他日期函数 .....	332
Timezone 函数 .....	332
设置时间函数 .....	333
In... 函数 .....	333
Start ... end 函数 .....	334
Day numbering 函数 .....	337
addmonths .....	338
addyears .....	338
age .....	339
converttolocaltime .....	340
day .....	343
dayend .....	343
daylightsaving .....	344
dayname .....	344
daynumberofquarter .....	346
daynumberofyear .....	347
daystart .....	348
firstworkdate .....	350
GMT .....	351
hour .....	351
inday .....	352
indaytotime .....	353
inlunarweek .....	354
inlunarweektodate .....	356
inmonth .....	358
inmonths .....	359
inmonthstodate .....	361
inmonthtodate .....	363

---

inquarter .....	364
inquartertoday .....	366
inweek .....	367
inweektoday .....	369
inyear .....	371
inyeartoday .....	373
lastworkdate .....	375
localtime .....	376
lunarweekend .....	377
lunarweekname .....	378
lunarweekstart .....	380
makedate .....	381
maketime .....	382
makeweekdate .....	382
minute .....	383
month .....	383
monthend .....	384
monthname .....	385
monthsend .....	386
monthsname .....	388
monthsstart .....	390
monthstart .....	392
networkdays .....	393
now .....	394
quarterend .....	395
quartername .....	397
quarterstart .....	398
second .....	399
setdateyear .....	400
setdateyearmonth .....	401
timezone .....	402
today .....	402
UTC .....	403
week .....	403
weekday .....	404
weekend .....	405
weekname .....	406
weekstart .....	408
weekyear .....	410
year .....	411
yearend .....	411
yearname .....	412
yearstart .....	414
yeartoday .....	416
5.6 指数和对数函数 .....	417
5.7 字段函数 .....	418

---

计数函数 .....	418
字段和选择项函数 .....	418
GetAlternativeCount - 图表函数 .....	419
GetCurrentSelections - 图表函数 .....	420
GetExcludedCount - 图表函数 .....	421
GetFieldSelections - 图表函数 .....	422
GetNotSelectedCount - 图表函数 .....	423
GetPossibleCount - 图表函数 .....	424
GetSelectedCount - 图表函数 .....	425
5.8 文件函数 .....	426
文件函数概述 .....	426
Attribute .....	428
ConnectionString .....	435
FileName .....	436
FileDir .....	436
FileExtension .....	436
FileName .....	437
FilePath .....	437
FileSize .....	437
FileTime .....	438
GetFolderPath .....	439
QvdCreateTime .....	440
QvdFieldName .....	440
QvdNoOfFields .....	441
QvdNoOfRecords .....	442
QvdTableName .....	443
5.9 财务函数 .....	444
财务函数概述 .....	444
BlackAndSchole .....	445
FV .....	446
nPer .....	446
Pmt .....	447
PV .....	448
Rate .....	449
5.10 格式函数 .....	450
格式函数概述 .....	450
Date .....	451
Dual .....	452
Interval .....	453
Money .....	454
Num .....	455
Time .....	456
Timestamp .....	456
5.11 一般数字函数 .....	457
常见数字函数概述 .....	457

组合和排列函数 .....	458
模函数 .....	458
奇偶校验函数 .....	459
舍入函数 .....	459
BitCount .....	459
Ceil .....	460
Combin .....	460
Div .....	461
Even .....	461
Fabs .....	462
Fact .....	462
Floor .....	462
Fmod .....	463
Frac .....	463
Mod .....	464
Odd .....	465
Permut .....	465
Round .....	465
Sign .....	466
5.12 地理空间函数 .....	467
地理空间函数概述 .....	467
GeoAggrGeometry .....	468
GeoBoundingBox .....	469
GeoCountVertex .....	469
GeoGetBoundingBox .....	470
GeoGetPolygonCenter .....	470
GeoInvProjectGeometry .....	471
GeoMakePoint .....	471
GeoProject .....	472
GeoProjectGeometry .....	472
GeoReduceGeometry .....	473
5.13 解释函数 .....	473
解释函数概述 .....	474
Date# .....	475
Interval# .....	475
Money# .....	476
Num# .....	477
Text .....	478
Time# .....	479
Timestamp# .....	479
5.14 内部记录函数 .....	480
行函数 .....	480
列函数 .....	481
字段函数 .....	481
透视表函数 .....	482

数据加载脚本中的内部记录函数 .....	483
Above - 图表函数 .....	483
Below - 图表函数 .....	487
Bottom - 图表函数 .....	490
Column - 图表函数 .....	494
Dimensionality - 图表函数 .....	495
Exists .....	496
FieldIndex .....	498
FieldValue .....	499
FieldValueCount .....	500
LookUp .....	502
NoOfRows - 图表函数 .....	503
Peek .....	504
Previous .....	506
Top - 图表函数 .....	508
Secondarydimensionality .....	511
After - 图表函数 .....	511
Before - 图表函数 .....	512
First - 图表函数 .....	513
Last - 图表函数 .....	514
ColumnNo - 图表函数 .....	515
NoOfColumns - 图表函数 .....	515
5.15 逻辑函数 .....	515
5.16 映射函数 .....	516
映射函数概述 .....	516
ApplyMap .....	517
MapSubstring .....	518
5.17 数学函数 .....	519
5.18 NULL 函数 .....	520
NULL 函数概述 .....	520
IsNull .....	521
NULL .....	521
5.19 范围函数 .....	522
基本范围函数 .....	522
计数器范围函数 .....	523
统计范围函数 .....	524
财务范围函数 .....	524
RangeAvg .....	525
RangeCorrel .....	527
RangeCount .....	528
RangeFractile .....	530
RangeIRR .....	532
RangeKurtosis .....	533
RangeMax .....	534
RangeMaxString .....	536

RangeMin .....	537
RangeMinString .....	539
RangeMissingCount .....	540
RangeMode .....	542
RangeNPV .....	544
RangeNullCount .....	545
RangeNumericCount .....	546
RangeOnly .....	548
RangeSkew .....	548
RangeStdev .....	550
RangeSum .....	551
RangeTextCount .....	553
RangeXIRR .....	554
RangeXNPV .....	555
5.20 图表中的排名函数 .....	556
Rank - 图表函数 .....	557
HRank - 图表函数 .....	559
5.21 统计分布函数 .....	561
统计分布函数概述 .....	561
CHIDIST .....	562
CHIINV .....	563
FDIST .....	563
FINV .....	564
NORMDIST .....	565
NORMINV .....	565
TDIST .....	566
TINV .....	567
5.22 字符串函数 .....	567
字符串函数概述 .....	567
Capitalize .....	570
Chr .....	571
Evaluate .....	571
FindOneOf .....	571
Hash128 .....	572
Hash160 .....	572
Hash256 .....	573
Index .....	573
KeepChar .....	574
Left .....	574
Len .....	575
Lower .....	575
LTrim .....	575
Mid .....	576
Ord .....	577
PurgeChar .....	577



Repeat .....	578
Replace .....	578
Right .....	579
RTrim .....	579
SubField .....	580
SubStringCount .....	581
TextBetween .....	582
Trim .....	583
Upper .....	583
5.23 系统函数 .....	583
系统函数概述 .....	583
GetObjectField - 图表函数 .....	585
IsPartialReload .....	586
ProductVersion .....	586
StateName - 图表函数 .....	586
5.24 表格函数 .....	586
表格函数概述 .....	587
FieldName .....	588
FieldNumber .....	589
NoOfFields .....	589
NoOfRows .....	590
5.25 三角函数和双曲函数 .....	590
<b>6 文件系统访问限制 .....</b>	<b>593</b>
当连接到基于文件的 ODBC 和 OLE DB 数据连接时的安全性 .....	593
6.1 标准模式中的限制 .....	593
系统变量 .....	593
常规脚本语句 .....	594
脚本控制语句 .....	595
文件函数 .....	596
系统函数 .....	597
6.2 禁用标准模式 .....	597
Qlik Sense .....	598
Qlik Sense Desktop .....	598
<b>7 Qlik Sense 不支持的 QlikView 函数和语句 .....</b>	<b>599</b>
7.1 Qlik Sense 不支持的脚本语句 .....	599
7.2 Qlik Sense 不支持的函数 .....	599
<b>8 Qlik Sense 不推荐的函数和语句 .....</b>	<b>600</b>
8.1 Qlik Sense 不推荐的脚本语句 .....	600
8.2 Qlik Sense 不推荐的脚本语句参数 .....	600
8.3 Qlik Sense 不推荐的函数 .....	601
ALL 限定符 .....	602

# 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 Server。

### 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 Server

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

### 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。

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。脚本由一系列连续执行的脚本语句和关键字构成。

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。

一般数据库必须通过 Microsoft ODBC 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。

所有脚本语句必须以分号“;”结束。

可通过本节中的主题访问脚本语法的详细说明。

### 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：

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

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

示例:

alias 语句的说明如下：

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

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

下面是正确的语句：

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

下面则是错误的语句：

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

## 2.3 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。

### 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。

#### Syntax:

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

#### 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。

#### 示例 1:

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

#### 示例 2:

```
// Example 2 - List all QV related files on disk
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 ('C:')
```

## 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。

### Syntax:

```
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	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	<b>while</b> 或 <b>until</b> 条件子句在任何 <b>do..loop</b> 语句中必须只能出现一次，即要么在 <b>do</b> 之后，要么在 <b>loop</b> 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 <b>exit do</b> 子句，则脚本执行会转移至表示循环结束的 <b>loop</b> 子句之后的第一个语句。 <b>exit do</b> 子句可通过选择性使用 <b>when</b> 或 <b>unless</b> 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
```

## 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。

### Syntax:

```
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	说明
condition	用于评估 True 或 False 的逻辑表达式。
when / unless	<b>exit script</b> 语句可通过选择性使用 <b>when</b> 或 <b>unless</b> 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。

### Syntax:

```
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	说明
counter	一个变量名。如果 <i>counter</i> 在 <b>next</b> 之后指定，变量名必须与对应的 <b>for</b> 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 <i>counter</i> 变量的第一个值。
expr2	一个表达式，可决定每执行一次循环 <i>counter</i> 变量增加的值。
expr3	一个表达式，可决定每执行一次循环 <i>counter</i> 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。

### 示例 1: 加载文件序列

```
// LOAD files file1.csv..file9.csv
for a=1 to 9
    LOAD * from file$(a).csv;
next
```

### 示例 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。

Syntax:

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	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 <b>var</b> 在 <b>next</b> 之后指定，变量名必须与对应的 <b>for each</b> 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。

Syntax:

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

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。

参数	说明
filelist mask	<p>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。</p> <div>  此参数仅在标准模式下支持库连接。 </div>
dirlist mask	<p>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。</p> <div>  此参数仅在标准模式下支持库连接。 </div>
fieldvaluelist mask	此语法迭代已经加载到 Qlik Sense 的字段值。

**示例 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
```

**示例 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://MyData')
```

**示例 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：

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。

另请：if (第 315 页) (脚本和图表函数)

#### Syntax:

```
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	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。

### 示例 1:

```
if a=1 then
    LOAD * from abc.csv;
    SQL SELECT e, f, g from tab1;
end if
```

### 示例 2:

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

### 示例 3:

```
if x>0 then
    LOAD * from pos.csv;
elseif x<0 then
    LOAD * from neg.csv;
else
    LOAD * from zero.txt;
end if
```

## 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。

### Syntax:

```
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。

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。

### 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。

**示例 1:**

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

**示例 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。

**Syntax:**

```
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	说明
expression	任意表达式。

参数	说明
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 <b>case</b> 子句中都无匹配值，则将执行 <b>default</b> 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
```

## 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 **add** 前缀添加至脚本中的任意 **LOAD**、**SELECT** 或 **map...using** 语句。仅在部分重新加载期间适用。

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

### Bundle

**Bundle** 前缀用于包括外部文件(如图片或声音文件)或连接至字段值的对象，以便使其存储到 qvf 文件中。

```
Bundle [Info] ( loadstatement | selectstatement)
```

### Concatenate



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

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

### Crosstable

**crosstable** 前缀用于将交叉表转换成垂直表。

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

### First

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

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

### Generic

使用 **generic** 前缀可以打开和加载通用数据库。

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

### Hierarchy

**hierarchy** 前缀用于将层次表格转换成在 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)
```

### Image\_Size

此子句与 **Info** 前缀一同使用，用以调整数据库管理系统中的图片大小，使其能够适合于字段。

```
Info [Image_Size(width,height) ] ( loadstatement | selectstatement )
```

### Info

**info** 前缀用于将外部信息(如文本文件、图片或视频)链接到字段值。

```
Info( 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 )
```

### 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

可在显式 **Join** 前缀前面使用 **outer** 前缀以指定外部联接。在外部联接中，两表格之间所有的组合都可生成。由此生成的表格将包含原始数据表格的字段值组合，在原始数据表格中，链接字段值呈现在一个或两个表格中。**outer** 关键字是可选的。

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

### Replace

**replace** 前缀用于删除整个 Qlik Sense 表格，并使用加载或选择的新表格进行替换。

```
Replace[only] (loadstatement | selectstatement | map...usingstatement)
```

### 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 **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 **add** 前缀添加至脚本中的任意 **LOAD**、**SELECT** 或 **map...using** 语句。仅在部分重新加载期间适用。



目前，使用 Qlik Engine API 仅支持部分重新加载。

### Syntax:

```
Add [only] (loadstatement | selectstatement | mapstatement)
```

在部分重新加载 Qlik Sense 表格期间，**add LOAD/add SELECT** 语句会为其生成表格名(前提是此表格存在)，并且附加 **add LOAD/add SELECT** 语句的结果。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。**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	说明
only	可选限定符表示应在正常(非部分)重新加载期间忽视语句。

#### 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 <i>Persons.csv</i> 加载数据，并存储到 Qlik Sense 表格 Tab1 中。 <i>NewPersons.csv</i> 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 <i>NewPersons.csv</i> 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 <i>Persons.csv</i> 加载数据，并存储到 Qlik Sense 表格 Tab1 中。 <i>NewPersons.csv</i> 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 <i>NewPersons.csv</i> 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 <i>Persons.csv</i> 加载数据，并存储到 Qlik Sense 表格 Tab1 中。忽略加载 <i>NewPersons.csv</i> 的语句。  在部分重新加载期间，将会从 <i>NewPersons.csv</i> 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。

#### Syntax:

```
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	说明
incremental	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。注意，incremental 选项只能与 <b>LOAD</b> 语句和文本文件，以及旧数据发生更改或被删除而导致无法使用增量加载的地方！
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。

### 示例 1：

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

### 示例 2：

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

### 示例 3：

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

## Bundle

**Bundle** 前缀用于包括外部文件(如图片或声音文件)或连接至字段值的对象，以便使其存储到 qvf 文件中。

### Syntax:

```
Bundle [Info] ( loadstatement | selectstatement)
```



**Qlik Sense 不支持显示用 *Bundle* 和 *Info* 加载的信息。**

为了保持易移性，可以将尾部文件包括在 .qvf 文件中。为此可以使用 **Bundle** 前缀。束信息文件将在此过程被压缩，但尽管如此，还是会占用此文件和 RAM 的其他空间。因此，在操作此解决方案之前，请务必考虑束文件的大小和数量。

info 前缀可通过图表 info 函数以常规 info 前缀形式，或通过特殊语法 `qmem://fieldname/fieldvalue` 或 `qmem://fieldname/<index>` 以内部文件形式从布局引用，其中 index 是字段值内部索引。

**参数：**

参数	说明
Info	如果一条外部信息(如图片或声音文件)将要连接至字段值，这可以在通过 <b>Info</b> 前缀加载的表格中完成。 当使用 <b>Bundle</b> 时，可以忽略 <b>Info</b> 前缀。

**示例：**

```
Bundle Info LOAD * From flagoced.csv;
Bundle SQL SELECT * from infotable;
```

## Concatenate

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

**Syntax:**

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

串联大体上与 **SQL UNION** 语句相同，但有两点不同：

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

**参数：**

参数	说明
tablename	现有表格的名称。

**示例：**

```
Concatenate LOAD * From file2.csv;
```

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

## Crosstable

**crosstable** 前缀用于将交叉表转换成垂直表。

### Syntax:

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

### 参数:

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

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

### 示例:

```
Crosstable (Month, Sales) LOAD * from ex1.csv;
Crosstable (Month,Sales,2) LOAD * from ex2.csv;
Crosstable (A,B) SELECT * from table3;
```

## 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。

### Syntax:

```
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 可包含在括号中(如 (n))，但这并不是强制要求。

### 示例：

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

## Generic

使用 **generic** 前缀可以打开和加载通用数据库。

### Syntax:

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

通过 **generic** 语句加载的表格不可自动连接。

### 示例：

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

## Hierarchy

**hierarchy** 前缀用于将层次表格转换成在 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。

### Syntax:

```
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	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 <b>ParentName</b> 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 <b>NodeName</b> 。
PathName	用于命名新建 <b>Path</b> 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 <b>Path</b> 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。
Depth	用于命名新建 <b>Depth</b> 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) LOAD
    NodeID,
    ParentID,
    NodeName,
    Attribute
    From data.xls (biff, embedded labels, table is [Sheet1$]);
```

**HierarchyBelongsTo**

此前缀用于将层次表格转换成在 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。

**Syntax:**

```
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	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 <b>DepthDiff</b> 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, ParentID, Node, Tree, ParentName) LOAD
    NodeID,
    ParentID,
    NodeName
    From data.xls (biff, embedded labels, table is [Sheet1$]);
```

**Image\_Size**

此子句与 **Info** 前缀一同使用，用以调整数据库管理系统中的图片大小，使其能够适合于字段。

**Syntax:**

```
Info [Image_Size(width,height )] ( loadstatement | selectstatement )
```

**参数：**

参数	说明
width	用像素指定的图像宽度。
height	用像素指定的图像高度。

**示例：**

```
Info Image_Size(122,122) SQL SELECT ID, Photo From infotable;
```

**Info**

**info** 前缀用于将外部信息(如文本文件、图片或视频)链接到字段值。

**Syntax:**

```
Info( loadstatement | selectstatement )
```



*Qlik Sense 不支持显示用 **Bundle** 和 **Info** 加载的信息。*

如果一条外部消息(如文本文件、图片或视频)将要链接到一个字段值,这可以在通过 **info** 前缀加载的表格中完成。(在某些情况下,将首选通过 **qvf** 前缀把信息存储在 **bundle** 文件内。)此表格必须只包含两列,第一列包含构成关键信息的字段值,第二列包含信息元素(如图片等的文件名称)。

例如,同样的限制适用于来自数据库管理系统的图片。在二进制字段上,BLOB info select 语句可生成显式 **bundle**,即二进制数据将直接在 qvf 文件中提取或存储。在 **SELECT** 语句中,二进制数据必须是第二个字段。

如果需要调整图片大小,则可以使用 **image\_size** 子句。

#### 示例:

```
Info LOAD * from flagoe.cd.csv;
Info SQL SELECT * from infotable;
Info SQL SELECT Key, Picture From infotable;
```

### 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。

#### Syntax:

```
Inner ( Join | Keep ) [ (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	<b>LOAD</b> 或 <b>SELECT</b> 语句适用于加载的表格。

#### 示例 1:

Table1	
A	B
1	aa
2	cc
3	ee

Table2	
A	C
1	xx
4	yy

QVTable:  
 SQL SELECT \* From table1;  
 inner join SQL SELECT \* From table2;

QVTable		
A	B	C
1	aa	xx

### 示例 2:

QVTab1:  
 SQL SELECT \* From Table1;  
 QVTab2:  
 inner keep SQL SELECT \* From Table2;

QVTab1	
A	B
1	aa

QVTab2	
A	C
1	xx

**keep** 示例中的两个表格通过 A。

## 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。

### Syntax:

```
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。

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	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 or selectstatement	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 <b>IntervalMatch</b> 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)。

示例 1：

以下两个表格中，第一个表格定义了不同订单生产的开始时间和结束时间。第二个表格列出了众多离散事件。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。

OrderLog

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

EventLog

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

通常首先加载两个表格，然后将字段 *Time* 链接到通过 *Start* 和 *End* 字段定义的时间间隔：

```
SELECT * from OrderLog;
```

```
SELECT * from Eventlog;
IntervalMatch ( Time ) SELECT Start, End from OrderLog;
```

现在可在 Qlik Sense 中创建以下表窗格：

Tablebox

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

## 示例 2: (使用 keyfield)

```
Inner Join IntervalMatch (Date,Key) LOAD FirstDate, LastDate, Key resident Key;
```

## 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。

### Syntax:

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

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

### 参数:

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

### 示例:

```
Join LOAD * from abc.csv;
```

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

```
tab1:
```

```
LOAD * from file1.csv;
tab2:
LOAD * from file2.csv;
... ..
join (tab1) LOAD * from file3.csv;
```

## 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。

### Syntax:

```
(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	说明
tablename	可以将命名的表格与加载的表格进行比较。
loadstatement 或 selectstatement	<b>LOAD</b> 或 <b>SELECT</b> 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 (第 574 页)

### Syntax:

```
Left ( Join | Keep ) [ (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	<b>LOAD</b> 或 <b>SELECT</b> 语句适用于加载的表格。

### 示例:

Table1	
A	B
1	aa
2	cc
3	ee

Table2	
A	C
1	xx
4	yy

QVTable:

```
SELECT * From table1;
```

```
Left Join Sselect * From table2;
```

QVTable		
A	B	C
1	aa	xx
2	cc	
3	ee	

QVTab1:



```
SELECT * From Table1;
QVTab2:
Left Keep SELECT * From Table2;
```

QVTab1	
A	B
1	aa
2	cc
3	ee

QVTab2	
A	C
1	xx

**keep** 示例中的两个表格通过 A。

```
tab1:
LOAD * From file1.csv;
tab2:
LOAD * From file2.csv;
... ..
Left Keep (tab1) LOAD * From file3.csv;
```

### 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。

#### Syntax:

```
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：

```
Mapping LOAD * from x.csv
Mapping SQL SELECT a, b from map1
map1:
mapping LOAD * inline [
x,y
US,USA
U.S.,USA
America,USA ];
```

## 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)。

### Syntax:

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

### 示例:

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

## Outer

可在显式 **Join** 前缀前面使用 **outer** 前缀以指定外部联接。在外部联接中,两表格之间所有的组合都可生成。由此生成的表格将包含原始数据表格的字段值组合,在原始数据表格中,链接字段值呈现在一个或两个表格中。**outer** 关键字是可选的。

### Syntax:

```
Outer 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	<b>LOAD</b> 或 <b>SELECT</b> 语句适用于加载的表格。

### 示例:

Table1	
A	B
1	aa
2	cc
3	ee

Table2	
A	C
1	xx
4	yy

```
SQL SELECT * from table1;
```

```
join SQL SELECT * from table2;
```

OR

```
SQL SELECT * from table1;
```

```
outer join SQL SELECT * from table2;
```

Joined table		
A	B	C
1	aa	xx
2	cc	-
3	ee	-
4	-	yy

## Replace

**replace** 前缀用于删除整个 Qlik Sense 表格，并使用加载或选择的新表格进行替换。



目前，使用 Qlik Engine API 仅支持部分重新加载。

### Syntax:

```
Replace [only] (loadstatement |selectstatement |map...usingstatement)
```

可将 **replace** 前缀添加至脚本中的任意 **LOAD**、**SELECT** 或 **map...using** 语句。**replace LOAD/replace SELECT** 语句对放置整个 Qlik Sense 表格都有效果，其中表格名称由 **replace LOAD/replace SELECT** 语句生成，并且可使用包含 **replace LOAD/replace SELECT** 语句结果的新表格进行更换。部分重载和完全重新加载期间的效果相同。**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	说明
only	可选限定符表示应在正常（非部分）重新加载期间忽视语句。

### 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。

示例	结果
Tab1: Replace only LOAD * from File1.csv;	在常规重新加载期间，此语句会被忽略。  在部分重新加载期间，任意 Qlik Sense 表格以前命名的 Tab1 起初会被删除。此后，将会从 File1.csv 加载新数据，并存储到 Tab1。
Tab1: LOAD a,b,c from File1.csv; Replace LOAD a,b,c from File2.csv;	在常规重新加载期间，文件 File1.csv 首先会读取至 Qlik Sense 表格 Tab1，然后立即删除并由从 File2.csv 加载的新数据替换。来自 File1.csv 的全部数据会丢失。  在部分重新加载期间，整个 Qlik Sense 表格 Tab1 起初会被删除。此后，使用从 File2.csv 加载的新数据进行替换。
Tab1: LOAD a,b,c from File1.csv; Replace only LOAD a,b,c from File2.csv;	常规重新加载期间，将会从 File1.csv 加载数据，并存储到 Qlik Sense 表格 Tab1 中。File2.csv 会被忽略。  在部分重新加载期间，整个 Qlik Sense 表格 Tab1 起初会被删除。此后，使用从 File2.csv 加载的新数据进行替换。来自 File1.csv 的全部数据会丢失。

## Right

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

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



是否按同一名称查找字符串函数？请参阅：[Right \(第 579 页\)](#)

## Syntax:

```
Right (Join | Keep) [(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	<b>LOAD</b> 或 <b>SELECT</b> 语句适用于加载的表格。

## 示例：

Table1	
A	B

1	aa
2	cc
3	ee

Table2	
A	C
1	xx
4	yy

QVTable:

SQL SELECT \* from table1;  
right join SQL SELECT \* from table2;

QVTable		
A	B	C
1	aa	xx
4	-	yy

QVTab1:

SQL SELECT \* from Table1;

QVTab2:

right keep SQL SELECT \* from Table2;

QVTab1	
A	B
1	aa

QVTab2	
A	C
1	xx
4	yy

**keep** 示例中的两个表格通过 A。

tab1:

LOAD \* from file1.csv;

tab2:

LOAD \* from file2.csv;

... ..

right keep (tab1) LOAD \* from file3.csv;

## 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。

**Syntax:**

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

**参数:**

参数	说明
p	<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。</p> <p>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。</p>

**示例:**

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



括号可有可无。

**Semantic**

可 **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。

**Syntax:**

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

Semantic 加载将会创建显示在筛选器窗格中用于导航数据的语义字段。

不能串联通过 **semantic** 语句加载的表格。

**示例:**

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

**Unless**

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

**Syntax:**

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

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

**when** 前缀可以在已有一个或多个语句的语句上使用, 包括其他 **unless** 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	<b>exit for</b> 、 <b>exit do</b> 或 <b>exit sub</b> 子句或 <b>exit script</b> 语句。

**示例:**

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

## When

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

**Syntax:**

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

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

**when** 前缀可以在已有一个或多个语句的语句上使用, 包括其他 **when** 或 **unless** 前缀。

**Syntax:**

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	<b>exit for</b> 、 <b>exit do</b> 或 <b>exit sub</b> 子句或 <b>exit script</b> 语句。

**示例 1:**

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

**示例 2:**

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

**示例 3:**

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

### 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}
```

#### Binary

**binary** 语句用于加载另一个 Qlik Sense 应用程序或者 QlikView 11.2 或更低版本文档的数据，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。

```
Binary file  
file ::= [ 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

**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** 语句用于创建字段和组定义，您可以在其中定义字段或函数之间的关系。可以使用一组字段



定义来自动生成可用作维度的导出字段。例如，您可以创建日历定义，并用它来从日期字段生成相关的维度，如年份、月份、周和日期。

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

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

### Derive

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

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

### Direct Query

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

```
Direct Query [path]
```

### Directory

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

```
Directory [path]
```

### Disconnect

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

```
Disconnect
```

### drop field

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



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

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

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

### drop table

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



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

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

### Execute

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

```
Execute commandline
```

### FlushLog

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

```
FlushLog
```

### Force

**force** 语句用于强制 Qlik Sense 将后面的 **LOAD** 和 **SELECT** 语句的字段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。

```
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 ]
```

### 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计算等号“=”右边的表达式。

```
Let variablename=expression
```

### 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 SQL 或 ODBC 连接发送任意 OLE DB 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

此脚本函数用于创建 QVD 或 CSV 文件。

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

### Tag

此脚本函数用于提供将标签分配给一个或多个字段的方法。如果试图标记应用程序中不存在的字段名称，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 fields fieldlist using mapname  
Tag field fieldname 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

用于提供从一个或多个字段移除标记的方法。如果试图取消标记应用程序中不存在的字段名称，则将忽略该取消标记。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。

```
Untag fields fieldlist using mapname  
Untag field fieldname 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。

### Syntax:

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

### 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 <b>SELECT</b> 和 <b>LOAD</b> 语句。通过新的 <b>alias</b> 语句可以在脚本后面的任何位置定义字段名的新别名。

## Binary

**binary** 语句用于加载另一个 Qlik Sense 应用程序或者 QlikView 11.2 或更低版本文档的数据，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。



在脚本中只允许有一个 **binary** 语句，并且该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。

### Syntax:

```
binary [path] filename
```

### 参数：

参数	说明
filename	文件名称，包括文件扩展名 .qvw 或 .qvf。
path	<p>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。</p> <p><b>示例：'lib://Table Files'</b></p> <p>在传统脚本模式下，同时支持以下路径格式：</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例：c:\data\</b></p> <ul style="list-style-type: none"> <li>相对于包含此脚本行的应用程序。</li> </ul> <p><b>示例：data\</b></p>

### 示例

Binary lib://MyData/customer.qvw;	在此例中， <i>customer.qvw</i> 必须位于与 MyData 数据连接相连的文件夹中。
-----------------------------------	---

Binary customer.qvw;	在此例中, <i>customer.qvw</i> 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。

### Syntax:

```
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	说明
<i>*fieldlist</i>	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。
<i>mapname</i>	先前在映射 <b>LOAD</b> 或映射 <b>SELECT</b> 语句中读取的映射表的名称。
<i>fieldname</i>	要添加注释的字段名。
<i>comment</i>	要添加到字段的注释。

### 示例 1:

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

### 示例 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。

**Syntax:**

```
comment [tables] tablelist using mapname
```

**参数:**

参数	说明
<i>tablelist</i>	(table{,table})
<i>mapname</i>	先前在映射 <b>LOAD</b> 或映射 <b>SELECT</b> 语句中读取的映射表的名称。

**Syntax:**

要设置单个注释，可使用以下语法：

```
comment [table] tablename with comment
```

**参数:**

参数	说明
<i>tablename</i>	要添加注释的表格的名称。
<i>comment</i>	comment 是应添加到表格的注释。

**示例 1:**

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

**示例 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。



此语句仅在标准模式下支持文件夹数据连接。

**Syntax:**

```
ODBC CONNECT TO connect-string
```



```

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

```

参数：

参数	说明
connect-string	<p>connect-string ::= datasource { ; conn-spec-item }</p> <p>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。</p> <p><b>datasource</b> 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。</p> <p>conn-spec-item ::= <b>DBQ</b>=database_specifier   <b>DriverID</b>=driver_specifier   <b>UID</b>=userid   <b>PWD</b>=password</p> <p>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。</p>
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。

示例 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。

示例 2：

```
LIB CONNECT TO 'MyDataConnection';
```

### 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** 语句用于创建字段和组定义，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

#### Syntax:

```
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。  <b>示例：</b>  Calendar:
tag_list	以逗号分隔的标记列表，要应用到根据字段定义导出的字段。应用标记是可选的，但如果不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。  <b>示例：</b>  '\$date'
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。  <b>示例：</b>  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> <b>As</b> field_name <b>tagged</b> tag。使用 \$1 来引用应根据导出的字段生成的数据字段。  <b>示例：</b>  Year(\$1) <b>As</b> Year <b>tagged</b> '\$year'

#### 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

**Syntax:**

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

**参数:**

参数	说明
definition_name	<p>字段定义的名称, 以冒号为结尾。</p> <p><b>示例:</b></p> <p>MyCalendar:</p>
existing_definition	<p>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。</p> <p><b>示例:</b></p> <p>using Calendar</p>
parameter_assignment	<p>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。</p> <p><b>示例:</b></p> <p>first_month_of_year = 4</p>

**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数据字段的数据字段，或者根据字段标签明确或默认导出它们。

### Syntax:

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

### 参数：

参数	说明
definition	在导出字段时使用的字段定义的名称。  <b>示例：Calendar</b>
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。  <b>示例：OrderDate, ShippingDate</b>
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。  <b>示例：'\$date'</b>

### 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。

#### Syntax:

```
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名称： <i>fieldname {, fieldname}</i> 。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	<p>此处没有定义数据库 <b>WHERE</b> 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 <b>LIKE</b> 运算符、<b>IS NULL</b> 和 <b>IS NOT NULL</b>，不包括 <b>IN</b>、<b>BETWEEN</b>。</p> <p><b>NOT</b> 是一元运算符，而非某些关键字的修饰符。</p> <p>示例：</p> <pre> WHERE x &gt; 100 AND "Region Code" IN ('south', 'west') WHERE Code IS NOT NULL and Code LIKE '%prospect' WHERE NOT X in (1,2,3) </pre> <p>不能将最后一个示例写成如下所示：</p> <pre> WHERE X NOT in (1,2,3) </pre>

### 示例：

在本例中，数据库表称为 TableName，包含使用的字段 Dim1、Dim2、Num1、Num2 和 Num3。将 Dim1 和 Dim2 加载到 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;
```

可以更改成以下 **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,  
  
        Moles/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。

### Syntax:

```
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	说明
<b>path</b>	<p>可解释为 qvf 文件的路径的文本。</p> <p>该路径是文件的路径，即：</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例：c:\data\</b></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例：data\</b></p> <ul style="list-style-type: none"> <li>URL 地址 (HTTP 或 FTP)，指向一个互联网或内联网的位置。</li> </ul> <p><b>示例：http://www.qlik.com</b></p>

### 示例：

```
Directory lib://Data/;  
Directory c:\userfiles\data;
```

## Disconnect

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

### Syntax:

```
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 field

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



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

### Syntax:

```
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。

### Syntax:

```
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:  SQL SELECT* from Trans; LOAD Customer, Sum( sales ) resident Tab1 group by Month; drop table Tab1;	结果，内存中仅保留聚合值。Trans 数据会被忽略。

## 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。



在标准模式下不支持此语句。

### Syntax:

```
execute commandline
```

### 参数：

参数	说明
<i>commandline</i>	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：

- 您必须在传统模式下运行(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
```

### 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。

**Syntax:**

```
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。

**Syntax:**

```
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：**

```
Force Capitalization;  
Force Case Upper;  
Force Case Lower;  
Force 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。

**Syntax:**

```
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 ]
```

**参数：**

参数	说明
distinct	<b>distinct</b> 是一个只应加载第一条重复记录使用的谓词。

参数	说明
fieldlist	<p><i>fieldlist</i> ::= ( *   <i>field</i> { , *   <i>field</i> } )</p> <p>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。</p> <p><i>field</i> ::= ( <i>fieldref</i>   <i>expression</i> ) [ <b>as</b> <i>aliasname</i> ]</p> <p>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。</p> <p><i>fieldref</i> ::= ( <i>fieldname</i>   @<i>fieldnumber</i>   @<i>startpos</i>:<i>endpos</i> [ <i>I</i>   <i>U</i>   <i>R</i>   <i>B</i>   <i>T</i> ] )</p> <p><i>fieldname</i> 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：</p> <p>@<i>fieldnumber</i> 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。</p> <p>@<i>startpos</i>:<i>endpos</i> 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 <i>n</i> 用作结束位置。</p> <ul style="list-style-type: none"> <li>如果 @<i>startpos</i>:<i>endpos</i> 后紧随字符 <b>I</b> 或 <b>U</b>，字节读取将分别解释为二进制带符号 (<b>I</b>) 或不带符号 (<b>U</b>) 整数 (Intel 字节序)。数字读取位置必须是 1, 2 或 4。</li> <li>如果 @<i>startpos</i>:<i>endpos</i> 后紧随字符 <b>R</b>，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。</li> <li>如果 @<i>startpos</i>:<i>endpos</i> 后紧随字符 <b>B</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。</li> </ul> <p><i>expression</i> 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。</p> <p><b>as</b> 用于为字段设定新名称。</p>

参数	说明
from	<p>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 <b>from</b></p> <p><i>file ::= [ path ] filename</i></p> <p><b>示例: 'lib://Table Files/'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例: c:\data\</b></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例: data\</b></p> <ul style="list-style-type: none"> <li>URL 地址 (HTTP 或 FTP), 指向一个互联网或内联网的位置。</li> </ul> <p><b>示例: http://www.qlik.com</b></p> <p>如果路径被省略, Qlik Sense 则在由 <b>Directory</b> 语句指定的目录中搜索文件。如果没有 <b>Directory</b> 语句, 那么 Qlik Sense 将在工作目录 <i>C:\Users\{user}\Documents\Qlik\Sense\Apps</i> 中搜索。</p> <div style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"> <p> 在 Qlik Sense 服务器安装中, 工作目录是在 Qlik Sense Repository Service 中指定, 默认情况下是 <i>C:\ProgramData\Qlik\Sense\Apps</i>。有关详细信息, 请参阅 Qlik Management Console 帮助。</p> </div> <p><i>filename</i> 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。</p> <p><i>format-spec ::= ( fspec-item {, fspec-item} )</i></p> <p>格式规格包含数个格式规格项目的列表 (在括号中)。</p>
from_field	<p>如果需从之前加载的字段加载数据, 则使用 <b>from_field</b> 语句。</p> <p><i>fieldassource::=(tablename, fieldname)</i></p> <p>该字段是之前加载过的 <i>tablename</i> 和 <i>fieldname</i> 的名称。</p> <p><i>format-spec ::= ( fspec-item {, fspec-item} )</i></p> <p>格式规格包含数个格式规格项目的列表 (在括号中)。</p>

参数	说明
inline	<p><b>inline</b> 当数据需要输入脚本而不是从文件中加载时使用该符号。</p> <pre>data ::= [ text ]</pre> <p>通过 <b>inline</b> 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 <b>inline</b> 子句文本中重复该操作，即，键入脚本时按压输入键。</p> <pre>format-spec ::= ( fspec-item {, fspec-item} )</pre> <p>格式规格包含数个格式规格项目的列表(在括号中)。</p>
resident	<p>如果需从之前加载的表格加载数据，则使用 <b>resident</b> 语句。</p> <p><i>table label</i> 是加在创建于原始表格的 <b>LOAD</b> 或 <b>SELECT</b> 语句之前的标签。该标签需要在末尾加上冒号。</p>
autogenerate	<p><b>autogenerate</b> 是数据需要 Qlik Sense 自动生成时使用。</p> <pre>size ::= number</pre> <p><i>Number</i> 是用来指示生成记录数字的整数。</p> <p>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 <b>Peek</b> 函数加载的表格中的单个字段值。</p>
where	<p><b>where</b> 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 <i>criterion</i> 为 True，则将其包括在选择项内。</p> <p><i>criterion</i> 是一个逻辑表达式。</p>
while	<p><b>while</b> 是用于显示记录是否应该反复读取的子句。只要 <i>criterion</i> 为 True，则同一记录被读取。为了能发挥作用，<b>while</b> 子句通常应包含 <b>IterNo()</b> 函数。</p> <p><i>criterion</i> 是一个逻辑表达式。</p>
group by	<p><b>group by</b> 是用于定义应聚合(组合)的字段子句。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。</p> <pre>groupbyfieldlist ::= (fieldname {,fieldname} )</pre>
order by	<p><b>order by</b> 是用于被加于 <b>load</b> 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。</p> <p>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名称或其数字指定(第一个字段为 1)。</p> <pre>orderbyfieldlist ::= fieldname [ sortorder ] {, fieldname [ sortorder ] }</pre> <p><i>sortorder</i> 要么是 <i>asc</i>(对于升序)，要么是 <i>desc</i>(对于降序)。如果未指定任何 <i>sortorder</i>，将假设 <i>asc</i>。</p> <p><i>fieldname</i>、<i>path</i>、<i>filename</i> 和 <i>aliasname</i> 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 <i>fieldname</i>。但是，通过子句 (<i>aliasname</i>) 创建的字段不在此范围之内，且不能用于相同的 <b>load</b> 语句内。</p>



如果 **from**、**inline**、**resident**、**from\_field** 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 (MyData) 加载分隔符分隔的数据文件：

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

通过库连接 (MyData) 加载所有分隔符分隔的数据文件：

```
LOAD * from 'lib://MyData/*.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);
```

选择特定字段，重命名和计算字段

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 and 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://MyData/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 des;
```

从之前加载的字段中加载数据

从之前加载的表格 Characters 中加载字段 Types 作为 A:

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

从随后的表格中加载数据(前置加载)

使用随后的 **SELECT** 语句从已加载的 Table1 中加载 A、B 以及计算字段 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

## 格式规格项目

每种格式规格项目定义表格文件的特定属性：

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

## 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。

## Syntax:

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

## 参数：

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

**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)
```

**另请参阅:**

□ [Load\(第 77 页\)](#)

**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。

**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 **.xlsx** 格式。

**html** 如果表格是 html 页面或文件的一部分，则应使用 html。

**xml** xml( 可扩展标记语言) 是一种通用标记语言，用于以文本格式表示数据结构。

**qvd** **qvd** 格式是 QVD 文件的专用格式，可从 Qlik Sense 应用程序导出。

**qvx** **qvx** 是一种用于 Qlik Sense 高性能输出的文件/数据流格式。

**Delimiter**

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。

**Syntax:**

```
delimiter is char
```

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：

'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);
```

另请参阅：

▢ [Load\( 第 77 页\)](#)

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。

Syntax:

```
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);
```

另请参阅：

▢ [Load\( 第 77 页\)](#)

### 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。

#### Syntax:

```
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。在这种情况下，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。

#### 示例 1:

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

#### 示例 2:

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

---

#### 另请参阅:

□ [Load\(第 77 页\)](#)

### 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。

#### Syntax:

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

标题长度可以用字节数 (**header is n**) 标题，也可以用行数 (**header is line** 或 **header is n lines**) 表示。**n** 必须为正整数，代表标题长度。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。

---

另请参阅:

▢ [Load\( 第 77 页\)](#)

### 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。

#### Syntax:

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

#### 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。

---

另请参阅:

▢ [Load\( 第 77 页\)](#)

### 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

#### Syntax:

```
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	说明
no quotes	只有在文本文件中无法使用引号时才使用。
msq	<p>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。</p> <p>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。</p>

## 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。

#### Syntax:

```
xmlsax
xmlsimple
pattern is path
```

**xmlsax** 和 **xmlsimple** 相互排斥，因此使用 xml 时只能指定其中一个。使用 *pattern* 时，将从指定标签的开始位置读取，一直到标签的结尾。如果 *path* 包含空格，则该路径必须使用引号括起来。



为了使用 **xmlsax**，Microsoft 的 xml 解析器，必须在电脑上安装 MSXML 3.0 或更高版本。例如，随 Windows XP 和 MS Internet Explorer 6 附带的 MSXML。还可以从 Microsoft 主页下载。

#### 另请参阅：

□ [Load\(第 77 页\)](#)

## KML

当加载用于图形可视化的 KML 文件时，使用此脚本说明符。

#### Syntax:

```
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)或数据点(例如，城市或地点，以 [经度, 纬度] 格式的点的点来表示)。

## 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计算等号“=”右边的表达式。

### Syntax:

```
Let variablename=expression
```

**let** 一词可以省略，但省略后该语句将变为控制语句。这种没有关键字 **let** 的语句必须包含在单个脚本行中，并且可能以分号或换行符终止。

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”
Let T=now( );	\$(T) 将给定当前时间的值。

## 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。

### Syntax:

```
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	说明
<i>fieldlist</i>	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。
<i>mapname</i>	先前在 <b>mapping load</b> 或 <b>mapping select</b> 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。

### Syntax:

```
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	说明
*fieldlist	用逗号分隔的字段列表，应针对该列表启用 <b>NullAsNull</b> 。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。

### Syntax:

```
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	说明
*fieldlist	用逗号分隔的字段列表，应针对该列表启用 <b>NullAsValue</b> 。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。

**Syntax:**

```
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 **from** 和 **LOAD** 语句中 **SELECT** 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	说明
*fieldlist	用逗号分隔的字段列表，为此限定应开启。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 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**。三个字段将显示结果：**A**、**x.B**、**y.B**。

**示例 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。

**Syntax:**

```
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. 通过将有问题的一部分放置在 **/\*** 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	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 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

或者：**rename field** 或 **rename fields** 可用。

#### Syntax:

```
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。

#### 示例 1:

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

#### 示例 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。

或者：**rename table** 或 **rename tables** 可用。

#### Syntax:

```
Rename Table (using mapname | oldname to newname{ , oldname to newname })
Rename Tables (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。脚本将正常运行，但是第二个表格无法重命名。

**示例 1：**

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

**示例 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。

**Syntax:**

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

可以使用多个 **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	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：**

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 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。

#### Syntax:

```
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。

#### Syntax:

```
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** 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	说明
distinct	<b>distinct</b> 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	<b>distinctrow</b> 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	<p><b>fieldlist</b> ::= ( *   field ) { , field }</p> <p>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。</p> <p><b>fieldlist</b> ::= field { , field }</p> <p>一个或多个字段的列表，用逗号分开。</p> <p><b>field</b> ::= ( fieldref   expression ) [ as aliasname ]</p> <p>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。一些通常接受的运算符或函数为：+、-、*、/、&amp; (字符串串联)、sum(fieldname)、count(fieldname)、avg(fieldname)(average)、month(fieldname) 等。请参阅 ODBC 驱动程序的文档，了解更多信息。</p> <p><b>fieldref</b> ::= [ tablename. ] fieldname</p> <p><b>tablename</b> 和 <b>fieldname</b> 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。</p> <p><b>as</b> 子句用于为字段分配一个新名。</p>
from	<p><b>tablelist</b> ::= table { , table }</p> <p>要从其中选择字段表格列表。</p> <p><b>table</b> ::= tablename [ [ as ] aliasname ]</p> <p><b>tablename</b> 可以也可以不放在引号内。</p>
where	<p><b>where</b> 是一个子句，用于陈述一个记录是否应该包括在选择项内。</p> <p><b>criterion</b> 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、=、&lt;&gt; 或 # (不等于)、&gt;、&gt;=、&lt;、&lt;=、and、or、not、exists、some、all、in 和新的 <b>SELECT</b> 语句。有关详细信息，请参阅文档 ODBC 驱动程序或 OLE DB 提供者。</p>
group by	<b>group by</b> 是一个子句，用于将几个记录聚合 (组成) 为一个整体。对于某些字段来说，在一个组中，所有记录要么拥有一个相同的值，要么字段只能用于一个表达式内，如作为合计或平均值。基于一个或几个字段的表达式在字段符号的表达式中定义。
having	<b>having</b> 是一个子句，用于以一种与 <b>where</b> 子句在限定记录时相同的使用方式限定组。

参数	说明
order by	<b>order by</b> 是一个用于表述 <b>SELECT</b> 语句的结果表排序顺序的子句。
join	<b>join</b> 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 <b>Connect</b> 提供者偏好的引号。

**示例 1:**

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

**示例 2:**

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

**示例 3:**

```
SELECT `Order ID`, `Product ID`,  
`Unit Price` * Quantity * (1-Discount) as NetSales  
FROM `Order Details`;
```

**示例 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。

**Syntax:**

```
Set variablename=string
```

**示例 1:**

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

**示例 2:**

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

**示例 3:**

```
Set BudgetYear=2012;
```

## Sleep

**sleep** 语句用于在指定的时间暂停脚本执行。

### Syntax:

```
Sleep n
```

### 参数:

参数	说明
n	以毫秒为单位表示, 其中 <i>n</i> 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。

### 示例 1:

```
Sleep 10000;
```

### 示例 2:

```
Sleep t*1000;
```

## SQL

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

### Syntax:

```
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
<i>sql_command</i>	有效的 SQL 命令。

### 示例 1:

SQL Leave;

### 示例 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。

### Syntax:

**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。

#### Syntax:

**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。

#### Syntax:

**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。

#### Syntax:

```
Star is[ string ]
```

#### 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 <b>star is;</b> ，即无星号可用，除非明确指定。此定义会一直有效，直到新的 <b>star</b> 语句出现。

#### 示例：

```
Star is *;  
Star is %;  
Star is;
```

## Store

此脚本函数用于创建 QVD 或 CSV 文件。

### Syntax:

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

该语句将创建一个明确命名的 QVD 或 CSV 文件。该语句仅会从一个数据表格中导出字段。如果要从多个表格中导出字段，必须明确命名之前在脚本中生成的 join 以创建应导出的数据表。

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

### 参数:

参数	说明
<i>fieldlist</i> ::= ( *   <i>field</i> ) { , <i>field</i> }	<p>要选择的字段列表。使用 * 作为字段列表，则其表示全部字段。</p> <p><i>field</i>::= <i>fieldname</i> [ <b>as</b> <i>aliasname</i> ]</p> <p><i>fieldname</i>是指与<i>table</i>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。)</p> <p><i>aliasname</i> 是指生成的 QVD 或 CSV 文件中所用字段的替代名称。</p>
<i>table</i>	脚本标签表示要用作数据源的已加载表格。

参数	说明
<i>filename</i>	<p>目标文件的名称，包括现有文件夹数据连接的有效路径。</p> <p><b>示例：</b> <i>'lib://Table Files/target.qvd'</i></p> <p>在传统脚本模式下，同时支持以下路径格式：</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例：</b> <i>c:\data\</i></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例：</b> <i>data\</i></p> <p>如果路径被省略，Qlik Sense 则在由 <b>Directory</b> 语句指定的目录中存储文件。如果没有 <b>Directory</b> 语句，那么 Qlik Sense 将在工作目录 <i>C:\Users\{user}\Documents\Qlik\Sense\Apps</i> 中存储文件。</p>
<i>format-spec ::= ( txt   qvd )</i>	<p>格式规范包含文本 <b>txt</b>(对于文本文件)或文本 <b>qvd</b>(对于 qvd 文件)。如果省略格式规范，则假定为 <b>qvd</b>。</p>

**示例：**

```
Store mytable into xyz.qvd (qvd);
Store * from mytable into 'lib://FolderConnection/myfile.qvd';
Store Name, RegNo from mytable into xyz.qvd;
Store Name as a, RegNo as b from mytable into 'lib://FolderConnection/myfile.qvd';
store mytable into myfile.txt (txt);
store * from mytable into 'lib://FolderConnection/myfile.qvd';
```

**Tag**

此脚本函数用于提供将标签分配给一个或多个字段的方法。如果试图标记应用程序中不存在的字段名称，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。

**Syntax:**

```
Tag fields fieldlist using mapname
Tag field fieldname with tagname
```

**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标记。



参数	说明
mapname	先前在 <b>mapping Load</b> 或 <b>mapping Select</b> 语句中加载的映射表的名称。
fieldname	应标记的字段名。
tagname	要应用到字段的标记名称。

**示例 1:**

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

**示例 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。

**Syntax:**

```
Trace string
```

**示例 1:**

```
Trace Main table loaded;
```

**示例 2:**

```
Let MyMessage = NoOfRows('MainTable') & ' rows in Main Table';
Trace $(MyMessage);
```

## 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。

**Syntax:**

```
Unmap *fieldlist
```

**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。

**Syntax:**

```
Unqualify *fieldlist
```

**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 <b>Qualify</b> 语句。

**示例 1:**

```
Unqualify *;
```

**示例 2:**

```
Unqualify TransID;
```

## Untag

用于提供从一个或多个字段移除标记的方法。如果试图取消标记应用程序中不存在的字段名称，则将忽略该取消标记。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。

**Syntax:**

```
Untag fields fieldlist using mapname
Untag field fieldname with tagname
```

参数：

参数	说明
fieldlist	用逗号分隔的字段列表，其标记应移除。
mapname	先前在映射 <b>LOAD</b> 或映射 <b>SELECT</b> 语句中加载的映射表的名称。
fieldname	应取消标记的字段名。
tagname	应从字段移除的标记名称。

示例 1：

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

示例 2：

```
Untag field Alpha with MyTag2;];
```

## 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 Repository Service 中指定，默认情况下是 `C:\ProgramData\Qlik\Sense\Apps`。有关详细信息，请参阅 Qlik Management Console 帮助。

## 2.4 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。可以使用变量概述定义变量或使用数据加载编辑器在脚本中定义变量，其中变量从 **Let**、**Set** 或数据加载脚本的其他控制语句获取其值。



编辑表格时，您也可以从变量概述使用 Qlik Sense 变量。

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美元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。

定义变量的语法为：

```
set variablename = string
```

或

```
let variable = expression
```

。Set 命令用于将文本赋值给变量等号的右边，而 Let 命令用于对表达式进行求值。

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 HidePrefix = $ ; //, 此变量将字符 '$' 作为值。
```

```
let vToday = Num(Today()); // 用于返回当天的日期序列号。
```

## 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，必须也从变量概述删除变量。

## 将变量值加载为字段值

如果要使用 LOAD 语句将变量值加载为字段值，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，在加载为字段值时，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
```

## 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：

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	第 125 页
数字解释变量	第 116 页
系统变量	第 109 页
值处理变量	第 114 页

## 系统变量

一部分系统变量由系统所定义，用于提供有关系统和 Qlik Sense 应用程序的信息。

### 系统变量概述

一部分函数在概述后面进行了详细描述。对于这些函数，可以单击语法中的函数名称即时访问有关该特定函数的更多信息。

#### Floppy

用于返回找到的第一个软盘驱动器的驱动器号，通常是 **a:**。这是系统定义的变量。

#### Floppy



在标准模式下不支持此变量。

#### CD

用于返回找到的第一个 CD-ROM 驱动器的驱动器号。如果未找到任何 CD-ROM，随后会返回 **c:**。这是系统定义的变量。

#### CD



在标准模式下不支持此变量。

#### Include

**Include/Must\_Include** 变量用于指定包含应包括在脚本中并作为脚本代码计算值的文本的文件。您可以将部分脚本代码存储在单独的文本文件中，并可以在多个应用程序中重复使用它。这是用户定义的变量。

```
$(Include =filename)
$(Must_Include=filename)
```

#### HidePrefix

所有以此文本字符串开始的字段名会按与系统字段相同的方式隐藏。这是用户定义的变量。

#### HidePrefix

#### HideSuffix

所有以此文本字符串结束的字段名会按与系统字段相同的方式隐藏。这是用户定义的变量。

#### HideSuffix

#### QvPath

用于返回浏览字符串到可执行的 Qlik Sense 文件。这是系统定义的变量。

#### QvPath



在标准模式下不支持此变量。

#### QvRoot

用于返回可执行的 Qlik Sense 的根目录。这是系统定义的变量。

### QvRoot



在标准模式下不支持此变量。

### QvWorkPath

用于返回浏览字符串到当前 Qlik Sense 应用程序。这是系统定义的变量。

### QvWorkPath



在标准模式下不支持此变量。

### QvWorkRoot

用于返回当前 Qlik Sense 应用程序的根目录。这是系统定义的变量。

### QvWorkRoot



在标准模式下不支持此变量。

### StripComments

如果此变量设置为 0, 则禁止剥离脚本中的 /\*..\*/ 和 // 注释。如果未定义此变量, 则会始终执行注释剥离。

### StripComments

### Verbatim

全部字段值通常会自动剥离前导和尾部空格 (ASCII 32), 然后再加载到 Qlik Sense 数据库。将此变量设置为 1 会暂停剥离空格。Tab (ASCII 9) 和硬空格 (ANSI 160) 字符永远都不会剥离。

### Verbatim

### OpenUrlTimeout

此变量用于定义 Qlik Sense 应在从 URL 源(如 HTML 页面)获取数据时考虑的超时(秒)。如果省略, 则超时约为 20 分钟。

### OpenUrlTimeout

### WinPath

用于返回浏览字符串到 Windows。这是系统定义的变量。

### WinPath



在标准模式下不支持此变量。

### WinRoot

返回 Windows 的根目录。这是系统定义的变量。

### WinRoot



在标准模式下不支持此变量。

### CollationLocale

指定要用于排序顺序和搜索匹配的区域设置。该值是区域设置的区域性名称，如“zh-CN”。这是系统定义的变量。

### CollationLocale

### 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。

#### Syntax:

### 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。

#### Syntax:

### 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。您可以将部分脚本代码存储在单独的文本文件中，并可以在多个应用程序中重复使用它。这是用户定义的变量。



此变量仅在标准模式下支持文件夹数据连接。



### Syntax:

```
$(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。



构造函数 **set Include =filename** 不适用。

### 示例：

```
$(Include=abc.txt);
$(Must_Include=lib://MyDataFiles\abc.txt);
```

## 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。

### Syntax:

```
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。

### Syntax:

```
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。

### Syntax:

#### Verbatim

### 示例：

```
set Verbatim = 1;
```

## 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。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。这是用户定义的变量。

#### Syntax:

```
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。

#### Syntax:

```
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。

#### Syntax:

```
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。

#### Syntax:

```
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'。

## 数字解释变量

数字解释变量是系统定义的变量，即这些变量是在创建新的应用程序时根据当前操作系统区域设置自动生成的。在 Qlik Sense Desktop 中，这取决于计算机操作系统的设置，在 Qlik Sense 中，它取决于已安装 Qlik Sense 的服务器的操作系统。

包括的变量位于 Qlik Sense 的新应用程序脚本顶部，并且可以在执行脚本时替换操作系统默认设置为某种数字格式设置。您可以随意删除、编辑或复制这些变量。



如果您想要为某些区域创建应用程序，最简单的方法可能是在操作系统中包含所需区域设置的计算机上使用 Qlik Sense Desktop 创建应用程序。然后，应用程序包含该区域的适当地区设置，您可以将其移动到所选择的 Qlik Sense 服务器以便进一步开发。

## 数字解释变量概述

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

### 货币格式

#### MoneyDecimalSep

定义的小数位分隔符会替换操作系统(地区设置)的货币小数位符号。

**MoneyDecimalSep**

#### MoneyFormat

定义的符号会替换操作系统(地区设置)的货币符号。

**MoneyFormat**

#### MoneyThousandSep

定义的千位分隔符会替换操作系统(地区设置)的货币数字分组符号。

**MoneyThousandSep**

### 数字格式

#### **DecimalSep**

定义的小数位分隔符会替换操作系统(地区设置)的小数位符号。

```
DecimalSep
```

#### **ThousandSep**

定义的千位分隔符会替换操作系统(地区设置)的数字分组符号。

```
ThousandSep
```

### 时间格式

#### **DateFormat**

定义的格式会替换操作系统(地区设置)的日期格式。

```
DateFormat
```

#### **TimeFormat**

定义的格式会替换操作系统(地区设置)的时间格式。

```
TimeFormat
```

#### **TimestampFormat**

定义的格式会替换操作系统(地区设置)的日期和时间格式。

```
TimestampFormat
```

#### **MonthNames**

定义的格式会替换操作系统(地区设置)的普通月名称惯例。

```
MonthNames
```

#### **LongMonthNames**

定义的格式会替换操作系统(地区设置)的长月名称惯例。

```
LongMonthNames
```

#### **DayNames**

定义的格式会替换操作系统(地区设置)的普通日名称惯例。

```
DayNames
```

#### **LongDayNames**

定义的格式会替换操作系统(地区设置)的长普通日名称惯例。

```
LongDayNames
```

#### **FirstWeekDay**

整数用于定义将哪一天用作一周的第一天。

### **FirstWeekDay**

#### **BrokenWeeks**

此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。

有效设置为 1(一月)到 12(十二月)。默认设置为 1。

#### **Syntax:**

### **FirstMonthOfYear**

#### **示例:**

```
Set FirstMonthOfYear=4; //Sets the year to start in April
```

#### **BrokenWeeks**

此设置用于定义周是否已中断。

#### **Syntax:**

### **BrokenWeeks**

默认情况下，Qlik Sense 函数使用连续的周。这意味着：

- 在某些年份中，第 1 周在 12 月开始，而在其他年份中，第 52 或 53 周延续到 1 月。
- 在 1 月中，第 1 周始终至少有 4 天。

替代方法是使用不连续的周。

- 第 52 或 53 周不延续到 1 月。
- 第 1 周在 1 月 1 日开始，因此在大部分情况下不是完整的一周。

可以使用以下值：

- 0(表示使用连续周)
- 1(表示使用不连续周)

#### **示例:**

```
Set BrokenWeeks=0; //(use unbroken weeks)
Set BrokenWeeks=1; //(use broken weeks)
```

### DateFormat

定义的格式会替换操作系统(地区设置)的日期格式。

#### Syntax:

```
DateFormat
```

#### 示例:

```
Set DateFormat='M/D/YY'; //(US format)
Set DateFormat='DD/MM/YY'; //(UK date format)
Set DateFormat='YYYY-MM-DD'; //(ISO date format)
```

### DayNames

定义的格式会替换操作系统(地区设置)的普通日名称惯例。

#### Syntax:

```
DayNames
```

#### 示例:

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

### DecimalSep

定义的小数位分隔符会替换操作系统(地区设置)的小数位符号。

#### Syntax:

```
DecimalSep
```

#### 示例:

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

### FirstWeekDay

整数用于定义将哪一天用作一周的第一天。

#### Syntax:

```
FirstWeekDay
```

默认情况下, Qlik Sense 函数使用周一作为一周的第一天。可以使用以下值:

- 0(表示周一)
- 1(表示周二)
- 2(表示周三)
- 3(表示周四)

- 4(表示周五)
- 5(表示周六)
- 6(表示周日)

**示例：**

```
Set FirstWeekDay=6; //(set Sunday as the first day of the week)
```

### LongDayNames

定义的格式会替换操作系统(地区设置)的长普通日名称惯例。

**Syntax:**

```
LongDayNames
```

**示例：**

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

### LongMonthNames

定义的格式会替换操作系统(地区设置)的长月名称惯例。

**Syntax:**

```
LongMonthNames
```

**示例：**

```
Set LongMonthNames='January;February;March;April;May;June - -
```

### MoneyDecimalSep

定义的小数位分隔符会替换操作系统(地区设置)的货币小数位符号。

**Syntax:**

```
MoneyDecimalSep
```

**示例：**

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

### MoneyFormat

定义的符号会替换操作系统(地区设置)的货币符号。

**Syntax:**

```
MoneyFormat
```

**示例：**



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

### MoneyThousandSep

定义的千位分隔符会替换操作系统(地区设置)的货币数字分组符号。

#### Syntax:

```
MoneyThousandSep
```

#### 示例:

```
Set MoneyThousandSep=', ';
```

### MonthNames

定义的格式会替换操作系统(地区设置)的普通月名称惯例。

#### Syntax:

```
MonthNames
```

#### 示例:

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

### 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。

#### Syntax:

```
ReferenceDay
```

默认情况下, Qlik Sense 函数使用 4 作为参考日。这意味着第 1 周必须包含 1 月 4 日, 换句话说, 第 1 周始终至少具有 1 月份的前 4 天。

以下值可用于设置不同参考日:

- 1(表示 1 月 1 日)
- 2(表示 2 月 1 日)
- 3(表示 3 月 1 日)
- 4(表示 4 月 1 日)
- 5(表示 5 月 1 日)
- 6(表示 6 月 1 日)
- 7(表示 7 月 1 日)

#### 示例:

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

### ThousandSep

定义的千位分隔符会替换操作系统(地区设置)的数字分组符号。

### Syntax:

**ThousandSep**

### 示例:

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

### TimeFormat

定义的格式会替换操作系统(地区设置)的时间格式。

### Syntax:

**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。

### Syntax:

**TimestampFormat**

### 示例:

```
Set TimestampFormat='M/D/YY hh:mm:ss[.fff]';
```

## 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='YYYY-MM-DD'
```

### 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]';
```

### 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

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。

### 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。

### Syntax:

#### ErrorMode

### 参数:

参数	说明
<b>ErrorMode=1</b>	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)。
<b>ErrorMode=0</b>	Qlik Sense 只需忽略故障，并继续在下一个脚本语句上执行脚本。
<b>ErrorMode=2</b>	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。

### Syntax:

**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对偶值。以下错误代码存在：

错误代码	说明
0	无错误
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	表格未找到
11	字段未找到
12	文件格式错误
13	BIFF 错误
14	BIFF 加密错误
15	BIFF 不受支持版本错误
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。

### Syntax:

**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。

### Syntax:

**ScriptErrorList**

## 2.5 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：

expression ::= (constant	constant	
	fieldref	
	operator1 expression	
	expression operator2 expression	
	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。

**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 }**

参数的数量和类型不是任意的。它取决于所使用的函数。

表达式和函数还可自由嵌套, 并且只要表达式返回可解释的值, Qlik Sense 就不会显示任何错误信息。

## 3 可视化表达式

表达式是函数、字段和数学运算符 (+ \* / =) 的组合。表达式用于处理应用程序中的数据，以便生成可以在可视化中看到的结果。在度量中，不限制使用表达式。您可以创建更有活力更强大的可视化，只需使用标题、副标题、脚注和维度的表达式。

这表示(例如)可视化标题不是静态文本，而是可以使用表达式获取的内容，其结果将根据做出的选择改变。




有关脚本函数和图表函数的详细参考，请参阅 [脚本语法和图表函数](#)。

### 3.1 定义聚合范围

通常，结合两个因子可以确定用于定义表达式中聚合值的记录。当在可视化中使用时，这些因子为：

- 维度值(图表表达式中的聚合)
- 选择项

总之，这些因子可定义聚合的范围。您可能会遇到希望计算忽略选择项、维度或同时忽略这二者的情况。在图表函数中，为此可以使用 TOTAL 限定符、集合分析或两者的组合。

方法	说明
TOTAL 限定符	<p>在聚合函数内使用合计限定符忽略维度值。</p> <p>将对所有可能的字段值执行聚合。</p> <p><b>TOTAL</b> 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。</p>
集合分析	<p>在聚合内使用集合分析将忽略选择项。将对在维度之间拆分的所有值执行聚合。</p>
TOTAL 限定符和集合分析	<p>在聚合内使用集合分析忽略选择项和维度。</p> <div>  此方法相当于使用 ALL 限定符。 </div>

#### 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。

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。

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。

Year	Quarter	Sum (Amount)	Sum({1} TOTAL Amount)	Sum(Amount)/Sum({1} TOTAL Amount)
		3000	10800	27,8%
2012	Q2	1700	10800	15,7%
2013	Q2	1300	10800	12%

示例中所使用的数据：

```
AggregationScope:
LOAD * inline [
Year Quarter Amount
2012 Q1 1100
2012 Q2 1700
2012 Q3 1400
2012 Q4 1800
2013 Q1 1000
2013 Q2 1300
2013 Q3 1100
2013 Q4 1400] (delimiter is ' ');
```

## 3.2 集合的语法

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_identifier ::= 1 | $ | $N | $_N | bookmark_id | bookmark_name
set_operator ::= + | - | * | /
set_modifier ::= < field_selection {, field_selection } >
field_selection ::= field_name [ = | += | -= | *= | /= ] element_set_expression
element_set_expression ::= element_set { set_operator element_set }
element_set ::= [ field_name ] | { element_list } | element_function
element_list ::= element { , element }
element_function ::= ( P | E ) ( [ set_expression ] [ field_name ] )
element ::= field_value | " search_mask "
```

## 3.3 集合修饰符

通过添加或更换选择项可修改集合。此类修改可写入集合表达式。

集合修饰符包括一个或几个字段名称, 每个字段后均有属于此字段范畴的选择项, 所有选择由 < 和 > 括起来。例如: <Year={2007,+2008},Region={US}>。字段名称和字段值可以照常引用, 例如: <[Sales Region]={'West coast', 'South America'}>。

集合修饰符可用于集合标识符中, 也可单独使用。它不能用于集合表达式。当用于集合标识符中时, 修饰符必须紧接在集合标识符之后, 例如 {<Year = {2007, 2008}>}。单独使用时, 就等于修改当前选择范围。

定义选择项的方式有多种, 如下所述。

### 基于另一个字段

一种简单的做法是, 选择项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。

## 基于元素集(修饰符中的字段值列表)

最常用的做法是，选择项基于放入波形括号中的字段值列表，各值以逗号分隔，例如 `<Year = {2007, 2008}>`。波浪括号中定义的是元素集，所含元素可以是字段值或字段值搜索。搜索始终使用双引号定义，例如，`<Ingredient = {"*Garlic"}>` 将选择包含字符串“garlic”的所有原料。搜索不区分大小写，并且会在不包含的值上进行。

空白元素集，明显的例子如 `<Product = {}>`，不明显的例子如 `<Product = {"Perpetuum Mobile"}>` (未搜索到匹配记录)，均表示没有产品，即生成的记录集与任何产品都不相关。请注意，此集合不能通过通常的选择项获得，除非在其他字段中选择，例如：**TransactionID**。

## 强制排除

最后，对于字段输入模式，仍有可能需要强制排除。如果要强制排除特定字段值，需要在字段名称前加“~”。

示例和结果：

示例	结果
<code>sum( {1&lt;Region= {USA}&gt;} Sales )</code>	返回 USA 区域的销售额，忽略当前选择项
<code>sum( {\$&lt;Region = &gt;} Sales )</code>	返回当前选择项的销售额，但移除“Region”中的选择项
<code>sum( {&lt;Region = &gt;} Sales )</code>	返回与上一例相同的销售额。当省略要修改的集合时，则假定 \$。
	<div>  <p>上两例的语法被解释为在“Region”中“没有选择项”，也就是说所有区域假定其他选择项可用。它不等同于解释为没有区域的语法 <code>&lt;Region = {}&gt;</code> (或等号右侧的任何其他文本默认生成空白元素集)。</p> </div>
<code>sum( {\$&lt;Year = {2000}, Region = {US, SE, DE, UK, FR}&gt;} Sales )</code>	返回当前选择项的销售额，但“Year”和“Region”中均有新选择项。
<code>sum( {\$&lt;~Ingredient = {"*garlic"}&gt;} Sales )</code>	返回当前选择项的销售额，但强制排除所有包含字符串“garlic”的原料。
<code>sum( {\$&lt;Year = {"2*"}&gt;} Sales )</code>	返回当前选择项的销售额，但在字段“Year”中选择所有以数字“2”开头的年份，例如：2000 年及之后的年份。
<code>sum( {\$&lt;Year = {"2*", "198*"}&gt;} Sales )</code>	与上例一样，但现在选择项中还包括 20 世纪 80 年代。
<code>sum( {\$&lt;Year = {"&gt;1978&lt;2004"}&gt;} Sales )</code>	与上例一样，但现在包含数字搜索，以便指定任意范围。

## 集合修饰符和集合运算符

字段中的选择项可以使用作用于不同元素集合的集合运算符定义。例如，修饰符 `<Year = {"20*", 1997} - {2000}>` 将选择以“20”开头的所有年份以及“1997”年，但“2000”年除外。

示例和结果：

示例	结果
<code>sum( {\$&lt;Product = Product + {OurProduct1} - {OurProduct2} &gt;} Sales )</code>	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”。
<code>sum( {\$&lt;Year = Year + ({"20*", 1997} - {2000}) &gt;} Sales )</code>	返回当前选择项的销售额，但字段“Year”字中包含附加选择项：1997 和 所有以“20”开头的年份，但是不包括 2000。  注意：如果当前选择项中包含 2000，它也将修改后被包括进来。
<code>sum( {\$&lt;Year = (Year + {"20*", 1997}) - {2000} &gt;} Sales )</code>	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。
<code>sum( {\$&lt;Year = {"*"} - {2000}, Product = {"*bearing*"} &gt;} Sales )</code>	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。

## 集合修饰符使用赋值和默认集合运算符

此表示法定义了新选择项，忽略了字段中的当前选择项。然而，如果想要选择项基于字段中的当前选择项，并添加字段值，例如，可能需要特定修饰符 `<Year = Year + {2007, 2008}>`。其简短等效的编写方式是 `<Year += {2007, 2008}>`，即以赋值运算符默认定义并集。同样，交集、补集和对称差集可由赋值运算符默认定义为“\*=”、“-=”和“/=”。

示例和结果：

示例	结果
<code>sum( {\$&lt;Product += {OurProduct1, OurProduct2} &gt;} Sales )</code>	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。
<code>sum( {\$&lt;Year += {"20*", 1997} - {2000} &gt;} Sales )</code>	返回当前选择项的销售额，但使用默认并集添加在选择中添加：1997 和 所有以“20”开头的年份，但是不包括 2000。  注意：如果当前选择项中包含 2000，它也将修改后被包括进来。如同 <code>&lt;Year=Year + ({"20*", 1997} - {2000})&gt;</code> 。
<code>sum( {\$&lt;Product *= {OurProduct1} &gt;} Sales )</code>	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。

## 集合修饰符和高级搜索

使用通配符和聚合的高级搜索可用于定义集合。

示例和结果：

示例	结果
<code>sum( {\$-1&lt;Product = {"*Internal*", "*Domestic*"}&gt;} Sales )</code>	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。
<code>sum( {\$&lt;Customer = {"=Sum({1&lt;Year = {2007}&gt;} Sales ) &gt; 1000000"}&gt;} Sales )</code>	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。

## 集合修饰符和货币符号扩展

变量和其他货币符号扩展可以在集合表达式中使用。

示例和结果：

示例	结果
<code>sum( {\$&lt;Year = {\$(#vLastYear)}&gt;} Sales )</code>	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 用于货币符号扩展。
<code>sum( {\$&lt;Year = {\$(#=Only (Year)-1)}&gt;} Sales )</code>	返回与当前选择项相关的上一年的销售额。在这里，货币符号扩展被用于计算上一年份。

## 集合修饰符和默认字段值定义

下面介绍如何使用嵌套集合定义来定义字段值集合。

在这种情况下，必须使用 Element 函数 P() 和 E()，分别呈现正值的元素集和字段排除值。在括号内，可以指定一个集合表达式和一个字段，例如 P({1} Customer)。这些函数不能在其他表达式中使用：

示例和结果：

示例	结果
<code>sum( {\$&lt;Customer = P({1&lt;Product={'Shoe'}&gt;} Customer)&gt;} Sales )</code>	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。
<code>sum( {\$&lt;Customer = P({1&lt;Product={'Shoe'}&gt;})&gt;} Sales )</code>	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。

示例	结果
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。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。

## 3.4 可视化表达式和聚合语法

可视化(图表)表达式和聚合所使用的语法如以下部分所述。

### 图表表达式的一般语法

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。

### 聚合的一般语法

aggrexpression ::= ( fieldref	
-------------------------------	--



operator1 aggexpression	
aggexpression operator2 aggexpression	
functioninaggr	
( aggexpression )	)

**fieldref** 是字段名。

`functionaggr ::= functionname ( parameters2 )`

表达式和函数因此可以自由放置，只要 **fieldref** 始终正好被一个聚合函数包围，并且如果表达式返回一个可解释的值，Qlik Sense 就不会给出任何错误信息。

## 4 运算符

本节介绍可在 Qlik Sense 中使用的运算符。可以使用两种类型的运算符：

- 一元运算符(只需要一个操作数)
- 二元运算符(需要两个操作数)

大多数运算符是二元的。

可定义以下运算符：

- 位运算符
- 逻辑运算符
- 数字运算符
- 关系运算符
- 字符串运算符

### 4.1 位运算符

所有位运算符可将操作数转换(截断)为带正负号的整数(32 位)，并以相同方式返回结果。逐位执行所有运算。如果不能将操作数解释为一个数字，该操作将返回 NULL。

**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

**>>**      位右移      该操作返回向右移的第一个操作数。步数在第二个操作数中进行定义。

**示例：**

8 >> 2 返回 2

**<<**      位左移      该操作返回向左移的第一个操作数。步数在第二个操作数中进行定义。

**示例：**

8 << 2 返回 32

## 4.2 逻辑运算符

所有逻辑运算符都可解释逻辑操作数并返回结果 True (-1) 或 False (0)。

**not**                      逻辑反。很少使用的一元运算符。此运算返回操作数的逻辑反值。

**and**                     逻辑与。此运算返回操作数的逻辑与。

**or**                      逻辑或。此运算返回操作数的逻辑或。

**Xor**                     逻辑异或。此运算返回操作数的逻辑异或。运算规则很像逻辑或，但不同的是，如果两个操作数都是 True，则结果为 False。

## 4.3 数字运算符

全部数字运算符使用数值式操作数并返回数值结果。

**+**                      正值(一元运算符)符号或算术加法。二元运算返回两个操作数的总和。

**-**                      负值(一元运算符)符号或算术减法。一元运算返回操作数乘以 -1 的结果，而二元运算返回这两个操作数的差值。

**\***                      算术乘法。此运算返回两个操作数的相乘结果。

**/**                      算术除法。此运算返回两个操作数之间的比率。

## 4.4 关系运算符

所有关系运算符均会比较操作数值，并返回 True (-1) 或 False (0) 作为结果。所有关系运算符均为二进制。

**<**                      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。

<b>&gt;=</b>	大于或等于	如果两个操作数可使用数值解释,即可进行数值比较。运算操作会返回比较评估的逻辑值。
<b>=</b>	等于	如果两个操作数可使用数值解释,即可进行数值比较。运算操作会返回比较评估的逻辑值。
<b>&lt;&gt;</b>	不等于	如果两个操作数可使用数值解释,即可进行数值比较。运算操作会返回比较评估的逻辑值。
<b>precedes</b>		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:**

```
' 11' precedes ' 2' 返回 True
```

比较该示例与以下示例:

```
' 11' < ' 2' 返回 False
```

<b>follows</b>		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:**

```
' 23' follows ' 111' 返回 True
```

比较该示例与以下示例:

```
' 23' > ' 111' 返回 False
```

## 4.5 字符串运算符

有两种字符串运算符。一种使用操作数的字符串值并返回字符串结果。另一种比较操作数,然后返回布尔值以表明匹配情况。

<b>&amp;</b>	字符串串联运算。此运算可以返回一个文本字符串,包含两个轮流操作数字符串。
--------------	--------------------------------------

**示例:**

```
'abc' & 'xyz' 返回“abcxyz”
```

<b>like</b>	字符串与通配符字符相比较。如果运算符之前的字符串与运算符之后的字符串相匹配,则此运算将返回布尔值 True (-1)。第二个字符串可能包含星号 (*) 通配符(任意数量的任意字符)或问号 (?) (一个任意字符)。
-------------	---

**示例：**

'abc' like 'a\*' 用于返回 True (-1)

'abcd' like 'a?c\*' 用于返回 True (-1)

'abc' like 'a??bc' 用于返回 False (0)

## 5 脚本和图表表达式中的函数

本部分介绍可用于 Qlik Sense 数据加载脚本和图表表达式以转换和聚合数据的函数。

许多函数能够以相同的方式用于数据加载脚本和图表表达式，但也有一些例外：

- 一些函数只能用于数据加载脚本，称为脚本函数。
- 一些函数只能用于图表表达式，称为图表函数。
- 一些函数可用于数据加载脚本和图表表达式，但在参数和应用方面存在差异。在不同的主题中分别介绍了脚本函数或图表函数。

### 5.1 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返回单个结果的函数，在此类函数中，聚合通过图表维度或脚本中的 **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。

#### 在数据加载脚本中使用聚合函数

聚合函数只能在 **LOAD** 语句内使用。

#### 在图表表达式中使用聚合函数

单个聚合函数的参数表达式不能包含其他聚合函数。

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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。

#### 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** 函数用于高级聚合，其中在另一个聚合函数中要将 **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。

**Syntax:**

```
Aggr ({SetExpression} [DISTINCT] [NODISTINCT] expr, dim{, dimension})
```

**Return data type:** 双

**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。
dim	维度确定了表达式中值的阵列。这是一个单一字段，并且不能为表达式。
dimension	可选。可以通过表达式进一步扩展一个或多个维度。
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 <b>distinct</b> 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。
NODISTINCT	如果表达式参数前面是 <b>nodistinct</b> 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 <b>aggr</b> 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时返回单个数值。例如，通过合计 **Aggr()** 报表中客户的销售额计算销售额平均值，然后计算合计结果的平均值：**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此函数。

#### 限制：

每个维度必须是单个字段，不能是表达式(计算维度)。

#### 示例和结果：

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	-	19

创建一个以 **Customer**、**Product**、**UnitPrice** 和 **UnitSales** 为维度的表格。

示例	结果
Avg(Aggr(Sum (UnitSales*UnitPrice), Customer))	<p>在表格中添加表达式作为度量。Aggr(Sum(UnitSales*UnitPrice), Customer) 的值。该表格可查找 <b>Customer</b> 的销售额总值，然后返回一组值：三个 <b>Customer</b> 值 295、715 和 120。</p> <p>这些值可用作 <b>Avg()</b> 函数的导入值，以查找销售额平均值 376.6667。(您必须已经选择属性面板中 <b>Presentation</b> 下的 <b>Totals</b>。)</p>

#### 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|25
Canutility|AA|8|15
Canutility|CC|1|19
] (delimiter is '|');
```

#### 另请参阅：

□ 基本聚合函数(第 144 页)

## 基本聚合函数

### 基本聚合函数概述

基本聚合函数是一组最常用的聚合函数。

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

### 数据加载脚本中的基本聚合函数

#### FirstSortedValue

**FirstSortedValue()** 将返回来自 **value** 指定表达式的值，相当于 **sort\_weight** 参数排序的结果，例如，单价最低的产品名称。排序顺序中的第 **n** 个值，可在 **rank** 中指定。如果在指定 **rank** 下多个结果值共享同一 **sort\_weight**，则此函数返回 NULL。排序的值会迭代于 **group by** 子句定义的大量记录，或者如果 **group by** 子句未定义，就会在整个数据集之间聚合。

```
FirstSortedValue ([ distinct ] expression, sort_weight [, rank ])
```

#### Max

**Max()** 用于查找表达式中聚合数据的最高数值，该数值由 **group by** 子句定义。通过指定 **rank n**，可以



查找第 n 个最高值。

```
Max ( expression[, rank])
```

### Min

**Min()** 用于返回表达式中聚合数据的最低数值, 该数值由 **group by** 子句定义。通过指定 **rank n**, 可以查找第 n 个最低值。

```
Min ( expression[, rank])
```

### Mode

**Mode()** 用于返回表达式中聚合数据的最常出现的值(即模式值), 该值由 **group by** 子句定义。**Mode()** 函数用于返回数字值和文本值。

```
Mode (expression )
```

### Only

**Only()** 用于返回一个值(如果从聚合数据得出一个且只有一个可能值)。如果记录只包含一个值, 则返回该值, 否则返回 NULL 值。使用 **group by** 子句计算多个记录的值。**Only()** 函数用于返回数字值和文本值。

```
Only (expression )
```

### Sum

**Sum()** 用于计算表达式中聚合的值的总和, 该总和由 **group by** 子句定义。

```
Sum ([distinct]expression)
```

## 图表表达式中的基本聚合函数

图表聚合函数只能在图表表达式的字段中使用。单个聚合函数的参数表达式不能包含其他聚合函数。

### FirstSortedValue

**FirstSortedValue()** 将返回来自 **value** 指定表达式的值, 相当于 **sort\_weight** 参数排序的结果, 例如, 单价最低的产品名称。排序顺序中的第 n 个值, 可在 **rank** 中指定。如果在指定 **rank** 下多个结果值共享同一 **sort\_weight**, 则此函数返回 NULL。

```
FirstSortedValue - 图表函数 ([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] value, sort_weight [,rank])
```

### Max

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

```
Max - 图表函数 ([{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。

**Syntax:**

**FirstSortedValue** ([ distinct ] value, sort-weight [, rank ])

**Return data type:** 双

**参数:**

参数	说明
value Expression	此函数用于查找表达式 <b>value</b> 的值,相当于 <b>sort_weight</b> 的排序结果。
sort-weight Expression	该表达式包含要排序的数据。找到 <b>sort_weight</b> 的第一个(最低)值,由 <b>value</b> 表达式的对应值确定。如果在 <b>sort_weight</b> 前面加一个减号,则此函数会返回最后一个(最高)排序值。
rank Expression	通过指定一个大于 1 的 <b>rank</b> “n”,您会获得第 n 个排序值。
distinct	如果在函数参数前出现单词 <b>DISTINCT</b> ,则将忽略计算该函数参数生成的副本。

**示例和结果:**

添加示例脚本到应用程序并运行。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
<pre>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;</pre>	<pre>Customer MyProductWithSmallestOrderByCustomer Astrida CC Betacab AA Canutility AA Divadip DD</pre> <p>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。</p> <p>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，CC 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。</p>
<p>前提是 <b>Temp</b> 表格像之前的示例一样加载：</p> <pre>LOAD Customer,FirstSortedValue(Product, - UnitSales) as MyProductWithLargestOrderByCustomer Resident Temp Group By Customer;</pre>	<pre>Customer MyProductWithLargestOrderByCustomer Astrida AA Betacab DD Canutility CC Divadip -</pre> <p>减号位于 sort_weight 参数之首，因此，该函数将最大的排在第一个。</p> <p>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。</p>
<p>前提是 <b>Temp</b> 表格像之前的示例一样加载：</p> <pre>LOAD Customer,FirstSortedValue(distinct Product, -UnitSales) as MyProductWithSmallestOrderByCustomer Resident Temp Group By Customer;</pre>	<pre>Customer MyProductWithLargestOrderByCustomer Astrida AA Betacab DD Canutility CC Divadip AA</pre> <p>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。</p>

## 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。

## Syntax:

```
FirstSortedValue([SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] value,
sort_weight [,rank])
```

## Return data type: 双

## 参数:

参数	说明
value	输出字段。此函数用于查找表达式 <b>value</b> 的值，相当于 <b>sort_weight</b> 的排序结果。
sort_weight	输入字段。该表达式包含要排序的数据。找到 <b>sort_weight</b> 的第一个(最低)值，由 <b>value</b> 表达式的对应值确定。如果在 <b>sort_weight</b> 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 <b>rank</b> “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 <b>DISTINCT</b> ，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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

## 示例和结果:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20

Customer	Product	UnitSales	UnitPrice
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19

示例	结果
firstsortedvalue (Product, UnitPrice)	BB, 这是具有最低 Product(9) 的 UnitPrice。
firstsortedvalue (Product, UnitPrice, 2)	BB, 这是具有第二低 UnitPrice(10) 的 Product。
firstsortedvalue (Customer, - UnitPrice, 2)	Betacab, 这是具有第二高 customer(20) 的 Product 的 UnitPrice。
firstsortedvalue (Customer, UnitPrice, 3)	NULL, 因为有两个相同 customer(第三低) Astrida(15) 的 Canutility 值(rank 和 UnitPrice)。  使用 distinct 限定符来确保不会出现意外的空结果。
firstsortedvalue (Customer, - UnitPrice*UnitSales, 2)	Canutility, 这是具有第二高销售订单值(Customer 乘以 UnitPrice (120)) 的 UnitSales。

#### 示例中所使用的数据:

```
ProductData:
LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD||25
Canutility|AA|8|15
Canutility|CC||19
] (delimiter is '|');
```

## Max

**Max()** 用于查找表达式中聚合数据的最高数值, 该数值由 **group by** 子句定义。通过指定 **rank n**, 可以查找第 **n** 个最高值。

#### Syntax:

```
Max ( expr [, rank] )
```

**Return data type:** 数字

#### 参数:

参数	说明
expr Expression	表达式或字段包含要度量的数据。
rank Expression	<b>rank</b> 的默认值为 1，相当于最高值。通过指定 <b>rank</b> 为 2，将返回第二个最高值。如果指定 <b>rank</b> 为 3，将返回第三个最高值，以此类推。

### 示例和结果：

添加示例脚本到应用程序并运行。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	
<b>Temp:</b> 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 ' ');  <b>Max:</b> LOAD Customer, Max(UnitSales) as MyMax, Resident Temp Group By Customer;	Customer	MyMax
	Astrida	18
	Betacab	5
	Canutility	8
前提是 <b>Temp</b> 表格像之前的示例一样加载：  LOAD Customer, Max(UnitSales,2) as MyMaxRank2 Resident Temp Group By Customer;	Customer	MyMaxRank2
	Astrida	10
	Betacab	4
	Canutility	-

## Max - 图表函数

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



您可能还想查看 **FirstSortedValue** 和 **rangemax**，其功能与 **Max** 函数相似。

### Syntax:

```
Max ([{SetExpression}] [TOTAL [<fld {,fld}>]] expr [,rank])
```

**Return data type:** 数字

**参数:**

参数	说明
expr	表达式或字段包含要度量的数据。
rank	<b>rank</b> 的默认值为 1, 相当于最高值。通过指定 <b>rank</b> 为 2, 将返回第二个最高值。如果指定 <b>rank</b> 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 <b>TOTAL</b> , 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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**示例和结果:**

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	结果
Max(UnitSales)	10, 因为这是 unitSales 中的最大值。
订单的值通过销售的单位数量 (UnitSales) 乘以单位价格计算得出。 Max(UnitSales*UnitPrice)	150, 因为这是计算 (UnitSales)*(UnitPrice) 所有可能值的结果的最大值。
Max(UnitSales, 2)	9, 这是第二大值。

示例	结果
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。
选择 Customer B。 Max({1} TOTAL UnitSales)	10, 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 '|');
```

#### 另请参阅:

▢ *FirstSortedValue* - 图表函数(第 148 页)

▢ *RangeMax*(第 534 页)

## 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。

#### Syntax:

```
Min ( expr [, rank] )
```

**Return data type:** 数字

#### 参数:

参数	说明
expr Expression	表达式或字段包含要度量的数据。
rank Expression	<b>rank</b> 的默认值为 1, 相当于最小值。通过指定 <b>rank</b> 为 2, 将返回第二个最小值。如果指定 <b>rank</b> 为 3, 将返回第三个最小值, 以此类推。



**示例和结果：**

添加示例脚本到应用程序并运行。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	
<b>Temp:</b> 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 ' ');  <b>Min:</b> LOAD Customer, Min(UnitSales) as MyMin Resident Temp Group By Customer;	Customer	MyMin
	Astrida	2
	Betacab	4
	Canutility	8
前提是 <b>Temp</b> 表格像之前的示例一样加载：  LOAD Customer, Min(UnitSales,2) as MyMinRank2 Resident Temp Group By Customer;	Customer	MyMinRank2
	Astrida	9
	Betacab	5
	Canutility	-

**Min - 图表函数**

**Min()** 用于查找聚合数据中的最低值。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。

**Syntax:**

```
Min([SetExpression] [TOTAL [<fld {,fld}>]]) expr [,rank])
```

**Return data type:** 数字

**参数：**

参数	说明
expr	表达式或字段包含要度量的数据。
rank	<b>rank</b> 的默认值为 1, 相当于最高值。通过指定 <b>rank</b> 为 2, 将返回第二个最高值。如果指定 <b>rank</b> 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 <b>TOTAL</b> , 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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

#### 示例和结果:

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15
Astrida	BB	9	9
Betacab	BB	5	10
Betacab	CC	2	20
Betacab	DD	-	25
Canutility	AA	8	15
Canutility	CC	-	19



*Min() 函数必须根据表达式所提供值的阵列返回一个非 NULL 值(如果有)。因此, 在这些示例中, 由于数据中有 NULL 值, 函数将返回根据表达式评估的第一个非 NULL 值。*

示例	结果
Min(UnitSales)	2, 因为此值是 UnitSales 中的最小非 NULL 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。  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 值。
选择 Customer B。  Min({1} TOTAL UnitSales)	40, 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 '|');
```

#### 另请参阅:

- *FirstSortedValue* - 图表函数(第 148 页)
- *RangeMin*(第 537 页)

## 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。

#### Syntax:

```
Mode ( expr )
```

**Return data type:** 双

参数	说明
expr Expression	表达式或字段包含要度量的数据。

**限制：**

如果不只有一个值经常出现，则返回 NULL。

**示例和结果：**

添加示例脚本到应用程序并运行。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
<pre>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;</pre>	<p>MyMostOftenSoldProduct</p> <p>AA</p> <p>因为 AA 是唯一售出过多次的产品。</p>

**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。

**Syntax:**

```
Mode ({[SetExpression] [TOTAL [<fld {,fld}>]]} expr)
```

**Return data type:** 双

**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
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	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**示例和结果：**

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	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 '|');

```

另请参阅：

▢ [Avg - 图表函数\(第 193 页\)](#)

▢ [Median - 图表函数\(第 222 页\)](#)

## 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。

**Syntax:**

```
Only ( expr )
```

**Return data type:** 双

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：**

添加示例脚本到应用程序并运行。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	
<pre>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;</pre>	<p>Customer</p> <p>Astrida</p>	<p>MyUniqIDCheck</p> <p>1</p> <p>因为只有客户 Astrida 拥有包括 CustomerID 的完整记录。</p>

## 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。

### Syntax:

```
Only([SetExpression]) [TOTAL [<fld {,<fld>>]] expr)
```

**Return data type:** 双

### 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	<p>如果在函数参数前面出现单词 <b>TOTAL</b>,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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>



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:

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	结果
Only({<UnitPrice={9}>> Product})	BB, 因为这是 Product 为 9 的唯一 UnitPrice。
Only({<Product={DD}>> Customer})	B, 因为销售 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。

#### Syntax:

```
sum ( [ distinct] expr)
```

**Return data type:** 数字



参数：

参数	说明
distinct	如果在表达式前面出现单词 <b>distinct</b> , 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：

添加示例脚本到应用程序并运行。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	
<pre>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;</pre>	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。


**Syntax:**

```
Sum([[{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr])
```

**Return data type:** 数字

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
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	<p>如果在函数参数前出现单词 <b>DISTINCT</b>，则将忽略计算该函数参数生成的副本。</p> <div>  虽然支持 <b>DISTINCT</b> 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。 </div>
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

#### 示例和结果：

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	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。
选择 Customer B。  Sum({1} TOTAL UnitSales*UnitPrice)	505，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 '|');
```

### 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。

**Syntax:**

```
Count( [distinct ] expr)
```

**Return data type:** 整数

**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 <b>distinct</b> ，则将忽略所有重复值。

**示例和结果:**

添加示例脚本到应用程序并运行。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
<pre>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;</pre>	<pre>Customer OrdersByCustomer Astrida 3 Betacab 3 Canutility 2 Divadip 2</pre> <p>只要表格中的表格包含维度 Customer，否则 OrdersByCustomer 的结果为 3, 2。</p>
前提是 <b>Temp</b> 表格像之前的示例一样加载：	<pre>TotalOrderNumber 10</pre>
<p>前提是 <b>Temp</b> 表格像第一个示例一样加载：</p> <pre>LOAD Count(distinct OrderNumber) as TotalOrdersNumber Resident Temp;</pre>	<pre>TotalOrderNumber 9</pre> <p>因为 OrderNumber 的两个值都具有相同的值 1。</p>

### 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。

**Syntax:**

```
Count ({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

**Return data type:** 整数

**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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

#### 示例和结果：

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	结果
Count(OrderNumber)	<p>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。</p> <div>  <p>“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。</p> </div>
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。
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。

示例	结果
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。

**Syntax:**

```
MissingCount ( [ distinct ] expr)
```

**Return data type:** 整数

**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 <b>distinct</b> , 则将忽略所有重复值。

**示例和结果:**

添加示例脚本到应用程序并运行。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
<pre>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;</pre>	<p>Customer MissingOrdersByCustomer Astrida 0 Betacab 1 Canutility 2 Divadip 0</p> <p>第二个语句指定:</p> <p>TotalMissingCount 3 在包含该维度的表格中。</p>
<p>前提是 <b>Temp</b> 表格像之前的示例一样加载:</p> <pre>LOAD MissingCount(distinct OrderNumber) as TotalMissingCountDistinct Resident Temp;</pre>	<p>TotalMissingCountDistinct 1 因为只有一个 OrderNumber 缺少一个值。</p>

## 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。

### Syntax:

```
MissingCount({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

**Return data type:** 整数


### 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 <b>DISTINCT</b> , 则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>, 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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>



示例和结果：

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	结果
MissingCount ([OrderNumber])	3, 因为 10 个 OrderNumber 字段中有 3 个是空白  <div>  “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。 </div>
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。

### Syntax:

```
NullCount ( [ distinct ] expr)
```

**Return data type:** 整数

### 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 <b>distinct</b> , 则将忽略所有重复值。

### 示例和结果:

添加示例脚本到应用程序并运行。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
<pre>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;</pre>	<p>Customer NullOrdersByCustomer</p> <p>Astrida 0</p> <p>Betacab 0</p> <p>Canutility 1</p> <p>第二个语句指定:</p> <p>TotalNullCount</p> <p>1</p> <p>在包含该维度的表格中，因为只有一条记录包含 NULL 值。</p>

## NullCount - 图表函数

**NullCount()** 用于聚合每个图表维度中 NULL 值的数量。

**Syntax:**

```
NullCount({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

**Return data type:** 整数

**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 <b>DISTINCT</b> , 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 <b>TOTAL</b> , 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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**示例和结果:**

示例	结果
NullCount([OrderNumber])	1, 因为我们在内联 <b>LOAD</b> 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。

**Syntax:**

```
NumericCount ( [ distinct ] expr)
```

**Return data type:** 整数

**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 <b>distinct</b> , 则将忽略所有重复值。

**示例和结果:**

添加示例脚本到应用程序并运行。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										
<pre>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;</pre>	<table> <tr> <th>Customer</th><th>NumericCountByCustomer</th></tr> <tr> <td>Astrida</td><td>3</td></tr> <tr> <td>Betacab</td><td>2</td></tr> <tr> <td>Canutility</td><td>0</td></tr> <tr> <td>Divadip</td><td>2</td></tr> </table>	Customer	NumericCountByCustomer	Astrida	3	Betacab	2	Canutility	0	Divadip	2
Customer	NumericCountByCustomer										
Astrida	3										
Betacab	2										
Canutility	0										
Divadip	2										
<pre>LOAD NumericCount(OrderNumber) as TotalNumericCount Resident Temp;</pre>	<p>第二个语句指定:</p> <p>TotalNumericCount</p> <p>7</p> <p>在包含该维度的表格中。</p>										
<p>前提是 <b>Temp</b> 表格像之前的示例一样加载:</p> <pre>LOAD NumericCount(distinct OrderNumber) as TotalNumericCountDistinct Resident Temp;</pre>	<p>TotalNumericCountDistinct</p> <p>6</p> <p>因为只有一个 OrderNumber 与另一个重复, 因此结果为 6 不会重复。</p>										

## NumericCount - 图表函数

**NumericCount()** 用于聚合每个图表维度中数值的数量。

## Syntax:

```
NumericCount ({ [SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]] } expr)
```

**Return data type:** 整数


## 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 <b>DISTINCT</b> ，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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

## 示例和结果:

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	结果
NumericCount ([OrderNumber])	7, 因为 OrderNumber 中的 10 个字段中有 3 个是空白。  <div>  “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。 </div>
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。

**Syntax:**

```
TextCount ( [ distinct ] expr)
```

**Return data type:** 整数

**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 <b>distinct</b> ，则将忽略所有重复值。

**示例和结果：**

添加示例脚本到应用程序并运行。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										
<pre>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;</pre>	<table> <thead> <tr> <th>Customer</th><th>ProductTextCount</th></tr> </thead> <tbody> <tr> <td>Astrida</td><td>3</td></tr> <tr> <td>Betacab</td><td>3</td></tr> <tr> <td>Canutility</td><td>2</td></tr> <tr> <td>Divadip</td><td>2</td></tr> </tbody> </table>	Customer	ProductTextCount	Astrida	3	Betacab	3	Canutility	2	Divadip	2
Customer	ProductTextCount										
Astrida	3										
Betacab	3										
Canutility	2										
Divadip	2										
<pre>LOAD Customer,TextCount(OrderNumber) as OrderNumberTextCount Resident Temp Group By Customer;</pre>	<table> <thead> <tr> <th>Customer</th><th>OrderNumberTextCount</th></tr> </thead> <tbody> <tr> <td>Astrida</td><td>0</td></tr> <tr> <td>Betacab</td><td>1</td></tr> <tr> <td>Canutility</td><td>2</td></tr> <tr> <td>Divadip</td><td>0</td></tr> </tbody> </table>	Customer	OrderNumberTextCount	Astrida	0	Betacab	1	Canutility	2	Divadip	0
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。

**Syntax:**

```
TextCount ({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

**Return data type:** 整数

**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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

#### 示例和结果：

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	结果
TextCount([Product])	<p>10, 因为 Product 中的 10 个字段全部是文本字段。</p> <div>  <p>“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。</p> </div>
TextCount([OrderNumber])	3, 因为已对空白单元格计数。通常将使用此函数检查是否有数字字段包含文本值或为非零值。
TextCount (DISTINCT [Product])/Count ([Product])	对 Product 不同文本值的所有数量 (4) 计数，并除以 Product 值的总数 (10)。结果为 0.4。

示例中所使用的数据：



```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|1|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB|||| 25
Canutility|AA||||15
Canutility|CC||||19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

## 财务聚合函数

本部分介绍财务运作中与付款和现金流相关的聚合函数。

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

### 数据加载脚本中的财务聚合函数

#### IRR

**IRR()** 函数用于返回聚合内部回报率，以揭示迭代于 group by 子句定义的大量记录上的表达式的数值表示的现金流系列。

```
IRR (expression)
```

#### XIRR

**XIRR()** 函数用于返回聚合内部回报率，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表(不必为周期性的)。所有付款全年折扣。

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

#### NPV

**NPV()** 用于根据迭代于 group by 子句定义的大量记录上的 **value** 的数值表示的一系列未来付款(负值)和收入(正值)以及每周期的 **discount\_rate**, 返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

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

#### XNPV

**XNPV()** 函数用于返回聚合净现值，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表(不必为周期性的)。比率为每周期的利率。所有付款全年折扣。

```
XNPV (rate, valueexpression, dateexpression)
```

### 图表表达式中的财务聚合函数

以下财务聚合函数可用于图表中。

### IRR

**IRR()** 用于返回通过在图表维度上迭代的 **value** 指定表达式中数值表示的一系列现金流的聚合内部回报率。

**IRR - 图表函数** ([TOTAL [<fld {,fld}>]] value)

### NPV

**NPV()** 用于根据每周期的 **discount\_rate** 和通过图表维度迭代 **value** 的数值表示的一系列未来付款 (负值) 和收入 (正值) 返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

**NPV - 图表函数** ([TOTAL [<fld {,fld}>]] discount\_rate, value)

### XIRR

**XIRR()** 用于返回通过图表维度迭代 **pmt** 和 **date** 指定表达式中成对数值表示的现金流时间表 (即不一定是周期性的)。所有付款全年折扣。

**XIRR - 图表函数 (第 183 页)** ([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) 的聚合净现值。所有付款全年折扣。

**XNPV - 图表函数** ([TOTAL [<fld {,fld}>]] discount\_rate, pmt, date)

### IRR

**IRR()** 函数用于返回聚合内部回报率，以揭示迭代于 group by 子句定义的大量记录上的表达式的数值表示的现金流系列。

这些现金流不必是均值，因为它们可用于年金。但是，现金流必须定期出现，例如每月或每年。内部回报率是指投资回报的利率，该利率由定期出现的支出 (负值) 和收入 (正值) 构成。计算函数至少需要一个正值和一个负值。

#### Syntax:

**IRR** (value)

**Return data type:** 数字

#### 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:

添加示例脚本到应用程序并运行。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。

**示例和结果：**

示例	结果	
<b>Cashflow:</b> 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 ' ');  <b>Cashflow1:</b> LOAD Year,IRR(Payments) as IRR2013 Resident Cashflow Group By Year;	Year  2013	IRR2013  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。

**Syntax:**

```
IRR([TOTAL [<fld {,fld}>]] value)
```

**Return data type:** 数字

**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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

文本值，NULL 值和缺失值都忽略不计。

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。  <div>  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。 </div>

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 '|');
```

另请参阅：

- ▢ [XIRR - 图表函数\(第 183 页\)](#)
- ▢ [Aggr - 图表函数\(第 142 页\)](#)

## NPV

**NPV()** 用于根据迭代于 group by 子句定义的大量记录上的 **value** 的数值表示的一系列未来付款(负值)和收入(正值)以及每周期的 **discount\_rate**，返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

**Syntax:**

**NPV**(discount\_rate, value)

**Return data type:** 数字。结果默认采用货币数字格式。

**参数：**

参数	说明
discount_rate	<b>discount_rate</b> 是在整个期间的折扣率。
value	表达式或字段包含要度量的数据。

**限制:**

文本值, NULL 值和缺失值都忽略不计。

**示例和结果:**

添加示例脚本到应用程序并运行。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。

示例	结果		
<b>Cashflow:</b> 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 ' ');  <b>Cashflow1:</b> LOAD Year,NPV(0.2, Payments) as NPV1_2013 Resident Cashflow Group By Year;	Year	NPV1_2013	
	2013	-\$540.12	
前提是 <b>Cashflow</b> 表格像之前的示例一样加载: LOAD Year,NPV(Discount, Payments) as NPV2_2013 Resident Cashflow Group By Year, Discount; 注意, Group By 子句按 Year 和 Discount 对结果进行 排序。将第一个参数 discount_rate 指定为字段 (Discount), 而不是一个特定数字, 因此需要第二个排 序标准。字段可以包含不同的值, 因此必须对聚合记 录进行排序以允许不同的 Year 和 Discount 值。	Year	Discount	NPV2_2013
	2013	0.1	-\$3456.05
	2013	0.2	\$5666.67

**NPV - 图表函数**

**NPV()** 用于根据每周期的 **discount\_rate** 和通过图表维度迭代 **value** 的数值表示的一系列未来付款 (负值) 和收入 (正值) 返回投资聚合净现值。假设为在每个周期结束时发生的付款和收入。

**Syntax:**

```
NPV([TOTAL [<fld {,fld}>]] discount_rate, value)
```

**Return data type:** 数字 结果默认采用货币数字格式。

**参数:**

参数	说明
discount_rate	<b>discount_rate</b> 是在整个期间的折扣率。
value	表达式或字段包含要度量的数据。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p> <p><b>TOTAL</b> 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。</p>

**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，请结合计算维度使用 **Aggr** 函数。

文本值，NULL 值和缺失值都忽略不计。

**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 '|');
```

**另请参阅：**

- [XNPV - 图表函数\(第 186 页\)](#)
- [Aggr - 图表函数\(第 142 页\)](#)

## XIRR

**XIRR()** 函数用于返回聚合内部回报率，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表(不必为周期性的)。所有付款全年折扣。

### Syntax:

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

**Return data type:** 数字

### 参数:

参数	说明
pmt	付款。表达式或字段包含与在 <b>date</b> 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 <b>pmt</b> 中指定的现金流支付对应的日期时间表。

### 限制:

数据对的任意部分或两部分内存在文本值、NULL 值和缺失值会导致整个数据对被忽略。

### 示例和结果:

添加示例脚本到应用程序并运行。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。

示例	结果	
Cashflow: LOAD 2013 as Year, * inline [ Date Discount Payments 2013-01-01 0.1 -10000 2013-03-01 0.1 3000 2013-10-30 0.1 4200 2014-02-01 0.2 6800 ] (delimiter is ' ');  Cashflow1: LOAD Year,XIRR(Payments, Date) as XIRR2013 Resident Cashflow Group By Year;	Year	XIRR2013
	2013	0.5385

## XIRR - 图表函数

**XIRR()** 用于返回通过图表维度迭代 **pmt** 和 **date** 指定表达式中成对数值表示的现金流时间表(即不一定是周期性的)。所有付款全年折扣。

### Syntax:

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

**Return data type:** 数字

**参数:**

参数	说明
pmt	付款。表达式或字段包含与在 <b>date</b> 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 <b>pmt</b> 中指定的现金流支付对应的日期时间表。
TOTAL	如果在函数参数前面出现单词 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**限制:**

**pmt** 和 **date** 不能包含聚合函数，除非这些内部聚合包含 **TOTAL** 限定符。有关高级嵌套聚合函数的更多信息，请结合计算维度使用 **Aggr** 函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

**示例和结果:**

示例	结果
XIRR(Payments, Date)	0.5385

**示例中所使用的数据:**

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Discount|Payments
2013-01-01|0.1|-10000
2013-03-01|0.1|3000
2013-10-30|0.1|4200
2014-02-01|0.2|6800
] (delimiter is '|');
```

**另请参阅:**

- [IRR - 图表函数\(第 179 页\)](#)
- [Aggr - 图表函数\(第 142 页\)](#)



## XNPV

**XNPV()** 函数用于返回聚合净现值，以揭示迭代于 group by 子句定义的大量记录上的 **pmt** 和 **date** 表达式的成对数值表示的现金流明细表(不必为周期性的)。比率为每周期的利率。所有付款全年折扣。

### Syntax:

```
XNPV(discount_rate, pmt, date)
```

**Return data type:** 数字。结果默认采用货币数字格式。。

### 参数:

参数	说明
discount_rate	<b>discount_rate</b> 是在整个期间的折扣率。
pmt	表达式或字段包含要度量的数据。
date	表达式或字段包含与在 <b>pmt</b> 中指定的现金流支付对应的日期时间表。

### 限制:

数据对的任意部分或两部分内存在文本值、NULL 值和缺失值会导致整个数据对被忽略。

### 示例和结果:

添加示例脚本到应用程序并运行。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。

示例	结果	
<b>Cashflow:</b> 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 ' ');  <b>Cashflow1:</b> LOAD Year,XNPV(0.2, Payments, Date) as XNPV1_2013 Resident Cashflow Group By Year;	Year  2013	XNPV1_2013  \$2104.37

示例	结果		
前提是 <b>Cashflow</b> 表格像之前的示例一样加载： LOAD Year,XNPV(Discount, Payments, Date) as XNPV2_2013 Resident Cashflow Group By Year, Discount; 注意，Group By子句按 Year和 Discount 对结果进行排序。将第一个参数 discount_rate 指定为字段 (Discount), 而不是一个特定数字，因此需要第二个排序标准。字段可以包含不同的值，因此必须对聚合记录进行排序以允许不同的 Year和 Discount 值。	Year	Discount	XNPV2_2013
	2013	0.1	-\$3164.35
	2013	0.2	\$6800.00

## XNPV - 图表函数

**XNPV()** 用于返回通过图表维度迭代 **pmt** 和 **date** 指定表达式中成对数值表示的现金流时间表(不一定是周期性)的聚合净现值。所有付款全年折扣。

### Syntax:

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

**Return data type:** 数字 结果默认采用货币数字格式。

### 参数:

参数	说明
discount_rate	<b>discount_rate</b> 是在整个期间的折扣率。
pmt	付款。表达式或字段包含与在 <b>date</b> 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 <b>pmt</b> 中指定的现金流支付对应的日期时间表。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

### 限制:

**discount\_rate**、**pmt** 和 **date** 不能包含聚合函数，除非这些内部聚合包含 **TOTAL** 或 **ALL** 限定符。有关高级嵌套聚合函数的更多信息，请结合计算维度使用 **Aggr** 函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

### 示例和结果:

示例	结果
XNPV(Discount, Payments, Date)	-\$3164.35

### 示例中所使用的数据：

```
Cashflow:
LOAD 2013 as Year, * inline [
Date|Discount|Payments
2013-01-01|0.1|-10000
2013-03-01|0.1|3000
2013-10-30|0.1|4200
2014-02-01|0.2|6800
] (delimiter is '|');
```

### 另请参阅：

▢ [NPV - 图表函数\(第 181 页\)](#)

▢ [Aggr - 图表函数\(第 142 页\)](#)

## 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分位数(位数)对应的值。

```
Fractile (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分位数(位数)对应的值。

```
Fractile - 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 ( $r^2/(1-r^2)$ ), 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)
```

**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。

**Syntax:**

**Avg** ([DISTINCT] expr)

**Return data type:** 数字

**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 <b>distinct</b> , 则将忽略所有重复值。

**示例和结果:**

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>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;</pre>	<pre>Customer MyAverageSalesByCustomer Astrida 48.916667 Betacab 44.916667 Canutility 56.916667 Divadip 63.083333</pre> <p>这可以通过创建包括以下度量的表格在表格中进行检查:</p> <pre>Sum(Sales)/12</pre>



示例	结果
<p>前提是 <b>Temp</b> 表格像之前的示例一样加载：</p> <pre>LOAD Customer,Avg(DISTINCT Sales) as MyAvgSalesDistinct Resident Temp Group By Customer;</pre>	<pre>Customer MyAverageSalesByCustomer Astrida 43.1 Betacab 43.909091 Canutility 55.909091 Divadip 61</pre> <p>只会对特殊值进行计数。用总数除以非重复值的个数。</p>

## 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。

### Syntax:

```
Avg ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

**Return data type:** 数字

### 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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

### 限制:

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

### 示例和结果:

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

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
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

Customer	Sum([Sales])	Avg([Sales])	Avg(TOTAL Sales)	Avg(DISTINCT Sales)	Avg({1} TOTAL Sales)
	2566	53.46	53.458333	51.862069	53.458333
Astrida	587	48.92	53.458333	43.1	53.458333
Betacab	539	44.92	53.458333	43.909091	53.458333
Canutility	683	56.92	53.458333	55.909091	53.458333
Divadip	757	63.08	53.458333	61	53.458333

示例	结果
Avg(Sales)	对于包含维度 Customer 和度量 Avg([Sales]) 的表格，如果显示合计，则结果为 2566。
Avg([TOTAL (Sales)])	对于 Customer 的全部值，结果为 53.458333，因为 TOTAL 限定符意味着忽略维度。
Avg(DISTINCT (Sales))	合计为 51.862069，因为使用 Distinct 限定符意味着仅评估每个 Sales Customer 中的唯一值。

### 示例中所使用的数据：

```

Monthnames:
LOAD * INLINE [
Month, 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 (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 '|');

```

要按正确顺序对月份进行排序，在创建可视化后，转到属性面板的 **Sorting** 部分，选择 **Month**，然后勾选复选框 **Sort by expression**。在表达式框中，输入 Monthnumber。

另请参阅：

▢ [Aggr - 图表函数 \(第 142 页\)](#)

### 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)。

#### Syntax:

```
Correl(value1, value2)
```

**Return data type:** 数字

#### 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：

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>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;</pre>	<p>在包含维度 <code>Correl_Salary</code> 的表格中，在数据加载脚本中计算的 <code>Correl()</code> 结果将显示:0.9270611</p>

## 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)。

### Syntax:

```
Correl ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] value1, value2 )
```

**Return data type:** 数字

### 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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

### 限制:

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

数据对任意部分或两部分中的文本值，NULL 值和缺失值在整个数据对中忽略不计。

### 示例和结果：

示例	结果
<code>Correl(Age, Salary)</code>	对于包含维度 <code>Employee name</code> 和度量 <code>Correl(Age, Salary)</code> 的表格，结果为 0.9270611。仅显示合计单元格的结果。
<code>Correl(TOTAL Age, Salary)</code>	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 <code>Gender</code> 的筛选器面板，并在其中做出选择，则在选择 <code>Female</code> 时，您将看到结果 0.951；在选择 <code>Male</code> 时，您将看到结果 0.939。这是因为此选择项排除了不属于 <code>Gender</code> 的其他值的所有结果。
<code>Correl({1} TOTAL Age, Salary)</code>	0.927.与选择项无关。这是因为集合表达式 <code>{1}</code> 忽略了所有选择项和维度。
<code>Correl(TOTAL &lt;Gender&gt; Age, Salary)</code>	在合计单元格中，结果为 0.927，对于 <code>Male</code> 的全部值，结果为 0.939，对于 <code>Female</code> 的全部值，结果为 0.951。此结果相当于在筛选器窗格中基于 <code>Gender</code> 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 '|');
```

### 另请参阅：

- [Aggr - 图表函数\(第 142 页\)](#)
- [Avg - 图表函数\(第 193 页\)](#)
- [RangeCorrel\(第 527 页\)](#)

### 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分位数(位数)对应的值。

#### Syntax:

```
Fractile(expr, fraction)
```

**Return data type:** 数字

#### 参数:

参数	说明
expr	表达式或字段包含要度量的数据。
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:

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>Table1: crosstable LOAD recno() as ID, * inline [ Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2 ] (delimiter is ' ');  Fractile1: LOAD Type, Fractile(Value,0.75) as MyFractile Resident Table1 Group By Type;</pre>	<p>在包含维度 Type 和 MyFractile 的表格中，在数据加载脚本中计算的 Fractile() 结果为：</p> <pre>Type MyFractile Comparison 27.5 Observation 36</pre>

## 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分位数(位数)对应的值。

### Syntax:

```
Fractile([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr,
fraction)
```

**Return data type:** 数字

### 参数:

参数	说明
expr	表达式或字段包含要度量的数据。
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。

参数	说明
DISTINCT	如果在函数参数前出现单词 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

### 限制：

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

### 示例和结果：

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

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 * INLINE [
Month, 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 (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 '|');
```

要按正确顺序对月份进行排序，在创建可视化后，转到属性面板的**Sorting**部分，选择**Month**，然后勾选复选框**Sort by expression**。在表达式框中，输入 Monthnumber。

另请参阅：

▢ [Aggr - 图表函数\(第 142 页\)](#)

### 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。

**Syntax:**

```
Kurtosis([distinct ] expr )
```

**Return data type:** 数字

**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 <b>distinct</b> ，则将忽略所有重复值。

**示例和结果：**

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>Table1: crosstable LOAD recno() as ID, * inline [ Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2 ] (delimiter is ' ');  Kurtosis1: LOAD Type, Kurtosis(Value) as MyKurtosis1, Kurtosis(DISTINCT Value) as MyKurtosis2 Resident Table1 Group By Type;</pre>	<p>在包含维度 Type、MyKurtosis1 和 MyKurtosis2 的表格中，在数据加载脚本中计算的 Kurtosis() 结果为：</p> <pre>Type MyKurtosis1 MyKurtosis2 Comparison -1.1612957 -1.4982366 Observation -1.1148768 -0.93540144</pre>

## 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。

### Syntax:

```
Kurtosis ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

**Return data type:** 数字

### 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
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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

**示例和结果：**

Type	Value																		
Comparison	2	27	38	31	1	19	1	34	3	1	2	3	2	1	2	1	3	29	37
Observation	35	40	12	15	21	14	46	10	28	48	16	30	32	48	31	22	12	39	195

示例	结果
Kurtosis (Value)	对于包含维度 type 和度量 Kurtosis(Value) 的表格，如果显示表格的合计，并且数字格式设置为 3 个有效数字，则结果为 1.252。对于 comparison，结果为 1.161，对于 observation，结果为 1.115。
Kurtosis (TOTAL Value)	对于 type 的全部值，结果为 1.252，因为 TOTAL 限定符意味着忽略维度。

**示例中所使用的数据：**

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
```

```

28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');

```

另请参阅：

▢ [Avg - 图表函数\(第 193 页\)](#)

## 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。

**Syntax:**

```
LINEST_B (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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)	<p>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。</p> <p>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。</p>

**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。

另请参阅：

▢ [如何使用 linest 函数的示例\(第 236 页\)](#)

## 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。

## Syntax:

```
LINEST_B([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

**Return data type:** 数字

## 参数:

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0_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。  <div>  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。 </div>
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 <b>DISTINCT</b> , 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 <b>TOTAL</b> , 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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

## 限制:

表达式不能包含聚合函数, 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, 请结合计算维度使用 **Aggr** 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。

## 另请参阅:

□ 如何使用 *linest* 函数的示例(第 236 页)

▢ [Avg - 图表函数\(第 193 页\)](#)

## 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。

**Syntax:**

```
LINEST_DF (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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。

**另请参阅:**

▢ [如何使用 linest 函数的示例\(第 236 页\)](#)

## 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。

**Syntax:**

```
LINEST_DF([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

**Return data type:** 数字

**参数:**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	<p>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。</p> <div>  <p>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。</p> </div>
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 <b>DISTINCT</b> , 则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>, 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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制:**

表达式不能包含聚合函数, 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, 请结合计算维度使用 **Aggr** 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。

**另请参阅:**

- 如何使用 *linest* 函数的示例(第 236 页)
- *Avg* - 图表函数(第 193 页)

**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。

**Syntax:**

```
LINEST_F (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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)	<p>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。</p> <p>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。</p>

**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。

**另请参阅:**

□ [如何使用 linest 函数的示例\(第 236 页\)](#)

**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 ( $r^2/(1-r^2)$ ), 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。

**Syntax:**

```
LINEST_F([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

**Return data type:** 数字

**参数:**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	<p>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。</p> <div>  <p>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。</p> </div>
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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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。

**另请参阅：**

▢ [如何使用 \*linest\* 函数的示例\(第 236 页\)](#)

▢ [Avg - 图表函数\(第 193 页\)](#)

**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。

**Syntax:**

```
LINEST_M (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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)	<p>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。</p> <p>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。</p>

**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。

**另请参阅：**

□ 如何使用 *linest* 函数的示例(第 236 页)

**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。

**Syntax:**

```
LINEST_M([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value [, y0_const [, x0_const]])
```

**Return data type:** 数字

**参数：**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。  <div>  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。 </div>
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 <b>DISTINCT</b> ，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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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。

另请参阅：

□ [如何使用 \*linest\* 函数的示例\(第 236 页\)](#)

□ [Avg - 图表函数\(第 193 页\)](#)

### 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。

**Syntax:**

```
LINEST_R2 (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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。

另请参阅：

□ [如何使用 \*linest\* 函数的示例\(第 236 页\)](#)

## 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  $r^2$  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。

## Syntax:

```
LINEST_R2([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

**Return data type:** 数字

## 参数:

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0, 可以强制回归线通过单一固定坐标。  <div>  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。 </div>
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 <b>DISTINCT</b> , 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 <b>TOTAL</b> , 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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

## 限制:

表达式不能包含聚合函数, 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, 请结合计算维度使用 **Aggr** 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。

## 另请参阅:

□ 如何使用 *linest* 函数的示例(第 236 页)

▢ *Avg - 图表函数* (第 193 页)

## 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。

**Syntax:**

```
LINEST_SEB (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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。

另请参阅:

▢ *如何使用 linest 函数的示例* (第 236 页)

## 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。

**Syntax:**

```
LINEST_SEB ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_  
value[, y0_const[, x0_const]])
```

**Return data type:** 数字

**参数:**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	<p>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。</p> <div>  <p>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。</p> </div>
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 <b>DISTINCT</b> , 则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>, 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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制:**

表达式不能包含聚合函数, 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, 请结合计算维度使用 **Aggr** 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。

**另请参阅:**

- [如何使用 \*linest\* 函数的示例\(第 236 页\)](#)
- [Avg - 图表函数\(第 193 页\)](#)

**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。

**Syntax:**

```
LINEST_SEM (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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)	<p>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。</p> <p>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。</p>

**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。

**另请参阅:**

□ [如何使用 `linest` 函数的示例\(第 236 页\)](#)

**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。

**Syntax:**

```
LINEST_SEM([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

**Return data type:** 数字

**参数:**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	<p>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。</p> <div>  <p>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。</p> </div>

参数	说明
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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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。

**另请参阅：**

- ▢ 如何使用 *linest* 函数的示例(第 236 页)
- ▢ Avg - 图表函数(第 193 页)

**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。

**Syntax:**

```
LINEST_SEY (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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。



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。

**另请参阅:**

□ 如何使用 *linest* 函数的示例(第 236 页)

**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。

**Syntax:**

```
LINEST_SEY([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])
```

**Return data type:** 数字

**参数:**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0, 可以强制回归线通过单一固定坐标。  <div>  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。 </div>
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 <b>DISTINCT</b> , 则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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。

**另请参阅：**

□ [如何使用 \*linest\* 函数的示例\(第 236 页\)](#)

□ [Avg - 图表函数\(第 193 页\)](#)

**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。

**Syntax:**

```
LINEST_SSREG (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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)	<p>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。</p> <p>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。</p>

**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。

另请参阅：

□ 如何使用 *linest* 函数的示例(第 236 页)

## 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。

**Syntax:**

```
LINEST_SSREG([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value,
x_value[, y0_const[, x0_const]])
```

**Return data type:** 数字

**参数：**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。  <div>  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。 </div>
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 <b>DISTINCT</b> ，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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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。

另请参阅：

▢ [如何使用 `linest` 函数的示例](#)( 第 236 页)

▢ [Avg - 图表函数](#)( 第 193 页)

## 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。

**Syntax:**

```
LINEST_SSRESID (y-value, x-value[, y0 [, x0 ]])
```

**Return data type:** 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。

另请参阅：

▢ [如何使用 `linest` 函数的示例](#)( 第 236 页)

## 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。

**Syntax:**

```
LINEST_SSRESID([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value,
x_value[, y0_const[, x0_const]])
```

**Return data type:** 数字

**参数:**

参数	说明
y_value	表达式或字段要度量的 y 值的范围。
x_value	表达式或字段要度量的 x 值的范围。
y0、x0	可以声明可选值 y0 强制回归线在某一指定点通过 y 轴。通过同时声明 y0 和 x0，可以强制回归线通过单一固定坐标。  <div>  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。 </div>
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 <b>DISTINCT</b> ，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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

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:**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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。

**另请参阅:**

- 如何使用 *linest* 函数的示例(第 236 页)
- Avg - 图表函数(第 193 页)

## 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。

**Syntax:**

**Median (expr)****Return data type:** 数字**参数:**

参数	说明
expr	表达式或字段包含要度量的数据。

**示例和结果:**

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>Table1: crosstable LOAD recno() as ID, * inline [ Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2 ] (delimiter is ' ');  Median1: LOAD Type, Median(Value) as MyMedian Resident Table1 Group By Type;</pre>	<p>在包含维度 Type 和 MyMedian 的表格中，在数据加载脚本中计算的 Median() 结果为：</p> <pre>Type MyMedian Comparison 2.5 Observation 26.5</pre>

**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。

**Syntax:**

**Median**([SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)

**Return data type:** 数字

**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 <b>DISTINCT</b> ，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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**限制:**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

**示例和结果:**

Type	Value																		
Comparison	2	27	38	31	1	19	1	34	3	1	2	3	2	1	2	1	3	29	37
Observation	35	40	12	15	21	14	46	10	28	48	16	30	32	48	31	22	12	39	195

示例	结果
Median (Value)	对于包含维度 Type 和度量 Median(Value) 的表格，如果显示合计，则结果为 19，对于 Comparison，结果为 2.5，对于 Observation，结果为 26.5。
Median (TOTAL Value))	对于 Type 的全部值，结果为 19，因为 TOTAL 限定符意味着忽略维度。

**示例中所使用的数据:**

Table1:  
 crosstable LOAD recno() as ID, \* inline [  
 observation|comparison

```
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

另请参阅：

▢ [Avg - 图表函数\(第 193 页\)](#)

### 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。

**Syntax:**

```
Skew([ distinct] expr)
```

**Return data type:** 数字

**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 <b>distinct</b> , 则将忽略所有重复值。

**示例和结果：**

添加示例脚本到应用程序并运行。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。



示例	结果
<pre>Table1: crosstable LOAD recno() as ID, * inline [ Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2 ] (delimiter is ' ');  Skew1: LOAD Type, Skew(Value) as MySkew Resident Table1 Group By Type;</pre>	<p>在包含维度 Type 和 MySkew 的表格中，在数据加载脚本中计算的 Skew() 结果为：</p> <pre>Type MySkew Comparison 0.86414768 Observation 0.32625351</pre>

## 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。

### Syntax:

```
Skew ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

**Return data type:** 数字

### 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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

### 限制：

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

### 示例和结果：

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1			9		4									9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

示例	结果
Skew (value)	对于包含维度 Type 和度量 Skew(Value) 的表格，如果显示合计，并且数字格式设置为 3 个有效数字，则结果为 0.235。对于 Comparison，结果为 0.864，对于 observation，结果为 0.3265。
Skew (TOTAL Value))	对于 Type 的全部值，结果为 0.235，因为 TOTAL 限定符意味着忽略维度。

### 示例中所使用的数据：

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
```

```
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

---

另请参阅：

□ [Avg - 图表函数\(第 193 页\)](#)

### 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。

**Syntax:**

```
Stdev([distinct] expr)
```

**Return data type:** 数字

**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 <b>distinct</b> ，则将忽略所有重复值。

**示例和结果：**

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>Table1: crosstable LOAD recno() as ID, * inline [ Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2 ] (delimiter is ' ');  Stdev1: LOAD Type, Stdev(Value) as MyStdev Resident Table1 Group By Type;</pre>	<p>在包含维度 Type 和 MyStdev 的表格中，在数据加载脚本中计算的 Stdev() 结果为：</p> <pre>Type MyStdev Comparison 14.61245 Observation 12.50799</pre>

## 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。

### Syntax:

```
Stdev ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

**Return data type:** 数字

### 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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

**示例和结果：**

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1			9		4									9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

示例	结果
Stdev (Value)	对于包含维度 Type 和度量 Stdev(Value) 的表格，如果显示合计，则结果为 15.475，对于 Comparison，结果为 14.612，对于 observation，结果为 12.508。
Stdev (TOTAL Value))	对于 Type 的全部值，结果为 15.475，因为 TOTAL 限定符意味着忽略维度。

**示例中所使用的数据：**

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
```

```
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

另请参阅：

▢ [Avg - 图表函数\(第 193 页\)](#)

▢ [STEXYX - 图表函数\(第 234 页\)](#)

### 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。

**Syntax:**

```
Sterr ([distinct] expr)
```

**Return data type:** 数字

**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 <b>distinct</b> ，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：**

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>Table1: crosstable LOAD recno() as ID, * inline [ Observation Comparison 35 2 40 27 12 38 15 31 21 1 14 19 46 1 10 34 28 3 48 1 16 2 30 3 32 2 48 1 31 2 22 1 12 3 39 29 19 37 25 2 ] (delimiter is ' ');  Sterr1: LOAD Type, Sterr(Value) as MySterr Resident Table1 Group By Type;</pre>	<p>在包含维度 Type 和 MySterr 的表格中，在数据加载脚本中计算的 Sterr() 结果为：</p> <pre>Type MySterr Comparison 3.2674431 Observation 2.7968733</pre>

## 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 ( $\text{stdev}/\sqrt{n}$ )。

### Syntax:

```
Sterr ([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] expr)
```

**Return data type:** 数字

### 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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 函数。

文本值，NULL 值和缺失值都忽略不计。

**示例和结果：**

Type	Value																			
Comparison	2	2	3	3	1	1	1	3	3	1	2	3	2	1	2	1	3	2	3	2
		7	8	1		9		4										9	7	
Observation	35	4	1	1	2	1	4	1	2	4	1	3	3	4	3	2	1	3	1	2
		0	2	5	1	4	6	0	8	8	6	0	2	8	1	2	2	9	9	5

示例	结果
Sterr (Value)	对于包含维度 type 和度量 sterr(Value) 的表格，如果显示合计，则结果为 2.447，对于 comparison，结果为 3.267，对于 observation，结果为 2.797。
Sterr (TOTAL Value))	对于 type 的全部值，结果为 2.447，因为 TOTAL 限定符意味着忽略维度。

**示例中所使用的数据：**

```
Table1:
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
```



```

32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');

```

另请参阅：

▢ [Avg - 图表函数\(第 193 页\)](#)

▢ [STEYX - 图表函数\(第 234 页\)](#)

## 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** 子句定义的一系列记录迭代的 x-expression 和 y-expression 的成对数值呈现的一系列坐标。

**Syntax:**

```
STEYX (y-value, x-value)
```

**Return data type:** 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。

**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：**

添加示例脚本到应用程序并运行。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。

示例	结果
<pre>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;</pre>	<p>在包含维度 <code>MySTEYX</code> 的表格中，在数据加载脚本中计算的 <code>STEYX()</code> 结果为 2.0714764。</p>

## 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。

### Syntax:

```
STEYX([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value)
```

**Return data type:** 数字

### 参数:

参数	说明
y_value	表达式或字段，其中包含要度量的已知 y 值范围。
x_value	表达式或字段，其中包含要度量的已知 x 值范围。
SetExpression	聚合函数会默认聚合选择项定义的可能记录集合。可选记录集合可由集合分析表达式定义。
DISTINCT	如果在函数参数前出现单词 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。

参数	说明
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

**限制：**

表达式不能包含聚合函数，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，请结合计算维度使用 **Aggr** 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：****Data series**

KnownX	17	16	14	11	10	8	7	6	5	5	5	4
KnownY	15	14	12	9	9	10	6	2	3	5	8	7

示例	结果
Steyx(KnownY, KnownX)	2.071( 如果数字格式设置为 3 个小数位。)
Steyx(TOTAL KnownY, KnownX)	<p>在全部维度中，结果为 2.071( 如果未做出任何选择)。</p> <p>在全部维度中，结果为 2.121( 例如，如果为 KnownX 选择了选择项 4、5 和 6)。</p>

**示例中所使用的数据：**

```
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 '|');
```

另请参阅：

▢ *Avg* - 图表函数(第 193 页)

▢ *Sterr* - 图表函数(第 231 页)

### 如何使用 linest 函数的示例

linest 函数用于查找与线性回归分析相关的值。本节介绍如何通过使用样本数据查找 Qlik Sense 中可用的 linest 函数值来创建可视化内容。linest 函数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。


### 加载样本数据

执行以下操作：



1. 创建新应用程序。
2. 在数据加载编辑器中，输入以下内容：

```
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;
```

- 单击  以加载数据。

### 显示数据加载脚本计算的结果

- 执行以下操作：  
在数据加载编辑器中，单击  以转到应用视图，创建新表格，然后打开此表格。
- 单击  编辑以编辑表格。
- 在 **Charts** 中添加表格，然后在 **Fields** 中添加以下列：
  - Linest\_B
  - Linest\_DF
  - Linest\_F
  - Linest\_M
  - Linest\_R2
  - Linest\_SEB
  - Linest\_SEM
  - Linest\_SEY
  - Linest\_SSREG
  - Linest\_SSRESID

表格包含在数据加载脚本中执行 linest 计算的结果，如下所示：



Linest_B	Linest_DF	Linest_F	Linest_M	Linest_R2	Linest_SEB
-35.047	18	20.788	8.605	0.536	22.607

Linest_SEM	Linest_SEY	Linest_SSREG	Linest_SSRESID
1.887	48.666	49235.014	42631.186

### 创建 linest 图表函数可视化内容

执行以下操作：

- 在数据加载编辑器中，单击  以转到应用视图，创建新表格，然后打开此表格。
- 单击  编辑以编辑表格。
- 在 **图表** 中添加折线图，在 **字段** 中添加 X 作为维度，添加 Sum(Y) 作为度量。  
创建线图来表示根据 Y 绘制的 X 图形，在其中可以计算 linest 函数。
- 在 **Charts** 中添加使用以下表达式作为维度的表格：  
 valueList('Linest\_b', 'Linest\_df', 'Linest\_f', 'Linest\_m', 'Linest\_r2', 'Linest\_SEB', 'Linest\_SEM', 'Linest\_SEY', 'Linest\_SSREG', 'Linest\_SSRESID')  
 这样可以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。

5. 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))
```

这样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

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

## 统计检验函数

本节介绍用于统计检验的函数，其中分为三类函数。这些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。该函数可用于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。该函数可用于将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。

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])
```

另请参阅：

▢ [T 检验函数\(第 242 页\)](#)

▢ [Z 检验函数\(第 272 页\)](#)

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。

**Syntax:**

```
Chi2Test_chi2(col, row, actual_value[, expected_value])
```

**Return data type:** 数字

**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 )
```

**另请参阅：**

- [如何在图表中使用 chi2-test 函数的示例\( 第 285 页\)](#)
- [如何在数据加载脚本中使用 chi2-test 函数的示例\( 第 287 页\)](#)

**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。

**Syntax:**

```
Chi2Test_df(col, row, actual_value[, expected_value])
```

**Return data type:** 数字

**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 )
```

另请参阅：

- [如何在图表中使用 \*chi2-test\* 函数的示例\(第 285 页\)](#)
- [如何在数据加载脚本中使用 \*chi2-test\* 函数的示例\(第 287 页\)](#)

### 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。

Syntax:

```
Chi2Test_p(col, row, actual_value[, expected_value])
```

Return data type: 数字

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 )
```

另请参阅：

- [如何在图表中使用 `chi2-test` 函数的示例\(第 285 页\)](#)
- [如何在数据加载脚本中使用 `chi2-test` 函数的示例\(第 287 页\)](#)

### 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。

另请： [创建典型的 t-test 报告\(第 289 页\)](#)

#### 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 ( 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 (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 (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 (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 (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 (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 (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 (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 (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 (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 ( 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 (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 ( 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 (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 (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 (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 (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 (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 (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 (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 (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 (value)
```

ttest1\_t

**TTest1\_t()** 用于返回值系列的聚合 T 值。

```
TTest1_t (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 (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 **numeric** 函数，用于返回值系列的聚合置信区间值。

```
TTest1w_conf (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 (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 (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 (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 (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 (weight, value)
```

ttest1w\_t

**TTest1w\_t()** 用于返回值系列的聚合 T 值。

```
TTest1w_t ( 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 (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。

**Syntax:**

```
TTest_conf ( grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数:**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅：**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTest_df (grp, value [, eq_var])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。

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 <b>Type</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅：**

□ [创建典型的 t-test 报告\( 第 289 页\)](#)

**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。

**Syntax:**

```
TTest_dif (grp, value [, eq_var] )
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅：**

▢ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTest_lower (grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

另请参阅：

□ [创建典型的 t-test 报告\( 第 289 页\)](#)

### 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。

**Syntax:**

```
TTest_sig (grp, value [, eq_var])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

另请参阅：

□ [创建典型的 t-test 报告\( 第 289 页\)](#)

## 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。

### Syntax:

```
TTest_sterr (grp, value [, eq_var])
```

**Return data type:** 数字

### 参数:

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

### 另请参阅:

▢ [创建典型的 t-test 报告\(第 289 页\)](#)

## 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。



为能使用此函数，您必须使用 **crosstable** 在脚本中加载样本值。

#### Syntax:

```
TTest_t(grp, value[, eq_var])
```

**Return data type:** 数字

#### 参数：

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

#### 另请参阅：

□ [创建典型的 t-test 报告\(第 289 页\)](#)

#### 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。

#### Syntax:

```
TTest_upper (grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅：**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTestw_conf (weight, grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅：**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTestw_df (weight, grp, value [, eq_var])
```

**Return data type:** 数字

**参数：**

参数	说明
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0), 则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅:**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTestw_dif (weight, group, value)
```

**Return data type:** 数字

**参数:**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。

参数	说明
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。

**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 )
```

**另请参阅:**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTestw_lower (weight, grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数:**

参数	说明
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。

参数	说明
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略, 应将 <b>sig</b> 设置为 0.025, 对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0), 则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅:**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTestw_sig ( weight, grp, value [, eq_var])
```

**Return data type:** 数字

**参数:**

参数	说明
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0), 则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅：**

□ [创建典型的 t-test 报告\( 第 289 页\)](#)

**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。

**Syntax:**

```
TTestw_sterr (weight, grp, value [, eq_var])
```

**Return data type:** 数字

**参数：**

参数	说明
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

另请参阅：

▢ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
ttestw_t (weight, grp, value [, eq_var])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

另请参阅：

▢ [创建典型的 t-test 报告\(第 289 页\)](#)

## 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。

### Syntax:

```
TTestw_upper (weight, grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

### 参数:

参数	说明
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 <b>Type</b> 。
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

### 另请参阅:

□ [创建典型的 t-test 报告\(第 289 页\)](#)

## 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。

### Syntax:

```
TTest1_conf (value [, sig ])
```

**Return data type:** 数字

### 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 <b>Value</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 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 )
```

### 另请参阅:

□ [创建典型的 t-test 报告\(第 289 页\)](#)

### 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。

### Syntax:

```
TTest1_df (value)
```

**Return data type:** 数字

### 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 <b>Value</b> 。

**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 )
```

**另请参阅：**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTest1_dif (value)
```

**Return data type:** 数字

**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 <b>Value</b> 。

**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 )
```

另请参阅：

□ [创建典型的 t-test 报告\(第 289 页\)](#)

### 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。

#### Syntax:

```
TTest1_lower (value [, sig])
```

**Return data type:** 数字

**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 <b>Value</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 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 )
```

另请参阅：

□ [创建典型的 t-test 报告\(第 289 页\)](#)

### 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。

### Syntax:

```
TTest1_sig (value)
```

**Return data type:** 数字

### 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 <b>Value</b> 。

### 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 )
```

### 另请参阅:

□ [创建典型的 t-test 报告\(第 289 页\)](#)

### 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。

### Syntax:

```
TTest1_sterr (value)
```

**Return data type:** 数字

### 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 <b>Value</b> 。

### 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 )
```

### 另请参阅：

□ [创建典型的 t-test 报告\( 第 289 页\)](#)

### 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。

### Syntax:

```
TTest1_t (value)
```

**Return data type:** 数字

### 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 <b>Value</b> 。

### 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 )
```

### 另请参阅：

□ [创建典型的 t-test 报告\( 第 289 页\)](#)



## 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。

### Syntax:

```
TTest1_upper (value [, sig])
```

**Return data type:** 数字

### 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 <b>Value</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 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 )
```

### 另请参阅:

□ [创建典型的 t-test 报告\(第 289 页\)](#)

## TTest1w\_conf

**TTest1w\_conf()** 是一个 **numeric** 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。

### Syntax:

```
TTest1w_conf (weight, value [, sig ])
```

**Return data type:** 数字

**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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略, 应将 <b>sig</b> 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 )
```

**另请参阅:**

□ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTest1w_df (weight, value)
```

**Return data type:** 数字

**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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 )
```

**另请参阅：**

▢ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTest1w_dif (weight, value)
```

**Return data type:** 数字

**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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 )`

另请参阅：

▢ [创建典型的 t-test 报告\( 第 289 页\)](#)

`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。

**Syntax:**

**TTest1w\_lower** (weight, value [, sig ])

**Return data type:** 数字

**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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 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 )
```

另请参阅：

▢ [创建典型的 t-test 报告\( 第 289 页\)](#)

`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。

### Syntax:

```
TTest1w_sig (weight, value)
```

**Return data type:** 数字

### 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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 )
```

### 另请参阅:

□ [创建典型的 t-test 报告\(第 289 页\)](#)

### 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。

### Syntax:

```
TTest1w_sterr (weight, value)
```

**Return data type:** 数字

### 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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 )
```

**另请参阅：**

▢ [创建典型的 t-test 报告\(第 289 页\)](#)

**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。

**Syntax:**

```
TTest1w_t ( weight, value)
```

**Return data type:** 数字

**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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 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 )`

另请参阅：

□ [创建典型的 t-test 报告\( 第 289 页\)](#)

`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。

**Syntax:**

`TTest1w_upper (weight, value [, sig])`

**Return data type:** 数字

**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 <b>Value</b> 。
weight	<b>value</b> 中的每个值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 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 )`

另请参阅：

□ [创建典型的 t-test 报告\( 第 289 页\)](#)

## 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。

另请：如何使用 z-test 函数的示例(第 292 页)

### 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 (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 (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 (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 (value [, sigma])
```

ztest\_z

**ZTest\_z()** 用于返回值系列的聚合 Z 值。

```
ZTest_z (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 (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 (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 (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 (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 (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 (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 (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 (weight, value [, sigma])
```

ztestw\_z

**ZTestw\_z()** 用于返回值系列的聚合 Z 值。

```
ZTestw_z (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。

### Syntax:

```
ZTest_z (value[, sigma])
```

**Return data type:** 数字

**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 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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 )
```

**另请参阅：**

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

**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。

**Syntax:**

```
ZTest_sig(value[, sigma])
```

**Return data type:** 数字

**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 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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)
```

### 另请参阅：

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

### 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。

### Syntax:

```
ZTest_dif(value[, sigma])
```

**Return data type:** 数字

### 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 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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)
```

### 另请参阅：

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

### 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。

### Syntax:

```
ZTest_sterr(value[, sigma])
```

**Return data type:** 数字

### 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 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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)
```

### 另请参阅:

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

### 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。

### Syntax:

```
ZTest_conf(value[, sigma[, sig]])
```

**Return data type:** 数字

### 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 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，则会使用实际样本标准偏差。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 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)
```

**另请参阅：**

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

**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。

**Syntax:**

```
ZTest_lower (grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

**另请参阅：**

□ [如何使用 z-test 函数的示例\( 第 292 页\)](#)

**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。

**Syntax:**

```
ZTest_upper (grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

另请参阅：

□ [如何使用 z-test 函数的示例\( 第 292 页\)](#)

### 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。

**Syntax:**

```
ZTestw_z (weight, value [, sigma])
```

**Return data type:** 数字

**参数：**

参数	说明
value	值应通过 <b>value</b> 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	<b>value</b> 中的每个样本值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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)
```

另请参阅：

□ [如何使用 z-test 函数的示例\( 第 292 页\)](#)

### 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。

### Syntax:

```
ZTestw_sig (weight, value [, sigma])
```

**Return data type:** 数字

### 参数:

参数	说明
value	值应通过 <b>value</b> 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	<b>value</b> 中的每个样本值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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)
```

### 另请参阅:

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

### 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。

### Syntax:

```
ZTestw_dif ( weight, value [, sigma])
```

**Return data type:** 数字



**参数：**

参数	说明
value	值应通过 <b>value</b> 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	<b>value</b> 中的每个样本值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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)
```

**另请参阅：**

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

**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。

**Syntax:**

```
ZTestw_sterr (weight, value [, sigma])
```

**Return data type:** 数字

**参数：**

参数	说明
value	值应通过 <b>value</b> 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	<b>value</b> 中的每个样本值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。

参数	说明
sigma	如果已知，则可以在 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，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)
```

**另请参阅：**

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

**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。

**Syntax:**

```
ZTest_conf(weight, value[, sigma[, sig]])
```

**Return data type:** 数字

**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	<b>value</b> 中的每个样本值都可以根据 <b>weight</b> 中的相应加权值计数一次或多次。
sigma	如果已知，则可以在 <b>sigma</b> 中声明标准偏差。如果省略 <b>sigma</b> ，则会使用实际样本标准偏差。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 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)
```

**另请参阅：**

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

**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。

**Syntax:**

```
ZTestw_lower (grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

另请参阅：

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

## 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。

### Syntax:

```
ZTestw_upper (grp, value [, sig [, eq_var]])
```

**Return data type:** 数字

**参数：**

参数	说明
value	可以计算样本值。样本值必须按照在 <b>group</b> 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 <b>Value</b> 。
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 <b>Type</b> 。
sig	可以在 <b>sig</b> 中指定双尾级显著性。如果省略，应将 <b>sig</b> 设置为 0.025，对应于 95% 置信区间。
eq_var	如果指定 <b>eq_var</b> 为 False (0)，则可以假定两个样本的单独方差。如果指定 <b>eq_var</b> 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 )
```

另请参阅：

□ [如何使用 z-test 函数的示例\(第 292 页\)](#)

## 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，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  以加载数据。

### 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。随即打开表格视图。
2. 单击  编辑以编辑表格。
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：

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：

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：

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。

	A	B	C	D	E	F
I	15	7	9	20	26	19
II	10	11	7	15	21	16


加载样本数据

执行以下操作：

1. 创建新应用程序。

2. 在数据加载编辑器中，输入以下内容：

```
// Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the top
of the script.
Sample_1:
LOAD * inline [
Grp,Grade,Count
I,A,15
I,B,7
I,C,9
I,D,20
I,E,26
I,F,19
II,A,10
II,B,11
II,C,7
II,D,15
II,E,21
II,F,16
];
```

3. 单击  以加载数据。

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  以加载数据。

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：

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 t-test 的 Qlik Sense Comparison 函数来创建这些表格。这些样本的相应表格如下所示：

#### Group Statistics

Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

#### Independent Sample Test

	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)
Equal Variance not Assumed	3.534	37.116717335823	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	3.534	38	0.001	15.2	4.30101	6.49306	23.9069

### 加载样本数据

执行以下操作：


1. 创建具有新表格的新应用，并打开该表格。
2. 在数据加载编辑器中输入以下内容：  
 Table1:  
 crosstable LOAD recno() as ID, \* inline [  
 Observation|Comparison  
 35|2  
 40|27  
 12|38  
 15|31  
 21|1  
 14|19  
 46|1  
 10|34  
 28|3

```

48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');



```

在此加载脚本中包括 **recno()**，因为 **crosstable** 需要三个参数。因此，**recno()** 仅提供额外参数，在这种情况下提供每一行的 ID。如果不使用此函数，则不会加载 **Comparison** 样本值。

3. 单击  以加载数据。

### 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。这样将打开表格视图。
2. 单击  编辑以编辑表格。
3. 在图表中添加表格，在字段中添加以下表达式作为度量：

标签	表达式
N	Count(Value)
Mean	Avg(Value)
Standard Deviation	Stdev(Value)
Standard Error Mean	Sterr(Value)

4. 在表格中添加 Type 作为维度。
5. 单击 **排序**，并将 Type 移到排序列表顶部。


结果：

这些样本的 Group Statistics 表格如下所示：

Type	N	Mean	Standard Deviation	Standard Error Mean
Comparison	20	11.95	14.61245	3.2674431
Observation	20	27.15	12.507997	2.7968933

### 创建 Two Independent Sample Student's T-test 表格

执行以下操作：

1. 单击  编辑以编辑表格。
2. 在表格中添加以下表达式作为维度。=valueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1))

3. 在图表中添加使用以下表达式作为度量的表格：

标签	表达式
conf	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_conf(Type, Value),TTest_conf(Type, Value, 0))
t	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_t(Type, Value),TTest_t(Type, Value, 0))
df	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_df(Type, Value),TTest_df(Type, Value, 0))
Sig. (2-tailed)	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_sig(Type, Value),TTest_sig(Type, Value, 0))
Mean Difference	TTest_dif(Type, Value)
Standard Error Difference	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_sterr(Type, Value),TTest_sterr(Type, Value, 0))
95% Confidence Interval of the Difference (Lower)	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_lower(Type, Value,(1-(95)/100)/2),TTest_lower(Type, Value,(1-(95)/100)/2, 0))
95% Confidence Interval of the Difference (Upper)	if(ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1)),TTest_upper(Type, Value,(1-(95)/100)/2),TTest_upper (Type, Value,(1-(95)/100)/2, 0))

结果：

这些样本的 **Independent Sample Test** 表格如下所示：

	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval of the Difference (Lower)	95% Confidence Interval of the Difference (Upper)
Equal Variance not Assumed	3.534	37.1167173358	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	3.534	38	0.001	15.2	4.30101	6.49306	23.9069

### 如何使用 z-test 函数的示例

z-test 函数用于查找与大量数据样本的 z-test 统计分析相关的值，通常大于 30，其中方差为已知。本节介绍如何通过使用样本数据查找 Qlik Sense 中可用的 z-test 函数值来创建可视化内容。有关语法和参数说明，请参阅单独的 z-test 图表函数主题。

### 加载样本数据

此处使用的样本数据与 t-test 函数示例中所使用的样本数据相同。对于 Z 检验分析，样本数据大小通常被视为过小，但足以用于说明如何在 Qlik Sense 中使用不同的 z-test 函数。

执行以下操作：

1. 创建具有新表格的新应用，并打开该表格。



如果为 t-test 函数创建了应用，则可以使用该应用，并为这些函数创建新表格。

2. 在数据加载编辑器中，输入以下内容：

Table1:

```
crosstable LOAD recno() as ID, * inline [
Observation|Comparison
```

```
35|2
```

```
40|27
```

```
12|38
```

```
15|31
```

```
21|1
```

```
14|19
```

```
46|1
```

```
10|34
```

```
28|3
```

```
48|1
```

```
16|2
```

```
30|3
```

```
32|2
```

```
48|1
```

```
31|2
```

```
22|1
```


```
12|3
```

```
39|29
```

```
19|37
```



```
25|2 ] (delimiter is '|');
```

在此加载脚本中包括 **recno()**，因为 **crosstable** 需要三个参数。因此，**recno()** 仅提供额外参数，在这种情况下提供每一行的 ID。如果不使用此函数，则不会加载 **Comparison** 样本值。

3. 单击  以加载数据。

### 创建 z-test 图表函数可视化内容

执行以下操作：

1. 在数据加载编辑器中，单击  以转到应用视图，然后单击您在加载数据时创建的表格。随即打开表格视图。
2. 单击  编辑以编辑表格。

- 在图表中添加表格，在**字段**中添加 Type 作为维度。
- 在表格中添加以下表达式作为度量。

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。但是对于 ZTest Sig，例如，使用数字格式：**自定义**，然后将各样式调整为**###**。

结果：

样本数据的 z-test 函数结果表格中将包含以下值：

Type	ZTest Conf	ZTest Dif	ZTest Sig	ZTest Sterr	ZTest Z
Comparison	6.40	11.95	0.000123	3.27	3.66
Value	5.48	27.15	0.001	2.80	9.71

#### 创建 z-testw 图表函数可视化内容

z-testw 函数在输入数据系列采用加权两列格式时使用。表达式需要使用参数 weight 的值。此处的示例始终使用值 2，但您可以使用表达式，用于定义每个观测项的 weight 值。

示例和结果：

z-test 函数使用相同的样本数据和数字格式，z-testw 函数的结果表格将包含以下值：

Type	ZTestw Conf	ZTestw Dif	ZTestw Sig	ZTestw Sterr	ZTestw Z
Comparison	3.53	2.95	5.27e-005	1.80	3.88
Value	2.97	34.25	0	4.52	20.49

## 字符串聚合函数

本节介绍字符串相关的聚合函数。

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

### 数据加载脚本中的字符串聚合函数

#### Concat

**Concat()** 用于组合字符串值。此脚本函数用于返回迭代于 **group by** 子句定义的许多记录的所有表达式值的聚合字符串串联。

```
Concat ([ distinct ] expression [, delimiter [, sort-weight]])
```

#### FirstValue

**FirstValue()** 用于返回首先从表达式定义的记录加载, 然后通过 **group by** 子句排序的值。



此函数仅可用作脚本函数。

```
FirstValue (expression)
```

#### LastValue

**LastValue()** 用于返回最后从表达式定义的记录加载, 然后通过 **group by** 子句排序的值。



此函数仅可用作脚本函数。

```
LastValue (expression)
```

#### MaxString

**MaxString()** 用于查找表达式中的字符串值, 并返回通过 **group by** 子句定义的大量记录存储的最后一个文本值。

```
MaxString (expression )
```

#### MinString

**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存储的第一个文本值。

```
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, 并以文本排序顺序返回最后一个文本值。

**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，并以文本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。

**Syntax:**

**Concat** ([ distinct ] string [, delimiter [, sort-weight]])

**Return data type:** 字符串

**参数:**

表达式或字段，其中包含要处理的字符串。

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 <b>sort-weight</b> 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 <b>distinct</b> ，则将忽略所有重复值。

**示例和结果:**

添加示例脚本到应用程序并运行。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。

示例	结果	
<b>TeamData:</b> 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 ' ');  <b>Concat1:</b> LOAD SalesGroup,Concat(Team) as TeamConcat1 Resident TeamData Group By SalesGroup;	SalesGroup	TeamConcat1
	East	AlphaBetaDeltaGammaGamma
	West	EpsilonEtaThetaZeta
前提是 <b>TeamData</b> 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	TeamConcat2
	East	Alpha-Beta-Delta-Gamma
	West	Epsilon-Eta-Theta-Zeta
前提是 <b>TeamData</b> 表格像之前的示例一样加载:  LOAD SalesGroup,Concat(distinct Team,'-',Amount) as TeamConcat2 Resident TeamData Group By SalesGroup;	因为已经为 <b>sort-weight</b> 添加参数, 因此将会按维度 Amount 值对结果进行排序。	
	SalesGroup	TeamConcat2
	East	Delta-Beta-Gamma-Alpha
	West	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。

### Syntax:

```
Concat({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] string[,  
delimiter[, sort_weight]])
```

**Return data type:** 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。



参数	说明
sort-weight	串联的顺序可由维度 <b>sort-weight</b> 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 <b>DISTINCT</b> ，则将忽略计算该函数参数生成的副本。
TOTAL	<p>如果在函数参数前面出现单词 <b>TOTAL</b>，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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

#### 示例和结果：

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

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。

示例	结果
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 <b>sort-weight</b> 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。

### Syntax:

**FirstValue** ( expr )

**Return data type:** 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：**

添加示例脚本到应用程序并运行。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。

示例	结果	
<pre>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;</pre>	SalesGroup	FirstTeamLoaded
	East	Gamma
	West	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。

**Syntax:**

```
LastValue ( expr )
```

**Return data type:** 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：**

添加示例脚本到应用程序并运行。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	
<pre>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;</pre>	SalesGroup	LastTeamLoaded
	East	Beta
	West	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存储的最后一个文本值。

#### Syntax:

```
MaxString ( expr )
```

**Return data type:** 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：

添加示例脚本到应用程序并运行。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。

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	MaxString1
	East	Gamma
	West	Zeta
前提是 <b>TeamData</b> 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	MaxString2
	East	01/11/2013
	West	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，并以文本排序顺序返回最后一个文本值。

### Syntax:

```
MaxString({[SetExpression] [TOTAL [<fld{, fld}>]]} expr)
```

**Return data type:** 双

**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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**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：

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



此表显示了具有相应 *Product* 值的维度 *Customer* 的所有值。在表格的实际表格可视化中，每行都有一个 *Customer* 和 *Product* 值。

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

**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存储的第一个文本值。

**Syntax:****MinString** ( expr )**Return data type:** 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:**

添加示例脚本到应用程序并运行。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。

示例	结果	
<b>TeamData:</b> 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 ' ');  <b>Concat1:</b> LOAD SalesGroup,MinString(Team) as MinString1 Resident TeamData Group By SalesGroup;	SalesGroup	MinString1
	East	Alpha
	West	Epsilon
前提是 <b>TeamData</b> 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	MinString2
	East	01/05/2013
	West	01062/2013

**MinString - 图表函数**

**MinString()** 用于查找表达式或字段中的字符串值，并以文本排序顺序返回第一个文本值。

**Syntax:**

```
MinString([SetExpression] [TOTAL [<fld {, fld}>]]) expr)
```

**Return data type:** 双**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 <b>TOTAL</b> ，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。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

**示例和结果:**

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

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。

### Syntax:

**ValueList** (v1 {, ...})

**Return data type:** 双

**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：**

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。																																				
=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: <div><table><tr><th colspan="4">ValueList()</th></tr><tr><th>Created dimension</th><th>Year</th><th colspan="2">Added expression</th></tr><tr><td></td><td></td><td></td><td>522.00</td></tr><tr><td>Number of Orders</td><td>2012</td><td></td><td>5.00</td></tr><tr><td>Number of Orders</td><td>2013</td><td></td><td>7.00</td></tr><tr><td>Average Order Size</td><td>2012</td><td></td><td>13.20</td></tr><tr><td>Average Order Size</td><td>2013</td><td></td><td>15.43</td></tr><tr><td>Total Amount</td><td>2012</td><td></td><td>66.00</td></tr><tr><td>Total Amount</td><td>2013</td><td></td><td>108.00</td></tr></table></div>	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
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 [
SalesID|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。

### Syntax:

```
ValueLoop (from [, to [, step ]])
```

Return data type: 双

### 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:

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。

一般规则是不允许在 Qlik Sense 图表表达式中嵌套聚合函数。仅在以下情况下才允许嵌套：

- 在内部聚合函数中使用 **TOTAL** 限定符。



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))
```

包括 **TOTAL** 限定符对于 Qlik Sense 接受此类嵌套绝对必要，同时对于所需比较也极为必要。此类嵌套需求极为常用，是很好的做法。

### 另请参阅：

▢ [Aggr - 图表函数 \(第 142 页\)](#)

## 5.2 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 **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 255 之间，用于定义颜色。

```
HSL (hue, saturation, luminosity)
```

### RGB

**RGB()** 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** 定义颜色。

```
RGB (r, g, b)
```

### Color

在表达式中使用 **Color()** 会返回相关图表调色板内  $n$  颜色值的颜色呈现形式。颜色呈现形式是对偶值，而文本呈现形式显示为 'RGB(r, g, b)' 形式，其中  $r$ 、 $g$  和  $b$  均为 0 和 255 之间的值，分别代表红色、绿色和蓝色值。数字呈现形式为整数，代表红色、绿色和蓝色分量。

**Color** ( $n$ )

### 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 RGB 颜色呈现形式。

**Colormix1** ( $\text{Value}$  ,  $\text{ColorZero}$  ,  $\text{ColorOne}$ )

$\text{Value}$  为 0 和 1 之间的真实数字。

- 如果  $\text{Value} = 0$ ，则会返回第一种颜色。
- 如果  $\text{Value} = 1$ ，则会返回第二种颜色。
- 如果  $0 < \text{Value} < 1$ ，则会返回适当的中间值底纹。

$\text{ColorZero}$  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。

$\text{ColorOne}$  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。

### 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 RGB 颜色呈现形式，同时指定中心 (0) 位置的中间颜色。

**Colormix2** ( $\text{Value}$  ,  $\text{ColorMinusOne}$  ,  $\text{ColorOne}$  [ ,  $\text{ColorZero}$  ])

$\text{Value}$  为 -1 和 1 之间的真实数字。

- 如果  $\text{Value} = -1$ ，则会返回第一种颜色。
- 如果  $\text{Value} = 1$ ，则会返回第二种颜色。
- 如果  $-1 < \text{Value} < 1$ ，则会返回适当的中间值底纹。

$\text{ColorMinusOne}$  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。

$\text{ColorOne}$  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。

$\text{ColorZero}$  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 RGB 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()** 会返回颜色表的颜色的呈现形式，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()** 会返回颜色表的颜色的呈现形式，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  $\alpha$  因子参数，在这种情况下，将会返回 ARGB 颜色呈现形式。 $\alpha$  因子为 0 表示完全透明， $\alpha$  因子为 255 表示完全不透明。

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	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。

**Syntax:**

```
ARGB(alpha, r, g, b)
```

**Return data type:** 双

**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** 因子。后面两位“00”表示 **red** 的数量、接下来两位“FF”表示 **green** 的数量，以及最后两位“00”表示 **blue** 的数量。

## RGB

**RGB()** 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** 定义颜色。

**Syntax:**

```
RGB (r, g, b)
```

**Return data type:** 双

**参数:**

参数	说明
r, g, b	红色，绿色和蓝色成分的值。颜色成分 0 对应无影响，其中一个 255 对应完全影响。



所有参数均必须为表达式，用于解算范围介于 0 至 255 之间的整数。

如果解释数值成分并以十六进制表示法格式化数值，则颜色成分的值比较明显。例如，浅绿色的编号为 4 278 255 360，其十六进制表示法为 FF00FF00。前两位“FF”(255) 表示 **alpha** 因子。在函数 **RGB** 和 **HSL** 中，这两位始终为“FF”(不透明)。后面两位“00”表示 **red** 的数量、接下来两位“FF”表示 **green** 的数量，以及最后两位“00”表示 **blue** 的数量。

### 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 255 之间，用于定义颜色。

**Syntax:**

```
HSL (hue, saturation, luminosity)
```

**Return data type:** 双

**参数:**

参数	说明
hue, saturation, luminosity	hue、saturation 和 luminosity 组件值。值 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 4 286 080 100，其十六进制表示法为 FF786464。前两位“FF”(255) 表示 **alpha** 因子。在函数 **RGB** 和 **HSL** 中，这两位始终为“FF”(不透明)。随后两位“78”表示 **hue** 组件，随后两位“64”表示 **saturation**，最后两位“64”表示 **luminosity** 组件。

## 5.3 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 (case1[ , case2 , case3 , ...] , 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  $a \leq x < b$  的双值作为文本值，其中  $a$  和  $b$  为  $\text{bin}$  的上限值和下限值，且下界为数值。

```
class (expression, interval [ , label [ , offset ]])
```

### 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量。比较区分大小写。

```
match ( str, expr1 [ , expr2,...exprN ])
```

### mixmatch

**mixmatch** 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数量。比较不区分大小写。

```
mixmatch ( str, expr1 [ , expr2,...exprN ])
```

### pick

**pick** 函数用于返回列表中的第  $n$  个表达式。

```
pick (n, expr1[ , expr2,...exprN])
```

### wildmatch

**wildmatch** 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数量。此函数允许在比较字符串中使用通配符( `*` 和 `?` )。比较不区分大小写。

```
wildmatch ( str, expr1 [ , expr2,...exprN ])
```

## alt

**alt** 函数用于返回首个具有有效数表示法的参数。如果未找到此类匹配，则将返回最后一个参数。可使用任何数目的参数。

### Syntax:

```
alt(case1[ , case2 , case3 , ...] , else)
```

**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。

示例和结果：

示例	结果
<pre>alt( date#( dat , 'YYYY/MM/DD' ),       date#( dat , 'MM/DD/YYYY' ),       date#( dat , 'MM/DD/YY' ),       'No valid date' )</pre>	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对偶值。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)。
<pre>alt(Sales,0) + alt(Margin,0)</pre>	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  $a \leq x < b$  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。

### Syntax:

```
class(expression, interval [ , label [ , offset ]])
```

### 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。

### 示例和结果:

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'

### 示例数据加载脚本:

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。源表格如下所示：

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。在本例中，我们使用内联数据加载源表格。

```
LOAD *,
class(Age, 10, 'age') As Agegroup;
```

```
LOAD * INLINE
[ Age, Name
25, John
42, Karen
53, Yoshi];
```

加载的结果数据如下所示：

Name	Age	Agegroup
John	25	20 <= age < 30
Karen	42	40 <= age < 50
Yoshi	53	50 <= age < 60

### 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。

#### Syntax:

```
if(condition , then , else)
```

if 函数有三个参数：*condition*、*then* 和 *else*，都是表达式。其他两个(*then* 和 *else*)可为任何类型。

#### 参数：

参数	说明
condition	进行逻辑解释的表达式。
then	可为任何类型的表达式。如果 <i>condition</i> 是 True，则 if 函数返回 <i>then</i> 表达式的值。
else	可为任何类型的表达式。如果 <i>condition</i> 是 False，则 if 函数返回 <i>else</i> 表达式的值。

#### 示例和结果：

示例	结果
if( Amount>= 0, 'OK', 'Alarm' )	此表达式将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'。

## 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量。比较区分大小写。

### Syntax:

```
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。

### 示例和结果：

示例	结果
match( M, 'Jan','Feb','Mar')	返回 2, 如果 M = Feb。 返回0, 如果 M = Apr 或 jan。

## mixmatch

**mixmatch** 函数用于将第一个参数与所有以下参数进行比较，并返回匹配表达式的数量。比较不区分大小写。

### Syntax:

```
mixmatch( str, expr1 [ , expr2,...exprN ])
```



如果您要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。

### 示例和结果：

示例	结果
mixmatch( M, 'Jan','Feb','Mar')	返回 1, 如果 M = jan

## pick

**pick** 函数用于返回列表中的第 *n* 个表达式。

### Syntax:

```
pick(n, expr1[ , expr2,...exprN])
```

### 参数：

参数	说明
n	n 是介于 1 和 N 之间的整数。

示例和结果：

示例	结果
<code>pick( N, 'A','B',4, 6 )</code>	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。此函数允许在比较字符串中使用通配符( \* 和 ?)。比较不区分大小写。

**Syntax:**

```
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。

示例和结果：

示例	结果
<code>wildmatch( M, 'ja*','fe?','mar')</code>	返回 1, 如果 M = January 返回 2, 如果 M = february

## 5.4 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。

### 计算函数概述

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。



此函数仅可用作脚本函数。

```
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。

### Syntax:

```
autonumber (expression[ , AutoID])
```

参数：

参数	说明
AutoID	为创建多个计数实例，如果 <b>autonumber</b> 函数用于脚本内不同的字段，则可选参数 <i>AutoID</i> 将用于命名每个计数。

#### 示例：创建合成键段

在此例中，我们使用 **autonumber** 函数创建合成键段以节省内存。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益。

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 字段创建合成键段。

```
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：

Region	Year	Month	Sales	RYMkey
North	2014	May	245	1

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

在此例中，如果您需要链接到其他表格，可以引用 RYMkey(例如 1)，而不是字符串“North2014May”。

现在，我们可以通过类似方式加载成本的源表格。在前置 Load 中已排除 Region、Year 和 Month 字段以避免创建合成关键字段，因为我们已经使用 **autonumber** 函数创建复合关键字段用于链接表格。

```
RegionCosts:
LOAD Costs,
AutoNumber(Region&Year&Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013, May, 167
North, 2014, May, 56
North, 2014, June, 199
South, 2014, June, 64
South, 2013, May, 172
South, 2013, May, 126
];
```

现在，我们可以将表格可视化添加到表格，并添加 Region、Year 和 Month 字段，以及销售额和成本的 Sum 度量。表格如下所示：

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals			1952	784
North	2014	June	127	199
North	2014	May	592	56
South	2014	June	645	64
South	2013	May	588	465

另请参阅：

▢ [autonumberhash128](#) (第 321 页)

▢ [autonumberhash256](#) (第 323 页)



## 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。

### Syntax:

```
autonumberhash128 (expression {, expression})
```

### 示例：创建合成键段

在此例中，我们使用 **autonumberhash128** 函数创建合成键段以节省内存。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。

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 字段创建合成键段。

```
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：

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

在此例中，如果您需要链接到其他表格，可以引用 RYMkey( 例如 1)，而不是字符串“North2014May”。

现在，我们可以通过类似方式加载成本的源表格。在前置 Load 中已排除 Region、Year 和 Month 字段以避免创建合成关键字段，因为我们已经使用 **autonumberhash128** 函数创建复合关键字段用于链接表格。

```
RegionCosts:
LOAD Costs,
AutoNumberHash128(Region, Year, Month) as RYMkey;
```

```
LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013, May, 167
North, 2014, May, 56
North, 2014, June, 199
South, 2014, June, 64
South, 2013, May, 172
South, 2013, May, 126
];
```

现在，我们可以将表格可视化添加到表格，并添加 Region、Year 和 Month 字段，以及销售额和成本的 Sum 度量。表格如下所示：

Region	Year	Month	Sum([Sales])	Sum([Costs])
Totals			1952	784
North	2014	June	127	199
North	2014	May	592	56
South	2014	June	645	64
South	2013	May	588	465

另请参阅：

▢ `autonumberhash256` (第 323 页)

▢ `autonumber` (第 318 页)

## 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。

### Syntax:

```
autonumberhash256(expression {, expression})
```

### 示例：创建合成键段

在此例中，我们使用 `autonumberhash256` 函数创建合成键段以节省内存。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。

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 字段创建合成键段。

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：

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

在此例中，如果您需要链接到其他表格，可以引用 RYMkey(例如 1)，而不是字符串“North2014May”。

现在，我们可以通过类似方式加载成本的源表格。在前置 Load 中已排除 Region、Year 和 Month 字段以避免创建合成关键字段，因为我们已经使用 **autonumberhash256** 函数创建复合关键字段用于链接表格。

```
RegionCosts:
LOAD Costs,
AutoNumberHash256(Region, Year, Month) as RYMkey;

LOAD * INLINE
[ Region, Year, Month, Costs
South, 2013, May, 167
North, 2014, May, 56
North, 2014, June, 199
South, 2014, June, 64
South, 2013, May, 172
South, 2013, May, 126
];
```

现在，我们可以将表格可视化添加到表格，并添加 Region、Year 和 Month 字段，以及销售额和成本的 Sum 度量。表格如下所示：

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

另请参阅：

▢ [autonumberhash128](#) (第 321 页)

▢ [autonumber](#) (第 318 页)

## 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。

**Syntax:**

**IterNo** ( )

示例和结果：

示例	结果												
<pre>LOAD     IterNo() as Day,     Date( StartDate + IterNo() - 1 ) as Date     while StartDate + IterNo() - 1 &lt;= EndDate;  LOAD * INLINE [StartDate, EndDate 2014-01-22, 2014-01-26 ];</pre>	<p>该 <b>LOAD</b> 语句将为 <b>StartDate</b> 和 <b>EndDate</b> 定义的范围之间的每个日期生成一条记录。</p> <p>在此例中，最终生成的表格如下所示：</p> <table> <tr> <th>Day</th><th>Date</th></tr> <tr> <td>1</td><td>2014-01-22</td></tr> <tr> <td>2</td><td>2014-01-23</td></tr> <tr> <td>3</td><td>2014-01-24</td></tr> <tr> <td>4</td><td>2014-01-25</td></tr> <tr> <td>5</td><td>2014-01-26</td></tr> </table>	Day	Date	1	2014-01-22	2	2014-01-23	3	2014-01-24	4	2014-01-25	5	2014-01-26
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。

**Syntax:**

**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：

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。

### Syntax:

```
RowNo ( [TOTAL] )
```

与对原始数据表格中记录进行计数的 **RecNo()** 相反，**RowNo()** 函数不会对 **where** 子句排除的记录进行计数，并且在原始数据表格串联到其他表格时，该函数不会重置。



如果使用前置 Load，即从同一表格读取的叠加的 **LOAD** 语句数，则只能在顶部的 **RowNo()** 语句中使用 **LOAD**。如果在后续的 **LOAD** 语句中使用 **RowNo()**，则将返回 0。

### 示例：数据加载脚本

原始数据表格加载：

```
Tab1:
LOAD * INLINE
[A, B
1, aa
2, cc
3, ee];
```

```
Tab2:
LOAD * INLINE
[C, D
5, xx
4, yy
6, zz];
```

加载选定行的记录数和行数：

```
QTab:
LOAD *,
RecNo( ),
RowNo( )
resident Tab1 where A<>2;
```

```
LOAD
C as A,
D as B,
RecNo( ),
RowNo( )
resident Tab2 where A<>5;
```

```
//we don't need the source tables anymore, so we drop them
Drop tables Tab1, Tab2;
```

所生成的 Qlik Sense 内部表格：

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。

### Syntax:

**RowNo** ( [TOTAL] )

**Return data type:** 整数

### 参数:

参数	说明
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数，则当前列段数据总是与整列相等。  <b>TOTAL</b> 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 <fld>。这些字段名应该是图表维度变量的子集。

### 示例和结果:

Customer	UnitSales	Row in Segment	Row Number
Astrida	4	1	1
Astrida	10	2	2
Astrida	9	3	3
Betacab	5	1	4
Betacab	2	2	5
Betacab	25	3	6



Customer	UnitSales	Row in Segment	Row Number
Canutility	8	1	7
Canutility		2	8
Divadip	4	1	9
Divadip		2	10

示例	结果
使用维度 <b>Customer</b> 和 <b>UnitSales</b> 创建包含表格的可视化，并添加 ROWNO( ) 和 ROWNO(TOTAL) 作为标记为 <b>Row in Segment</b> 和 <b>Row Number</b> 的度量。	<p><b>Row in Segment</b> 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。</p> <p><b>Row Number</b> 列会忽略可用于对表格中的行进行计数的维度。</p>
添加表达式： IF( ROWNO( )=1, 0, UnitSales / Above( UnitSales ) ) 作为度量。	<p>此表达式会为每个列段数据中的第一行返回 0，因此列将会显示：</p> <p>0、2.25、1.1111111、0、2.5、5、0、2.375、0 和 4。</p>

#### 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 '|');
```

#### 另请参阅：

□ [Above - 图表函数\(第 483 页\)](#)

## 5.5 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。要解决这个问题,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)。

### 日期和时间函数概述

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

#### 时间的整数表达式

##### second

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示秒的整数。

```
second (expression)
```

##### minute

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示分钟的整数。

```
minute (expression)
```

##### hour

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示小时的整数。

```
hour (expression)
```

##### day

此函数用于根据标准数字解释当 **expression** 小数部分被解释为日期时返回一个表示某天的整数。

```
day (expression)
```

##### week

此函数用于返回根据 ISO 8601 表示周数的整数。周数根据标准数字解释通过表达式的日期解释进行计算。

```
week (expression)
```

### month

此函数用于返回包含在环境变量 **MonthNames** 中定义的月份名称的对偶值和一个介于 1 至 12 的整数。月份根据标准数字解释通过表达式的日期解释进行计算。

```
month (expression)
```

### year

此函数用于根据标准数字解释当 **expression** 被解释为日期时返回一个表示年份的整数。

```
year (expression)
```

### weekyear

此函数用于返回根据 ISO 8601 周数所属的年份。星期数范围在 1 和大约 52 之间。

```
weekyear (expression)
```

### weekday

此函数用于返回包含以下名称的对偶值：在环境变量 **DayNames** 中定义的日期名称。介于 0-6 之间的整数对应于一周 (0-6) 的标定天。

```
weekday (date)
```

## Timestamp 函数

### now

此函数用于返回系统时钟的当前时间的时间戳。

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

### today

此函数用于返回系统时钟的当前日期。

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

### LocalTime

此函数用于返回指定时区的系统时钟的当前时间戳。

```
localtime ([timezone [, ignoreDST ]])
```

## Make 函数

### makedate

此函数用于返回根据年份 **YYYY**、月份 **MM** 和日期 **DD** 计算的日期。

```
makedate (YYYY [ , MM [ , DD ] ])
```

### makeweekdate

此函数用于返回根据年份 **YYYY**、星期 **WW** 和星期几 **D** 计算的日期。

```
makeweekdate (YYYY [ , WW [ , D ] ])
```

### maketime

此函数用于返回根据小时 **hh**、分钟 **mm** 和秒 **ss** 计算的时间。

```
maketime (hh [ , mm [ , ss [ .fff ] ] ])
```

### 其他日期函数

#### AddMonths

此函数用于返回在 **startdate** 后 **n** 个月内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 个月内发生的日期。

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

#### AddYears

此函数用于返回在 **startdate** 后 **n** 年内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 年内发生的日期。

```
addyears (startdate, n)
```

#### yeartodate

此函数用于判断输入日期是否在最后加载脚本的日期的年份以内，并返回 True(如果在)或返回 False(如果不在)。

```
yeartodate (date [ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

### Timezone 函数

#### timezone

此函数用于返回当前时区的名称，如 Windows 所定义。

```
timezone ( )
```

#### GMT

此函数用于返回来自系统时钟的当前 Greenwich Mean Time 和 Windows 时间设置。

```
GMT ( )
```

#### UTC

用于返回当前 Coordinated Universal Time。

```
UTC ( )
```

#### daylightsaving

用于返回如 Windows 所定义的当下为日间省时的调整。

```
daylightsaving ( )
```

#### converttolocaltime

将 UTC 或 GMT 时间戳转换为本地时间作为对偶值。此地方可为全世界任何一个城市，地方和时区。

```
converttolocaltime (timestamp [ , place [ , ignore_dst=false])
```

### 设置时间函数

#### setdateyear

此函数用于返回基于输入 **timestamp** 但将年份替换为 **year** 的时间戳。

```
setdateyear (timestamp, year)
```

#### setdateyearmonth

用于返回基于输入 **timestamp**, 但将年份替换为 **year** 和将月份替换为 **month** 的时间戳

```
setdateyearmonth (timestamp, year, month)
```

### In... 函数

#### inyear

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 的年份以内。

```
inyear (date, basedate , shift [, first_month_of_year = 1])
```

#### inyeartodate

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 为止以及包括 **base\_date** 最后毫秒的年份部分以内。

```
inyeartodate (date, basedate , shift [, first_month_of_year = 1])
```

#### inquarter

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 的季度以内。

```
inquarter (date, basedate , shift [, first_month_of_year = 1])
```

#### inquartertodate

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 为止以及包括 **base\_date** 最后毫秒的季度部分以内。

```
inquartertodate (date, basedate , shift [, first_month_of_year = 1])
```

#### inmonth

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 的月份以内。

```
inmonth (date, basedate , shift)
```

#### inmonthtodate

用于返回 True, 如果 **date** 位于包含 **basedate** 为止以及包括 **basedate** 最后毫秒的月份部分以内。

```
inmonthtodate (date, basedate , shift)
```

#### inmonths

此函数用于判断时间戳是否位于作为基准日期的同一个月、两个月、季度、四个月或半年以内。另外, 也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

```
inmonths (n, date, basedate , shift [, first_month_of_year = 1])
```

### **inmonthstodate**

此函数用于判断时间戳是否位于截止以及包括 **base\_date** 的最后毫秒的某个月、两个月、季度、四个月或半年周期的一部分以内。另外，它也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

```
inmonthstodate (n, date, basedate , shift [, first_month_of_year = 1])
```

### **inweek**

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 的星期以内。

```
inweek (date, basedate , shift [, weekstart])
```

### **inweektodate**

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 为止以及包括 **base\_date** 最后毫秒的星期部分以内。

```
inweektodate (date, basedate , shift [, weekstart])
```

### **inlunarweek**

此函数用于判断 **timestamp** 是否位于包含 **base\_date** 的阴历周以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
inlunarweek (date, basedate , shift [, weekstart])
```

### **inlunarweektodate**

此函数用于判断 **timestamp** 是否位于截止以及包括 **base\_date** 最后毫秒的阴历周的某部分以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
inlunarweektodate (date, basedate , shift [, weekstart])
```

### **inday**

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_timestamp** 的一天以内。

```
inday (timestamp, basetimestamp , shift [, daystart])
```

### **indaytotime**

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_timestamp** 为止以及包括 **base\_timestamp** 精确毫秒的日子部分以内。

```
indaytotime (timestamp, basetimestamp , shift [, daystart])
```

## Start ... end 函数

### **yearstart**

此函数用于返回与包含 **date** 的年份的第一天的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**。

```
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**

---

## 5 脚本和图表表达式中的函数

---

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

```
monthsstart (n, date [, shift = 0 [, first_month_of_year = 1]])
```

### monthsend

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的最后毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

```
monthsend (n, date [, shift = 0 [, first_month_of_year = 1]])
```

### monthsname

此函数用于返回一个显示值，表示时段各月份(根据 **MonthNames** 脚本变量的格式)和年的范围。基础数值与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应。

```
monthsname (n, date [, shift = 0 [, first_month_of_year = 1]])
```

### weekstart

此函数用于返回与包含 **date** 的日历周的第一天(周一)的第一毫秒时间戳对应的值。默认输出格式是在脚本中设置的 **DateFormat**。

```
weekstart (date [, shift = 0 [, weekoffset = 0]])
```

### weekend

此函数用于返回一个与包含 **date** 的日历周的最后一日(周日)最后一毫秒时间戳对应的值。默认输出格式为在脚本中设置的 **DateFormat**。

```
weekend (date [, shift = 0 [, weekoffset = 0]])
```

### weekname

此函数用于返回一个值，显示带有与包含 **date** 的周的第一天的第一毫秒时间戳对应的基本数值对应的年份和周数。

```
weekname (date [, shift = 0 [, weekoffset = 0]])
```

### lunarweekstart

此函数用于返回与包含 **date** 的阴历周的第一毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
lunarweekstart (date [, shift = 0 [, weekoffset = 0]])
```

### lunarweekend

此函数用于返回与包含 **date** 的阴历周的最后毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
lunarweekend (date [, shift = 0 [, weekoffset = 0]])
```

### lunarweekname

此函数用于返回一个显示值，显示与包含 **date** 的阴历周的第一天的第一毫秒时间戳对应的年份和阴



历周数。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

```
lunarweekname (date [, shift = 0 [, weekoffset = 0]])
```

### **daystart**

此函数用于返回与 **time** 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**。

```
daystart (timestamp [, shift = 0 [, dayoffset = 0]])
```

### **dayend**

此函数用于返回与 **time** 中包含的一天的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **TimestampFormat**。

```
dayend (timestamp [, shift = 0 [, dayoffset = 0]])
```

### **dayname**

此函数用于返回一个值，显示与包含 **time** 当天第一毫秒的时间戳对应的基本数值的日期。

```
dayname (timestamp [, shift = 0 [, dayoffset = 0]])
```

## Day numbering 函数

### **age**

**age** 函数用于返回某人在 **date\_of\_birth** 出生的 **timestamp**(完整年份)时的年龄。

```
age (timestamp, date_of_birth)
```

### **networkdays**

**networkdays** 函数用于返回工作日的编号(周一至周五)，在 **start\_date** 和 **end\_date** 之间，并将任何列出的可选 **holiday** 考虑在内。

```
networkdays (start:date, end_date {, holiday})
```

### **firstworkdate**

**firstworkdate** 函数用于返回最近的起始日以获得 **no\_of\_workdays**(周一至周五)，将任何列出的可选节假日考虑在内，不迟于 **end\_date**。**end\_date** 和 **holiday** 应为有效的日期或时间戳。

```
firstworkdate (end_date, no_of_workdays {, holiday} )
```

### **lastworkdate**

**lastworkdate** 函数用于返回最早的结束日以获得 **no\_of\_workdays**(周一至周五)，如果在 **start\_date** 开始考虑任何列出的可选 **holiday**。**start\_date** 和 **holiday** 应是有效的日期或时间戳。

```
lastworkdate (start_date, no_of_workdays {, holiday})
```

### **daynumberofyear**

此函数用于计算时间戳所属的年份的天数。从该年度的第一天的第一毫秒开始计算，但可以偏移第一个月。

```
daynumberofyear (date[,firstmonth])
```

### daynumberofquarter

此函数用于计算时间戳所属的季度的天数。

```
daynumberofquarter (date[,firstmonth])
```

## addmonths

此函数用于返回在 **startdate** 后 **n** 个月内发生的日期，或者如果 **n** 为负数，则用于返回 **startdate** 前 **n** 个月内发生的日期。

### Syntax:

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

**Return data type:** 双

**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	<b>mode</b> 指定是相对每月的开头还是相对每月的结尾添加月份。如果输入日期是 28 日或以上，且 <b>mode</b> 设置为 1，该函数将会返回一个距输入日期的月份结尾相同天数的日期。默认模式为 0。

**示例和结果：**

示例	结果
addmonths ('2003-01-29',3)	返回“2003-04-29”
addmonths ('2003-01-29',3,0)	返回“2003-04-29”
addmonths ('2003-01-29',3,1)	返回“2003-04-28”
addmonths ('2003-01-29',1,0)	返回“2003-02-28”
addmonths ('2003-01-29',1,1)	返回“2003-02-26”
addmonths ('2003-02-28',1,0)	返回“2003-03-28”
addmonths ('2003-02-28',1,1)	返回“2003-03-31”

## 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。

### Syntax:

**AddYears** (startdate, n)

**Return data type:** 双

**参数:**

参数	说明
startdate	作为时间戳的开始日期，例如“2012-10-12”。
n	作为正整数或负整数的年份数量。

**示例和结果:**

示例	结果
addyears ('2010-01-29', 3)	返回“2013-01-29”
addyears ('2010-01-29', -1)	返回“2009-01-29”

## age

**age** 函数用于返回某人在 **date\_of\_birth** 出生的 **timestamp**(完整年份)时的年龄。

**Syntax:**

**age** (timestamp, date\_of\_birth)

可以是表达式。

**Return data type:** 数字

**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。

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	结果																																							
<p>添加示例脚本到应用程序并运行。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。</p> <pre>Employess: 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  ); DoBTable: Load *, age(LocalTime(),DateOfBirth) As Age Resident Employees; Drop table Employees;</pre>	<p>结果列表显示了为表格中的每条记录返回的 age 值。</p> <table><tr><th>Member</th><th>DateOfBirth</th><th>Age</th></tr><tr><td>John</td><td>28/03/1989</td><td>25</td></tr><tr><td>Linda</td><td>10/12/1990</td><td>23</td></tr><tr><td>Steve</td><td>5/2/1992</td><td>22</td></tr><tr><td>Birg</td><td>31/3/1993</td><td>21</td></tr><tr><td>Raj</td><td>19/5/1994</td><td>20</td></tr><tr><td>Prita</td><td>15/9/1994</td><td>20</td></tr><tr><td>Su</td><td>11/12/1994</td><td>19</td></tr><tr><td>Goran</td><td>2/3/1995</td><td>19</td></tr><tr><td>Sunny</td><td>14/5/1996</td><td>18</td></tr><tr><td>Ajoa</td><td>13/6/1996</td><td>18</td></tr><tr><td>Daphne</td><td>7/7/1998</td><td>16</td></tr><tr><td>Biffy</td><td>4/8/2000</td><td>14</td></tr></table>	Member	DateOfBirth	Age	John	28/03/1989	25	Linda	10/12/1990	23	Steve	5/2/1992	22	Birg	31/3/1993	21	Raj	19/5/1994	20	Prita	15/9/1994	20	Su	11/12/1994	19	Goran	2/3/1995	19	Sunny	14/5/1996	18	Ajoa	13/6/1996	18	Daphne	7/7/1998	16	Biffy	4/8/2000	14
Member	DateOfBirth	Age																																						
John	28/03/1989	25																																						
Linda	10/12/1990	23																																						
Steve	5/2/1992	22																																						
Birg	31/3/1993	21																																						
Raj	19/5/1994	20																																						
Prita	15/9/1994	20																																						
Su	11/12/1994	19																																						
Goran	2/3/1995	19																																						
Sunny	14/5/1996	18																																						
Ajoa	13/6/1996	18																																						
Daphne	7/7/1998	16																																						
Biffy	4/8/2000	14																																						

## 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。

### Syntax:

```
ConvertToLocalTime(timestamp [, place [, ignore_dst=false]])
```

**Return data type:** 双

### 参数:

参数	说明
timestamp	时间戳或对时间戳求值的表达式都可用于转换。

参数	说明
<b>place</b>	<p>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：</p> <ul style="list-style-type: none"> <li>• GMT</li> <li>• GMT-12:00 - GMT-01:00</li> <li>• GMT+01:00 - GMT+14:00</li> <li>• UTC</li> <li>• UTC-12:00 - UTC-01:00</li> <li>• UTC+01:00 - UTC+14:00</li> </ul> <div>  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。 </div>
<b>ignore_dst</b>	如果要忽略 DST(夏令时)，则设置为 True。

结果时间调整为夏令时，除非将 **ignore\_dst** 设置为 True。

#### 有效的地方和时区

Abu Dhabi	Central America	Kabul	Newfoundland	Tashkent
Adelaide	Central Time (US & Canada)	Kamchatka	Novosibirsk	Tbilisi
Alaska	Chennai	Karachi	Nuku'alofa	Tehran
Amsterdam	Chihuahua	Kathmandu	Osaka	Tokyo
Arizona	Chongqing	Kolkata	Pacific Time (US & Canada)	Urumqi
Astana	Copenhagen	Krasnoyarsk	Paris	Warsaw
Athens	Darwin	Kuala Lumpur	Perth	Wellington
Atlantic Time (Canada)	Dhaka	Kuwait	Port Moresby	West Central Africa
Auckland	Eastern Time (US & Canada)	Kyiv	Prague	Vienna
Azores	Edinburgh	La Paz	Pretoria	Vilnius
Baghdad	Ekaterinburg	Lima	Quito	Vladivostok

## 有效的地方和时区

Baku	Fiji	Lisbon	Riga	Volgograd
Bangkok	Georgetown	Ljubljana	Riyadh	Yakutsk
Beijing	Greenland	London	Rome	Yerevan
Belgrade	Greenwich Mean Time : Dublin	Madrid	Samoa	Zagreb
Berlin	Guadalajara	Magadan	Santiago	
Bern	Guam	Mazatlan	Sapporo	
Bogota	Hanoi	Melbourne	Sarajevo	
Brasilia	Harare	Mexico City	Saskatchewan	
Bratislava	Hawaii	Mid-Atlantic	Seoul	
Brisbane	Helsinki	Minsk	Singapore	
Brussels	Hobart	Monrovia	Skopje	
Bucharest	Hong Kong	Monterrey	Sofia	
Budapest	Indiana (East)	Moscow	Solomon Is.	
Buenos Aires	International Date Line West	Mountain Time (US & Canada)	Sri Jayawardenepura	
Cairo	Irkutsk	Mumbai	St. Petersburg	
Canberra	Islamabad	Muscat	Stockholm	
Cape Verde Is.	Istanbul	Nairobi	Sydney	
Caracas	Jakarta	New Caledonia	Taipei	
Casablanca	Jerusalem	New Delhi	Tallinn	

## 示例和结果：

示例	结果
<code>convertToLocalTime('2007-11-10 23:59:00','Paris')</code>	返回“2007-11-11 00:59:00”以及相应的内部时间戳表示。
<code>convertToLocalTime(UTC(), 'GMT-05:00')</code>	返回北美东海岸的时间，例如纽约。
<code>convertToLocalTime(UTC(), 'GMT-05:00', True)</code>	返回北美东海岸的时间，例如纽约，而不调整日间节省时间。

## day

此函数用于根据标准数字解释当 **expression** 小数部分被解释为日期时返回一个表示某天的整数。

### Syntax:

```
day(expression)
```

**Return data type:** 整数

示例和结果：

示例	结果
day( '1971-10-12' )	返回 12
day( '35648' )	返回 6, 因为 35648 = 1997-08-06

## dayend

此函数用于返回与 **time** 中包含的一天的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **TimestampFormat**。

### Syntax:

```
DayEnd(time[, [period_no[, day_start]])
```

**Return data type:** 双

参数：

参数	说明
<b>time</b>	要求值的时间戳。
<b>period_no</b>	<b>period_no</b> 为整数，或解算为整数的表达式，其中值 0 表示该天包含 <b>time</b> 。 <b>period_no</b> 为负数表示前几天，正数表示随后的几天。
<b>day_start</b>	要指定不想从某一日的午夜开始工作，可指定一个偏移作为某日内时间的小数 <b>day_start</b> 。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。

示例	结果																										
<code>dayend('25/01/2013 16:45')</code>	返回 25/01/2013 23:59:59。																										
<code>dayend('25/01/2013 16:45', -1)</code>	返回 24/01/2013 23:59:59。																										
<code>dayend('25/01/2013 16:45', 0, 0.5)</code>	返回 26/01/2013 11:59:59。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找标记每个发票日期之后当天结束的时间戳。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, DayEnd(InvDate, 1) AS DEnd Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>dayend()</code> 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。</p> <table> <thead> <tr> <th>InvDate</th><th>DEnd</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>29/03/2012 23:59:59</td></tr> <tr><td>10/12/2012</td><td>11/12/2012 23:59:59</td></tr> <tr><td>5/2/2013</td><td>07/02/2013 23:59:59</td></tr> <tr><td>31/3/2013</td><td>01/04/2013 23:59:59</td></tr> <tr><td>19/5/2013</td><td>20/05/2013 23:59:59</td></tr> <tr><td>15/9/2013</td><td>16/09/2013 23:59:59</td></tr> <tr><td>11/12/2013</td><td>12/12/2013 23:59:59</td></tr> <tr><td>2/3/2014</td><td>03/03/2014 23:59:59</td></tr> <tr><td>14/5/2014</td><td>15/05/2014 23:59:59</td></tr> <tr><td>13/6/2014</td><td>14/06/2014 23:59:59</td></tr> <tr><td>7/7/2014</td><td>08/07/2014 23:59:59</td></tr> <tr><td>4/8/2014</td><td>05/08/2014 23:59:59</td></tr> </tbody> </table>	InvDate	DEnd	28/03/2012	29/03/2012 23:59:59	10/12/2012	11/12/2012 23:59:59	5/2/2013	07/02/2013 23:59:59	31/3/2013	01/04/2013 23:59:59	19/5/2013	20/05/2013 23:59:59	15/9/2013	16/09/2013 23:59:59	11/12/2013	12/12/2013 23:59:59	2/3/2014	03/03/2014 23:59:59	14/5/2014	15/05/2014 23:59:59	13/6/2014	14/06/2014 23:59:59	7/7/2014	08/07/2014 23:59:59	4/8/2014	05/08/2014 23:59:59
InvDate	DEnd																										
28/03/2012	29/03/2012 23:59:59																										
10/12/2012	11/12/2012 23:59:59																										
5/2/2013	07/02/2013 23:59:59																										
31/3/2013	01/04/2013 23:59:59																										
19/5/2013	20/05/2013 23:59:59																										
15/9/2013	16/09/2013 23:59:59																										
11/12/2013	12/12/2013 23:59:59																										
2/3/2014	03/03/2014 23:59:59																										
14/5/2014	15/05/2014 23:59:59																										
13/6/2014	14/06/2014 23:59:59																										
7/7/2014	08/07/2014 23:59:59																										
4/8/2014	05/08/2014 23:59:59																										

## daylightsaving

用于返回如 Windows 所定义的当下为日间省时的调整。

### Syntax:

**DaylightSaving( )**

**Return data type:** 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。



**Syntax:**

```
DayName (time[, period_no [, day_start]])
```

**Return data type:** 双**参数:**

参数	说明
<b>time</b>	要求值的时间戳。
<b>period_no</b>	<b>period_no</b> 为整数, 或解算为整数的表达式, 其中值 0 表示该天包含 <b>time</b> 。 <b>period_no</b> 为负数表示前几天, 正数表示随后的几天。
<b>day_start</b>	要指定不想从某一日的午夜开始工作, 可指定一个偏移作为某日内时间的小数 <b>day_start</b> 。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。

示例	结果
dayname('25/01/2013 16:45')	返回 25/01/2013。
dayname('25/01/2013 16:45', -1)	返回 24/01/2013。
dayname('25/01/2013 16:45', 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	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>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。</p> <pre>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;</pre>	<p>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。</p> <table> <thead> <tr> <th>InvDate</th><th>DName</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>29/03/2012 00:00:00</td></tr> <tr><td>10/12/2012</td><td>11/12/2012 00:00:00</td></tr> <tr><td>5/2/2013</td><td>07/02/2013 00:00:00</td></tr> <tr><td>31/3/2013</td><td>01/04/2013 00:00:00</td></tr> <tr><td>19/5/2013</td><td>20/05/2013 00:00:00</td></tr> <tr><td>15/9/2013</td><td>16/09/2013 00:00:00</td></tr> <tr><td>11/12/2013</td><td>12/12/2013 00:00:00</td></tr> <tr><td>2/3/2014</td><td>03/03/2014 00:00:00</td></tr> <tr><td>14/5/2014</td><td>15/05/2014 00:00:00</td></tr> <tr><td>13/6/2014</td><td>14/06/2014 00:00:00</td></tr> <tr><td>7/7/2014</td><td>08/07/2014 00:00:00</td></tr> <tr><td>4/8/2014</td><td>05/08/2014 00:00:00</td></tr> </tbody> </table>	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
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。

### Syntax:

```
DayNumberOfQuarter(timestamp[,start_tmonth])
```

**Return data type:** 整数

此函数使用的是基于 366 天的年份。

### 参数:

参数	说明
timestamp	要求值的日期。
start_month	通过在 2 和 12 之间(如果省略,则为 1)指定 <b>start_month</b> , 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 <b>start_month = 3</b> 。

示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																					
DayNumberOfQuarter('12/09/2014')	返回 74, 当前季度的天数。																					
DayNumberOfQuarter('12/09/2014',3)	返回 12, 当前季度的天数。 在此例中, 第一个季度从三月份开始( 因为已将 start_month 指定为 3)。这意味着当前季度为第三个季度, 从 9 月 1 日开始。																					
添加示例脚本到应用程序并运行。 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。  ProjectTable: LOAD recno() as InvID, INLINE [ StartDate 28/03/2014 10/12/2014 5/2/2015 31/3/2015 19/5/2015 15/9/2015 ]; NrDays: Load *, DayNumberOfQuarter(StartDate,4) As DayNrQtr Resident ProjectTable; Drop table ProjectTable;	结果列表显示了为表格中的每条记录返回的 DayNumberOfQuarter 值。 <table><tr><th>InvID</th><th>StartDate</th><th>DayNrQtr</th></tr><tr><td>1</td><td>28/03/2014</td><td>88</td></tr><tr><td>2</td><td>10/12/2014</td><td>71</td></tr><tr><td>3</td><td>5/2/2015</td><td>36</td></tr><tr><td>4</td><td>31/3/2015</td><td>91</td></tr><tr><td>5</td><td>19/5/2015</td><td>49</td></tr><tr><td>6</td><td>15/9/2015</td><td>77</td></tr></table>	InvID	StartDate	DayNrQtr	1	28/03/2014	88	2	10/12/2014	71	3	5/2/2015	36	4	31/3/2015	91	5	19/5/2015	49	6	15/9/2015	77
InvID	StartDate	DayNrQtr																				
1	28/03/2014	88																				
2	10/12/2014	71																				
3	5/2/2015	36																				
4	31/3/2015	91																				
5	19/5/2015	49																				
6	15/9/2015	77																				

### daynumberofyear

此函数用于计算时间戳所属的年份的天数。从该年度的第一天的第一毫秒开始计算, 但可以偏移第一个月。

#### Syntax:

```
DayNumberOfYear (timestamp[,start_month])
```

**Return data type:** 整数

此函数使用的是基于 366 天的年份。

#### 参数:

参数	说明
<b>timestamp</b>	要求值的日期。
<b>start_month</b>	通过在 2 和 12 之间(如果省略,则为 1)指定 <b>start_month</b> , 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 <b>start_month = 3</b> 。

示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																					
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。																					
添加示例脚本到应用程序并运行。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。  ProjectTable: LOAD recno() as InvID, INLINE [ StartDate 28/03/2014 10/12/2014 5/2/2015 31/3/2015 19/5/2015 15/9/2015 ]; NrDays: Load *, DayNumberOfYear(StartDate,4) As DayNrYear Resident ProjectTable; Drop table ProjectTable;	结果列表显示了为表格中的每条记录返回的 DayNumberOfYear 值。  <table><tr><th>InvID</th><th>StartDate</th><th>DayNrYear</th></tr><tr><td>1</td><td>28/03/2014</td><td>363</td></tr><tr><td>2</td><td>10/12/2014</td><td>254</td></tr><tr><td>3</td><td>5/2/2015</td><td>311</td></tr><tr><td>4</td><td>31/3/2015</td><td>366</td></tr><tr><td>5</td><td>19/5/2015</td><td>49</td></tr><tr><td>6</td><td>15/9/2015</td><td>168</td></tr></table>	InvID	StartDate	DayNrYear	1	28/03/2014	363	2	10/12/2014	254	3	5/2/2015	311	4	31/3/2015	366	5	19/5/2015	49	6	15/9/2015	168
InvID	StartDate	DayNrYear																				
1	28/03/2014	363																				
2	10/12/2014	254																				
3	5/2/2015	311																				
4	31/3/2015	366																				
5	19/5/2015	49																				
6	15/9/2015	168																				

## daystart

此函数用于返回与 **time** 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**。

**Syntax:**

```
DayStart(time[, [period_no[, day_start]])
```

**Return data type:** 双

**参数:**

参数	说明
<b>timestamp</b>	要求值的时间戳。
<b>period_no</b>	<b>period_no</b> 为整数，或解算为整数的表达式，其中值 0 表示该天包含 <b>time</b> 。 <b>period_no</b> 为负数表示前几天，正数表示随后的几天。
<b>day_start</b>	要指定不想从某一日的午夜开始工作，可指定一个偏移作为某日内时间的小数 <b>day_start</b> 。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。

示例	结果																										
<code>daystart('25/01/2013 16:45')</code>	返回 25/01/2013 00:00:00。																										
<code>daystart('25/01/2013 16:45', -1)</code>	返回 24/01/2013 00:00:00。																										
<code>daystart('25/01/2013 16:45', 0, 0.5 )</code>	返回 25/01/2013 12:00:00。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找标记每个发票日期之后当天开始的时间戳。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, DayStart(InvDate, 1) AS DStart Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>daystart()</code> 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。</p> <table> <thead> <tr> <th>InvDate</th><th>DStart</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>29/03/2012 00:00:00</td></tr> <tr><td>10/12/2012</td><td>11/12/2012 00:00:00</td></tr> <tr><td>5/2/2013</td><td>07/02/2013 00:00:00</td></tr> <tr><td>31/3/2013</td><td>01/04/2013 00:00:00</td></tr> <tr><td>19/5/2013</td><td>20/05/2013 00:00:00</td></tr> <tr><td>15/9/2013</td><td>16/09/2013 00:00:00</td></tr> <tr><td>11/12/2013</td><td>12/12/2013 00:00:00</td></tr> <tr><td>2/3/2014</td><td>03/03/2014 00:00:00</td></tr> <tr><td>14/5/2014</td><td>15/05/2014 00:00:00</td></tr> <tr><td>13/6/2014</td><td>14/06/2014 00:00:00</td></tr> <tr><td>7/7/2014</td><td>08/07/2014 00:00:00</td></tr> <tr><td>4/8/2014</td><td>05/08/2014 00:00:00</td></tr> </tbody> </table>	InvDate	DStart	28/03/2012	29/03/2012 00:00:00	10/12/2012	11/12/2012 00:00:00	5/2/2013	07/02/2013 00:00:00	31/3/2013	01/04/2013 00:00:00	19/5/2013	20/05/2013 00:00:00	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
InvDate	DStart																										
28/03/2012	29/03/2012 00:00:00																										
10/12/2012	11/12/2012 00:00:00																										
5/2/2013	07/02/2013 00:00:00																										
31/3/2013	01/04/2013 00:00:00																										
19/5/2013	20/05/2013 00:00:00																										
15/9/2013	16/09/2013 00:00:00																										
11/12/2013	12/12/2013 00:00:00																										
2/3/2014	03/03/2014 00:00:00																										
14/5/2014	15/05/2014 00:00:00																										
13/6/2014	14/06/2014 00:00:00																										
7/7/2014	08/07/2014 00:00:00																										
4/8/2014	05/08/2014 00:00:00																										

## firstworkdate

**firstworkdate** 函数用于返回最近的起始日以获得 **no\_of\_workdays**(周一至周五), 将任何列出的可选节假日考虑在内, 不迟于 **end\_date**。**end\_date** 和 **holiday** 应为有效的日期或时间戳。

### Syntax:

```
firstworkdate(end_date, no_of_workdays {, holiday} )
```

**Return data type:** 整数

### 参数:

参数	说明
<b>end_date</b>	要求值的结束日期的时间戳。
<b>no_of_workdays</b>	要实现的工作日天数。
<b>holiday</b>	<p>从工作日排除假期。假期可表述为开始日期和结束日期, 以逗号分隔。</p> <p><b>示例:</b> '25/12/2013', '26/12/2013'</p> <p>您可以排除多个假期, 以逗号分隔。</p> <p><b>示例:</b> '25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014'</p>

### 示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>firstworkdate ('29/12/2014', 9)</code>	返回 17/12/2014。
<code>firstworkdate ('29/12/2014', 9, '25/12/2014', '26/12/2014')</code>	返回 15/12/2014, 因为已将两天假期考虑在内。

示例	结果																					
<p>添加示例脚本到应用程序并运行。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。</p> <pre>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;</pre>	<p>结果列表显示了为表格中的每条记录返回的 FirstWorkDate 值。</p> <table><tr><th>InVID</th><th>EndDate</th><th>StartDate</th></tr><tr><td>1</td><td>28/03/2015</td><td>13/10/2014</td></tr><tr><td>2</td><td>10/12/2015</td><td>26/06/2015</td></tr><tr><td>3</td><td>5/2/2016</td><td>24/08/2015</td></tr><tr><td>4</td><td>31/3/2016</td><td>16/10/2015</td></tr><tr><td>5</td><td>19/5/2016</td><td>04/12/2015</td></tr><tr><td>6</td><td>15/9/2016</td><td>01/04/2016</td></tr></table>	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
InVID	EndDate	StartDate																				
1	28/03/2015	13/10/2014																				
2	10/12/2015	26/06/2015																				
3	5/2/2016	24/08/2015																				
4	31/3/2016	16/10/2015																				
5	19/5/2016	04/12/2015																				
6	15/9/2016	01/04/2016																				

## GMT

此函数用于返回来自系统时钟的当前 Greenwich Mean Time 和 Windows 时间设置。

### Syntax:

**GMT ( )**

**Return data type:** 双

### 示例:

gmt ( )

## hour

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示小时的整数。

### Syntax:

**hour (expression)**

**Return data type:** 整数

### 示例和结果:

示例	结果
hour( '09:14:36' )	返回 9
hour( '0.5555' )	返回 13( 因为 0.5555 = 13:19:55)

## inday

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_timestamp** 的一天以内。

### Syntax:

**InDay** (timestamp, base\_timestamp, period\_no[, day\_start])

**Return data type:** 布尔值

### 参数:

参数	说明
<b>timestamp</b>	想要用来与 <b>base_timestamp</b> 进行比较的日期和时间。
<b>base_timestamp</b>	日期和时间用于计算时间戳的值。
<b>period_no</b>	该天可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数, 其中值 0 表示该天包含 <b>base_timestamp</b> 。 <b>period_no</b> 为负数表示前几天, 正数表示随后的几天。
<b>day_start</b>	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 <b>day_start</b> 。例如, 0.125 表示上午 3 点。

### 示例和结果:

示例	结果
inday ( '12/01/2006 12:23:00', '12/01/2006 00:00:00', 0)	返回 True
inday ( '12/01/2006 12:23:00', '13/01/2006 00:00:00', 0)	返回 False
inday ( '12/01/2006 12:23:00', '12/01/2006 00:00:00', -1)	返回 False
inday ( '11/01/2006 12:23:00', '12/01/2006 00:00:00', -1)	返回 True
inday ( '12/01/2006 12:23:00', '12/01/2006 00:00:00', 0, 0.5)	返回 False
inday ( '12/01/2006 11:23:00', '12/01/2006 00:00:00', 0, 0.5)	返回 True



示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例检查发票日期是否在以 base_timestamp 开始的当天的任何时间内。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvTime 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InDay(InvTime, '28/03/2012 00:00:00', 0) AS InDayEx Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 inDay() 函数的返回值的列。</p> <table> <thead> <tr> <th>InvTime</th><th>InDayEx</th></tr> </thead> <tbody> <tr> <td>28/03/2012</td><td>-1 (True)</td></tr> <tr> <td>10/12/2012</td><td>0 (False)</td></tr> <tr> <td>5/2/2013</td><td>0 (False)</td></tr> <tr> <td>31/3/2013</td><td>0 (False)</td></tr> <tr> <td>19/5/2013</td><td>0 (False)</td></tr> <tr> <td>15/9/2013</td><td>0 (False)</td></tr> <tr> <td>11/12/2013</td><td>0 (False)</td></tr> <tr> <td>2/3/2014</td><td>0 (False)</td></tr> <tr> <td>14/5/2014</td><td>0 (False)</td></tr> <tr> <td>13/6/2014</td><td>0 (False)</td></tr> <tr> <td>7/7/2014</td><td>0 (False)</td></tr> <tr> <td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvTime	InDayEx	28/03/2012	-1 (True)	10/12/2012	0 (False)	5/2/2013	0 (False)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvTime	InDayEx																										
28/03/2012	-1 (True)																										
10/12/2012	0 (False)																										
5/2/2013	0 (False)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## indaytotime

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_timestamp** 为止以及包括 **base\_timestamp** 精确毫秒的日子部分以内。

### Syntax:

```
InDayToTime (timestamp, base_timestamp, period_no[, day_start])
```

**Return data type:** 布尔值

### 参数:

参数	说明
<b>timestamp</b>	想要用来与 <b>base_timestamp</b> 进行比较的日期和时间。
<b>base_timestamp</b>	日期和时间用于计算时间戳的值。
<b>period_no</b>	该天可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数，其中值 0 表示该天包含 <b>base_timestamp</b> 。 <b>period_no</b> 为负数表示前几天，正数表示随后的几天。
<b>day_start</b>	(可选) 如果不想从每一日的午夜开始处理，可指定一个偏移作为某日内时间的小数 <b>day_start</b> 。例如，0.125 表示上午 3 点。

示例和结果：

示例	结果																										
<code>indaytotime ('12/01/2006 12:23:00', '12/01/2006 23:59:00', 0)</code>	返回 True																										
<code>indaytotime ('12/01/2006 12:23:00', '12/01/2006 00:00:00', 0)</code>	返回 False																										
<code>indaytotime ('11/01/2006 12:23:00', '12/01/2006 23:59:00', -1)</code>	返回 True																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例检查发票时间戳是否在以 <b>base_timestamp</b> 开始的当天的 17:00:00 之前。</p> <pre>TempTable: LOAD RecNo() as InVID, * Inline [ InvTime 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InDayToTime(InvTime, '28/03/2012 17:00:00', 0) AS InDayExTT Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>indaytotime()</code> 函数的返回值的列。</p> <table> <thead> <tr> <th>InvTime</th><th>InDayExTT</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>-1 (True)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>0 (False)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvTime	InDayExTT	28/03/2012	-1 (True)	10/12/2012	0 (False)	5/2/2013	0 (False)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvTime	InDayExTT																										
28/03/2012	-1 (True)																										
10/12/2012	0 (False)																										
5/2/2013	0 (False)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inlunarweek

此函数用于判断 **timestamp** 是否位于包含 **base\_date** 的阴历周以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

**Syntax:**

```
InLunarWeek (timestamp, base_date, period_no[, first_week_day])
```

**Return data type:** 布尔值

**参数:**

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算阴历周的值。
<b>period_no</b>	阴历周可通过 <b>period_no</b> 偏移。period_no 为整数，其中值 0 表示该阴历周包含 <b>base_date</b> 。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。
<b>first_week_day</b>	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

**示例和结果:**

示例	结果
<code>inlunarweek('12/01/2013', '14/01/2013', 0)</code>	返回 True。由于 timestamp 的值，12/01/2013 属于 08/01/2013 至 14/01/2013 的周之内。
<code>inlunarweek('12/01/2013', '07/01/2013', 0)</code>	返回 False。由于 base_date，将属于阴历周的 07/01/2013 定义为 01/01/2013 至 07/01/2013。
<code>inlunarweek('12/01/2013', '14/01/2013', -1)</code>	返回 False。由于将 period_no 的值指定为 -1，表示将周移动到前一周，即 01/01/2013 到 07/01/2013。
<code>inlunarweek('07/01/2013', '14/01/2013', -1)</code>	返回 True。与前面的示例相比，时间戳是在考虑到向后移动后的周内。
<code>inlunarweek('11/01/2006', '08/01/2006', 0, 3)</code>	返回 False。由于将 first_week_day 的值指定为 3，表示从 04/01/2013 开始计算的一年，因此 base_date 的值在第一周内，并且 timestamp 的值属于 11/01/2013 至 17/01/2013 周内。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例检查发票日期是否在按四周从 <code>base_date</code> 值开始移动的周内。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InLunarWeek(InvDate, '11/01/2013', 4) AS InLWeekPlus4 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>inlunarweek()</code> 函数的返回值的列。</p> <p>对于 <code>InvDate</code> 5/2/2013 的值，此函数返回 <code>True</code>，因为 <code>base_date</code> 的值，11/01/2013 按四周移动，因此它应属于 5/02/2013 至 11/02/2013 的周内。</p> <table> <tr> <th>InvDate</th><th>InLWeekPlus4</th></tr> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>-1 (True)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </table>	InvDate	InLWeekPlus4	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	-1 (True)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InLWeekPlus4																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	-1 (True)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inlunarweektoday

此函数用于判断 **timestamp** 是否位于截止以及包括 **base\_date** 最后毫秒的阴历周的某部分以内。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

### Syntax:

```
InLunarWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

**Return data type:** 布尔值

### 参数:

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算阴历周的值。

参数	说明
<b>peroid_no</b>	阴历周可通过 <b>period_no</b> 偏移。period_no 为整数，其中值 0 表示该阴历周包含 <b>base_date</b> 。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。
<b>week_start</b>	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

示例和结果：

示例	结果																										
<code>inlunarweektodate('12/01/2013', '13/01/2013', 0)</code>	返回 True。由于 timestamp 的值，12/01/2013 属于 08/01/2013 至 13/01/2013 的周的一部分。																										
<code>inlunarweektodate('12/01/2013', '11/01/2013', 0)</code>	返回 False。由于 timestamp 的值晚于 base_date 的值，尽管这两个日期都属于 12/01/2012 之前的同一阴历周。																										
<code>inlunarweektodate('12/01/2006', '05/01/2006', 1)</code>	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。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例检查发票日期是否属于按四周从 base_date 值开始移动的周的一部分。</p> <p>TempTable:</p> <pre>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 ];</pre> <p>InvoiceData:</p> <pre>LOAD *, InLunarWeekToDate(InvDate, '07/01/2013', 4) AS InLWeek2DPlus4 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 inlunarweek() 函数的返回值的列。</p> <p>对于 InvDate5/2/2013 的值，此函数返回 True，因为 base_date 的值，11/01/2013 按四周移动，因此它属于 5/02/2013 至 07/02/2013 的周的一部分。</p> <table> <thead> <tr> <th>InvDate</th><th>InLWeek2DPlus4</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>-1 (True)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	InLWeek2DPlus4	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	-1 (True)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InLWeek2DPlus4																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	-1 (True)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inmonth

此函数用于返回 True, 如果 **timestamp** 位于包含 **base\_date** 的月份以内。

### Syntax:

```
InMonth (timestamp, base_date, period_no[, first_month_of_year])
```

**Return data type:** 布尔值

### 参数:

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算月份的值。
<b>period_no</b>	该月份可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数, 其中值 0 表示该月份包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几月, 为正数则表示随后的几月。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年, 可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

### 示例和结果:

示例	结果
<code>inmonth ('25/01/2013', '01/01/2013', 0 )</code>	返回 True
<code>inmonth('25/01/2013', '01/04/2013', 0)</code>	返回 False
<code>inmonth ('25/01/2013', '01/01/2013', -1)</code>	返回 False
<code>inmonth ('25/12/2012', '01/01/2013', -1)</code>	返回 True

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例检查发票日期是否属于 base_date 中的月份之后第四个月内的任何时间内(通过将 period_no 指定为 4)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InMonth(InvDate, '31/01/2013', 4) AS InMthPlus4 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 inmonth() 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>InMthPlus4</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>0 (False)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>-1 (True)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	InMthPlus4	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	0 (False)	31/3/2013	0 (False)	19/5/2013	-1 (True)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InMthPlus4																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	0 (False)																										
31/3/2013	0 (False)																										
19/5/2013	-1 (True)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inmonths

此函数用于判断时间戳是否位于作为基准日期的同一个月、两个月、季度、四个月或半年以内。另外，也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

### Syntax:

```
InMonths (n_months, timestamp, base_date, period_no [, first_month_of_year])
```

**Return data type:** 布尔值

### 参数:

参数	说明
<b>n_months</b>	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月)或 6(半年)。
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算周期的值。

参数	说明
<b>period_no</b>	该周期可通过 <b>period_no</b> 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 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几个时段, 为正数则表示随后的几个时段。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年, 可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>inmonths(4, '25/01/2013', '25/04/2013', 0)</code>	返回 True。由于 timestamp 的值, 25/01/2013 属于 01/01/2013 至 30/04/2013 的四个月周期内, 其中 <b>base_date</b> 的值在 25/04/2013 内。
<code>inmonths(4, '25/05/2013', '25/04/2013', 0)</code>	返回 False。由于 25/05/2013 在与前面示例相同的周期之外。
<code>inmonths(4, '25/11/2012', '01/02/2013', -1 )</code>	返回 True。由于 <b>period_no</b> 的值, -1 表示将搜索周期向后移动四个月中的其中一个周期( <b>n-months</b> 的值), 这可以使搜索周期介于 01/09/2012 至 31/12/2012. 之间
<code>inmonths( 4, '25/05/2006', '01/03/2006', 0, 3)</code>	返回 True。由于将 <b>first_month_of_year</b> 的值设置为 3, 这使得搜索周期介于 01/03/2006 至 30/04/2006 之内, 而不是 01/01/2006 至 30/07/2006。



示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中检查发票日期是否在 base_date 按一个两个月周期向前移动的两个月周期内(通过将 period_no 指定为 1)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InMonths(2, InvDate, '11/02/2013', 1) AS InMthsPlus1 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 InMonths() 函数的返回值的列。</p> <p>搜索周期介于 01/03/2013 至 30/04/2013 之间, 因为 base_date 的值从函数中的值 (11/02/2013) 开始向前移动两个月。</p> <table> <thead> <tr> <th>InvDate</th><th>InMthsPlus1</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>0 (False)</td></tr> <tr><td>31/3/2013</td><td>-1 (True)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	InMthsPlus1	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	0 (False)	31/3/2013	-1 (True)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InMthsPlus1																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	0 (False)																										
31/3/2013	-1 (True)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inmonthstodate

此函数用于判断时间戳是否位于截止以及包括 **base\_date** 的最后毫秒的某个月、两个月、季度、四个月或半年周期的一部分以内。另外, 它也可以用于判断时间戳是否位于上一个或下一个时间周期以内。

### Syntax:

```
InMonths (n_months, timestamp, base_date, period_no[, first_month_of_year])
```

**Return data type:** 布尔值

**参数:**

参数	说明
<b>n_months</b>	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月)或 6(半年)。
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算周期的值。
<b>period_no</b>	该周期可通过 <b>period_no</b> 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 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几个时段，为正数则表示随后的几个时段。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
inmonthstodate(4, '25/01/2013', '25/04/2013', 0)	返回 True。由于 timestamp 的值，25/01/2013 属于 01/01/2013 至 25/04/2013 结束的四个月周期内，其中 base_date 的值在 25/04/2013 内。
inmonthstodate(4, '26/04/2013', '25/04/2006', 0)	返回 False。由于 26/04/2013 在与前面示例相同的周期之外。
inmonthstodate(4, '25/09/2005', '01/02/2006', -1)	返回 True。由于 period_no 的值，-1 表示将搜索周期向后移动四个月中的其中一个周期( n-months 的值)，这可以使搜索周期介于 01/09/2012 至 01/02/2012. 之间
inmonthstodate(4, '25/04/2006', '01/06/2006', 0, 3)	返回 True。由于将 first_month_of_year 的值设置为 3，这使得搜索周期介于 01/03/2006 至 01/06/2006 之内，而不是 01/05/2006 至 01/06/2006。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中检查发票日期是否在两个月周期截止并包括 base_date 按两个月周期向前移动的一部分内(通过将 period_no 指定为 4)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InMonthsToDate(2, InvDate, '15/02/2013', 4) AS InMths2DPlus4 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 InMonths() 函数的返回值的列。</p> <p>搜索周期介于 01/09/2013 至 15/10/2013 之间，因为 base_date 的值从函数中的值 (15/02/2013) 开始向前移动八个月。</p> <table> <thead> <tr> <th>InvDate</th><th>InMths2DPlus4</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>0 (False)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>-1 (True)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	InMths2DPlus4	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	0 (False)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	-1 (True)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InMths2DPlus4																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	0 (False)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	-1 (True)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inmonthtodate

用于返回 True，如果 **date** 位于包含 **basedate** 为止以及包括 **basedate** 最后毫秒的月份部分以内。

### Syntax:

```
InMonthToDate (timestamp, base_date, period_no)
```

**Return data type:** 布尔值

### 参数:

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算月份的值。

参数	说明
<b>period_no</b>	该月份可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数，其中值 0 表示该月份包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几月，为正数则表示随后的几月。

示例和结果：

示例	结果																										
<code>inmonthtodate ('25/01/2013', '25/01/2013', 0)</code>	返回 True																										
<code>inmonthtodate ('25/01/2013', '24/01/2013', 0)</code>	返回 False																										
<code>inmonthtodate ('25/01/2013', '28/02/2013', -1)</code>	返回 True																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>通过将 <b>period_no</b> 指定为 4，以下示例检查发票日期是否在 <b>base_date</b> 中的月份之后，但在 <b>base_date</b> 中指定的当天结束之前的第四个月中。</p> <pre>TempTable: LOAD RecNo() as InVID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InMonthToDate(InvDate, '31/01/2013', 0, 4) AS InMthPlus42D Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>inmonthtodate()</code> 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>InMthPlus42D</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>0 (False)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>-1 (True)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	InMthPlus42D	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	0 (False)	31/3/2013	0 (False)	19/5/2013	-1 (True)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InMthPlus42D																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	0 (False)																										
31/3/2013	0 (False)																										
19/5/2013	-1 (True)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inquarter

此函数用于返回 True，如果 **timestamp** 位于包含 **base\_date** 的季度以内。

### Syntax:

```
InQuarter (timestamp, base_date, period_no[, first_month_of_year])
```

**Return data type:** 布尔值

参数：

参数	说明
<b>timestamps</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算季度的值。
<b>period_no</b>	该季度可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数，其中值 0 表示该季度包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几季，为正数则表示随后的几季。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

示例	结果
<code>inquarter ('25/01/2013', '01/01/2013', 0)</code>	返回 True
<code>inquarter ('25/01/2013', '01/04/2013', 0)</code>	返回 False
<code>inquarter ('25/01/2013', '01/01/2013', -1)</code>	返回 False
<code>inquarter ('25/12/2012', '01/01/2013', -1)</code>	返回 True
<code>inquarter ('25/01/2013', '01/03/2013', 0, 3)</code>	返回 False
<code>inquarter ('25/03/2013', '01/03/2013', 0, 3)</code>	返回 True

示例	结果
<p>添加示例脚本到应用程序并运行。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。</p> <p>通过将 <code>first_month_of_year</code> 的值设置为 4，并包括 <code>base_date</code> 31/01/2013，以下示例检查发票日期是否在指定财政年的第四个季度内。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InQuarter(InvDate, '31/01/2013', 0, 4) AS Qtr4FinYr1213 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>inquarter()</code> 函数的返回值的列。</p> <pre>InvDate      Qtr4Fin1213 28/03/2012   0 (False) 10/12/2012   0 (False) 5/2/2013     -1 (True) 31/3/2013    -1 (True) 19/5/2013    0 (False) 15/9/2013    0 (False) 11/12/2013   0 (False) 2/3/2014     0 (False) 14/5/2014    0 (False) 13/6/2014    0 (False) 7/7/2014     0 (False) 4/8/2014     0 (False)</pre>

## 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。

### Syntax:

```
InQuarterToDate (timestamp, base_date, period_no [, first_month_of_year])
```

**Return data type:** 布尔值

### 参数：

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算季度的值。

参数	说明
<b>period_no</b>	该季度可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数，其中值 0 表示该季度包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几季，为正数则表示随后的几季。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

示例	结果																										
<code>inquartertodate ('25/01/2013', '25/01/2013', 0)</code>	返回 True																										
<code>inquartertodate ( '25/01/2013', '24/01/2013', 0)</code>	返回 False																										
<code>inquartertodate ('25/01/2012', '01/02/2013', -1)</code>	返回 True																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>通过将 <b>first_month_of_year</b> 的值设置为 4，以下示例检查发票日期是否在指定财政年的第四个季度，并在 28/02/2013 结束之前。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InQuarterToDate(InvDate, '28/02/2013', 0, 4) AS Qtr42Date Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>inquartertodate()</code> 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>Qtr42Date</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>-1 (True)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	Qtr42Date	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	-1 (True)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	Qtr42Date																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	-1 (True)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inweek

此函数用于返回 True，如果 **timestamp** 位于包含 **base\_date** 的星期以内。

**Syntax:**

```
InWeek (timestamp, base_date, period_no[, first_week_day])
```

**Return data type:** 布尔值

**参数:**

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算星期的值。
<b>period_no</b>	该星期可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数, 其中值 0 表示该星期包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几个星期, 正数表示随后的几个星期。
<b>first_week_day</b>	默认情况下, 一周的第一天为星期一, 从星期天与星期一之间的午夜开始。要指示一周从另外一天开始, 应在 <b>first_week_day</b> 中指定偏移。这需要指定一个表示天数和/或天的分位数的整数。

**示例和结果:**

示例	结果
<code>inweek ('12/01/2006', '14/01/2006', 0)</code>	返回 True
<code>inweek ('12/01/2006', '20/01/2006', 0 )</code>	返回 False
<code>inweek ('12/01/2006', '14/01/2006', -1 )</code>	返回 False
<code>inweek ('07/01/2006', '14/01/2006', -1)</code>	返回 True
<code>inweek ('12/01/2006', '09/01/2006', 0, 3)</code>	返回 False 由于将 <b>first_week_day</b> 指定为 3(星期四), 这使得本周的第一天 09/01/2006 之后一周包含 12/01/2006。



示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例检查发票日期是否属于 base_date 中的周之后第四周的任何时间内(通过将 period_no 指定为 4)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InWeek(InvDate, '11/01/2013', 4) AS InWeekPlus4 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 inweek() 函数的返回值的列。</p> <p>InvDate5/2/2013 属于以下日期之后四周的某一周中: base_date:11/1/2013。</p> <table> <thead> <tr> <th>InvDate</th><th>InWeekPlus4</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>-1 (True)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	InWeekPlus4	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	-1 (True)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InWeekPlus4																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	-1 (True)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inweektoday

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。

### Syntax:

```
InWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

**Return data type:** 布尔值

### 参数:

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算星期的值。

参数	说明
<b>period_no</b>	该星期可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数，其中值 0 表示该星期包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几个星期，正数表示随后的几个星期。
<b>first_week_day</b>	默认情况下，一周的第一天为星期一，从星期天与星期一之间的午夜开始。要指示一周从另外一天开始，应在 <b>first_week_day</b> 中指定偏移。这需要指定一个表示天数和/或天的分位数的整数。

示例和结果：

示例	结果
<code>inweektoday ('12/01/2006', '12/01/2006', 0)</code>	返回 True
<code>inweektoday ('12/01/2006', '11/01/2006', 0)</code>	返回 False
<code>inweektoday ('12/01/2006', '18/01/2006', -1)</code>	返回 False 由于将 <b>period_no</b> 指定为 -1，依据 timestamp 衡量的有效日期为 11/01/2006。
<code>inweektoday ('11/01/2006', '12/01/2006', 0, 3 )</code>	返回 False 由于将 <b>first_week_day</b> 指定为 3(星期四)，这使得本周的第一天 12/01/2006 之后一周包含 12/01/2006。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例检查发票日期是否在 base_date 中的周之后第四周期间，但在 base_date 的值之前(通过将 period_no 指定为 4)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InWeekToDate(InvDate, '11/01/2013', 4) AS InWeek2DPlus4 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 inweek() 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>InWeek2DPlus4</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>0 (False)</td></tr> <tr><td>5/2/2013</td><td>-1 (True)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	InWeek2DPlus4	28/03/2012	0 (False)	10/12/2012	0 (False)	5/2/2013	-1 (True)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	InWeek2DPlus4																										
28/03/2012	0 (False)																										
10/12/2012	0 (False)																										
5/2/2013	-1 (True)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inyear

此函数用于返回 True，如果 **timestamp** 位于包含 **base\_date** 的年份以内。

### Syntax:

```
InYear (timestamp, base_date, period_no [, first_month_of_year])
```

**Return data type:** 布尔值

### 参数:

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。
<b>base_date</b>	日期用于计算年份的值。

参数	说明
<b>period_no</b>	该年份可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数，其中值 0 表示该年份包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几年，为正数表示随后几年。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>inyear ('25/01/2013', '01/01/2013', 0 )</code>	返回 True
<code>inyear ('25/01/2012', '01/01/2013', 0)</code>	返回 False
<code>inyear ('25/01/2013', '01/01/2013', -1)</code>	返回 False
<code>inyear ('25/01/2012', '01/01/2013', -1 )</code>	返回 True
<code>inyear ('25/01/2013', '01/01/2013', 0, 3)</code>	返回 False <b>base_date</b> 和 <b>first_month_of_year</b> 的值指定 timestamp 必须在 01/03/2012 和 28/02/2013 内
<code>inyear ('25/03/2013', '2013/07/01', 0, 3 )</code>	返回 True

示例	结果																										
<p>添加示例脚本到应用程序并运行。然后，至少要将结果列表中列出的字段添加到应用程序中的表格才能查看结果。</p> <p>通过将 <code>first_month_of_year</code> 的值设置为 4，并包括介于 1/4/2012 和 31/03/2013 之间 <code>base_date</code>，以下示例检查发票日期是否在指定财政年内。</p> <pre>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 ];</pre> <p>测试 <code>InvDate</code> 是否在 1/04/2012 至 31/03/2013 财政年内：</p> <pre>InvoiceData: LOAD *, InYear(InvDate, '31/01/2013', 0, 4) AS FinYr1213 Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>inyear()</code> 函数的返回值的列。</p> <table> <tr> <th>InvDate</th><th>FinYr1213</th></tr> <tr> <td>28/03/2012</td><td>0 (False)</td></tr> <tr> <td>10/12/2012</td><td>-1 (True)</td></tr> <tr> <td>5/2/2013</td><td>-1 (True)</td></tr> <tr> <td>31/3/2013</td><td>-1 (True)</td></tr> <tr> <td>19/5/2013</td><td>0 (False)</td></tr> <tr> <td>15/9/2013</td><td>0 (False)</td></tr> <tr> <td>11/12/2013</td><td>0 (False)</td></tr> <tr> <td>2/3/2014</td><td>0 (False)</td></tr> <tr> <td>14/5/2014</td><td>0 (False)</td></tr> <tr> <td>13/6/2014</td><td>0 (False)</td></tr> <tr> <td>7/7/2014</td><td>0 (False)</td></tr> <tr> <td>4/8/2014</td><td>0 (False)</td></tr> </table>	InvDate	FinYr1213	28/03/2012	0 (False)	10/12/2012	-1 (True)	5/2/2013	-1 (True)	31/3/2013	-1 (True)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	FinYr1213																										
28/03/2012	0 (False)																										
10/12/2012	-1 (True)																										
5/2/2013	-1 (True)																										
31/3/2013	-1 (True)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## inyeartodate

此函数用于返回 True，如果 **timestamp** 位于包含 **base\_date** 为止以及包括 **base\_date** 最后毫秒的年份部分以内。

### Syntax:

**InYearToDate** (timestamp, base\_date, period\_no[, first\_month\_of\_year])

**Return data type:** 布尔值

### 参数：

参数	说明
<b>timestamp</b>	想要用来与 <b>base_date</b> 进行比较的日期。

参数	说明
<b>base_date</b>	日期用于计算年份的值。
<b>period_no</b>	该年份可通过 <b>period_no</b> 偏移。 <b>period_no</b> 为整数，其中值 0 表示该年份包含 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几年，为正数表示随后几年。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

示例	结果
<code>inyeartodate ('2013/01/25', '2013/02/01', 0)</code>	返回 True
<code>inyeartodate ('2012/01/25', '2013/01/01', 0)</code>	返回 False
<code>inyeartodate ('2012/01/25', '2013/02/01', - )</code>	返回 True
<code>inyeartodate ('2012/11/25', '2013/01/31', 0, 4)</code>	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。
<code>inyeartodate ( '2013/3/31', '2013/01/31', 0, 4 )</code>	返回 False 与前面的示例相比，timestamp 的值仍在财政年内，但它在 base_date 的值之后，因此它属于一年的某一部分以外。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>通过将 <code>first_month_of_year</code> 的值设置为 4，以下示例检查发票日期是否在指定财政年，并在 31/01/2013 结束之前一年的某一部分内。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, InYearToDate(InvDate, '31/01/2013', 0, 4) AS FinYr2Date Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>inyeartodate()</code> 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>FinYr2Date</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>0 (False)</td></tr> <tr><td>10/12/2012</td><td>-1 (True)</td></tr> <tr><td>5/2/2013</td><td>0 (False)</td></tr> <tr><td>31/3/2013</td><td>0 (False)</td></tr> <tr><td>19/5/2013</td><td>0 (False)</td></tr> <tr><td>15/9/2013</td><td>0 (False)</td></tr> <tr><td>11/12/2013</td><td>0 (False)</td></tr> <tr><td>2/3/2014</td><td>0 (False)</td></tr> <tr><td>14/5/2014</td><td>0 (False)</td></tr> <tr><td>13/6/2014</td><td>0 (False)</td></tr> <tr><td>7/7/2014</td><td>0 (False)</td></tr> <tr><td>4/8/2014</td><td>0 (False)</td></tr> </tbody> </table>	InvDate	FinYr2Date	28/03/2012	0 (False)	10/12/2012	-1 (True)	5/2/2013	0 (False)	31/3/2013	0 (False)	19/5/2013	0 (False)	15/9/2013	0 (False)	11/12/2013	0 (False)	2/3/2014	0 (False)	14/5/2014	0 (False)	13/6/2014	0 (False)	7/7/2014	0 (False)	4/8/2014	0 (False)
InvDate	FinYr2Date																										
28/03/2012	0 (False)																										
10/12/2012	-1 (True)																										
5/2/2013	0 (False)																										
31/3/2013	0 (False)																										
19/5/2013	0 (False)																										
15/9/2013	0 (False)																										
11/12/2013	0 (False)																										
2/3/2014	0 (False)																										
14/5/2014	0 (False)																										
13/6/2014	0 (False)																										
7/7/2014	0 (False)																										
4/8/2014	0 (False)																										

## lastworkdate

**lastworkdate** 函数用于返回最早的结束日以获得 **no\_of\_workdays** (周一至周五)，如果在 **start\_date** 开始考虑任何列出的可选 **holiday**。**start\_date** 和 **holiday** 应是有效的日期或时间戳。

### Syntax:

```
lastworkdate(start_date, no_of_workdays {, holiday})
```

**Return data type:** 双

### 参数:

参数	说明
<b>start_date</b>	要求值的开始日期的时间戳。
<b>no_of_workdays</b>	要实现的工作日天数。

参数	说明
<b>holiday</b>	<p>从工作日排除假期。假期可表述为开始日期和结束日期，以逗号分隔。</p> <p><b>示例：</b> '25/12/2013', '26/12/2013'</p> <p>您可以排除多个假期，以逗号分隔。</p> <p><b>示例：</b> '25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014'</p>

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																					
lastworkdate ('19/12/2014', 9)	返回“31/12/2014”																					
lastworkdate ('19/12/2014', 9, '2014-12-25', '2014-12-26')	返回 02/01/2015, 因为已将两天假期考虑在内。																					
添加示例脚本到应用程序并运行。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。	结果列表显示了为表格中的每条记录返回的 LastWorkDate 值。																					
ProjectTable: LOAD *, recno() as InVID, INLINE [ StartDate 28/03/2014 10/12/2014 5/2/2015 31/3/2015 19/5/2015 15/9/2015 ] ; NrDays: Load *, LastWorkDate(StartDate,120) As EndDate Resident ProjectTable; Drop table ProjectTable;	<table><thead><tr><th>InVID</th><th>StartDate</th><th>EndDate</th></tr></thead><tbody><tr><td>1</td><td>28/03/2014</td><td>11/09/2014</td></tr><tr><td>2</td><td>10/12/2014</td><td>26/05/2015</td></tr><tr><td>3</td><td>5/2/2015</td><td>27/07/2015</td></tr><tr><td>4</td><td>31/3/2015</td><td>14/09/2015</td></tr><tr><td>5</td><td>19/5/2015</td><td>02/11/2015</td></tr><tr><td>6</td><td>15/9/2015</td><td>29/02/2016</td></tr></tbody></table>	InVID	StartDate	EndDate	1	28/03/2014	11/09/2014	2	10/12/2014	26/05/2015	3	5/2/2015	27/07/2015	4	31/3/2015	14/09/2015	5	19/5/2015	02/11/2015	6	15/9/2015	29/02/2016
InVID	StartDate	EndDate																				
1	28/03/2014	11/09/2014																				
2	10/12/2014	26/05/2015																				
3	5/2/2015	27/07/2015																				
4	31/3/2015	14/09/2015																				
5	19/5/2015	02/11/2015																				
6	15/9/2015	29/02/2016																				

## localtime

此函数用于返回指定时区的系统时钟的当前时间戳。

**Syntax:**

```
LocalTime([timezone [, ignoreDST ]])
```

**Return data type:** 双

**参数：**



参数	说明
timezone	将 <b>timezone</b> 指定为一个字符串，其中包含在 <b>Windows Control Panel</b> 的 <b>Time Zone</b> 下专为 <b>Date and Time</b> 列出的任何一个地理位置，或指定为“GMT+hh:mm”格式的字符串。  如果未指定时区，则将返回本地时间。
ignoreDST	如果 <b>ignoreDST</b> 为 -1 (True)，则将忽略夏令时。

**示例和结果：**

以下示例基于在 2014-10-22 12:54:47 本地时间调用的函数，本地时区为 GMT+01:00。

示例	结果
localtime ()	返回本地时间 2014-10-22 12:54:47。
localtime ('London')	返回伦敦本地时间 2014-10-22 11:54:47。
localtime ('GMT+02:00')	返回 GMT+02:00 时区的本地时间 2014-10-22 13:54:47。
localtime ('Paris',-1')	返回巴黎本地时间 2014-10-22 11:54:47，忽略夏令时。

**lunarweekend**

此函数用于返回与包含 **date** 的阴历周的最后毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

**Syntax:**

```
LunarweekEnd(date[, period_no[, first_week_day]])
```

**Return data type:** 双**参数：**

参数	说明
date	要求值的日期。
period_no	<b>period_no</b> 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 <b>date</b> 。 <b>period_no</b> 为负数表示前几个阴历星期，为正数表示随后的几个阴历星期。
first_week_day	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

**示例和结果：**

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句

中指定。可以根据要求更改示例中的格式。

示例	结果																										
<code>lunarweekend('12/01/2013')</code>	返回 14/01/2013 23:59:59。																										
<code>lunarweekend('12/01/2013', -1)</code>	返回 7/01/2013 23:59:59。																										
<code>lunarweekend('12/01/2013', 0, 1)</code>	返回 15/01/2013 23:59:59。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找每个发票日期的阴历周的最后一天，其中 <code>date</code> 按一周移动(通过将 <code>period_no</code> 指定为 1)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, LunarWeekEnd(InvDate, 1) AS LwkEnd Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>lunarweekend()</code> 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。</p> <table> <thead> <tr> <th>InvDate</th><th>LWkEnd</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>07/04/2012</td></tr> <tr><td>10/12/2012</td><td>22/12/2012</td></tr> <tr><td>5/2/2013</td><td>18/02/2013</td></tr> <tr><td>31/3/2013</td><td>08/04/2013</td></tr> <tr><td>19/5/2013</td><td>27/05/2013</td></tr> <tr><td>15/9/2013</td><td>23/09/2013</td></tr> <tr><td>11/12/2013</td><td>23/12/2013</td></tr> <tr><td>2/3/2014</td><td>11/03/2014</td></tr> <tr><td>14/5/2014</td><td>27/05/2014</td></tr> <tr><td>13/6/2014</td><td>24/06/2014</td></tr> <tr><td>7/7/2014</td><td>15/07/2014</td></tr> <tr><td>4/8/2014</td><td>12/08/2014</td></tr> </tbody> </table>	InvDate	LWkEnd	28/03/2012	07/04/2012	10/12/2012	22/12/2012	5/2/2013	18/02/2013	31/3/2013	08/04/2013	19/5/2013	27/05/2013	15/9/2013	23/09/2013	11/12/2013	23/12/2013	2/3/2014	11/03/2014	14/5/2014	27/05/2014	13/6/2014	24/06/2014	7/7/2014	15/07/2014	4/8/2014	12/08/2014
InvDate	LWkEnd																										
28/03/2012	07/04/2012																										
10/12/2012	22/12/2012																										
5/2/2013	18/02/2013																										
31/3/2013	08/04/2013																										
19/5/2013	27/05/2013																										
15/9/2013	23/09/2013																										
11/12/2013	23/12/2013																										
2/3/2014	11/03/2014																										
14/5/2014	27/05/2014																										
13/6/2014	24/06/2014																										
7/7/2014	15/07/2014																										
4/8/2014	12/08/2014																										

## lunarweekname

此函数用于返回一个显示值，显示与包含 **date** 的阴历周的第一天的第一毫秒时间戳对应的年份和阴历周数。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

### Syntax:

```
LunarWeekName (date [, period_no[, first_week_day]])
```

**Return data type:** 双

**参数:**

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 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 <b>date</b> 。 <b>period_no</b> 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。
<b>first_week_day</b>	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

示例和结果:

示例	结果																										
<code>lunarweekname('12/01/2013')</code>	返回 2006/02。																										
<code>lunarweekname('12/01/2013', -1)</code>	返回 2006/01。																										
<code>lunarweekname('12/01/2013', 0, 1)</code>	返回 2006/02。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>在此例中, 对于表格中的每个发票日期, 根据周所在的年度创建阴历周名称, 并将其与阴历周编号进行相关联(通过将 <code>period_no</code> 指定为 1 按一周移动)。</p> <p>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            ];</p> <p>InvoiceData:            LOAD *,            LunarWeekName(InvDate, 1) AS LwkName            Resident TempTable;            Drop table TempTable;</p>	<p>结果列表包含原始日期和包括            lunarweekname() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。</p> <table> <thead> <tr> <th>InvDate</th><th>LWkName</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>2012/14</td></tr> <tr><td>10/12/2012</td><td>2012/51</td></tr> <tr><td>5/2/2013</td><td>2013/07</td></tr> <tr><td>31/3/2013</td><td>2013/14</td></tr> <tr><td>19/5/2013</td><td>2013/21</td></tr> <tr><td>15/9/2013</td><td>2013/38</td></tr> <tr><td>11/12/2013</td><td>2013/51</td></tr> <tr><td>2/3/2014</td><td>2014/10</td></tr> <tr><td>14/5/2014</td><td>2014/21</td></tr> <tr><td>13/6/2014</td><td>2014/25</td></tr> <tr><td>7/7/2014</td><td>2014/28</td></tr> <tr><td>4/8/2014</td><td>2014/32</td></tr> </tbody> </table>	InvDate	LWkName	28/03/2012	2012/14	10/12/2012	2012/51	5/2/2013	2013/07	31/3/2013	2013/14	19/5/2013	2013/21	15/9/2013	2013/38	11/12/2013	2013/51	2/3/2014	2014/10	14/5/2014	2014/21	13/6/2014	2014/25	7/7/2014	2014/28	4/8/2014	2014/32
InvDate	LWkName																										
28/03/2012	2012/14																										
10/12/2012	2012/51																										
5/2/2013	2013/07																										
31/3/2013	2013/14																										
19/5/2013	2013/21																										
15/9/2013	2013/38																										
11/12/2013	2013/51																										
2/3/2014	2014/10																										
14/5/2014	2014/21																										
13/6/2014	2014/25																										
7/7/2014	2014/28																										
4/8/2014	2014/32																										

## lunarweekstart

此函数用于返回与包含 **date** 的阴历周的第一毫秒的时间戳对应的值。Qlik Sense 中的阴历周将 1 月 1 日定义为一周的第一天。

### Syntax:

```
LunarweekStart(date[, period_no[, first_week_day]])
```

**Return data type:** 双

### 参数:

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 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 <b>date</b> 。 <b>period_no</b> 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。
<b>first_week_day</b>	偏移可以大小或小于零。这可以按指定的天数和/或某日内时间的小数对更改一年的开始。

### 示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>lunarweekstart('12/01/2013')</code>	返回 08/01/2013。
<code>lunarweekstart('12/01/2013', -1)</code>	返回 01/01/2013。
<code>lunarweekstart('12/01/2013', 0, 1 )</code>	返回 09/01/2013。 由于是通过将 <b>first_week_day</b> 设置为 1 来指定偏移量, 这意味着已将一年的开始更改为 02/01/2013。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找每个发票日期的阴历周的第一天，其中 date 按一周移动(通过将 period_no 指定为 1)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, LunarWeekStart(InvDate, 1) AS LwkStart Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 lunarweekstart () 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。</p> <table> <tr> <th>InvDate</th><th>LWkStart</th></tr> <tr> <td>28/03/2012</td><td>01/04/2012</td></tr> <tr> <td>10/12/2012</td><td>16/12/2012</td></tr> <tr> <td>5/2/2013</td><td>12/02/2013</td></tr> <tr> <td>31/3/2013</td><td>02/04/2013</td></tr> <tr> <td>19/5/2013</td><td>21/05/2013</td></tr> <tr> <td>15/9/2013</td><td>17/09/2013</td></tr> <tr> <td>11/12/2013</td><td>17/12/2013</td></tr> <tr> <td>2/3/2014</td><td>05/03/2014</td></tr> <tr> <td>14/5/2014</td><td>21/05/2014</td></tr> <tr> <td>13/6/2014</td><td>18/06/2014</td></tr> <tr> <td>7/7/2014</td><td>09/07/2014</td></tr> <tr> <td>4/8/2014</td><td>06/08/2014</td></tr> </table>	InvDate	LWkStart	28/03/2012	01/04/2012	10/12/2012	16/12/2012	5/2/2013	12/02/2013	31/3/2013	02/04/2013	19/5/2013	21/05/2013	15/9/2013	17/09/2013	11/12/2013	17/12/2013	2/3/2014	05/03/2014	14/5/2014	21/05/2014	13/6/2014	18/06/2014	7/7/2014	09/07/2014	4/8/2014	06/08/2014
InvDate	LWkStart																										
28/03/2012	01/04/2012																										
10/12/2012	16/12/2012																										
5/2/2013	12/02/2013																										
31/3/2013	02/04/2013																										
19/5/2013	21/05/2013																										
15/9/2013	17/09/2013																										
11/12/2013	17/12/2013																										
2/3/2014	05/03/2014																										
14/5/2014	21/05/2014																										
13/6/2014	18/06/2014																										
7/7/2014	09/07/2014																										
4/8/2014	06/08/2014																										

## makedate

此函数用于返回根据年份 **YYYY**、月份 **MM** 和日期 **DD** 计算的日期。

### Syntax:

**MakeDate** (YYYY [ , MM [ , DD ] ])

**Return data type:** 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，则假定为 1(第 1 天)。

示例和结果：

示例	结果
makedate(2012)	返回 2012-01-01
makedate(12)	返回 2012-01-01
makedate(2012,12)	返回 2012-12-01
makedate(2012,2,14)	返回 2012-02-14

## maketime

此函数用于返回根据小时 **hh**、分钟 **mm** 和秒 **ss** 计算的时间。

**Syntax:**

```
MakeTime(hh [ , mm [ , ss ] ])
```

**Return data type:** 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。

示例和结果：

示例	结果
maketime( 22 )	返回 22:00:00
maketime( 22, 17 )	返回 22:17:00
maketime( 22, 17, 52 )	返回 22:17:52

## makeweekdate

此函数用于返回根据年份 **YYYY**、星期 **WW** 和星期几 **D** 计算的日期。

**Syntax:**

```
MakeWeekDate(YYYY [ , WW [ , D ] ])
```

**Return data type:** 双

参数：

参数	说明
YYYY	作为整数的年份。
WW	作为整数的周。
D	作为整数的星期几。 如果未指定星期几，则假定为 0(星期一)。

示例和结果：

示例	结果
<code>makeweekdate(2014,6,6)</code>	返回 2014-02-09
<code>makeweekdate(2014,6,1)</code>	返回 2014-02-04
<code>makeweekdate(2014,6)</code>	返回 2014-02-03(假定普通日为 0)

## minute

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示分钟的整数。

**Syntax:**

```
minute(expression)
```

**Return data type:** 整数

示例和结果：

示例	结果
<code>minute ( '09:14:36' )</code>	返回 14
<code>minute ( '0.5555' )</code>	返回 19( 因为 0.5555 = 13:19:55)

## month

此函数用于返回包含在环境变量 **MonthNames** 中定义的月份名称的对偶值和一个介于 1 至 12 的整数。月份根据标准数字解释通过表达式的日期解释进行计算。

**Syntax:**

```
month(expression)
```

**Return data type:** 双

示例和结果：

示例	结果
<code>month( '2012-10-12' )</code>	返回 Oct
<code>month( '35648' )</code>	返回 Aug, 因为 35648 = 1997-08-06

## monthend

此函数用于返回与包含 **date** 的月份的最后一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

**Syntax:**

```
MonthEnd(date[, period_no])
```

**Return data type:** 双

**参数：**

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数，如果为 0 或忽略，表示该月包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几个月，为正数则表示随后的几个月。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>monthend('19/02/2012')</code>	返回 29/02/2012 23:59:59。
<code>monthend('19/02/2001', -1)</code>	返回 31/02/2012 23:59:59。



示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例用于查找表格中每个发票日期的第一天，其中 base_date 中的月份为偏移(通过将 period_no 指定为 4)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, MonthEnd(InvDate, 4) AS MthEnd Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 monthend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。</p> <table> <thead> <tr> <th>InvDate</th><th>MthEnd</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>31/07/2012</td></tr> <tr><td>10/12/2012</td><td>30/04/2013</td></tr> <tr><td>5/2/2013</td><td>30/06/2013</td></tr> <tr><td>31/3/2013</td><td>31/07/2013</td></tr> <tr><td>19/5/2013</td><td>30/09/2013</td></tr> <tr><td>15/9/2013</td><td>31/01/2014</td></tr> <tr><td>11/12/2013</td><td>30/04/2014</td></tr> <tr><td>2/3/2014</td><td>31/07/2014</td></tr> <tr><td>14/5/2014</td><td>30/09/2014</td></tr> <tr><td>13/6/2014</td><td>31/10/2014</td></tr> <tr><td>7/7/2014</td><td>30/11/2014</td></tr> <tr><td>4/8/2014</td><td>31/12/2014</td></tr> </tbody> </table>	InvDate	MthEnd	28/03/2012	31/07/2012	10/12/2012	30/04/2013	5/2/2013	30/06/2013	31/3/2013	31/07/2013	19/5/2013	30/09/2013	15/9/2013	31/01/2014	11/12/2013	30/04/2014	2/3/2014	31/07/2014	14/5/2014	30/09/2014	13/6/2014	31/10/2014	7/7/2014	30/11/2014	4/8/2014	31/12/2014
InvDate	MthEnd																										
28/03/2012	31/07/2012																										
10/12/2012	30/04/2013																										
5/2/2013	30/06/2013																										
31/3/2013	31/07/2013																										
19/5/2013	30/09/2013																										
15/9/2013	31/01/2014																										
11/12/2013	30/04/2014																										
2/3/2014	31/07/2014																										
14/5/2014	30/09/2014																										
13/6/2014	31/10/2014																										
7/7/2014	30/11/2014																										
4/8/2014	31/12/2014																										

## monthname

此函数用于返回一个显示值，该值显示该月(根据 **MonthNames** 脚本变量的格式)以及年，伴随一个与该月第一天第一毫秒的时间戳对应的基础数值。

### Syntax:

```
MonthName (date[, period_no])
```

Return data type: 双

### 参数:

参数	说明
date	要求值的日期。
period_no	period_no 为整数，如果为 0 或忽略，表示该月包含 date。period_no 为负数表示前几月，为正数则表示随后的几月。

示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																										
<code>monthname('19/10/2013')</code>	返回 Oct 2013。 因为在此例和其他示例中，已将 <b>SET Monthnames</b> 语句设置为 Jan;Feb;Mar，以此类推。																										
<code>monthname('19/10/2013', -1)</code>	返回 Sep 2013。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>在此例中，对于表格中的每个发票日期，根据从该年度的 <code>base_date</code> 移动四个月的月份名称创建月份名称。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, MonthName(InvDate, 4) AS MthName Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>monthname()</code> 函数的返回值的列。在</p> <table> <thead> <tr> <th>InvDate</th><th>MthName</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>Jul 2012</td></tr> <tr><td>10/12/2012</td><td>Apr 2013</td></tr> <tr><td>5/2/2013</td><td>Jun 2013</td></tr> <tr><td>31/3/2013</td><td>Jul 2013</td></tr> <tr><td>19/5/2013</td><td>Sep 2013</td></tr> <tr><td>15/9/2013</td><td>Jan 2014</td></tr> <tr><td>11/12/2013</td><td>Apr 2014</td></tr> <tr><td>2/3/2014</td><td>Jul 2014</td></tr> <tr><td>14/5/2014</td><td>Sep 2014</td></tr> <tr><td>13/6/2014</td><td>Oct 2014</td></tr> <tr><td>7/7/2014</td><td>Nov 2014</td></tr> <tr><td>4/8/2014</td><td>Dec 2014</td></tr> </tbody> </table>	InvDate	MthName	28/03/2012	Jul 2012	10/12/2012	Apr 2013	5/2/2013	Jun 2013	31/3/2013	Jul 2013	19/5/2013	Sep 2013	15/9/2013	Jan 2014	11/12/2013	Apr 2014	2/3/2014	Jul 2014	14/5/2014	Sep 2014	13/6/2014	Oct 2014	7/7/2014	Nov 2014	4/8/2014	Dec 2014
InvDate	MthName																										
28/03/2012	Jul 2012																										
10/12/2012	Apr 2013																										
5/2/2013	Jun 2013																										
31/3/2013	Jul 2013																										
19/5/2013	Sep 2013																										
15/9/2013	Jan 2014																										
11/12/2013	Apr 2014																										
2/3/2014	Jul 2014																										
14/5/2014	Sep 2014																										
13/6/2014	Oct 2014																										
7/7/2014	Nov 2014																										
4/8/2014	Dec 2014																										

### monthsend

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的最后毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

#### Syntax:

```
MonthsEnd (n_month, date[, period_no [, first_month_of_year]])
```

**Return data type:** 双

参数：

参数	说明
<b>n_months</b>	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月)或 6(半年)。
<b>date</b>	要求值的日期。
<b>period_no</b>	该周期可通过 <b>period_no</b> 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 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几个时段，为正数则表示随后的几个时段。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
monthsend(4, '19/07/2013')	返回 31/08/2013。
monthsend(4, '19/10/2013', -1)	返回 31/08/2013。
monthsend(4, '19/10/2013', 0, 2)	返回 31/01/2014。 因为该年度的开始变成 2 月。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例用于查找每个发票日期的两个月周期最后一天的结束，按一个两个月周期向前移动。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, MonthsEnd(2, InvDate, 1) AS BiMthsEnd Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 MonthsEnd() 函数的返回值的列。</p> <table> <tr> <th>InvDate</th><th>BiMthsEnd</th></tr> <tr> <td>28/03/2012</td><td>30/06/2012</td></tr> <tr> <td>10/12/2012</td><td>28/02/2013</td></tr> <tr> <td>5/2/2013</td><td>30/04/2013</td></tr> <tr> <td>31/3/2013</td><td>30/04/2013</td></tr> <tr> <td>19/5/2013</td><td>31/08/2013</td></tr> <tr> <td>15/9/2013</td><td>31/12/2013</td></tr> <tr> <td>11/12/2013</td><td>28/02/2014</td></tr> <tr> <td>2/3/2014</td><td>30/06/2014</td></tr> <tr> <td>14/5/2014</td><td>31/08/2014</td></tr> <tr> <td>13/6/2014</td><td>31/08/2014</td></tr> <tr> <td>7/7/2014</td><td>31/10/2014</td></tr> <tr> <td>4/8/2014</td><td>31/10/2014</td></tr> </table>	InvDate	BiMthsEnd	28/03/2012	30/06/2012	10/12/2012	28/02/2013	5/2/2013	30/04/2013	31/3/2013	30/04/2013	19/5/2013	31/08/2013	15/9/2013	31/12/2013	11/12/2013	28/02/2014	2/3/2014	30/06/2014	14/5/2014	31/08/2014	13/6/2014	31/08/2014	7/7/2014	31/10/2014	4/8/2014	31/10/2014
InvDate	BiMthsEnd																										
28/03/2012	30/06/2012																										
10/12/2012	28/02/2013																										
5/2/2013	30/04/2013																										
31/3/2013	30/04/2013																										
19/5/2013	31/08/2013																										
15/9/2013	31/12/2013																										
11/12/2013	28/02/2014																										
2/3/2014	30/06/2014																										
14/5/2014	31/08/2014																										
13/6/2014	31/08/2014																										
7/7/2014	31/10/2014																										
4/8/2014	31/10/2014																										

## monthsname

此函数用于返回一个显示值，表示时段各月份(根据 **MonthNames** 脚本变量的格式)和年的范围。基础数值与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应。

### Syntax:

```
MonthsName (n_months, date[, period_no[, first_month_of_year]])
```

**Return data type:** 双

### 参数:

参数	说明
<b>n_months</b>	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一: 1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月)或 6(半年)。
<b>date</b>	要求值的日期。

参数	说明
<b>period_no</b>	该周期可通过 <b>period_no</b> 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 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几个时段, 为正数则表示随后的几个时段。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年, 可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>monthsname(4, '19/10/2013')</code>	返回 Sep-Dec 2013。 因为在此例和其他示例中, 已将 <b>SET Monthnames</b> 语句设置为 Jan;Feb;Mar, 以此类推。
<code>monthsname(4, '19/10/2013', -1)</code>	返回 May-Aug 2013。
<code>monthsname(4, '19/10/2013', 0, 2)</code>	返回 Oct-Jan 2014。 因为已将该年度指定为从 2 月开始, 因此四个月周期在下一年度的第一个月结束。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>在此例中，对于表格中的每个发票日期，根据从该年度的两个月周期中的月份范围创建月份名称。范围为 4x2 个月的偏移(通过将 period_no 指定为 4)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, MonthsName(2, InvDate, 4) AS MthsName Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 monthsname() 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>MthsName</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>Nov-Dec 2012</td></tr> <tr><td>10/12/2012</td><td>Jul-Aug 2013</td></tr> <tr><td>5/2/2013</td><td>Sep-Oct 2013</td></tr> <tr><td>31/3/2013</td><td>Nov-Dec2013</td></tr> <tr><td>19/5/2013</td><td>Jan-Feb 2014</td></tr> <tr><td>15/9/2013</td><td>May-Jun 2014</td></tr> <tr><td>11/12/2013</td><td>Jul-Aug 2014</td></tr> <tr><td>2/3/2014</td><td>Nov-Dec 2014</td></tr> <tr><td>14/5/2014</td><td>Jan-Feb 2015</td></tr> <tr><td>13/6/2014</td><td>Jan-Feb 2015</td></tr> <tr><td>7/7/2014</td><td>Mar-Apr 2015</td></tr> <tr><td>4/8/2014</td><td>Mar-Apr 2015</td></tr> </tbody> </table>	InvDate	MthsName	28/03/2012	Nov-Dec 2012	10/12/2012	Jul-Aug 2013	5/2/2013	Sep-Oct 2013	31/3/2013	Nov-Dec2013	19/5/2013	Jan-Feb 2014	15/9/2013	May-Jun 2014	11/12/2013	Jul-Aug 2014	2/3/2014	Nov-Dec 2014	14/5/2014	Jan-Feb 2015	13/6/2014	Jan-Feb 2015	7/7/2014	Mar-Apr 2015	4/8/2014	Mar-Apr 2015
InvDate	MthsName																										
28/03/2012	Nov-Dec 2012																										
10/12/2012	Jul-Aug 2013																										
5/2/2013	Sep-Oct 2013																										
31/3/2013	Nov-Dec2013																										
19/5/2013	Jan-Feb 2014																										
15/9/2013	May-Jun 2014																										
11/12/2013	Jul-Aug 2014																										
2/3/2014	Nov-Dec 2014																										
14/5/2014	Jan-Feb 2015																										
13/6/2014	Jan-Feb 2015																										
7/7/2014	Mar-Apr 2015																										
4/8/2014	Mar-Apr 2015																										

## monthsstart

此函数用于返回与包含基准日期的一个月、两个月、季度、四个月或半年的第一毫秒的时间戳对应的值。另外，它也可以用于判断上一个或下一个时间周期的时间戳。

### Syntax:

```
MonthsStart(n_months, date[, period_no [, first_month_of_year]])
```

**Return data type:** 双

### 参数:

参数	说明
<b>n_months</b>	用于定义时段的月数。整数或解算为整数的表达式必须为下列之一：1(相当于 inmonth() 函数)、2(两个月)、3(相当于 inquarter() 函数)、4(四个月)或 6(半年)。
<b>date</b>	要求值的日期。

参数	说明
<b>period_no</b>	该周期可通过 <b>period_no</b> 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 <b>base_date</b> 。 <b>period_no</b> 为负数表示前几个时段, 为正数则表示随后的几个时段。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年, 可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																										
<code>monthsstart(4, '19/10/2013')</code>	返回 1/09/2013。																										
<code>monthsstart(4, '19/10/2013', -1)</code>	返回 01/05/2013。																										
<code>monthsstart(4, '19/10/2013', 0, 2 )</code>	返回 01/10/2013。 因为该年度的开始变成 2 月。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例用于查找每个发票日期的两个月周期的第一天, 按一个两个月周期向前移动。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, MonthsStart(2, InvDate, 1) AS BiMthsStart Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 MonthsStart() 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>BiMthsStart</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>01/05/2012</td></tr> <tr><td>10/12/2012</td><td>01/01/2013</td></tr> <tr><td>5/2/2013</td><td>01/03/2013</td></tr> <tr><td>31/3/2013</td><td>01/05/2013</td></tr> <tr><td>19/5/2013</td><td>01/07/2013</td></tr> <tr><td>15/9/2013</td><td>01/11/2013</td></tr> <tr><td>11/12/2013</td><td>01/01/2014</td></tr> <tr><td>2/3/2014</td><td>01/05/2014</td></tr> <tr><td>14/5/2014</td><td>01/07/2014</td></tr> <tr><td>13/6/2014</td><td>01/07/2014</td></tr> <tr><td>7/7/2014</td><td>01/09/2014</td></tr> <tr><td>4/8/2014</td><td>01/09/2014</td></tr> </tbody> </table>	InvDate	BiMthsStart	28/03/2012	01/05/2012	10/12/2012	01/01/2013	5/2/2013	01/03/2013	31/3/2013	01/05/2013	19/5/2013	01/07/2013	15/9/2013	01/11/2013	11/12/2013	01/01/2014	2/3/2014	01/05/2014	14/5/2014	01/07/2014	13/6/2014	01/07/2014	7/7/2014	01/09/2014	4/8/2014	01/09/2014
InvDate	BiMthsStart																										
28/03/2012	01/05/2012																										
10/12/2012	01/01/2013																										
5/2/2013	01/03/2013																										
31/3/2013	01/05/2013																										
19/5/2013	01/07/2013																										
15/9/2013	01/11/2013																										
11/12/2013	01/01/2014																										
2/3/2014	01/05/2014																										
14/5/2014	01/07/2014																										
13/6/2014	01/07/2014																										
7/7/2014	01/09/2014																										
4/8/2014	01/09/2014																										

## monthstart

此函数用于返回与包含 **date** 的月份的第一天的第一毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

### Syntax:

```
MonthStart(date[, period_no])
```

Return data type: 双

### 参数:

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数，如果为 0 或忽略，表示该月包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几个月，为正数则表示随后的几个月。

### 示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>monthstart('19/10/2001')</code>	返回 01/10/2001。
<code>monthstart('19/10/2001', -1)</code>	返回 01/09/2001。



示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例用于查找表格中每个发票日期的当月的第一天，其中 base_date 按四个月移动(通过将 period_no 指定为 4)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, MonthStart(InvDate, 4) AS MthStart Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 monthstart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。</p> <table> <thead> <tr> <th>InvDate</th><th>MthStart</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>01/07/2012</td></tr> <tr><td>10/12/2012</td><td>01/04/2013</td></tr> <tr><td>5/2/2013</td><td>01/06/2013</td></tr> <tr><td>31/3/2013</td><td>01/07/2013</td></tr> <tr><td>19/5/2013</td><td>01/09/2013</td></tr> <tr><td>15/9/2013</td><td>01/01/2014</td></tr> <tr><td>11/12/2013</td><td>01/04/2014</td></tr> <tr><td>2/3/2014</td><td>01/07/2014</td></tr> <tr><td>14/5/2014</td><td>01/09/2014</td></tr> <tr><td>13/6/2014</td><td>01/10/2014</td></tr> <tr><td>7/7/2014</td><td>01/11/2014</td></tr> <tr><td>4/8/2014</td><td>01/12/2014</td></tr> </tbody> </table>	InvDate	MthStart	28/03/2012	01/07/2012	10/12/2012	01/04/2013	5/2/2013	01/06/2013	31/3/2013	01/07/2013	19/5/2013	01/09/2013	15/9/2013	01/01/2014	11/12/2013	01/04/2014	2/3/2014	01/07/2014	14/5/2014	01/09/2014	13/6/2014	01/10/2014	7/7/2014	01/11/2014	4/8/2014	01/12/2014
InvDate	MthStart																										
28/03/2012	01/07/2012																										
10/12/2012	01/04/2013																										
5/2/2013	01/06/2013																										
31/3/2013	01/07/2013																										
19/5/2013	01/09/2013																										
15/9/2013	01/01/2014																										
11/12/2013	01/04/2014																										
2/3/2014	01/07/2014																										
14/5/2014	01/09/2014																										
13/6/2014	01/10/2014																										
7/7/2014	01/11/2014																										
4/8/2014	01/12/2014																										

## networkdays

**networkdays** 函数用于返回工作日的编号(周一至周五)，在 **start\_date** 和 **end\_date** 之间，并将任何列出的可选 **holiday** 考虑在内。

### Syntax:

```
networkdays (start_date, end_date [, holiday])
```

**Return data type:** 整数

### 参数:

参数	说明
<b>start_date</b>	要求值的开始日期的时间戳。
<b>end_date</b>	要求值的结束日期的时间戳。

参数	说明
<b>holiday</b>	<p>从工作日排除假期。假期可表述为开始日期和结束日期，以逗号分隔。</p> <p><b>示例：</b> '25/12/2013', '26/12/2013'</p> <p>您可以排除多个假期，以逗号分隔。</p> <p><b>示例：</b> '25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014'</p>

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																																																				
networkdays ('19/12/2013', '07/01/2014')	返回 14。以下示例没有将假期考虑在内。																																																				
networkdays ('19/12/2013', '07/01/2014', '25/12/2013', '26/12/2013')	返回 12。以下示例将 25/12/2013 至 26/12/2013 的假期考虑在内。																																																				
networkdays ('19/12/2013', '07/01/2014', '25/12/2013', '26/12/2013', '31/12/2013', '01/01/2014')	返回 10。以下示例将两个假期考虑在内。																																																				
添加示例脚本到应用程序并运行。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。	结果列表显示了为表格中的每条记录返回的 NetworkDays 值。																																																				
PayTable: LOAD *, recno() as InvID, INLINE [ InvRec InvPaid 28/03/2012 28/04/2012 10/12/2012 01/01/2013 5/2/2013 5/3/2013 31/3/2013 01/5/2013 19/5/2013 12/6/2013 15/9/2013 6/10/2013 11/12/2013 12/01/2014 2/3/2014 2/4/2014 14/5/2014 14/6/2014 13/6/2014 14/7/2014 7/7/2014 14/8/2014 4/8/2014 4/9/2014 ] (delimiter is ' '); NrDays: Load *, NetworkDays(InvRec,InvPaid) As PaidDays Resident PayTable; Drop table PayTable;	<table><tr><th>InvID</th><th>InvRec</th><th>InvPaid</th><th>PaidDays</th></tr><tr><td>1</td><td>28/03/2012</td><td>28/04/2012</td><td>23</td></tr><tr><td>2</td><td>10/12/2012</td><td>01/01/2013</td><td>17</td></tr><tr><td>3</td><td>5/2/2013</td><td>5/3/2013</td><td>21</td></tr><tr><td>4</td><td>31/3/2013</td><td>01/5/2013</td><td>23</td></tr><tr><td>5</td><td>19/5/2013</td><td>12/6/2013</td><td>18</td></tr><tr><td>6</td><td>15/9/2013</td><td>6/10/2013</td><td>15</td></tr><tr><td>7</td><td>11/12/2013</td><td>12/01/2014</td><td>23</td></tr><tr><td>8</td><td>2/3/2014</td><td>2/4/2014</td><td>23</td></tr><tr><td>9</td><td>14/5/2014</td><td>14/6/2014</td><td>23</td></tr><tr><td>10</td><td>13/6/2014</td><td>14/7/2014</td><td>22</td></tr><tr><td>11</td><td>7/7/2014</td><td>14/8/2014</td><td>29</td></tr><tr><td>12</td><td>4/8/2014</td><td>4/9/2014</td><td>24</td></tr></table>	InvID	InvRec	InvPaid	PaidDays	1	28/03/2012	28/04/2012	23	2	10/12/2012	01/01/2013	17	3	5/2/2013	5/3/2013	21	4	31/3/2013	01/5/2013	23	5	19/5/2013	12/6/2013	18	6	15/9/2013	6/10/2013	15	7	11/12/2013	12/01/2014	23	8	2/3/2014	2/4/2014	23	9	14/5/2014	14/6/2014	23	10	13/6/2014	14/7/2014	22	11	7/7/2014	14/8/2014	29	12	4/8/2014	4/9/2014	24
InvID	InvRec	InvPaid	PaidDays																																																		
1	28/03/2012	28/04/2012	23																																																		
2	10/12/2012	01/01/2013	17																																																		
3	5/2/2013	5/3/2013	21																																																		
4	31/3/2013	01/5/2013	23																																																		
5	19/5/2013	12/6/2013	18																																																		
6	15/9/2013	6/10/2013	15																																																		
7	11/12/2013	12/01/2014	23																																																		
8	2/3/2014	2/4/2014	23																																																		
9	14/5/2014	14/6/2014	23																																																		
10	13/6/2014	14/7/2014	22																																																		
11	7/7/2014	14/8/2014	29																																																		
12	4/8/2014	4/9/2014	24																																																		

## now

此函数用于返回系统时钟的当前时间的时间戳。

**Syntax:**

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

**Return data type:** 双**参数:**

参数	说明
timer_mode	<p>可以具有以下值:</p> <p>0(最后完成数据加载的时间)</p> <p>1(函数调用的时间)</p> <p>2(应用程序打开的时间)</p>



如果在数据加载脚本中使用此函数,则 **timer\_mode=0** 将会生成最后完成数据加载的时间,而 **timer\_mode=1** 将会提供当前数据加载的函数调用时间。

**示例和结果:**

示例	结果
now( 0)	返回最后数据加载完成的时间。
now( 1)	<ul style="list-style-type: none"> <li>• 当在可视化表达式中使用时,此函数返回函数调用的时间。</li> <li>• 当在数据加载脚本中使用时,此函数返回当前数据加载的函数调用时间。</li> </ul>
now( 2)	返回应用程序打开的时间。

**quarterend**

此函数用于返回与包含 **date** 的季度的最后毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

**Syntax:**

```
QuarterEnd(date[, period_no[, first_month_of_year]])
```

**Return data type:** 双**参数:**

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数，其中值 0 表示该季度包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几季，为正数则表示随后的几季。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																										
<code>quarterend('29/10/2005')</code>	返回 31/12/2005 23:59:59。																										
<code>quarterend('29/10/2005', -1)</code>	返回 30/09/2005 23:59:59。																										
<code>quarterend('29/10/2005', 0, 3)</code>	返回 30/11/2005 23:59:59。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例用于查找表格中每个发票日期的季度的第一天，其中将该年度的第一个月指定为 3 月。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, QuarterEnd(InvDate, 0, 3) AS QtrEnd Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>quarterend()</code> 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。</p> <table> <thead> <tr> <th>InvDate</th><th>QtrEnd</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>31/05/2012</td></tr> <tr><td>10/12/2012</td><td>28/02/2013</td></tr> <tr><td>5/2/2013</td><td>28/02/2013</td></tr> <tr><td>31/3/2013</td><td>31/05/2013</td></tr> <tr><td>19/5/2013</td><td>31/05/2013</td></tr> <tr><td>15/9/2013</td><td>30/11/2013</td></tr> <tr><td>11/12/2013</td><td>28/02/2014</td></tr> <tr><td>2/3/2014</td><td>31/05/2014</td></tr> <tr><td>14/5/2014</td><td>31/05/2014</td></tr> <tr><td>13/6/2014</td><td>31/08/2014</td></tr> <tr><td>7/7/2014</td><td>31/08/2014</td></tr> <tr><td>4/8/2014</td><td>31/08/2014</td></tr> </tbody> </table>	InvDate	QtrEnd	28/03/2012	31/05/2012	10/12/2012	28/02/2013	5/2/2013	28/02/2013	31/3/2013	31/05/2013	19/5/2013	31/05/2013	15/9/2013	30/11/2013	11/12/2013	28/02/2014	2/3/2014	31/05/2014	14/5/2014	31/05/2014	13/6/2014	31/08/2014	7/7/2014	31/08/2014	4/8/2014	31/08/2014
InvDate	QtrEnd																										
28/03/2012	31/05/2012																										
10/12/2012	28/02/2013																										
5/2/2013	28/02/2013																										
31/3/2013	31/05/2013																										
19/5/2013	31/05/2013																										
15/9/2013	30/11/2013																										
11/12/2013	28/02/2014																										
2/3/2014	31/05/2014																										
14/5/2014	31/05/2014																										
13/6/2014	31/08/2014																										
7/7/2014	31/08/2014																										
4/8/2014	31/08/2014																										

## quartername

此函数用于返回一个显示值，该值显示季度的月(根据 **MonthNames** 脚本变量的格式)以及年，伴随一个与该季度第一天第一毫秒的时间戳对应的基础数值。

### Syntax:

```
QuarterName(date[, period_no[, first_month_of_year]])
```

Return data type: 双

### 参数:

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数，其中值 0 表示该季度包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几季，为正数则表示随后的几季。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

### 示例和结果:

示例	结果
quartername('29/10/2013')	返回 Oct-Dec 2013。
quartername('29/10/2013', -1)	返回 Jul-Sep 2013。
quartername('29/10/2013', 0, 3)	返回 Sep-Nov 2013。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>在此例中，对于表格中的每个发票日期，根据从一年的 base_date 获取的月份名称创建季度名称，其中将一年的第一个月指定为 4 月。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, QuarterName(InvDate, 0, 4) AS QtrName Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 quartername() 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>QtrName</th></tr> </thead> <tbody> <tr> <td>28/03/2012</td><td>Jan-Mar 2011 因为这反映了从 2011 年开始的财政年。</td></tr> <tr> <td>10/12/2012</td><td>Oct-Dec 2012</td></tr> <tr> <td>5/2/2013</td><td>Jan-Mar 2012</td></tr> <tr> <td>31/3/2013</td><td>Jan-Mar 2012</td></tr> <tr> <td>19/5/2013</td><td>Apr-Jun 2013</td></tr> <tr> <td>15/9/2013</td><td>Jul-Sep 2013</td></tr> <tr> <td>11/12/2013</td><td>Oct-Dec 2013</td></tr> <tr> <td>2/3/2014</td><td>Jan-Mar 2013</td></tr> <tr> <td>14/5/2014</td><td>Apr-Jun 2014</td></tr> <tr> <td>13/6/2014</td><td>Apr-Jun 2014</td></tr> <tr> <td>7/7/2014</td><td>Jul-Sep 2014</td></tr> <tr> <td>4/8/2014</td><td>Jul-Sep 2014</td></tr> </tbody> </table>	InvDate	QtrName	28/03/2012	Jan-Mar 2011 因为这反映了从 2011 年开始的财政年。	10/12/2012	Oct-Dec 2012	5/2/2013	Jan-Mar 2012	31/3/2013	Jan-Mar 2012	19/5/2013	Apr-Jun 2013	15/9/2013	Jul-Sep 2013	11/12/2013	Oct-Dec 2013	2/3/2014	Jan-Mar 2013	14/5/2014	Apr-Jun 2014	13/6/2014	Apr-Jun 2014	7/7/2014	Jul-Sep 2014	4/8/2014	Jul-Sep 2014
InvDate	QtrName																										
28/03/2012	Jan-Mar 2011 因为这反映了从 2011 年开始的财政年。																										
10/12/2012	Oct-Dec 2012																										
5/2/2013	Jan-Mar 2012																										
31/3/2013	Jan-Mar 2012																										
19/5/2013	Apr-Jun 2013																										
15/9/2013	Jul-Sep 2013																										
11/12/2013	Oct-Dec 2013																										
2/3/2014	Jan-Mar 2013																										
14/5/2014	Apr-Jun 2014																										
13/6/2014	Apr-Jun 2014																										
7/7/2014	Jul-Sep 2014																										
4/8/2014	Jul-Sep 2014																										

## quarterstart

此函数用于返回与包含 **date** 的季度的第一毫秒的时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

### Syntax:

```
QuarterStart(date[, period_no[, first_month_of_yea]])
```

**Return data type:** 双

### 参数:

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数，其中值 0 表示该季度包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几季，为正数则表示随后的几季。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																										
<code>quarterstart('29/10/2005')</code>	返回 01/10/2005。																										
<code>quarterstart('29/10/2005', -1 )</code>	返回 01/07/2005。																										
<code>quarterstart('29/10/2005', 0, 3)</code>	返回 01/09/2005。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例用于查找表格中每个发票日期的季度的第一天，其中将该年度的第一个月指定为 3 月。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, QuarterStart(InvDate, 0, 3) AS QtrStart Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>quarterstart()</code> 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。</p> <table> <thead> <tr> <th>InvDate</th><th>QtrStart</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>01/03/2012</td></tr> <tr><td>10/12/2012</td><td>01/12/2012</td></tr> <tr><td>5/2/2013</td><td>01/12/2012</td></tr> <tr><td>31/3/2013</td><td>01/03/2013</td></tr> <tr><td>19/5/2013</td><td>01/03/2013</td></tr> <tr><td>15/9/2013</td><td>01/09/2013</td></tr> <tr><td>11/12/2013</td><td>01/12/2013</td></tr> <tr><td>2/3/2014</td><td>01/03/2014</td></tr> <tr><td>14/5/2014</td><td>01/03/2014</td></tr> <tr><td>13/6/2014</td><td>01/06/2014</td></tr> <tr><td>7/7/2014</td><td>01/06/2014</td></tr> <tr><td>4/8/2014</td><td>01/06/2014</td></tr> </tbody> </table>	InvDate	QtrStart	28/03/2012	01/03/2012	10/12/2012	01/12/2012	5/2/2013	01/12/2012	31/3/2013	01/03/2013	19/5/2013	01/03/2013	15/9/2013	01/09/2013	11/12/2013	01/12/2013	2/3/2014	01/03/2014	14/5/2014	01/03/2014	13/6/2014	01/06/2014	7/7/2014	01/06/2014	4/8/2014	01/06/2014
InvDate	QtrStart																										
28/03/2012	01/03/2012																										
10/12/2012	01/12/2012																										
5/2/2013	01/12/2012																										
31/3/2013	01/03/2013																										
19/5/2013	01/03/2013																										
15/9/2013	01/09/2013																										
11/12/2013	01/12/2013																										
2/3/2014	01/03/2014																										
14/5/2014	01/03/2014																										
13/6/2014	01/06/2014																										
7/7/2014	01/06/2014																										
4/8/2014	01/06/2014																										

## second

此函数用于根据标准数字解释当 **expression** 小数部分被解释为时间时返回一个表示秒的整数。

**Syntax:**

```
second (expression)
```

**Return data type:** 整数

示例和结果：

示例	结果
<code>second( '09:14:36' )</code>	返回 36
<code>second( '0.5555' )</code>	返回 55( 因为 0.5555 = 13:19:55)

## setdateyear

此函数用于返回基于输入 **timestamp** 但将年份替换为 **year** 的时间戳。

**Syntax:**

```
setdateyear (timestamp, year)
```

**Return data type:** 双

参数：

参数	说明
<b>timestamp</b>	标准的 Qlik Sense 时间戳( 通常只是一个日期)。
<b>year</b>	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。

示例	结果
<code>setdateyear ( '29/10/2005', 2013)</code>	返回“29/102013”
<code>setdateyear ( '29/10/2005 04:26', 20013)</code>	返回“29//10/20013 04:26” 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	结果	
添加示例脚本到应用程序并运行。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。	结果列表包含原始日期和在其中已将年份设置为 2013 的列。	
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 ];	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

用于返回基于输入 **timestamp**，但将年份替换为 **year** 和将月份替换为 **month** 的时间戳。

### Syntax:

```
SetDateYearMonth (timestamp, year, month)
```

**Return data type:** 双

### 参数:

参数	说明
<b>timestamp</b>	标准的 Qlik Sense 时间戳(通常只是一个日期)。
<b>year</b>	一个四位数年份。
<b>month</b>	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。

示例	结果																										
<code>setdateyearmonth ('29/10/2005', 2013, 3)</code>	返回“29/03/2013”																										
<code>setdateyearmonth ('29/10/2005 04:26', 2013, 3)</code>	返回“29/03/2013 04:26” 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格式。																										
<p>添加示例脚本到应用程序并运行。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。</p> <pre> SetYearMonth: Load *, SetDateYearMonth(testdates, 2013) as NewYearMonth 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 ]; </pre>	<p>结果列表包含原始日期和在其中已将年份设置为 2013 的列。</p> <table> <thead> <tr> <th>testdates</th><th>NewYearMonth</th></tr> </thead> <tbody> <tr><td>1/11/2012</td><td>1//3/2013</td></tr> <tr><td>10/12/2012</td><td>10/3/2013</td></tr> <tr><td>2/1/2012</td><td>2/3/2013</td></tr> <tr><td>1/5/2013</td><td>1/3/2013</td></tr> <tr><td>19/5/2013</td><td>19/3/2013</td></tr> <tr><td>15/9/2013</td><td>15/3/2013</td></tr> <tr><td>11/12/2013</td><td>11/3/2013</td></tr> <tr><td>2/3/2014</td><td>2/3/2013</td></tr> <tr><td>14/5/2014</td><td>14/3/2013</td></tr> <tr><td>13/6/2014</td><td>13/3/2013</td></tr> <tr><td>7/7/2014</td><td>7/3/2013</td></tr> <tr><td>4/8/2014</td><td>4/3/2013</td></tr> </tbody> </table>	testdates	NewYearMonth	1/11/2012	1//3/2013	10/12/2012	10/3/2013	2/1/2012	2/3/2013	1/5/2013	1/3/2013	19/5/2013	19/3/2013	15/9/2013	15/3/2013	11/12/2013	11/3/2013	2/3/2014	2/3/2013	14/5/2014	14/3/2013	13/6/2014	13/3/2013	7/7/2014	7/3/2013	4/8/2014	4/3/2013
testdates	NewYearMonth																										
1/11/2012	1//3/2013																										
10/12/2012	10/3/2013																										
2/1/2012	2/3/2013																										
1/5/2013	1/3/2013																										
19/5/2013	19/3/2013																										
15/9/2013	15/3/2013																										
11/12/2013	11/3/2013																										
2/3/2014	2/3/2013																										
14/5/2014	14/3/2013																										
13/6/2014	13/3/2013																										
7/7/2014	7/3/2013																										
4/8/2014	4/3/2013																										

### timezone

此函数用于返回当前时区的名称，如 Windows 所定义。

#### Syntax:

**TimeZone ( )**

**Return data type:** 字符串

#### 示例:

`timezone( )`

### today


此函数用于返回系统时钟的当前日期。

#### Syntax:

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

**Return data type:** 双

**参数:**

参数	说明
timer_mode	<p>可以具有以下值:</p> <p>0( 最后完成数据加载的日期)</p> <p>1( 函数调用的日期)</p> <p>2( 应用程序打开的日期)</p>
<p> 如果在数据加载脚本中使用此函数, 则 <b>timer_mode=0</b> 将会生成最后完成数据加载的日期, 而 <b>timer_mode=1</b> 将会提供当前数据加载的日期。</p>	

**示例和结果:**

示例	结果
Today( 0)	返回最后完成数据加载的日期。
Today( 1)	<p>当在可视化表达式中使用, 此函数返回函数调用的日期。</p> <p>当在数据加载脚本中使用, 此函数返回当前数据加载开始时的日期。</p>
Today( 2)	返回应用程序打开的日期。

## UTC

用于返回当前 Coordinated Universal Time。

**Syntax:**

```
UTC ( )
```

**Return data type:** 双

**示例:**

```
utc( )
```

## week

此函数用于返回根据 ISO 8601 表示周数的整数。周数根据标准数字解释通过表达式的日期解释进行计算。

**Syntax:**

```
week (expression)
```

**Return data type:** 整数

示例和结果：

示例	结果
week( '2012-10-12' )	使用不连续周的默认设置(已禁用)时，返回 41。  如果将 <b>BrokenWeeks</b> 变量设置为 1(已启用)，则本例将返回 42。  另请： <i>BrokenWeeks</i> (第 118 页)
week( '35648' )	返回 32, 因为 35648 = 1997-08-06

## 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。

**Syntax:**

```
weekday (date [, weekstart=0])
```

**Return data type:** 双

**参数：**

参数	说明
<b>date</b>	要求值的日期。
<b>weekstart</b>	<p>如果不指定 <b>weekstart</b>, 则变量 <b>FirstWeekDay</b> 的值将用作一周的第一天。</p> <p>如果要使用其他天作为一周的第一天, 请将 <b>weekstart</b> 设置为：</p> <ul style="list-style-type: none"> <li>• 0, 表示周一</li> <li>• 1, 表示周二</li> <li>• 2, 表示周三</li> <li>• 3, 表示周四</li> <li>• 4, 表示周五</li> <li>• 5, 表示周六</li> <li>• 6, 表示周日</li> </ul> <p>此函数返回的整数现在将使用您使用 <b>weekstart</b> 设置的一周的第一天作为基数 (0)。</p> <p>另请： <i>FirstWeekDay</i>(第 119 页)</p>

示例和结果：

在以下示例中，**FirstWeekDay** 设置为 0(除非另有说明)。

示例	结果
<code>weekday( '1971-10-12' )</code>	返回“Tue”和 1
<code>weekday( '1971-10-12' , 6)</code>	返回“Tue”和 2。  在此示例中，我们使用 Sunday (6) 作为一周的第一天。
<code>SET FirstWeekDay = 6;</code>  ... <code>weekday( '1971-10-12')</code>	返回“Tue”和 2。

## weekend

此函数用于返回一个与包含 **date** 的日历周的最后一日(周日)最后一毫秒时间戳对应的值。默认输出格式为在脚本中设置的 **DateFormat**。

**Syntax:**

```
WeekEnd(date [, period_no[, first_week_day]])
```

**Return data type:** 双

**参数：**

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>shift</b> 为整数，其中值 0 表示该星期包含 <b>date</b> 。shift 为负数，表示前几星期，正数表示随后的几星期。
<b>first_week_day</b>	指定一周的开始日期。如果忽略，使用 <b>FirstWeekDay</b> 变量的值。  可能的 <b>first_week_day</b> 值为： <ul style="list-style-type: none"> <li>• 0, 表示周一</li> <li>• 1, 表示周二</li> <li>• 2, 表示周三</li> <li>• 3, 表示周四</li> <li>• 4, 表示周五</li> <li>• 5, 表示周六</li> <li>• 6, 表示周日</li> </ul> 另请： <i>FirstWeekDay</i> ( 第 119 页)

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																										
<code>weekend('10/01/2013')</code>	返回 12/01/2013 23:59:59。																										
<code>weekend('10/01/2013', -1)</code>	返回 06/01/2013 23:59:59。																										
<code>weekend('10/01/2013', 0, 1)</code>	返回 14/01/2013 23:59:59。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找后跟每个发票日期的周的周中的最后一天。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, WeekEnd(InvDate, 1) AS WkEnd Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>weekend()</code> 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。</p> <table> <thead> <tr> <th>InvDate</th><th>WkEnd</th></tr> </thead> <tbody> <tr> <td>28/03/2012</td><td>08/04/2012</td></tr> <tr> <td>10/12/2012</td><td>23/12/2012</td></tr> <tr> <td>5/2/2013</td><td>17/02/2013</td></tr> <tr> <td>31/3/2013</td><td>07/04/2013</td></tr> <tr> <td>19/5/2013</td><td>26/05/2013</td></tr> <tr> <td>15/9/2013</td><td>22/09/2013</td></tr> <tr> <td>11/12/2013</td><td>22/12/2013</td></tr> <tr> <td>2/3/2014</td><td>09/03/2014</td></tr> <tr> <td>14/5/2014</td><td>25/05/2014</td></tr> <tr> <td>13/6/2014</td><td>22/06/2014</td></tr> <tr> <td>7/7/2014</td><td>20/07/2014</td></tr> <tr> <td>4/8/2014</td><td>17/08/2014</td></tr> </tbody> </table>	InvDate	WkEnd	28/03/2012	08/04/2012	10/12/2012	23/12/2012	5/2/2013	17/02/2013	31/3/2013	07/04/2013	19/5/2013	26/05/2013	15/9/2013	22/09/2013	11/12/2013	22/12/2013	2/3/2014	09/03/2014	14/5/2014	25/05/2014	13/6/2014	22/06/2014	7/7/2014	20/07/2014	4/8/2014	17/08/2014
InvDate	WkEnd																										
28/03/2012	08/04/2012																										
10/12/2012	23/12/2012																										
5/2/2013	17/02/2013																										
31/3/2013	07/04/2013																										
19/5/2013	26/05/2013																										
15/9/2013	22/09/2013																										
11/12/2013	22/12/2013																										
2/3/2014	09/03/2014																										
14/5/2014	25/05/2014																										
13/6/2014	22/06/2014																										
7/7/2014	20/07/2014																										
4/8/2014	17/08/2014																										

### weekname

此函数用于返回一个值，显示带有与包含 **date** 的周的第一天的第一毫秒时间戳对应的基本数值对应的年份和周数。

#### Syntax:

```
WeekName (date[, period_no[, first_week_day]])
```

**Return data type:** 双

参数：

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>shift</b> 为整数，其中值 0 表示该星期包含 <b>date</b> 。shift 为负数，表示前几星期，正数表示随后的几星期。
<b>first_week_day</b>	<p>指定一周的开始日期。如果忽略，使用 <b>FirstWeekDay</b> 变量的值。</p> <p>可能的 <b>first_week_day</b> 值为：</p> <ul style="list-style-type: none"><li>• 0, 表示周一</li><li>• 1, 表示周二</li><li>• 2, 表示周三</li><li>• 3, 表示周四</li><li>• 4, 表示周五</li><li>• 5, 表示周六</li><li>• 6, 表示周日</li></ul> <p>另请： <i>FirstWeekDay</i>( 第 119 页)</p>

示例和结果：

示例	结果
<code>weekname('12/01/2013')</code>	返回 2013/02。
<code>weekname('12/01/2013', -1)</code>	返回 2013/01。
<code>weekname('12/01/2013', 0, 1)</code>	返回 2013/02。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>在此例中，对于表格中的每个发票日期，根据周所在的年度创建周名称，并将其与周编号进行相关联(通过将 period_no 指定为 1 按一周移动)。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, WeekName(InvDate, 1) AS WkName Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 weekname() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。</p> <table> <thead> <tr> <th>InvDate</th><th>WkName</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>2012/14</td></tr> <tr><td>10/12/2012</td><td>2012/51</td></tr> <tr><td>5/2/2013</td><td>2013/07</td></tr> <tr><td>31/3/2013</td><td>2013/14</td></tr> <tr><td>19/5/2013</td><td>2013/21</td></tr> <tr><td>15/9/2013</td><td>2013/38</td></tr> <tr><td>11/12/2013</td><td>2013/51</td></tr> <tr><td>2/3/2014</td><td>2014/10</td></tr> <tr><td>14/5/2014</td><td>2014/21</td></tr> <tr><td>13/6/2014</td><td>2014/25</td></tr> <tr><td>7/7/2014</td><td>2014/29</td></tr> <tr><td>4/8/2014</td><td>2014/33</td></tr> </tbody> </table>	InvDate	WkName	28/03/2012	2012/14	10/12/2012	2012/51	5/2/2013	2013/07	31/3/2013	2013/14	19/5/2013	2013/21	15/9/2013	2013/38	11/12/2013	2013/51	2/3/2014	2014/10	14/5/2014	2014/21	13/6/2014	2014/25	7/7/2014	2014/29	4/8/2014	2014/33
InvDate	WkName																										
28/03/2012	2012/14																										
10/12/2012	2012/51																										
5/2/2013	2013/07																										
31/3/2013	2013/14																										
19/5/2013	2013/21																										
15/9/2013	2013/38																										
11/12/2013	2013/51																										
2/3/2014	2014/10																										
14/5/2014	2014/21																										
13/6/2014	2014/25																										
7/7/2014	2014/29																										
4/8/2014	2014/33																										

## weekstart

此函数用于返回与包含 **date** 的日历周的第一天(周一)的第一毫秒时间戳对应的值。默认输出格式是在脚本中设置的 **DateFormat**。

### Syntax:

```
WeekStart(date [, period_no[, first_week_day]])
```

**Return data type:** 双

**参数:**

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>shift</b> 为整数，其中值 0 表示该星期包含 <b>date</b> 。shift 为负数，表示前几星期，正数表示随后的几星期。



参数	说明
<b>first_week_day</b>	<p>指定一周的开始日期。如果忽略, 使用 <b>FirstWeekDay</b> 变量的值。</p> <p>可能的 <b>first_week_day</b> 值为:</p> <ul style="list-style-type: none"><li>• 0, 表示周一</li><li>• 1, 表示周二</li><li>• 2, 表示周三</li><li>• 3, 表示周四</li><li>• 4, 表示周五</li><li>• 5, 表示周六</li><li>• 6, 表示周日</li></ul> <p>另请: <i>FirstWeekDay</i>( 第 119 页)</p>

示例和结果:

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
<code>weekstart('12/01/2013')</code>	返回 07/01/2013。
<code>weekstart('12/01/2013', -1 )</code>	返回 31/11/2012。
<code>weekstart('12/01/2013', 0, 1)</code>	返回 08/01/2013。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找后跟每个发票日期的周的第一天。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, WeekStart(InvDate, 1) AS WkStart Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 weekstart() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。</p> <table> <thead> <tr> <th>InvDate</th><th>WkStart</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>02/04/2012</td></tr> <tr><td>10/12/2012</td><td>17/12/2012</td></tr> <tr><td>5/2/2013</td><td>11/02/2013</td></tr> <tr><td>31/3/2013</td><td>01/04/2013</td></tr> <tr><td>19/5/2013</td><td>20/05/2013</td></tr> <tr><td>15/9/2013</td><td>16/09/2013</td></tr> <tr><td>11/12/2013</td><td>16/12/2013</td></tr> <tr><td>2/3/2014</td><td>03/03/2014</td></tr> <tr><td>14/5/2014</td><td>19/05/2014</td></tr> <tr><td>13/6/2014</td><td>16/06/2014</td></tr> <tr><td>7/7/2014</td><td>14/07/2014</td></tr> <tr><td>4/8/2014</td><td>11/08/2014</td></tr> </tbody> </table>	InvDate	WkStart	28/03/2012	02/04/2012	10/12/2012	17/12/2012	5/2/2013	11/02/2013	31/3/2013	01/04/2013	19/5/2013	20/05/2013	15/9/2013	16/09/2013	11/12/2013	16/12/2013	2/3/2014	03/03/2014	14/5/2014	19/05/2014	13/6/2014	16/06/2014	7/7/2014	14/07/2014	4/8/2014	11/08/2014
InvDate	WkStart																										
28/03/2012	02/04/2012																										
10/12/2012	17/12/2012																										
5/2/2013	11/02/2013																										
31/3/2013	01/04/2013																										
19/5/2013	20/05/2013																										
15/9/2013	16/09/2013																										
11/12/2013	16/12/2013																										
2/3/2014	03/03/2014																										
14/5/2014	19/05/2014																										
13/6/2014	16/06/2014																										
7/7/2014	14/07/2014																										
4/8/2014	11/08/2014																										

## weekyear

此函数用于返回根据 ISO 8601 周数所属的年份。星期数范围在 1 和大约 52 之间。

### Syntax:

**weekyear** (expression)

**Return data type:** 整数

示例和结果：

示例	结果
weekyear( '1996-12-30' )	返回 1997, 因为 1998 的第 1 周从 1996/12/30 星期一开始
weekyear( '1997-01-02' )	返回 1997
weekyear( '1997-12-28' )	返回 1997

示例	结果
<code>weekyear( '1997-12-30' )</code>	返回 1998, 因为 1998 的第 1 周从 1997/12/29 星期一 开始
<code>weekyear( '1999-01-02' )</code>	返回 1998, 因为 1998 的第 53 周从 1999-01-03 开始

**限制:**

部分年份第 1 周始于十二月, 例如, 1997 年 12 月。其他年份始于上一年的第 53 周, 例如 1999 年 1 月。对于少数日子, 相应星期序数可能属于其他年份, 此时函数 **year** 和 **weekyear** 将返回不同的值。

**year**

此函数用于根据标准数字解释当 **expression** 被解释为日期时返回一个表示年份的整数。

**Syntax:**

```
year (expression)
```

**Return data type:** 整数

示例和结果:

示例	结果
<code>year( '2012-10-12' )</code>	返回 2012
<code>year( '35648' )</code>	返回 1997, 因为 35648 = 1997-08-06

**yearend**

此函数用于返回与包含 **date** 的年份的最后一天的最后毫秒时间戳对应的值。默认的输出格式为在脚本中所设置的 **DateFormat**。

**Syntax:**

```
YearEnd( date[, period_no[, first_month_of_year = 1]])
```

**Return data type:** 双

**参数:**

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数, 其中值 0 表示该年份包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几年, 为正数表示随后几年。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年, 可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																										
yearend ( '19/10/2001' )	返回 31/12/2001 23:59:59。																										
yearend ( '19/10/2001', -1 )	返回 31/12/2000 23:59:59。																										
yearend ( '19/10/2001', 0, 4)	返回 31/03/2002 23:59:59。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找每个发票日期的年份的最后一天，其中将该年度的第一个月指定为 4 月。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, YearEnd(InvDate, 0, 4) AS YrEnd Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 yearend() 函数的返回值的列。您可以通过在属性面板中指定格式来显示完整的时间戳。</p> <table> <thead> <tr> <th>InvDate</th><th>YrEnd</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>31/03/2011</td></tr> <tr><td>10/12/2012</td><td>31/03/2012</td></tr> <tr><td>5/2/2013</td><td>31/03/2013</td></tr> <tr><td>31/3/2013</td><td>31/03/2013</td></tr> <tr><td>19/5/2013</td><td>31/03/2014</td></tr> <tr><td>15/9/2013</td><td>31/03/2014</td></tr> <tr><td>11/12/2013</td><td>31/03/2014</td></tr> <tr><td>2/3/2014</td><td>31/03/2014</td></tr> <tr><td>14/5/2014</td><td>31/03/2015</td></tr> <tr><td>13/6/2014</td><td>31/03/2015</td></tr> <tr><td>7/7/2014</td><td>31/03/2015</td></tr> <tr><td>4/8/2014</td><td>31/03/2015</td></tr> </tbody> </table>	InvDate	YrEnd	28/03/2012	31/03/2011	10/12/2012	31/03/2012	5/2/2013	31/03/2013	31/3/2013	31/03/2013	19/5/2013	31/03/2014	15/9/2013	31/03/2014	11/12/2013	31/03/2014	2/3/2014	31/03/2014	14/5/2014	31/03/2015	13/6/2014	31/03/2015	7/7/2014	31/03/2015	4/8/2014	31/03/2015
InvDate	YrEnd																										
28/03/2012	31/03/2011																										
10/12/2012	31/03/2012																										
5/2/2013	31/03/2013																										
31/3/2013	31/03/2013																										
19/5/2013	31/03/2014																										
15/9/2013	31/03/2014																										
11/12/2013	31/03/2014																										
2/3/2014	31/03/2014																										
14/5/2014	31/03/2015																										
13/6/2014	31/03/2015																										
7/7/2014	31/03/2015																										
4/8/2014	31/03/2015																										

### yearname

此函数用于返回一个四位数年份的显示值，带有与包含 **date** 的年份的第一天的第一毫秒时间戳对应的基本数值。

**Syntax:**

```
YearName (date[, period_no[, first_month_of_year]] )
```

**Return data type:** 双

**参数:**

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数, 其中值 0 表示该年份包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几年, 为正数表示随后几年。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年, 可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。该显示值将为一个字符串, 表示两年。

**示例和结果:**

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果
yearname ( '19/10/2001' )	返回 2001。
yearname ( '19/10/2001', -1 )	返回 2000。
yearname ( '19/10/2001', 0, 4 )	返回 2001-2002。

示例	结果																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例为在表格中找到的每个发票日期的年份创建四加四位数的名称。这是因为已将该年度的第一个月指定为 4 月。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, YearName(InvDate, 0, 4) AS YrName Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 yearname() 函数的返回值的列。</p> <table> <thead> <tr> <th>InvDate</th><th>YrName</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>2011-2012</td></tr> <tr><td>10/12/2012</td><td>2012-2013</td></tr> <tr><td>5/2/2013</td><td>2012-2013</td></tr> <tr><td>31/3/2013</td><td>2012-2013</td></tr> <tr><td>19/5/2013</td><td>2013-2014</td></tr> <tr><td>15/9/2013</td><td>2013-2014</td></tr> <tr><td>11/12/2013</td><td>2013-2014</td></tr> <tr><td>2/3/2014</td><td>2013-2014</td></tr> <tr><td>14/5/2014</td><td>2014-2015</td></tr> <tr><td>13/6/2014</td><td>2014-2015</td></tr> <tr><td>7/7/2014</td><td>2014-2015</td></tr> <tr><td>4/8/2014</td><td>2014-2015</td></tr> </tbody> </table>	InvDate	YrName	28/03/2012	2011-2012	10/12/2012	2012-2013	5/2/2013	2012-2013	31/3/2013	2012-2013	19/5/2013	2013-2014	15/9/2013	2013-2014	11/12/2013	2013-2014	2/3/2014	2013-2014	14/5/2014	2014-2015	13/6/2014	2014-2015	7/7/2014	2014-2015	4/8/2014	2014-2015
InvDate	YrName																										
28/03/2012	2011-2012																										
10/12/2012	2012-2013																										
5/2/2013	2012-2013																										
31/3/2013	2012-2013																										
19/5/2013	2013-2014																										
15/9/2013	2013-2014																										
11/12/2013	2013-2014																										
2/3/2014	2013-2014																										
14/5/2014	2014-2015																										
13/6/2014	2014-2015																										
7/7/2014	2014-2015																										
4/8/2014	2014-2015																										

## yearstart

此函数用于返回与包含 **date** 的年份的第一天的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**。

### Syntax:

```
YearStart(date[, period_no[, first_month_of_year]])
```

**Return data type:** 双

参数：

参数	说明
<b>date</b>	要求值的日期。
<b>period_no</b>	<b>period_no</b> 为整数，其中值 0 表示该年份包含 <b>date</b> 。 <b>period_no</b> 为负数表示前几年，为正数表示随后几年。
<b>first_month_of_year</b>	如果您不想从一月开始处理(财政)年，可在 <b>first_month_of_year</b> 中指定一个介于 2 和 12 之间的值。

示例和结果：

以下示例使用日期格式 **DD/MM/YYYY**。日期格式已经在数据加载脚本顶部的 **SET DateFormat** 语句中指定。可以根据要求更改示例中的格式。

示例	结果																										
<code>yearstart ('19/10/2001')</code>	返回 01/01/2001。																										
<code>yearstart ('19/10/2001', -1)</code>	返回 01/01/2000。																										
<code>yearstart ('19/10/2001', 0, 4)</code>	返回 01/04/2001。																										
<p>添加示例脚本到应用程序并运行。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。</p> <p>以下示例在表格中查找每个发票日期的年份的第一天，其中将该年度的第一个月指定为 4 月。</p> <pre>TempTable: LOAD RecNo() as InvID, * Inline [ InvDate 28/03/2012 10/12/2012 5/2/2013 31/3/2013 19/5/2013 15/9/2013 11/12/2013 2/3/2014 14/5/2014 13/6/2014 7/7/2014 4/8/2014 ];  InvoiceData: LOAD *, YearStart(InvDate, 0, 4) AS YrStart Resident TempTable; Drop table TempTable;</pre>	<p>结果列表包含原始日期和包括 <code>yearstart()</code> 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。</p> <table> <thead> <tr> <th>InvDate</th><th>YrStart</th></tr> </thead> <tbody> <tr><td>28/03/2012</td><td>01/04/2011</td></tr> <tr><td>10/12/2012</td><td>01/04/2012</td></tr> <tr><td>5/2/2013</td><td>01/04/2012</td></tr> <tr><td>31/3/2013</td><td>01/04/2012</td></tr> <tr><td>19/5/2013</td><td>01/04/2013</td></tr> <tr><td>15/9/2013</td><td>01/04/2013</td></tr> <tr><td>11/12/2013</td><td>01/04/2013</td></tr> <tr><td>2/3/2014</td><td>01/04/2013</td></tr> <tr><td>14/5/2014</td><td>01/04/2014</td></tr> <tr><td>13/6/2014</td><td>01/04/2014</td></tr> <tr><td>7/7/2014</td><td>01/04/2014</td></tr> <tr><td>4/8/2014</td><td>01/04/2014</td></tr> </tbody> </table>	InvDate	YrStart	28/03/2012	01/04/2011	10/12/2012	01/04/2012	5/2/2013	01/04/2012	31/3/2013	01/04/2012	19/5/2013	01/04/2013	15/9/2013	01/04/2013	11/12/2013	01/04/2013	2/3/2014	01/04/2013	14/5/2014	01/04/2014	13/6/2014	01/04/2014	7/7/2014	01/04/2014	4/8/2014	01/04/2014
InvDate	YrStart																										
28/03/2012	01/04/2011																										
10/12/2012	01/04/2012																										
5/2/2013	01/04/2012																										
31/3/2013	01/04/2012																										
19/5/2013	01/04/2013																										
15/9/2013	01/04/2013																										
11/12/2013	01/04/2013																										
2/3/2014	01/04/2013																										
14/5/2014	01/04/2014																										
13/6/2014	01/04/2014																										
7/7/2014	01/04/2014																										
4/8/2014	01/04/2014																										

## yeartodate

此函数用于判断输入日期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)。

### Syntax:

```
YearToDate(date [ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

**Return data type:** 布尔值

如果未使用可选参数，年初至今指日历年中 1 月 1 日以后任何一天，包括最近一次脚本执行日期。

### 参数：

参数	说明
date	要作为时间戳评估的日期，如“2012-10-12”。
yearoffset	通过指定 <b>yearoffset</b> , <b>yeartodate</b> 对于其他年份的同一时期返回 True。 <b>yearoffset</b> 为负表示上一年，偏移量为正表示下一年。通过指定 <b>yearoffset</b> = -1 获得至今的最近一年。如果忽略，则假设为 0。
firstmonth	通过在 1 和 12 之间(如果省略，则为 1)指定 <b>firstmonth</b> ，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 <b>firstmonth</b> = 5。
todaydate	通过指定一个 <b>todaydate</b> (如果忽略执行上次脚本时间戳)，这可作为该时期的上限移动该日。

### 示例和结果：

下例假设上次重新加载时间 = 2011-11-18

示例	结果
yeartodate( '2010-11-18')	返回 False
yeartodate( '2011-02-01')	返回 True
yeartodate( '2011-11-18')	返回 True
yeartodate( '2011-11-19')	返回 False
yeartodate( '2011-11-19', 0, 1, '2011-12-31')	返回 True
yeartodate( '2010-11-18', -1)	返回 True
yeartodate( '2011-11-18', -1)	返回 False
yeartodate( '2011-04-30', 0, 5)	返回 False
yeartodate( '2011-05-01', 0, 5)	返回 True



## 5.6 指数和对数函数

本部分介绍与指数计算和对数计算相关的函数。所有函数均可用于数据加载脚本和图表表达式。

在以下函数中，参数为表达式，其中 **x** 和 **y** 应解释为实值数。

### **exp**

自然指数函数  $e^x$ ，使用自然对数 **e** 作为底数。结果为正数。

```
exp(x )
```

**示例和结果：**

exp(3) 返回 20.085。

### **log**

**x** 的自然对数。仅在  $x > 0$  时才可定义此函数。结果为数字。

```
log(x )
```

**示例和结果：**

log(3) 返回 1.0986

### **log10**

**x** 的常用对数(以 10 为底)。仅在  $x > 0$  时才可定义此函数。结果为数字。

```
log10(x )
```

**示例和结果：**

log10(3) 返回 0.4771

### **pow**

返回 **x** 的 **y** 次幂。结果为数字。

```
pow(x,y )
```

**示例和结果：**

pow(3, 3) 返回 27

### **sqr**

**x** 平方(**x** 的 2 次幂)。结果为数字。

```
sqr (x )
```

**示例和结果：**

`sqr(3)` 返回 9

### **sqrt**

**x** 的平方根。仅在 **x** >= 0 时才可定义此函数。结果为正数。

```
sqrt(x)
```

**示例和结果：**

`sqrt(3)` 返回 1.732

## 5.7 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

`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])
```

`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)
```

`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)
```

`GetExcludedCount`

**GetExcludedCount()** 用于查找标识字段中排除(深灰色)值的数量。

```
GetExcludedCount - 图表函数 (第 421 页) (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])
```

### 字段和选择项函数

`GetCurrentSelections`

**GetCurrentSelections()** 用于返回应用程序中的当前选择项。

**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 **string**。

**GetFieldSelections** - 图表函数 ( field\_name [, value\_sep [, max\_values]])

### GetAlternativeCount - 图表函数

**GetAlternativeCount()**用于查找标识字段中可能(浅灰色)值的数量。

**Syntax:**

**GetAlternativeCount** (field\_name)

**Return data type:** 整数

**参数:**

参数	说明
field_name	包含要度量的数据范围的字段。

**示例和结果:**

以下示例使用两个加载到不同筛选器窗格的字段, 一个用于 **First name** 名称, 另一个用于 **Initials**。

示例	结果
假定选择 <b>John</b> (在 <b>First name</b> 中)。 GetAlternativeCount ([First name])	4, 因为 <b>First name</b> 中有 4 个唯一的排除(灰色)值。
假定已选择 <b>John</b> 和 <b>Peter</b> 。 GetAlternativeCount ([First name])	3, 因为 <b>First name</b> 中有 3 个唯一的排除(灰色)值。
假定未在 <b>First name</b> 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()** 用于返回应用程序中的当前选择项。

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。

### Syntax:

```
GetCurrentSelections ([record_sep [,tag_sep [,value_sep [,max_values]]]])
```

**Return data type:** 字符串

### 参数：

参数	说明
<code>record_sep</code>	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。
<code>tag_sep</code>	要置于字段名标记和字段值之间的分隔符。默认分隔符为“:”。
<code>value_sep</code>	置于字段值之间的分隔符。默认分隔符为“,”。
<code>max_values</code>	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。

### 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	结果
假定选择 <b>John</b> ( 在 <b>First name</b> 中)。 <code>GetCurrentSelections ()</code>	'First name: John'
假定已在 <b>First name</b> 中选择 <b>John</b> 和 <b>Peter</b> 。 <code>GetCurrentSelections ()</code>	'First name: John, Peter'
假定在 <b>First name</b> 中选择 <b>John</b> 和 <b>Peter</b> ，并在 <b>Initials</b> 中选择 <b>JA</b> 。 <code>GetCurrentSelections ()</code>	'First name: John, Peter  Initials: JA'

示例	结果
假定在 <b>First name</b> 中选择 <b>John</b> ，并在 <b>Initials</b> 中选择 <b>JA</b> 。  GetCurrentSelections ( chr(13)&chr(10) , ' = ' )	'First name = John  Initials = JA'
假定已在 <b>First name</b> 中选择除 Sue 以外的所有名称，并且在 <b>Initials</b> 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(深灰色)值的数量。

**Syntax:**

```
GetExcludedCount (field_name)
```

**Return data type:** 字符串

**限制:**

**GetExcludedCount()** 仅评估有相关值的字段，即没有选择项的字段。对于有选择项的字段，**GetExcludedCount()** 将返回 0。

**参数:**

参数	说明
field_name	包含要度量的数据范围的字段。

**示例和结果:**

以下示例使用两个加载到不同筛选器窗格的字段，一个用于 **First name** 名称，另一个用于 **Initials**。

示例	结果
假定选择 <b>John</b> ( 在 <b>First name</b> 中)。  <code>GetExcludedCount ([Initials])</code>	5, 因为 <b>Initials</b> 中有 5 个排除(灰色)值。第六个单元格 (JA) 为白色, 因为它与 John 中的选择项 <b>First name</b> 关联。
假定已选择 <b>John</b> 和 <b>Peter</b> 。  <code>GetExcludedCount ([Initials])</code>	3, 因为 Peter 与 <b>Initials</b> 中的 2 个值关联。
假定未在 <b>First name</b> 中选择任何值。  <code>GetExcludedCount ([Initials])</code>	0, 因为没有选择项。
假定选择 <b>John</b> ( 在 <b>First name</b> 中)。  <code>GetExcludedCount ([First name])</code>	0, 因为 <b>GetExcludedCount()</b> 仅评估有相关值的字段, 即没有选择项的字段。

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 **string**。

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。

**Syntax:**

```
GetFieldSelections ( field_name [, value_sep [, max_values]])
```

**Return data type:** 字符串

**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。

**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	结果
假定选择 <b>John</b> ( 在 <b>First name</b> 中)。  GetFieldSelections ([First name])	“John”
假定已选择 <b>John</b> 和 <b>Peter</b> 。  GetFieldSelections ([First name])	“John,Peter”
假定已选择 <b>John</b> 和 <b>Peter</b> 。  GetFieldSelections ([First name],'; ')	“John; Peter”
假定已在 <b>First name</b> 中选择 <b>John</b> 、 <b>Sue</b> 、 <b>Mark</b> 。  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。

**Syntax:**

```
GetNotSelectedCount(fieldname [, includeexcluded=false])
```

参数：

参数	说明
fieldname	要求值的字段的名称。
includeexcluded	如果 <b>includeexcluded</b> 表述为 True, 计数将包括在一个其他字段中被选择项排除的选定值。

示例：

```
GetNotSelectedCount( Country )
GetNotSelectedCount( Country, true )
```

## 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。

**Return data type:** 整数

**Syntax:**

```
GetPossibleCount (field_name)
```

参数：

参数	说明
field_name	包含要度量的数据范围的字段。

示例和结果：

以下示例使用两个加载到不同筛选器窗格的字段, 一个用于 **First name** 名称, 另一个用于 **Initials**。

示例	结果
假定选择 <b>John</b> ( 在 <b>First name</b> 中)。  <code>GetPossibleCount ([Initials])</code>	1, 因为 Initials 中有 1 个值与 <b>First name</b> 中的选择项 <b>John</b> 关联。



示例	结果
假定选择 <b>John</b> (在 <b>First name</b> 中)。  <code>GetPossibleCount ([First name])</code>	1, 因为 <b>First name</b> 中有 1 个选择项 <b>John</b> 。
假定选择 <b>Peter</b> (在 <b>First name</b> 中)。  <code>GetPossibleCount ([Initials])</code>	2, 因为 Peter 与 <b>Initials</b> 中的 2 个值关联。
假定未在 <b>First name</b> 中选择任何值。  <code>GetPossibleCount ([First name])</code>	5, 因为没有选择项, 并且 <b>First name</b> 中有 5 个唯一的值。
假定未在 <b>First name</b> 中选择任何值。  <code>GetPossibleCount ([Initials])</code>	6, 因为没有选择项, 并且 <b>Initials</b> 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。

**Syntax:**

```
GetSelectedCount (field_name [, include_excluded])
```

**Return data type:** 整数

**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 <b>True()</b> , 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 <b>False</b> 或省略, 则这些值不会包括在内。

**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	结果
假定选择 <b>John</b> ( 在 <b>First name</b> 中)。 <code>GetSelectedCount ([First name])</code>	1, 因为已在 <b>First name</b> 中选择一个值。
假定选择 <b>John</b> ( 在 <b>First name</b> 中)。 <code>GetSelectedCount ([Initials])</code>	0, 因为未在 <b>Initials</b> 中选择任何值。
在 <b>First name</b> 中没有选择任何选择项, 在 <b>Initials</b> 中选择所有值, 之后在 <b>Has cellphone</b> 中选择值 <b>Yes</b> 。 <code>GetSelectedCount ([Initials])</code>	6. 虽然带有 <b>Initials</b> MC 和 PD 的选择项的 <b>Has cellphone</b> 设置为 <b>No</b> , 但结果仍是 6, 因为参数 <code>include_excluded</code> 已设置为 <code>True()</code> 。

示例中所使用的数据：

```
Names:
LOAD * inline [
"First name"|"Last name"|"Initials"|"Has cellphone"
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC |No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');
```

## 5.8 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。支持下列文件格式: 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

**ConnectionString()** 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。

```
ConnectionString ()
```

### FileName

**FileName** 函数用于返回当前阅读表格广播的名称，没有路径或扩展名。

```
FileName ()
```

### 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** 函数用于返回当前阅读表格广播的名称，没有路径但包括扩展名。

```
FileName ()
```

### FilePath

**FilePath** 函数用于返回一个包含至当前阅读表格文件的完整路径的字符串。

```
FilePath ()
```

### FileSize

**FileSize** 函数用于返回一个包含文件 filename 字节大小的整数，或如果未指定 filename，则返回一个包含当前阅读的表格文件字节大小的整数。

```
FileSize ()
```

### FileTime

**FileTime** 函数用于返回文件 filename 的上一次修改日期和时间的戳。如果未指定 filename，则此函数将参考当前阅读的表格文件。

```
FileTime ([ filename ])
```

### GetFolderPath

**GetFolderPath** 函数用于返回 Microsoft Windows SHGetFolderPath 函数的值和返回相关路径。例如，**MyMusic**。注意，此函数不使用在 Windows Explorer 中看到的空间。

```
GetFolderPath ()
```

### QvdCreateTime

此脚本函数用于返回 QVD 文件的 XML-标题时间戳(如果有)，否则返回 NULL 值。

```
QvdCreateTime (filename)
```

### **QvdFieldName**

此脚本函数用于返回字段编号名 **fieldno**(如果其存在于 QVD 文件中)(否则, 返回 NULL 值)。

```
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。支持下列文件格式: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。

### Syntax:

```
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	说明
filename	<p>如有必要，媒体文件名称包括路径作为文件夹数据连接。</p> <p><b>示例：</b><i>'lib://Table Files/'</i></p> <p>在传统脚本模式下，同时支持以下路径格式：</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例：</b><i>c:\data\</i></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例：</b><i>data\</i></p>
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。因为在传统模式下仅支持 **GetFolderPath**，您需要将对 **GetFolderPath** 的引用替换为 lib:// 数据连接路径。

另请：文件系统访问限制(第 593 页)

#### 示例 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, // Unsynchronized 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

```

## 示例 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

```

### 示例 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

```

#### 示例 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。

#### Syntax:

```
ConnectString()
```

示例和结果:

示例	结果
<pre>LIB CONNECT TO 'Tutorial ODBC'; ConnectString: Load ConnectString() as ConnectString AutoGenerate 1;</pre>	<p>在字段 ConnectString 中返回“Tutorial ODBC”。</p> <p>此示例假设您拥有名为 Tutorial ODBC 的可用数据连接。</p>

## FileName

**FileName** 函数用于返回当前阅读表格广播的名称，没有路径或扩展名。

**Syntax:**

**FileName()**

示例和结果：

示例	结果
<pre>LOAD *, filename( ) as X from C:\UserFiles\abc.txt</pre>	<p>将在每个阅读记录中的 X 字段中返回“abc”。</p>

## 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。

**Syntax:**

**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：

示例	结果
<pre>Load *, filedir( ) as X from C:\UserFiles\abc.txt</pre>	<p>将在每个阅读记录中的 X 字段中返回“C:\UserFiles”。</p>

## 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。

**Syntax:**

**FileExtension()**

示例和结果：

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** 函数用于返回当前阅读表格广播的名称，没有路径但包括扩展名。

### Syntax:

**FileName()**

示例和结果：

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。

### Syntax:

**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：

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。

### Syntax:

**FileSize([filename])**

参数：

参数	说明
filename	<p>文件名 (如有必要) 包括路径, 作为文件夹或 Web 文件的数据连接。如果您没有指定文件名, 将使用当前正在读取的表格文件。</p> <p><b>示例: 'lib://Table Files/'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例: c:\data\</b></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例: data\</b></p> <ul style="list-style-type: none"> <li>URL 地址 (HTTP 或 FTP), 指向一个互联网或内联网的位置。</li> </ul> <p><b>示例: http://www.qlik.com</b></p>

示例和结果:

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://MyData/xyz.xls' )	将返回文件 xyz.xls 的大小。

## FileTime

**FileTime** 函数用于返回文件 filename 的上一次修改日期和时间的戳。如果未指定 filename, 则此函数将参考当前阅读的表格文件。

**Syntax:**

```
FileTime( [ filename ] )
```

**参数:**

参数	说明
filename	<p>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。</p> <p><b>示例: 'lib://Table Files'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例: c:\data\</b></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例: data\</b></p> <ul style="list-style-type: none"> <li>URL 地址 (HTTP 或 FTP), 指向一个互联网或内联网的位置。</li> </ul> <p><b>示例: http://www.qlik.com</b></p>

示例和结果:

示例	结果
LOAD *, FileTime( ) as X from abc.txt;	将以整数形式在每个阅读记录的 X 字段中以时间戳形式返回文件 (abc.txt) 上一次修改的日期和时间。
FileTime( 'xyz.xls' )	将返回文件 xyz.xls 上一次修改的时间戳。

## GetFolderPath

**GetFolderPath** 函数用于返回 Microsoft Windows SHGetFolderPath 函数的值并返回相关路径。例如, **MyMusic**。注意, 此函数不用于在 Windows Explorer 中看到的空间。



在标准模式下不支持此函数。

**Syntax:**

**GetFolderPath ( )**

**示例:**

此脚本用于加载表格中三个常用的文件夹路径。

```
LOAD
  GetFolderPath('Music') as MyMusic,
  GetFolderPath('MyPictures') as MyPictures,
  GetFolderPath('Windows') as Windows
AutoGenerate 1;
```

## 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。

### Syntax:

```
QvdCreateTime(filename)
```

### 参数:

参数	说明
filename	<p>QVD 文件名(如有必要)包括路径, 作为文件夹或 Web 数据连接。</p> <p><b>示例: 'lib://Table Files'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例: c:\data\</b></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例: data\</b></p> <ul style="list-style-type: none"> <li>URL 地址(HTTP 或 FTP), 指向一个互联网或内联网的位置。</li> </ul> <p><b>示例: http://www.qlik.com</b></p>

### 示例:

```
QvdCreateTime('MyFile.qvd')
QvdCreateTime('C:\MyDir\MyFile.qvd')
QvdCreateTime('lib://data\MyFile.qvd')
```

## QvdFieldName

此脚本函数用于返回字段编号名 **fieldno**(如果其存在于 QVD 文件中)(否则, 返回 NULL 值)。

### Syntax:

```
QvdFieldName(filename , fieldno)
```

### 参数:



参数	说明
filename	<p>QVD 文件名 (如有必要) 包括路径, 作为文件夹或 Web 数据连接。</p> <p><b>示例: 'lib://Table Files/'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例: c:\data\</b></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例: data\</b></p> <ul style="list-style-type: none"> <li>URL 地址 (HTTP 或 FTP), 指向一个互联网或内联网的位置。</li> </ul> <p><b>示例: http://www.qlik.com</b></p>
fieldno	QVD 文件中所含的表格内的字段编号 (以 0 为基础)。

**示例:**

```
QvdFieldName ('MyFile.qvd', 3)
QvdFieldName ('C:\MyDir\MyFile.qvd', 5)
QvdFieldName ('lib://data\MyFile.qvd', 5)
```

## QvdNoOfFields

此脚本函数用于返回 QVD 文件中的字段数。

**Syntax:**

```
QvdNoOfFields (filename)
```

**参数:**

参数	说明
filename	<p>QVD 文件名 (如有必要) 包括路径, 作为文件夹或 Web 数据连接。</p> <p><b>示例: 'lib://Table Files/'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"><li>absolute</li></ul> <p><b>示例: c:\data\</b></p> <ul style="list-style-type: none"><li>相对 Qlik Sense 应用程序工作目录的相对路径。</li></ul> <p><b>示例: data\</b></p> <ul style="list-style-type: none"><li>URL 地址 (HTTP 或 FTP), 指向一个互联网或内联网的位置。</li></ul> <p><b>示例: http://www.qlik.com</b></p>

**示例:**

```
QvdNoOfFields ('MyFile.qvd')
QvdNoOfFields ('C:\MyDir\MyFile.qvd')
QvdNoOfFields ('lib://data\MyFile.qvd')
```

## QvdNoOfRecords

此脚本函数用于返回 QVD 文件中的当前记录数。

**Syntax:**

```
QvdNoOfRecords (filename)
```

**参数:**

参数	说明
filename	<p>QVD 文件名 (如有必要) 包括路径, 作为文件夹或 Web 数据连接。</p> <p><b>示例: 'lib://Table Files/'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"> <li>absolute</li> </ul> <p><b>示例: c:\data\</b></p> <ul style="list-style-type: none"> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> </ul> <p><b>示例: data\</b></p> <ul style="list-style-type: none"> <li>URL 地址 (HTTP 或 FTP), 指向一个互联网或内联网的位置。</li> </ul> <p><b>示例: http://www.qlik.com</b></p>

**示例:**

```
QvdNoOfRecords ('MyFile.qvd')
QvdNoOfRecords ('C:\MyDir\MyFile.qvd')
QvdNoOfRecords ('lib://data\MyFile.qvd')
```

## QvdTableName

此脚本函数用于返回存储在 QVD 文件中的表格名称。

**Syntax:**

```
QvdTableName (filename)
```

**参数:**

参数	说明
filename	<p>QVD 文件名 (如有必要) 包括路径, 作为文件夹或 Web 数据连接。</p> <p><b>示例: 'lib://Table Files/'</b></p> <p>在传统脚本模式下, 同时支持以下路径格式:</p> <ul style="list-style-type: none"> <li>absolute</li> <li><b>示例: c:\data\</b></li> <li>相对 Qlik Sense 应用程序工作目录的相对路径。</li> <li><b>示例: data\</b></li> <li>URL 地址 (HTTP 或 FTP), 指向一个互联网或内联网的位置。</li> <li><b>示例: http://www.qlik.com</b></li> </ul>

**示例:**

```
QvdTableName ('MyFile.qvd')
QvdTableName ('C:\MyDir\MyFile.qvd')
QvdTableName ('lib://data\MyFile.qvd')
```

## 5.9 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, 不变付款额以及不变利率一项投资的未来值。结果拥有一个 **money** 的默认数字格式。

```
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。结果拥有一个 **money** 的默认数字格式。

```
Pmt (rate, nper, pv [ ,fv [ , type ] ] )
```

#### **PV**

此函数用于返回一项投资的现在价值。结果拥有一个 **money** 的默认数字格式。

```
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)
```

**Return data type:** 数字

**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	每个时间周期的股价波动性 (%)。
risk_free_rate	每个时间周期无风险利率 (%)。
type	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。

示例和结果：

示例	结果
BlackAndSchole(130, 4, 68.5, 0.4, 0.04, 'call')  这用于计算期权的理论价格，如果以每股 130 的价格购买 4 年，则现在每股增值 68.5。假设每年股价波动为 40%，无风险利率为 4%。	返回 11.245

## FV

此函数用于返回基于周期性，不变付款额以及不变利率一项投资的未来值。结果拥有一个 **money** 的默认数字格式。

**Syntax:**

```
FV(rate, nper, pmt [ ,pv [ , type ] ])
```

**Return data type:** 数字。结果默认采用货币数字格式。。

**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。如果省略了 <b>pmt</b> ，则必须包括 <b>pvt</b> 参数。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 <b>pv</b> ，假设它为 0(零)，并且必须包括 <b>pmt</b> 参数。
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 <b>fv</b> ，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 <b>type</b> ，假设为 0。

示例和结果：

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。

**Syntax:**

```
nPer(rate, pmt, pv [ ,fv [ , type ] ])
```

**Return data type:** 数字**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。如果省略了 <b>pmt</b> , 则必须包括 <b>pv</b> 参数。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 <b>pv</b> , 假设它为 0(零), 并且必须包括 <b>pmt</b> 参数。
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 <b>fv</b> , 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 <b>type</b> , 假设为 0。

**示例和结果:**

示例	结果
<p>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?</p> <p><code>nPer(0.005, -20, 0, 800)</code></p>	<p>返回 36.56</p>

## 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。结果拥有一个 **money** 的默认数字格式。

```
Pmt(rate, nper, pv [ ,fv [ , type ] ] )
```

**Return data type:** 数字。结果默认采用货币数字格式。。

要想算出贷款期间的付款总额, 将返回的 **pmt** 值乘以 **nper**。

**参数:**

参数	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。如果省略了 <b>pmt</b> , 则必须包括 <b>pv</b> 参数。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 <b>pv</b> , 假设它为 0(零), 并且必须包括 <b>pmt</b> 参数。
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 <b>fv</b> , 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 <b>type</b> , 假设为 0。

示例和结果:

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: <code>Pmt(0.1/12,8,20000)</code>	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: <code>Pmt(0.1/12,8,20000,0,1)</code>	返回 - \$2,573.21

## PV

此函数用于返回一项投资的现在价值。结果拥有一个 **money** 的默认数字格式。

```
PV(rate, nper, pmt [ ,fv [ , type ] ])
```

**Return data type:** 数字。结果默认采用货币数字格式。。

现在价值是一系列未来付款现在价值的总额。例如, 当借钱时, 贷款额对于贷款人来说就是现值。

参数:

参数	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。如果省略了 <b>pmt</b> , 则必须包括 <b>pv</b> 参数。付款以负数表示, 例如 -20。



参数	说明
p <sup>v</sup>	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 <b>p<sup>v</sup></b> , 假设它为 0(零), 并且必须包括 <b>p<sup>m</sup>t</b> 参数。
f <sup>v</sup>	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 <b>f<sup>v</sup></b> , 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 <b>type</b> , 假设为 0。

示例和结果:

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?  <code>PV(0.07/12, 12*5, -100, 0, 0)</code>	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 %。

**Syntax:**

```
Rate(nper, pmt, pv [, fv [, type ] ])
```

**Return data type:** 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	说明
rate	每周期的利率。
nper	年金中付款周期的总数。
p <sup>m</sup> t	每个周期的付款。它无法改变年金的周期。如果省略了 <b>p<sup>m</sup>t</b> , 则必须包括 <b>p<sup>v</sup></b> 参数。付款以负数表示, 例如 -20。
p <sup>v</sup>	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 <b>p<sup>v</sup></b> , 假设它为 0(零), 并且必须包括 <b>p<sup>m</sup>t</b> 参数。
f <sup>v</sup>	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 <b>f<sup>v</sup></b> , 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 <b>type</b> , 假设为 0。

示例和结果：

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？  <code>Rate(60,-300,10000)</code>	返回 2.00%

### 5.10 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。



为了清晰起见，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。

#### 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()** 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)中设置的数字格式，以及可选的小数位和千分位分隔符，以数字形式设置表达式的格式。

```
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])
```

另请参阅：

□ [解释函数\(第 473 页\)](#)

## 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。

**Syntax:**

```
Date (number[, format])
```

**Return data type:** 双

**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，则使用在操作系统中设置的日期格式。

**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	结果	设置 1	设置 2
Date( A ) 其中 A=35648	字符串:	97-08-06	8/6/97
	数字:	35648	35648
Date( A, 'YY.MM.DD' ) 其中 A=35648	字符串:	97.08.06	97.08.06
	数字:	35648	35648
Date( A, 'DD.MM.YYYY' ) 其中 A=35648.375	字符串:	06.08.1997	06.08.1997
	数字:	35648.375	35648.375
Date( A, 'YY.MM.DD' ) 其中 A=8/6/97	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。

### Syntax:

**Dual** (text, number)

### Return data type: 双

### 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，所有字段值都可能都是对偶值。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:

示例	说明
在脚本中添加下例并运行。  <pre>Load dual ( NameDay,NumDay ) as DayOfWeek inline [ NameDay,NumDay Monday,0 Tuesday,1 Wednesday,2 Thursday,3 Friday,4 Saturday,5 Sunday,6 ];</pre>	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。
<pre>Load Dual('Q' &amp; Ceil(Month (Now())/3), Ceil(Month(Now ())/3)) as Quarter AutoGenerate 1;</pre>	本例提供当前季度。如果在一年的第一个三个月中运行 <b>Now()</b> 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 将按其数值顺序排序:1 到 4。
<pre>Dual('Q' &amp; Ceil(Month(Date) /3), Ceil(Month(Date)/3)) as Quarter</pre>	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。
<pre>Dual(WeekYear(Date) &amp; '-w' &amp; Week(Date), WeekStart (Date)) as YearWeek</pre>	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。

### Syntax:

```
Interval(number[, format])
```

### Return data type: 双

### 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:。

示例	字符串	数字
Interval( A ) 其中 A=0.375	9:00:00	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。

### Syntax:

```
Money (number[, format[, dec_sep[, thou_sep]])
```

**Return data type:** 双

### 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 设置 1: kr ##0,00, MoneyThousandSep''
- MoneyFormat 设置 2: \$ #,##0.00, MoneyThousandSep','

示例	结果	设置 1	设置 2
Money( A ) 其中 A=35648	字符串:	kr 35 648,00	\$ 35,648.00
	数字:	35648.00	35648.00
Money( A, '#,##0 ¥', '.' , ',' ) 其中 A=3564800	字符串:	3,564,800 ¥	3,564,800 ¥
	数字:	3564800	3564800

## Num

**Num()** 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)中设置的数字格式,以及可选的小数位和千分位分隔符,以数字形式设置表达式的格式。

### Syntax:

```
Num(number[, format[, dec_sep [, thou_sep]])
```

**Return data type:** 双

**参数:**

参数	说明
number	可以设置数字的格式。
format	说明如何设置结果日期字符串格式的字符串。如果省略,则使用在操作系统中设置的日期格式。
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。

**示例和结果:**

以下示例假设采用以下默认设置:

- 数字格式设置 1: ###0
- 数字格式设置 2: #,##0

示例	结果	设置 1	设置 2
Num( A, '0.0' ) 其中 A=35648.375	字符串:	35 648 375	35648.375
	数字:	35648375	35648.375
Num( A, '#,##0.##', '.' , ',' ) 其中 A=35648	字符串:	35,648.00	35,648.00
	数字:	35648	35648

示例	结果	设置 1	设置 2
Num( pi( ), '0,00' )	字符串:	3,14	003
	数字:	3.141592653	3.141592653

## 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。

### Syntax:

```
Time(number[, format])
```

**Return data type:** 双

**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	结果	设置 1	设置 2
Time( A ) 其中 A=0.375	字符串:	9:00:00	09.00.00
	数字:	0.375	0.375
Time( A ) 其中 A=35648.375	字符串:	9:00:00	09.00.00
	数字:	35648.375	35648.375
Time( A, 'hh-mm' ) 其中 A=0.99999	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。

### Syntax:



**Timestamp**(number[, format])**Return data type:** 双**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:

- TimeStampFormat 设置 1: YY-MM-DD hh:mm:ss
- TimeStampFormat 设置 2: M/D/YY hh:mm:ss

示例	结果	设置 1	设置 2
Timestamp( A ) 其中 A=35648.375	字符串:	97-08-06 09:00:00	1997-08-06 9:00:00
	数字:	35648.375	35648.375
Timestamp( A, 'YYYY-MM-DD hh.mm') 其中 A=35648	字符串:	1997-08-06 00.00	1997-08-06 00.00
	数字:	35648	35648

## 5.11 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()** 用于查找将数值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： $\text{Combin}(p,q) = p! / q!(p-q)!$   
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： $\text{Permut}(p,q) = (p)! / (p - q)!$   
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()** 用于将数值向上取整到指定 **step** 间隔的最接近倍数。用 **offset** 值( 如果已指定) 加上该结果, 或者减去该结果( 如果 **offset** 为负数)。

```
Ceil(x[, step[, offset]])
```

floor

**Floor()** 用于将数值向下取整到指定 **step** 间隔的最接近倍数。用 **offset** 值( 如果已指定) 加上该结果, 或者减去该结果( 如果 **offset** 为负数)。

```
Floor(x[, step[, offset]])
```

round

**Round()** 用于返回将 **x** 向上或向下取整到 **step** 最接近倍数的结果。用 **offset** 值( 如果已指定) 加上该结果, 或者减去该结果( 如果 **offset** 为负数)。

```
Round( x [ , base [ , offset ] ] )
```

## BitCount

**BitCount()** 用于查找将数值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。

**Syntax:**

```
BitCount(integer_number)
```

**Return data type:** 整数

**示例和结果:**

示例	结果
BitCount ( 3 )	3 的二进制是 101, 因此返回 2
BitCount ( -1 )	-1 的二进制是 64 个 1, 因此返回 64

## Ceil

**Ceil()** 用于将数值向上取整到指定 **step** 间隔的最接近倍数。用 **offset** 值(如果已指定)加上该结果, 或者减去该结果(如果 **offset** 为负数)。

与 **floor** 函数比较, 此函数对输入数值向下取整。

### Syntax:

```
Ceil(x[, step[, offset]])
```

**Return data type:** 整数

### 示例和结果:

示例	结果
Ceil( 2.4 )	返回 3
Ceil( 2.6 )	返回 3
Ceil( 3.88 , 0.1 )	返回 3.9
Ceil( 3.88 , 5 )	返回 5
Ceil( 1.1 , 1 )	返回 2
Ceil( 1.1 , 1 , 0.5 )	返回 1.5
Ceil( 1.1 , 1 , -0.01 )	返回 1.99

## 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:  $\text{Combin}(p,q) = p! / q!(p-q)!$  选择项目的顺序不重要。

### Syntax:

```
Combin(p, q)
```

**Return data type:** 整数

### 限制:

非整数项目将会被截短。

### 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？ <code>Combin( 35,7 )</code>	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。

### Syntax:

```
Div(integer_number1, integer_number2)
```

**Return data type:** 整数

### 示例和结果：

示例	结果
<code>Div( 7,2 )</code>	返回 3
<code>Div( 7.1,2.3 )</code>	返回 3
<code>Div( 9,3 )</code>	返回 3
<code>Div( -4,3 )</code>	返回 -1
<code>Div( 4,-3 )</code>	返回 -1
<code>Div( -4,-3 )</code>	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)。

### Syntax:

```
Even(integer_number)
```

**Return data type:** 布尔值

### 示例和结果：

示例	结果
<code>Even( 3 )</code>	返回 0 False
<code>Even( 2 * 10 )</code>	返回 -1 True
<code>Even( 3.14 )</code>	返回 NULL

## Fabs

**Fabs()** 用于返回 **x** 的绝对值。结果为正数。

**Syntax:**

```
fabs (x)
```

**Return data type:** 数字

**示例和结果:**

示例	结果
<code>fabs( 2.4 )</code>	返回 2.4
<code>fabs( -3.8 )</code>	返回 3.8

## Fact

**Fact()** 用于返回正整数 **x** 的阶乘。

**Syntax:**

```
Fact (x)
```

**Return data type:** 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	结果
<code>Fact( 1 )</code>	返回 1
<code>Fact( 5 )</code>	返回 120 ( $1 * 2 * 3 * 4 * 5 = 120$ )
<code>Fact( -5 )</code>	返回 NULL

## Floor

**Floor()** 用于将数值向下取整到指定 **step** 间隔的最接近倍数。用 **offset** 值( 如果已指定) 减去该结果, 或者如果 **offset** 为负数, 则加上该结果。

与 **ceil** 函数比较, 此函数对输入数值向上取整。

**Syntax:**

```
Floor(x[, step[, offset]])
```

**Return data type:** 数字

**示例和结果:**

示例	结果
Floor( 2,4 )	返回 0
Floor( 4,2 )	返回 4
Floor( 3.88 , 0.1 )	返回 3.8
Floor( 3.88 , 5 )	返回 0
Floor( 1.1 , 1 )	返回 1
Floor( 1.1 , 1 , 0.5 )	返回 0.5

## 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。

**Syntax:**

```
fmod(a, b)
```

**Return data type:** 数字

**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  $\text{Frac}(x) + \text{Floor}(x) = x$  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  $-1.4$ ,  $\text{Floor}(-1.4) = -2$ , 将生成以下结果:

$-1.4$  的小数部分  $= 1.4 - (-2) = -1.4 + 2 = 0.6$

**Syntax:**

```
Frac(x)
```

**Return data type:** 数字

**示例和结果:**

示例	结果
<code>Frac( 11.43 )</code>	返回 0.43
<code>Frac( -1.4 )</code>	返回 0.6

## 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。

**Syntax:**

```
Mod(integer_number1, integer_number2)
```

**Return data type:** 整数

**限制:**

**integer\_number2** 必须大于 0。

**示例和结果:**

示例	结果
<code>Mod( 7,2 )</code>	返回 1
<code>Mod( 7.5,2 )</code>	返回 NULL
<code>Mod( 9,3 )</code>	返回 0
<code>Mod( -4,3 )</code>	返回 2
<code>Mod( 4,-3 )</code>	返回 NULL
<code>Mod( -4,-3 )</code>	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)。

**Syntax:**

```
Odd(integer_number)
```

**Return data type:** 布尔值

**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： $\text{Permut}(p,q) = (p)! / (p - q)!$  选择项目的顺序很重要。

**Syntax:**

```
Permut(p, q)
```

**Return data type:** 整数

**限制:**

非整数型参数将被截短。

**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()** 用于返回将 **x** 向上或向下取整到 **step** 最接近倍数的结果。用 **offset** 值( 如果已指定) 加上该结果，或者减去该结果( 如果 **offset** 为负数)。**step** 的默认值为 1。

如果 **x** 正处于一个区间的中间，则对其向上取整。

**Syntax:**

```
Round(x[, step[, offset]])
```

**Return data type:** 数字**示例和结果:**

示例	结果
Round( 3.8 )	返回 4
Round( 3.8, 4 )	返回 4
Round( 2.5 )	返回 3。向上取整, 因为 2.5 正好是默认步进间隔的一半。
Round( 2, 4 )	返回 4。向上取整, 因为 2 正好是步进间隔 4 的一半。
Round( 2, 6 )	返回 0。向下取整, 因为 2 小于步进间隔 6 的一半。
Round( 3.88 , 0.1 )	返回 3.9
Round( 3.88 , 5 )	返回 5
Round( 1.1 , 1 , 0.5 )	返回 1.5

## Sign

**Sign()** 用于根据 x 是正数、0 或负数分别返回 1, 0 或 -1。**Syntax:**

```
Sign(x)
```

**Return data type:** 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	结果
Sign( 66 )	返回 1
Sign( 0 )	返回 0
Sign( - 234 )	返回 -1

## 5.12 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。

### 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()** 可用于将几何体聚合到区域中，以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, long_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。

**Syntax:**

```
GeoAggrGeometry (field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(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。



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 `GeoReduceGeometru()`。

### 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。

**Syntax:**

```
GeoBoundingBox(field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(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。

**Syntax:**

```
GeoCountVertex(field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(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。

**Syntax:**

```
GeoGetBoundingBox (field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(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。

**Syntax:**

```
GeoGetPolygonCenter (field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(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。

**Syntax:**

```
GeoInvProjectGeometry(type, field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(或点集)，或者一个区域。

**示例:**

示例	结果
在 Load 语句中: GeoInvProjectGeometry (‘mercator’,AreaPolygon) as InvProjectGeometry	使用 Mercator 投影的反向转换来转换加载作为 <b>AreaPolygon</b> 的几何体，并存储作为 <b>InvProjectGeometry</b> 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。

**Syntax:**

```
GeoMakePoint(lat_field_name, lon_field_name)
```

**Return data type:** 字符串

**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。
long_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。

**Syntax:**

```
GeoProject(type, field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(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:**

示例	结果
在 Load 语句中: GeoProject('mercator',Area) as GetProject	Mercator 投影应用于加载作为 <b>Area</b> 的几何体，并将结果作为 <b>GetProject</b> 存储。

## GeoProjectGeometry

**GeoProjectGeometry()** 可用于将几何体聚合到区域中，并可应用投影。

**Syntax:**

```
GeoProjectGeometry(type, field_name)
```

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(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 <b>AreaPolygon</b> 加载的几何体， 并作为 <b>ProjectGeometry</b> 存储以便在可视化中使用。

## GeoReduceGeometry

**GeoReduceGeometry()** 可用于将几何体聚合到区域中，以显示个别区域的边界线。

**Syntax:**

**GeoReduceGeometry** (field\_name)

**Return data type:** 字符串

**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。这可以是提供经度或纬度的一个点(或点集)，或者一个区域。

**GeoReduceGeometry()** 执行与 **GeoAggrGeometry()** 类似的功能，用于组合地理空间边界数据。单个边界线与聚合前数据的差别会显示在地图上。

## 5.13 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对偶值，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。



为了清晰起见, 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默认的日期格式。

```
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)中设置的格式, 以及可选的小数位和千分位分隔符, 计算表达式的数字货币值。

```
Money#(text[, format[, dec_sep[, thou_sep ] ] ])
```

#### Num#

**Num#()** 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)中设置的数字格式, 以及可选的小数位和千分位分隔符, 计算表达式的数字值。

```
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])

另请参阅:

□ 格式函数(第 450 页)

## Date#

**Date#** 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, 计算表达式的日期值。

**Syntax:**

**Date#** (text[, format])

**Return data type:** 双

**参数:**

参数	说明
<b>text</b>	可以计算文本字符串值。
<b>format</b>	说明如何设置结果日期字符串格式的字符串。如果省略, 则使用在操作系统中设置的日期格式。

**示例和结果:**

以下示例假定了以下两个操作系统设置:

- 日期格式默认设置 1: YY-MM-DD
- 日期格式默认设置 2: M/D/YY

示例	结果	设置 1	设置 2
Date#( A ) 其中 A="8/6/97"	字符串:	08/06/1997	08/06/1997
	数字:	-	35648
Date#( A, 'YYYY.MM.DD' ) 其中 A="1997.08.06"	字符串:	1997.08.06	1997.08.06
	数字:	35648	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。

**Syntax:**

```
Interval#(text[, format])
```

**Return data type:** 双**参数:**

参数	说明
text	可以计算文本字符串值。
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。

**interval#** 函数与 **time#** 函数相似,但使用的是所有值的间隔,而 **time#** 值必须介于 0 和 24 小时之间。

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:。

示例	结果	
Interval#( A, 'D hh:mm' ) 其中 A='1 09:00'	字符串:	1 09:00
	数字:	1.375
Interval#( A-B ) 其中 A='97-08-06 09:00:00' 和 B='97-08-05 00:00:00'	字符串:	1.375
	数字:	1.375

## Money#

**Money#()** 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) 中设置的格式, 以及可选的小数位和千分位分隔符, 计算表达式的数字货币值。

**Syntax:**

```
Money#(text[, format[, dec_sep [, thou_sep ] ] ])
```

**Return data type:** 双**参数:**

参数	说明
text	可以计算文本字符串值。
format	说明如何设置结果日期字符串格式的字符串。如果省略，则使用在操作系统中设置的日期格式。
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

示例	结果	设置 1	设置 2
Money#(A , '# ##0,00 kr' ) 其中 A=35 648,37 kr	字符串：	35 648.37 kr	35 648.37 kr
	数字：	35648.37	3564837
Money#( A, ' \$#', '.', ',' ) 其中 A= \$35,648.37	字符串：	\$35,648.37	\$35,648.37
	数字：	35648.37	35648.37

## Num#

**Num#()** 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) 中设置的数字格式，以及可选的小数位和千分位分隔符，计算表达式的数字值。

**Syntax:**

```
Num# (text[, format[, dec_sep [, thou_sep ] ] ])
```

**Return data type:** 双

**参数：**

参数	说明
text	可以计算文本字符串值。
format	说明如何设置结果日期字符串格式的字符串。如果省略，则使用在操作系统中设置的日期格式。

参数	说明
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。

示例和结果：

以下示例假定了以下两个操作系统设置：

- 数字格式默认设置 1: ###0
- 数字格式默认设置 2: #,##0

示例	结果	设置 1	设置 2
Num#( A, '#' ) 其中 A=35,648.375	字符串：	35,648.375	35648.375
	数字：	-	35648.375
Num#( A, '#.#', '.', ',' ) 其中 A=35,648.375	字符串：	35,648.375	35,648.375
	数字：	35648.375	35648.375
Num#( A, '#.#', ',', '.' ) 其中 A=35648.375	字符串：	35648.375	35648.375
	数字：	35648375	35648375
Num#( A, 'abc#,#' ) 其中 A=abc123,4	字符串：	abc123,4	abc123,4
	数字：	123.4	1234

## 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。

**Syntax:**

**Text** (expr)

**Return data type:** 双

示例和结果：

示例	结果	
Text( A ) 其中 A=1234	字符串：	1234
	数字：	-
Text( pi( ) )	字符串：	3.1415926535898
	数字：	-

## 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。。

**Syntax:**

```
time#(text[, format])
```

**Return data type:** 双

**参数:**

参数	说明
text	可以计算文本字符串值。
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:**

以下示例假定了以下两个操作系统设置:

- 时间格式默认设置 1: hh:mm:ss
- 时间格式默认设置 2: hh.mm.ss

示例	结果	设置 1	设置 2
time#( A ) 其中 A=09:00:00	字符串:	9:00:00	9:00:00
	数字:	0.375	-
time#( A, 'hh.mm' ) 其中 A=09.00	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。

**Syntax:**

```
timestamp#(text[, format])
```

**Return data type:** 双

**参数:**

参数	说明
text	可以计算文本字符串值。
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：

以下示例假设按照操作系统设置：

- 日期格式默认设置 1: YY-MM-DD
- 日期格式默认设置 2: M/D/YY
- 时间格式默认设置 1: hh:mm:ss
- 时间格式默认设置 2: hh.mm.ss

示例	结果	设置 1	设置 2
timestamp#( A ) 其中 A=8/6/97 09:00:00	字符串：	1997-08-06 9:00:00	1997-08-06 9:00:00
	数字：	-	35648.375
timestamp#( A, 'YYYY-MM-DD hh_mm' ) 其中 A=8/6/97 09_00	字符串：	1997-08-06 09_00	1997-08-06 09_00
	数字：	35648.375	35648.375

## 5.14 内部记录函数

内部记录函数可用于：

- 数据加载脚本(当对当前记录的评估需要一个来自以前加载的数据记录的值时)。
- 图表表达式(当需要来自可视化数据集的其他值时)。



当图表内部记录函数用于任何图表表达式时，按图表 Y 值排序或者按垂直表表达式列排序不可用。因此，这些排序替代项会自动禁用。

当使用这些函数时，会自动禁用零值。

## 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** ( )

## 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 **integer** 函数, 用于查找字段中特殊值的数量。

```
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 [ , expression ]
```

### 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 (fieldname, matchfieldname, matchfieldvalue [, tablename])
```

### Peek

**Peek()** 用于在表格中查找已经加载或内部内存中存在的行的字段值。可以将行号指定为表格。

```
Peek (fieldname [ , row [ , tablename ] ]
```

### 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 (expression )
```

另请参阅：

▢ [范围函数\(第 522 页\)](#)

## 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。

### Syntax:

```
Above ([TOTAL] expr [ , offset [,count]])
```

**Return data type:** 双

**参数：**

参数	说明
expr	表达式或字段包含要度量的数据。

参数	说明
offset	指定 offsetn(大于 0) 后, 将表达式评估从当前行开始向上移动 n 行。  指定 0 偏移量可以计算当前行上的表达式的值。  指定负偏移量数值后, 使 Above 函数效果类似于具有相应正偏移量数值的 Below 函数。
count	通过指定第三个参数 <b>count</b> 大于 1, 函数将返回一连串 <b>count</b> 值, 每个值对应一个从原始单元格开始向上计数的 <b>count</b> 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(第 522 页)
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数, 则当前列段数据总是与整列相等。

在列段数据的第一行中返回 NULL 值, 因为其上没有行。



列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算, 不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度, 或者如果已指定 TOTAL 限定符, 则计算整个表格中的表达式。



如果表格或表格等同物有多个垂直维度, 当前列段数据将只包括值与所有维度列的当前行相同的行, 但按内部字段排序显示最后维度的列除外。

#### 限制:

递归调用将返回 NULL 值。

#### 示例和结果:

##### 示例 1:

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

示例 1 的表格可视化。

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，**Astrida** 的行将显示行 **Betacab + Betacab** (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))` 创建一个值以及根据 **Higher?** 创建一个标有 `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。

### 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算函数的值。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。

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

示例 2 的表格可视化。

### 示例 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。

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 3 的表格可视化。

示例 4:	结果								
<p><b>Above</b> 函数可用作范围函数的输入。例如：RangeAvg (Above(Sum(Sales),1,3))。</p>	<p>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。</p> <p>以 Customer 为维度的表格为 RangeAvg() 表达式提供了以下结果。</p> <table> <tr> <td>Astrida</td><td>-</td></tr> <tr> <td>Betacab</td><td>587</td></tr> <tr> <td>Canutility</td><td>563</td></tr> <tr> <td>Divadip:</td><td>603</td></tr> </table>	Astrida	-	Betacab	587	Canutility	563	Divadip:	603
Astrida	-								
Betacab	587								
Canutility	563								
Divadip:	603								

示例中所使用的数据：

```
Monthnames:
LOAD * INLINE [
Month, 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 (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 '|');

```

要按正确顺序对月份进行排序，在创建可视化后，转到属性面板的**Sorting**部分，选择**Month**，然后勾选复窗格**Sort by expression**。在表达式框中，输入 Monthnumber。

另请参阅：

- *Below* - 图表函数(第 487 页)
- *Bottom* - 图表函数(第 490 页)
- *Top* - 图表函数(第 508 页)
- *RangeAvg*(第 525 页)

## 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。

**Syntax:**

```
Below([TOTAL] expression [ , offset [,count ]])
```

**Return data type:** 双

**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 <b>offsetn</b> 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 <b>Below</b> 函数效果类似于具有相应正偏移量数值的 <b>Above</b> 函数。
count	通过指定第三个参数 <b>count</b> 大于 1，函数将返回一连串 <b>count</b> 值，每个值对应一个从原始单元格开始向下计数的 <b>count</b> 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(第 522 页)
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数，则当前列段数据总是与整列相等。

在列段数据的最后一行中返回 NULL 值，因为该行下面没有其他行。



列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算，不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度，或者如果已指定 **TOTAL** 限定符，则计算整个表格中的表达式。



如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

#### 限制：

递归调用将返回 NULL 值。

#### 示例和结果：

##### 示例 1：

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	-	-	-	-

示例 1 的表格可视化。

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))。



## 5 脚本和图表表达式中的函数

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算函数的值。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。请参阅: **Above** 函数中的示例 2, 了解更多信息。

示例 2:	结果								
<b>Below</b> 函数可用作范围函数的输入。例如: <b>RangeAvg (Below(Sum(Sales),1,3))</b> 。	<p>在 <b>Below()</b> 函数的参数中, 将 <b>offset</b> 设置为 1, 并将 <b>count</b> 设置为 3。函数在列段数据(其中有一行)的当前行正下方的三行上查找表达式 <b>Sum(Sales)</b> 的结果。这三个值用作 <b>RangeAvg()</b> 函数的输入, 用于查找所提供的数字范围中的平均值。</p> <p>以 <b>Customer</b> 为维度的表格为 <b>RangeAvg()</b> 表达式提供了以下结果。</p>								
	<table><tr><td>Astrida</td><td>659.67</td></tr><tr><td>Betacab</td><td>720</td></tr><tr><td>Canutility</td><td>757</td></tr><tr><td>Divadip:</td><td>-</td></tr></table>	Astrida	659.67	Betacab	720	Canutility	757	Divadip:	-
Astrida	659.67								
Betacab	720								
Canutility	757								
Divadip:	-								

示例中所使用的数据:

```
Monthnames:
LOAD * INLINE [
Month, 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 (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 '|');
```

要按正确顺序对月份进行排序，在创建可视化后，转到属性面板的**Sorting**部分，选择**Month**，然后勾选复选框**Sort by expression**。在表达式框中，输入 Monthnumber。

---

另请参阅：

- *Above* - 图表函数(第 483 页)
- *Bottom* - 图表函数(第 490 页)
- *Top* - 图表函数(第 508 页)
- *RangeAvg*(第 525 页)

### 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。

**Syntax:**

```
Bottom([TOTAL] expr [ , offset [,count ]])
```

**Return data type:** 双

**参数：**

参数	说明
expr	表达式或字段包含要度量的数据。
offset	指定 <b>offsetn</b> 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 <b>Bottom</b> 函数效果类似于具有相应正偏移量数值的 <b>Top</b> 函数。
count	通过指定第三个参数 <b>count</b> 大于 1，函数返回的不是一个值，而是一连串 <b>count</b> 值，每个值对应当前列段数据的最后一个 <b>count</b> 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(第 522 页)
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数，则当前列段数据总是与整列相等。



列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算，不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度，或者如果已指定 **TOTAL** 限定符，则计算整个表格中的表达式。



如果表格或表格等同物有多个垂直维度，当前列段数据将只包括值与所有维度列的当前行相同的行，但按内部字段排序显示最后维度的列除外。

#### 限制：

递归调用将返回 NULL 值。

#### 示例和结果：

##### 示例：1

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

示例 1 的表格可视化。

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，因为此值是以下底行的值：**Divadip**。

## 5 脚本和图表表达式中的函数

此表格还显示了更复杂的度量：根据  $\text{Sum}(\text{Sales}) + \text{Bottom}(\text{Sum}(\text{Sales}))$  创建的一个值以及标有 **Bottom offset 3** 的一个值，后者使用表达式  $\text{Sum}(\text{Sales}) + \text{Bottom}(\text{Sum}(\text{Sales}), 3)$  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算函数的值。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  $\text{Bottom}(\text{Sum}(\text{Sales}))$ 。**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。)

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 的第一个表格。End value 度量的 Bottom 值基于 Month (Dec)。

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

示例 2 的第二个表格。End value 度量的 Bottom 值基于 Product (Astrida 的 BB)。

请参阅：**Above** 函数中的示例 2，了解更多信息。

示例：3	结果								
<p><b>Bottom</b> 函数可用作范围函数的输入。例如：<code>RangeAvg (Bottom(Sum(Sales),1,3))</code>。</p>	<p>在 <b>Bottom()</b> 函数的参数中，将 <code>offset</code> 设置为 1，并将 <code>count</code> 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 <b>Sum(Sales)</b> 的结果(因为 <code>offset=1</code>)，并且列段数据(其中有一行)上方有两行。这三个值用作 <code>RangeAvg()</code> 函数的输入，用于查找所提供的数字范围中的平均值。</p> <p>以 <b>Customer</b> 为维度的表格为 <code>RangeAvg()</code> 表达式提供了以下结果。</p>								
	<table> <tr> <td>Astrida</td><td>659.67</td></tr> <tr> <td>Betacab</td><td>659.67</td></tr> <tr> <td>Canutility</td><td>659.67</td></tr> <tr> <td>Divadip:</td><td>659.67</td></tr> </table>	Astrida	659.67	Betacab	659.67	Canutility	659.67	Divadip:	659.67
Astrida	659.67								
Betacab	659.67								
Canutility	659.67								
Divadip:	659.67								

```
Monthnames:
LOAD * INLINE [
Month, 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 (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 '|');
```

要按正确顺序对月份进行排序，在创建可视化后，转到属性面板的**Sorting**部分，选择**Month**，然后勾选复选框**Sort by expression**。在表达式框中，输入 `Monthnumber`。

另请参阅：

□ [Top - 图表函数\(第 508 页\)](#)

## 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。

**Syntax:**

**Column** (ColumnNo)

**Return data type:** 双

**参数:**

参数	说明
ColumnNo	表格中包含度量的列的列数。  <div>  <i>Column() 函数会忽略维度列。</i> </div>

**限制:**

如果 **ColumnNo** 引用没有度量的列，则返回 NULL 值。

递归调用将返回 NULL 值。

**示例和结果:**

**示例: 总销售额百分比**

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	505	29.70
A	AA	16	4	64	505	12.67
A	BB	9	9	81	505	16.04
B	BB	10	5	50	505	9.90
B	CC	20	2	40	505	7.92
B	DD	25	-	0	505	0.00
C	AA	15	8	120	505	23.76
C	CC	19	-	0	505	0.00

**示例: 所选客户的销售额百分比**

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
A	AA	15	10	150	295	50.85
A	AA	16	4	64	295	21.69
A	BB	9	9	81	295	27.46

示例	结果
使用以下表达式将 Order Value 作为度量添加到表格中: $\text{sum}(\text{UnitPrice} * \text{UnitSales})$ 。	根据 Order Value 列获取 Column(1) 的结果, 因为此列是第一个度量列。
使用以下表达式将 Total Sales Value 添加为度量: $\text{sum}(\text{TOTAL UnitPrice} * \text{UnitSales})$	根据 Total Sales Value 获取 Column(2) 的结果, 因为此列是第二个度量列。
使用以下表达式将 % Sales 添加为度量: $100 * \text{Column}(1) / \text{Column}(2)$	请参阅示例 <a href="#">总销售额百分比(第 494 页)</a> 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 <a href="#">所选客户的销售额百分比(第 494 页)</a> 。

#### 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。

#### Syntax:

**Dimensionality ( )**

**Return data type:** 整数

#### 限制:

此函数仅可用于图表。返回除合计以外的所有行的维度数，即 0。对于除透视表之外的所有图表类型，它会返回除总计之外的所有行的维度数，即 0。

**示例：**

dimensionality 的典型应用是：仅当某维度有一个值时您想要进行计算。

示例	结果
<p>对于包含维度 UnitSales 的表格，您可能只想指示某发票已发送：</p> <p>IF(Dimensionality()=3, "Invoiced")。</p>	

### 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。

**Syntax:**

```
Exists (field_name [, expr])
```

**Return data type:** 布尔值

**参数：**

参数	说明
field_name	用于评估要搜索的字段名称的名称或字符串表达式。字段到目前为止必须存在于由脚本加载的数据中。
expr	对字段值进行求值以便查找在 <b>field-name</b> 中指定的字段的表达式。如果省略，指定字段中的当前记录值为假设值。

**示例和结果：**

示例	结果
Exists (Employee)	如果当前记录中的字段值 <b>Employee</b> 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)。
Exists(Employee, 'Bill')	<p>如果在字段 <b>Employee</b> 的当前内容中发现字段值 <b>'Bill'</b>，则返回 -1 (True)。</p> <p>语句 Exists (Employee, Employee) 和 Exists (Employee) 功能相同。</p>



示例	结果								
<pre> Employees: LOAD * inline [ Employee ID Salary Bill 001 20000 John 002 30000 Steve 003 35000 ] (delimiter is ' ');  Citizens: Load * inline [ Name Address Bill New York Mary London Steve Chicago Lucy Paris John Miami ] (delimiter is ' ');  EmployeeAddresses: Load Name as Employee, Address Resident Citizens where Exists (Employee, Name);  Drop Tables Employees, Citizens; </pre>	<p>这将会在数据模型中生成名为 EmployeeAddresses 的表格，可以将该表格看作是使用维度 Employee 和 Address 的表格可视化。</p> <p>where 子句: where Exists (Employee, Name), 是指只能从表格 Citizens 将同时位于 Employees 中的姓名加载到新表格。Drop 语句将删除临时表格 Employees 和 Citizens 以避免混淆。</p> <table> <tr> <th>Employee</th><th>Address</th></tr> <tr> <td>Bill</td><td>New York</td></tr> <tr> <td>John</td><td>Miami</td></tr> <tr> <td>Steve</td><td>Chicago</td></tr> </table>	Employee	Address	Bill	New York	John	Miami	Steve	Chicago
Employee	Address								
Bill	New York								
John	Miami								
Steve	Chicago								
<p>使用以下语句替换前一个示例的样本数据中构建表格 EmployeeAddresses 的的语句(使用 where not Exists)。</p> <pre> NonEmployee: Load Name as Employee, Address Resident Citizens where not Exists (Employee, Name); </pre>	<p>where 子句包括 not: where not Exists (Employee, Name), 是指只能从表格 Citizens 将不在 Employees 中的姓名加载到新表格。</p> <table> <tr> <th>Employee</th><th>Address</th></tr> <tr> <td>Mary</td><td>London</td></tr> <tr> <td>Lucy</td><td>Paris</td></tr> </table>	Employee	Address	Mary	London	Lucy	Paris		
Employee	Address								
Mary	London								
Lucy	Paris								

示例中所使用的数据:

```

Employees:
LOAD * inline [
Employee|ID|Salary
Bill|001|20000
John|002|30000
Steve|003|35000
] (delimiter is '|');

```

```

Citizens:
Load * inline [
Name|Address
Bill|New York
Mary|London
Steve|Chicago
Lucy|Paris
John|Miami
] (delimiter is '|');

```

```
EmployeeAddresses:
```

```
Load Name as Employee, Address Resident Citizens where Exists (Employee, Name);
```

```
Drop Tables Employees, Citizens;
```

### FieldIndex

**FieldIndex()** 用于返回字段 **field\_name**(按加载顺序) 中的字段值 **value** 的位置。

#### Syntax:

```
FieldIndex(field_name , value)
```

**Return data type:** 整数

#### 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	<b>field_name</b> 字段的值。

#### 限制:

如果无法在字段 **field\_name** 的字段值中找到 **value**，则返回 0。

#### 示例和结果:

下例使用以下字段: 表 **Names** 中的 **First name**。

示例	结果
将示例数据添加到应用程序并运行。	加载表格 <b>Names</b> 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 <b>First name</b> 字段的加载顺序中第一个显示。请注意, 在筛选器窗格中, <b>John</b> 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 <b>FieldIndex()</b> 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: 假定表格 <b>Names</b> 已加载, 在以下示例数据中:	

示例	结果
<pre>John1: Load FieldIndex('First name','John') as MyJohnPos Resident Names;</pre>	MyJohnPos=1, 因为“John”在 <b>First name</b> 字段的加载顺序中第一个显示。请注意, 在筛选器窗格中, <b>John</b> 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。
<pre>Peter1: Load FieldIndex('First name','Peter') as MyPeterPos Resident Names;</pre>	MyPeterPos=4, 因为 <b>FieldIndex()</b> 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。

**Syntax:**

```
FieldValue(field_name , elem_no)
```

**Return data type:** 双

**参数:**

参数	说明
field_name	需要值的字段的名称。例如, 表格中的列。必须指定作为字符串值。这意味着必须用单引号将字段名称括起来。
elem_no	按加载顺序返回值的字段的位置(元素)数量。这相当于表格中的行, 但它取决于加载元素(行)的顺序。

**限制:**

如果 **elem\_no** 大于字段值数量，则返回 NULL。

#### 示例和结果：

下例使用以下字段：表 **First name** 中的 **Names**。

示例	结果
将示例数据添加到应用程序并运行。	加载表格 <b>Names</b> 作为样本数据。
图表函数：在包含维度 <b>First name</b> 的表格中，添加作为度量：	
FieldValue('First name','1')	John, 因为 John 在 <b>First name</b> 字段的加载顺序中第一个显示。请注意，在筛选器窗格中， <b>John</b> 将作为从顶部开始的第 2 个值显示在 <b>Jane</b> 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','7')	NULL, 因为 <b>First name</b> 字段中只有 6 个值。
脚本函数：假定表格 <b>Names</b> 已加载，在以下示例数据中：	
John1: Load FieldValue('First name',1) as MyPos1 Resident Names;	MyPos1=John, 因为“John”在 <b>First name</b> 字段的加载顺序中第一个显示。
Peter1: Load FieldValue('First name',7) as MyPos2 Resident Names;	MyPos2s= - (Null), 因为 <b>First name</b> 字段中只有 6 个值。

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 FieldValue('First name',1) as MyPos1
Resident Names;

Peter1:
Load FieldValue('First name',7) as MyPos2
Resident Names;
```

## FieldValueCount

**FieldValueCount()** 是一个 **integer** 函数，用于查找字段中特殊值的数量。

**Syntax:****FieldValueCount**(field\_name)**Return data type:** 整数**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:**

下例使用以下字段: 表 **First name** 中的 **Names**。

示例	结果
将示例数据添加到应用程序并运行。	加载表格 <b>Names</b> 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 <b>Peter</b> 显示两次。
FieldValueCount('Initials')	6, 因为 <b>Initials</b> 只有特殊值。
脚本函数: 加载指定的表格 <b>Names</b> 作为示例数据:	
John1: Load FieldValueCount('First name') as MyFieldCount1 Resident Names;	MyFieldCount1=5, 因为“John”显示两次。
John1: 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:

**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。

#### Syntax:

```
lookup(field_name, match_field_name, match_field_value [, table_name])
```

**Return data type:** 双

#### 参数：

参数	说明
field_name	需要返回值的字段的名称。输入值必须为字符串(例如引用的文字)。
match_field_name	要在其中查找 <b>match_field_value</b> 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 <b>match_field_name</b> 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 <b>table_name</b> ，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。

#### 示例和结果：

示例	结果																														
<p>样本数据使用以下格式的 <b>Lookup()</b> 函数：</p> <pre>Lookup('Category', 'ProductID', ProductID, 'ProductList')</pre> <p>添加示例脚本到应用程序并运行。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。</p> <pre>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</pre>	<p>首先加载 <b>ProductList</b> 表格。</p> <p><b>Lookup()</b> 函数用于构建 <b>OrderData</b> 表格。它将第三个参数指定为 <b>ProductID</b>。这是用于在 <b>ProductList</b> 的第二个参数'<b>ProductID</b>'中查找值的字段，用单引号括起来表示。</p> <p>此函数返回“<b>Category</b>”的值( <b>ProductList</b> 表格中)，然后加载作为 <b>CategoryID</b>。</p> <p><b>drop</b> 语句用于从数据模型删除 <b>ProductList</b> 表格( 因为不再需要)，从而保留含有以下结果的 <b>OrderData</b> 表格：</p> <table><tr><th>ProductID</th><th>InvoiceID</th><th>CustomerID</th><th>Units</th><th>CategoryID</th></tr><tr><td>1</td><td>1</td><td>Astrida</td><td>8</td><td>1</td></tr><tr><td>2</td><td>1</td><td>Astrida</td><td>6</td><td>1</td></tr><tr><td>3</td><td>2</td><td>Betacab</td><td>10</td><td>2</td></tr><tr><td>3</td><td>3</td><td>Divadip</td><td>5</td><td>2</td></tr><tr><td>4</td><td>4</td><td>Divadip</td><td>10</td><td>3</td></tr></table>	ProductID	InvoiceID	CustomerID	Units	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
ProductID	InvoiceID	CustomerID	Units	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																											



*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。

另请参阅：

□ *ApplyMap* ( 第 517 页)

## 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。

**Syntax:**

**NoOfRows** ( [TOTAL] )**Return data type:** 整数**参数:**

参数	说明
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数, 则当前列段数据总是与整列相等。

**示例:**

```
if( RowNo( )= NoOfRows( ), 0, Above( sum( Sales )))
```

**另请参阅:**

▢ [RowNo - 图表函数\(第 328 页\)](#)

## Peek

**Peek()** 用于在表格中查找已经加载或内部内存中存在的行的字段值。可以将行号指定为表格。

**Syntax:**

```
Peek (field_name[, row_no[, table_name ] ])
```

**Return data type:** 双**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 <b>row</b> , 则假定为 -1。
table_name	表格标签不能以冒号结束。如果未指定 <b>table_name</b> , 则假定为当前表格。如果用于 <b>LOAD</b> 语句之外或指向另外一个表格, 则必须包括 <b>table_name</b> 。

**限制:**


在内部表格的首个记录中, 此函数返回 NULL 值。

**示例和结果:**



示例	结果														
<p>添加示例脚本到应用程序并运行。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。</p> <pre>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 ' ');  FirstEmployee: Load EmployeeCode, Peek(EmployeeCode,0) As EmpCode Resident EmployeeDates;</pre>	<p>EmpCode = 101, 因为 Peek(EmployeeCode,0) 返回表格 EmployeeDates 的 EmployeeCode 中的第一个值。</p> <p>替代参数 <b>row_no</b> 返回表格中其他行的值，如下所示：</p> <p>Peek(EmployeeCode,2) 用于返回表格中的第三个值：102。</p> <p>但是，请注意，如果没有将表格指定为第三个参数 <b>table_no</b>，则此函数引用当前表格（在此例中，为内部表格）。Peek(EmployeeCode,-2) 的结果是一个倍数值：</p> <table> <thead> <tr> <th>EmployeeCode</th><th>EmpCode</th></tr> </thead> <tbody> <tr><td>101</td><td>-</td></tr> <tr><td>102</td><td>-</td></tr> <tr><td>103</td><td>101</td></tr> <tr><td>104</td><td>102</td></tr> <tr><td>105</td><td>103</td></tr> <tr><td>106</td><td>104</td></tr> </tbody> </table>	EmployeeCode	EmpCode	101	-	102	-	103	101	104	102	105	103	106	104
EmployeeCode	EmpCode														
101	-														
102	-														
103	101														
104	102														
105	103														
106	104														
<pre>FirstEmployee: Load EmployeeCode, Peek(EmployeeCode,- 2,'EmployeeDates') As EmpCode Resident EmployeeDates;</pre>	<p>通过将参数 <b>table_no</b> 指定为 'EmployeeDates'，此函数返回表格 EmployeeDates 中的 EmployeeCode 的第二个至最后一个值：105。</p>														

示例	结果																																				
<p><b>Peek()</b> 函数可用于引用尚未加载的数据。</p> <p>添加示例脚本到应用程序并运行。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。</p> <pre>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) &amp;', '&amp;value,value) AS List RESIDENT T1 ORDER BY ID ASC; DROP TABLE T1;</pre>	<p>在应用程序的表格中创建维度为 <b>ID</b>、<b>List</b> 和 <b>Value</b> 的表格。</p> <table><tr><th>ID</th><th>List</th><th>Value</th></tr><tr><td>1</td><td>6</td><td>6</td></tr><tr><td>1</td><td>6,3</td><td>3</td></tr><tr><td>1</td><td>6,3,4</td><td>4</td></tr><tr><td>2</td><td>11</td><td>11</td></tr><tr><td>2</td><td>11,10</td><td>10</td></tr><tr><td>2</td><td>11,10,1</td><td>1</td></tr><tr><td>3</td><td>8</td><td>8</td></tr><tr><td>3</td><td>8,7</td><td>7</td></tr><tr><td>5</td><td>13</td><td>13</td></tr><tr><td>5</td><td>13,2</td><td>2</td></tr><tr><td>5</td><td>13,2,78</td><td>78</td></tr></table> <p><b>IF()</b> 语句是根据临时表格 T1 构建。</p> <p>Peek(ID) 引用当前表格 T2 的上一行中的字段 ID。</p> <p>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。</p> <p>结算表达式的语句如下所示：</p> <p>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。</p> <p>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。</p>	ID	List	Value	1	6	6	1	6,3	3	1	6,3,4	4	2	11	11	2	11,10	10	2	11,10,1	1	3	8	8	3	8,7	7	5	13	13	5	13,2	2	5	13,2,78	78
ID	List	Value																																			
1	6	6																																			
1	6,3	3																																			
1	6,3,4	4																																			
2	11	11																																			
2	11,10	10																																			
2	11,10,1	1																																			
3	8	8																																			
3	8,7	7																																			
5	13	13																																			
5	13,2	2																																			
5	13,2,78	78																																			



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。

## 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。

**Syntax:****Previous** (expr)**Return data type:** 双**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 <b>previous()</b> 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。

**示例和结果:**

添加示例脚本到应用程序并运行。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。

示例	结果																																							
<pre>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 ' ');</pre>	<p>通过在 <b>Load</b> 语句中使用 <b>Previous ()</b> 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。</p> <table><tr><th>Month</th><th>Sales</th><th>Increase</th></tr><tr><td>1</td><td>12</td><td>-</td></tr><tr><td>2</td><td>13</td><td>1</td></tr><tr><td>3</td><td>15</td><td>2</td></tr><tr><td>4</td><td>17</td><td>2</td></tr><tr><td>5</td><td>21</td><td>4</td></tr><tr><td>6</td><td>21</td><td>0</td></tr><tr><td>7</td><td>22</td><td>1</td></tr><tr><td>8</td><td>23</td><td>1</td></tr><tr><td>9</td><td>32</td><td>9</td></tr><tr><td>10</td><td>35</td><td>3</td></tr><tr><td>11</td><td>40</td><td>5</td></tr><tr><td>12</td><td>41</td><td>1</td></tr></table>	Month	Sales	Increase	1	12	-	2	13	1	3	15	2	4	17	2	5	21	4	6	21	0	7	22	1	8	23	1	9	32	9	10	35	3	11	40	5	12	41	1
Month	Sales	Increase																																						
1	12	-																																						
2	13	1																																						
3	15	2																																						
4	17	2																																						
5	21	4																																						
6	21	0																																						
7	22	1																																						
8	23	1																																						
9	32	9																																						
10	35	3																																						
11	40	5																																						
12	41	1																																						

## Top - 图表函数

**Top()** 用于评估表格中列段数据第一(顶部)行的表达式。要计算的行取决于 **offset** 值, 如果存在, 则默认计算顶行。对于除表格以外的图表, **Top()** 用于计算图表的等效垂直表中当前列的第一行的值。

### Syntax:

```
Top([TOTAL] expr [ , offset [,count ]])
```

### Return data type: 双

### 参数:

参数	说明
expr	表达式或字段包含要度量的数据。
offset	指定 <b>offset</b> 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 <b>Top</b> 函数效果类似于具有相应正偏移量数值的 <b>Bottom</b> 函数。
count	通过指定第三个参数 <b>count</b> 大于 1, 函数将返回一连串 <b>count</b> 值, 每个值对应当前列段数据的最后一个 <b>count</b> 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。 <i>范围函数(第 522 页)</i>
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数, 则当前列段数据总是与整列相等。



列段数据是指按当前排序顺序拥有与维度相同的值的连续单元格子集。内部记录图表函数在列段数据中进行计算, 不包括等效垂直表图表中的最右侧维度。如果图表中只有一个维度, 或者如果已指定 **TOTAL** 限定符, 则计算整个表格中的表达式。



如果表格或表格等同物有多个垂直维度, 当前列段数据将只包括值与所有维度列的当前行相同的行, 但按内部字段排序显示最后维度的列除外。

### 限制:

递归调用将返回 NULL 值。

### 示例和结果:

#### 示例: 1

## 5 脚本和图表表达式中的函数

Top and Bottom					
Customer	Q	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

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，因为此值是以下顶行的值：**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：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算函数的值。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。

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 的第一个表格。First value 度量的 Top 值基于 Month (Jan)。

Customer	Product	Month	Sum(Sales)	Firstvalue
			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

示例 2 的第二个表格。Firstvalue 度量的 Top 值基于 Product( Astrida 的 AA)。

请参阅：**Above** 函数中的示例 2，了解更多信息。

示例：3	结果								
<p><b>Top</b> 函数可用作范围函数的输入。例如：RangeAvg (Top(Sum(Sales),1,3))。</p>	<p>在 <b>Top()</b> 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 <b>Sum(Sales)</b> 的结果 (因为 offset=1)，并且(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。</p> <p>以 <b>Customer</b> 为维度的表格为 RangeAvg() 表达式提供了以下结果。</p>								
	<table> <tr> <td>Astrida</td><td>603</td></tr> <tr> <td>Betacab</td><td>603</td></tr> <tr> <td>Canutility</td><td>603</td></tr> <tr> <td>Divadip:</td><td>603</td></tr> </table>	Astrida	603	Betacab	603	Canutility	603	Divadip:	603
Astrida	603								
Betacab	603								
Canutility	603								
Divadip:	603								

```

Monthnames:
LOAD * INLINE [
Month, 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 (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 '|');
```

要按正确顺序对月份进行排序，在创建可视化后，转到属性面板的**Sorting**部分，选择**Month**，然后勾选复选框**Sort by expression**。在表达式框中，输入 Monthnumber。

另请参阅：

- *Bottom* - 图表函数(第 490 页)
- *Above* - 图表函数(第 483 页)
- *Sum* - 图表函数(第 161 页)
- *RangeAvg*(第 525 页)
- 范围函数(第 522 页)

### 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。

**Syntax:**

```
secondarydimensionality ( )
```

**Return data type:** 整数

**secondarydimensionality** 函数用于透视表之外时始终返回 0。

### After - 图表函数

**After()** 用于返回使用透视表的维度值评估的表达式值，因为维度值显示在透视表的行段内当前列之后的列。

**Syntax:**

```
after([ total ] expression [ , offset [,n ]])
```



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
expression	表达式或字段包含要度量的数据。
offset	指定 <b>offsetn</b> (大于 1) 后, 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 <b>After</b> 函数效果类似于具有相应正偏移量数值的 <b>Before</b> 函数。
n	通过指定第三个参数 n 大于 1, 函数将返回一连串 n 值, 每个值对应一个从原始单元格开始向右计数的 n 表格行。
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 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。

#### Syntax:

```
before([ total ] expression [ , offset [,n ]])
```



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
expression	表达式或字段包含要度量的数据。



参数	说明
offset	指定 <b>offsetn</b> (大于 1) 后, 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 <b>Before</b> 函数效果类似于具有相应正偏移量数值的 <b>After</b> 函数。
n	通过指定第三个参数 n 大于 1, 函数将返回一连串 n 值, 每个值对应一个从原始单元格开始向左计数的 n 表格行。
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 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。

#### Syntax:

```
first([ total ] expression [ , offset [,n ]])
```

#### 参数:

参数	说明
expression	表达式或字段包含要度量的数据。
offset	指定 <b>offsetn</b> (大于 1) 后, 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 <b>First</b> 函数效果类似于具有相应正偏移量数值的 <b>Last</b> 函数。
n	通过指定第三个参数 n 大于 1, 函数将返回一连串 n 值, 每个值对应一个从原始单元格开始向右计数的 n 表格行。

参数	说明
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 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。

**Syntax:**

```
last([ total ] expression [ , offset [,n ]])
```

**参数：**

参数	说明
expression	表达式或字段包含要度量的数据。
offset	指定 <b>offsetn</b> (大于 1) 后，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 <b>First</b> 函数效果类似于具有相应正偏移量数值的 <b>Last</b> 函数。
n	通过指定第三个参数 n 大于 1，函数将返回一连串 n 值，每个值对应一个从原始单元格开始向左计数的 n 表格行。
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 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。

**Syntax:**

```
ColumnNo([total])
```

**参数:**

参数	说明
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数, 则当前列段数据总是与整列相等。

如果透视表有多个水平维度, 则当前行片断将只包括值与所有维度行中当前列相同的列, 除显示字段排序间上一次水平维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义。

**示例:**

```
if( ColumnNo( )=1, 0, sum( Sales ) / before( sum( Sales )))
```

### NoOfColumns - 图表函数

**NoOfColumns()** 用于返回透视表的当前行段中的列数。

**Syntax:**

```
NoOfColumns([total])
```

**参数:**

参数	说明
TOTAL	如果表格是单维度或如果将 <b>TOTAL</b> 限定符用作参数, 则当前列段数据总是与整列相等。

如果透视表有多个水平维度, 则当前行片断将只包括值与所有维度行中当前列相同的列, 除显示字段排序间上一次维度的行之外。透视表的水平维度的内部字段排序只需依据从上至下的维度顺序定义。

**示例:**

```
if( ColumnNo( )=NoOfColumns( ), 0, after( sum( Sales )))
```

## 5.15 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：

Value	IsNum(Value)	IsText(Value)
23	-1	0
Green	0	-1
Blue	0	-1
12	-1	0
33Red	0	-1

## 5.16 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** 脚本函数用于将任何表达式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** 脚本函数用于将任何表达式映射至先前加载的映射表。

### Syntax:

```
ApplyMap('map_name', expression [ , default_mapping ] )
```

**Return data type:** 双

### 参数:

参数	说明
map_name	以前通过 <b>mapping load</b> 或 <b>mapping select</b> 语句创建的映射表的名称。该名称必须用单直引号引起来。
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。

### 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：
```

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。

### Syntax:

```
MapSubstring('map_name', expression)
```

**Return data type:** 字符串

### 参数：

参数	说明
map_name	先前在 <b>mapping load</b> 或 <b>mapping select</b> 语句中读取的映射表的名称。名称必须用直的单引号括起来。
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：

Model	Description
Twixie	Red Cotton Small
Boomer	Blue Polyester Large
Raven	Yellow Polyester Medium
Seedling	Red Cotton Large
SeedlingPlus	Red Cotton Large with hood
Younger	Blue Cotton with patch
MultiStripe	Red Yellow Blue Cotton Small/Medium/Large

### 5.17 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  $\pi$  (3.14159...) 的值。

```
pi( )
```

**rand**

该函数返回 0 与 1 之间的随机数字。并且，该函数也可用于创建样本数据。

```
rand( )
```

**示例：**

以下示例脚本用于创建拥有 1000 条记录且包含随机选择的大写字符的表格，即范围为 65 至 91 (65+26) 的字符。

```
Load  
    Chr( Floor(rand() * 26) + 65) as UCaseChar,  
    RecNo() as ID  
Autogenerate 1000;
```

**true**

返回一个对偶值，文本值 'True' 和数值 "-1"，可以用作表达式的逻辑真。

```
true( )
```

### 5.18 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。

**Null**

**Null** 函数用于返回 NULL 值。

```
NULL( )
```

**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 )
```

## 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)。

### Syntax:

```
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。

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T
2	-	-	T
3	Value	Value	F

## NULL

**Null** 函数用于返回 NULL 值。

### Syntax:

**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。

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T
2	-	-	T
3	Value	Value	F

## 5.19 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 **dual** 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()** 用于返回值系列的总和。将所有非数值视为 0, 不同于 + 运算符。

```
RangeSum(first_expr[, Expression])
```

### 计数器范围函数

RangeCount

**RangeCount()** 用于返回在指定规范或表达式内找到的值、文本和数字的数量。

```
RangeCount(first_expr[, Expression])
```

RangeMissingCount

**RangeMissingCount()** 用于查找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()** 用于返回按数值数量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付款(负值)和收入(正值)的投资的净现值。结果拥有一个 **money** 的默认数字格式。

```
RangeNPV (discount_rate, value[, value][, Expression])
```

### RangeXIRR

**RangeXIRR()** 用于返回现金流计划表的内部回报率(不必是周期性的)。要计算一系列周期性现金流的内部回报率,请使用 **RangeIRR** 函数。

```
RangeXIRR (values, dates[, Expression])
```

### RangeXNPV

**RangeXNPV()** 用于返回现金流计划表的净现值(不必是周期性的)。结果默认采用货币数字格式。要计算一系列周期性现金流的净现值,请使用 **RangeNPV** 函数。

```
RangeXNPV (discount_rate, values, dates[, Expression])
```

另请参阅:

□ [内部记录函数\(第 480 页\)](#)

## 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。

**Syntax:**

```
RangeAvg (first_expr[, Expression])
```

**Return data type:** 数字

**参数:**

该函数的参数表达式可能包含内部记录函数和第三可选参数,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。

**限制:**

如果找不到任何数值,则返回 NULL 值。

**示例和结果:**

示例	结果														
RangeAvg (1,2,4)	返回 2.33333333														
RangeAvg (1,'xyz')	返回 1														
RangeAvg (null( ), 'abc')	返回 NULL														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> 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 ]; </pre>	<p>结果列表显示了为表格中的每条记录返回的 MyRangeAvg 值。</p> <table> <thead> <tr> <th>RangeID</th><th>MyRangeAvg</th></tr> </thead> <tbody> <tr><td>1</td><td>7</td></tr> <tr><td>2</td><td>4</td></tr> <tr><td>3</td><td>6</td></tr> <tr><td>4</td><td>12.666</td></tr> <tr><td>5</td><td>6.333</td></tr> <tr><td>6</td><td>5</td></tr> </tbody> </table>	RangeID	MyRangeAvg	1	7	2	4	3	6	4	12.666	5	6.333	6	5
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。

MyField	RangeAvg (Above(MyField,0,3))	
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	6.666666667	等于 RangeAvg(10,2,8)
18	9.333333333	
5	10.333333333	
9	10.666666667	

```

RangeTab:
LOAD * INLINE [
MyField
10
2

```

```
8
18
5
9
] ;
```

另请参阅：

- ▢ [Avg - 图表函数\(第 193 页\)](#)
- ▢ [Count - 图表函数\(第 165 页\)](#)

## 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。

**Syntax:**

```
RangeCorrel(x_values , y_values[, Expression])
```

**Return data type:** 数字

如果手动提供值，则将其输入为 (x,y) 对。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	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 <b>x-value</b> 或一系列 <b>y-values</b> 。
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 值和缺失值都忽略不计。

**示例和结果：**

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	结果										
<p>添加示例脚本到应用程序并运行。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。</p> <pre> 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 ' '); </pre>	<p>在以 ID1 作维度，以 RangeCorrel (x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6)) 作度量的表中，<b>RangeCorrel()</b> 函数在六个 x,y 函数中查找 <b>Correl</b> 值，以查找每个 ID1 值。</p> <table> <tr> <th>ID1</th><th>MyRangeCorrel</th></tr> <tr> <td>01</td><td>-0.9517</td></tr> <tr> <td>02</td><td>-0.5209</td></tr> <tr> <td>03</td><td>-0.5209</td></tr> <tr> <td>04</td><td>-0.1599</td></tr> </table>	ID1	MyRangeCorrel	01	-0.9517	02	-0.5209	03	-0.5209	04	-0.1599
ID1	MyRangeCorrel										
01	-0.9517										
02	-0.5209										
03	-0.5209										
04	-0.1599										
<pre> XY: LOAD recno() as RangeID, * Inline [ X Y 2 3 6 8 9 4 8 5 ](delimiter is ' '); </pre>	<p>在以 RangeID 作维度，以 RangeCorrel(Below (X,0,4,BelowY,0,4)) 作度量的表中，<b>RangeCorrel()</b> 函数使用 <b>Below()</b> 函数的结果，因为第三参数 (count) 设置为4，可从已加载的表XY 生成一系列四个对 x-y 值。</p> <table> <tr> <th>RangeID</th><th>MyRangeCorrel2</th></tr> <tr> <td>01</td><td>0.2492</td></tr> <tr> <td>02</td><td>-0.9959</td></tr> <tr> <td>03</td><td>-1.0000</td></tr> <tr> <td>04</td><td>-</td></tr> </table> <p>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() 函数生成的系列值为：(6,8,9,4,8,5)、(9,4,8,5) 和 (8,5)，最后一个生成空结果。</p>	RangeID	MyRangeCorrel2	01	0.2492	02	-0.9959	03	-1.0000	04	-
RangeID	MyRangeCorrel2										
01	0.2492										
02	-0.9959										
03	-1.0000										
04	-										

另请参阅：

▢ [Correl - 图表函数\(第 196 页\)](#)

## RangeCount

**RangeCount()** 用于返回在指定规范或表达式内找到的值、文本和数字的数量。

**Syntax:**



**RangeCount** (first\_expr[, Expression])

**Return data type:** 整数

**参数:**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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。

**限制:**

不对 NULL 值计数。

**示例和结果:**

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														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> 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 ]; </pre>	<p>结果列表显示了为表格中的每条记录返回的 MyRangeCount 值。</p> <table> <thead> <tr> <th>RangeID</th><th>MyRangeCount</th></tr> </thead> <tbody> <tr><td>1</td><td>3</td></tr> <tr><td>2</td><td>3</td></tr> <tr><td>3</td><td>3</td></tr> <tr><td>4</td><td>3</td></tr> <tr><td>5</td><td>3</td></tr> <tr><td>6</td><td>3</td></tr> </tbody> </table>	RangeID	MyRangeCount	1	3	2	3	3	3	4	3	5	3	6	3
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()** 函数的第二个和第三个参数作为 3，它会返回当前行上方三个字段中的值，如果拥有足够的行，会将其作为 **RangeSum()** 函数的输入。

**示例中所使用的数据:**

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
];
```

另请参阅：

▢ [Count - 图表函数\(第 165 页\)](#)

## 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接近排行之间的线性插值。*

**Syntax:**

```
RangeFractile(fractile, first_expr[, Expression])
```

**Return data type:** 数字

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，并在其内部返回一系列值。

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。														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> 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 ]; </pre>	<p>结果列表显示了为表格中的每条记录返回的 MyRangeFrac 值。</p> <table> <tr> <th>RangeID</th><th>MyRangeFrac</th></tr> <tr><td>1</td><td>6</td></tr> <tr><td>2</td><td>4.5</td></tr> <tr><td>3</td><td>8</td></tr> <tr><td>4</td><td>11</td></tr> <tr><td>5</td><td>5</td></tr> <tr><td>6</td><td>4</td></tr> </table>	RangeID	MyRangeFrac	1	6	2	4.5	3	8	4	11	5	5	6	4
RangeID	MyRangeFrac														
1	6														
2	4.5														
3	8														
4	11														
5	5														
6	4														

**示例和结果:**

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

**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行,这相当于 RangeFractile(0.5, 3,4,6),即为值系列 3、4 和 6 计算分位数 0.5。对于下面表格中的前两行,如果当前行上方没有任何行,则相应地减少范围中值的数量。将会为其他内部记录函数产生类似的结果。

**示例中所使用的数据:**

MyField	RangeFractile(0.5, Above(Sum(MyField),0,3))
1	1
2	1.5

MyField	RangeFractile(0.5, Above(Sum(MyField),0,3))
3	2
4	3
6	4

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
1
2
3
4
5
6
];
```

另请参阅：

▢ [Above - 图表函数\(第 483 页\)](#)

▢ [Fractile - 图表函数\(第 199 页\)](#)

## RangeIRR

**RangeIRR()** 用于返回按数值数量表示的一系列现金流的内部回报率。

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。

**Syntax:**

```
RangeIRR (value[, value][, Expression])
```

**Return data type:** 数字

**参数：**

参数	说明
value	由内部记录函数和第三个可选参数返回的单个值或一系列值。计算此函数至少需要一个正值和一个负值。
Expression	可选表达式或字段包含要度量的数据范围。

**限制：**

文本值，NULL 值和缺失值都忽略不计。

示例	结果														
RangeIRR(-70000,12000,15000,18000,21000,26000)	返回 0,0866														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> 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 ' '); </pre>	<p>结果列表显示了为表格中的每条记录返回的 RangeIRR 值。</p> <table> <thead> <tr> <th>RangeID</th><th>RangeIRR</th></tr> </thead> <tbody> <tr><td>1</td><td>5.000</td></tr> <tr><td>2</td><td>0.8708</td></tr> <tr><td>3</td><td>-</td></tr> <tr><td>4</td><td>5.8419</td></tr> <tr><td>5</td><td>0.9318</td></tr> <tr><td>6</td><td>-0.2566</td></tr> </tbody> </table>	RangeID	RangeIRR	1	5.000	2	0.8708	3	-	4	5.8419	5	0.9318	6	-0.2566
RangeID	RangeIRR														
1	5.000														
2	0.8708														
3	-														
4	5.8419														
5	0.9318														
6	-0.2566														

另请参阅：

□ [内部记录函数\(第 480 页\)](#)

## RangeKurtosis

**RangeKurtosis()** 用于返回与数值范围的峰度对应的值。

**Syntax:**

```
RangeKurtosis(first_expr[, Expression])
```

**Return data type:** 数字

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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。

**限制：**

如果找不到任何数值，则返回 NULL 值。

**示例和结果：**

示例	结果
RangeKurtosis (1,2,4,7)	返回 -0.28571428571429

**另请参阅：**

▢ *Kurtosis - 图表函数(第 202 页)*

## RangeMax

**RangeMax()** 用于返回在表达式或字段内找到的最高数值。

**Syntax:**

```
RangeMax (first_expr[, Expression])
```

**Return data type:** 数字

**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：**

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	结果														
<p>添加示例脚本到应用程序并运行。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。</p> <pre>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 ];</pre>	<p>结果列表显示了为表格中的每条记录返回的 MyRangeMax 值。</p> <table> <thead> <tr> <th>RangeID</th><th>MyRangeMax</th></tr> </thead> <tbody> <tr><td>1</td><td>10</td></tr> <tr><td>2</td><td>7</td></tr> <tr><td>3</td><td>8</td></tr> <tr><td>4</td><td>18</td></tr> <tr><td>5</td><td>9</td></tr> <tr><td>6</td><td>9</td></tr> </tbody> </table>	RangeID	MyRangeMax	1	10	2	7	3	8	4	18	5	9	6	9
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。

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。

**Syntax:**

```
RangeMaxString(first_expr[, Expression])
```

**Return data type:** 字符串

**参数:**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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
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。

MyField	RangeMaxString(Above(MaxString(MyField),0,3))
10	10



MyField	RangeMaxString(Above(MaxString(MyField),0,3))
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
] ;
```

另请参阅：

▢ [MaxString - 图表函数 \(第 301 页\)](#)

## RangeMin

**RangeMin()** 用于返回在表达式或字段内找到的最低数值。

**Syntax:**

```
RangeMin (first_expr[, Expression])
```

**Return data type:** 数字

**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：

示例	结果														
<code>RangeMin (1,2,4)</code>	返回 1														
<code>RangeMin (1,'xyz')</code>	返回 1														
<code>RangeMin (null( ), 'abc')</code>	返回 NULL														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> 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 ]; </pre>	<p>结果列表显示了为表格中的每条记录返回的 <b>MyRangeMin</b> 值。</p> <table> <thead> <tr> <th>RangeID</th><th>MyRangeMin</th></tr> </thead> <tbody> <tr><td>1</td><td>5</td></tr> <tr><td>2</td><td>2</td></tr> <tr><td>3</td><td>2</td></tr> <tr><td>4</td><td>9</td></tr> <tr><td>5</td><td>5</td></tr> <tr><td>6</td><td>2</td></tr> </tbody> </table>	RangeID	MyRangeMin	1	5	2	2	3	2	4	9	5	5	6	2
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：

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
] ;
```

另请参阅：

□ [Min - 图表函数\(第 153 页\)](#)

## 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。

**Syntax:**

```
RangeMinString(first_expr[, Expression])
```

**Return data type:** 字符串

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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
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。

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
];
```

另请参阅：

□ [MinString - 图表函数\(第 303 页\)](#)

## RangeMissingCount

**RangeMissingCount()** 用于查找表达式或字段中非数字值(包括 NULL)的数量。

**Syntax:**

```
RangeMissingCount(first_expr[, Expression])
```

**Return data type:** 整数

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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
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。

**示例中所使用的数据：**

禁用 **MyField** 排序可确保示例符合预期。

MyField	RangeMissingCount(Above(MinString(MyField),0,3))
2	返回 2, 因为此行上面没有行, 因此 3 个值中缺失 2 个。
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")。
1	返回 1, 因为 3 行中有 1 行包含非数字值 ("abc")。
2	返回 2, 因为 3 行中有 2 行包含非数字值 ("def"和"abc")。
2	返回 2, 因为 3 行中有 2 行包含非数字值 (" xyz"和"def")。
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
];
```

另请参阅：

▢ *MissingCount* - 图表函数(第 168 页)

## RangeMode

**RangeMode()** 用于查找表达式或字段中最常出现的值(即模式值)。

**Syntax:**

```
RangeMode (first_expr {, Expression})
```

**Return data type:** 数字

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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。

**限制：**

如果多个值共享最高频率，则返回 NULL。

**示例和结果：**

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	结果														
<p>添加示例脚本到应用程序并运行。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。</p> <pre>RangeTab3: LOAD recno() as RangeID, RangeMax(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 ];</pre>	<p>结果列表显示了为表格中的每条记录返回的 MyRangeMode 值。</p> <table> <thead> <tr> <th>RangeID</th><th>MyRangeMode</th></tr> </thead> <tbody> <tr><td>1</td><td>-</td></tr> <tr><td>2</td><td>-</td></tr> <tr><td>3</td><td>8</td></tr> <tr><td>4</td><td>-</td></tr> <tr><td>5</td><td>5</td></tr> <tr><td>6</td><td>-</td></tr> </tbody> </table>	RangeID	MyRangeMode	1	-	2	-	3	8	4	-	5	5	6	-
RangeID	MyRangeMode														
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。

MyField	RangeMode(Above(MyField,0,3))
10	返回 10，因为上面没有行，因此单个值是最常出现的。
2	-
8	-
18	-
5	-

MyField	RangeMode(Above(MyField,0,3))
18	18
7	-
9	-

另请参阅：

□ [Mode - 图表函数\(第 156 页\)](#)

## RangeNPV

**RangeNPV()** 用于返回基于折扣率和一系列未来付款(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。

**Syntax:**

```
RangeNPV(discount_rate, value[,value][, Expression])
```

**Return data type:** 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。每个值都可能都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														
<p>添加示例脚本到应用程序并运行。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。</p> <pre>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 ' ');</pre>	<p>结果列表显示了为表格中的每条记录返回的 RangeNPV 值。</p> <table> <thead> <tr> <th>RangeID</th><th>RangeNPV</th></tr> </thead> <tbody> <tr><td>1</td><td>\$-49.13</td></tr> <tr><td>2</td><td>\$777.78</td></tr> <tr><td>3</td><td>\$98.77</td></tr> <tr><td>4</td><td>\$25.51</td></tr> <tr><td>5</td><td>\$250.83</td></tr> <tr><td>6</td><td>\$20.40</td></tr> </tbody> </table>	RangeID	RangeNPV	1	\$-49.13	2	\$777.78	3	\$98.77	4	\$25.51	5	\$250.83	6	\$20.40
RangeID	RangeNPV														
1	\$-49.13														
2	\$777.78														
3	\$98.77														
4	\$25.51														
5	\$250.83														
6	\$20.40														

另请参阅：

□ [内部记录函数\(第 480 页\)](#)

## RangeNullCount

**RangeNullCount()** 用于查找表达式或字段中 NULL 值的数量。

**Syntax:**

```
RangeNullCount (firstexpr [, Expression])
```

**Return data type:** 整数

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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
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。

示例中所使用的数据：



在以下示例中复制 **MyField** 不会导致出现 NULL 值。

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。
null	返回 1, 因为当前行为 NULL 值。
'xyz'	返回 1, 因为上面的行为 NULL 值。
9	返回 1, 因为当前行上面两行值为 NULL 值。

示例中所使用的数据：

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
null()
'xyz'
9
] ;
```

另请参阅：

□ [NullCount - 图表函数 \(第 170 页\)](#)

## RangeNumericCount

**RangeNumericCount()** 用于查找表达式或字段中数字值的数量。

**Syntax:**

```
RangeNumericCount (first_expr[, Expression])
```

**Return data type:** 整数

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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
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。

示例中所使用的数据：



禁用 **MyField** 排序可确保示例符合预期。

MyField	RangeNumericCount(Above(MaxString(MyField),0,3))
10	1
abc	1
8	2
def	1
xyz	1
9	1

另请参阅：

▢ [NumericCount - 图表函数\(第 173 页\)](#)

## RangeOnly

**RangeOnly()** 是一个 **dual** 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。

### Syntax:

```
RangeOnly (first_expr[, Expression])
```

**Return data type:** 双

### 参数:

该函数的参数表达式可能包含内部记录函数和第三可选参数，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

### 另请参阅:

▢ [Only - 图表函数\(第 159 页\)](#)

## RangeSkew

**RangeSkew()** 用于返回与数值范围的偏度对应的值。

### Syntax:

```
RangeSkew (first_expr[, Expression])
```

**Return data type:** 数字

### 参数:

该函数的参数表达式可能包含内部记录函数和第三可选参数，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。

### 限制：

如果找不到任何数值，则返回 NULL 值。

### 示例和结果：

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：

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
] ;

```

另请参阅：

▢ [Skew - 图表函数\(第 225 页\)](#)

## RangeStdev

**RangeStdev()** 用于查找数字系列的标准偏差。

**Syntax:**

```
RangeStdev(first_expr[, Expression])
```

**Return data type:** 数字

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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。

**限制：**

如果找不到任何数值，则返回 NULL 值。

**示例和结果：**

示例	结果
RangeStdev (1,2,4)	返回 1.5275252316519
RangeStdev (null( ))	返回 NULL
RangeStdev (above (SalesValue),0,3))	返回从当前行与当前行上两行中计算的 above() 函数返回的三个值的范围的滑动标准。

示例中所使用的数据：

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
] ;
```

另请参阅：

▢ [Stdev - 图表函数 \(第 228 页\)](#)

## RangeSum

**RangeSum()** 用于返回值系列的总和。将所有非数值视为 0，不同于 + 运算符。

**Syntax:**

```
RangeSum (first_expr[, Expression])
```

**Return data type:** 数字

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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。

**限制：**

与 + 运算符不同，**RangeSum** 函数将所有非数字值视为 0。

**示例和结果：**

示例	结果														
<code>RangeSum (1,2,4)</code>	返回 7														
<code>RangeSum (5,'abc')</code>	返回 5														
<code>RangeSum (null( ))</code>	返回 0														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> 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 ]; </pre>	<p>结果列表显示了为表格中的每条记录返回的 <b>MyRangeSum</b> 值。</p> <table> <thead> <tr> <th>RangeID</th><th>MyRangeSum</th></tr> </thead> <tbody> <tr><td>1</td><td>21</td></tr> <tr><td>2</td><td>12</td></tr> <tr><td>3</td><td>18</td></tr> <tr><td>4</td><td>38</td></tr> <tr><td>5</td><td>19</td></tr> <tr><td>6</td><td>15</td></tr> </tbody> </table>	RangeID	MyRangeSum	1	21	2	12	3	18	4	38	5	19	6	15
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。

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
] ;
```

另请参阅：

- ▢ [Sum - 图表函数\(第 161 页\)](#)
- ▢ [Above - 图表函数\(第 483 页\)](#)

## RangeTextCount

**RangeTextCount()** 用于返回表达式或字段中文本值的数量。

**Syntax:**

```
RangeTextCount (first_expr[, Expression])
```

**Return data type:** 整数

**参数：**

该函数的参数表达式可能包含内部记录函数和第三可选参数，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
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。

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
];
```

另请参阅：

□ [TextCount - 图表函数\(第 175 页\)](#)

## RangeXIRR

**RangeXIRR()** 用于返回现金流计划表的内部回报率(不必是周期性的)。要计算一系列周期性现金流的内部回报率，请使用 **RangeIRR** 函数。

**Syntax:**

```
RangeXIRR(values, dates[, Expression])
```

**Return data type:** 数字

**参数：**

参数	说明
values	对应付款日期计划表的现金流或一系列现金流。每个值都可能由内部记录函数和第三个可选参数返回的单个值或一系列值。系列值必须至少包含一个正值和一个负值。
dates	对应现金流支付的付款日期或付款日期计划表。
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。

所有付款全年折扣。

示例	结果														
RangeXIRR(-2500,'2008-01-01',2750,'2008-09-01')	返回 0.1532														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> RangeTab3: LOAD *, recno() as RangeID, RangeXIRR(Field1,Field2,Field3) as RangeXIRR; 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 ' '); </pre>	<p>结果列表显示了为表格中的每条记录返回的 RangeXIRR 值。</p> <table> <thead> <tr> <th>RangeID</th><th>RangeXIRR</th></tr> </thead> <tbody> <tr><td>1</td><td>-</td></tr> <tr><td>2</td><td>0.5893</td></tr> <tr><td>3</td><td>0.5089</td></tr> <tr><td>4</td><td>0.4476</td></tr> <tr><td>5</td><td>0.4476</td></tr> <tr><td>6</td><td>2.5886</td></tr> </tbody> </table>	RangeID	RangeXIRR	1	-	2	0.5893	3	0.5089	4	0.4476	5	0.4476	6	2.5886
RangeID	RangeXIRR														
1	-														
2	0.5893														
3	0.5089														
4	0.4476														
5	0.4476														
6	2.5886														

**另请参阅：**

▢ [RangeIRR \(第 532 页\)](#)

## RangeXNPV

**RangeXNPV()** 用于返回现金流计划表的净现值(不必是周期性的)。结果默认采用货币数字格式。要计算一系列周期性现金流的净现值，请使用 **RangeNPV** 函数。

**Syntax:**

```
RangeXNPV(discount_rate, values, dates[, Expression])
```

**Return data type:** 数字

**参数：**

参数	说明
discount_rate	每周期的利率。
values	对应付款日期计划表的现金流或一系列现金流。每个值都可能由内部记录函数和第三个可选参数返回的单个值或一系列值。系列值必须至少包含一个正值和一个负值。
dates	对应现金流支付的付款日期或付款日期计划表。

**限制：**

文本值，NULL 值和缺失值都忽略不计。

所有付款全年折扣。

示例	结果														
RangeXNPV(0.1, -2500, '2008-01-01', 2750, '2008-09-01')	返回 80.25														
<p>添加示例脚本到应用程序并运行。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。</p> <pre> RangeTab3: LOAD *, recno() as RangeID, RangeXNPV(Field1,Field2,Field3) as RangeNPV; LOAD * INLINE [ Field1 Field2 Field3 10 5 -6000 2 NULL 7000 8 'abc' 8000 18 11 9000 5 5 9000 9 4 2000 ] (delimiter is ' '); </pre>	<p>结果列表显示了为表格中的每条记录返回的 RangeXNPV 值。</p> <table> <thead> <tr> <th>RangeID</th><th>RangeXNPV</th></tr> </thead> <tbody> <tr> <td>1</td><td>\$-49.13</td></tr> <tr> <td>2</td><td>\$777.78</td></tr> <tr> <td>3</td><td>\$98.77</td></tr> <tr> <td>4</td><td>\$25.51</td></tr> <tr> <td>5</td><td>\$250.83</td></tr> <tr> <td>6</td><td>\$20.40</td></tr> </tbody> </table>	RangeID	RangeXNPV	1	\$-49.13	2	\$777.78	3	\$98.77	4	\$25.51	5	\$250.83	6	\$20.40
RangeID	RangeXNPV														
1	\$-49.13														
2	\$777.78														
3	\$98.77														
4	\$25.51														
5	\$250.83														
6	\$20.40														

## 5.20 图表中的排名函数

这些函数只可用于图表表达式中。



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]])
```

### 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。

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。

#### Syntax:

```
Rank ([TOTAL [<fld {, fld}>]] expr[, mode[, fmt]])
```

**Return data type:** 双

#### 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	<p>如果图表是一维或如果表达式前面有 <b>TOTAL</b> 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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

排行以双值形式返回，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：

值	说明
0(默认)	<p>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。</p> <p>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。</p> <p>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。</p>
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:

值	说明
0(默认)	全部行中的低值 - 高值(如 3-4)。
1	全部行的高值。
2	第一行的低值,以后各行都为空白。

**mode 4** 和 **fmt 2** 的行顺序由图表维度的排序决定。

**示例和结果:**

使用维度 **Product** 和 **Sales** 创建两个可视化,使用 **Product** 和 **UnitSales** 创建其他可视化。添加度量,如下表所示。

示例	结果
使用维度 <b>Customer</b> 和 <b>Sales</b> 以及度量 <b>Rank (Sales)</b> 创建表格	<p>结果取决于维度的排序顺序。如果按 <b>Customer</b> 对表格排序,表格会列出 <b>Astrida</b> 的所有 <b>Sales</b> 值,然后列出 <b>Betacab</b> 的所有同类型值,以此类推。<b>Sales</b> 值为 12 的 <b>Rank (Sales)</b> 结果将显示 10, <b>Sales</b> 值为 13 的相同字段结果将显示 9,以此类推,并且对 <b>Sales</b> 值为 78 的排行值返回 1。请注意,即使有 12 个 <b>Sales</b> 值,也仅显示 11 行,因为有两个 <b>Sales</b> 值相同 (78)。下一个列段数据从 <b>Betacab</b> 开始,在此列段数据中其第一个 <b>Sales</b> 值是 12。此字段的排行值 <b>Rank(Sales)</b> 显示为 11。</p> <p>如果此表格按 <b>Sales</b> 排序,则列段数据包含 <b>Sales</b> 值以及相应的 <b>Customer</b>。因为有两个 <b>Sales</b> 值是 12(对于 <b>Astrida</b> 和 <b>Betacab</b>),因此该列段数据每个 <b>Customer</b> 值的 <b>Rank (Sales)</b> 值都是 1-2。这是因为有两个 <b>Customer</b> 值的 <b>Sales</b> 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 <b>fmt</b> 默认值 (0) 时的结果。</p>

示例	结果
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 <b>mode</b> 和 <b>fmt</b> 分别设置为 1 和 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 '|');
```

#### 另请参阅:

□ [Sum - 图表函数\(第 161 页\)](#)

## 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。

#### Syntax:

```
HRank([ total ] expression [ , mode [ , format ] ])
```

**Return data type:** 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	说明
expression	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。
format	指定函数结果的文本呈现形式。
TOTAL	<p>如果图表是一维或如果表达式前面有 <b>TOTAL</b> 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。</p> <p><b>TOTAL</b> 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 &lt;fld&gt;。这些字段名应该是图表维度变量的子集。</p>

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于 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：

值	说明
0(默认)	<p>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。</p> <p>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。</p> <p>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。</p>
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：



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 )
```

## 5.21 统计分布函数

以下描述的统计分布函数都是通过使用 Cephess 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, 请参阅: <http://www.netlib.org/cephes/>。Cephess 函数库经获得许可使用。

统计分布 DIST 函数用于度量提供值给定分布点的分布函数概率。INV 函数用于在给定分布概率的情况下计算值。相反, 统计聚合函数组用于计算各种统计假设检验系列统计检验值的聚合值。

所有函数均可用于数据加载脚本和图表表达式。

### 统计分布函数概述

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

#### CHIDIST

**CHIDIST()** 用于返回单尾  $\chi^2$  分布概率。 $\chi^2$  分布与  $\chi^2$  检验相关联。

```
CHIDIST (value, degrees_freedom)
```

#### CHIINV

**CHIINV()** 用于返回单尾  $\chi^2$  分布概率的相反值。

```
CHIINV (prob, degrees_freedom)
```

#### NORMDIST

**NORMDIST()** 用于返回指定方式及标准误差的累积正态分布。如果  $\text{mean} = 0$  和  $\text{standard\_dev} = 1$ , 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)
```

### TDIST

**TDIST()** 用于返回学生 t 分布的概率，其中数值是一个将要为其计算概率的 t 的计算值。

```
TDIST (value, degrees_freedom, tails)
```

### TINV

**TINV()** 用于作为一个概率和自由度函数返回学生 t 分布的 t 值。

```
TINV (prob, degrees_freedom)
```

### FDIST

**FDIST()** 用于返回 F 概率分布。

```
FDIST (value, degrees_freedom1, degrees_freedom2)
```

### FINV

**FINV()** 用于返回 F 概率分布的相反值。

```
FINV (prob, degrees_freedom1, degrees_freedom2)
```

另请参阅：

▢ [统计聚合函数\(第 187 页\)](#)

## CHIDIST

**CHIDIST()** 用于返回单尾  $\chi^2$  分布概率。 $\chi^2$  分布与  $\chi^2$  检验相关联。

**Syntax:**

```
CHIDIST(value, degrees_freedom)
```

**Return data type:** 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。

此函数以以下方式与 **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。

**Syntax:**

```
CHIINV(prob, degrees_freedom)
```

**Return data type:** 数字

**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。

此函数以以下方式与 **CHIDIST** 函数关联：

If prob = CHIDIST(value,df), then CHIINV(prob, df) = value

**限制：**

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
CHIINV(0.9237827, 15 )	返回 8.0000

## FDIST

**FDIST()** 用于返回 F 概率分布。

**Syntax:**

```
FDIST(value, degrees_freedom1, degrees_freedom2)
```

**Return data type:** 数字

**参数：**

参数	说明
value	您想要用于评估分布的值。 <b>Value</b> 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。

此函数以以下方式与 **FINV** 函数关联：

If prob = FDIST(value, df1, df2), then FINV(prob, df1, df2) = value

#### 限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
FDIST(15, 8, 6)	返回 0.0019

## FINV

**FINV()** 用于返回 F 概率分布的相反值。

#### Syntax:

```
FINV(prob, degrees_freedom1, degrees_freedom2)
```

**Return data type:** 数字

#### 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。

此函数以以下方式与 **FDIST** 函数关联：

If prob = FDIST(value, df1, df2), then FINV(prob, df1, df2) = value

#### 限制：

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
FINV(0.0019369, 8, 6)	返回 15.0000

## NORMDIST

**NORMDIST()** 用于返回指定方式及标准误差的累积正态分布。如果 mean = 0 和 standard\_dev = 1, 则此函数返回标准正态分布。

### Syntax:

```
NORMDIST(value, mean, standard_dev)
```

**Return data type:** 数字

### 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值。
standard_dev	用于表示分布的标准偏差的正值。

此函数以以下方式与 **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。

### Syntax:

```
NORMINV(prob, mean, standard_dev)
```

**Return data type:** 数字

### 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。

参数	说明
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。

此函数以以下方式与 **NORMDIST** 函数关联：

If prob = NORMDIST(value, m, sd), then NORMINV(prob, m, sd) = value

**限制：**

所有参数均必须为数字，如不是则会返回 NULL 值。

示例和结果：

示例	结果
NORMINV( 0.6914625, 0, 1 )	返回 0.5000

## TDIST

**TDIST()** 用于返回学生 t 分布的概率，其中数值是一个将要为其计算概率的 t 的计算值。

**Syntax:**

```
TDIST(value, degrees_freedom, tails)
```

**Return data type:** 数字

**参数：**

参数	说明
value	您想要用于评估分布的值，且不能为负数。
degrees_freedom	表示自由度数的正整数。
tails	必须为 1(单尾分布)或 2(双尾分布)。

此函数以以下方式与 **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。

### Syntax:

```
TINV(prob, degrees_freedom)
```

**Return data type:** 数字

### 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。

此函数以以下方式与 **TDIST** 函数关联：

If prob = TDIST(value, df ,2), then TINV(prob, df) = value。

### 示例和结果:

示例	结果
TINV(0.3253086, 30 )	返回 1.0000

## 5.22 字符串函数

本节介绍用于处理和操作字符串的函数。在以下函数中，参数为表达式，其中的 **s** 应被解释为字符串。

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。

#### ApplyCodepage

应用不同的代码页到表达式内所述的字段或文本。代码页必须是数字格式。

```
ApplyCodepage(text, codepage)
```

#### Capitalize

**Capitalize()** 用于返回包含首字母大写的单词的字符串。

```
Capitalize (text)
```

### Chr

**Chr()** 用于返回与输入整数对应的 Unicode 字符。

```
Chr (int)
```

### Evaluate

**Evaluate()** 用于确定是否可将输入文本字符串作为有效的 Qlik Sense 表达式来计算值，如果可以，则以字符串形式返回该表达式的值。如果输入字符串不是有效的表达式，则返回 NULL。

```
Evaluate (expression_text)
```

### FindOneOf

**FindOneOf()** 用于搜索字符串，以便从一组提供的字符中找到任意字符出现的位置。如果不提供第三个参数(值大于 1)，则返回任意字符在搜索集合中首次出现的位置。如果未找到匹配值，则返回 0。

```
FindOneOf (text, char_set[, count])
```

### Hash128

**Hash128()** 用于返回 128 位哈希的组合输入表达式值。结果为 22 个字符的字符串。

```
Hash128 (expr[, expression])
```

### Hash160

**Hash160()** 用于返回 160 位哈希的组合输入表达式值。结果为 27 个字符的字符串。

```
Hash160 (expr[, expression])
```

### Hash256

**Hash256()** 用于返回 256 位哈希的组合输入表达式值。结果为 43 个字符的字符串。

```
Hash256 (expr[, expression])
```

### Index

**Index()** 用于搜索字符串，以便找到所提供子字符串第 n 次出现的开始位置。可选的第三个参数用于提供值 n，如果省略，则值为 1。如果为负值，则从字符串的结尾开始搜索。字符串中的位置从 1 开始编号。

```
Index (text, substring[, count])
```

### KeepChar

**KeepChar()** 用于返回特定字符串，其中包含第一个字符串但不包含第二个字符串所包含的任何字符的字符。

```
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前 n 个字符，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)
```

### 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()** 用于返回从第二个参数所定义字符位置开始的部分字符串，并且字符串长度由第三个参数定义，或者在省略第三个参数的情况下，包含从第二个参数所定义字符位置开始的其余字符串。字符串第一个位置编号为 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 (char )
```

### PurgeChar

**PurgeChar()** 用于返回特定字符串，其中包含输入字符串中除第二个参数字符串的所有字符以外的所有字符。

```
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()** 用于返回父字符串中作为分隔符出现在指定字符之间的文本。如果不提供可选的第三个参数(值大于 1)，则返回第一次出现的分隔符之间的文本字符串。

```
TextBetween (text, sub_string)
```

### **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()** 用于转换输入字符串中的所有字符，以大写表达式中的所有数据。

```
Upper (text)
```

## **Capitalize**

**Capitalize()** 用于返回包含首字母大写的所有单词的字符串。

### **Syntax:**

```
Capitalize(text)
```

**Return data type:** 字符串

示例和结果：

示例	结果
Capitalize ( 'my little pony' )	返回 'My Little Pony'
Capitalize ( 'AA bb cC Dd' )	返回 'Aa Bb Cc Dc'

## Chr

**Chr()** 用于返回与输入整数对应的 Unicode 字符。

**Syntax:**

**Chr** (int)

**Return data type:** string

示例和结果：

示例	结果
Chr(65)	返回字符串 'A'

## 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。

**Syntax:**

**Evaluate** (expression\_text)

**Return data type:** 双



此字符串函数不可用于图表表达式。

示例和结果：

示例	结果
Evaluate ( 5 * 8 )	返回 '40'

## 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。

**Syntax:**

**FindOneOf** (text, char\_set[, count])

**Return data type:** 整数

**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。

**示例和结果:**

示例	结果
FindOneOf( 'my example text string', 'et%s')	返回“4”。
FindOneOf( 'my example text string', 'et%s', 3)	返回“12”。因为搜索针对所有字符:e、t、% 或 s和“t”是第三次出现的位置，因此是在位置 12。
FindOneOf( 'my example text string', '%&')	返回“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。

**Syntax:**

```
Hash128(expr{, expression})
```

**Return data type:** 字符串

**示例:**

```
Hash128 ( 'abc', 'xyz', '123' )
Hash128 ( Region, Year, Month )
```

## Hash160

**Hash160()** 用于返回 160 位哈希的组合输入表达式值。结果为 27 个字符的字符串。

**Syntax:**

```
Hash160(expr{, expression})
```

**Return data type:** 字符串

**示例:**

```
Hash160 ( 'abc', 'xyz', '123' )
Hash160 ( Region, Year, Month )
```

## Hash256

**Hash256()** 用于返回 256 位哈希的组合输入表达式值。结果为 43 个字符的字符串。

### Syntax:

```
Hash256 (expr{, expression})
```

**Return data type:** 字符串

### 示例:

```
Hash256 ( 'abc', 'xyz', '123' )
Hash256 ( Region, Year, Month )
```

## 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。

### Syntax:

```
Index (text, substring[, count])
```

**Return data type:** 整数

### 参数:

参数	说明
text	原始字符串。
substring	在 text 中搜索的字符串。
count	定义搜索哪一次出现的 <b>substring</b> 。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)

示例	结果
Left( Date, Index( Date, '-' ) -1 ) where <b>Date</b> = 1997-07-14	返回 1997
Mid( Date, Index( Date, '-', 2 ) -2, 2 ) where <b>Date</b> = 1997-07-14	返回 07

## KeepChar

**KeepChar()** 用于返回特定字符串，其中包含第一个字符串但不包含第二个字符串所包含的任何字符的字符。

**Syntax:**

**KeepChar** (text, keep\_chars)

**Return data type:** 字符串

**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。

示例和结果:

示例	结果
KeepChar ( 'a1b2c3', '123' )	返回“123”。
KeepChar ( 'a1b2c3', '1234' )	返回“123”。
KeepChar ( 'a1b22c3', '1234' )	返回“1223”。

另请参阅:

▢ *PurgeChar* (第 577 页)

## Left

**Left()** 用于返回特定字符串，其中包含输入字符串的前 n 个字符，其中字符数量由第二个参数决定。

**Syntax:**

**Left** (text, count)

**Return data type:** 字符串

**参数:**

参数	说明
text	原始字符串。
count	定义从字符串 <b>text</b> 左侧开始包含的字符数。

示例和结果：

示例	结果
Left('abcdef', 3)	返回 'abc'
Left(Date, 4) where <b>Date</b> = 1997-07-14	返回“1997”

另请： *Index* (第 573 页)，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。

**Syntax:**

```
Len(text)
```

**Return data type:** 整数

示例和结果：

示例	结果
Len(Name) where <b>Name</b> = 'Peter'	返回“5”

## Lower

**Lower()** 用于将输入字符串中的所有字符转换为小写字符。

**Syntax:**

```
Lower(text)
```

**Return data type:** 字符串

示例和结果：

示例	结果
Lower('abcD')	返回 'abcd'

## LTrim

**LTrim()** 用于返回由任何前导空格剪裁的输入字符串。

**Syntax:****LTrim(text)****Return data type:** 字符串

示例和结果:

示例	结果
LTrim( ' abc' )	返回 'abc'
LTrim( 'abc ' )	返回 'abc'

**Mid**

**Mid()** 用于返回从第二个参数所定义字符位置开始的部分字符串，并且字符串长度由第三个参数定义，或者在省略第三个参数的情况下，包含从第二个参数所定义字符位置开始的其余字符串。字符串第一个位置编号为 1。

**Syntax:****Mid(text, start[, count])****Return data type:** 字符串

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 <b>start</b> 所定义位置开始的所有字符。

示例和结果:

示例	结果
Mid('abcdef', 3 )	返回“cdef”
Mid('abcdef', 3, 2 )	返回“cd”
Mid( Date, 3 ) where Date = 970714	返回“0714”
Mid( Date, 3, 2 ) where Date = 970714	返回“07”

另请参阅:

□ [Index \(第 573 页\)](#)



## Ord

**Ord()** 用于返回输入字符串第一个字符的 Unicode 代码点数。

### Syntax:

```
Ord(char)
```

**Return data type:** 整数

示例和结果：

示例	结果
Ord('A')	返回整数 65。
Ord('Ab')	返回整数 65。

## PurgeChar

**PurgeChar()** 用于返回特定字符串，其中包含输入字符串中除第二个参数字符串的所有字符以外的所有字符。

### Syntax:

```
PurgeChar(text, remove_chars)
```

**Return data type:** 字符串

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。

**Return data type:** 字符串

示例和结果：

示例	结果
PurgeChar ( 'a1b2c3','123' )	返回“abc”

另请参阅：

▢ [KeepChar\(第 574 页\)](#)

## 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。

### Syntax:

```
Repeat(text[, repeat_count])
```

**Return data type:** 字符串

### 参数：

参数	说明
text	原始字符串。
repeat_count	定义字符串 <b>text</b> 的字符在输出字符串中重复的次数。

### 示例和结果：

示例	结果
Repeat( ' * ', rating ) when <b>rating</b> = 4	返回 *****

## 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。

### Syntax:

```
Replace(text, from_str, to_str)
```

**Return data type:** 字符串

### 参数：

参数	说明
text	原始字符串。
from_str	在输入字符串 <b>text</b> 内可能出现一次或多次的字符串。
to_str	替换在字符串 <b>text</b> 内出现的所有 <b>from_str</b> 的字符串。

### 示例和结果：

示例	结果
Replace('abccde', 'cc', 'xyz')	返回 'abxyzde'

另请参阅：

## 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。

**Syntax:**

```
Right(text, count)
```

**Return data type:** 字符串

**参数：**

参数	说明
text	原始字符串。
count	定义从字符串 <b>text</b> 右侧开始包含的字符数。

示例和结果：

示例	结果
Right('abcdef', 3)	返回 'def'
Right( Date, 2 ) where <b>Date</b> = 1997-07-14	返回“14”

## RTrim

**RTrim()** 用于返回由任何尾部空格剪裁的输入字符串。

**Syntax:**

```
RTrim(text)
```

**Return data type:** 字符串

示例和结果：

示例	结果
RTrim( ' abc' )	返回 'abc'
RTrim( 'abc ' )	返回 'abc'

## 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。

### Syntax:

```
SubField(text, delimiter[, field_no ])
```

**Return data type:** 字符串

### 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 <b>text</b> 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 <b>text</b> 的哪些子字符串。如果是负值，则会从字符串右侧开始提取子字符串。也就是说，如果 <b>field_no</b> 是正值，则是从右向左而不是从左向右搜索字符串。



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。

### 示例和结果:

示例	结果
<code>SubField(S, ';' ,2)</code>	如果 <b>S</b> 为 'abc;cde;efg', 则返回 'cde'。
<code>SubField(S, ';' ,1)</code>	如果 <b>S</b> 为空字符串, 则返回 NULL。
<code>SubField(S, ';' ,1)</code>	如果 <b>S</b> 为 ';', 则返回一个空字符串。

示例	结果																																										
<p>添加示例脚本到应用程序并运行。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。</p> <pre>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;</pre>	<table><thead><tr><th>Name</th><th>FirstName</th><th>Surname</th></tr></thead><tbody><tr><td>Dave Owen</td><td>Dave</td><td>Owen</td></tr><tr><td>Joe Tem</td><td>Joe</td><td>Tem</td></tr></tbody></table>	Name	FirstName	Surname	Dave Owen	Dave	Owen	Joe Tem	Joe	Tem																																	
Name	FirstName	Surname																																									
Dave Owen	Dave	Owen																																									
Joe Tem	Joe	Tem																																									
<p>假设您有一个变量，其值为路径名 vMyPath，</p> <pre>Set vMyPath=\Users\ext_jrb\Documents\Qlik\Sense\Apps;。</pre>	<p>在文本和图片图表中，您可以添加度量，如：</p> <p>SubField(vMyPath, '\',-3)，将生成“Qlik”，因为他是变量 vMyPath 右端第三个子字符。</p>																																										
<p>此示例演示了如何使用 <b>Subfield()</b> 函数的多个实例，每个实例都不考虑 field_no 参数，其中相同的 <b>LOAD</b> 语句会创建所有组合的 Cartesian 产品。<b>DISTINCT</b> 选项用于避免创建重复记录。</p> <p>添加示例脚本到应用程序并运行。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。</p> <pre>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 ' ');</pre>	<table><thead><tr><th>Instrument</th><th>Player</th><th>Project</th></tr></thead><tbody><tr><td>Guitar</td><td>Mike</td><td>Music</td></tr><tr><td>Guitar</td><td>Mike</td><td>Video</td></tr><tr><td>Guitar</td><td>Mike</td><td>OST</td></tr><tr><td>Guitar</td><td>Neil</td><td>Music</td></tr><tr><td>Guitar</td><td>Neil</td><td>Video</td></tr><tr><td>Guitar</td><td>Neil</td><td>OST</td></tr><tr><td>Synth</td><td>Jen</td><td>Music</td></tr><tr><td>Synth</td><td>Jen</td><td>Video</td></tr><tr><td>Synth</td><td>Jen</td><td>OST</td></tr><tr><td>Synth</td><td>Jo</td><td>Music</td></tr><tr><td>Synth</td><td>Neil</td><td>Music</td></tr><tr><td>Synth</td><td>Neil</td><td>Video</td></tr><tr><td>Synth</td><td>Neil</td><td>OST</td></tr></tbody></table>	Instrument	Player	Project	Guitar	Mike	Music	Guitar	Mike	Video	Guitar	Mike	OST	Guitar	Neil	Music	Guitar	Neil	Video	Guitar	Neil	OST	Synth	Jen	Music	Synth	Jen	Video	Synth	Jen	OST	Synth	Jo	Music	Synth	Neil	Music	Synth	Neil	Video	Synth	Neil	OST
Instrument	Player	Project																																									
Guitar	Mike	Music																																									
Guitar	Mike	Video																																									
Guitar	Mike	OST																																									
Guitar	Neil	Music																																									
Guitar	Neil	Video																																									
Guitar	Neil	OST																																									
Synth	Jen	Music																																									
Synth	Jen	Video																																									
Synth	Jen	OST																																									
Synth	Jo	Music																																									
Synth	Neil	Music																																									
Synth	Neil	Video																																									
Synth	Neil	OST																																									

## 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。

**Syntax:**

```
SubStringCount(text, sub_string)
```

**Return data type:** 整数

**参数:**

参数	说明
text	原始字符串。
sub_string	在输入字符串 <b>text</b> 内可能出现一次或多次的字符串。

**示例和结果:**

示例	结果
SubStringCount ( 'abcdefgxyz', 'cd' )	返回“2”
SubStringCount ( 'abcdefgxyz', 'dc' )	返回“0”

## TextBetween

**TextBetween()** 用于返回父字符串中作为分隔符出现在指定字符之间的文本。如果不提供可选的第三个参数(值大于 1)，则返回第一次出现的分隔符之间的文本字符串。

**Syntax:**

```
TextBetween(text, delimiter1, delimiter2[, n])
```

**Return data type:** 字符串

**参数:**

参数	说明
text	原始字符串。
delimiter1	指定要在 <b>text</b> 中搜索的第一个分隔符(或字符串)。
delimiter2	指定要在 <b>text</b> 中搜索的第二个分隔符(或字符串)。
count	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分隔符对之间的字符。

**示例和结果:**

示例	结果
TextBetween('<abc>', '<', '>')	返回 'abc'
TextBetween('<abc><de>', '<', '>', 2)	返回 'de'

## Trim

**Trim()** 用于返回由任何前导和尾部空格剪裁的输入字符串。

**Syntax:**

```
Trim(text)
```

**Return data type:** 字符串

示例和结果：

示例	结果
Trim( ' abc' )	返回 'abc'
Trim( 'abc ' )	返回 'abc'
Trim( ' abc ' )	返回 'abc'

## Upper

**Upper()** 用于转换输入字符串中的所有字符，以大写表达式中的所有数据。

**Syntax:**

```
Upper(text)
```

**Return data type:** 字符串

示例和结果：

示例	结果
Upper(' abcd')	返回 'ABCD'

## 5.23 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。

```
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 ( )
```

### 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

此函数用于返回维度的名称。`Index` 是一个可选整数，表明应返回所使用的维度。



### 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)

#### IsPartialReload

此函数返回 - 如果当前部分重新加载, 则返回 1 (True), 否则为 0 (False)。

#### IsPartialReload ()

#### OSUser

此函数返回操作系统返回的包含当前用户名的字符串。此函数均可用于数据加载脚本和图表表达式。

#### OSUser ( )

#### 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。

#### 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

此图表函数用于返回所使用的对象的状态名称。当对象状态更改时, Qlik Sense 开发者可以使用该函数制作动态文本和颜色。需要特别注意的是该函数仅能作用于对象。它不可以在图表表达式中用来定义该表达式涉及的任何状态。

#### StateName - 图表函数 ()

另请参阅:

▢ [GetFolderPath \(第 439 页\)](#)

## GetObjectField - 图表函数

此函数用于返回维度的名称。**Index** 是一个可选整数, 表明应返回所使用的维度。

#### Syntax:

#### GetObjectField ([index])

示例:

GetObjectField(2)

## 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)。

### Syntax:

```
IsPartialReload()
```

## ProductVersion

此函数用于返回完整的 Qlik Sense 版本和内部版本号作为一个字符串。

### Syntax:

```
ProductVersion()
```

## StateName - 图表函数

此图表函数用于返回所使用的对象的状态名称。当对象状态更改时, Qlik Sense 开发者可以使用该函数制作动态文本和颜色。需要特别注意的是该函数仅能作用于对象。它不能在图表表达式中用来定义该表达式涉及的任何状态。

### Syntax:

```
StateName ()
```



只能使用 Qlik Engine API 定义和分配交替状态。

### 示例 1:

```
Dynamic Text
='Region - ' & if(StateName() = '$', 'Default', StateName())
```

### 示例 2:

```
Dynamic Colors
if(StateName() = 'Group 1', rgb(152, 171, 206),
    if(StateName() = 'Group 2', rgb(187, 200, 179),
        rgb(210, 210, 210)
    )
)
```

## 5.24 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：

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。

### Syntax:

```
FieldName(field_number ,table_name)
```

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。

Syntax:

```
FieldNumber(field_name ,table_name)
```

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。

Syntax:

```
NoOfFields(table_name)
```

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。

**Syntax:**

```
NoOfRows (table_name)
```

参数：

参数	说明
table_name	表格的名称。

示例：

```
LET a = NoOfRows('tab1');
```

## 5.25 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  $-1 \leq x \leq 1$  时才可定义此函数。结果是介于 0 和  $\pi$  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** 的正弦。仅在  $-1 \leq x \leq 1$  时才可定义此函数。结果是介于  $-\pi/2$  和  $\pi/2$  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  $-\pi/2$  和  $\pi/2$  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  $-\pi$  和  $+\pi$  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 )
```

### **示例：**

以下脚本代码用于加载示例表格，然后加载包含值计算的三角函数和双曲操作的表格。

```
SampleData:
LOAD * Inline
[Value
-1
0
1];

Results:
Load *,
cos(Value),
acos(Value),
sin(Value),
asin(Value),
tan(Value),
atan(Value),
atan2(Value, Value),
```

```
cosh(value),  
sinh(value),  
tanh(value)  
RESIDENT SampleData;  
  
Drop Table SampleData;
```



## 6 文件系统访问限制

出于安全原因，标准模式下的 Qlik Sense 不支持数据加载脚本中的绝对或相对路径，或者展示文件系统的函数和变量。

但是，由于 QlikView 支持绝对和相对路径，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。

### 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发生这种情况。



如果暴露指向数据文件的路径已成为一个问题，则我们建议使用文件夹数据连接来连接到数据文件(如果可能)。

### 6.1 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。错误消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，不会生成错误消息或日志文件条目，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

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:。
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:。

变量	标准模式	旧模式	定义
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:库连接	支持的输入:库连接或绝对/相对路径	<b>Include/Must_Include</b> 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。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

语句	标准模式	旧模式	定义
Binary	支持的输入:库连接	支持的输入:库连接或绝对/相对路径	<b>binary</b> 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: 库连接	支持的输入: 库连接或绝对/相对路径	<b>CONNECT</b> 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: 库连接	支持的输入: 库连接或绝对/相对路径	<b>Directory</b> 语句用于定义在后续 <b>LOAD</b> 语句中查找数据文件的目录, 直到出现新的 <b>Directory</b> 语句。
Execute	不支持	支持的输入: 库连接或绝对/相对路径	<b>Execute</b> 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: 库连接	支持的输入: 库连接或绝对/相对路径	用于返回浏览字符串到可执行的 Qlik Sense 文件。
Store into ...	支持的输入: 库连接	支持的输入: 库连接或绝对/相对路径	用于返回可执行的 Qlik Sense 的根目录。

## 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: 库连接 返回的输出: 库连接	支持的输入: 库连接或绝对/相对路径  返回的输出: 库连接或绝对文件路径, 具体取决于输入	filelist mask 语法会在匹配 <b>filelist mask</b> 的当前目录中生成逗号分隔的全部文件列表。 <b>dirlist mask</b> 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

函数	标准模式	旧模式	定义
Attribute()	支持的输入: 库连接	支持的输入: 库连接或绝对/相对路径	以文本形式返回不同媒体文件的元标签的值。
ConnectionString()	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: 库连接或绝对文件路径, 具体取决于输入	<b>FileDir</b> 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: 库连接或绝对文件路径, 具体取决于输入	<b>FilePath</b> 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: 库连接	支持的输入: 库连接或绝对/相对路径	<b>FileSize</b> 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: 库连接	支持的输入: 库连接或绝对/相对路径	<b>FileTime</b> 函数用于返回文件 filename 的上一次修改日期和时间的戳。如果未指定 filename, 则此函数将参考当前阅读的表格文件。
GetFolderPath()	不支持	返回的输出: 绝对路径	<b>GetFolderPath</b> 函数用于返回 Microsoft Windows SHGetFolderPath 函数的值和返回相关路径。例如, <b>MyMusic</b> 。注意, 此函数不用于在 Windows Explorer 中看到的空间。

函数	标准模式	旧模式	定义
QvdCreateTime()	支持的输入: 库连接	支持的输入: 库连接或绝对/相对路径	此脚本函数用于返回 QVD 文件的 XML-标题时间戳(如果有), 否则返回 NULL 值。
QvdFieldName()	支持的输入: 库连接	支持的输入: 库连接或绝对/相对路径	此脚本函数用于返回字段编号名 <b>fieldno</b> (如果其存在于 QVD 文件中)(否则, 返回 NULL 值)。
QvdNoOfFields()	支持的输入: 库连接	支持的输入: 库连接或绝对/相对路径	此脚本函数用于返回 QVD 文件中的字段数。
QvdNoOfRecords()	支持的输入: 库连接	支持的输入: 库连接或绝对/相对路径	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: 库连接	支持的输入: 库连接或绝对/相对路径	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

函数	标准模式	旧模式	定义
DocumentPath()	不支持	返回的输出: 绝对路径	此函数用于返回一个包含至当前 Qlik Sense 应用程序完整路径的字符串。
GetRegistryString()	不支持	支持	返回一个称为注册关键字的值及一个给定的注册路径。此函数可用于图表及脚本等类似程序中。

## 6.2 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。

执行以下操作:

1. 在文本编辑器中打开 *C:\Users\{user}\Documents\Qlik\Sense\Settings.ini*。
2. 将 *StandardReload=1* 更改为 *StandardReload=0*。
3. 保存文件, 并启动 Qlik Sense Desktop, 该程序将在旧模式下运行。

StandardReload 的可用设置是:

- 1(标准模式)
- 0(旧模式)

## 7 Qlik Sense 不支持的 QlikView 函数和语句

在 Qlik Sense 中也会支持可用于 QlikView 加载脚本和图表表达式的大部分函数和语句，但有一些例外，如下所述。

### 7.1 Qlik Sense 不支持的脚本语句

下表介绍了 Qlik Sense 不支持的 QlikView 脚本语句。

语句	注释
<b>Command</b>	使用 <b>SQL</b> 。
<b>InputField</b>	

### 7.2 Qlik Sense 不支持的函数

下表介绍了 Qlik Sense 不支持的 QlikView 脚本和图表函数。

- **GetCurrentField**
- **GetExtendedProperty**
- **Input**
- **InputAvg**
- **InputSum**
- **MsgBox**
- **NoOfReports**
- **ReportComment**
- **ReportId**
- **ReportName**
- **ReportNumber**

## 8 Qlik Sense 不推荐的函数和语句

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。由于兼容性原因，它们仍可按照预期方式起作用，但最好是根据本部分中的建议更新代码，因为可能会在未来的版本中将它们删除。

### 8.1 Qlik Sense 不推荐的脚本语句

下表介绍了不建议用于 Qlik Sense 的 QlikView 脚本语句。

语句	推荐
<b>Command</b>	使用 <b>SQL</b> 。
<b>Bundle</b>	Qlik Sense 不支持显示用 <b>Bundle</b> 和 <b>Info</b> 加载的信息。
<b>Info</b>	
<b>CustomConnect</b>	使用 <b>Custom Connect</b> 。

### 8.2 Qlik Sense 不推荐的脚本语句参数

下表介绍了不建议用于 Qlik Sense 的 QlikView 脚本语句参数。

语句	参数
<b>Buffer</b>	使用 <b>Incremental</b> : <ul style="list-style-type: none"><li>• <b>Inc</b>(不推荐)</li><li>• <b>Incr</b>(不推荐)</li></ul>



语句	参数
----	----

<b>LOAD</b>	以下参数关键字由 QlikView 文件转换向导生成。在重新加载数据时保留功能，但 Qlik Sense 不会提供使用以下参数生成语句的指导支持/向导：
-------------	--

- Bottom
- Cellvalue
- Col
- Colmatch
- Colsplit
- Colxtr
- Compound
- Contain
- Equal
- Every
- Expand
- Filters
- Intarray
- Interpret
- Length
- Longer
- Numerical
- Pos
- Remove
- Rotate
- Row
- Rowcnd
- Shorter
- Start
- Strcnd
- Top
- Transpose
- Unwrap

---

### 8.3 Qlik Sense 不推荐的函数

下表介绍了不建议用于 Qlik Sense 的 QlikView 脚本和图表函数。

函数	推荐
<b>NumAvg</b>	使用范围函数。
<b>NumCount</b>	另请：范围函数(第 522 页)
<b>NumMax</b>	
<b>NumMin</b>	
<b>NumSum</b>	
<b>QliktechBlue</b> <b>QliktechGray</b>	使用其他颜色函数。 <b>QliktechBlue()</b> 由 <b>RGB(8, 18, 90)</b> 替代且 <b>QliktechGray</b> 由 <b>RGB(158, 148, 137)</b> 替代可以获得相同的颜色。 另请：颜色函数(第 308 页)
<b>QlikViewVersion</b>	使用 <b>ProductVersion</b> 。 另请：ProductVersion (第 586 页)
<b>QVUser</b>	
<b>Year2Date</b>	使用 <b>YearToDate</b> 。
<b>Vrank</b>	使用 <b>Rank</b> 。
<b>WildMatch5</b>	使用 <b>WildMatch</b> 。

## 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。