スクリプト構文およびチャート関数

Qlik Sense®
May 2024

Contents

1 Qlik Sense とは?

Qlik Sense はデータ分析のプラットフォームです。Qlik Sense があれば、データの分析や発見を独自に行うことができ、複数のグループや組織の間で知識を共有し、データ分析をすることも可能です。Qlik Sense ではユーザーが自問自答しながら、インサイトを深めることができます。また、Qlik Sense なら、同僚と共同で意思決定を行うことも可能です。

1.1 Qlik Sense でできること

一般的なビジネス インテリジェンス (BI) 製品は、すでにわかっている質問に答えるためのサポートを提供します。しかし、フォローアップの質問をはじめ、レポートやビジュアライゼーションから派生した質問については、どうでしょうか?Qlik Sense を使用すれば、連想的なエクスペリエンスを駆使して相次ぐ質問に回答し、必要なインサイトに到達することができます。Qlik Sense では、簡単なクリック操作で自由にデータを探索し、各ステップで学びながら、それまでの発見に基づいて次のステップに進むことができます。

1.2 Qlik Sense の仕組み

Qlik Sense は、情報のビューを迅速に生成します。Qlik Sense では、事前定義の静的レポートが不要で、他のユーザーに依存することもありません。簡単なクリック操作でデータについて学べます。クリックするたびに、Qlik Sense は即座に反応し、選択された内容を基に新しく計算し直されたデータとビジュアライゼーションで Qlik Sense のすべてのビジュアライゼーションとビューを更新します。

アプリモデル

巨大なビジネス アプリケーションを展開し管理する代わりに、再利用、変更、他のユーザーとの共有が可能な独自の Qlik Sense アプリを作成することができます。アプリモデルを使用すれば、専門家から新しいレポートやビジュアライゼーションを入手する必要がなく、その都度、自問自答することで次のステップを見出すことができます。

連想的エクスペリエンス

Qlik Sense は、データ内のすべての関係を自動的に管理し、**green/white/gray** のメタファーを使用して情報を表示します。選択した値は緑色でハイライトされ、関連性のあるデータは白、除外 (関連性のない) データはグレーでハイライトされます。こうした迅速なフィードバックにより、新しい疑問について考え、継続的なデータ探索や発見が可能になります。

コラボレーションとモビリティ

Qlik Sense は、時間や場所に関係なく、同僚とのコラボレーションを可能にします。連想的エクスペリエンスやコラボレーションといった Qlik Sense の機能は、すべてモバイル機器でも利用可能です。Qlik Sense があれば、どこにいても自問自答し、同僚からのフォローアップ質問に答えることができます。

1.3 Qlik Sense の展開方法

Qlik Sense の展開には、Qlik Sense Desktop と Qlik Sense Enterprise の 2 種類の方法があります。

Qlik Sense Desktop

これはインストールが容易な単一ユーザー向けのバージョンで、通常はローカル コンピューターにインストールされます。

Qlik Sense Enterprise

このバージョンを使用して、Qlik Sense サイトを展開します。サイトは、共通の論理リポジトリまたはセントラル ノードに接続している1つまたは複数のサーバー コンピューターの集合体です。

1.4 Qlik Sense サイトの管理方法

Qlik 管理コンソール があれば、簡単かつ直感的な方法で Qlik Sense サイトを構成、管理、監視できます。ライセンスやアクセス、セキュリティルールの管理、ノードやデータ ソース接続の設定、コンテンツとユーザーなどの多くのアクティビティとリソースの同期が可能です。

1.5 Qlik Sense の拡張とユーザー独自の目的への適応

Qlik Sense には、独自の拡張機能を開発して、以下のようなさまざまな目的で Qlik Sense を適応・統合できるよう柔軟な API および SDK が備えられています。

拡張とマッシュアップ機能の構築

ここでは、JavaScript を使用してウェブ開発を行い、Qlik Sense アプリのカスタム ビジュアライゼーションである拡張機能を構築できます。または、マッシュアップ API を使用して Qlik Sense のコンテンツでウェブサイトを構築できます。

クライアントの構築

クライアントを .NET で構築し、Qlik Sense オブジェクトを独自のアプリケーションに埋め込むことができます。Qlik Sense クライアント プロトコルを使用して、WebSocket 通信を取り扱うことのできる任意のプログラミング言語でネイティブ クライアントを構築することも可能です。

サーバー ツールの構築

サービスとユーザー ディレクトリ API を使うと、Qlik Sense サイトを管理するための独自のツールを構築できます。

他のデータ ソースへの接続

Qlik Sense コネクタを作成して、カスタム データ ソースからデータを取得できます。

2 スクリプト構文の概要

2.1 スクリプト構文について

スクリプトで、ロジックに含まれるデータ ソース名やテーブル名、項目名が定義されます。さらに、アクセス権を定義する項目もスクリプトで定義されます。スクリプトは、連続して実行されるいくつかのステートメントで構成されます。

Qlik Sense コマンド ライン構文およびスクリプト構文は、Backus-Naur 形式またはBNF コードと呼ばれる表記で記述されます。

新しい Qlik Sense ファイルが作成された時点で、最初のコード行はすでに生成されています。これらのデータ型変換変数の初期値は、OS の地域の設定から取得されます。

スクリプトは、連続して実行される多数のステートメントとキーワードで構成されています。すべてのスクリプト ステートメントは、セミコロン「;」で終わる必要があります。

LOAD-ステートメントで式と関数を使用して、ロードされたデータを変換できます。

コンマ、タブ、セミコロンを区切り記号として含むテーブル ファイルでは、**LOAD**-ステートメントが使用されます。デフォルトでは、**LOAD**-ステートメントはファイルのすべての項目をロードします。

一般的なデータベースには、ODBC または OLE DB データベース コネクタを通じアクセスできます。ここでは、標準 SQL ステートメントが使用されます。使用可能な SQL 構文は、ODBC ドライバの種類によって異なります。

さらに、カスタム コネクタを使用して、その他のデータ ソースにアクセスできます。

2.2 Backus-Naur (バッカスナウア) 形式とは?

Qlik Sense コマンド ライン構文およびスクリプト構文は、Backus-Naur 形式またはBNF コードと呼ばれる表記で記述されます。

以下の表に、BNF コードで使用されるシンボルとその解釈についての説明を記載しています。

記号

シンボル	説明
|	論理 OR。両側のいずれかのシンボルを使用できます。
( )	丸括弧は優先を意味します。BNF 構文を構成するために使用されます。
[ ]	角括弧。囲まれた項目はオプションです。
{ }	波括弧。囲まれた項目を 0 回以上繰り返します。
シンボル	非終端構文カテゴリ。他のシンボルにさらに分割できます。つまり、上記のシンボルと他の非終端シンボル、テキスト文字列などの組み合わせです。
::=	シンボルを定義するブロックの開始マークです。
LOAD	1 つのテキスト文字列から成る終端シンボル。このとおりにスクリプトに書き込む必要があります。

終端シンボルはすべて、**bold face** フォントで表記されます。例えば、" (" は優先順位を定義する丸括弧として解釈され、"**(**" はスクリプトで出力される文字として解釈されます。

alias ステートメントは次のように記述されます。

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

これは、テキスト文字列 "alias"、任意の項目名、テキスト文字列 "as"、任意の alias 名の順番で続くと解釈されます。任意の数の "fieldname as alias" をコンマで区切って指定できます。

次のステートメントは有効です。

```
alias a as first;

alias a as first, b as second;

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

次のステートメントは無効です。

```
alias a as first b as second;

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

3 スクリプトのステートメントとキーワード

Qlik Sense のスクリプトは多数のステートメントで構成されています。ステートメントは、正規のスクリプト ステートメントまたはスクリプト制御 ステートメントのどちらかになります。先頭にプレフィックスが付くステートメントもあります。

一般に正規 ステートメントは、何らかの形でデータの操作に使用されます。これらのステートメントはスクリプト内で何行でも記述できますが、必ずセミコロン「;」で終了する必要があります。

通常、制御 ステートメントはスクリプト実行の流れを制御するために使用されます。制御 ステートメントの各節は 1 つのスクリプト行に収める必要があり、セミコロン「;」または改行コードで終了する必要があります。

プレフィックスは、必要に応じて正規 ステートメントに適用できますが、制御 ステートメントには適用できません。ただし、**when** および **unless** プレフィックスは、一部の制御 ステートメント節のサフィックスとして使用できます。

次のセクションでは、スクリプト ステートメントと制御 ステートメント、プレフィックスをアルファベット順にリストアップしています。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。ただし、ステートメントで使用される項目名と変数名は大文字と小文字が区別されます。

3.1 スクリプト制御 ステートメント

Qlik Sense のスクリプトは多数のステートメントで構成されています。ステートメントは、正規のスクリプト ステートメントまたはスクリプト制御 ステートメントのどちらかになります。

通常、制御 ステートメントはスクリプト実行の流れを制御するために使用されます。制御 ステートメントの各節は 1 スクリプト行に収める必要があり、セミコロンまたは改行コードで終了する必要があります。

プレフィックスは、制御 ステートメントには適用されません。ただし、例外として、**when** および **unless** プレフィックスは、数個の特定の制御 ステートメントで使用できます。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。

スクリプト制御 ステートメントの概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Call
call 制御 ステートメントは、事前に **sub** ステートメントで定義されているサブルーチンを呼び出します。

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

Do..loop
do..loop 制御 ステートメントはスクリプト反復構文で、論理条件が満たされるまで、1 つまたは複数のステートメントを実行します。

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

Exit script

この制御ステートメントは、スクリプトの実行を停止します。スクリプト内の任意の場所に挿入できます。

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

For each ..next

for each..next 制御ステートメントは、コンマ区切りリストの各値に対して、1 つまたは複数のステートメントを実行するスクリプト反復構文です。**for** と**next** で囲まれたループ内のステートメントは、リストの各値で指定された回数分実行されます。

```
For each..next var in list

[statements]

[exit for [ ( when | unless ) condition ]

[statements]

next [var]
```

For..next

for..next 制御ステートメントは、カウンタ付きのスクリプト反復構文です。**for** と**next** で囲まれたループ内のステートメントは、カウンタ変数の初期値と最終値で指定された回数分実行されます。

```
For..next counter = expr1 to expr2 [ stepexpr3 ]

[statements]

[exit for [ ( when | unless ) condition ]

[statements]

Next [counter]
```

If..then

if..then 制御ステートメントは、1 つ以上の論理条件に応じて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

> ***if..then*** *ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、使用可能な 4 つの節 (**if..then**、**elseif..then**、**else**、**end if**) が行をまたぐことはできません。*

```
If..then..elseif..else..end if condition then

  [ statements ]
```

```
{ elseif condition then

  [ statements ] }

[ else

  [ statements ] ]

end if
```

Sub

sub..end sub 制御ステートメントは、**call** ステートメントで呼び出されるサブルーチンを定義します。

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

Switch

switch 制御ステートメントは、数式の値に基づいて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

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

Call

call 制御ステートメントは、事前に **sub** ステートメントで定義されているサブルーチンを呼び出します。

構文:

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

引数:

引数

引数	説明
name	サブルーチンの名前。
paramlist	サブルーチンに送られる実パラメータのコンマ区切りのリスト。リスト内の各アイテムは、項目名、変数、または任意の数式です。

call ステートメントで呼び出されるサブルーチンは、スクリプトの実行中に先に出現する **sub** ステートメントで定義される必要があります。

パラメータはサブルーチンにコピーされます。**call** ステートメントのパラメータが数式ではなく変数の場合、パラメータはサブルーチンが終了したときにコピーして戻されます。

制限事項:

- **call** ステートメントは、制御ステートメントであり、セミコロンまたは改行コードで終わるため、行をまたぐことはできません。
- 例えば [if..then] 制御文内の [Sub..end sub] を使用してサブルーチンを定義すると、同じ制御文内からはサブルーチンしか呼び出すことはできません。

この例では、フォルダおよびそのサブフォルダ内のすべての Qlik 関連ファイルを一覧表示し、ファイル情報をテーブルに保存します。Apps という名前のデータ接続がフォルダに作成されていることが前提です。

DoDirサブルーチンは、パラメータとして'lib://Apps'フォルダを参照して呼び出されます。サブルーチンの内部には再帰呼び出しCall DoDir (Dir)が存在します。この関数では、サブフォルダ内でファイルを再帰的に探索します。

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

Call DoDir ('lib://Apps')
```

Do..loop

do..loop 制御ステートメントはスクリプト反復構文で、論理条件が満たされるまで、1 つまたは複数のステートメントを実行します。

構文:

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

> ***do..loop** ステートメントは制御ステートメントであり、セミコロンまたは改行コードで終わっているため、使用可能な 3 つの節 (**do**、**exit do**、**loop**) が行をまたぐことはできません。*

引数:

引数

引数	説明
condition	True または False の評価を実施する論理式。
statements	1 つ以上の Qlik Sense スクリプト ステートメントのグループ。
while / until	**while** または **until** 条件節は、**do..loop** ステートメントに 1 つだけ必要です (例えば、**do** あるいは **loop** の後)。各条件は、初出の場合に限り解釈されますが、ループ内に出現した場合は毎回評価されます。
exit do	**exit do** 節がループ内で出現した場合、スクリプトの実行はループの終了を示す **loop** 節の後の最初のステートメントに移ります。**exit do** 節は、**when** や **unless** サフィックスを使用して条件を付けることができます。

```
// LOAD files file1.csv..file9.csv

Set a=1;

Do while a<10

LOAD * from file$(a).csv;

Let a=a+1;

Loop
```

End

End スクリプト キーワードは、**If** 節、**Sub** 節、**Switch** 節を閉じるために使用されます。

Exit

Exit スクリプト キーワードは **Exit Script** ステートメントの一部ですが、**Do** 節、**For** 節、**Sub** 節から抜けるためにも使用されます。

Exit script

この制御ステートメントは、スクリプトの実行を停止します。スクリプト内の任意の場所に挿入できます。

構文:

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

exit script ステートメントは、制御 ステートメントであり、セミコロンまたは改行 コードで終わるため、行 をまたぐことはできません。

引数:

引数

引数	説明
condition	True または False の評価 を実施する論理式。
when / unless	**exit script** ステートメントは、**when** や**unless** 節 をオプションで使用して、条件 を付けることができます。

```
//Exit script
Exit Script;

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

For..next

for..next 制御 ステートメントは、カウンタ付きのスクリプト反復構文です。**for** と**next** で囲まれたループ内のステートメントは、カウンタ変数の初期値と最終値で指定された回数分実行されます。

構文:

```
For counter = expr1 to expr2 [ step expr3 ]

[statements]

[exit for [ ( when | unless ) condition ]

[statements]

Next [counter]
```

数式 *expr1*、*expr2*、および *expr3* は、ループが最初に挿入される際に評価されます。カウンタ変数の値はループ内のステートメントで変更できますが、これは良いプログラミングとは言えません。

exit for 節がループ内で出現した場合、スクリプトの実行はループの終了を示す **next** 節の後の最初のステートメントに移ります。**exit for** 節は、**when** や **unless** サフィックスを使用して条件を付けることができます。

> ***for..next*** *ステートメントは制御 ステートメントであり、セミコロンまたは改行 コードで終わっているため、使用可能な 3 つの節 (**for..to..step**、**exit for**、**next**) が行 をまたぐことはできません。*

引数:

引数

引数	説明
counter	変数名。*counter* が **next** の後に指定されている場合は、対応する **for** の後に検出されるものと同じ変数名である必要があります。
expr1	ループが実行される *counter* 変数の最初の値を判定する数式。
expr2	ループが実行される *counter* 変数の最後の値を判定する数式。
expr3	ループが実行されるたびに *counter* 変数の増分を示す値を判定する数式。
condition	True または False の評価を実施する論理式。
statements	1 つ以上の Qlik Sense スクリプト ステートメントのグループ。

Example 1: 連続ファイルのロード

```
// LOAD files file1.csv..file9.csv

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

Example 2: ランダムな数のファイルのロード

この例では、*x1.csv*、*x3.csv*、*x5.csv*、*x7.csv*、*x9.csv* というデータファイルが存在すると仮定します。ロードプロセスは、`if rand( )<0.5 then` 条件によってランダム ポイントで停止します。

```
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 制御ステートメントは、コンマ区切りリストの各値に対して、1 つまたは複数のステートメントを実行するスクリプト反復構文です。**for** と **next** で囲まれたループ内のステートメントは、リストの各値で指定された回数分実行されます。

構文:
現在のディレクトリ内のファイルとディレクトリ名のリストの生成を可能にする特殊構文です。

```
for each var in list

[statements]

[exit for [ ( when | unless ) condition ]

[statements]

next [var]
```

引数:

引数

引数	説明
var	ループ実行のたびに、リストから新しい値を取得するスクリプト変数名。**var** が **next** の後に指定されている場合は、対応する **for each** の後に検出されるものと同じ変数名である必要があります。

var 変数の値は、ループ内のステートメントで変更できますが、これは良いプログラミングとは言えません。

exit for 節がループ内で出現した場合、スクリプトの実行はループの終了を示す **next** 節の後の最初のステートメントに移ります。**exit for** 節は、**when** や **unless** サフィックスを使用して条件を付けることができます。

> ***for each..next*** *ステートメントは制御ステートメントであり、セミコロンまたは改行コードで終わっているため、使用可能な 3 つの節 (**for each**、**exit for**、**next**) が行をまたぐことはできません。*

構文:

```
list := item { , item }

item := constant | (expression) | filelist mask | dirlist mask |
fieldvaluelist mask
```

引数

引数	説明
constant	任意の数値または文字列。スクリプトに直接書き込まれた文字列は単一引用符で囲む必要があります。単一引用符で囲まれていない文字列は、変数として解釈され、変数の値が使用されます。数字は単一引用符で囲む必要はありません。
expression	任意の式。
mask	有効なファイル名の文字や、標準的なワイルドカード文字 ***** と **?** を含むファイル名またはディレクトリ名のマスク。 絶対ファイルパスや lib:// パスを使用できます。

引数	説明
condition	True または False の評価を実施する論理式。
statements	1 つ以上の Qlik Sense スクリプト ステートメントのグループ。
filelist mask	この構文は、ファイル名のマスクに一致する現在のディレクトリ内にある、すべてのファイルのコンマ区切りリストを生成します。  *この引数は、標準モードのライブラリ接続のみをサポートします。*
dirlist mask	この構文は、ディレクトリ名のマスクに一致する現在のフォルダ内にある、すべてのディレクトリのコンマ区切りリストを生成します。  *この引数は、標準モードのライブラリ接続のみをサポートします。*
fieldvaluelist mask	この構文は、Qlik Sense にすでにロードされた項目の値を使って繰り返されます。

 *Qlik Web ストレージ プロバイダ コネクタとその他の DataFile 接続は、ワイルドカード (***** および **?**) の使用に対応していません。*

Example 1: ファイルのリストのロード

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

Example 2: ファイル リストをディスクに作成

この例では、フォルダにあるすべての Qlik Sense 関連ファイルのリストをロードしています。

```
sub DoDir (Root)
   for each Ext in 'qvw', 'qva', 'qvo', 'qvs', 'qvc', 'qvf', 'qvd'

      for each File in filelist (Root&'/*.' &Ext)

         LOAD
            '$(File)' as Name,
            FileSize( '$(File)' ) as Size,
            FileTime( '$(File)' ) as FileTime
         autogenerate 1;

      next File

   next Ext
   for each Dir in dirlist (Root&'/*' )
```

```
      call DoDir (Dir)

   next Dir

end sub

call DoDir ('lib://DataFiles')
```

Example 3: 項目の値を使って繰り返し

この例は、ロードされた値のリストである FIELD を使って繰り返され、新しい項目 NEWFIELD を生成します。FIELD の 1 つの値につき、2 つの NEWFIELD レコードが作成されます。

```
load * inline [
FIELD
one
two
three
];

FOR Each a in FieldValueList('FIELD')
LOAD '$(a)' &'-'&RecNo() as NEWFIELD AutoGenerate 2;
NEXT a
```

この結果、テーブルは次のようになります。

Example table

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

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

if..then 制御ステートメントは、1 つ以上の論理条件に応じて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

通常、制御ステートメントはスクリプト実行の流れを制御するために使用されます。チャートの数式では、代わりに **if** 条件付き関数を使用してください。

構文:

```
If  condition then

  [ statements ]

{ elseif condition then
```

```
  [ statements ] }

[ else

  [ statements ] ]

end if
```

if..then ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、使用可能な 4 つの節 (**if..then**、**elseif..then**、**else**、**end if**) が行をまたぐことはできません。

引数:

引数

引数	説明
condition	True か False で評価できる論理式です。
statements	1 つ以上の Qlik Sense スクリプト ステートメントのグループ。

Example 1:

```
if a=1 then
      LOAD * from abc.csv;

      SQL SELECT e, f, g from tab1;
end if
```

Example 2:

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

Example 3:

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

Next

Next スクリプト キーワードは、**For** ループを閉じるために使用されます。

Sub..end sub

sub..end sub 制御ステートメントは、**call** ステートメントで呼び出されるサブルーチンを定義します。

構文:

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

引数はサブルーチンにコピーされ、**call** ステートメントで対応する実パラメータが変数名の場合は、サブルーチンの終了後、コピーして戻されます。

サブルーチンに **call** ステートメントで渡される実パラメータよりも仮パラメータが多い場合は、余分なパラメータは NULL に初期化され、サブルーチン内でローカル変数として使用できます。

引数:

引数

引数	説明
name	サブルーチンの名前。
paramlist	サブルーチンの仮パラメータの変数名のコンマ区切りリスト。これはサブルーチン内の変数として使用できます。
statements	1 つ以上の Qlik Sense スクリプトステートメントのグループ。

制限事項:

- **sub** ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、2 つの節 (**sub**、**end sub**) が行をまたぐことはできません。
- 例えば [`if..then`] 制御文内の [`Sub..end sub`]を使用してサブルーチンを定義すると、同じ制御文内からはサブルーチンしか呼び出すことはできません。

Example 1:

```
Sub INCR (I,J)

I = I + 1

Exit Sub when I < 10

J = J + 1

End Sub

Call INCR (X,Y)
```

Example 2: - パラメータ転送

```
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)
```

上記の結果、サブルーチン内でローカルに A は 1、B は 4、C は NULL に初期化されます。

サブルーチンを終了する際、グローバル変数 A は 2 を値として取得します (サブルーチンからコピーして返されます)。2 番目の実パラメータ"(X+1)*2" は変数ではないため、コピーして返されません。最後に、グローバル変数 C はサブルーチン呼び出しの影響を受けません。

Switch..case..default..end switch

switch 制御ステートメントは、数式の値に基づいて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

構文:

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

> ***switch** ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、使用可能な 4 つの節 (**switch**、**case**、**default**、**end switch**) が行をまたぐことはできません。*

引数:

引数

引数	説明
expression	任意の式。
valuelist	比較される数式の値のコンマ区切りのリスト。スクリプトの実行は、値リストの値が数式の値と等しい最初のグループのステートメントで続行されます。値リストの各値は、任意の数式の場合があります。**case** 節で一致しない場合は、**default** 節 (指定した場合) のステートメントが実行されます。
statements	1 つ以上の Qlik Sense スクリプトステートメントのグループ。

```
Switch I
```

```
Case 1

LOAD '$(I): CASE 1' as case autogenerate 1;

Case 2

LOAD '$(I): CASE 2' as case autogenerate 1;

Default

LOAD '$(I): DEFAULT' as case autogenerate 1;

End Switch
```

To

To スクリプト キーワードは、次のスクリプト ステートメントで使用されます。

3.2 スクリプトのプレフィックス

プレフィックスは、必要に応じて正規ステートメントに適用できますが、制御ステートメントには適用できません。ただし、**when** および **unless** プレフィックスは、一部の制御ステートメント節のサフィックスとして使用できます。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。ただし、ステートメントで使用される項目名と変数名は大文字と小文字が区別されます。

スクリプトのプレフィックスの概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Add

スクリプト内の任意の **LOAD** または **SELECT** ステートメントに **Add** プレフィックスを追加して、別のテーブルにレコードを追加するように指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。**Add** プレフィックスは **Map** ステートメントでも使用できます。

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

```
Add [ Only ] mapstatement
```

Buffer

QVD ファイルは、**buffer** プレフィックスを使用して、自動的に作成、管理することができます。このプレフィックスは、スクリプトの **LOAD** ステートメントおよび **SELECT** ステートメントのほとんどで使用できます。つまり、ステートメントの結果をキャッシュ/バッファする際には、QVD ファイルが使用されます。

```
Buffer[(option [ , option])] ( loadstatement | selectstatement )
```

```
option::= incremental | stale [after] amount [(days | hours)]
```

Concatenate

連結される2つのテーブルに異なる項目セットが存在する場合、**Concatenate** プレフィックスを使用すると2つのテーブルを強制的に連結できます。

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

Crosstable

crosstable ロードプレフィックスは、「クロス テーブル」または「ピボット テーブル」の構造化データを転置するために使用されます。このように構造化されたデータは、スプレッドシート ソースを操作するときによく見られます。**crosstable** ロードプレフィックスの出力と目的は、このような構造を通常の列指向のテーブルに変換することです。これは、この構造のほうが Qlik Sense での分析に適しているためです。

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

First

First または **LOAD** ステートメントへの **SELECT (SQL)** プレフィックスは、データ ソース テーブルから最大レコード数をロードする際に使用します。

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

Generic

Generic ロードプレフィックスを使用すると、エンティティ属性値モデル化データ(EAV) を従来の正規化されたリレーショナル テーブル構造に変換できます。EAV モデリングは、「汎用データ モデリング」または「オープン スキーマ」とも呼ばれます。

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

Hierarchy

hierarchy プレフィックスを使用して、親子階層テーブルを、Qlik Sense データモデルで有用なテーブルに変換します。これは、**LOAD** や **SELECT** ステートメントの前に置き、ロード ステートメントの結果をテーブル変換の入力として使用します。

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

HierarchBelongsTo

このプレフィックスを使用して、親子階層テーブルを、Qlik Sense データモデルで有用なテーブルに変換します。これは、**LOAD** や **SELECT** ステートメントの前に置き、ロード ステートメントの結果をテーブル変換の入力として使用します。

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

Inner

join および **keep** プレフィックスの前には、プレフィックス **inner** を置くことができます。

join の前に使用すると、内部結合を指定できます。結果のテーブルには、生データテーブルからの項目値の組み合わせのみが含まれます。連結項目値は双方のテーブルに示されます。**keep** の前に使用すると、Qlik Sense に保存される前に、双方の生データテーブルが共通部分に縮小されます。
を参照してください。

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

IntervalMatch

IntervalMatch プレフィックスを使うと、不連続数値を 1 つ以上の数値間隔に一致させるテーブル、そしてオプションとして 1 つ以上の追加キーの値を一致させるテーブルを作成できます。

```
IntervalMatch (matchfield)(loadstatement | selectstatement )
```

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

Join

join プレフィックスは、ロード済みのテーブルを名前が付いた既存テーブルまたは直前に作成されたデータテーブルと結合します。

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

Keep

keep プレフィックスは **join** プレフィックスに類似しています。**join** プレフィックスのように、ロード済みテーブルと既存の名前付きテーブルまたは直前に作成されたデータテーブルを比較します。しかし、ロード済みテーブルと既存のテーブルを結合する代わりに、テーブル データの共通部分に基づき、Qlik Sense に保存される前に、2 つのうち一方または両方のテーブルを縮小する効果があります。実施された比較は、すべての共通項目で行われる自然結合に相当します (対応する結合と同じ方法など)。ただし、2 つのテーブルは結合されず、別の名前付きテーブルとして Qlik Sense に保存されます。

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

Left

Join および **Keep** プレフィックスの前には、プレフィックス **left** を置くことができます。

join の前に使用すると、左結合を指定します。結果のテーブルには、生データテーブルからの項目値の組み合わせのみが含まれます。連結項目値は最初のテーブルに示されます。**keep** の前に使用すると、Qlik Sense に保存される前に、2 つ目の生データテーブルは 1 つ目のテーブルとの共通部分に縮小されます。

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

Mapping

mapping プレフィックスは、マッピング テーブルの作成に使用します。マッピング テーブルは、スクリプト実行中に項目値と項目名を置き換えるといった操作で使用できます。

```
マッピング  ( loadstatement | selectstatement )
```

Merge

Merge プレフィックスをスクリプト内の任意の **LOAD** または **SELECT** ステートメントに追加して、ロードされたテーブルを別のテーブルに統合する必要があることを指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。

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

NoConcatenate

NoConcatenate プレフィックスは、同一の項目セットでロードされた 2 つのテーブルを、強制的に別個の内部テーブルとして扱います (そうでない場合、自動的に連結されます)。

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

Outer

外部結合を指定するために、明示的な **Join** プレフィックスの前にプレフィックス **Outer** を付加できます。外部結合では、2 つのテーブル間のすべての組み合わせが生成されます。結果のテーブルには、生データテーブルからの項目値の組み合わせが含まれます。連結項目値は一方または双方のテーブルに示されます。**Outer**キーワードはオプションで、結合プレフィックスが指定されていない場合のデフォルトの結合タイプです。

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

Partial reload

フル リロードは、常に既存のデータモデルのすべてのテーブルを削除することから始まり、次にロードスクリプトを実行します。

A *部分的なリロード (page 100)* はこれを行いません。代わりに、データモデル内のすべてのテーブルを保持し、[**Add**]、[**Merge**]、または [**Replace**] プレフィックスが前に付いた [**Load**] ステートメントと [**Select**] ステートメントのみを実行します。他のデータテーブルはコマンドの影響を受けません。引数 **only** は、ステートメントが部分的なリロード中にのみ実行され、フル リロード中には無視されることを示します。次の表は、部分的および完全なリロードのステートメントの実行の概要です。

Replace

Replace プレフィックスをスクリプト内の任意の **LOAD** または **SELECT** ステートメントに追加して、ロードされたテーブルを別のテーブルに置き換えるように指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。**Replace** プレフィックスは **Map** ステートメントでも使用できます。

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

```
Replace [only] mapstatement
```

Right

Join および **Keep** プレフィックスの前には、プレフィックス **right** を置くことができます。

join の前に使用すると、右結合を指定します。結果のテーブルには、生データテーブルからの項目値の組み合わせのみが含まれます。連結項目値は 2 番目のテーブルに示されます。**keep** の前に使用すると、Qlik Sense に保存される前に、1 つ目の生データテーブルは 2 つ目のテーブルとの共通部分に縮小されます。

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

Sample

sample または **LOAD** ステートメントの **SELECT** プレフィックスは、データ ソースからランダムにレコード サンプルをロードする際に使用します。

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

Semantic

semantic プレフィックスを使用すると、レコード間の関係を含むテーブルをロードできます。これはテーブル内における、親 (会社) や所属先、前任者といった 1 つのレコード ポイントから別のポイントへの自己参照です。

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

Unless

unless プレフィックスとサフィックスは、条件節の作成に使用します。条件節は、ステートメントまたは exit 節を評価するかどうかを決定します。これは、**if..end if** ステートメントの簡単な代替として使用されることがあります。

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

When

when プレフィックスとサフィックスは、条件節の作成に使用します。条件節は、ステートメントまたは exit 節を実行するかどうかを決定します。これは、**if..end if** ステートメントの簡単な代替として使用されることがあります。

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

Add

スクリプト内の任意の **LOAD** または **SELECT** ステートメントに **Add** プレフィックスを追加して、別のテーブルにレコードを追加するように指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。**Add** プレフィックスは **Map** ステートメントでも使用できます。

> *部分的なリロードが正しく機能するためには、部分的なリロードがトリガーされる前に、アプリをデータで開く必要があります。*

[**リロード**] ボタンを使用して部分的なリロードを実行します。Qlik Engine JSON API を使用することもできます。

構文:

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

```
Add [only] mapstatement
```

通常の (部分的ではない) リロード中、**Add LOAD** 構造は通常の **LOAD** ステートメントとして機能します。レコードが生成され、テーブルに保存されます。

Concatenate プレフィックスが使用されている場合、または同じ項目のセットを持つテーブルが存在する場合、レコードは関連する既存のテーブルに追加されます。それ以外の場合、**Add LOAD** 構造は新しいテーブルを作成します。

部分的なリロードでも同じことができます。唯一の違いは、**Add LOAD** 構造が新しいテーブルを作成しないことです。レコードを追加する必要がある前のスクリプト実行からの関連テーブルが常に存在します。

その際、重複チェックは行われないため、**Add** プレフィックスを使用するステートメントには、多くの場合、重複を防ぐdistinct 修飾子または where 節を記述します。

Add Map...Using ステートメントでは、パーシャル スクリプトの実行中もマッピングが発生します。

引数:

引数

引数	説明
only	ステートメントが部分的なリロード中にのみ実行される必要があることを示すオプションの修飾子。通常の (部分的ではない) リロード中は無視する必要があります。

例と結果:

例	結果
Tab1: LOAD Name, Number FROM Persons.csv; Add LOAD Name, Number FROM newPersons.csv;	通常のリロードでは、データは *Persons.csv* からロードされ、Qlik Sense テーブル Tab1 に保存されます。*NewPersons.csv* のデータは、同じ Qlik Sense テーブルに連結されます。 パーシャル リロードを実行している場合、データは *NewPersons.csv* からロードされ、Qlik Sense テーブル Tab1 に追加されます。重複チェックは実行されません。
Tab1: SQL SELECT Name, Number FROM Persons.csv; Add LOAD Name, Number FROM NewPersons.csv where not exists (Name);	重複チェックは、Name が以前にロードされたテーブル データに存在するかどうか確認することで行われます。 通常のリロードでは、データは *Persons.csv* からロードされ、Qlik Sense テーブル Tab1 に保存されます。*NewPersons.csv* のデータは、同じ Qlik Sense テーブルに連結されます。 パーシャル リロードを実行している場合、データは *NewPersons.csv* からロードされ、Qlik Sense テーブル Tab1 に追加されます。重複チェックは、Name が以前にロードされたテーブル データに存在するかどうか確認することで行われます。
Tab1: LOAD Name, Number FROM Persons.csv; Add Only LOAD Name, Number FROM NewPersons.csv where not exists(Name);	通常のリロードでは、データは *Persons.csv* からロードされ、Qlik Sense テーブル Tab1 に保存されます。*NewPersons.csv* をロードするステートメントは無視されます。 パーシャル リロードを実行している場合、データは *NewPersons.csv* からロードされ、Qlik Sense テーブル Tab1 に追加されます。重複チェックは、Name が以前にロードされたテーブル データに存在するかどうか確認することで行われます。

Buffer

QVD ファイルは、**buffer** プレフィックスを使用して、自動的に作成、管理することができます。このプレフィックスは、スクリプトの **LOAD** ステートメントおよび **SELECT** ステートメントのほとんどで使用できます。つまり、ステートメントの結果をキャッシュ/バッファする際には、QVD ファイルが使用されます。

構文:

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

オプションを使用していない場合、最初のスクリプト実行で作成された QVD バッファが無限に使用されます。

バッファファイルは *Buffers* サブフォルダに保存されます。通常は、*C:\ProgramData\Qlik\Sense\Engine\Buffers* (サーバー インストール環境) または *C:\Users\{user}\Documents\Qlik\Sense\Buffers* (Qlik Sense Desktop) となります。

QVD ファイルの名前は計算で求められた名前であり、後続の **LOAD** または **SELECT** ステートメントとその他の識別情報の全体の 160 ビットの 16 進ハッシュとなります。つまり、QVD バッファは、後続の **LOAD** または **SELECT** ステートメントの変更によって無効になります。

通常、バッファを作成したアプリのスクリプトで一切参照されなくなったり、アプリが存在しなくなると、QVD バッファは削除されます。

引数:

引数

引数	説明
incremental	incremental オプションを使用すると、基底ファイルの一部のみを読み取る機能が有効になります。以前のファイル サイズは、QVD ファイルの XML ヘッダーに保存されます。これは、ログ ファイルで特に便利です。過去にロードされたレコードは、すべて QVD ファイルから読み取られますが、以降の新しいレコードについては元のソースから読み取った上で QVD ファイルを更新します。 incremental オプションは **LOAD** ステートメントとテキスト ファイルでのみ使用できます。古いデータが変更または削除された場合、増分ロードは使用できません。
stale [after] amount [(days | hours)]	amount は期間を指定する数字で、10 進数を使用できます。単位が省略されている場合は、日数と見なされます。 通常、stale after オプションは、元のデータに一般的なタイムスタンプがない DB ソースで使用します。そのため、それ以外の場合は QVD スナップショットを使用できる期間を指定します。stale after 節は、QVD バッファが作成されてから有効期限切れになるまでの期間を指定します。それまでの間、QVD バッファがデータ ソースとして使用され、期間終了後は元のデータ ソースが使用されます。その後、QVD バッファ ファイルが自動更新され、新しい期間が開始します。

制限事項:

このスクリプトには、多くの制限が存在します。最も代表的な例としては、複雑なステートメントの核にファイル **LOAD** または **SELECT** ステートメントを含めなければならないという条件が挙げられます。

Example 1:

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

Example 2:

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

Example 3:

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

Concatenate

Concatenate は、データセットを既存のメモリ内テーブルに吹きできるスクリプトロードプレフィックスです。これは、異なるトランザクションデータのセットを単一の中央ファクトテーブルに追加したり、複数のソースから取得される特定タイプの共通参照データセットを構築したりするためによく使われます。SQL UNION 演算子の機能と似ています。

concatenate 操作で出力されたテーブルには、そのテーブルの下にデータの新しい行が付記された元のデータセットが含まれます。ソースとターゲットテーブルには、異なる項目が含まれている可能性があります。項目が異なる場合、出力されるテーブルは、ソーステーブルとターゲットテーブルの両方に存在するすべての項目を組み合わせた結果を評価するために幅が広くなります。

構文:

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

引数

引数	説明
tablename	既存テーブルの名前。命名されたテーブルは Concatenate 操作のターゲットであり、ロードされたレコードはそのテーブルに追加されます。tablename パラメータが使用されない場合、ターゲットテーブルはこのステートメントの前に最後にロードされたテーブルとなります。
loadstatement/selectstatement	tablename 引数の後の loadstatement/selectstatement 引数は、指定したテーブルに連結されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
Concatenate (Transactions) Load …. ;	Concatenate プレフィックスの下にある Load ステートメントにロードされたデータは、Transactions という既存のメモリ内テーブルに追加されます (Transactions というテーブルが、ロードスクリプトでこの時点までにロードされたという前提)。

例 1 – 連結ロードプレフィックスを使って、複数のデータセットをターゲットテーブルに追加する

ロードスクリプトと結果

概要

この例では、2 つのスクリプトを順にロードします。

- 最初のロードスクリプトには、Transactions というテーブルに送信される日付と金額を含む初期データセットが含まれています。
- 2 番目のロードスクリプトには次の内容が含まれます:
  - Concatenate プレフィックスを使って初期データセットに追加される 2 番目のデータセット。このデータセットには、初期データセットにはなかった追加項目の type があります。
  - Concatenate プレフィックス。

データロードエディタを開き、以下のロードスクリプトを新しいタブに追加します。

最初のロードスクリプト

```
Transactions:
Load * Inline [

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount

最初のロード スクリプト結果テーブル

ID	日付	amount
3750	08/30/2018	23.56
3751	09/07/2018	556.31
3752	09/16/2018	5.75
3753	09/22/2018	125.00
3754	09/22/2018	484.21
3756	09/22/2018	59.18
3757	09/23/2018	177.42

テーブルには初期データセットが表示されています。

2 番目のロード スクリプト

データ ロード エディタを開き、以下のロード スクリプトを追加します。

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

結果

データをロードしてシートに移動します。この項目を軸として作成します。

- type

2 番目のロード スクリプト結果テーブル

ID	日付	amount	タイプ
3750	08/30/2018	23.56	-
3751	09/07/2018	556.31	-
3752	09/16/2018	5.75	-

ID	日付	amount	タイプ
3753	09/22/2018	125.00	-
3754	09/22/2018	484.21	-
3756	09/22/2018	59.18	-
3757	09/23/2018	177.42	-
3758	10/01/2018	164.27	内部
3759	10/03/2018	384.00	外部
3760	10/06/2018	25.82	内部
3761	10/09/2018	312.00	内部
3762	10/15/2018	4.56	内部
3763	10/16/2018	90.24	内部
3764	10/18/2018	19.32	外部

最初の 7 つのレコードの [type] 項目の null 値がロードされ、type が定義されていないことに注意してください。

例 2 – 黙示的連結を使って、複数のデータセットをターゲットテーブルに追加する

ロードスクリプトと結果

概要

黙示的にデータを追加する典型的使用例は、同じ構造のデータを持つ複数のファイルをロードし、それらをすべてターゲットテーブルに追加する方法です。

例えば、次のような構文を使ってファイル名に wildcards を使用します:

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

または次のような構造を使ってループに使用します:

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

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

next file
```

黙示的連結は、たとえスクリプト内で互いに後に定義されていなくても、同名の項目がロードされた 2 つのテーブル間で行われます。これにより、データが予定外にテーブルに追加される可能性があります。同一項目を持つセカンダリテーブルをこのように追加したくない場合は、NoConcatenate ロードプレフィックスを使います。別のテーブル名タグでテーブルの名称を変更しても、黙示的連結を防止するのに不十分です。詳細については、「NoConcatenate (page 90)」を参照してください。

この例では、2 つのスクリプトを順にロードします。

- 最初のロードスクリプトには、Transactions というテーブルに送信される 4 つの項目を含む初期データセットが含まれています。
- 2 番目のロードスクリプトには、最初のデータセットとして同一項目を持つデータセットが含まれます。

データロードエディタを開き、以下のロードスクリプトを新しいタブに追加します。

最初のロードスクリプト

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount
- type

最初のロードスクリプト結果テーブル

ID	日付	タイプ	amount
3758	10/01/2018	内部	164.27
3759	10/03/2018	外部	384.00
3760	10/06/2018	内部	25.82
3761	10/09/2018	内部	312.00
3762	10/15/2018	内部	4.56
3763	10/16/2018	内部	90.24
3764	10/18/2018	外部	19.32

テーブルには初期データセットが表示されています。

2 番目のロードスクリプト

データロードエディタを開き、以下のロードスクリプトを追加します。

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

結果

データをロードしてシートに移動します。

2 番目のロード スクリプト結果テーブル

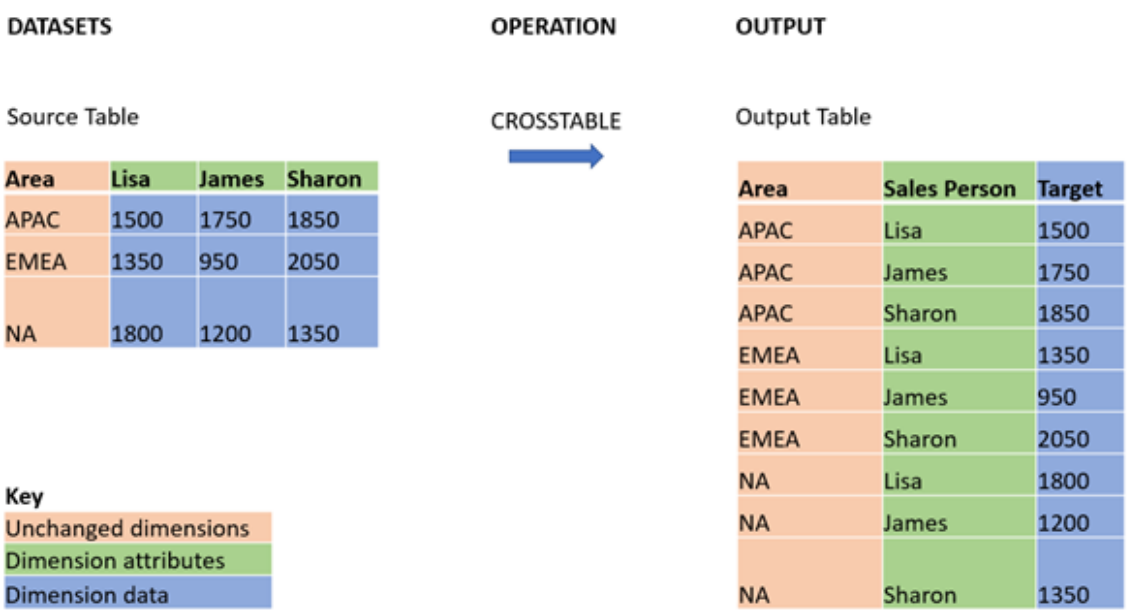
ID	日付	タイプ	amount
3758	10/01/2018	内部	164.27
3759	10/03/2018	外部	384.00
3760	10/06/2018	内部	25.82
3761	10/09/2018	内部	312.00
3762	10/15/2018	内部	4.56
3763	10/16/2018	内部	90.24
3764	10/18/2018	外部	19.32
3765	11/03/2018	内部	129.40
3766	11/05/2018	外部	638.50

2 番目のデータセットは、同一項目があったため、初期データセットに黙示的に連結されました。

Crosstable

crosstable ロードプレフィックスは、「クロス テーブル」または「ピボット テーブル」の構造化データを転置するために使用されます。このように構造化されたデータは、スプレッドシート ソースを操作するときによく見られます。**crosstable** ロードプレフィックスの出力と目的は、このような構造を通常の列指向のテーブルに変換することです。これは、この構造のほうが Qlik Sense での分析に適しているためです。

クロステーブルとして構造化されたデータとクロステーブル変換後の同等の構造の例



構文:

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

引数

引数	説明
attribute field name	転置される水平方向の軸 (ヘッダー行) を説明する目的の出力項目名。
data field name	転置される軸の水平方向のデータ(ヘッダー行の下のデータ値のマトリクス) をを説明する目的の出力項目名。
n	汎用的な形式に変換されるテーブルに先行する修飾子項目、または変更されなかった軸の数。既定値は 1 です。

このスクリプト関数は、次の関数に関連しています。

関連する関数

関数	相互作用
Generic (page 57)	エンティティ属性値の構造化データセットを取得し、それを通常のリレーショナル テーブル構造に変換して、検出された各属性をデータの新しい項目または列に分離する変換ロードプレフィックス。

例 1 – ピボットされた売上データの変換 (単純)

ロードスクリプトと結果

概要

データロードエディターを開き、以下の最初のロードスクリプトを新しいタブに追加します。

最初のロードスクリプトには、crosstable スクリプトプレフィックスが後で適用されるデータセットが含まれており、crosstable を適用するセクションはコメントアウトされています。これは、ロードスクリプトでこのセクションを無効にするためにコメント構文が使用されたことを意味します。

2 番目のロードスクリプトは最初のスクリプトと同じですが、crosstable の適用がコメント解除されています (コメント構文を削除することで有効になります)。スクリプトがこう表示されているのは、データの変換におけるこのスクリプト関数の価値を強調するためです。

最初のロードスクリプト (関数は適用されません)

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

//Final:
//Load Product,
//Date(Date#(MonthText,'MMM YYYY'),'MMM YYYY') as Month,
//Sales

//Resident tmpData;

//Drop Table tmpData;
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

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

結果 テーブル

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

このスクリプトは、月ごとに1つの列、製品ごとに1つの行を持つcrosstableを示しています。現在の形式では、このデータを分析するのは簡単ではありません。すべての数値を1つの項目に、すべての月を別の項目、3列のテーブルに含める方がはるかに良いでしょう。次のセクションでは、この変換をcrosstableに対して行う方法について説明します。

2番目のロードスクリプト(関数が適用されます)

// を削除して、スクリプトのコメントを解除します。これで、ロードスクリプトは次のようになります。

```
tmpData:
Crosstable (MonthText, Sales)
Load * inline [
Product, Jan 2021, Feb 2021, Mar 2021, Apr 2021, May 2021, Jun 2021
A, 100, 98, 103, 63, 108, 82
B, 284, 279, 297, 305, 294, 292
C, 50, 53, 50, 54, 49, 51];

Final:
Load Product,
Date(Date#(MonthText,'MMM YYYY'),'MMM YYYY') as Month,
Sales

Resident tmpData;

Drop Table tmpData;
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- Product
- Month
- Sales

結果 テーブル

製品	月	売上
A	Jan 2021	100

製品	月	売上
A	Feb 2021	98
A	Mar 2021	103
A	Apr 2021	63
A	May 2021	108
A	Jun 2021	82
B	Jan 2021	284
B	Feb 2021	279
B	Mar 2021	297
B	Apr 2021	305
B	May 2021	294
B	Jun 2021	292
C	Jan 2021	50
C	Feb 2021	53
C	Mar 2021	50
C	Apr 2021	54
C	May 2021	49
C	Jun 2021	51

スクリプトプレフィックスが適用されると、crosstable は `Month` に 1 つの列、`Sales` に 1 つの列を持つストレートテーブルに変換されます。これにより、データが読みやすくなります。

例 2 – ピボットされた売上目標データを垂直テーブル構造に変換する (中間)

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Targets」というテーブルにロードされるデータセット。
- `crosstable` ロードプレフィックス。ピボットされた営業担当者の名前を `Sales Person` というラベルの付いた独自の項目に置き換えます。
- `Target` という項目に構造化された、関連する売上目標データ。

ロード スクリプト

```
SalesTargets:
CROSSTABLE([Sales Person],Target,1)
LOAD
*
INLINE [
Area, Lisa, James, Sharon
APAC, 1500, 1750, 1850
EMEA, 1350, 950, 2050
NA, 1800, 1200, 1350
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- Area
- Sales Person

このメジャーを追加します。

```
=Sum(Target)
```

結果テーブル

エリア	営業 担当者	=Sum(Target)
APAC	James	1750
APAC	Lisa	1500
APAC	Sharon	1850
EMEA	James	950
EMEA	Lisa	1350
EMEA	Sharon	2050
NA	James	1200
NA	Lisa	1800
NA	Sharon	1350

ピボットされた入力テーブルとしてデータの表示を複製する場合は、シートに同等のピボット テーブルを作成できます。

次の手順を実行します。

1. 作成したテーブルをコピーしてシートに貼り付けます。
2. **ピボット テーブル** チャート オブジェクトを、新しく作成したテーブル コピーの上にドラッグします。**[変換]** を選択します。

3. [✔ 編集の完了] をクリックします。
4. Sales Person 項目を垂直列シェルフから水平列シェルフにドラッグします。

次の表は、Qlik Senseに表示されているように、最初の表形式のデータを示しています。

Qlik Sense に示すように、元の結果テーブル

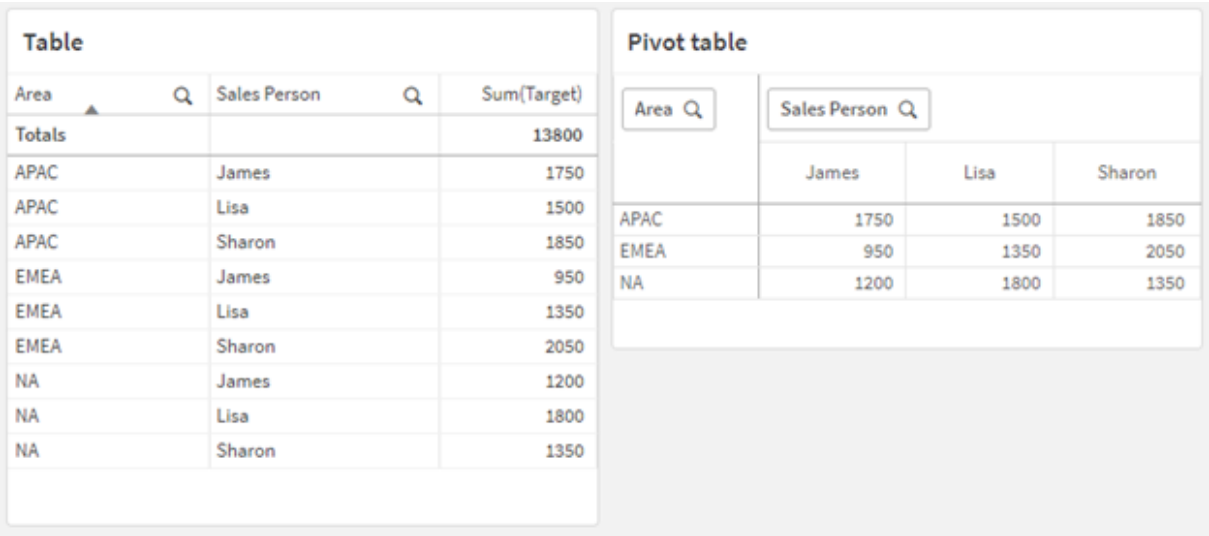
エリア	営業 担当者	=Sum(Target)
合計	-	13800
APAC	James	1750
APAC	Lisa	1500
APAC	Sharon	1850
EMEA	James	950
EMEA	Lisa	1350
EMEA	Sharon	2050
NA	James	1200
NA	Lisa	1800
NA	Sharon	1350

同等のピボットテーブルは次のように、各営業担当者の名前の列は Sales Personの大きな行に含まれる形式になります。

Sales Person 項目が水平方向にピボットされた同等のピボットテーブル

エリア	James	Lisa	Sharon
APAC	1750	1500	1850
EMEA	950	1350	2050
NA	1350	1350	1350

テーブルとして表示されたデータ、および Sales Person 項目が水平方向にピボットされた同等のピボットテーブルの例



例 3 – ピボットされた売上および目標データを垂直テーブル構造に変換する (上級)

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 地域と月ごとに編成された売上と目標のデータを表すデータセット。これは、「SalesAndTargets」というテーブルにロードされます。
- crosstable ロードプレフィックス。これは、Month Year 軸を専用項目にピボット解除するために使用され、売上と目標金額のマトリックスを Amount と呼ばれる専用項目に転置するのにも使用されます。
- テキストから日付への変換関数 date#を使用した、テキストから適切な日付への Month Year 項目の変換。この日付変換された Month Year 項目は、Join ロードプレフィックスを介して SalesAndTarget テーブルに結合されます。

ロードスクリプト

```
SalesAndTargets:
CROSSTABLE(MonthYearAsText,Amount,2)
LOAD
	*
INLINE [
Area	Type	Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Aug-22	Sep-22	Oct-22	Nov-22	Dec-22
APAC	Target	425	425	425	425	425	425	425	425	425	425	425	425
APAC	Actual	435	434	397	404	458	447	413	458	385	421	448	397
EMEA	Target	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5	362.5
EMEA	Actual	363.5	359.5	337.5	361.5	341.5	337.5	379.5	352.5	327.5	337.5	360.5	334.5
NA	Target	375	375	375	375	375	375	375	375	375	375	375	375
NA	Actual	378	415	363	356	403	343	401	365	393	340	360	405
```

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

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

JOIN (SalesAndTargets)
LOAD * RESIDENT tmp;

DROP TABLE tmp;
DROP FIELD MonthYearAsText;
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- Area
- Month Year

次のメジャーを、ラベル Actual を使って作成します。

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

またこのメジャーを、ラベル Target を使って作成します。

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

結果テーブル (切り抜き)

エリア	月 年	実績	対象
APAC	Jan-22	435	425
APAC	Feb-22	434	425
APAC	Mar-22	397	425
APAC	Apr-22	404	425
APAC	May-22	458	425
APAC	Jun-22	447	425
APAC	Jul-22	413	425
APAC	Aug-22	458	425
APAC	Sep-22	385	425
APAC	Oct-22	421	425
APAC	Nov-22	448	425
APAC	Dec-22	397	425
EMEA	Jan-22	363.5	362.5
EMEA	Feb-22	359.5	362.5

ピボットされた入力テーブルとしてデータの表示を複製する場合は、シートに同等のピボットテーブルを作成できます。

次の手順を実行します。

1. 作成したテーブルをコピーしてシートに貼り付けます。
2. **ピボットテーブル** チャートオブジェクトを、新しく作成したテーブル コピーの上にドラッグします。**[変換]** を選択します。
3. **[✔ 編集の完了]** をクリックします。
4. `Month Year` 項目を垂直列シェルフから水平列シェルフにドラッグします。
5. `Values` アイテムを、垂直列シェルフから水平列シェルフにドラッグします。

次の表は、Qlik Sense に表示されているように、最初のテーブル形式のデータを示しています。

Qlik Sense に示すように、元の結果テーブル (切り抜き)

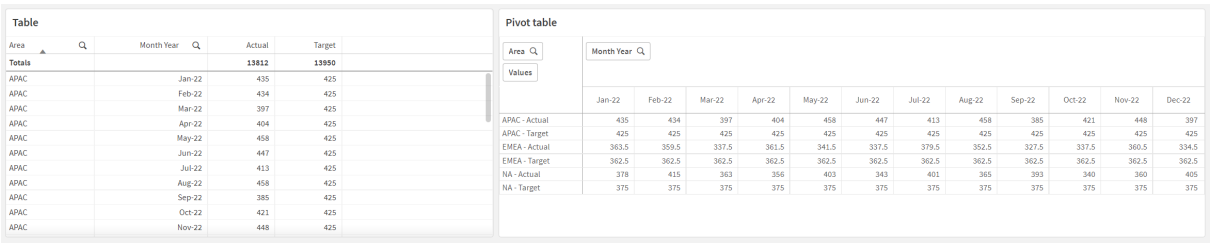
エリア	月 年	実績	対象
合計	-	13812	13950
APAC	Jan-22	435	425
APAC	Feb-22	434	425
APAC	Mar-22	397	425
APAC	Apr-22	404	425
APAC	May-22	458	425
APAC	Jun-22	447	425
APAC	Jul-22	413	425
APAC	Aug-22	458	425
APAC	Sep-22	385	425
APAC	Oct-22	421	425
APAC	Nov-22	448	425
APAC	Dec-22	397	425
EMEA	Jan-22	363.5	362.5
EMEA	Feb-22	359.5	362.5

同等のピボットテーブルは次のように、その年の個別の月の列は `Month Year`の大きな行に含まれる形式になります。

Month Year 項目が水平方向にピボットされた同等のピボット テーブル (切り抜き)

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

テーブルとして表示されたデータ、および Month Year 項目が水平方向にピボットされた同等のピボット テーブルの例



First

First または LOAD (SQL) ステートメントへの SELECT プレフィックスは、データ ソース テーブルから最大レコード数をロードする際に使用します。First プレフィックスを使用する一般的なユース ケースは、大きなレコードから小さなサブセットのレコードを取得する場合および/またはデータのロード ステップが遅い場合です。定義された「n」数のレコードがロードされ次第、ロード ステップが途中で終了し、スクリプトの残りの実行が通常どおり続行されます。

構文:

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

引数

引数	説明
n	読み取り対象の最大レコード件数を示す整数を評価する任意の数式。n は (n) のように括弧で囲む場合があります。
loadstatement | selectstatement	n 引数に続くload statement/select statement は、設定された最大レコード数でロードする必要がある指定されたテーブルを定義します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
FIRST 10 LOAD * from abc.csv;	この例では、Excel ファイルから最初の 10 行を取得します。
FIRST (1) SQL SELECT * from Orders;	この例では、Orders データセットから最初に選択された行を取得します。

例 – 最初の 5 行をロードする

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2020 年の最初の 2 週間の日付のデータセット。
- 最初の 5 つのレコードのみをロードするようにアプリケーションに指示する First 変数。

ロードスクリプト

```
Sales:
FIRST 5
LOAD
*
Inline [
date,sales
```

```
01/01/2020,6000
01/02/2020,3000
01/03/2020,6000
01/04/2020,8000
01/05/2020,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
01/12/2020,7000
01/13/2020,7000
01/14/2020,7000
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、Date を項目として、sum(sales) をメジャーとして追加します。

結果テーブル

Date	Sum(Sales)
01/01/2020	6000
01/02/2020	3000
01/03/2020	6000
01/04/2020	8000
01/05/2020	5000

このスクリプトは、Sales テーブルの最初の 5 つのレコードのみをロードします。

Generic

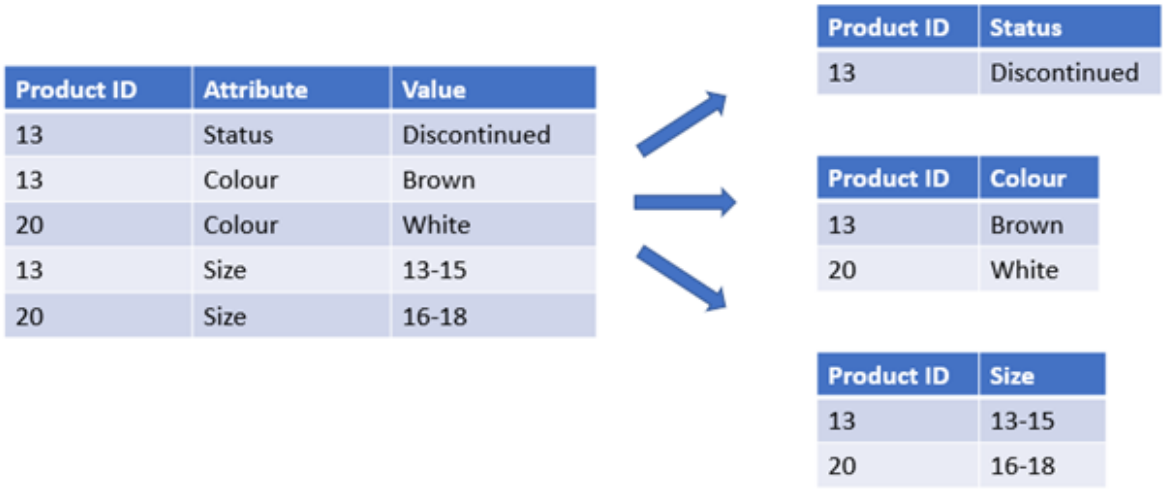
Generic ロードプレフィックスを使用すると、エンティティ属性値モデル化データ (EAV) を従来の正規化されたリレーショナル テーブル構造に変換できます。EAV モデリングは、「汎用データ モデリング」または「オープン スキーマ」とも呼ばれます。

EAV モデル データと同等の非正規化 リレーショナル テーブルの例

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

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

EAV モデル データと同等の一連の正規化 リレーショナル テーブルの例



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

Product ID	Status
13	Discontinued

Product ID	Colour
13	Brown
20	White

Product ID	Size
13	13-15
20	16-18

Qlik で EAV モデル化 されたデータを読み込んで分析することは技術的に可能ですが、多くの場合、同等の従来のリレーショナル データ構造を使用する方が簡単です。

構文:

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

これらのトピックは、この関数を使用するのに役立つかもしれません。

関連 トピック

トピック	説明
Crosstable (page 45)	Crosstable ロードプレフィックスは、水平方向のデータを垂直方向のデータに変換します。純粋に機能的な観点からは、Generic ロードプレフィックスとは逆の変換を実行しますが、プレフィックスは通常まったく異なるユース ケースに役立ちます。
データの管理 の **汎用データベース**	EAV 構造化データ モデルについては、ここで詳しく説明します。

例 1 – Generic ロードプレフィックスを使用した EAV 構造化データの変換

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、Transactions という名前のテーブルに読み込まれるデータセットが含まれています。データセットには、日付項目が含まれます。既定の MonthNames 定義が使用されます。

ロード スクリプト

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `Color`。

このメジャーを追加します。

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

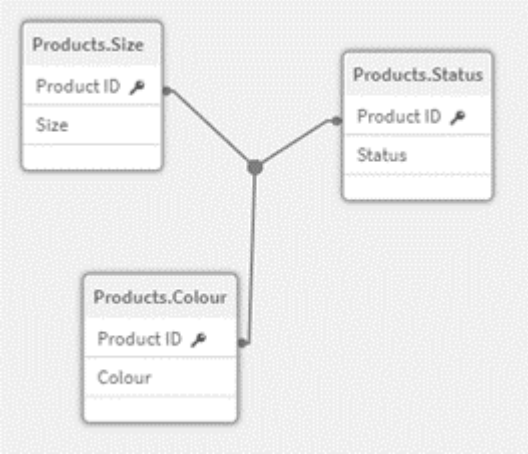
色別の製品数の検査ができるようになりました。

結果テーブル

色	=Count([Product ID])
茶色	します。
白色	2

データモデルの形状に注意してください。各属性は、元のターゲットテーブル タグ `Product`に従って名前が付けられた個別のテーブルに分割されています。各テーブルには、サフィックスとして属性があります。この一例は `Product.Color`です。結果であるProduct Attribute 出力レコードは、`Product ID`によって関連付けられます。

結果のデータモデル ビューア表現



レコードの結果テーブル:

Products.Status

製品 ID	ステータス
13	製造中止
2	製造中止

レコードの結果テーブル:

Products.Size

製品 ID	サイズ
13	13-15
20	16-18
45	16-18

レコードの結果テーブル:

Products.Color

製品 ID	色
13	茶色
5	茶色
44	茶色
45	茶色
20	白色
2	白色

例 2 – Generic ロードプレフィックスを使用しない EAV 構造化データの分析

ロード スクリプトとチャートの数式

概要

この例では、EAV 構造化データを元の形式で分析する方法を示します。

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、EAV 構造にある Products という名前のテーブルに読み込まれるデータセットが含まれています。

この例では、色属性ごとに製品をカウントしています。このように構造化されたデータを分析するには、属性値 Color を持つ製品の式レベルのフィルタリングを適用する必要があります。

さらに、個々の属性を軸や項目として選択することはできないため、効果的なビジュアライゼーションを構築する方法を決定するのが難しくなります。

ロード スクリプト

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: Value。

次のメジャーを作成します:

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

色別の製品数の検査ができるようになりました。

レコードの結果テーブル: Products.Status

値	=Count({<Attribute={'Color'}>} [Product ID])
茶色	します。
白色	2

例 3 – Generic ロードからの結果の出力テーブルの非正規化 (高度)

ロードスクリプトとチャートの数式

概要

この例では、Generic ロードプレフィックスによって生成された正規化されたデータ構造を非正規化して、統合された Product 軸テーブルに戻す方法を示します。これは、データモデルのパフォーマンス チューニングの一部として使用できる高度なモデリング手法です。

データ ロード エディタを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプト

```
Products:

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

RENAME TABLE Products.Color TO Products;

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

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

結果

データモデル ビューアを開き、結果のデータモデルの形状を確認します。非正規化されたテーブルが 1 つだけ存在します。Products.Size、Products.Status、および Products.Color の 3 つの中間出力テーブルの組み合わせです。

結果の内部データモデル

製品案内
製品 ID
ステータス
色
サイズ

レコードの結果テーブル: 製品

製品 ID	ステータス	色	サイズ
13	製造中止	茶色	13-15
20	-	白色	16-18
2	製造中止	白色	-
5	-	茶色	-
44	-	茶色	-
45	-	茶色	16-18

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: Color。

このメジャーを追加します:

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

結果テーブル

色	=Count([Product ID])
茶色	します。
白色	2

Hierarchy

hierarchy プレフィックスを使用して、親子階層テーブルを、Qlik Sense データモデルで有用なテーブルに変換します。これは、**LOAD** や **SELECT** ステートメントの前に置き、ロードステートメントの結果をテーブル変換の入力として使用します。

このプレフィックスを使用すると、展開ノードテーブルが作成されます。通常、レコード数は入力テーブルと同じですが、階層の各レベルがさらに別の項目に格納されます。パス項目は、ツリー構造で使用できます。

構文:

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

入力テーブルは、隣接するノードテーブルでなければなりません。通常、隣接するノードテーブルは、各レコードがノードと一致し、親ノードへの参照を含む項目が含まれます。このようなテーブルでは、ノードは 1 つのレコードにしか保存されませんが、子ノードをいくつでも持つことができます。当然のことながら、テーブルには、ノードの属性が記述された追加項目が含まれている可能性があります。

このプレフィックスを使用すると、展開ノードテーブルが作成されます。通常、レコード数は入力テーブルと同じですが、階層の各レベルがさらに別の項目に格納されます。パス項目は、ツリー構造で使用できます。

通常、入力テーブルには各ノードに 1 件のレコードが含まれており、出力テーブルにも同数のレコードが含まれます。しかし、場合によっては複数の親を持つノードがあり、1 つのノードが入力テーブル内にある複数のレコードで表されることがあります。その場合、出力テーブルのレコード数は入力テーブルを上回ります。

ノード ID 列にない親 ID を持つノード(親 ID がないノードを含む) はすべて、ルートとみなされます。また、直接/間接を問わず、ルートノードに接続されているノードのみをロードし、循環参照を回避します。

親ノードのノード名とノードのパス、ノードの階層レベルを含む追加項目を作成することもできます。

引数:

引数

引数	説明
NodeID	ノード ID を含む項目の名前。この項目は入力テーブルになければなりません。
ParentID	親ノードのノード ID を含む項目の名前。この項目は入力テーブルになければなりません。
NodeName	ノード名が含まれる項目の名前。この項目は入力テーブルになければなりません。
ParentName	新しい **ParentName** 項目に名前を付けるための文字列。省略すると、この項目は作成されません。
ParentSource	ノードパスの構築に使用するノード名が含まれた項目の名前。このパラメータはオプションです。省略すると、**NodeName** が使われます。
PathName	新しい **Path** 項目に名前を付けるための文字列で、ルートからノードへのパスが含まれます。このパラメータはオプションです。省略すると、この項目は作成されません。
PathDelimiter	新しい **Path** 項目の区切り記号として使用する文字列。このパラメータはオプションです。省略すると「/」が使われます。
Depth	新しい **Depth** 項目に名前を付けるための文字列で、ノードの階層レベルを含みます。このパラメータはオプションです。省略すると、この項目は作成されません。

```
Hierarchy(NodeID, ParentID, NodeName, ParentName, NodeName, PathName, '\', Depth) LOAD *
inline [

NodeID, ParentID, NodeName

1, 4, London

2, 3, Munich

3, 5, Germany

4, 5, UK

5, , Europe

];
```

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

HierarchyBelongsTo

このプレフィックスを使用して、親子階層テーブルを、Qlik Sense データモデルで有用なテーブルに変換します。これは、**LOAD** や **SELECT** ステートメントの前に置き、ロードステートメントの結果をテーブル変換の入力として使用します。

このプレフィックスを使用すると、階層における先祖ノードと子ノードの関係をすべて含むテーブルが作成されます。その結果、先祖項目を使用して階層のツリー全体を選択できるようになります。ほとんどの場合、出力テーブルには各ノードにつき複数のレコードが含まれています。

構文:

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

入力テーブルは、隣接するノードテーブルでなければなりません。通常、隣接するノードテーブルは、各レコードがノードと一致し、親ノードへの参照を含む項目が含まれます。このようなテーブルでは、ノードは 1 つのレコードにしか保存されませんが、子ノードをいくつでも持つことができます。当然のことながら、テーブルには、ノードの属性が記述された追加項目が含まれている可能性があります。

このプレフィックスを使用すると、階層における先祖ノードと子ノードの関係をすべて含むテーブルが作成されます。その結果、先祖項目を使用して階層のツリー全体を選択できるようになります。ほとんどの場合、出力テーブルには各ノードにつき複数のレコードが含まれています。

異なるレベルのノードを持つ追加項目を作成することも可能です。

引数:

引数

引数	説明
NodeID	ノードID を含む項目の名前。この項目は入力テーブルになければなりません。
ParentID	親ノードのノードID を含む項目の名前。この項目は入力テーブルになければなりません。
NodeName	ノード名が含まれる項目の名前。この項目は入力テーブルになければなりません。
AncestorID	新しい親 ID フィールドに名前を付けるための文字列で、親ノードの ID が含まれます。
AncestorName	新しい先祖項目に名前を付けるための文字列で、先祖ノードの名前が含まれます。
DepthDiff	新しい **DepthDiff** 項目に名前を付けるための文字列で、先祖ノードと関連している親ノードのレベルが含まれます。このパラメータはオプションです。省略すると、この項目は作成されません。

```
HierarchyBelongsTo (NodeID, AncestorID, NodeName, AncestorID, AncestorName, DepthDiff) LOAD *
inline [

NodeID, AncestorID, NodeName

1, 4, London

2, 3, Munich

3, 5, Germany

4, 5, UK

5, , Europe

];
```

Results

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

Inner

join および **keep** プレフィックスの前には、プレフィックス **inner** を置くことができます。**join** の前に使用すると、内部結合を指定できます。結果のテーブルには、生データテーブルからの項目値の組み合わせのみが含まれます。連結項目値は双方のテーブルに示されます。**keep** の前に使用すると、Qlik Sense に保存される前に、双方の生データテーブルが共通部分に縮小されます。

構文:

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

引数:

引数

引数	説明
tablename	名前が付いたテーブルが、ロード済みのテーブルと比較されます。
loadstatement または selectstatement	ロード済みテーブルの **LOAD** または **SELECT** ステートメントです。

例

ロードスクリプト

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Table1:
Load * inline [
Column1, Column2
```

```
A, B
1, aa
2, cc
3, ee ];

Table2:
Inner Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

結果

結果のテーブル

Column1	Column2	Column3
A	B	C
1	aa	xx

説明

この例は、最初の (左) テーブルと2 番目の (右) テーブルの両方に存在する値のみが結合される内部結合出力を示しています。

IntervalMatch

IntervalMatch プレフィックスを使うと、不連続数値を1 つ以上の数値間隔に一致させるテーブル、そしてオプションとして1 つ以上の追加キーの値を一致させるテーブルを作成できます。

構文:

```
IntervalMatch (matchfield)(loadstatement | selectstatement )
```

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

IntervalMatch プレフィックスは、間隔をロードする **LOAD** または **SELECT** ステートメントの前に配置する必要があります。不連続データポイントを含む項目 (以下の例では Time) および追加キーは、**IntervalMatch** プレフィックスを含むステートメントの前に Qlik Sense にロードされていなければなりません。このプレフィックスはデータベース テーブルからこの項目を読み取るのではなく、ロードされた間隔テーブルとキーを変換して追加列 (不連続数値データ点) を含むテーブルを生成します。また、新しいテーブルで不連続データポイントと間隔、キー項目の値の組み合わせごとにレコードが1 つ存在するようレコード数を増やします。

間隔は重なる場合があり、不連続値は一致する間隔すべてにリンクされます。

IntervalMatch プレフィックスがキー項目で展開される際、プレフィックスを使用して、不連続数値を1 つ以上の数値間隔と照合し、同時に1 つまたは複数の追加キーを照合するテーブルが作成されます。

未定義の間隔範囲が無視されないようにするには、間隔の下限または上限を構成する項目へのNULL値のマッピングを許可しなければならない可能性があります。その場合は、**NullAsValue** ステートメントを使用するか、不連続数値データポイントの前または後でNULL値を数値に置き換える明示的なテストを実施します。

引数:

引数

引数	説明
matchfield	間隔にリンクする不連続の数値が含まれた項目。
keyfield	変換で一致させる追加属性が含まれた項目。
loadstatement orselectstatement	最初の項目に各間隔の下限、2つ目の項目に各間隔の上限を含むテーブルが生成されます。キー一致を使用している場合、3つ目以降の項目には **IntervalMatch** ステートメントのキー項目が含まれます。間隔は常に閉じているので、終端は間隔に含まれます。数値以外の範囲では間隔が無視されます (未定義として対処)。

Example 1:

下記の2つのテーブルのうち、最初のテーブルは個別イベントの数を表示し、2番目のテーブルは注文の製造開始時間と終了時間を定義します。**IntervalMatch** プレフィックスを使用すると、2つのテーブルが論理的に接続され、イベントの影響を受けた注文を特定したり、どのシフトでどの注文が処理されたかを確認することができます。

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

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

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

テーブル **OrderLog** には、追加の列: *Time* が含まれるようになりました。レコードの数も展開されます。

Table with additional column

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

Example 2: (keyfield の使用)

上記の同じ例で、キー項目として *ProductionLine* を追加します。

```
EventLog:

LOAD * Inline [

Time, Event, Comment, ProductionLine

00:00, 0, Start of shift 1, P1

01:00, 0, Start of shift 1, P2

01:18, 1, Line stop, P1

02:23, 2, Line restart 50%, P1

04:15, 3, Line speed 100%, P1

08:00, 4, Start of shift 2, P1

09:00, 4, Start of shift 2, P2

11:43, 5, End of production, P1

11:43, 5, End of production, P2

];


OrderLog:

LOAD * INLINE [

Start, End, Order, ProductionLine
```

```
01:00, 03:35, A, P1

02:30, 07:58, B, P1

03:04, 10:27, C, P1

07:23, 11:43, D, P2

];


//Link the field Time to the time intervals defined by the fields Start and End and match the
values

// to the key ProductionLine.

Inner Join

IntervalMatch ( Time, ProductionLine )

LOAD Start, End, ProductionLine

Resident OrderLog;
```

テーブル ボックスが次のように作成できるようになりました。

Tablebox example

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

Join

join プレフィックスは、ロード済みのテーブルを名前が付いた既存テーブルまたは直前に作成されたデータテーブルと結合します。

データを結合すると、追加の項目または属性のセット (ターゲットテーブルにまだ存在しないもの) によってターゲットテーブルが拡張されます。ソースデータセットとターゲットテーブルの間の共通の項目名は、新しい入力レコードを関連付ける方法を決定するために使用されます。これは一般に「自然結合」と呼ばれます。Qlik 結合操作では、結合の関連付けの一意性と使用される結合の種類に応じて、結果のターゲットテーブルのレコードが開始時よりも増減する可能性があります。

結合には次の 4 つのタイプがあります。

Left join

左結合は、最も一般的な結合タイプです。たとえば、トランザクション データセットがあり、それを参照データセットと組み合わせたい場合、通常は `Left Join` を使用します。最初にトランザクション テーブルをロードし、次に `Left Join` プレフィックスを介して既にロードされているトランザクション テーブルに結合しながら、参照データセットをロードします。`Left Join` は、すべてのトランザクションをそのまま保持し、一致が見つかった補足参照データ項目を追加します。

Inner join

一致する関連付けがある結果のみを対象とする 2 つのデータセットがある場合は、`Inner Join`の使用を検討してください。これにより、一致するものが見つからない場合、ロードされたソースデータとターゲットテーブルの両方からすべてのレコードが削除されます。その結果、結合操作が行われる前よりもターゲットテーブルのレコードが減少する可能性があります。

Outer join

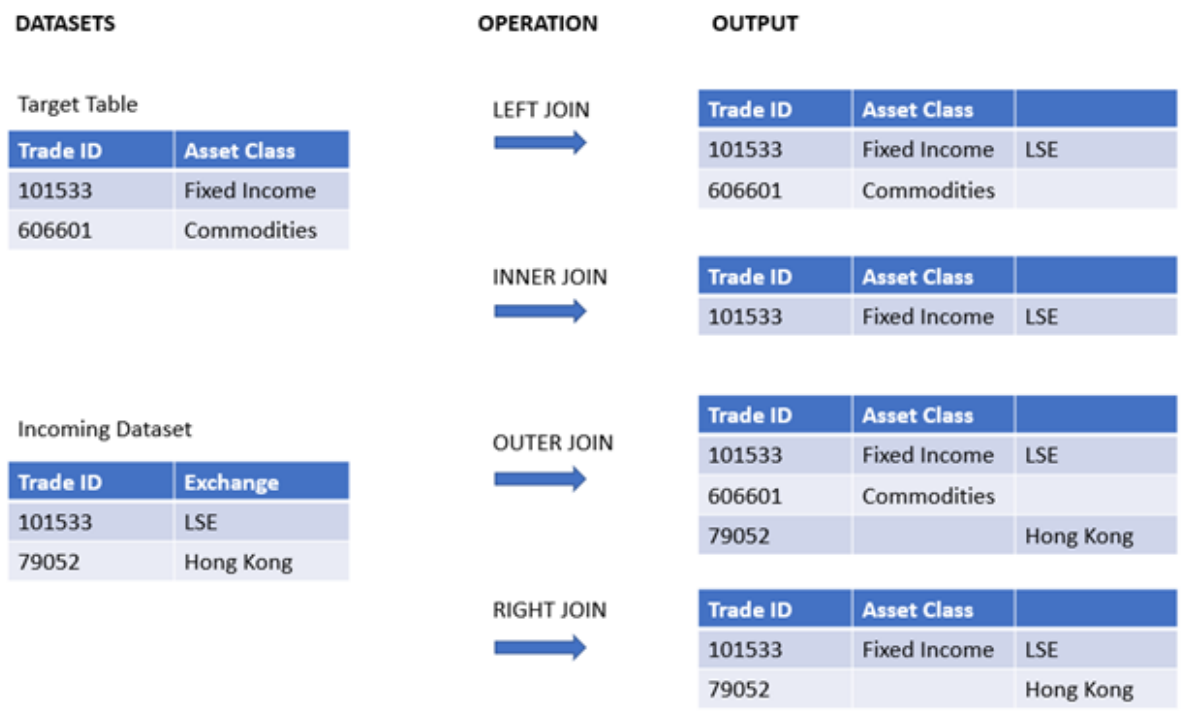
ターゲットレコードとすべての着信レコードの両方を保持する必要がある場合は、`Outer Join`を使用します。一致が見つからない場合、結合の反対側の項目は未入力 (null) のままですが、レコードの各セットは引き続き保持されます。

type キーワードを省略した場合、既定の結合タイプは外部結合となります。

Right join

この結合タイプは、ロードされるすべてのレコードを保持しながら、結合の対象となるテーブル内のレコードを、着信レコードに関連付けの一致があるレコードのみに減らします。これはニッチな結合タイプであり、事前にロードされたレコードのテーブルを必要なサブセットにトリミングする手段として使用されることがあります。

さまざまなタイプの結合操作からの結果セットの例



結合操作のソースとターゲットの間に共通の項目名がない場合、結合はすべての行のデカルト積になります。これは「クロス結合」と呼ばれます。

「クロス結合」操作による結果セットの例

構文:

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

引数

引数	説明
tablename	名前が付いたテーブルが、ロード済みのテーブルと比較されます。
loadstatementまたは selectstatement	ロード済みテーブルの **LOAD** または **SELECT** ステートメントです。

これらのトピックは、この関数を使用するのに役立つかもしれません。

関連トピック

トピック	説明
データの管理 で **Join** と**Keep** を使用したテーブルの結合	このトピックでは、データセットの「結合」と「保持」の概念について詳しく説明します。
Keep (page 81)	Keep ロードプレフィックスは Join プレフィックスに似ていますが、ソースデータセットとターゲットデータセットを結合しません。代わりに、採用された操作のタイプ (内側、外側、左、または右) に従って各データセットをトリムします。

例 1 - 左結合: 参照データセットを使用してターゲットテーブルを強化する

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Changesという名前のテーブルに読み込まれる、変更レコードを表すデータセット。これには、ステータスIDキー項目が含まれます。
- 変更ステータスを表す2番目のデータセット。ロードされ、左の Join ロードプレフィックスで結合することによって元の変更レコードと結合されます。

この左結合により、共通のステータスIDに基づいて着信ステータスレコードの一致が見つかったステータス属性を追加しながら、変更レコードを確実にそのまま残すことができます。

ロードスクリプト

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

```
10323  1       08/11/2022      26/11/2022      High
10326  2       11/11/2022      05/12/2022      None
10138  2       07/05/2022      03/08/2022      None
10031  3       20/01/2022      25/03/2022      Low
10040  1       29/01/2022      22/04/2022      None
10134  1       03/05/2022      08/07/2022      Low
10334  2       19/11/2022      06/02/2023      Low
10220  2       28/07/2022      06/09/2022      None
10264  1       10/09/2022      17/10/2022      Medium
10116  1       15/04/2022      24/04/2022      None
10187  2       25/06/2022      24/08/2022      Low
] (delimiter is '\t');

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

結果

データモデル ビューアを開き、データモデルの形状を確認します。非正規化されたテーブルが1つだけ存在します。これは、元のすべての変更レコードの組み合わせであり、一致するステータス属性が各変更レコードに結合されています。

結果の内部データモデル

変更
変更 ID
ステータスID
開始予定日付
終了予定日付
ビジネスへの影響
ステータス
サブ ステータス

データモデル ビューアでプレビュー ウィンドウを展開すると、この完全な結果セットの一部がテーブルに編成されて表示されます。

データ モデル ビューアでの変更 テーブルのプレビュー

変更 ID	ステータス ID	開始予定日付	終了予定日付	ビジネスへの影響	ステータス	サブ ステータス
10030	4	19/01/2022	23/02/2022	なし	終了	キャンセル済み
10031	3	20/01/2022	25/03/2022	低	終了	完了
10015	3	04/01/2022	15/02/2022	低	終了	完了
10103	1	02/04/2022	29/05/2022	中間	開く	開始されていない
10116	1	15/04/2022	24/04/2022	なし	開く	開始されていない
10134	1	03/05/2022	08/07/2022	低	開く	開始されていない
10264	1	10/09/2022	17/10/2022	中間	開く	開始されていない
10040	1	29/01/2022	22/04/2022	なし	開く	開始されていない
10323	1	08/11/2022	26/11/2022	高	開く	開始されていない
10187	2	25/06/2022	24/08/2022	低	開く	開始済み
10185	2	23/06/2022	08/09/2022	なし	開く	開始済み
10220	2	28/07/2022	06/09/2022	なし	開く	開始済み
10326	2	11/11/2022	05/12/2022	なし	開く	開始済み
10138	2	07/05/2022	03/08/2022	なし	開く	開始済み
10334	2	19/11/2022	06/02/2023	低	開く	開始済み

[ステータス] テーブルの 5 行目 (ステータス ID:「5」、ステータス:「Closed」、サブステータス:「Obsolete」) は [変更] テーブルのどのレコードにも対応していないため、この行の情報は上記の結果セットには表示されません。

データ ロード エディタに戻ります。データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `Status`。

このメジャーを追加します。

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

これで、ステータスごとに変更の数を調べることができます。

結果テーブル

ステータス	=Count([Change ID])
開く	12
終了	3

例 2 – 内部結合: 一致するレコードのみを組み合わせる

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Changes という名前のテーブルに読み込まれる、変更レコードを表すデータセット。
- ソースシステム JIRA から派生した変更レコードを表す 2 番目のデータセット。変更ステータスを表す 2 番目のデータセット。ロードされ、左の Inner Join ロードプレフィックスで結合することによって元の変更レコードと結合されます。

この Inner Join により、両方のデータセットで見つかった 5 つの変更レコードのみが確実に保持されます。

ロードスクリプト

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

JIRA_changes:
Inner Join (Changes)
Load
  [Ticket ID] AS [Change ID],
  [Source System]
inline
```

```
[
Ticket ID      Source System
10000  JIRA
10030  JIRA
10323  JIRA
10134  JIRA
10334  JIRA
10220  JIRA
20000  TFS
] (delimiter is '\t');
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- Source System
- Change ID
- Business Impact

これで、結果の 5 つのレコードを調べることができます。Inner Join からの結果テーブルには、両方のデータセットで一致する情報を持つレコードのみが含まれます。

結果テーブル

ソース システム	変更 ID	ビジネスへの影響
JIRA	10030	なし
JIRA	10134	低
JIRA	10220	なし
JIRA	10323	高
JIRA	10334	低

例 3 – 外部結合: 重複するレコード セットの結合

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Changes という名前のテーブルに読み込まれる、変更レコードを表すデータセット。
- ソース システム JIRA から派生した変更レコードを表す 2 番目のデータセット。変更ステータスを表す 2 番目のデータセット。これは、ロードされ、左の Outer Join ロードプレフィックスで結合することによって元の変更レコードと結合されます。

これにより、両方のデータセットから重複するすべての変更レコードが確実に保持されます。

ロードスクリプト

```
// 8 Change records

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

// 6 Change records

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- Source System
- Change ID
- Business Impact

これで、結果の 10 つのレコードを調べることができます。

結果テーブル

ソース システム	変更 ID	ビジネスへの影響
JIRA	10030	なし

ソース システム	変更 ID	ビジネスへの影響
JIRA	10134	低
JIRA	10220	なし
JIRA	10323	-
JIRA	10334	低
JIRA	10597	-
-	10015	低
-	10031	低
-	10040	なし
-	10138	なし

例 4 – 右結合: セカンダリマスター データセットによるターゲット テーブルのトリミング

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Changes という名前のテーブルに読み込まれる、変更レコードを表すデータセット。
- Teamwork のソース システムからの変更レコードを表す 2 番目のデータセット。これはロードされ、Right Join ロードプレフィックスで結合することによって元のレコードと結合されます。

これにより、Teamwork 変更レコードのみが保持され、ターゲット テーブルに一致する Change IDがなくても Teamwork レコードが失われることはありません。

ロード スクリプト

```
Changes:

Load * inline [
Change ID      Status ID       Scheduled Start Date    Scheduled End Date      Business Impact
10030  4       19/01/2022      23/02/2022      None
10015  3       04/01/2022      15/02/2022      Low
10103  1       02/04/2022      29/05/2022      Medium
10185  2       23/06/2022      08/09/2022      None
10323  1       08/11/2022      26/11/2022      High
10326  2       11/11/2022      05/12/2022      None
10138  2       07/05/2022      03/08/2022      None
10031  3       20/01/2022      25/03/2022      Low
10040  1       29/01/2022      22/04/2022      None
10134  1       03/05/2022      08/07/2022      Low
10334  2       19/11/2022      06/02/2023      Low
10220  2       28/07/2022      06/09/2022      None
```

```
10264  1       10/09/2022      17/10/2022      Medium
10116  1       15/04/2022      24/04/2022      None
10187  2       25/06/2022      24/08/2022      Low
] (delimiter is '\t');

Teamwork_changes:
Right Join (Changes)
Load
  [Ticket ID] AS [Change ID],
  [Source System]
inline
[
Ticket ID      Source System
10040  Teamwork
10015  Teamwork
10103  Teamwork
10031  Teamwork
50231  Teamwork
] (delimiter is '\t');
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- Source System
- Change ID
- Business Impact

これで、結果の 5 つのレコードを調べることができます。

結果テーブル

ソース システム	変更 ID	ビジネスへの影響
チームワーク	10015	低
チームワーク	10031	低
チームワーク	10040	なし
チームワーク	10103	中間
チームワーク	50231	-

Keep

keep プレフィックスは **join** プレフィックスに類似しています。**join** プレフィックスのように、ロード済みテーブルと既存の名前付きテーブルまたは直前に作成されたデータテーブルを比較します。しかし、ロード済みテーブルと既存のテーブルを結合する代わりに、テーブル データの共通部分に基づき、Qlik Sense に保存される前に、2 つのうち一方または両方のテーブルを縮小する効果があります。実施された比較は、すべての共通項目で行われる自然結合に相当します (対応する結合と同じ方法など)。ただし、2 つのテーブルは結合されず、別の名前付きテーブルとして Qlik Sense に保存されます。

構文:

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

keep プレフィックスの前には、**inner** または **left**、**right** のプレフィックスを配置する必要があります。

Qlik Sense スクリプト言語における明示的な **join** プレフィックスは、2 つのテーブルの完全な結合を実行します。その結果、1 つのテーブルが生成されます。通常、このような結合を行うと、かなり大きなテーブルが作成されます。そのため、Qlik Sense では、テーブルを結合する代わりに複数のテーブル間で関連付けを行います。これにより、メモリの使用量が削減され、処理速度がアップすると同時に柔軟性が極めて高まります。このような理由から、一般的には Qlik Sense スクリプトでの明示的な結合は避ける必要があります。keep の機能は、明示的な結合の使用回数を減らすよう設計されています。

引数:

引数

引数	説明
tablename	名前が付いたテーブルが、ロード済みのテーブルと比較されます。
loadstatement または selectstatement	ロード済みテーブルの **LOAD** または **SELECT** ステートメントです。

```
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

Join および **Keep** プレフィックスの前には、プレフィックス **left** を置くことができます。

join の前に使用すると、左結合を指定します。結果のテーブルには、生データテーブルからの項目値の組み合わせのみが含まれます。連結項目値は最初のテーブルに示されます。**keep** の前に使用すると、Qlik Sense に保存される前に、2 つ目の生データテーブルは 1 つ目のテーブルとの共通部分に縮小されます。

同じ名前で文字列関数を検索していた場合参照先: Left (page 1432)

構文:

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

引数:

引数

引数	説明
tablename	名前が付いたテーブルが、ロード済みのテーブルと比較されます。
loadstatementまたは selectstatement	ロード済みテーブルの **LOAD** または **SELECT** ステートメントです。

例

ロードスクリプト

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Table1:
Load * inline [
Column1, Column2
A, B
1, aa
2, cc
3, ee ];

Table2:
Left Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

結果

結果のテーブル

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

説明

この例は、最初の (左) テーブルに存在する値のみが結合される左結合出力を示しています。

マッピング

mapping プレフィックスは、マッピング テーブルの作成に使用します。マッピング テーブルは、スクリプト実行中に項目値と項目名を置き換えるといった操作で使用できます。

構文:

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

mapping プレフィックスは **LOAD** または **SELECT** ステートメントの前に入れることができ、ロード ステートメントの結果をマッピング テーブルとして保存します。マッピングは、US や U.S.、アメリカの USA への置換など、スクリプト実行中に項目の値を置き換える効果的な方法です。マッピング テーブルは 2 列構成で、1 列目には比較値、2 列目はマッピング値が含まれます。マッピング テーブルは一時的にメモリに保存され、スクリプト実行後に自動的に削除されます。

マッピング テーブルのコンテンツには、**Map ... Using** および **Rename Field** ステートメント、**Applymap()** および **Mapsubstring()** 関数を使用してアクセスできます。

この例では、Salesperson とその居住国の国コードのリストをロードします。国コードを国名に置き換えるために、国コードを国にマッピングしたテーブルを使用します。このマッピング テーブルでは、3 つの国のみが定義されており、他の国は 'Rest of the world' としてマッピングされています。

```
// 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';
```
この結果、テーブルは次のようになります。

Mapping table

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

マージ

Merge プレフィックスをスクリプト内の任意の **LOAD** または **SELECT** ステートメントに追加して、ロードされたテーブルを別のテーブルに統合する必要があることを指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。

一般的な使用例は、変更ログをロードし、これを使用して inserts、updates、deletes を既存のテーブルに適用する場合です。

> *部分的なリロードが正しく機能するためには、部分的なリロードがトリガーされる前に、アプリをデータで開く必要があります。*

[リロード] ボタンを使用して部分的なリロードを実行します。Qlik Engine JSON API を使用することもできます。

構文:

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

引数:

引数

引数	説明
only	ステートメントが部分的なリロード中にのみ実行される必要があることを示すオプションの修飾子。通常の (部分的ではない) リロード中はステートメントは無視されます。
SequenceNoField	操作の順序を定義する日付と時刻またはシーケンス番号を含む項目の名前。

引数	説明
SequenceNoVar	マージされるテーブルの SequenceNoField の最大値が割り当てられる変数の名前。
ListOfKeys	主キーを指定する項目名のコンマ区切りのリスト。
Operation	load ステートメントの最初の項目には、次の操作は含まれている必要があります: 'Insert'、'Update'あるいは'Delete'。'i'、'u'および 'd'も受け入れ可能です。

一般的機能

通常の (部分的ではない) リロード中、**[Merge] [LOAD]** 構造は通常の **[Load]** ステートメントとして機能しますが、古い廃止されたレコードと削除のマークが付けられたレコードを削除する追加機能があります。**Load** ステートメントの最初の項目は、操作に関する情報: (Insert、Update または Delete) を保持する必要があります。

ロードされたレコードごとに、レコード ID が以前にロードされたレコードと比較され、最新のレコード (シーケンス番号による) のみが保持されます。最新のレコードが Delete でマークされている場合、何も保持されません。

ターゲット テーブル

変更するテーブルは、一連のフィールドにより決まります。同じ一連のフィールド(最初のフィールドと操作を除く) を持つテーブルが既に存在する場合、これは変更すべき関連テーブルになります。または、**Concatenater**(連結) プレフィックスを使用してテーブルを指定することもできます。ターゲット テーブルが指定されていない場合、**Merge LOAD** 構築結果は新規テーブルに保存されます。

Concatenate (連結) プレフィックスが使用されている場合、結果としてできたテーブルには、既存テーブルの結合とマージへの入力に対応する一連フィールドが含まれます。したがって、ターゲット テーブルは、マージへの入力として使用される変更ログよりも多くのフィールドを持つ可能性があります。

部分的なリロードは、完全なリロードと同じように実行されます。1つの違いは、部分的なリロードは、新規テーブルを作成することがほとんどないことです。**Only**句を使用しない限り、前のスクリプト実行と同じ一連のフィールドを持つターゲット テーブルは常に存在します。

シーケンス番号

ロードされた変更ログが累積ログである場合、つまり、すでにロードされている変更が含まれている場合、パラメーター SequenceNoVar を **Where** 句で使用して、入力データの量を制限できます。次に、項目 SequenceNoField が SequenceNoVar より大きいレコードのみをロードするように **Merge LOAD** を作成できます。完了すると、**Merge LOAD** は、SequenceNoField 項目に表示される最大値を持つ新しい値を SequenceNoVar に割り当てます。

演算

Merge LOAD は、ターゲット テーブルよりも少ないフィールドを持つことができます。操作が異なれば、欠落しているフィールドの処理も異なります:

挿入: **Merge LOAD** で欠落しているが、ターゲット テーブルに存在するフィールドは、ターゲット テーブルで NULL を取得します。

削除: 欠落しているフィールドは結果に影響しません。いずれにせよ関連レコードは削除されます。

更新: **Merge LOAD** で一覧表示されているフィールドは、ターゲットテーブルで更新されます。欠落しているフィールドは、変更されません。これは、以下の2つステートメントが同一でないことを意味します:

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

最初のステートメントは、一覧表示されたレコードを更新し、F2 を NULL に変更します。2 番目のステートメントは F2 を変更しませんが、その代わり、ターゲットテーブルに値を残します。

例

例 1: 指定されたテーブルとの簡単なマージ

この例では、Persons という名前のインライン テーブルに 3 つの行がロードされています。次に、**Merge** はテーブルを次のように変更します。

- 行 (*Mary, 4*) を追加します。
- 行 (*Steven, 3*) を削除します。
- 番号 *5* を *Jake* に割り当てます。

[*LastChangeDate*] 変数は、**[マージ]** が実行された後、[*ChangeDate*] 列の最大値に設定されます。

ロードスクリプト

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Set DateFormat='D/M/YYYY';
Persons:
load * inline [
Name, Number
Jake, 3
Jill, 2
Steven, 3
];

Merge (ChangeDate, LastChangeDate) on Name Concatenate(Persons)
LOAD * inline [
Operation, ChangeDate,   Name,     Number
Insert,    1/1/2021,     Mary,     4
Delete,    1/1/2021,     Steven,
Update,    2/1/2021,     Jake,     5
];
```

結果

Merge Loadの前は、結果のテーブルは次のように表示されます。

Resulting table

Name	Number
Jake	3
Jill	2
Steven	3

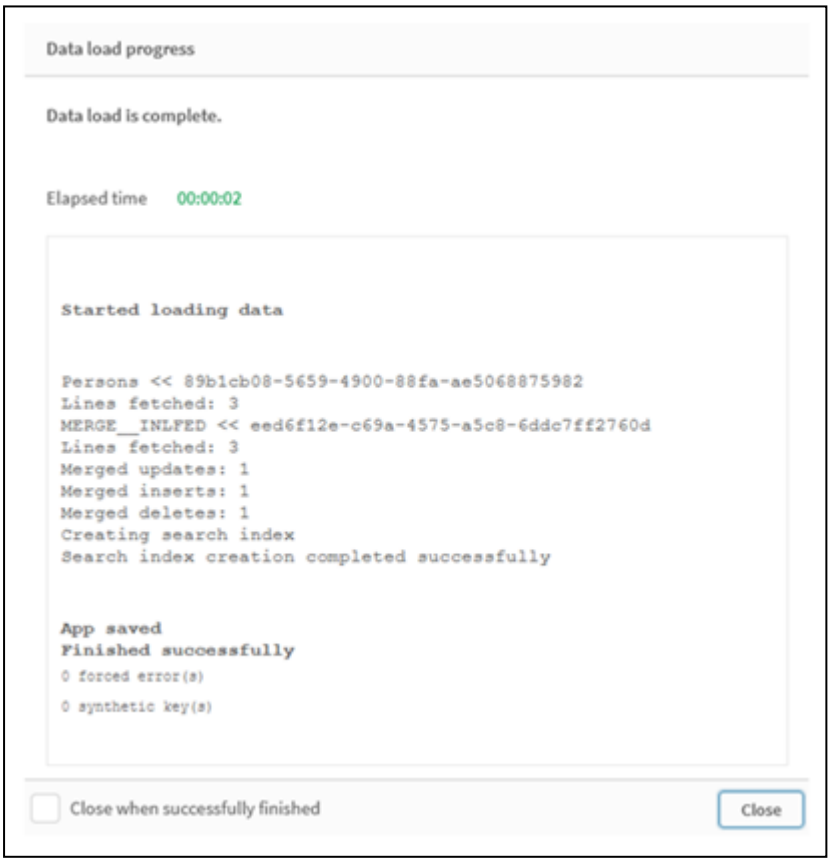
MergeLoadの後、テーブルは次のように表示されます:

Resulting table

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

データがロードされると、**[データのロード進行状況]** ダイアログ ボックスに実行された操作が表示されます。

[データのロード進行状況] ダイアログ ボックス



例 2: フィールドが欠落しているデータ ロード スクリプト

この例では、上記と同じデータがロードされますが、現在では各人のIDが含まれます。

Mergeはテーブルを以下のように変更します:

- 行 (*Mary, 4*) を追加します。
- 行 (*Steven, 3*) を削除します
- 番号 *5* を *Jake* に割り当てます
- 番号 *6* を *Jill* に割り当てます。

ロードスクリプト

ここでは、2つの**Merge Load**ステートメントを使用します。1つは「Insert」と「Delete」用で、もう1つは「Update」用です。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Set DateFormat='D/M/YYYY';
Persons:
Load * Inline [
PersonID, Name, Number
1, Jake, 3
2, Jill, 2
3, Steven, 3
];

Merge (ChangeDate, LastChangeDate) on PersonID Concatenate(Persons)
Load * Inline [
Operation, ChangeDate,   PersonID, Name,     Number
Insert,    1/1/2021,     4,        Mary,     4
Delete,    1/1/2021,     3,        Steven,
];

Merge (ChangeDate, LastChangeDate) on PersonID Concatenate(Persons)
Load * Inline [
Operation, ChangeDate,   PersonID, Number
Update,    2/1/2021,     1,        5
Update,    3/1/2021,     2,        6
];
```

結果

MergeLoadステートメントの後、テーブルは次のように表示されます:

Resulting table

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

2番目の**Merge**ステートメントにはフィールドの**Name**が含まれていないため、その結果、名前が変更されていないことに注意してください。

例 3: データロードスクリプト - ChangeDate を使用した Where 句による部分的なリロード

次の例では、**Only**引数は、**Merge**コマンドが部分的なリロード中にのみ実行されることを指定しています。更新は、以前にキャプチャされた [LastChangeDate] に基づいてフィルタリングされます。**Merge** が終了すると、LastChangeDate 変数には、マージ中に処理された ChangeDate 列の最大値が割り当てられます。

ロードスクリプト

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

NoConcatenate

NoConcatenate プレフィックスは、同一の項目セットでロードされた 2 つのテーブルを、強制的に別個の内部テーブルとして扱います (そうでない場合、自動的に連結されます)。

構文:

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

既定では、項目数が同じで一致する項目名を持つテーブルがスクリプトにロード済みのテーブルにロードされた場合、Qlik Sense がこれら 2 つのテーブルを自動連結します。これは、2 つめのテーブルの名前が異なっていても発生します。

ただし、スクリプトプレフィックス NoConcatenate が 2 番目のテーブルの Load ステートメントまたは select ステートメントの前に含まれていた場合、これら 2 つのテーブルは別にロードされます。

NoConcatenate の典型的なユースケースは、テーブルの臨時コピーを作成してそのコピーの臨時変換を実行しながらも、元のデータのコピーを保持する必要がある場合などです。NoConcatenate により、ソーステーブルに黙示的に追加せずにコピーを確実に作成できます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
Source: LOAD A,B from file1.csv; CopyOfSource: NoConcatenate LOAD A,B resident Source;	メジャーとしてAとBを持つテーブルがロードされます。同じ項目を持つ2番目のテーブルは、NoConcatenate変数を使って別のロードされます。

例 1 – 黙示的連結

ロードスクリプトと結果

概要

この例では、2つのスクリプトを順に追加します。

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactionsというテーブルに送信される日付と金額を含む初期データセット。

最初のロードスクリプト

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount

最初の結果テーブル

ID	日付	amount
1	08/30/2018	23.56

ID	日付	amount
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

2 番目のロード スクリプト

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 同一項目を持つ 2 番目のデータセットが Sales というテーブルに送信されます。

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

結果

データをロードしてテーブルに移動します。

2 番目の結果テーブル

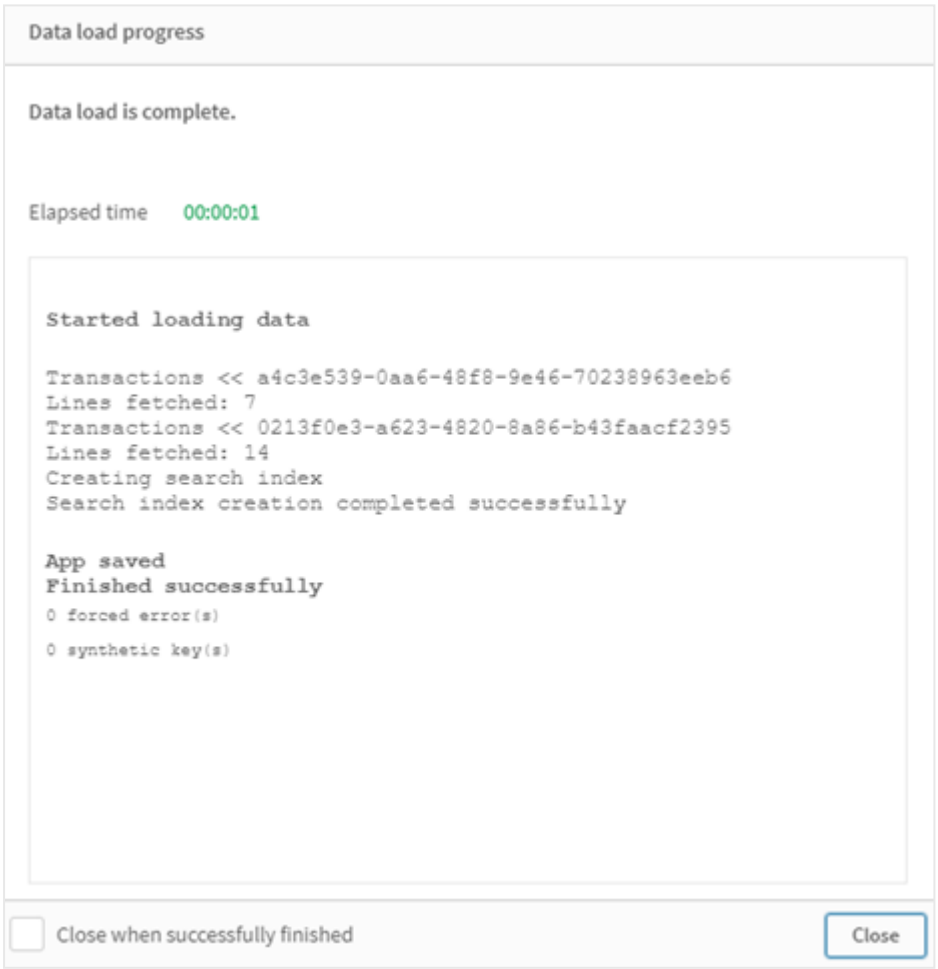
ID	日付	amount
1	08/30/2018	23.56
2	09/07/2018	556.31
3	09/16/2018	5.75
4	09/22/2018	125.00
5	09/22/2018	484.21
6	09/22/2018	59.18
7	09/23/2018	177.42

ID	日付	amount
8	10/01/2018	164.27
9	10/03/2018	384.00
10	10/06/2018	25.82
11	10/09/2018	312.00
12	10/15/2018	4.56
13	10/16/2018	90.24
14	10/18/2018	19.32

スクリプトを実行すると、Sales テーブルが既存の Transactions テーブルに黙示的に連結されます。これは、2つのデータセットの項目数と、項目名が同一だからです。これは、2 番目のテーブル名タグが結果セットを 'Sales' という名前にしようとしても発生します。

データロード進捗状況を確認することにより、Sales データセットが黙示的に連結されたいることがわかります。

トランザクション データが黙示的に連結されていることを示すデータロード進捗状況ログ。



例 2 – ケース シナリオを使用する

ロード スクリプトと結果

概要

このユース ケース シナリオには次の要素があります:

- 次のようなトランザクション データセット:
  - ID
  - 日付
  - 金額 (GBP)
- 通貨テーブル:
  - USD からGBP の為替レート
- 次のような 2 番目のトランザクション データセット:
  - ID
  - 日付
  - 金額 (USD)

5 つのスクリプトを順にロードします。

- 最初のロード スクリプトには、Transactions というテーブルに送信される日付と金額 (GBP) を含む初期データセットが含まれています。
- 2 番目のロード スクリプトには次の内容が含まれます:
  - Transactions_in_USD というテーブルに送信される日付と金額を含む 2 番目のデータセット。
  - 黙示的連結を回避するために、Transactions_in_USD データセットの Load ステートメントの前に配置された noconcatenate プレフィックス。
- 3 番目のロード スクリプトには、Transactions_in_USD テーブルで GBP とUSD の為替レートを作成するのに使用される join プレフィックスが含まれます。
- 4 番目のロード スクリプトには、Transactions_in_USD を初期の Transactions テーブルに追加する concatenate プレフィックスが含まれます。
- 5 番目のロード スクリプトには、データが Transactions テーブルに連結された Transactions_in_USD テーブルを削除する drop table ステートメントが含まれます。

最初のロード スクリプト

```
Transactions:

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

```
7, 12/23/2018, 177.42
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount

最初のロードスクリプト結果

ID	日付	amount
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42

テーブルには、金額 (GBP) を持つ初期データセットが表示されています。

2 番目のロードスクリプト

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

結果

データをロードしてテーブルに移動します。

2 番目のロードスクリプト結果

ID	日付	amount
1	12/30/2018	23.56

ID	日付	amount
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42
8	01/01/2019	164.27
9	01/03/2019	384.00
10	01/06/2019	25.82
11	01/09/2019	312.00
12	01/15/2019	4.56
13	01/16/2019	90.24
14	01/18/2019	19.32

Transactions_in_USD テーブルからの 2 番目のデータセットが追加されたことがわかります。

3 番目のロード スクリプト

このロード スクリプトは、USD から GBP の為替レートを Transactions_in_USD テーブルに追加します。

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

結果

データをロードして、データ モデル ビューアに移動します。Transactions_in_USD テーブルを選択すると、各既存レコードの「レート」項目値が 0.7 になっていることがわかります。

4 番目のロード スクリプト

resident load を使うと、このロード スクリプトは金額を USD に換算後、Transactions_in_USD テーブルを Transactions テーブルに連結します。

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

結果

データをロードしてテーブルに移動します。8 ~ 14 行目に GBP 建ての金額で新しい入力が表示されます。

4 番目のロードスクリプト結果

ID	日付	amount
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42
8	01/01/2019	114.989
8	01/01/2019	164.27
9	01/03/2019	268.80
9	01/03/2019	384.00
10	01/06/2019	18.074
10	01/06/2019	25.82
11	01/09/2019	218.40
11	01/09/2019	312.00
12	01/15/2019	3.192
12	01/15/2019	4.56
13	01/16/2019	63.168
13	01/16/2019	90.24
14	01/18/2019	13.524
14	01/18/2019	19.32

5 番目のロードスクリプト

このロードスクリプトは、4 番目のロードスクリプト結果テーブルから重複エントリを削除し、GBP 建ての金額を持つエントリのみを残します。

```
drop tables Transactions_in_USD;
```

結果

データをロードしてテーブルに移動します。

5 番目のロードスクリプト結果

ID	日付	amount
1	12/30/2018	23.56
2	12/07/2018	556.31
3	12/16/2018	5.75
4	12/22/2018	125.00
5	12/22/2018	484.21
6	12/22/2018	59.18
7	12/23/2018	177.42
8	01/01/2019	114.989
9	01/03/2019	268.80
10	01/06/2019	18.074
11	01/09/2019	218.40
12	01/15/2019	3.192
13	01/16/2019	63.168
14	01/18/2019	13.524

5 番目のロードスクリプトをロードした後、結果テーブルには両方のトランザクションデータセットに存在した 14 件のトランザクションすべてが表示されます。ただし、トランザクション 8 ~ 14 では金額が GBP に変換されています。

2 番目のロードスクリプトで `Transactions_in_USD` の前に使用された `NoConcatenate` プレフィックスを削除した場合、スクリプトに「テーブル「`Transactions_in_USD`」が見つかりません」というエラーが発生します。これは、`Transactions_in_USD` テーブルが初期 `Transactions` テーブルに連結するはずだったからです。

Only

Only スクリプトキーワードは、集計関数、またはパーシャル リロード接頭辞 **Add**、**Replace** および **Merge** の構文の一部として使用します。

Outer

外部結合を指定するために、明示的な **Join** プレフィックスの前にプレフィックス **Outer** を付加できます。外部結合では、2 つのテーブル間のすべての組み合わせが生成されます。結果のテーブルには、生データテーブルからの項目値の組み合わせが含まれます。連結項目値は一方または双方のテーブルに示されます。**Outer**キーワードはオプションで、結合プレフィックスが指定されていない場合のデフォルトの結合タイプです。

構文:

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

引数:

引数

引数	説明
tablename	名前が付いたテーブルが、ロード済みのテーブルと比較されます。
loadstatementまたは selectstatement	ロード済みテーブルの **LOAD** または **SELECT** ステートメントです。

例

ロードスクリプト

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Table1:
Load * inline [
Column1, Column2
A, B
1, aa
2, cc
3, ee ];

Table2:
Outer Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

結果のテーブル

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

説明

この例では、2 つのテーブル Table1 とTable2 が、Table1 というラベルの付いた単一のテーブルにマージされています。このような場合、**外部**プレフィックスは、複数のテーブルを1 つのテーブルに結合して、単一のテーブルの値に対して集計を実行するためによく使用されます。

部分的なリロード

フル リロードは、常に既存のデータ モデルのすべてのテーブルを削除することから始まり、次にロード スクリプトを実行します。

部分的なリロードはこれを行いません。代わりに、データ モデル内のすべてのテーブルを保持し、[**Add**]、[**Merge**]、または [**Replace**] プレフィックスが前に付いた [**Load**] ステートメントと[**Select**] ステートメントのみを実行します。他のデータ テーブルはコマンドの影響を受けません。引数 **only** は、ステートメントが部分的なリロード中にのみ実行され、フル リロード中には無視されることを示します。次の表は、部分的および完全なリロードのステートメントの実行の概要です。

ステートメント	完全なリロード	部分的なリロード
Load ...	ステートメントが実行されます	ステートメントが実行されません
Add/Replace/Merge Load ...	ステートメントが実行されます	ステートメントが実行されます
Add/Replace/Merge Only Load ...	ステートメントが実行されません	ステートメントが実行されます

部分的なリロードには、完全なリロードと比較していくつかの利点があります。

- 最近変更されたデータのみをロードする必要があるため、より高速です。データセットが大きい場合、違いは大きくなります。
- ロードされるデータが少ないため、消費されるメモリも少なくなります。
- ソース データへのクエリがより高速に実行され、ネットワークの問題のリスクが軽減されるため、信頼性が向上します。

部分的なリロードが正しく機能するためには、部分的なリロードがトリガーされる前に、アプリをデータで開く必要があります。

[**リロード**] ボタンを使用して部分的なリロードを実行します。Qlik Engine JSON API を使用することもできます。

制限事項

部分的なリロード中ではなく、完全なリロード中に存在したテーブルに関するコマンドがある場合、部分的なリロードは失敗します。

例

コマンドの例

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

この場合、<Table_removed_after_full_reload> は部分的なリロードではなく、完全なリロードに存在したテーブルです。

回避方法

回避方法として、次の IF ステートメントのあるコマンドを囲むことができます:

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

部分的なリロードにより、データから値を削除できます。ただし、これは、内部で維持されるテーブルである個別の値のリストには反映されません。したがって、部分的なリロード後、リストには、最後の完全なリロード以降に項目に存在したすべての個別の値が含まれます。これは、部分的なリロード後に現在存在する値よりも多い場合があります。これは、FieldValueCount() および FieldValue() 関数の出力に影響します。FieldValueCount() は、現在の項目値の数よりも大きい数を返す可能性があります。

例

例 1

ロード スクリプト

例のスクリプトをアプリに追加し、部分的なリロードを実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

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

結果

Resulting table

Num	Count(Num)
11	1
12	1
13	1
14	1
15	1
16	1

Num	Count(Num)
17	1
18	1
19	1
20	1

説明

ステートメントは、部分的なリロード中にのみ実行されます。「distinct」プレフィックスが省略されている場合、**[Num]**フィールドの計測は、後続の部分的なリロードごとに増加します。

例 2

ロード スクリプト

例のスクリプトをアプリに追加し。フル リロードを実行し、結果を表示します。次に、部分的なリロードを実行し、結果を表示します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
T1:
Load recno() as ID, recno() as Value autogenerate 10;

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

結果

Output table after full reload

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

Output table after partial reload

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

説明

最初のテーブルは完全なリロード中にロードされ、2番目のテーブルは部分的なリロード中に最初のテーブルを置き換えるだけです。

Replace

Replace スクリプト キーワードは、文字列関数、またはパーシャル リロードの接頭辞として使用します。

Replace

Replace プレフィックスをスクリプト内の任意の **LOAD** または **SELECT** ステートメントに追加して、ロードされたテーブルを別のテーブルに置き換えるように指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。**Replace** プレフィックスは **Map** ステートメントでも使用できます。

> *部分的なリロードが正しく機能するためには、部分的なリロードがトリガーされる前に、アプリをデータで開く必要があります。*

[**リロード**] ボタンを使用して部分的なリロードを実行します。Qlik Engine JSON API を使用することもできます。

構文:

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

```
Replace [only] mapstatement
```

通常の (部分的ではない) リロード中、**Replace LOAD** 構造は通常の **LOAD** ステートメントとして機能しますが、前に **Drop Table** が付きます。最初に古いテーブルが削除され、次にレコードが生成されて新しいテーブルとして保存されます。

[**Concatenate**] プレフィックスが使用されている場合、または同じ項目のセットを持つテーブルが存在する場合、これはドロップする関連テーブルになります。それ以外の場合、ドロップするテーブルはなく、**Replace LOAD** の構造は通常の **LOAD** と同じになります。

部分的なリロードでも同じことができます。唯一の違いは、ドロップする前のスクリプト実行からのテーブルが常に存在することです。**Replace LOAD** 構造は常に最初に古いテーブルを削除し、次に新しいテーブルを作成します。

Replace Map...Using ステートメントでは、パーシャル スクリプトの実行中もマッピングが発生します。

引数:

引数

引数	説明
only	ステートメントが部分的なリロード中にのみ実行される必要があることを示すオプションの修飾子。通常の (部分的ではない) リロード中は無視する必要があります。

例と結果:

例	結果
`Tab1: Replace LOAD * from File1.csv;`	通常のリロードおよびパーシャル リロードでは、まず Qlik Sense テーブル Tab1 が削除されます。次に File1.csv から新しいデータがロードされ、Tab1 に保存されます。
`Tab1: Replace only LOAD * from File1.csv;`	通常のリロード中、このステートメントは無視されます。 パーシャル リロードでは、以前に指定した Tab1 テーブルの Qlik Sense が最初に削除されます。次に File1.csv から新しいデータがロードされ、Tab1 に保存されます。
`Tab1: LOAD a,b,c from File1.csv; Replace LOAD a,b,c from File2.csv;`	通常のリロードでは、最初にファイル File1.csv が Qlik Sense テーブル Tab1 に読み取られますが、このファイルは直ちに削除され、File2.csv からロードされた新しいデータに置き換えられます。その結果、File1.csv のデータはすべて失われます。 パーシャル リロードでは、まず Qlik Sense テーブル Tab1 が削除されます。次に File2.csv からロードされた新しいデータに置き換えられます。
`Tab1: LOAD a,b,c from File1.csv; Replace only LOAD a,b,c from File2.csv;`	通常のリロードでは、データは File1.csv からロードされ、Qlik Sense テーブル Tab1 に保存されます。その際、File2.csv は無視されます。 パーシャル リロードでは、まず Qlik Sense テーブル Tab1 が削除されます。次に File2.csv からロードされた新しいデータに置き換えられます。その結果、File1.csv のデータはすべて失われます。

Right

Join および **Keep** プレフィックスの前には、プレフィックス **right** を置くことができます。

join の前に使用すると、右結合を指定します。結果のテーブルには、生データテーブルからの項目値の組み合わせのみが含まれます。連結項目値は 2 番目のテーブルに示されます。**keep** の前に使用すると、Qlik Sense に保存される前に、1 つ目の生データテーブルは 2 つ目のテーブルとの共通部分に縮小されます。

同じ名前で文字列関数を検索していた場合参照先: Right (page 1440)

構文:

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

引数:

引数

引数	説明
tablename	名前が付いたテーブルが、ロード済みのテーブルと比較されます。
loadstatement または selectstatement	ロード済みテーブルの **LOAD** または **SELECT** ステートメントです。

例

ロードスクリプト

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Table1:
Load * inline [
Column1, Column2
A, B
1, aa
2, cc
3, ee ];

Table2:
Right Join Load * inline [
Column1, Column3
A, C
1, xx
4, yy ];
```

結果

結果のテーブル

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

説明

この例は、2 番目の (右) テーブルに存在する値のみが結合される右結合出力を示しています。

Sample

sample または **LOAD** ステートメントの **SELECT** プレフィックスは、データ ソースからランダムにレコード サンプルをロードする際に使用します。

構文:

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

評価される式は、データセットからQlik Sense アプリケーションに読み込まれるレコードのパーセントではなく、読み込まれた各レコードがアプリケーションに読み込まれる確率を定義します。つまり、値 `p = 0.5` を指定するということは、レコードの合計数の 50% がロードされるということではなく、各レコードについて Qlik Sense アプリケーションにロードされる確率が 50% あるということです。

引数

引数	説明
p	0 以上 1 以下の数を評価する任意の数式。数は、指定したレコードが読み取られる確率を示します。 レコードはすべて読み取られますが、Qlik Sense にロードされるのは一部のみです。

使用に適しているケース

サンプル データを大きいテーブルから取得する場合、データ、分布、または項目内容の性質を理解するのにサンプルが有用です。データのサブセットを持ち込むため、データのロードが短時間で済み、スクリプトのテストを迅速に実施できます。`First` とは異なり、`Sample` 関数は最初の 2 ~ 3 行に限定されず、テーブル全体からデータを持ち込みます。これにより、ケースによってはデータの表示がより正確になります。

次の例は、`Sample` スクリプトプレフィックスの考えられる使用方法です。

```
Sample 0.15 SQL SELECT * from Longtable;

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

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データ ロード スクリプトの `SET DateFormat` ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロード スクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしているQlik Sense サーバーがスウェーデンに設定されてい場合、データ ロード エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – インライン テーブルからのサンプル

ロード スクリプトと結果

概要

この例では、スクリプトが、7 件のレコードを含むデータセットからのサンプルのデータセットを、インライン テーブルからの Transactions というテーブルへロードします。

ロード スクリプト

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- amount

次のメジャーを追加します。

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

結果テーブル

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

この例で使用したロードの反復では、7 件のレコードはすべて読み込まれましたが、データ テーブルにロードされたのレコードは 4 件のみでした。ロードを再実行すると、異なる数値、および異なるレコードのセットがアプリケーションにロードされることになります。

例 2 – 自動生成されたテーブルからのサンプル

ロードスクリプトと結果

概要

この例では、Autogenerate を使用することで、100 件のレコードのデータセットが項目 date、id、および amount で作成されます。ただし、Sample プレフィックスは 0.1 の値と併用されます。

ロードスクリプト

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

Autogenerate(100);
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- amount

次のメジャーを追加します:

結果テーブル

ID	日付	=Sum(amount)
48	9/28/2013	763
20	5/15/2013	752
19	11/8/2013	657
25	3/24/2013	522
27	8/23/2013	389
81	6/1/2013	53
100	8/15/2013	17

この例で使用したロードの反復では、7 件のレコードが作成されたデータセットからロードされました。改めて、ロードを再実行すると、異なる数値、および異なるレコードのセットがアプリケーションにロードされることになります。

Semantic

semantic ロードプレフィックスは、ツリー構造、自己参照型の親子構造化データなどのリレーショナル データおよび/またはグラフとして記述できるデータを接続および管理するために Qlik Sense で使用できる特別なタイプの項目を作成します。

semantic ロードは、*Hierarchy (page 63)* および *HierarchyBelongsTo (page 65)* プレフィックスと同様に機能することに注意してください。3 つのプレフィックスはすべて、リレーショナル データをトラバースするための効果的なフロントエンド ソリューションのビルディング ブロックとして使用できます。

構文:

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

次の表に示すように、セマンティック ロードでは、3 つまたは 4 つの項目ちょうど入力が必要であり、順序付けられた各項目が何を示すかが厳密に定義されています。

セマンティック ロード項目

項目名	項目の説明
第 1 項目:	このタグは、関係がある 2 つのオブジェクトの最初のものを表します。
2 番目の項目:	このタグは、最初のオブジェクトと 2 番目のオブジェクトの間の「前方」関係を記述するために使用されます。最初のオブジェクトが子で、2 番目のオブジェクトが親である場合、子から親への関係をたどるように、「親」または「~ の親」を示す関係タブを作成できます。
3 番目の項目:	このタグは、関係がある 2 つのオブジェクトの 2 番目のものを表します。
4 番目の項目:	この項目はオプションです。このタグは、最初のオブジェクトと 2 番目のオブジェクトの間の「後方」または「逆」関係を記述するために使用されます。最初のオブジェクトが子で、2 番目のオブジェクトが親である場合、親から子への関係をたどるように、「子」または「~ の子」を示す関係タブを作成できます。4 番目の項目を追加しない場合、2 番目の項目 タグは、いずれかの方向の関係を記述するために使用されます。その場合、タグの一部として矢印記号が自動的に追加されます。

次のコードは、semantic プレフィックスの例です。

```
Semantic
Load
Object,
‘Parent’ AS Relationship,
NeighbouringObject AS Object,
‘Child’ AS Relationship
from graphdata.csv;
```

3 番目の項目に 1 番目の項目と同じラベルを付けることは許可されており、一般的な方法です。これにより、自己参照ルックアップが作成されるため、一度に 1 段階離れた関連オブジェクトまでたどることができます。3 番目の項目に同名が付いていない場合、最終的な結果は、オブジェクトから 1 ステップ離れた直接のリレーショナル ネイバーへの単純なルックアップになりますが、この出力はほとんど実用的ではありません。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データ ロード スクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロード スクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データ ロード エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関連する関数

関数	相互作用
Hierarchy (page 63)	Hierarchy ロードプレフィックスは、ノードを親子やその他のグラフ状のデータ構造に分割して整理し、それらをテーブルに変換するために使用されます。
HierarchyBelongsTo (page 65)	Hierarchy ロードプレフィックスは、親子やその他のグラフ状のデータ構造の先祖を見つけて整理し、それらをテーブルに変換するために使用されます。

例 - セマンティックプレフィックスを使用して関係を接続するための特別な項目を作成する

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- GeographyTree という名前のテーブルに読み込まれる、地理的関係レコードを表すデータセット。
  - 各エントリでは、行頭に ID があり、行末に ParentID があります。
- Relation というラベルの付いた 1 つの特別な動作項目を追加する semantic プレフィックス。

ロード スクリプト

```
GeographyTree:
LOAD
```

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

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します。

- Id
- Geography

次に、Relation を軸としてフィルター パネルを作成します。[**編集の完了**] をクリックします。

結果テーブル

Id	地理
1	世界
2	ヨーロッパ
3	アジア
4	北米
5	南アフリカ
6	イギリス
7	Germany

Id	地理
8	Sweden
9	韓国
10	北朝鮮
11	中国
12	ロンドン
13	バーミンガム

フィルター パネル

関係
子
親

テーブルの Geography 軸から [**ヨーロッパ**] をクリックし、フィルター パネルの Relation 軸から [**子**] をクリックします。テーブルの予想結果に注意してください。

ヨーロッパの「子」を示す
結果表

Id	地理
6	イギリス
7	Germany
8	Sweden

[**子**] をもう一度クリックすると、もう1 ステップ下にある、英国の「子」である場所が表示されます。

イギリスの「子」を示す
結果テーブル

Id	地理
12	ロンドン
13	バーミンガム

Unless

unless プレフィックスとサフィックスは、条件節の作成に使用します。条件節は、ステートメントまたは exit 節を評価するかどうかを決定します。これは、**if..end if** ステートメントの簡単な代替として使用されることがあります。

構文:

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

statement または **exitstatement** は、**condition** が False と評価された場合に限り、実行されます。

unless プレフィックスは、他の **when** や **unless** プレフィックスなどの 1 つまたは複数のステートメントを含むステートメントで使用できます。

引数

引数	説明
condition	True または False の評価を実施する論理式。
statement	制御ステートメント以外の任意の Qlik Sense スクリプト ステートメント。
exitstatement	**exit for**、**exit do**、**exit sub** 節、あるいは **exit script** ステートメント。

使用に適しているケース

`Unless` ステートメントはブール値の結果を返します。一般的に、このタイプの関数は、ユーザーがスクリプトの一部を条件付きでロードまたは除外する場合の条件として使用されます。

次の行では、`Unless` 関数の 3 つの使用例が示されています:

```
exit script unless A=1;

unless A=1 LOAD * from myfile.csv;

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

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データ ロード スクリプトの `SET DateFormat` ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロード スクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データ ロード エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – Unless プレフィックス

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 変数 A の作成。値 1 が提供されます。
- 変数 A = 2 でない限り、「Transactions」という名前のテーブルにロードされるデータセット。

ロード スクリプト

```
LET A = 1;

UNLESS A = 2

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount

結果テーブル

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

変数 A がスクリプトの最初で値 1 を割り当てているため、Unless プレフィックスの後の条件が評価され、FALSE の結果を返します。その結果、スクリプトが Load ステートメントを実行し続けます。結果テーブルでは、Transactions テーブルからのすべてのレコードを確認できます。

この変数値が 2 に設定された場合、データがデータモデルにロードされます。

例 2 – Unless サフィックス

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトは、初期データセットを Transactions というテーブルにロードすることから開始します。スクリプトは次に、Transactions テーブルのレコードが 10 件未満でない限り終了します。

この条件でスクリプトが終了しない場合、その後のトランザクションのセットは Transactions テーブルに連結され、このプロセスが反復されます。

ロード スクリプト

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


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

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

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

Concatenate
LOAD
*
Inline [
```

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount

結果テーブル

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

ロードスクリプトの 3 つのデータセットにはそれぞれ 7 件のレコードがあります。

最初のデータセット (トランザクション id 1 ~ 7) は、アプリケーションにロードされます。Unless 条件は、Transactions テーブルの行が 10 未満かどうかを評価します。これにより TRUE と評価され、2 番目のデータセット (トランザクション id 8 ~ 14) がアプリケーションにロードされます。2 番目の Unless 条件は、Transactions テーブルのレコードが 10 件未満かどうかを評価します。これにより、FALSE と評価され、スクリプトは終了します。

例 3 – 複数の Unless プレフィックス

ロードスクリプトと結果

概要

データロードエディタを開き、以下のロードスクリプトを新しいタブに追加します。

この例では、1 件のトランザクションを含むデータセットが、Transactions というテーブルとして作成されます。次に「for」ループがトリガーされ、2 つのネストされた unless ステートメントが次を評価します:

1. Transactions テーブルに含まれるレコードが 100 件を超えない限り
2. Transactions テーブルに含まれるレコード件数が 6 の倍数でない限り

これらの条件が FALSE の場合、さらに 7 件のレコードが生成され、既存の Transactions テーブルに連結されます。このプロセスは、2 つのトランザクションのうちどちらかが値 TRUE を返すまで繰り返されます。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:id。

結果テーブル

ID
0
1
2
3
4
5
30 を超える追加行

「for」ループで発生するネストされたunlessステートメントは次を評価します:

1. Transactionsテーブルには100を超える行がありますか?
2. Transactionsテーブルに含まれるレコード件数は6の倍数ですか?

両方のunlessステートメントがFALSEの値を返すと、さらに7件のレコードが生成され、既存のTransactionsテーブルに連結されます。

これらのステートメントは値FALSEを5回返し、その時点ではTransactionsテーブルに合計36行が含まれることになります。

この後、2番目のUnlessステートメントが値TRUEを返すため、この後のLoadステートメントは実行されなくなります。

When

whenプレフィックスとサフィックスは、条件節の作成に使用します。条件節は、ステートメントまたはexit節を実行するかどうかを決定します。これは、**if..end if**ステートメントの簡単な代替として使用されることがあります。

構文:

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

戻り値データ型: ブール値

Qlik Senseでは、真のブール値は-1で表現され、偽の値は0で表現されます。

statementまたは**exitstatement**は、conditionがTrueと評価された場合に限り、実行されます。

Whenプレフィックスは、他のWhenやUnlessプレフィックスなどの1つまたは複数のステートメントを含むステートメントで使用できます。

使用に適しているケース

Whenステートメントはブール値の結果を返します。一般的に、このタイプの関数は、ユーザーがスクリプトの一部をロードまたは除外する場合の条件として使用されます。

引数

引数	説明
condition	TRUEまたはFALSEの評価を実施する論理式
statement	制御ステートメント以外の任意のQlik Senseスクリプトステートメント。
exitstatement	**exit for**、**exit do**、**exit sub**節、あるいは**exit script**ステートメント。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトのSET DateFormatステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更

できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
`exit script when A=1;`	ステートメント `A=1` が TRUE と評価された場合、スクリプトは停止します。
`when A=1 LOAD * from myfile.csv;`	ステートメント `A=1` が TRUE と評価された場合、`myfile.csv` スクリプトは停止します。
`when A=1 unless B=2 drop table Tab1;`	ステートメント `A=1` が TRUE と評価され、`B=2` が FALSE と評価された場合、`Tab1` テーブルが停止します。

例 1 – When プレフィックス

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「`Transactions`」というテーブルに送信される日付と金額を含むデータセット。
- `A` が作成され、`1` の値を持つことを述べた `Let` ステートメント。
- `A` が `1` と等しい場合、スクリプトがロードし続けるという条件をテーブルする `When` 条件。

ロードスクリプト

```
LET A = 1;

WHEN A = 1

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

```
7, 09/23/2018, 177.42
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount

結果テーブル

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

変数 A がスクリプトの最初で値 1 を割り当てているため、when プレフィックスの後の条件が評価され、TRUE の結果を返します。返される結果が TRUE であるため、スクリプトが Load ステートメントを実行し続けます。結果テーブルからのすべてのレコードを確認できます。

この変数値が 1 と等しくない任意の値に設定された場合、データがデータモデルにロードされます。

例 2 – When サフィックス

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Transactions」というテーブルに送信される日付と金額を含む 3 つのデータセット。
  - 最初のデータセットには、トランザクション 1 ~ 7 が含まれます。
  - 2 番目のデータセットには、トランザクション 8 ~ 14 が含まれます。
  - 3 番目のデータセットには、トランザクション 15 ~ 21 が含まれます。
- 「Transactions」テーブルに含まれる行が 10 を超えるかどうかを決定する when 条件。when ステートメントのいずれかが TRUE と評価された場合、ロードスクリプトは停止します。この条件は、3 つの各データセットの終わりに配置されます。

ロード スクリプト

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

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

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

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

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- amount

結果テーブル

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

3 つのデータセットのそれぞれに 7 件のトランザクションがあります。最初のデータセットにはトランザクション 1 ~ 7 が含まれ、アプリケーションにロードされます。この Load ステートメントの後の when 条件は、FALSE と評価されます。これは、「Transactions」テーブルに含まれている行が 10 未満だからです。ロードスクリプトは次のデータセットに続きます。

2 番目のデータセットにはトランザクション 8 ~ 14 が含まれ、アプリケーションにロードされます。2 番目の when 条件は TRUE と評価されます。これは、「Transactions」テーブルに 10 を超える行があるためです。そのため、スクリプトは終了します。

例 3 – 複数の When プレフィックス

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 1 件のトランザクションを含むデータセットが、「Transactions」というテーブルとして作成されます。
- トリガーされる For ループには、2 つのネストされた When 条件が含まれますが、これは次の是非を評価します:
    1. 「Transactions」テーブルに含まれるレコードが 100 件を下回っている。
    2. 「Transactions」テーブルに含まれるレコード件数が 6 の倍数でない。

ロード スクリプト

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- id

結果テーブルに表示されているのは最初の 5 つのトランザクション ID のみですが、ロード スクリプトは 36 行作成するため、When 条件が満たされたら終了します。

結果テーブル

ID
0
1
2
3
4
5
30 を超える追加行

For ループのネストされた When 条件は、次の質問を評価します:

- 「Transactions」テーブルの行は 100 を下回っていますか?
- 「Transactions」テーブルに含まれるレコード件数は 6 の倍数ですか?

両方の when 条件が TRUE の値を返すと、さらに 7 件のレコードが生成され、既存の「Transactions」テーブルに連結されます。

when 条件は、TRUE の値を 5 回返します。その時点では、「Transactions」テーブルに合計 36 行のデータが存在します。

「Transactions」テーブルで 36 行のデータが作成されると、2 番目の when ステートメントが値 FALSE を返すため、この後の Load ステートメントは実行されなくなります。

3.3 スクリプト正規ステートメント

一般に正規ステートメントは、何らかの形でデータの操作に使用されます。これらのステートメントはスクリプト内で何行でも記述できますが、必ずセミコロン「;」で終了する必要があります。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。ただし、ステートメントで使用される項目名と変数名は大文字と小文字が区別されます。

スクリプト正規ステートメントの概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Alias

alias ステートメントは、エイリアスの設定に使用します。後続のスクリプトで項目が出現すると、エイリアスに従って項目の名前がその都度変更されます。

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

Autonumber

このステートメントは、スクリプトの実行中に発生する項目の値の個々の評価値について、一意の整数値を作成します。

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

Binary

binary ステートメントは、別の QlikView アプリや ドキュメントからデータ (セクション アクセス データなど) をロードする際に使用します。

```
Binary  [path] filename
```

comment

データベースやスプレッドシートの項目のコメント (メタデータ) を表示する方法を提供します。アプリに存在しない項目名は無視されます。項目名が何度も発生する場合は、最後の値が使用されます。

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

comment table

データベースやスプレッドシートのテーブルのコメント (メタデータ) を表示する方法を提供します。

```
Comment table tablelist using mapname
```

```
Comment table tablename with comment
```

Connect

この機能は Qlik Sense SaaS では使用できません。

CONNECT ステートメントは、Qlik Sense が OLE DB/ODBC インターフェースから一般的なデータベースにアクセスする方法を定義する際に使用します。ODBC の場合、まず ODBC アドミニストレータを使用して、データソースを指定する必要があります。

```
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 ステートメントは、項目定義を作成するために使用されます。このステートメントでは、項目や関数間の関係も定義できます。一連の項目の定義は、取得項目の自動生成に使用できます。取得項目は、軸として使用することもできます。例えば、カレンダー定義を作成して、日付項目から年、月、週、日などの関連する軸を生成するために使用できます。

```
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 ステートメントは、**Declare** ステートメントで作成された項目定義に基づいて、取得項目を生成するために使用されます。使用するデータ項目を指定して項目を取得することもできますし、項目タグに基づき明示的に、あるいは黙示的に取得することもできます。

```
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 ステートメントは、Direct Discovery 関数を使用している ODBC または OLE DB 接続からテーブルへのアクセスを可能にします。

```
Direct Query [path]
```

Directory

Directoryステートメントは、新たな**Directory**ステートメントが作成されるまで、後続の**LOAD**ステートメントのどのディレクトリでデータファイルを検索するか定義します。

```
Directory [path]
```

Disconnect

Disconnect ステートメントは、現在の ODBC/OLE DB/カスタム接続を終了します。このステートメントはオプションです。

```
Disconnect
```

drop field

drop field ステートメントを使用すると、スクリプトの実行中にいつでもデータモデルやメモリから1つ以上の Qlik Sense 項目を削除できます。テーブルの「個別」のプロパティは、**drop field** ステートメントの後に削除されます。

> ***drop field*** *と* ***drop fields*** *では同じ結果が得られるため、どちらを使用しても構いません。テーブルが指定されていない場合は、その項目が存在するすべてのテーブルから項目が削除されます。*

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

drop table

drop table ステートメントを使用すると、スクリプトの実行中にいつでもデータモデルやメモリから1つ以上の Qlik Sense 内部テーブルを削除できます。

> *形式は* ***drop table*** *と* ***drop tables*** *のどちらでも構いません。*

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

Execute

Execute ステートメントはその他のプログラムの実行に使用しますが、Qlik Sense ではデータのロードを行います。例えば、必要な変換を行う場合などです。

```
Execute commandline
```

FlushLog

FlushLogステートメントによって、Qlik Senseは強制的にスクリプトバッファの内容をスクリプトログファイルに書き込みます。

```
FlushLog
```

Force

force ステートメントにより、Qlik Sense は後続の **LOAD** および **SELECT** ステートメントの項目値を大文字のみ、小文字のみ、常に先頭を大文字化、またはそのまま (混合) として強制的に解釈します。このステートメントを使用すると、異なる表記規則に従って作成されたテーブルの項目値を関連付けられます。

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

LOAD

LOAD ステートメントは、ファイル、スクリプトで定義されたデータ、事前にロードされたテーブル、Web ページ、後続の **SELECT** ステートメントの結果、または自動生成されたデータから項目をロードします。分析接続からデータをロードすることもできます。

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

Let

let ステートメントは、**set** ステートメントを補完し、スクリプト変数を定義する際に使用します。**let** ステートメントでは、**set** ステートメントとは逆に、変数に代入する前に、スクリプトの実行時に「=」の右側の数式が評価されます。

```
Let variablename=expression
```

Loosen Table

スクリプト実行中に **Loosen Table** ステートメントを使用すると、1 つまたは複数の Qlik Sense 内部データテーブルに対して明示的に疎結合を宣言できます。テーブルが疎結合している場合、項目値間のすべての関連付けは削除されます。疎結合したテーブルの各項目を独立した未結合のテーブルとしてロードしても、同じ効果が得られます。疎結合は、データ構造の異なる部分を一時的に隔離するテストの間に有用です。疎結合したテーブルは、点線によりテーブル ビューアで識別できます。スクリプトで **Loosen Table** ステートメントを 1 度以上使用すると、Qlik Sense はスクリプト実行前に疎結合化されたテーブルの設定を無視します。

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

Map ... using

map ... using ステートメントは、特定のマッピング テーブルの値に、特定の項目値または数式をマップするために使用されます。マッピング テーブルは、**Mapping** ステートメントで作成します。

```
Map  *fieldlist Using  mapname
```

NullAsNull

NullAsNull ステートメントは、以前 **NullAsValue** ステートメントで設定された文字列値への NULL 値の変換を無効にします。

```
NullAsNull *fieldlist
```

NullAsValue

NullAsValue ステートメントは、NULL を値に変換する項目を指定します。

```
NullAsValue *fieldlist
```

Qualify

Qualify ステートメントは、項目名の修飾を切り替える際に使用します (項目名がプレフィックスとしてテーブル名を取得するなど)。

```
Qualify *fieldlist
```

Rem

rem ステートメントは、スクリプト内に備考やコメントを挿入するため、またスクリプトを削除することなく一時的に無効にするために使用します。

```
Rem string
```

Rename Field

このスクリプト関数は、既存の 1 つ以上の Qlik Sense 項目をロードした後、名前を変更します。

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

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

Rename Table

このスクリプト関数は、既存の 1 つ以上の Qlik Sense 内部テーブルをロードした後、名前を変更します。

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

Section

section ステートメントでは、後続の **LOAD** および **SELECT** ステートメントをデータとして、またはアクセス権の定義としてみなすかどうかを定義できます。

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

Select

ODBC データソースまたは OLE DB プロバイダの項目選択は、標準的な SQL **SELECT** ステートメントを介して実行されます。ただし、**SELECT** ステートメントが許可されるかどうかは、使用する ODBC ドライバまたは OLE DB プロバイダによって異なります。

```
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 ステートメントは、スクリプト変数を定義する際に使用します。これらは、文字列、パス、ドライバなどの代入に使用されます。

```
Set variablename=string
```

Sleep

sleep ステートメントは、指定した時間におけるスクリプトの実行を停止します。

```
Sleep n
```

SQL

SQL ステートメントを使用すると、ODBC または OLE DB 接続から任意の SQL コマンドを送信できます。

```
SQL sql_command
```

SQLColumns

sqlcolumns ステートメントは、**connect** が実行される ODBC または OLE DB データソースの列を記述する項目セットを返します。

```
SQLColumns
```

SQLTables

sqltables ステートメントは、**connect** が実行されている ODBC または OLE DB データソースのテーブルを説明する項目をセットで返します。

```
SQLTables
```

SQLTypes

sqltypes ステートメントは、**connect** が実行される ODBC または OLE DB データソースの種類を記述する項目セットを返します。

```
SQLTypes
```

Star

star ステートメントを使用すると、データベースの項目すべての値セットを表す文字列を設定できます。これは、後続の **LOAD** および **SELECT** ステートメントに影響を与えます。

```
Star is [ string ]
```

Store

Store ステートメントは、QVD、Parquet、CSV、または TXT ファイルを作成します。

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

Tag

このスクリプト ステートメントは、1 つ以上の項目またはテーブルにタグを割り当てる方法を提供します。アプリにない項目またはテーブルにタグを付けしようとしても、無視されます。項目名やタグ名の競合が発生する場合は、最後の値が使用されます。

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

Trace

trace ステートメントを使用すると、[**ロード スクリプトの進捗**] ウィンドウとスクリプトのログ ファイルに使用した文字列が書き込まれます。これはデバッグの際に非常に有用です。**trace** ステートメントの前に計算される変数の $ 拡張を使用すると、メッセージをカスタマイズできます。

```
Trace string
```

Unmap

Unmap ステートメントは、前に **Map ... Using** ステートメントによって指定した項目値の、後続のロードされた項目のマッピングを無効にします。

```
Unmap *fieldlist
```

Unqualify

Unqualify ステートメントは、**Qualify** ステートメントで事前に有効化された項目名の修飾を無効にする際に使用します。

```
Unqualify *fieldlist
```

Untag

このスクリプト ステートメントは、項目またはテーブルからタグを削除する方法を提供します。アプリにない項目またはテーブルのタグを外そうとしても、無視されます。

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

Alias

alias ステートメントは、エイリアスの設定に使用します。後続のスクリプトで項目が出現すると、エイリアスに従って項目の名前がその都度変更されます。

構文:

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

引数:

引数

引数	説明
fieldname	ソース データに含まれる項目の名前
aliasname	代わりに使用するエイリアス名

例と結果:

例	結果
`Alias ID_N as NameID;`	
`Alias A as Name, B as Number, C as Date;`	このステートメントで定義された名前の変更は、その後の **SELECT** ステートメントと**LOAD** ステートメントで使用されます。項目名として定義された新しいエイリアスは、その後のスクリプトの任意の場所で、新しい **alias** ステートメントによって定義されます。

AutoNumber

このステートメントは、スクリプトの実行中に発生する項目の値の個々の評価値について、一意の整数値を作成します。

LOAD ステートメント内に *autonumber (page 572)* 関数を使用することもできますが、最適化されたロードを使用する必要がある場合は、いくつか制限があります。最適化されたロードを作成するには、**QVD** ファイルのデータをロードしてから、**AutoNumber** ステートメントを使用して値をシンボル キーに変換します。

構文:

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

引数:

引数

引数	説明
*fieldlist	値を一意の整数値に置き換える必要がある項目のコンマ区切りのリスト。 名前が一致するすべての項目が含まれるように、項目名にワイルドカード文字 ? および * を使用できます。* を使用すると、すべての項目を含めることができます。ワイルドカードを使用する場合、項目名を引用符で囲む必要があります。
namespace	**Using** namespace はオプションです。別々の項目内の同じ値で同じキーを使用する名前空間を作成する場合に、このオプションを使用できます。 このオプションを使用しない場合、すべての項目でキー インデックスが別個になります。

制限事項:

スクリプトに複数の **LOAD** ステートメントを使用する場合、最後の **LOAD** ステートメントの後ろに **AutoNumber** ステートメントを配置する必要があります。

例 - AutoNumber を使用したスクリプト

スクリプトの例

この例では、データは最初に **AutoNumber** ステートメントなしでロードされます。次に、**AutoNumber** ステートメントが追加されて効果が示されます。

例で使用されているデータ

以下のデータをインライン データとしてデータロードエディタにロードして、以下のスクリプトの例を作成します。**AutoNumber** ステートメントはコメント アウトしたままにしておきます。

```
RegionSales:
LOAD *,
Region &'|'& Year &'|'& Month as KeyToOtherTable
INLINE
[ Region, Year, Month, Sales
North, 2014,   May,    245
North, 2014,   May,    347
North, 2014,   June,   127
South, 2014,   June,   645
South, 2013,   May,    367
South, 2013,   May,    221
];

Budget:
LOAD Budget,
Region &'|'& Year &'|'& Month as KeyToOtherTable
INLINE
[Region, Year, Month, Budget
North,  2014,   May,    200
North,  2014,   May,    350
North,  2014,   June,   150
South,  2014,   June,   500
South,  2013,   May,    300
South,  2013,   May,    200
];

//AutoNumber KeyToOtherTable;
```

ビジュアライゼーションの作成

Qlik Sense シートに 2 つのテーブル ビジュアライゼーションを作成します。**KeyToOtherTable**、**Region**、**Year**、**Month**、**Sales** を軸として 1 番目のテーブルに追加します。**KeyToOtherTable**、**Region**、**Year**、**Month**、**Budget** を軸として 2 番目のテーブルに追加します。

結果

RegionSales テーブル

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

Budget テーブル

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

説明

この例は、2 つのテーブルをリンクする複合項目 **KeyToOtherTable** を示しています。**AutoNumber** は使用しません。**KeyToOtherTable** 値の長さに注意してください。

AutoNumber ステートメントを追加する

ロード スクリプトの **AutoNumber** ステートメントのコメントを解除します。

```
AutoNumber KeyToOtherTable;
```

結果

RegionSales テーブル

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

KeyToOtherTable	Region	Year	Month	Sales
4	South	2013	May	367
4	South	2014	June	645

Budget テーブル

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

説明

KeyToOtherTable 項目値は一意の整数値に置き換えられ、その結果、項目値の長さが短縮され、メモリが節約されました。両方のテーブルのキー項目は **AutoNumber** の影響を受け、テーブルはリンクされたままになります。この例は、デモのために、簡略化したものになっていますが、この方法が効果的なのは、多数の行が含まれるテーブルで使用した場合です。

Binary

binary ステートメントは、別の Qlik Sense アプリや QlikView ドキュメントからデータ (セクション アクセス データなど) をロードする際に使用します。シート、ストーリー、ビジュアライゼーション、マスター アイテム、変数といった、アプリのその他の要素は含まれません。

このスクリプトで許可される **binary** ステートメントは 1 つのみです。**binary** は、通常、スクリプトの先頭に置かれる SET ステートメントの前であっても、必ず最初のステートメントでなければなりません。

構文:

```
binary [path] filename
```

引数:

引数

引数	説明
path	フォルダデータ接続への参照となるファイルへのパス。ファイルがQlik Sense作業ディレクトリに存在しない場合に必要になります。 ***'lib://Table Files/'*** レガシースクリプトモードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • このスクリプト行を含むアプリの相対パス。 ***data***
filename	ファイル拡張子 .qvw または .qvf を含むファイル名。

制限事項:

binary を使用し、アプリIDを参照して同じQlik Sense Enterprise展開でアプリからデータをロードすることはできません。*.qvf* ファイルからのみロードできます。

例

文字列	説明
`Binary lib://DataFolder/customer.qvw;`	この例では、ファイルは [**フォルダー**] データ接続になければなりません。これは、例えば管理者がQlik Senseサーバー上に作成したフォルダです。データロードエディターの [**接続の新規作成**] をクリックし、[**ファイルの場所**] にある [**フォルダー**] を選択します。
`Binary customer.qvf;`	この例では、ファイルはQlik Sense作業ディレクトリになければなりません。
`Binary c:\qv\customer.qvw;`	絶対ファイルパスを使用するこの例は、レガシースクリプトモードで動作します。

Comment field

データベースやスプレッドシートの項目のコメント(メタデータ)を表示する方法を提供します。アプリに存在しない項目名は無視されます。項目名が何度も発生する場合は、最後の値が使用されます。

構文:

```
comment [fields] *fieldlist using mapname
```

```
comment [field] fieldname with comment
```

使用するマッピング テーブルは 2 列で構成され、1 列目に項目名、2 列目にはコメントが含まれます。

引数:

引数

引数	説明
**fieldlist*	コメントする項目のコンマ区切りリスト。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。
mapname	マッピング **LOAD** またはマッピング **SELECT** ステートメントで、以前読み取られたマッピング テーブルの名前。
fieldname	コメントする項目の名前。
comment	項目に追加するコメント。

Example 1:

```
commentmap:

mapping LOAD * inline [

a,b

Alpha,This field contains text values

Num,This field contains numeric values

];

comment fields using commentmap;
```

Example 2:

```
comment field Alpha with AFieldContainingCharacters;

comment field Num with '*A field containing numbers';

comment Gamma with 'Mickey Mouse field';
```

Comment table

データベースやスプレッドシートのテーブルのコメント (メタデータ) を表示する方法を提供します。

アプリに存在しないテーブル名は無視されます。テーブル名が何度も発生する場合は、最後の値が使用されます。データソースからコメントを読み取るには、キーワードを使用します。

構文:

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

引数:

引数

引数	説明
tablelist	(table{,table})
mapname	マッピング **LOAD** またはマッピング **SELECT** ステートメントで、以前読み取られたマッピングテーブルの名前。
tablename	コメントするテーブルの名前。
comment	テーブルに追加するコメント。

Example 1:

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

Example 2:

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

Connect

CONNECT ステートメントは、Qlik Sense が OLE DB/ODBC インターフェースから一般的なデータベースにアクセスする方法を定義する際に使用します。ODBC の場合、まず ODBC アドミニストレータを使用して、データソースを指定する必要があります。

この機能は Qlik Sense SaaS では使用できません。

このステートメントは、標準モードのフォルダデータ接続のみに対応しています。

構文:

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

```
LIB CONNECT TO connection
```

引数:

引数

引数	説明
connect-string	connect-string ::= datasourcename { ; conn-spec-item } 接続文字列は、データソースの名前と1つまたは複数の接続指定アイテムのリスト(オプション) で構成されます。データソース名に空白が含まれる場合や接続指定アイテムがリストされている場合、接続文字列を引用符で囲む必要があります。 **datasourcename** は定義されたODBC データソース、あるいはOLE DB プロバイダを定義する文字列でなくてはなりません。 conn-spec-item ::=**DBQ=**database_specifier |**DriverID=**driver_specifier |**UID=**userid |**PWD=**password 使用可能な接続指定アイテムは、データベースにより異なります。上記以外のアイテムを使用できるデータベースもあります。OLE DB では、一部の接続特有の項目は必須に指定されており、オプションではありません。
connection	データロードエディタに保管されるデータ接続名。

ODBC が **CONNECT** の前に配置されている場合、ODBC インターフェースが使用されます。それ以外の場合は OLE DB が使用されます。

LIB CONNECT TO を使用すると、データロードエディタで作成したデータ接続でデータベースに接続されます。

Example 1:

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

このステートメントで定義されたデータソースは、新しい **CONNECT** ステートメントが作成されるまで、その後の **Select (SQL)** ステートメントに使用されます。

Example 2:

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

Connect32

このステートメントは **CONNECT** ステートメントと同じ方法で使用されますが、64 ビットシステムで強制的に 32 ビットODBC/OLE DB プロバイダを使用します。カスタム接続には適用されません。

Connect64

このステートメントは **CONNECT** ステートメントと同じ方法で使用されますが、64 ビットプロバイダを強制的に使用します。カスタム接続には適用されません。

Declare

Declare ステートメントは、項目定義を作成するために使用されます。このステートメントでは、項目や関数間の関係も定義できます。一連の項目の定義は、取得項目の自動生成に使用できます。取得項目は、軸として使用することもできます。例えば、カレンダー定義を作成して、日付項目から年、月、週、日などの関連する軸を生成するために使用できます。

Declare は、新しい項目定義の設定にも、既存の定義に基づく項目定義の作成にも使用できます。

新しい項目定義の設定

構文:

```
definition_name:

Declare [Field[s]] Definition [Tagged tag_list ]

[Parameters parameter_list ]

Fields field_list
```

引数:

引数	説明
definition_name	項目定義の名前 (末尾にコロン)。 *項目定義の名前として**autoCalendar**は使用しないでください。この名前は、自動生成されるカレンダーテンプレートとして予約済みです。* `Calendar:`
tag_list	項目定義をもとに作成された項目に適用するタグのコンマで区切りのリスト。タグの適用はオプションですが、$date、$numeric、$text などのソート順を指定するのに使用するタグを適用しない場合、生成される項目はデフォルトのロード順序でソートされます。 `'$date'Thank you for bringing this to our attention, and apologies for the inconvenience.`

引数	説明
parameter_list	コンマ区切りのパラメータリスト。パラメータは name=value 形式で定義され、初期値が割り当てられます。初期値は、項目定義が再利用時に、上書きできます。オプション。 first_month_of_year = 1
field_list	項目定義が使用される際に生成される項目のカンマ区切りのリスト。項目は <expression> **As** field_name **tagged** tag の形式で定義されます。自動取得項目の生成元のデータ項目を参照するには $1 を使用します。 Year($1) As Year tagged ('$numeric')

```
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')
;
```

カレンダーが定義され、ロードされたデータ項目にカレンダーを適用できるようになりました。この場合、**Derive** 節を使用して OrderDate と ShippingDate にカレンダーを適用できます。

既存の項目定義の再使用

構文:

```
<definition name>:

Declare [Field][s] Definition

Using <existing_definition>

[With <parameter_assignment> ]
```

引数:

引数	説明
definition_name	項目定義の名前 (末尾にコロン)。 `MyCalendar:`
existing_definition	新しい項目定義の作成時に再利用する項目定義。作成した新しい項目定義は、項目式で使用されている値を変更するために parameter_assignment を使用する場合を除いて、元になった定義と同様に機能します。 `Using Calendar`
parameter_assignment	コンマ区切りのパラメータ割り当てリスト。パラメータの割り当ては `name=value` の形式で定義され、元になる項目定義に設定されているパラメータ値を上書きします。オプション。 `first_month_of_year = 4`

この例では、上の例で作成したカレンダー定義を再利用します。4 月から始まる会計年度を使用します。first_month_of_year パラメータに 4 を割り当てます。このパラメータは、定義されている DayNumberOfYear 項目に影響を与えます。

この例は、上の例のサンプル データと項目定義の使用を前提としています。

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



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

このデータ スクリプトをリロードすると、生成された項目を OrderDate.MyCalendar.* と ShippingDate.MyCalendar.* という名前でシート エディタで利用できるようになります。

Derive

Derive ステートメントは、**Declare** ステートメントで作成された項目定義に基づいて、取得項目を生成するために使用されます。使用するデータ項目を指定して項目を取得することもできますし、項目タグに基づき明示的に、あるいは黙示的に取得することもできます。

構文:

```
Derive [fields]] From [Field[s]] field_list Using definition
```

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

```
Derive [Field[s]] From Implicit [Tag[s]] Using definition
```

引数:

引数

引数	説明
definition	項目の取得時に使用する項目定義の名前。 **Calendar**
field_list	項目定義に基づいて生成される取得項目の生成元になるデータ項目のコンマ区切りのリスト。データ項目は、スクリプトでロード済みの項目にする必要があります。 **OrderDate, ShippingDate**
tag_list	コンマ区切りのタグ リスト。取得項目は、このリストのいずれかのタグを持つすべてのデータ項目に対して生成されます。タグのリストは丸括弧で囲む必要があります。 **('$date','$timestamp')**

- 特定のデータ項目から項目を取得する。
  この場合、OrderDate 項目とShippingDate 項目を指定します。
  `DERIVE FIELDS FROM FIELDS OrderDate,ShippingDate USING Calendar;`
- 特定のタグを持つすべて項目から項目を取得する。
  Calendar に基づいて、$date タグを持つすべての項目から項目を取得します。
  `DERIVE FIELDS FROM EXPLICIT TAGS ('$date') USING Calendar;`
- 項目定義タグを持つすべての項目から項目を取得します。
  この場合、Calendar 項目定義と同じタグ (この場合は、$date) を持つすべてのデータ項目から項目を取得します。
  `DERIVE FIELDS FROM IMPLICIT TAG USING Calendar;`

Direct Query

DIRECT QUERY ステートメントは、Direct Discovery 関数を使用しているODBC または OLE DB 接続からテーブルへのアクセスを可能にします。

構文:

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

DIMENSION、**MEASURE**、**DETAIL** の各キーワードは、どのような順番でも使用できます。

DIMENSION と**FROM** キーワード節は、すべての **DIRECT QUERY** ステートメントに必要です。**FROM** キーワードは、**DIMENSION** キーワードの後に配置する必要があります。

DIMENSIONキーワードの後ろに直接指定した項目は、メモリにロードされ、インメモリとDirect Discoveryデータ間の関連付けの設定に使用されます。

> ***DIRECT QUERY** ステートメントに、**DISTINCT** 節や **GROUP BY** 節を含めることはできません。*

MEASURE キーワードを使用して、Qlik Sense が「メタレベル」では認識する項目を定義します。メジャー項目の実際のデータは、データロードプロセス中にデータベースの中にのみ存在し、ビジュアライゼーションで使用されるチャートの数式が起動するアドホックベースで取得されます。

通常、軸として使用されることになる、不連続値を含む項目は、**DIMENSION** キーワードでロードする必要がありますが、集計においてのみ使用する数値は、**MEASURE** キーワードを使って選択しなければなりません。

DETAIL 項目は、コメント項目など、ユーザーが詳細をドリルダウンするテーブル ボックスに表示したいと考える可能性のある情報や詳細を提供します。**DETAIL** 項目をチャートの数式で使用することはできません。

DIRECT QUERY ステートメントは、SQL をサポートするデータ ソースに対して中立です。このため、同一の **DIRECT QUERY** ステートメントを変更することなく異なるSQL データベースで使用することができます。Direct Discovery は、必要に応じてデータベースに適したクエリを生成します。

ネイティブ データソース構文は、ユーザーがクエリするデータベースを把握していて、SQL にデータベース特定の拡張機能を利用したい場合に使用できます。ネイティブ データソース構文は、次の場合にサポートされます。

- **DIMENSION** および **MEASURE** 節の項目式
- **WHERE** 節のコンテンツ

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"')
```

> *次の用語はキーワードとして使用されているため、引用符なしで列や項目名として使用することはできません。and, as, detach, detail, dimension, distinct, from, in, is, like, measure, native, not, or, where*

引数:

引数	説明
fieldlist	コンマ区切りの項目指定リスト、*fieldname {, fieldname}*. 項目指定を項目名にすることも可能で、データベースの列名とQlik Sense 項目名に同じ名前が使用される場合があります。または、データベース数式や列名が Qlik Sense の項目名を指定する場合、項目指定を「項目エイリアス」にすることもできます。
tablelist	データのロード元となるデータベースのテーブル名またはビュー名のリストです。これは一般的には、データベース上で実行された JOIN が含まれるビューとなります。

where_clause	データベースの **WHERE** 節の完全な構文は、ここでは定義されませんが、ほとんどの SQL「関係式」は許容されます。これには、関数呼び出しの使用、文字列用の **LIKE** 演算子、**IS NULL**、および **IS NOT NULL** が含まれます。**IN. BETWEEN** は含まれていません。
	
	NOT は、特定のキーワードの修飾子と対照的な単項演算子です。
	
	例:
	
	`WHERE x > 100 AND "Region Code" IN ('south', 'west')`
	`WHERE Code IS NOT NULL and Code LIKE '%prospect'`
	`WHERE NOT X in (1,2,3)`
	最後の例を次のように記述することはできません。
	
	`WHERE X NOT in (1,2,3)`

この例では、TableName と呼ばれるデータベース テーブル (Dim1、Dim2、Num1、Num2、Num3 項目を含む) が使用されます。Dim1 とDim2 は Qlik Sense データセットにロードされます。

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

Dim1 とDim2 は、軸として使用可能です。Num1、Num2、Num3 は、集計に利用可能です。Dim1 とDim2 もまた、集計に利用できます。Dim1 とDim2 が使用される集計タイプは、それぞれのデータタイプによって決まります。例えば、多くの場合 **DIMENSION** 項目には名前や口座番号といった文字列データが含まれています。こうした項目は集計できませんが、`count(Dim1)` を使ってカウントすることはできます。

***DIRECT QUERY** ステートメントは、スクリプト エディターに直接書き込まれます。**DIRECT QUERY** ステートメントの構造をシンプルにするには、データ接続から**SELECT**ステートメントを生成し、生成したスクリプトを編集して**DIRECT QUERY** ステートメントに変更します。*
*例えば、**SELECT** ステートメントを*

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

*次の **DIRECT QUERY** ステートメントに変更できます。*

```
DIRECT QUERY
 DIMENSION
 SalesOrderID,
 RevisionNumber

MEASURE
 SubTotal,
 TaxAmt

DETAIL
OrderDate

FROM MyDB.Sales.SalesOrderHeader;
```

Direct Discovery 項目リスト

項目リストは、コンマ区切りの項目指定リスト、*fieldname {, fieldname}* です。項目指定を項目名にすることも可能で、データベースの列名と項目名に同じ名前が使用される場合があります。または、データベース数式や列名が Qlik Sense の項目名を指定する場合、項目指定を項目エイリアスにすることもできます。

項目名は、単純な名前あるいは引用符で囲まれた名前にすることもできます。単純な名前は、Unicode 英数文字で始まりその後に英数文字や数値、アンダースコアの組み合わせが続きます。引用符で囲まれた名前は、二重引用符で始まりその後任意の文字が続きます。引用符で囲まれた名前に二重引用符が含まれると、それらの引用符は隣接した 2 つの引用部を使用して表されます。

Qlik Sense 項目名は、大文字と小文字を区別します。データベースの項目名は、データベースに応じて大文字と小文字を区別することも、しないこともできます。Direct Discovery クエリは、すべての項目識別子およびエイリアスの大文字と小文字の区別情報を保持します。次の例では、エイリアス "MyState" は内部で使用され、データベース列 "STATEID" からデータを保存します。

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

これは、エイリアスを持つ**SQL Select** ステートメントの結果とは異なります。エイリアスが明示的に引用されてない場合、結果には参照先データベースが返した列のデフォルトの文字設定 (大文字/小文字の) に従った列が含まれています。次の例では、**SQL Select** ステートメントは Oracle データベースに対し "MYSTATE," をエイリアスが大文字小文字混合として指定されているにもかかわらず、すべて大文字で内部 Qlik Sense エイリアスとして生成しています。**SQL Select** ステートメントは、データベースが返した列名を使用し、Oracle ではすべて大文字になります。

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

これを回避するには、LOAD ステートメントを使用してエイリアスを指定します。

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

この例では、"STATEID" 列は Qlik Sense によって "MyState" として内部的に保存されます。

大半のデータベース スカラ式は、項目指定として許容されます。また、関数呼び出しも項目指定で使用できます。数式には、ブール値や数値、単一引用符に囲まれた文字列 (埋め込み型単一引用符は連続する単一引用符で表されます) などの制約を含めることができます。

```
DIRECT QUERY
   DIMENSION
      SalesOrderID, RevisionNumber
   MEASURE
      SubTotal AS "Sub Total"
FROM AdventureWorks.Sales.SalesOrderHeader;

DIRECT QUERY
   DIMENSION
      "SalesOrderID" AS "Sales Order ID"
   MEASURE
      SubTotal,TaxAmt,(SubTotal-TaxAmt) AS "Net Total"
FROM AdventureWorks.Sales.SalesOrderHeader;

DIRECT QUERY
   DIMENSION
      (2*Radius*3.14159) AS Circumference,
```

```
            Molecules/6.02e23 AS Moles

         MEASURE

            Num1 AS numA

      FROM TableName;


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

Direct Discovery は、**LOAD** ステートメントでの集計の使用はサポートしません。集計を使用すると、予測不能な結果が生じます。次のような **LOAD** ステートメントは使用しないでください。

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

SUMは、**LOAD**ステートメントに含めることはできません。

また、Direct Discovery は Qlik Sense 関数 (**Direct Query** ステートメント内) に対応していません。例えば、**DIMENSION** 項目に対する次のような指定は、"Mth" 項目がビジュアライゼーションの軸として使用されている場合には失敗します。

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

Directory

Directoryステートメントは、新たな**Directory**ステートメントが作成されるまで、後続の**LOAD**ステートメントのどのディレクトリでデータファイルを検索するか定義します。

構文:

```
Directory[path]
```

Directory ステートメントを **path** なしで、あるいは省略して発行すると、Qlik Sense は Qlik Sense 作業ディレクトリを探します。

引数:

引数

引数	説明
path	data ファイルのパスとして解釈されるテキスト。 ファイルのパスは以下のいずれかになります。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***

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

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

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

Disconnect

Disconnect ステートメントは、現在の ODBC/OLE DB/カスタム接続を終了します。このステートメントはオプションです。

構文:

```
Disconnect
```

新しい **connect** ステートメントの実行が開始または終了した時点で、接続が自動的に終了します。

```
Disconnect;
```

Drop

Drop スクリプト キーワードを使用すると、データベースからテーブルや項目をドロップできます。

Drop field

drop field ステートメントを使用すると、スクリプトの実行中にいつでもデータ モデルやメモリから1つ以上の Qlik Sense 項目を削除できます。テーブルの「個別」のプロパティは、**drop field** ステートメントの後に削除されます。

drop field *と* ***drop fields*** *では同じ結果が得られるため、どちらを使用しても構いません。テーブルが指定されていない場合は、その項目が存在するすべてのテーブルから項目が削除されます。*

構文:

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

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

Drop table

drop table ステートメントを使用すると、スクリプトの実行中にいつでもデータ モデルやメモリから1つ以上の Qlik Sense 内部テーブルを削除できます。

構文:

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

形式は ***drop table*** *と* ***drop tables*** *のどちらでも構いません。*

以下のアイテムが削除されます。

- 実際のテーブル
- 残されたテーブルに属さないすべての項目。
- 残された項目に含まれる項目値 (削除されたテーブルから排他的に発生)。

例と結果:

例	結果
drop table Orders, Salesmen, T456a;	メモリから3つのテーブルが削除されます。
Tab1: Load * Inline [Customer, Items, UnitPrice Bob, 5, 1.50]; Tab2: LOAD Customer, Sum(Items * UnitPrice) as Sales resident Tab1 group by Customer; drop table Tab1;	テーブル *Tab2* を作成すると、テーブル *Tab1* がドロップされます。

Drop table

drop table ステートメントを使用すると、スクリプトの実行中にいつでもデータモデルやメモリから1つ以上の Qlik Sense 内部テーブルを削除できます。

構文:

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

*形式は **drop table** と**drop tables** のどちらでも構いません。*

以下のアイテムが削除されます。

- 実際のテーブル
- 残されたテーブルに属さないすべての項目。
- 残された項目に含まれる項目値 (削除されたテーブルから排他的に発生)。

例と結果:

例	結果
drop table Orders, Salesmen, T456a;	メモリから3つのテーブルが削除されます。

例	結果
Tab1: Load * Inline [Customer, Items, UnitPrice Bob, 5, 1.50]; Tab2: LOAD Customer, Sum(Items * UnitPrice) as Sales resident Tab1 group by Customer; drop table Tab1;	テーブル *Tab2* を作成すると、テーブル *Tab1* がドロップされます。

Execute

Execute ステートメントはその他のプログラムの実行に使用しますが、Qlik Sense ではデータのロードを行います。例えば、必要な変換を行う場合などです。

この機能は Qlik Sense SaaS では使用できません。

このステートメントは標準モードではサポートされていません。

構文:

```
execute commandline
```

引数:

引数

引数	説明
commandline	オペレーティング システムがコマンド ラインとして解釈するテキスト。絶対ファイル パスまたは lib:// フォルダ パスを参照できます。

Execute を使用する場合は、以下の条件を満たしている必要があります。

- レガシー モードで稼働する必要がある (Qlik Sense および Qlik Sense Desktop に該当する場合)。
- *Settings.ini* で OverrideScriptSecurity を 1 に設定する必要がある (Qlik Sense に該当する場合)。*Settings.ini* が *C:\ProgramData\Qlik\Sense\Engine* にあり、空のファイルである。

*OverrideScriptSecurity で **Execute** を有効に設定すると、すべてのユーザーがサーバーでファイルを実行できるようになります。たとえば、実行可能ファイルをアプリに添付し、データ ロード スクリプトでファイルを実行することができます。*

次の手順を実行します。

1. *Settings.ini* のコピーを作成し、テキストエディタで開いてください。
2. ファイルの最初の行に *[Settings 7]* が含まれているか確認します。
3. 新しい行を挿入し、*OverrideScriptSecurity=1* を入力します。
4. ファイルの最後に空の行を挿入します。
5. ファイルを保存します。
6. 編集済みファイルの *Settings.ini* 代替として使用します。
7. Qlik Sense Engine Service (QES) を再起動します。

Qlik Sense がサービスとして起動している場合は、コマンドが予想通りに動作しない場合があります。

```
Execute C:\Program Files\Office12\Excel.exe;
Execute lib://win\notepad.exe // win is a folder connection referring to c:\windows
```

Field/Fields

Field および **Fields** スクリプトキーワードは、**Declare**、**Derive**、**Drop**、**Comment**、**Rename**、**Tag/Untag** の各ステートメントで使用されます。

FlushLog

FlushLogステートメントによって、Qlik Senseは強制的にスクリプトバッファの内容をスクリプトログファイルに書き込みます。

構文:

```
FlushLog
```

バッファのコンテンツがログファイルに書き込まれます。このコマンドは、スクリプトの実行が失敗した際に失われたデータを取得できるため、特にデバッグに有用です。

```
FlushLog;
```

Force

force ステートメントにより、Qlik Sense は後続の **LOAD** および **SELECT** ステートメントの項目値を大文字のみ、小文字のみ、常に先頭を大文字化、またはそのまま (混合) として強制的に解釈します。このステートメントを使用すると、異なる表記規則に従って作成されたテーブルの項目値を関連付けられます。

構文:

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

何も指定されない場合、大文字と小文字を混在させると見なされます。force ステートメントは、新たな force ステートメントが実行されるまで有効です。

force ステートメントは、アクセス セクションでは使用できず、ロードされる項目値は大文字と小文字が区別されません。

例と結果

例	結果

この例には、大文字と小文字を変えない方法が示されています。
`FORCE Capitalization;`
`Capitalization:`
`LOAD * Inline [`
`ab`
`Cd`
`eF`
`GH`
`];`
	Capitalization テーブルには、次の値が含まれます。
	`Ab`
	`Cd`
	`Ef`
	`Gh`
	すべての値は大文字で表示されます。

この例には、大文字で表記する方法が示されています。
`FORCE Case Upper;`
`CaseUpper:`
`LOAD * Inline [`
`ab`
`Cd`
`eF`
`GH`
`];`
	CaseUpperテーブルには、次の値が含まれます。
	`AB`
	`CD`
	`EF`
	`GH`
	すべての値は大文字です。

例	結果
この例には、小文字で表記する方法が示されています。 FORCE Case Lower; CaseLower: LOAD * Inline [ab Cd eF GH];	**CaseLower**テーブルには、次の値が含まれます。 ab cd ef gh すべての値は小文字です。
この例では、大文字と小文字を混在させる方法が示されています。 FORCE Case Mixed; CaseMixed: LOAD * Inline [ab Cd eF GH];	**CaseMixed**テーブルには、次の値が含まれます。 ab Cd eF GH すべての値は、スクリプトでの表示どおりとなります。

参照先:

From

From スクリプト キーワードは、**Load** ステートメントでは、ファイルを参照するために使用され、**Select** ステートメントでは、データベース テーブルを参照したり、表示したりするために使用されます。

Load

LOAD ステートメントは、ファイル、スクリプトで定義されたデータ、事前にロードされたテーブル、Web ページ、後続の **SELECT** ステートメントの結果、または自動生成されたデータから項目をロードします。分析接続からデータをロードすることもできます。

構文:

```
LOAD [ distinct ] fieldlist

[( from file [ format-spec ] |

from_field fieldassource [format-spec]|

inline data [ format-spec ] |

resident table-label |

autogenerate size ) |extension pluginname.functionname([script]
tabledescription)]

[ where criterion | while criterion ]

[ group by groupbyfieldlist ]

[order by orderbyfieldlist ]
```

引数

引数	説明
distinct	一意のレコードのみをロードする場合、**distinct** を述語として使用できます。重複するレコードがある場合は、1 つめのインスタンスがロードされます。 先行する LOAD を使用している場合、**distinct** はロード先のテーブルにのみ反映されるので、Load ステートメントの先頭に **distinct** を配置する必要があります。

引数	説明
fieldlist	*fieldlist ::= (* | field{, * | field })* ロードする項目のリスト。項目リストとして ***** を使用すると、テーブルのすべての項目が指定されます。 *field ::= (fieldref | expression) [***as** *aliasname]* 項目定義には、リテラル、既存項目への参照、または数式を含める必要があります。 *fieldref ::= (fieldname* |@*fieldnumber* |@*startpos:endpos [* ***I*** | ***U*** | ***R*** | ***B*** | **T**]) *fieldname* は、テーブル内の項目名と同じテキストです。項目名にスペースなどが含まれる場合は、ストレート二重引用符または角括弧で囲む必要があります。明示的に表現できない項目名については、次のような表記規則を使用します。 @*fieldnumber* は、区切り記号付きテーブル ファイルの項目番号を表します。「@」が前に付いた正の整数でなければなりません。常に 1 から項目の数まで、番号が振られています。 @*startpos*:*endpos* は、固定長レコードが含まれるファイル内の項目の開始および終了位置を表します。位置はどちらも正の整数でなければなりません。2 つの番号の前に「@」を付け、コロン (:) で区切る必要があります。常に 1 から位置の数までの番号が付けられます。最後の項目で、**n** は終了位置として使用されます。 • @*startpos*:*endpos* の直後に **I** か **U** の文字が続く場合は、バイトの読み取りは符号付き (**I**) バイナリまたは符号なし (**U**) の整数 (Intel のバイト順) と解釈されます。読み取られる位置の数は、1、2、または 4 です。 • @*startpos*:*endpos*の直後に文字 **R** が続く場合は、読み取られるバイトはバイナリの実数 (IEEE 32 ビットまたは 64 ビットの浮動小数点) として解釈されます。読み取られる位置の数は、4 または 8 です。 • @*startpos*:*endpos* の直後に文字 **B** が続く場合は、読み取られるバイトは COMP-3 標準に従った BCD (Binary Coded Decimal) 数として解釈されます。任意のバイト数を指定できます。 *expression* は、同じテーブルにある 1 つまたは複数の項目に基づいた数値関数または文字列関数です。詳細については、数式の構文を参照してください。 項目に新しい名前を割り当てるには、**as** を使用します。

引数	説明
from	**from** は、データをフォルダーまたは Web ファイル データ接続を使用するファイルからロードする必要がある場合に使用します *file ::= [path] filename* ***'lib://Table Files/'*** パスを省略すると、Qlik Sense は、**Directory** ステートメントで指定されたディレクトリのファイルを検索します。**Directory** ステートメントがない場合、Qlik Sense は作業ディレクトリ *C:\Users\{user}\Documents\Qlik\Sense\Apps* を検索します。 *Qlik Sense の場合、作業ディレクトリは Qlik Sense Repository Service で指定されています。これはデフォルトで、C:\ProgramData\Qlik\Sense\Apps です。* *filename* には、標準の DOS ワイルドカード文字 (* および ?) が含まれる場合があります。これにより、指定されたディレクトリ内にあるすべての一致ファイルがロードされます。 *format-spec ::= (fspec-item { , fspec-item })* この書式指定は、括弧に囲まれた複数の書式指定アイテムのリストで構成されます。 **レガシー スクリプティング モード** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com*** •

引数	説明
from_field	事前にロードされた項目からデータをロードする場合は、**from_field** を使用します。 *fieldassource::=***(***tablename, fieldname***)*** 項目は、事前にロードされた *tablename* と *fieldname* の名前です。 *format-spec ::= (fspec-item {, fspec-item })* この書式指定は、括弧に囲まれた複数の書式指定アイテムのリストで構成されます。詳細については、「*書式指定アイテム (page 165)*」を参照してください。 ***from_field*** *は、テーブルの項目を区切るときのリスト区切り記号として、カンマのみをサポートします。*
inline	スクリプト内でデータを入力し、ファイルからロードしない場合は、**inline** を使用します。 *data ::= [text]* **inline** 句を通じて入力されるデータは、角括弧、引用符、またはバックティックなどの特定の文字で囲む必要があります。括弧で囲まれたテキストは、ファイルのコンテンツと同じ方法で解釈されます。そのため、テキストファイル内で新しい行を挿入する場合は、**inline** 句のテキスト内でも新しい行を挿入する必要があります。つまり、スクリプトを入力するときに Enter キーを押します。 単純なインライン ロードでは、列の数は最初の行で定義されます。 *format-spec ::= (fspec-item {, fspec-item })* 他のロードされたテーブルで使用できる同じ書式指定アイテムの多くを使用して、インライン ロードをカスタマイズできます。これらのアイテムは括弧内にリストされています。詳細については、「*書式指定アイテム (page 165)*」を参照してください。 インライン ロードの詳細については、「インライン ロードを使用したデータのロード」を参照してください。
resident	事前にロード済みのテーブルからデータをロードする場合は、**resident** を使用します。 *table label* は、元のテーブルを作成した **LOAD** または **SELECT** ステートメントの前に配置されるラベルです。ラベルの最後にはコロン (:) を記述します。
autogenerate	Qlik Sense でデータを自動生成する場合は、**autogenerate** を使用します。 *size ::= number* *Number* は、生成するレコード数を示す整数です。 **Peek** 関数を使用して、以前にロードされたテーブルの 1 つの項目値を参照しない限り、項目のリストには、外部データ ソースまたは以前にロードされたテーブルからデータを取得する必要のある数式を記述できません。

引数	説明
extension	分析接続からデータをロードすることができます。サーバーサイド拡張 (SSE) プラグインで定義されている関数を呼び出す、またはスクリプトを評価する **extension** 節を使用する必要があります。 SSE プラグインに単一のテーブルを送ることができます。単一のデータテーブルが返されます。返す項目名が SSE プラグインで指定されていない場合、項目には Field1, Field2 などの名前が付けられます。 `Extension pluginname.functionname( tabledescription );` • SSE プラグインの関数を使用したデータのロード *tabledescription ::= (table { ,tablefield})* テーブルの項目を記述していない場合、項目はロードの順で使用されます。 • SSE プラグイン内のスクリプト評価によるデータのロード *tabledescription ::= (script, table { ,tablefield})* **テーブル項目定義におけるデータ型の扱い** データ型は、分析接続で自動的に検出されます。データに数値が含まれず、少なくとも 1 個の非 NULL テキスト文字列が含まれる場合、その項目はテキストとみなされます。それ以外の場合は数値とみなされます。 **String()** または **Mixed()** で項目名を囲むことで、データ型を強制的に指定できます。 • **String()** は項目をテキストに指定します。項目が数値の場合、デュアル値のテキスト部分が抽出され、変換は実行されません。 • **Mixed()** は項目をデュアルに指定します。 **拡張**テーブル項目定義以外では、**String()**、**Mixed()** は使用できず、テーブル項目定義では他の Qlik Sense 関数を使用できません。 **分析接続に関する詳細** 分析接続は、使用する前に設定が必要です。
where	**where** 節は、レコードを選択に含めるかどうかを示します。*criterion* が True の場合は選択が含まれます。 *criterion* は論理式です。
while	**while** は、レコードを繰り返し読み取るかどうかを示す節です。*criterion* が True の場合は、同じレコードが読み取られます。通常、**while** 節には **IterNo( )** 関数が含まれていなければなりません。 *criterion* は論理式です。

引数	説明
group by	データを集計 (グループ化) すべき項目を定義するには、**group by** 節を使用します。集計項目は、ロードする数式に挿入しなければなりません。集計項目以外の項目は、ロードした数式に含まれる集計関数の外部で使用できます。 *groupbyfieldlist ::= (fieldname { ,fieldname })*
order by	**order by** 節は、**load** ステートメントで処理される前に、常駐テーブルのレコードをソートします。1 つ以上の項目の昇順または降順で、常駐テーブルをソートできます。最初に数値、次に各国の照合順でソートされます。この節は、データソースが常駐テーブルの場合に限り使用できます。 順序項目は、常駐テーブルをソートする項目を指定します。項目は、名前または常駐テーブル内での番号 (最初の項目が番号 1) で指定できます。 orderbyfieldlist ::= fieldname [sortorder] { , fieldname [sortorder] } *sortorder* は、昇順の *asc* または降順の *desc* のどちらかになります。*sortorder* を指定しない場合は、*asc* と見なされます。 *fieldname*、*path*、*filename*、*aliasname* は、それぞれの名前を示すテキスト文字列です。ソーステーブルのフィールドは *fieldname* として使用できます。ただし、as 節 (*aliasname*) を使用して作成された項目は範囲外になり、同じ **load** ステートメント内では使用できません。

from、**inline**、**resident**、**from_field**、**extension**、または **autogenerate** 節でデータのソースが指定されない場合、データは直後の **SELECT** または **LOAD** ステートメントの結果からロードされます。後続のステートメントには、プレフィックスを記述できません。

さまざまなファイル形式のロード
区切り記号付きデータファイルを既定のオプションでロードします。

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

区切り記号付きデータファイルをライブラリ接続 (DataFiles) からでロードします。

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

区切り記号付きデータファイルをすべてライブラリ接続 (DataFiles) からでロードします。

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

コンマを区切り記号として指定し、埋め込みラベル付きで、区切り記号付きファイルをロードします。

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

タブを区切り記号として指定し、埋め込みラベル付きで、区切り記号付きファイルをロードします。

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

difファイルを埋め込みヘッダー付きでロードします。

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

固定長レコードファイルから、ヘッダーなしで3項目をロードします。

```
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);
```

QVXファイルを、絶対パスを指定してロードします。

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

Webファイルのロード
Webファイル データ接続における既定のURLセットからロード:

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

特定のURLからロードし、Webファイル データ接続におけるURLセットを上書き:

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

ドル記号展開を使用し、変数内の特定のURLセットからロード:

```
SET dynamicURL = 'http://localhost/foo.bar';
```

```
LOAD * from [lib://MyWebFile] (URL is '$(dynamicURL)');
```

特定の項目の選択、項目名の変更および項目の計算
区切り記号付きファイルから特定の3項目のみロードします。

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

ラベルなしでファイルをロードする場合、Rename最初の項目の名前をAに変更し、2番目の項目の名前をBに変更します。

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

Nameを、FirstName、空白文字、およびLastNameの連結としてロードします。

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

Quantity、Price、およびValue (QuantityとPriceの積) をロードします。

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

特定のレコードの選択
一意のレコードのみロードします。複製されたレコードは破棄されます。

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

項目Litresがゼロ(0)より大きい値を持つレコードのみロードします。

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

ファイル上にないデータおよび自動生成されたデータのロード
CatIDおよび Category という2 項目のインライン データを持つテーブルをロードします。

```
LOAD * Inline

[CatID, Category

0,Regular

1,Occasional

2,Permanent];
```

UserID、Password、および Access という3 項目のインライン データを持つテーブルをロードします。

```
LOAD * Inline [UserID, Password, Access

A, ABC456, User

B, VIP789, Admin];
```

10 000 行を持つテーブルをロードします。項目 A には、読み取られたレコード (1,2,3,4,5...) の数が含まれ、項目 B には 0 ~ 1 間の乱数が含まれます。

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

autogenerate 後の丸かっこはオプションです。

事前にロードされているテーブルからのデータのロード
最初に、区切り記号付きテーブル ファイルをロードし、tab1 という名前を付けます。

```
tab1:

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

ロード済みのテーブル tab1 から tab2 として項目をロードします。

```
tab2:

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

ロード済みのテーブル tab1 から項目 (ただし、A が B より大きいレコードのみ) をロードします。

```
tab3:

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

ロード済みのテーブル tab1 から、A によって指定された項目をロードします。

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

ロード済みのテーブル tab1 から、最初の項目によって指定された項目をロードし、次に 2 番目の項目によって指定された項目をロードします。

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

ロード済みのテーブル tab1 から、C によって指定された項目を降順でロードし、次に B によって指定された項目を昇順でロードし、最初の項目を降順でロードします。

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

事前にロードされている項目からのデータのロード
ロード済みのテーブル Characters から、項目 Types を A としてロードします。

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

後続のテーブルからのデータのロード (先行する LOAD)
後続の **SELECT** ステートメントにロードされている Table1 から、A、B、および計算された項目 X と Y をロードします。

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

SELECT A,B,C,D from Table1;
```

データのグループ化
ArtNo によってグループ化 (集計) された項目をロードします。

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

Week と ArtNo によってグループ化 (集計) された項目をロードします。

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

1 つのレコードの反復読み取り
この例では、ひとつの項目に各生徒の成績が要約して含まれている、入力ファイル Grades.csv があります。

```
Student,Grades

Mike,5234

John,3345

Pete,1234

Paul,3352
```

成績は、1 から 5 に分かれていて、科目 Math、English、Science、および History を表しています。**IterNo( )** 関数をカウンタとして使用して、各レコードを複数回、**while** 句で読み取り、成績を個々の値に分けることができます。読み取るたびに、**Mid** 関数で成績が抽出され、Grade に保存され、科目が **pick** 関数を使用して選択され、Subject に保存されます。最後の **while** 節には、全成績が読み取られたか確認するテスト (この場合、生徒 1 人に付き 4 教科分) が含まれています。その後、次の生徒の成績を読み取ります。

```
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));
```

結果は、このデータが含まれるテーブルにあります。

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

分析接続からのロード
次のサンプル データを使用します。

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

関数を使用したデータのロード

以下の例では、カスタム関数 *Calculate(Parameter1, Parameter2)* を含む *P* という名前の分析接続プラグインがあるものと仮定しています。この関数は、*Field1* および *Field2* という項目を含む、テーブル *Results* を返します。

```
Load * Extension P.Calculate( Values{A, C} );
```
項目 A および C を関数に送るときに返されるすべての項目をロードします。

```
Load Field1 Extension P.Calculate( Values{A, C} );
```
項目 A および C を関数に送るときに項目 Field1 のみをロードします。

```
Load * Extension P.Calculate( Values );
```
項目 A および B を関数に送るときに返されるすべての項目をロードします。項目が指定されていない場合、A および B がテーブル内の順序で最初に使用されます。

```
Load * Extension P.Calculate( Values {C, C});
```
項目 C を関数の両方のパラメータに送るときに返されるすべての項目をロードします。

```
Load * Extension P.Calculate( Values {String(A), Mixed(B)});
```
文字列として指定された項目 A および数値として指定された B を関数に送るときに返されるすべての項目をロードします。

スクリプト評価によるデータのロード

```
Load A as A_echo, B as B_echo Extension R.ScriptEval( 'q;', Values{A, B} );
```
A および B の値を送るときにスクリプト q によって返されるテーブルをロードします。

```
Load * Extension R.ScriptEval( '$(My_R_Script)', Values{A, B} );
```
A および B の値を送るときに My_R_Script 変数に格納されるスクリプトによって返されるテーブルをロードします。

```
Load * Extension R.ScriptEval( '$(My_R_Script)', Values{B as D, *} );
```
D、A および C に名前が変更された B の値を送るときに My_R_Script 変数に格納されるスクリプトによって返されるテーブルをロードします。* を使用して、参照されていない残りの項目を送信します。

DataFiles 接続のファイル拡張子は大文字と小文字を区別します。例: .qvd。

書式指定アイテム

各書式指定アイテムは、次のようなテーブル ファイルのプロパティを定義します。

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

文字セット

文字セットは、ファイルで使われる文字セットを定義する**LOAD** ステートメントのファイル指定子です。

QlikView で使われていた**ansi**、**oem** 、**mac** 指定子も使えますが、Qlik Sense で **LOAD** ステートメントを作成する場合、これらの指定子は生成されません。

構文:

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

引数:

引数

引数	説明
utf8	UTF-8 文字セット
unicode	Unicode 文字セット
ansi	Windows、コードページ 1252
oem	DOS、OS/2、AS400、その他
mac	コードページ 10000
codepage is	**codepage** 指定子を使用すると、あらゆる Windows コードページを *N* として使用できます。

制限事項:

oem 文字セットからの変換は macOS には実装されていません。何も指定されない場合、Windows ではコードページ 1252 と見なされます。

```
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)
```

参照先:

Load (page 155)

テーブル形式

テーブルの書式は、ファイルの種類を定義する**LOAD** ステートメントのファイル指定子です。何も指定されない場合、*.txt* ファイルと見なされます。

テーブル形式のタイプ

タイプ	説明
txt	区切り記号付きテキストファイルでは、テーブル内の列は区切り文字で区切られます。
fix	固定レコード長ファイルでは、各項目は正確に特定の文字数を幅とする文字列です。 通常、多くの固定レコード長ファイルには、ラインフィードによって区切られたレコードが含まれていますが、レコードのサイズをバイト単位で指定したり、**Record is** を使用して複数行に拡大したりする、より高度なオプションがあります。 *データにマルチバイト文字が含まれている場合、書式がバイト単位の固定長に基づいているため、項目分割が不揃いになる場合があります。*
dif	*.dif* (Data Interchange Format) ファイルでは、テーブルを定義する特殊な書式が使用されます。
biff	Qlik Sense は、*biff* 形式 (Binary Interchange File Format) を使用して、標準の Excel ファイルのデータを解釈することもできます。
ooxml	Excel 2007 以降のバージョンでは、ooxml *.xslx* 形式を使用しています。 **Table is** 指定子を使用して、テーブルとしてロードされるシート名を定義できます。 *Table is (page 171)*
html	テーブルが html ページまたはファイルの一部になっている場合は、html を使用してください。
xml	xml (Extensible Markup Language) は、テキスト形式でデータ構造を示す一般的なマークアップ言語です。 **Table is** 指定子を使用して、テーブルとしてロードされる XML のパスを定義できます。 *Table is (page 171)*
qvd	*qvd* 形式は、Qlik Sense アプリからエクスポートされた独自の QVD ファイル形式です。
qvx	*qvx* は、Qlik Sense への高効率アウトプットを提供するファイル/ストリーム形式です。

タイプ	説明
parquet	Apache Parquet は、大きいデータセットの保存とクエリに非常に効率的な列型ストレージ フォーマットです。 ネストされたデータを含む Parquet ファイルでは、**Table is** を使用してロードするテーブルを Parquet ファイルから指定できます。例: `LOAD * FROM [lib://DataFiles/company.parquet] (parquet, table is [company:salesrep.salesrep]);`。 *Table is (page 171)*

Delimiter is

区切り記号付きテーブル ファイルでは、**delimiter is** 指定子を使用して、任意の区切り記号を指定できます。この指定子は、区切り記号付きの .txt ファイル専用です。

構文:

```
delimiter is char
```

引数:

引数

引数	説明
char	127 ASCII 文字から単一の文字を指定します。

さらに、以下の値も使用できます。

オプション値

値	説明
'\t'	引用符付き/引用符なしのタブ記号を表します。
'\\'	円記号 (\ ) を表します。
'spaces'	1 つ以上のスペースのあらゆる組み合わせを表します。32 を下回る ASCII 値を持つ印刷不能な文字で、例外として CR と LF はスペースと解釈されます。

何も指定されていない場合は、**delimiter is ','** と見なされます。

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

参照先:

- *Load (page 155)*

No eof

no eof 指定子は、区切り記号付き **.txt** ファイルをロードする場合、ファイルの最後の文字を無視する際に使用します。

構文:

```
no eof
```

no eof 指定子を使用している場合、ファイルの最後を示すコードポイント26の付いた文字は無視され、項目の値の一部となります。

この指定子は、区切り記号付きファイルにのみ使用できます。

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

参照先:

- *Load (page 155)*

Labels

Labels は、ファイルのどこに項目名が位置するかを定義する **LOAD** ステートメントのファイル指定子です。

構文:

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

項目名は、ファイル内のさまざまな場所に配置できます。1件目のレコードに項目名が含まれる場合は、**embedded labels** を使用します。項目名がない場合は、**no labels** を使用します。*dif* ファイルでは、明示的な項目名を持つ別のヘッダー セクションが使用されることがあります。その場合、**explicit labels** を使用します。何も指定しないと、**embedded labels**、*dif* ファイルと見なされます。

Example 1:

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

Example 2:

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

参照先:

- *Load (page 155)*

Header is

テーブル ファイルでヘッダーのサイズを指定します。**header is** 指定子で、任意のヘッダー長を指定できます。ヘッダーは、Qlik Sense で使用されないテキスト セクションです。

構文:

```
header is n
```

```
header is line
```

```
header is n lines
```

ヘッダー長は、バイト単位 (**header is n**)、または行単位 (**header is line** または **header is n lines**) で指定できます。**n** は、ヘッダー長を表す正の整数です。何も指定されていない場合、**header is 0** と見なされます。**header is** 指定子は、テーブル ファイルにのみ使用できます。

これは、Qlik Sense によってデータとして解釈されてはならないヘッダー テキスト行を含むデータ ソース テーブルの例です。

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

header is 1 lines 指定子を使用すると、最初の行はデータとしてロードされません。この例の **embedded labels** 指定子は、最初の非除外行に項目ラベルが含まれるものとして解釈するようQlik Sense に指示します。

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

結果として、2 つの項目 Col1 とCol2を含むテーブルが作成されます。

参照先:

Load (page 155)

Record is

固定レコード長のファイルでは、レコード長を **record is** 指定子で指定する必要があります。

構文:

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

引数:

引数

引数	説明
n	バイト単位でレコード長を指定します。
line	行単位でレコード長を指定します。
n lines	行単位でレコード長を指定します (n はレコード長を表す正の整数)。

制限事項:

record is 指定子は、**fix** ファイルにのみ使用できます。

参照先:

Load (page 155)

Table is

Excel、XML、Parquet ファイルの場合、テーブル形式指定子でデータのロード元のテーブルを指定できます。

構文:

```
Table is table name
```

引数:

引数

引数	説明

table name	テーブルの名前を指定します。値はテーブル形式に応じて異なります。
	• Excel: シート名。
	• XML: ロードする XML の部分を指定するパス。
	• Parquet: *<node>.<node>.<node>* の形式でテーブルを指定するパス。
	ネストされた構造のテーブルを指定する場合は、**Table is** を使用します。
	たとえば、次のようなスキーマの Parquet データがあるとします。
	`Schema:`
	`Field(name: "Name", datatype: String),`
	`Field(name: "Age", datatype: Float),`
	`Field(name: "Phone", datatype: List(`
	`Field(name: "Item", datatype: Struct[`
	`Field(name: "Number", datatype: String)`
	Phone とそのネストされた項目を、引数 `Table is [Schema:Phone.Item]` を使用してテーブルとしてロードできます。これにより、テーブルにキー項目 *%Key_Phone* が生成されます。

Excel

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

防ぐために、

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

Parquet

company.parquet ファイルに次のスキーマが含まれています。

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

次の例では、ファイルの内容がテーブルにロードされます。最初の load ステートメントはルートグループをロードします。2 番目の load ステートメントは、*salesrep* グループの内容をテーブルとしてロードします。3 番目は、*headquarter* グループをテーブルとしてロードします。4 番目は、*region* グループをテーブルとしてロードします。

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

制限事項:

Table is 指定子は、Excel、XML、Parquet ファイルにのみ関連します。

Quotes

Quotes は、**LOAD** ステートメントのファイル指定子で、引用符を使用できるかどうか、また、引用符と区切り文字間の優先を定義します。テキストファイルのみで使用できます。

構文:

```
no quotes
```

msq

指定子を省略する場合、標準的な引用符 (" " または ' ') を用いることができますが、これらの使用は項目値の最初と最後の文字 (空白は不可) に限られます。

引数:

引数

引数	説明
no quotes	テキストファイルで、引用符が許可されない場合に使用します
msq	新しいスタイルの引用符を指定するためのもので、項目に複数行の内容を含めることができます。改行文字を含む項目は、ダブルクォートで囲む必要があります。 msq オプションの限界のひとつは、項目の内容の最初または最後の文字としてダブルクォート (") 文字がひとつだけ使われると、複数行の最初または最後として解釈され、ロードされるデータセットで予測できない結果につながる可能性があるという点です。この場合は、標準的な引用符を使用して指定子を省略してください。

XML

このスクリプト指定子は、xml ファイルをロードする際に使用します。**XML** 指定子の有効なオプションは、構文にリストされています。

Qlik Sense の DTD ファイルはロードできません。

構文:

```
xmlsimple
```

参照先:

- *Load (page 155)*

KML

このスクリプト指定子は、マップビジュアライゼーションで使用する KML ファイルをロードする際に使用します。

構文:

```
kml
```

KML ファイルは、ポリゴンで表されるエリアデータ (国や地域など)、ラインデータ (線路や道路など)、または [緯度, 軽度] の形式で表されるポイントデータ (都市や場所など) のいずれかで表すことができます。

URL is

このスクリプト指定子は、Web ファイルをロードするときに Web ファイル データ接続の URL を設定するために使用します。

構文:

```
URL is string
```

引数:

引数

引数	説明
string	ロードするファイルの URL を指定します。この指定により、使用されている Web ファイル接続の URL セットが上書きされます。

制限事項:

URL is 指定子は Web ファイルにのみ使用できます。既存の Web ファイル データ接続を使用する必要があります。

参照先:

- *Load (page 155)*

userAgent is

このスクリプト指定子は、Web ファイルをロードするときにブラウザーのユーザー エージェントを設定するために使用します。

構文:

```
userAgent is string
```

引数:

引数

引数	説明
string	ブラウザー ユーザー エージェント文字列を指定します。この指定により、既定のブラウザー ユーザー エージェント "Mozilla/5.0" は上書きされます。

制限事項:

userAgent is 指定子は Web ファイルにのみ使用できます。

参照先:

- *Load (page 155)*

Let

let ステートメントは、**set** ステートメントを補完し、スクリプト変数を定義する際に使用します。**let** ステートメントでは、**set** ステートメントとは逆に、変数に代入する前に、スクリプトの実行時に「=」の右側の数式が評価されます。

構文:

```
Let variablename=expression
```

例と結果:

例	結果
Set x=3+4; Let y=3+4; z=$(y)+1;	$(x) は '3+4' として評価されます $(y) は '7' として評価されます $(z) は '8' として評価されます **Set** ステートメントと**Let** ステートメントの違いに注意してください。**Set** ステートメントは文字列「3+4」を変数に割り当てますが、**Let** ステートメントは文字列を評価して7を変数に割り当てます。
Let T=now( );	$(T) には現在の時刻の値が渡されます。

Loosen Table

スクリプト実行中に **Loosen Table** ステートメントを使用すると、1つまたは複数の Qlik Sense 内部データテーブルに対して明示的に疎結合を宣言できます。テーブルが疎結合している場合、項目値間のすべての関連付けは削除されます。疎結合したテーブルの各項目を独立した未結合のテーブルとしてロードしても、同じ効果が得られます。疎結合は、データ構造の異なる部分を一時的に隔離するテストの間に有用です。疎結合したテーブルは、点線によりテーブル ビューアで識別できます。スクリプトで **Loosen Table** ステートメントを1度以上使用すると、Qlik Sense はスクリプト実行前に疎結合化されたテーブルの設定を無視します。

構文:

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

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

Loosen Table 構文または **Loosen Tables** 構文のどちらでも使用可能です。

*Qlik Sense が、対話的にまたはスクリプトで明示的に疎結合を宣言されたテーブルで、分割できないデータ構造の循環参照を検出すると、循環参照がなくなるまで、他のテーブルが1つ以上強制的に疎結合にされます。その場合、[**ループ警告**] ダイアログで、警告が表示されます。*

```
Tab1:
```

```
SELECT * from Trans;

Loosen Table Tab1;
```

Map

map ... using ステートメントは、特定のマッピング テーブルの値に、特定の項目値または数式をマップするために使用されます。マッピング テーブルは、**Mapping**ステートメントで作成します。

構文:

```
Map fieldlist Using  mapname
```

自動マッピングは、**Map ... Using** ステートメントの後、スクリプトが終わるまで、または **Unmap** ステートメントが作成されるまで、ロードされた項目に対して実行されます。

マッピングは、項目が Qlik Sense の内部テーブルに保存される一連のイベントの最後に実行されます。つまり、マッピングは数式に項目名が出現するたびに行われるのではなく、内部テーブルの項目名に値を保する際に実行されます。数式レベルでのマッピングが必要な場合は、**Applymap()** 関数を使用する必要があります。

引数:

引数

引数	説明
fieldlist	スクリプトのこの場所からマッピングされる項目のコンマ区切りリスト。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。
mapname	**mapping load** または **mapping select** ステートメントで、以前読み取られたマッピング テーブルの名前。

例と結果:

例	結果
Map Country Using Cmap;	マップ Cmap を使用して項目 Country のマッピングを有効にします。
Map A, B, C Using X;	マップ X を使用して、項目 A、B、C マッピングを有効にします。
Map * Using GenMap;	GenMap を使用し、すべての項目のマッピングを有効にします。

NullAsNull

NullAsNull ステートメントは、以前 **NullAsValue** ステートメントで設定された文字列値への NULL 値の変換を無効にします。

構文:

```
NullAsNull *fieldlist
```

NullAsValue ステートメントはスイッチとしての役割があり、**NullAsValue** または **NullAsNull** ステートメントを使用すると、スクリプトでオン/オフの切り替えができます。

引数:

引数

引数	説明
*fieldlist	**NullAsNull** が有効である項目のコンマ区切りリスト。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。

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

NullAsValue

NullAsValue ステートメントは、NULL を値に変換する項目を指定します。

構文:

```
NullAsValue *fieldlist
```

Qlik Sense のデフォルトでは、NULL 値は不明または未定義の値と見なされます。ただし、特定のデータベースのコンテキストでは、NULL 値は単なる欠損値ではなく、特殊な値と見なされることがあります。通常、NULL 値を他の NULL 値にリンクすることはできませんが、**NullAsValue** ステートメントではこの制約を一時的に無効にできます。

NullAsValue ステートメントにはスイッチの役割があり、その後のロードステートメントで有効になります。オン/オフの切り替えは、**NullAsNull** ステートメントを使用して実行できます。

引数:

引数

引数	説明
*fieldlist	**NullAsValue** が有効である項目のコンマ区切りリスト。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。

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

Qualify

Qualify ステートメントは、項目名の修飾を切り替える際に使用します (項目名がプレフィックスとしてテーブル名を取得するなど)。

構文:

```
Qualify *fieldlist
```

項目名をテーブル名で修飾する **qualify** ステートメントでは、異なるテーブルにある同じ名前の項目の自動結合を一時的に無効にできます。その結果、テーブルで検出された際に項目名が変更されます。新しい名前は、*tablename.fieldname* の形式になります。*Tablename* は現在のテーブルのラベルに相当します。ラベルが存在しない場合は、**LOAD** および **SELECT** ステートメントで **from** の後に現れる名前が採用されます。

修飾は、**qualify** ステートメントの後にロードされたすべての項目で行われます。

スクリプトの実行開始時、デフォルトでは修飾が常に無効に設定されています。**qualify** ステートメントを使用すれば、いつでも項目名の修飾を有効にできます。修飾を無効にするには、**Unqualify** ステートメントを使用します。

> ***qualify*** *ステートメントは、パーシャル リロードと併用できません。*

引数:

引数

引数	説明
*fieldlist	修飾を有効にする項目のコンマ区切りリスト。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。

Example 1:

```
Qualify B;

LOAD A,B from x.csv;

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

2 つのテーブル **x.csv** および **y.csv** は、**A** のみが関連付けられます。3 つの項目は、A、x.B、y.B という結果になります。

Example 2:

例に挙げているように、馴染みのないデータベースについては、1 つまたは少数の項目の関連付けから始めることをお勧めします。

```
qualify *;

unqualify TransID;

SQL SELECT * from tab1;

SQL SELECT * from tab2;

SQL SELECT * from tab3;
```

テーブル*tab1*、*tab2*、*tab3* の関連付けには、**TransID** のみが使用されます。

Rem

rem ステートメントは、スクリプト内に備考やコメントを挿入するため、またスクリプトを削除することなく一時的に無効にするために使用します。

構文:

```
Rem string
```

rem と次のセミコロン「**;**」の間に含まれるすべてがコメントと見なされます。

スクリプト内でコメント化を行うには、他に 2 つの方法があります。

1. **/*** と***/** の間に、コメント化したい部分を配置することにより、スクリプトのどこにでも (ただし、2 つの引用符の間以外) コメントを作成することができます。
2. スクリプト内で **//** を入力すると、同じ行内にある右側のテキストがすべてコメントになります。(例外として、//: はインターネットアドレスの一部に使用されます。)

引数:

引数

引数	説明
string	任意のテキスト。

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

Rename

Rename スクリプトキーワードを使用すると、ロード済みのテーブルや項目の名前を変更できます。

Rename field

このスクリプト関数は、既存の1つ以上のQlik Sense項目をロードした後、名前を変更します。

Qlik Senseでは、変数に項目や関数と同じ名前を付けることは推奨されていません。

rename field構文または**rename fields**構文のどちらでも使用可能です。

構文:

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

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

引数:

引数	説明
mapname	以前にロードされたマッピング テーブル (項目の古い名前と新しい名前が1組以上含まれる) の名前。
oldname	古い項目名。
newname	新しい項目名。

制限事項:

2つの項目名が同じになるように名前を変更することはできません。

Example 1:

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

Example 2:

```
FieldMap:

Mapping SQL SELECT oldnames, newnames from datadictionary;

Rename Fields using FieldMap;
```

Rename table

このスクリプト関数は、既存の1つ以上のQlik Sense内部テーブルをロードした後、名前を変更します。

rename table構文または**rename tables**構文のどちらでも使用可能です。

構文:

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

引数:

引数

引数	説明
mapname	以前にロードされたマッピング テーブル (テーブルの古い名前と新しい名前が1組以上含まれる) の名前。
oldname	古いテーブル名。
newname	新しいテーブル名。

制限事項:

2 つの異なる名前の付いたテーブル名を、同じ名前に変更することはできません。既存のテーブルと同じ名前にテーブルの名前を変更しようとすると、スクリプトでエラーが生成されます。

Example 1:

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

Example 2:

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

Search

Search ステートメントは、スマート検索で項目を含めたり除外したりします。

構文:

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

複数のSearch ステートメントを使用して含める項目の選択を絞り込むことができます。このステートメントは上から下に実行されます。

引数:

引数

引数	説明
*fieldlist	スマート検索に含めるまたは除外する項目のコンマ区切りリストです。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。

検索の例

ステートメント	説明
`Search Include *;`	スマート検索を使った検索にすべての項目を含めます。
`Search Exclude [*ID];`	スマート検索を使った検索からIDで終わるすべての項目を除外します。
`Search Exclude '*ID';`	スマート検索を使った検索からIDで終わるすべての項目を除外します。
`Search Include ProductID;`	スマート検索を使った検索に項目 ProductID を含めます。

これらの 3 つのステートメントをこの順序で組み合わせると、ID で終わる、ProductID 以外のすべての項目がスマート検索を使った検索から除外されます。

Section

section ステートメントでは、後続の **LOAD** および **SELECT** ステートメントをデータとして、またはアクセス権の定義としてみなすかどうかを定義できます。

構文:

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

何も指定されていない場合は、**section application** と見なされます。**section** 定義は、新たな **section** ステートメントが作成されるまで有効です。

```
Section access;

Section application;
```

Select

ODBC データソースまたは OLE DB プロバイダの項目選択は、標準的な SQL **SELECT** ステートメントを介して実行されます。ただし、**SELECT** ステートメントが許可されるかどうかは、使用する ODBC ドライバまたは OLE DB プロバイダによって異なります。**SELECT** ステートメントを使用するには、ソースへのオープンデータ接続が必要です。

構文:

```
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 ]
```

さらに、**union** 演算子を使用して、複数の **SELECT** ステートメントを 1 つに連結できる場合があります。

```
selectstatement Union selectstatement
```

SELECT ステートメントは、ODBC ドライバまたは OLE DB プロバイダによって解釈されるため、ODBC ドライバまたは OLE DB プロバイダの機能の違いにより、一般的な SQL 構文との差が生じる場合があります。

- **as** を使用できない場合があります。(その場合、*aliasname* は必ず *fieldname* の直後に記述する必要があります。)
- *aliasname* を使用すると **as** の記述が必要になる場合があります。
- **distinct**、**as**、**where**、**group by**、**order by**、**union** は、使用できないことがあります。
- ODBC ドライバによっては、前述した引用符の一部を使用できない場合があります。

*これは、SQL **SELECT** ステートメントのすべてを説明したものではありません。例えば、**SELECT** ステートメントはネストしたり、複数の結合を 1 つの **SELECT** ステートメントで実行したり、数式で膨大な関数を使用できる場合があります。*

引数:

引数

引数	説明
distinct	**distinct** は、選択した項目の値の組み合わせが重複している場合に 1 回だけロードする際に使用する述語です。
distinctrow	**distinctrow** は、ソース テーブルに含まれるレコードが重複している場合に 1 回だけロードする際に使用する述語です。
fieldlist	**fieldlist ::= (*| field) {, field }** 選択する項目のリスト。* を使用すると、テーブルのすべての項目が指定されます。 **fieldlist ::= field {, field }** コンマで区切られた 1 つまたは複数の項目のリスト。 **field ::= (fieldref| expression) [as aliasname]** 数式には、他の項目に基づいた数値や文字列関数を 1 つまたは複数使用できます。通常使用できる演算子および関数には、+、-、*、/、& (文字連結)、sum(fieldname)、count(fieldname)、avg(fieldname)(average)、month(fieldname) などがあります。詳細については、ODBC ドライバのマニュアルを参照してください。 **fieldref ::= [tablename.] fieldname** **tablename** と **fieldname** は、それぞれの名前を示すテキスト文字列です。スペースなどを含む場合は、ストレート二重引用符で囲む必要があります。 項目に新しい名前を割り当てるには、**as** 句を使用します。

引数	説明
from	**tablelist ::= table {, table }** 項目が選択されるテーブルのリスト。 **table ::= tablename [[as] aliasname]** **tablename** は、引用符で囲むことも囲まないことも可能です。
where	**where** 節は、レコードを選択に含めるかどうかを示します。 **criterion** は論理式で、非常に複雑になる場合があります。使用できる演算子には、数値演算子と関数 **=**、**<>** または **#(**等号否定)、**>**、**>=**、**<**、**<=**、**and**、**or**、**not**、**exists**、**some**、**all**、**in**、および新しい **SELECT** ステートメントがあります。詳細については、ODBC ドライバまたは OLE DB プロバイダのマニュアルを参照してください。
group by	**group by** 節は、複数のレコードを 1 つに集計 (グループ化) する際に使用します。1 つのグループに含まれる特定の項目のレコードは、すべて同じ値を持たなければなりません。そうでない場合は、項目は sum または average などの数式内でのみ使用する必要があります。1 つまたは複数の項目に基づいた数式は、項目記号の数式で定義されます。
having	**having** 節は、レコードを修飾する **where** 節と同様の方法で、グループを修飾する際に使用します。
order by	**order by** 節は、**SELECT** ステートメントの結果のテーブルのソート順を示します。
join	**join** 修飾子は、複数のテーブルを 1 つに結合することを示します。項目名とテーブル名に、スペースや各国語文字セットが含まれる場合は、引用符で囲む必要があります。Qlik Senseで自動生成されるスクリプトについては、**Connect**ステートメントのデータソース定義で指定される ODBC ドライバまたは OLE DB プロバイダ推奨の引用符が使用されます。

Example 1:

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

Example 2:

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

Example 3:

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

Example 4:

```
SELECT `Order Details`.`Order ID`,
Sum(`Order Details`.`Unit Price` * `Order Details`.Quantity) as `Result`
FROM `Order Details`, Orders
where Orders.`Order ID` = `Order Details`.`Order ID`
group by `Order Details`.`Order ID`;
```

Set

set ステートメントは、スクリプト変数を定義する際に使用します。これらは、文字列、パス、ドライバなどの代入に使用されます。

構文:

```
Set variablename=string
```

Example 1:

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

Example 2:

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

Example 3:

```
Set BudgetYear=2012;
```

Sleep

sleep ステートメントは、指定した時間におけるスクリプトの実行を停止します。

構文:

```
Sleep n
```

引数:

引数	説明
n	*n* は、3600000 ミリ秒 (1 時間) 未満の正の整数です。この値には数式も使用できます。

Example 1:

```
Sleep 10000;
```

Example 2:

```
Sleep t*1000;
```

SQL

SQL ステートメントを使用すると、ODBC または OLE DB 接続から任意の SQL コマンドを送信できます。

構文:

```
SQL sql_command
```

Qlik Sense で ODBC 接続を読み取り専用モードで開いている場合、データベースを更新する SQL ステートメントを送るとエラーが返されます。

以下の構文を使用できます。

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

SELECT ステートメントには、この構文が推奨されます (一貫性のため)。ただし、SQL プレフィックスは **SELECT** ステートメントではオプションです。

引数:

引数	説明
sql_command	有効な SQL コマンド。

Example 1:

```
SQL leave;
```

Example 2:

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

SQLColumns

sqlcolumns ステートメントは、**connect** が実行される ODBC または OLE DB データソースの列を記述する項目セットを返します。

構文:

```
SQLcolumns
```

この項目を **sqltables** および **sqltypes** コマンドで生成される項目と組み合わせると、データベースの把握に役立ちます。12 の標準項目は次のとおりです。

TABLE_QUALIFIER

TABLE_OWNER

TABLE_NAME

COLUMN_NAME

DATA_TYPE

TYPE_NAME

PRECISION

LENGTH

SCALE

RADIX

NULLABLE

REMARKS

これらの項目の詳細については、ODBC リファレンスを参照してください。

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

このコマンドをサポートしていない ODBC ドライバもあります。追加項目が生じる ODBC ドライバもあります。

SQLTables

sqltables ステートメントは、**connect** が実行されている ODBC または OLE DB データソースのテーブルを説明する項目をセットで返します。

構文:

```
SQLTables
```

この項目を **sqlcolumns** および **sqltypes** コマンドで生成される項目と組み合わせると、データベースの把握に役立ちます。5 つの標準項目は次のとおりです。

TABLE_QUALIFIER

TABLE_OWNER

TABLE_NAME

TABLE_TYPE

REMARKS

これらの項目の詳細については、ODBC リファレンスを参照してください。

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

このコマンドをサポートしていない ODBC ドライバもあります。追加項目が生じる ODBC ドライバもあります。

SQLTypes

sqltypes ステートメントは、**connect** が実行される ODBC または OLE DB データソースの種類を記述する項目セットを返します。

構文:
SQLTypes

この項目を **sqlcolumns** および **sqltables** コマンドで生成される項目と組み合わせると、データベースの把握に役立ちます。15 個の標準項目は次のとおりです。

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

これらの項目の詳細については、ODBC リファレンスを参照してください。

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

このコマンドをサポートしていない ODBC ドライバもあります。追加項目が生じる ODBC ドライバもあります。

Star

star ステートメントを使用すると、データベースの項目すべての値セットを表す文字列を設定できます。これは、後続の **LOAD** および **SELECT** ステートメントに影響を与えます。

構文:

```
Star is[ string ]
```

引数:

引数

引数	説明
string	任意のテキスト。文字列に空白がある場合は、引用符で囲む必要があります。 何も指定されていない場合は、**star is;** と見なされ、明示的に指定されている場合以外はスター マークを使用できません。この定義は、新たな **star** ステートメントが作成されるまで有効となります。

セクション アクセスを使用している場合、スクリプトのデータ部分 (**Section Application** の下) では **Star is** ステートメントは使用しないよう推奨します。ただし、スクリプトの **Section Access** 部分の保護項目では、スター文字に完全対応しています。この場合、**Star is** ステートメントはセクション アクセスでは常に暗黙的であるため、明示的な Star is ステートメントを使用する必要はありません。

制限事項

- キー項目、つまり、テーブルをリンクする項目でスター文字を使用することはできません。
- **Unqualify** ステートメントの影響を受ける項目では、テーブルをリンクする項目に影響を与える可能性があるため、スター文字を使用することはできません。
- 情報ロード テーブルやマッピング ロード テーブルなどの非論理テーブルではスター文字を使用できません。
- セクション アクセスの縮小項目 (データにリンクする項目) でスター文字が使用されている場合、セクション アクセスのこの項目に表示されている値を表します。データに存在する可能性はあるが、セクション アクセスに表示されていない他の値は表しません。
- **Section Access** 領域外で何らかのデータ削減の影響を受ける項目では、スター文字を使用できません。

例

以下の例は、セクション アクセスを扱うデータ ロード スクリプトを抽出したものです。

```
Star is *;



Section Access;

LOAD * INLINE [

ACCESS, USERID, OMIT

ADMIN, ADMIN,

USER, USER1, SALES
```

```
USER, USER2, WAREHOUSE

USER, USER3, EMPLOYEES

USER, USER4, SALES

USER, USER4, WAREHOUSE

USER, USER5, *

];


Section Application;

LOAD * INLINE [

SALES, WAREHOUSE, EMPLOYEES, ORDERS

1, 2, 3, 4

];
```

次の事項が適用されます。

- *Star* 記号は * です。
- ユーザー *ADMIN* にはすべての項目が表示されます。何も省略されていません。
- ユーザー *USER1* には、項目 *SALES* は表示されません。
- ユーザー *USER2* には、項目 *WAREHOUSE* は表示されません。
- ユーザー *USER3* には、項目 *EMPLOYEES* は表示されません。
- ユーザー *USER4* が、*SALES* および *WAREHOUSE* という、このユーザーの OMIT の 2 つの項目のソリューションに 2 回追加されます。
- *USER5* には「*」が追加されています。これは、OMIT にリストされているすべての項目が使用できないことを意味します。つまり、ユーザー *USER5* は項目 *SALES*、*WAREHOUSE*、*EMPLOYEES* を表示できませんが、このユーザーは項目 *ORDERS* を表示できます。

Store

Store ステートメントは、QVD、Parquet、CSV、または TXT ファイルを作成します。

構文:

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

このステートメントにより、明示的に命名された QVD、Parquet、またはテキストファイルが作成されます。

Parquet に保存する場合を除き、ステートメントは 1 つのデータテーブルからのみ項目をエクスポートできます。複数のテーブルの項目を QVD、CSV、または TXT ファイルにエクスポートする場合は、エクスポートするデータテーブルを作成するスクリプトで事前に明示的な join を作成する必要があります。Parquet ファイルにデータをネストすることで、複数のテーブルを 1 つの Parquet に保存できます。

テキスト値は UTF-8 形式で CSV ファイルにエクスポートされます。区切り文字を指定できます (**LOAD** を参照)。CSV ファイルへの **store** ステートメントでは、BIFF エクスポートを行えません。

データが整形式ではない場合、データが正しく解釈されるように、項目が二重引用符で囲まれる場合があります。これは、項目に引用符、カンマ、スペース、改行などの文字が含まれている場合などに発生する可能性があります。

引数:

コマンド引数の保存

引数	説明
*fieldlist::= (* | field) { , field })*	選択する項目のリスト。項目リストとして * を使用すると、すべての項目が指定されます。 *field::= fieldname [***as** *aliasname]* *fieldname* は、*table* の項目名と同じテキストです。(項目名にスペースや非標準的な文字などが含まれる場合、ストレート二重引用符または角括弧で囲む必要があります。) *aliasname* は、結果の QVD ファイルまたは CSV ファイルで使用される項目の別名です。
table	データソースとして使用するロード済みテーブルを表すスクリプトラベル。

引数	説明
filename	既存のフォルダデータ接続への有効なパスを含むターゲットファイルの名前。 ***'lib://Table Files/target.qvd'*** レガシースクリプトモードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data\sales.qvd*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data\sales.qvd*** パスを省略すると、Qlik Sense は **Directory** ステートメントで指定されたディレクトリにファイルを保存します。**Directory** ステートメントがない場合、Qlik Sense は作業ディレクトリ*C:\Users\{user}\Documents\Qlik\Sense\Apps* にファイルを保存します。 •

引数	説明
format-spec ::=((**txt** | **qvd** | **parquet**), **compression is** *codec*)	これらのファイル形式のいずれかに書式指定を設定できます。書式指定が省略されている場合は、**qvd** で処理されます。 • CSV および TXT ファイルの **txt**。 • QVD ファイルの **qvd**。 • Parquet ファイルの **parquet**。 **parquet** を使用する場合、[**圧縮は**] で使用する圧縮コーデックを設定することもできます。[**圧縮は**] で圧縮コーデックを指定しない場合、snappy が使用されます。次の圧縮設定を使用できます。 • uncompressed • snappy • gzip • lz4 • brotli • zstd • lz4_hadoop 例: `Store mytable into [lib://AttachedFiles/myfile.parquet] (parquet, compression is lz4);`

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

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

Store Name, RegNo from mytable into xyz.qvd;

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

Store mytable into myfile.txt (txt);

Store mytable into myfile.parquet (parquet);

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

Parquet ファイルでの保存

Parquet は強く型付けされたファイル フォーマットで、各フィールドには特定のデータ型 (in32、ダブル、日付と時刻、テキストなど) が 1 つずつ保存されます。Qlik Sense は、内部データを緩く型付けされたデュアルとして保存し、異なるソースからのデータを同じフィールドに混在させることができます。Parquet の各項目にはデュアルの一

部分しか保存できないため、各項目に何が含まれているかを知ることが重要です。既定では、Qlik Sense は項目タイプを使用して項目の保存方法を決定します。特定のフォーマットで Parquet ファイルにデータを保存する場合、ロード時に項目のデータ型を指定する必要があります。テキスト項目に数字、タイムスタンプ項目にテキストなど、Parquet ファイルの互換性のない項目にデータを保存しようとすると、NULL 値になってしまいます。

Parquet に保存するデータをロードするときに、既定の動作を変更できます。データ型を変更するためにフォーマットするか、Parquet で特定の列タイプを強制するためにタグ付けできます。

Parquet に保存するデータのフォーマット

Qlik Sense フォーマット関数を使用してデータを分類できます。たとえば、**Text()**、**Num()**、**Interval()** または **Timestamp()** は Parquet でデータを保存する際、データフォーマットを強制できます。Qlik Sense は、項目の属性と自動項目タグによって、ほぼ 20 種類のデータ型にデータを保存できます。詳細については、「*変換関数 (page 1241)*」を参照してください。

Num() と Text() でのデータのフォーマット

次の例は、Parquet に保存するデータを準備する例です。**Num()** は数値項目に適用されます。**Text()** はテキストと混合の両方に適用されます。混合の場合、**Text()** は Parquet で数値項目のように扱われ、テキスト値が NULL 値に変更されるのを防ぎます。

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

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

Parquet に保存するためのデータのタグ付け

Parquet にデータを保存する際、特定の列タイプを強制的に指定する `$parquet` タグをデータに付けます。各データ型は、対応するコントロール タグを追加することで強制できます。例えば、項目を INT32 としてParquet に保存するには、ロード スクリプトで `$parquet-int32` のタグを付けます。データ型によって、文字列か数値表現のどちらかのデュアル データが保存されます。

以下の Parqeut コントロール タグは、Parquet ファイルに保存する項目のタグ付けに使用できます。

Parquet コントロール タグ

コントロール タグ	デュアル	物理型	論理型	変換型
`$parquet-boolean`	数値	BOOLEAN	NONE	NONE
`$parquet-int32`	数値	INT32	NONE	NONE
`$parquet-int64`	数値	INT64	NONE	NONE

コントロール タグ	デュアル	物理型	論理型	変換型
$parquet-float	数値	FLOAT	NONE	NONE
$parquet-double	数値	DOUBLE	NONE	NONE
$parquet-bytearray	文字列	BYTE_ARRAY	NONE	UTF8
$parquet-bytearrayfix	数値	FIXED_LEN_BYTE_ARRAY	NONE	DECIMAL
$parquet-decimal	数値	INT64	DECIMAL	DECIMAL
$parquet-date	数値	INT32	DATE	DATE
$parquet-time	数値	INT64	TIME	TIME_MICROS
$parquet-timestamp	数値	INT64	TIMESTAMP	TIMESTAMP_MICROS
$parquet-string	文字列	BYTE_ARRAY	STRING	UTF8
$parquet-enum	文字列	BYTE_ARRAY	ENUM	ENUM
$parquet-interval	数値	FIXED_LEN_BYTE_ARRAY	INTERVAL	INTERVAL
$parquet-json	文字列	BYTE_ARRAY	JSON	JSON
$parquet-bson	文字列	BYTE_ARRAY	BSON	BSON
$parquet-uuid	文字列	FIXED_LEN_BYTE_ARRAY	UUID	NONE

Parquet に保存するためのデータのタグ付け

この例では、2 つのタグを使って Parquet 用のデータを定義しています。フィールド *num* は `$parquet-int32` とタグ付けされ、Parquet では INT32 として設定される数値フィールドとして定義されます。

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

ネストされたデータを Parquet ファイルに保存する

複数のテーブルを構造化データにネストすることで、Parquet ファイルに保存できます。**Store** は、スター スキーマの構造化 *ノード*とリスト *ノード*をサポートします。[**Delimiter is**] 指定子を使用すると、単一テーブルをネスト モードで保存することもできます。

テーブルを保存する場合は、含めるテーブルをカンマで区切って指定します。例: `STORE Table1, Table2, Table3 INTO [lib://<file location>/<file name>.parquet] (parquet);`。**Store** ステートメントの項目 リストを使用して、保存する項目を制御できます。例: `STORE Field1, Field2, FROM Table1, Table2 INTO [lib://<file location>/<file name>.parquet] (parquet);` 。項目 リスト内のすべての項目は、リストされている1つ以上のテーブルに存在する必要があります。**Store** ステートメントにある最初のテーブルは、スター スキーマのファクト テーブルとして使用されます。

項目名は、グループを作成してネストする方法を制御するために使用されます。既定では、項目名はピリオド (.) を使用して*ノード*に分割されます。区切り記号は、システム変数 *FieldNameDelimiter* を設定するか、指定子 [**Delimiter is**] を使用して変更できます。指定子はシステム変数を上書きします。

項目名は区切り記号によって分割され、それらの部分はネストされたグループを使用したスキーマを作成するために使用されます。例えば、`STORE Field1, Field1.Field2, Field1.Field3, Field1.Field4 FROM Table1 INTO [nested.parquet] (parquet, delimiter is '.');` により2つのグループ (*Group1* と *Group2*) が、*Fields1*、*Field2* そして *Field3*、*Field4* を使用して作成されます。

> *グループと項目は、スキーマ内のノードで同一名になるとは限りません。例えば、`STORE Address, Address.Street INTO [nested.parquet] (parquet, delimiter is '.');` は失敗します。Address は曖昧であり、両方ともデータ項目とグループであるためです。*

ネストされたデータを Parquet に保存すると、テーブル間のキーがスキーマのリンク *ノード*に変換されます。テーブルはスキーマの構造化 *ノード*に変換されます。項目名を使用して、既定の変換を上書きできます。

ネストされたデータを Parquet ファイルに保存する

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

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

headquarter:
LOAD * INLINE [
```

```
company, country, city
A&G, USA, Los Angeles
Cabro, USA, Albuquerque
Fenwick, USA, Baltimore
Camros, USA, Omaha
];

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

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

結果の Parquet ファイルには次のスキーマが含まれます。

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

制限事項

ネストされたデータを Parquet に保存するには、次の制限があります。

- ストアはマップ ノードをサポートしません。
- 保存には、ネストされた Parquet ファイルのロードによって生成されるキー項目は含まれません。
- キー項目にリンクされていないテーブルのデータを一緒に保存することはできません。
- ネストされたファイルはデータ モデルを非正規化します。参照されていない値は保存されず、複数回参照された値はコピーされます。

Table/Tables

Table および **Tables** スクリプト キーワードは、**Load** ステートメントの書式指定子、そして **Drop**、**Comment**、**Rename** ステートメントで使用されます。

Tag

このスクリプト ステートメントは、1 つ以上の項目またはテーブルにタグを割り当てる方法を提供します。アプリにない項目またはテーブルにタグを付けしようとしても、無視されます。項目名やタグ名の競合が発生する場合は、最後の値が使用されます。

構文:

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

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

```
Tag table tablelist with tagname
```

引数

引数	説明
fieldlist	カンマ区切りのリストで、タグ付けする必要がある 1 つまたは複数の項目。
mapname	**mapping Load** または **mapping Select** ステートメントで、以前ロードされたマッピング テーブルの名前。
tablelist	タグ付けする必要のあるテーブルのコンマ区切りのリスト。
tagname	項目に適用するタグの名前。

Example 1:

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

Example 2:

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

Trace

trace ステートメントを使用すると、[**ロード スクリプトの進捗**] ウィンドウとスクリプトのログ ファイルに使用した文字列が書き込まれます。これはデバッグの際に非常に有用です。**trace** ステートメントの前に計算される変数の $ 拡張を使用すると、メッセージをカスタマイズできます。

構文:

```
Trace string
```

Example 1:

次のステートメントは、「メイン」テーブルをロードする Load ステートメントの直後に使用できます。

```
Trace Main table loaded;
```
これにより、スクリプト実行ダイアログとログ ファイルに「メイン テーブルがロードされました」というテキストが表示されます。

Example 2:

次のステートメントは、「メイン」テーブルをロードするLoad ステートメントの直後に使用できます。

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

Trace $(MyMessage);
```
これにより、スクリプト実行ダイアログとログ ファイルの行数を示すテキストが表示されます (例:「メイン テーブルの 265,391 行」)。

Unmap

Unmap ステートメントは、前に **Map ... Using** ステートメントによって指定した項目値の、後続のロードされた項目のマッピングを無効にします。

構文:

```
Unmap *fieldlist
```

引数:

引数

引数	説明
*fieldlist	スクリプト内でマッピングの終了点に指定する項目のカンマ区切りリスト。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。

例と結果:

例	結果
Unmap Country;	項目 Country のマッピングを無効にします。
Unmap A, B, C;	項目 A、B、C のマッピングを無効にします。
Unmap * ;	すべての項目のマッピングを無効にします。

Unqualify

Unqualify ステートメントは、**Qualify** ステートメントで事前に有効化された項目名の修飾を無効にする際に使用します。

構文:

```
Unqualify *fieldlist
```

引数:

引数

引数	説明
*fieldlist	修飾を有効にする項目のコンマ区切りリスト。項目リストに * を使用すると、すべての項目が対象となります。項目名にはワイルドカード文字の * および ? を使用できます。ワイルドカード文字を使用する際には、項目名を引用符で囲まなければならない場合があります。 詳細については、**Qualify** ステートメントの説明を参照してください。

Example 1:

例に挙げているように、馴染みのないデータベースについては、1 つまたは少数の項目の関連付けから始めることをお勧めします。

```
qualify *;
unqualify TransID;
SQL SELECT * from tab1;
SQL SELECT * from tab2;
SQL SELECT * from tab3;
```

まず、すべての項目で修飾がオンになります。
次に、**TransID** の修飾がオフになります。
テーブル *tab1*、*tab2*、*tab3* の関連付けには、**TransID** のみが使用されます。他のすべての項目は、テーブル名で修飾されます。

Untag

このスクリプトステートメントは、項目またはテーブルからタグを削除する方法を提供します。アプリにない項目またはテーブルのタグを外そうとしても、無視されます。

構文:

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

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

```
Untag table tablelist with tagname
```

引数:

引数

引数	説明
fieldlist	カンマ区切りのリストで、タグを削除する必要がある 1 つまたは複数の項目。
mapname	マッピング **LOAD** またはマッピング **SELECT** ステートメントで以前ロードされたマッピングテーブルの名前。

引数	説明
tablelist	タグを外す必要のあるテーブルのコンマ区切りのリスト。
tagname	項目から削除するタグの名前。

Example 1:

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

Example 2:

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

3.4 作業ディレクトリ

スクリプトステートメントでファイルを参照するときに、パスが省略されている場合、Qlik Sense は次の順序でファイルを検索します。

1. **Directory** ステートメントによって指定されたディレクトリ(レガシー スクリプト モードでのみサポートされます)。
2. **Directory** ステートメントがない場合、Qlik Sense は作業ディレクトリを検索します。

Qlik Sense Desktop作業ディレクトリ

Qlik Sense Desktop では、作業ディレクトリは *C:\Users\{user}\Documents\Qlik\Sense\Apps* です。

Qlik Sense作業ディレクトリ

Qlik Sense の場合、作業ディレクトリは Qlik Sense Repository Service で指定されています。これはデフォルトで、*C:\ProgramData\Qlik\Sense\Apps* です。詳細については、Qlik 管理コンソール ヘルプを参照してください。

4 データロードエディタでの変数の使用

Qlik Senseの変数は、数値や英数字などの静的な値または計算を格納するコンテナです。アプリで変数を使用する場合、変数の値を変えると、その変数が使用されているすべての箇所に変更が反映されます。変数は、変数の概要で定義するか、データロードエディターを使用してスクリプト内に定義します。データロードスクリプトで **Let** または **Set** ステートメントを使用して、変数の値を設定します。

シートの編集時には、変数の概要にある Qlik Sense 変数を使用して作業することもできます。

4.1 概要

変数値の最初の文字が等記号 (=) の場合、Qlik Sense は値を式 (Qlik Sense 式) として評価し、式の実際のテキストではなく結果を表示または返します。

変数を使用すると、変数の代わりにその値が使用されます。変数はドル記号展開用のスクリプトやさまざまな制御ステートメントで使用できます。例えば、パスのように、同じ文字列がスクリプト内に何度も繰り返し出てくる場合に便利です。

一部の特別なシステム変数は、スクリプトの実行開始時に、以前の値とは関係なくQlik Sense によって設定されます。

4.2 変数の定義

変数は、静的な値または計算の結果を格納する機能を提供します。変数を定義するときは、次の構文を使用してください。

```
set variablename = string
```

停止

```
let variable = expression
```

Set ステートメントは文字列の割り当てに使用されます。等号の右側のテキストを変数に割り当てます。**Let** ステートメントは、スクリプトの実行時に等号の右側にある数式を評価し、数式の結果を変数に割り当てます。

変数では、大文字と小文字が区別されます。

Qlik Senseでは、変数に項目や関数と同じ名前を付けることは推奨されていません。

```
set x = 3 + 4; // 変数は、値として文字列「3 + 4」を取得します。

let x = 3 + 4; // は値として 7 を返します。
```

```
set x = Today(); // は値として「Today()」を返します。

let x = Today(); // は今日の日付を値として返します (例:「9/27/2021」)。
```

4.3 変数の削除

スクリプトから変数を削除してデータをリロードすると、変数はそのままアプリに残ります。アプリから完全に変数を削除するには、変数ダイアログからも変数を削除する必要があります。

4.4 項目値としての変数値のロード

変数値を**LOAD** ステートメントで項目値としてロードし、ドル展開の結果が数字や数式ではなくテキストの場合、展開した変数を1つの引用に含める必要があります。

この例では、スクリプトエラーの一覧を含むシステム変数をテーブルにロードします。**If** 句での ScriptErrorCount の展開に引用符は不要ですが、ScriptErrorList の展開には引用符が必要になることに注意してください。

```
IF $(ScriptErrorCount) >= 1 THEN

   LOAD '$(ScriptErrorList)' AS Error AutoGenerate 1;
END IF
```

4.5 変数の計算

Qlik Sense で計算済みの値を使って変数を使用する方法はいくつかあり、その結果は、これを定義する方法と数式で呼び出す方法によって異なります。

この例では、いくつかのインライン データをロードします。

```
LOAD * INLINE [
    Dim, Sales
    A, 150
    A, 200
    B, 240
    B, 230
    C, 410
    C, 330
];
```

2つの変数を定義してみましょう。

```
Let vSales  = 'Sum(Sales)' ;
Let vSales2  = '=Sum(Sales)' ;
```

2番目の変数では、数式の前に等号を追加します。これにより、変数が展開される前に計算され、数式が評価されます。

vSales 変数をそのまま使用する場合 (メジャーで使用する場合など)、その結果は文字列Sum(Sales) になります。つまり、計算は行われません。

ドル記号展開を追加して数式で $(vSales) を呼び出すと、変数が展開され、Sales の合計が表示されます。

最後に、$(vSales2) を呼び出すと、変数は展開される前に計算されます。つまり、表示される結果は Sales の合計です。メジャー数式として =$(vSales) を使用した場合と=$(vSales2) を使用した場合に結果がどう違うかを、下記の表に示します。

結果

Dim	$(vSales)	$(vSales2)
A	350	1560
B	470	1560
C	740	1560

ご覧のように、$(vSales) は軸値の小計になり、$(vSales2) は合計になります。

次のスクリプト変数を使用できます。

- *エラー変数 (page 274)*
- *データ型変換変数 (page 212)*
- *システム変数 (page 204)*
- *値を操作する変数 (page 210)*

4.6 システム変数

システム変数は、システムで定義されているものもあり、システムとQlik Sense アプリに関する情報を提供します。

システム変数の概要

関数の中には、概要の後に詳細が示されているものもあります。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

CreateSearchIndexOnReload

この変数は、データのリロードの間に検索インデックス ファイルが作成されるかどうかを定義します。

```
CreateSearchIndexOnReload
```

Floppy

見つかった最初のフロッピー ドライブのドライブ文字を返します。通常は *a:* です。これはシステム定義変数です。

```
Floppy
```

この変数は標準モードに対応していません。

CD

見つかった最初の CD-ROM ドライブのドライブ文字を返します。CD- ROM が見つからない場合は、*c:* が返されます。これはシステム定義変数です。

```
CD
```

この変数は標準モードに対応していません。

HidePrefix

このテキスト文字列で始まる項目名はすべて、システム項目と同様に非表示になります。これはユーザー定義変数です。

```
HidePrefix
```

HideSuffix

このテキスト文字列で終わる項目名はすべて、システム項目と同様に非表示になります。これはユーザー定義変数です。

```
HideSuffix
```

Include

Include/Must_Include 変数は、スクリプトにインクルードしてスクリプトコードとして評価する必要があるテキストが格納されたファイルを指定します。データの追加には使用されません。スクリプトコードの一部を別のテキストファイルに保存して、複数のアプリで再利用することができます。これはユーザー定義変数です。

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

OpenUrlTimeout

この変数は、URL ソース (ページなど) からデータを取得する際に、Qlik Sense が考慮すべきタイムアウトを秒単位でHTML 定義します。省略した場合、約 20 分でタイムアウトになります。

```
OpenUrlTimeout
```

QvPath

Qlik Sense 実行可能ファイルへの参照文字列を返します。これはシステム定義変数です。

```
QvPath
```

この変数は標準モードに対応していません。

QvRoot

Qlik Sense 実行可能ファイルのルートディレクトリを返します。これはシステム定義変数です。

```
QvRoot
```

この変数は標準モードに対応していません。

QvWorkPath

現在の Qlik Sense アプリへの参照文字列を返します。これはシステム定義変数です。

```
QvWorkPath
```

この変数は標準モードに対応していません。

QvWorkRoot

現在の Qlik Sense アプリのルートディレクトリを返します。これはシステム定義変数です。

```
QvWorkRoot
```

この変数は標準モードに対応していません。

StripComments

この変数を 0 に設定すると、スクリプト内の /*..*/ および // コメントの除去は禁止されます。この変数が定義されていない場合、コメントの除去は常に実行されます。

```
StripComments
```

Verbatim

通常、Qlik Sense データベースにロードする前に、すべての項目値から前後の空白文字 (ASCII 32) が自動的に除去されます。この変数を 1 に設定すると、空白の除去が一時停止されます。タブ (ASCII 9) 文字やハードスペース (ANSI 160) 文字が除去されることはありません。

```
Verbatim
```

WinPath

Windows への参照文字列を返します。これはシステム定義変数です。

```
WinPath
```

この変数は標準モードに対応していません。

WinRoot

Windows のルートディレクトリを返します。これはシステム定義変数です。

```
WinRoot
```

この変数は標準モードに対応していません。

CollationLocale

ソート順序および検索一致で使用するロケールを指定します。値は、ロケールのカルチャ名 ('en-US' など) になります。これはシステム定義変数です。

```
CollationLocale
```

CreateSearchIndexOnReload

この変数は、データのリロードの間に検索インデックス ファイルが作成されるかどうかを定義します。

構文:

```
CreateSearchIndexOnReload
```

データのリロードの間、またはユーザーによる最初の検索リクエストの後に、検索インデックス ファイルが作成されるよう定義できます。データのリロードの間に検索インデックス ファイルを作成する利点は、ユーザーによる最初の検索リクエストの待ち時間を回避できることです。これは、検索インデックスの作成によりデータのリロード時間が長くなるという点を考慮する必要があります。

この変数が省略されると、データのリロードの間に検索インデックス ファイルは作成されません。

セッション アプリでは、この変数の設定にかかわらずデータのリロードの間に検索インデックス ファイルは作成されません。

Example 1: データのリロードの間に検索インデックス項目を作成します。

```
set CreateSearchIndexOnReload=1;
```

Example 2: 最初の検索リクエストの後で検索インデックス項目を作成します。

```
set CreateSearchIndexOnReload=0;
```

HidePrefix

このテキスト文字列で始まる項目名はすべて、システム項目と同様に非表示になります。これはユーザー定義変数です。

構文:

```
HidePrefix
```

```
set HidePrefix='_' ;
```

このステートメントを使用すると、システム項目が非表示の場合、アンダースコア (_) から始まる項目名が項目名リストに表示されません。

HideSuffix

このテキスト文字列で終わる項目名はすべて、システム項目と同様に非表示になります。これはユーザー定義変数です。

構文:

```
HideSuffix
```

```
set HideSuffix='%';
```

このステートメントを使用すると、システム項目が非表示の場合、パーセント記号 (%) で終わる項目名が項目名リストに表示されません。

Include

Include/Must_Include 変数は、スクリプトにインクルードしてスクリプトコードとして評価する必要があるテキストが格納されたファイルを指定します。データの追加には使用されません。スクリプトコードの一部を別のテキストファイルに保存して、複数のアプリで再利用することができます。これはユーザー定義変数です。

この変数は、標準モードのフォルダデータ接続のみをサポートします。

構文:

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

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

この変数には、次の 2 つのバージョンがあります。

- **Include** は、ファイルが見つからない場合にエラーを生成しません。失敗した場合、何もしません。
- **Must_Include** は、ファイルが見つからない場合にエラーを生成します。

パスを指定していない場合は、ファイル名は Qlik Sense アプリの作業ディレクトリの相対パス名になります。絶対ファイルパス、または lib:// フォルダ接続へのパスも指定できます。等号記号の前後に空白文字を配置しないでください。

*構文 **set Include =**filename は適用できません。*

```
$(Include=abc.txt);
```

```
$(Must_Include=lib://DataFiles/abc.txt);
```

制限事項

Linux と比べて Windows では、UTF-8 エンコードファイルの相互互換性が限定されます。

UTF-8 とBOM (バイトオーダーマーク) の併用はオプションです。ファイルの始めに ASCII 以外のバイトを想定していないソフトウェア内で BOM とUTF-8 を使用すると干渉する場合がありますが、それ以外ではテキストストリームを処理できます。

- Windows システムでは、バイトストレージでのあいまいさがないにもかかわらず、UTF-8 内で BOM を使用してファイルが UTF-8 でエンコードされていることを識別します。
- Unix/Linux では Unicode に UTF-8 を使用しますが、コマンドファイルの構文と干渉するため BOM は使用しません。

このため、Qlik Sense にいくつかの影響が生じます。

- Windows では、UTF-8 BOM で始まるファイルはいずれも UTF-8 スクリプトファイルと見なされます。それ以外では、ANSI エンコードと見なされます。
- Linux では、システムの既定 8 ビットコードページが UTF-8 です。BOM が含まれていなくても UTF-8 が機能するのはこのためです。

この結果、移植性を保証することができません。Linux で解読できるファイルを Windows で作成 (あるいはその逆) できるとは限りません。BOM の処理方法が異なるため、これら 2 つのシステム間には、UTF-8 エンコードのファイルに関する相互互換性がありません。

OpenUrlTimeout

この変数は、URL ソース (ページなど) からデータを取得する際に、Qlik Sense が考慮すべきタイムアウトを秒単位でHTML 定義します。省略した場合、約 20 分でタイムアウトになります。

構文:

```
OpenUrlTimeout
```

```
set OpenUrlTimeout=10;
```

StripComments

この変数を 0 に設定すると、スクリプト内の /*..*/ および // コメントの除去は禁止されます。この変数が定義されていない場合、コメントの除去は常に実行されます。

構文:

```
StripComments
```

特定のデータベースのドライバは、/*..*/ を **SELECT** ステートメントにおける最適化のヒントとして使用します。この場合、**SELECT** ステートメントをデータベースのドライバに送信する前に、コメントを削除することはできません。

コメントが必要なステートメントの直後に、この変数を 1 にリセットすることをお勧めします。

```
set StripComments=0;
SQL SELECT * /* <optimization directive> */ FROM Table ;
set StripComments=1;
```

Verbatim

通常、Qlik Sense データベースにロードする前に、すべての項目値から前後の空白文字 (ASCII 32) が自動的に除去されます。この変数を1に設定すると、空白の除去が一時停止されます。タブ (ASCII 9) 文字やハードスペース (ANSI 160) 文字が除去されることはありません。

構文:

```
Verbatim
```

```
set Verbatim = 1;
```

4.7 値を操作する変数

このセクションでは、NULL とその他の値を処理する際に使用する変数について説明します。

値を操作する変数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

NullDisplay

定義済みの記号は、最下位レベルのデータで ODBCから取得されたすべての NULL値とコネクタの代わりに使用されます。これはユーザー定義変数です。

```
NullDisplay
```

NullInterpret

ここに定義される記号が、テキストファイルや Excel、インライン ステートメントに表示される場合に NULL として解釈されます。これはユーザー定義変数です。

```
NullInterpret
```

NullValue

NullAsValue ステートメントを使用する場合は、特定の文字列を含む **NullAsValue** の指定項目内のすべての NULL 値の代わりに定義した記号が使用されます。

```
NullValue
```

OtherSymbol

LOAD/SELECT ステートメントの前にある「他のすべての値」として処理される記号を定義します。これはユーザー定義変数です。

```
OtherSymbol
```

NullDisplay

定義済みの記号は、最下位レベルのデータでODBCから取得されたすべてのNULL値とコネクタの代わりに使用されます。これはユーザー定義変数です。

構文:

```
NullDisplay
```

```
set NullDisplay='<NULL>';
```

NullInterpret

ここに定義される記号が、テキストファイルやExcel、インライン ステートメントに表示される場合にNULL として解釈されます。これはユーザー定義変数です。

構文:

```
NullInterpret
```

```
set NullInterpret=' ';
set NullInterpret =;
```

これはExcelの空白値に対してNULL値を返しません(CSVテキストファイルには返します)。

```
set NullInterpret ='';
```

これはExcelの空白値に対してNULL値を返します。

NullValue

NullAsValue ステートメントを使用する場合は、特定の文字列を含む **NullAsValue** の指定項目内のすべてのNULL値の代わりに定義した記号が使用されます。

構文:

```
NullValue
```

```
NullAsValue Field1, Field2;
set NullValue='<NULL>';
```

OtherSymbol

LOAD/SELECT ステートメントの前にある「他のすべての値」として処理される記号を定義します。これはユーザー定義変数です。

構文:

```
OtherSymbol
```

```
set OtherSymbol='+';
LOAD * inline
[X, Y
a, a
b, b];
LOAD * inline
[X, Z
a, a
+, c];
```

項目値 Y='b' は、他の記号を通じて Z='c' とリンクします。

4.8 データ型変換変数

データ型変換変数はシステムによって定義されます。変数はロードスクリプトの最上部に含まれており、スクリプトを実行する際に数値の書式設定を適用します。データ型変換変数は、削除や編集、複製が可能です。

データ型変換変数は、新しいアプリが作成されるときに、オペレーティング システムの現在の地域設定に従って自動的に生成されます。Qlik Sense Desktop では、コンピューターのオペレーティング システムの設定に従います。Qlik Sense では、Qlik Sense がインストールされているサーバーのオペレーティング システムに従います。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディタは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

通貨書式

MoneyDecimalSep

定義した小数点記号が地域設定によって設定された通貨の小数点記号の代わりに使用されます。

```
MoneyDecimalSep
```

MoneyFormat

定義した記号が地域設定によって設定された通貨記号の代わりに使用されます。

```
MoneyFormat
```

MoneyThousandSep

定義した桁区切り記号が地域設定によって設定された通貨の桁区切り記号の代わりに使用されます。

```
MoneyThousandSep
```

数値書式

DecimalSep

定義した小数点記号が地域設定によって設定された小数点記号の代わりに使用されます。

```
DecimalSep
```

ThousandSep
定義した桁区切り記号がオペレーティング システム (地域設定) の桁区切り記号の代わりに使用されます。

```
ThousandSep
```

NumericalAbbreviation
数値の省略形を使用して、数字のスケール プレフィックスに使用する省略形を設定します。例えば、メガや 100 万 (106) には M、^{6}マイクロ (10-6) には μ^{-6}).

```
NumericalAbbreviation
```

時間書式

DateFormat
この環境変数は、アプリで既定として使用される日付書式を定義します。この書式は、日付の解釈と書式化の両方に使用されます。この変数が定義されていない場合、スクリプトの実行時にオペレーティング システムの地域設定の日付書式が取得されます。

```
DateFormat
```

TimeFormat
定義した書式がオペレーティング システム (地域設定) の時刻書式の代わりに使用されます。

```
TimeFormat
```

TimestampFormat
定義した書式がオペレーティング システム (地域設定) の日付と時刻の書式の代わりに使用されます。

```
TimestampFormat
```

MonthNames
定義した書式が地域設定の月名の表記規則の代わりに使用されます。

```
MonthNames
```

LongMonthNames
定義した書式が地域設定の長い月名の表記規則の代わりに使用されます。

```
LongMonthNames
```

DayNames
定義した書式が地域設定によって設定された曜日表記規則の代わりに使用されます。

```
DayNames
```

LongDayNames
定義した書式が地域設定の長い曜日名の表記規則の代わりに使用されます。

```
LongDayNames
```

FirstWeekDay
週の最初として使用する曜日を定義する整数です。

```
FirstWeekDay
```

BrokenWeeks
この設定は、週が分離しているかどうかを定義します。

```
BrokenWeeks
```

ReferenceDay
この設定は、週 1 を定義する際に 1 月 のどの曜日を参照に設定するかを定義します。

```
ReferenceDay
```

FirstMonthOfYear
この設定は、年の最初の月をどれにするかを定義し、毎月のオフセットを使用する財務年度を定義するために使用できます (例: 4 月 1 日から開始するなど)。

> *この設定は現在使用されていませんが、将来使用する目的で予約されています。*

有効有効な設定は、1 (1 月) から 12 (12 月) です。デフォルト設定は 1 です。

構文:

```
FirstMonthOfYear
```

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

BrokenWeeks

この設定は、週が分離しているかどうかを定義します。

構文:

```
BrokenWeeks
```

Qlik Sense では、アプリ作成時に地域設定がフェッチされ、対応する設定は環境変数としてスクリプトに保管されます。

米国のアプリ開発者は、分離した週に対応して、スクリプトで `Set BrokenWeeks=1;` をよく取得します。ヨーロッパのアプリ開発者は、未分離の週に対応して、スクリプトで `Set BrokenWeeks=0;` をよく取得します。

未分離の週とは次の意味です:

- 第 1 週が 12 月から始まる年もあれば、前年度の最終週が 1 月に渡る年もあります。
- ISO 8601 によると、通常第 1 週には、少なくとも 1 月の 4 日間が含まれます。Qlik Sense では、これは `ReferenceDay` 変数を使って構成できます。

分離された週とは次の意味です:

- 年度の最終週が1月に渡ることはありません。
- 第1週は1月1日から始まり、多くの場合は完全な1週間ではありません。

次の値を使用できます。

- 0 (= 分離しない週を使用)
- 1 (= 分離した週を使用)

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

週数と週番号の ISO 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

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

US 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

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

DateFormat

この環境変数は、アプリで既定値として使用される日付形式と、date() や date#() などの日付を返す関数を定義します。この形式は、日付の解釈と書式設定に使用されます。この変数が定義されていない場合、スクリプトの実行時に元の設定の日付形式が取得されます。

構文:

DateFormat

DateFormat 関数の例

例	結果
Set DateFormat='M/D/YY'; //(US format)	この DateFormat 関数の使用は、日付を米国形式で (月/日/年) として定義します。
Set DateFormat='DD/MM/YY'; //(UK date format)	この DateFormat 関数の使用は、日付を英国形式で (日/月/年) として定義します。

例	結果
`Set DateFormat='YYYY/MM/DD'; //(ISO date format)`	この DateFormat 関数の使用は、日付をISO 形式で (年/月/日) として定義します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – システム変数の既定

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 日付のデータセット。
- 米国の日付形式を使用する DateFormat 関数。

この例では、データセットが「Transactions」というテーブルにロードされます。これには、date 項目が含まれます。米国の DateFormat 定義が使用されます。このパターンは、日付をロードするときに、暗黙的なテキストから日付に変換するために使用されます。

ロードスクリプト

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

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- month

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

日付	月	=sum(amount)
01/01/2022	Jan	1000
02/01/2022	Feb	2123
03/01/2022	Mar	4124
04/01/2022	Apr	2431

テキストから日付への暗黙的な変換のために、DateFormat 定義 MM/DD/YYYY が使用されます。このため、[date] 項目は日付として正しく解釈されます。結果テーブルにあるとおり、日付の表示には同じ形式が使用されます。

例 2 – システム変数の変更

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 前述の例からデータセットと同じです。
- 「DD/MM/YYYY」形式を使用する DateFormat 関数。

ロードスクリプト

```
SET DateFormat='DD/MM/YYYY';
Transactions:
LOAD
date,
month(date) as month,
id,
amount
INLINE
[
date,id,amount
```

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- month

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

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

DateFormat 定義は「DD/MM/YYYY」に設定されていたため、最初の「/」記号の後に続く2桁が月として解釈され、その結果すべてのレコードは1月からのものになります。

例 3 – 日付の解釈

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 数値形式の日付によるデータセット。
- 「DD/MM/YYYY」形式を使用するDateFormat変数。
- date() 変数。

ロードスクリプト

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

Transactions:
Load
date(numerical_date),
month(date(numerical_date)) as month,
id,
```

```
amount
Inline
[
numerical_date,id,amount
43254,1,1000
43255,2,2123
43256,3,4124
43258,4,2431
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- month

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

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

ロードスクリプトで、date() 関数を使用し、数字表記の日付を日付形式に変換します。関数内には 2 番目の引数として指定された形式がないため、DateFormat 形式が使用されます。このため、日付項目では形式「MM/DD/YYYY」が使用されます。

例 4 – 外国の日付形式設定

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 日付のデータセット。
- DateFormat 変数 。「DD/MM/YYYY」形式ですが、スラッシュでコメント解除されています。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- month

このメジャーを作成します。

```
=sum(amount)
```

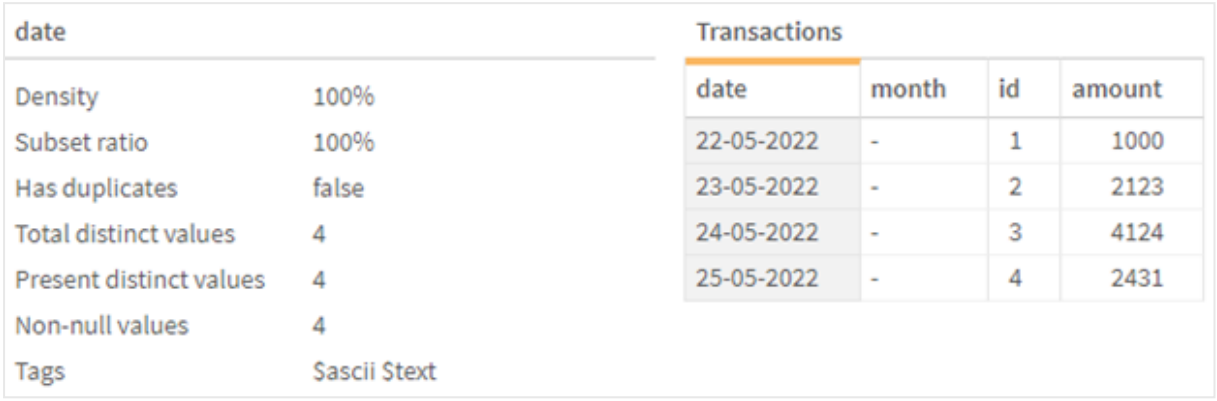
結果テーブル

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

最初のロードスクリプトで DateFormat に使用されるのは、デフォルトの「MM/DD/YYYY」です。トランザクションデータセット内の [date] 項目がこの形式ではないため、項目は日付として解釈されません。これは、[month] 項目が null である結果テーブルに表れています。

date 項目の「タグ」プロパティを調べると、データモデルビューア内の解釈済みデータ型を確認できます。

Transactions テーブルのプレビュー。date 項目の「タグ」では、テキスト入力データが暗黙的に日付/タイムスタンプに変換されていないことを示しているのに注意してください。



date	
Density	100%
Subset ratio	100%
Has duplicates	false
Total distinct values	4
Present distinct values	4
Non-null values	4
Tags	$ascii $text

Transactions

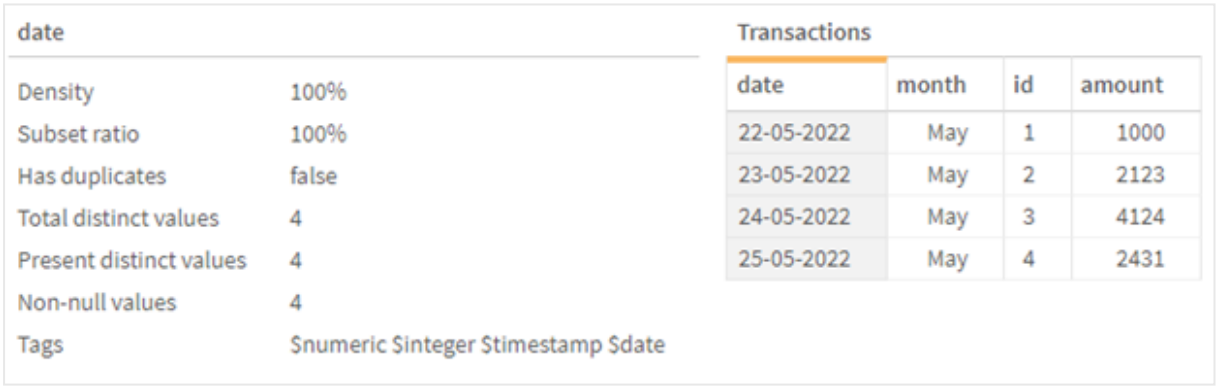
date	month	id	amount
22-05-2022	-	1	1000
23-05-2022	-	2	2123
24-05-2022	-	3	4124
25-05-2022	-	4	2431

これは、DateFormat システム変数を有効にすることで解決できます。

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

二重スラッシュを削除し、データをリロードします。

Transactions テーブルのプレビュー。date 項目の「タグ」では、テキスト入力データが暗黙的に日付/タイムスタンプに変換されていることを示しているのに注意してください。



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

Transactions

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

DayNames

定義した書式が地域設定によって設定された曜日表記規則の代わりに使用されます。

構文:

DayNames

変数を変更する場合、個々の値を区切るためにセミコロン； が必要です。

DayName 関数の例

関数の例	結果の定義
`Set DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';`	このように DayNames 関数を使用すると、曜日名が省略形で定義されます。
`Set DayNames='M;Tu;W;Th;F;Sa;Su';`	このように DayNames 関数を使用すると、曜日名がイニシャルで定義されます。

DayNames 関数は、多くの場合、次の関数と組み合わせて使用されます。

	関連する関数
関数	**相互作用**
weekday (page 1052)	DayNames を項目値として返すスクリプト関数。
Date (page 1210)	DayNames を項目値として返すスクリプト関数。
LongDayNames (page 232)	DayNames の長い形式の値。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – システム変数の既定値

ロードスクリプトと結果

概要

この例では、データセットの日付は MM/DD/YYYY 形式です。

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされる、日付を含むデータセット。
- date 項目。
- 既定の DayNames 定義。

ロードスクリプト

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

Transactions:
LOAD
date,
WeekDay(date) as dayname,
id,
amount
INLINE
[
```

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します。

- date
- dayname

このメジャーを作成します。

```
sum(amount)
```

結果テーブル

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

ロードスクリプトでは、WeekDay 関数が date 項目と共に指定された引数として使用されます。結果テーブルでは、この WeekDay 関数の出力は DayNames 定義の形式で曜日を表示します。

例 2 – システム変数の変更

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。最初の例と同じデータセットとシナリオが使用されます。

ただし、スクリプトの開始時に、DayNames 定義がアフリカーンス語の省略形の曜日を使用するように変更されています。

ロードスクリプト

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

Transactions:
Load
date,
WeekDay(date) as dayname,
id,
amount
```

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- dayname

このメジャーを作成します。

```
sum(amount)
```

結果テーブル

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

結果テーブルでは、この 関数の出力は 定義の形式で曜日を表示します。WeekDayDayNames

DayNames の言語がこの例のように変更されても、LongDayNames では引き続き英語の曜日が表示されることを覚えておくことが重要です。アプリケーションで両方の変数が使用されている場合は、これも変更する必要があります。

例 3 – 日付関数

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactionsという名前のテーブルにロードされる、日付を含むデータセット。
- date 項目。
- 既定の DayNames 定義。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- dayname

このメジャーを作成します。

```
sum(amount)
```

結果テーブル

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

既定の DayNames 定義が使用されます。ロードスクリプトでは、Date 関数が date 項目と共に最初の引数として使用されます。2 番目の引数は WWWです。この形式は、結果を DayNames 定義に格納された値に変換します。これは、結果テーブルの出力に表示されます。

DecimalSep

定義した小数点記号が地域設定によって設定された小数点記号の代わりに使用されます。

Qlik Sense は、認識可能な数字パターンが検出されるたびに、テキストを数字として自動的に解釈します。ThousandSep および DecimalSep システム変数は、テキストを数値として解析するときに適用されるパターンの構

成を決定します。ThousandSep 変数とDecimalSep 変数は、フロントエンドのチャートとテーブルで数値コンテンツを視覚化する際の既定の数値書式パターンを設定します。つまり、フロントエンド式の **[数値書式設定]**オプションに直接影響します。

コンマ「,」の 1000 区切り記号と「.」の小数点区切り記号を想定すると、これらは同等の数値に暗黙的に変換されるパターンの例です。

0,000.00

0000.00

0,000

これらは、テキストのまま変更されない、つまり数値に変換されないパターンの例です。

0.000,00

0,00

構文:

DecimalSep

関数の例

例	結果
Set DecimalSep='.';	「.」を小数点の記号として設定します。
Set DecimalSep=',';	「,」を小数点の記号として設定します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 – 異なる入力データに対する数値区切り変数の設定の影響

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 合計と日付のデータセットで、合計がさまざまな書式パターンで設定されています。
- Transactionsという名前のテーブル。
- 「.」に設定されたDecimalSep変数。
- 「,」に設定されたThousandSep変数。
- delimiter変数として設定されている「|」文字を使用して、行内のさまざまな項目を区切ります。

ロードスクリプト

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

Transactions:
Load date,
id,
amount as amount
Inline
[
date|id|amount
01/01/2022|1|1.000-45
01/02/2022|2|23.344
01/03/2022|3|4124,35
01/04/2022|4|2431.36
01/05/2022|5|4,787
01/06/2022|6|2431.84
01/07/2022|7|4132.5246
01/08/2022|8|3554.284
01/09/2022|9|3.756,178
01/10/2022|10|3,454.356
] (delimiter is '|');
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸amountとして追加します。

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

Amount	=Sum(amount)
合計	20814.7086
1.000-45	
3.756,178	
4124,35	
23.344	23.344
2431.36	2431.36
2431.84	2431.84

Amount	=Sum(amount)	
	3,454.356	3454.356
	3554.284	3554.284
	4132.5246	4132.5246
	4,787	4787

数値として解釈されない値はテキストのままで表示され、既定で左揃えになります。正常に変換された値はすべて右揃えで表示され、元の入力形式が維持されます。

数式列には、同等の数値が表示されます。既定の形式では、小数点区切り文字「.」のみが使用されます。これは、式構成の [**数値形式**] ドロップダウン設定でオーバーライドできます。

FirstWeekDay

週の最初として使用する曜日を定義する整数です。

構文:

```
FirstWeekDay
```

日付と時刻の表現の国際規格であるISO 8601では、月曜日が週初めの日となっています。月曜日も、イギリス、フランス、ドイツ、スウェーデンなど多くの国で週初めの日として使われています。

しかし米国やカナダなど他の国では、日曜日が週初めの日とみなされています。

Qlik Senseでは、アプリ作成時に地域設定がフェッチされ、対応する設定は環境変数としてスクリプトに保管されます。

米国のアプリ開発者は、日曜日に対応して、スクリプトで `Set FirstWeekDay=6;` をよく取得します。ヨーロッパのアプリ開発者は、月曜日に対応して、スクリプトで `Set FirstWeekDay=0;` をよく取得します。

FirstWeekDayに設定可能な値

値	毎日
0	月曜日
1	火曜日
2	水曜日
3	木曜日
4	金曜日
5	土曜日
6	日曜日

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

週数と週番号の ISO 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

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

US 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

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

例 1 – 既定値の使用 (スクリプト)

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

この例では、ロードスクリプトはデフォルトの Qlik Sense システム変数、FirstWeekDay=6 を使用します。このデータには、2020 年における最初の 14 日間のデータが含まれます。

ロードスクリプト

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

SET FirstWeekDay = 6;

Sales:
LOAD
        date,
        sales,
        week(date) as week,
        weekday(date) as weekday
Inline [
date,sales
```

```
01/01/2021,6000
01/02/2021,3000
01/03/2021,6000
01/04/2021,8000
01/05/2021,5000
01/06/2020,7000
01/07/2020,3000
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
01/12/2020,7000
01/13/2020,7000
01/14/2020,7000
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week
- weekday

結果テーブル

Date	週	weekday
01/01/2021	1	水
01/02/2021	1	Thu
01/03/2021	1	Fri
01/04/2021	1	Sat
01/05/2021	2	日
01/06/2020	2	月
01/07/2020	2	火
01/08/2020	2	水
01/09/2020	2	Thu
01/10/2020	2	Fri
01/11/2020	2	Sat
01/12/2020	3	日
01/13/2020	3	月
01/14/2020	3	火

デフォルト設定が使用されているため、FirstWeekDay システム変数は 6 に設定されています。結果テーブルには、それぞれの週が日曜日で始まるように表示されます (1 月 5 日と12 日)。

例 2 – FirstWeekDay 変数の変更 (スクリプト)

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

この例では、データには、2020 年における最初の 14 日間が含まれます。スクリプトの先頭で、FirstWeekDay 変数を 3 に設定します。

ロードスクリプト

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

SET FirstWeekDay = 3;

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week
- weekday

結果テーブル

Date	週	weekday
01/01/2021	52	水
01/02/2021	1	Thu
01/03/2021	1	Fri
01/04/2021	1	Sat
01/05/2021	1	日
01/06/2020	1	月
01/07/2020	1	火
01/08/2020	1	水
01/09/2020	2	Thu
01/10/2020	2	Fri
01/11/2020	2	Sat
01/12/2020	2	日
01/13/2020	2	月
01/14/2020	2	火

FirstWeekDay システム変数が 3 に設定されているため、各週は木曜日に始まります。結果テーブルには、それぞれの週が木曜日で始まるように表示されます (1 月 2 日と 9 日)。

LongDayNames

定義した書式が地域設定の長い曜日名の表記規則の代わりに使用されます。

構文:

LongDayNames

次の LongDayNames 関数の例では、曜日名を完全に定義しています。

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

変数を変更する場合、個々の値を区切るためにセミコロン ; が必要です。

LongDayNames 関数は、項目値として DayNames を返す *Date (page 1210)* 関数と組み合わせて使用できます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディタは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – システム変数の既定値

ロードスクリプトと結果

概要

データロードエディタを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされる、日付を含むデータセット。
- date 項目。
- 既定の LongDayNames 定義。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- dayname

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

日付	dayname	=sum(amount)
01/01/2022	土曜日	1000
02/01/2022	火曜日	2123
03/01/2022	火曜日	4124
04/01/2022	金曜日	2431

ロードスクリプトでは、dayname という項目を作成するには、Date 関数が date 項目と共に最初の引数として使用されます。関数の 2 番目の引数は WWWWの書式設定です。

この形式を使用すると、最初の引数の値が、変数 LongDayNamesに設定された、対応する完全な曜日名に変換されます。結果テーブルでは、作成した項目 dayname の項目値にこれが表示されます。

例 2 – システム変数の変更

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

最初の例と同じデータセットとシナリオが使用されます。ただし、スクリプトの開始時に、LongDayNames 定義がスペイン語の曜日を使用するように変更されています。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- dayname

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

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

ロードスクリプトでは、LongDayNames 変数が変更され、スペイン語で曜日が一覧表示されます。

次に、dayname という項目を作成します。これは、date 項目と共に最初の引数として使用される Date 関数です。

関数の 2 番目の引数は wwwwの書式設定です。この形式 Qlik Sense を使用すると、最初の引数の値が、変数 LongDayNamesに設定された、対応する完全な曜日名に変換されます。

結果テーブルでは、作成した項目 dayname の項目値に、スペイン語で書かれた完全な曜日が表示されます。

LongMonthNames

定義した書式が地域設定の長い月名の表記規則の代わりに使用されます。

構文:

LongMonthNames

変数を変更する場合、個々の値を区切るために ; を使用する必要があります。

次の LongMonthNames 関数の例では、曜日名を完全に定義しています。

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

LongMonthNames 関数は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
Date (page 1210)	DayNames を項目値として返すスクリプト関数。
LongDayNames (page 232)	DayNames の長い形式の値。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディタは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – システム変数の既定値

ロードスクリプトと結果

概要

データロードエディタを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされた日付のデータセット。
- date 項目。
- 既定の LongMonthNames 定義。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します。

- date
- monthname

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

日付	monthname	sum(amount)
01/01/2022	1月	1000.45
01/02/2022	1月	2123.34
01/03/2022	1月	4124.35
01/04/2022	1月	2431.36
01/05/2022	1月	4787.78
01/06/2022	1月	2431.84
01/07/2022	1月	2854.83
01/08/2022	1月	3554.28
01/09/2022	1月	3756.17
01/10/2022	1月	3454.35

既定の LongMonthNames 定義が使用されます。ロードスクリプトでは、month という項目を作成するには、Date 関数が date 項目と共に最初の引数として使用されます。関数の 2 番目の引数は MMMMの書式設定です。

この形式 Qlik Sense を使用すると、最初の引数の値が、変数 LongMonthNamesに設定された、対応する完全な月名に変換されます。結果テーブルでは、作成した項目 month の項目値にこれが表示されます。

例 2 – システム変数の変更

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされた日付のデータセット。
- date 項目。
- スペイン語で省略された曜日を使用するように変更された LongMonthNames 変数。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、sum(amount) をメジャー、これらの項目を軸として追加します。

- date
- monthname

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

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

ロードスクリプトでは、LongMonthNames 変数が変更され、スペイン語で月が一覧表示されます。次に、monthname という項目を作成するには、date 項目と共に最初の引数として使用される Date 関数です。関数の 2 番目の引数は MMMMの書式設定です。

この形式 Qlik Sense を使用すると、最初の引数の値が、変数 LongMonthNamesに設定された、対応する完全な月名に変換されます。結果テーブルでは、作成した項目 monthname の項目値にスペイン語で書かれた付きの名前が表示されます。

MoneyDecimalSep

定義した小数点記号が地域設定によって設定された通貨の小数点記号の代わりに使用されます。

既定により、Qlik Sense ではテーブル チャート内で数字とテキストが異なって表示されます。数字は右揃え、テキストは左揃えです。これにより、テキストから数字への変換の問題を容易に見つけることができます。Qlik Sense の結果を表示するこのページのいずれのテーブルもこの書式を使用します。

構文:

MoneyDecimalSep

Qlik Sense アプリケーションは、この書式に準拠したテキスト項目を貨幣価値として解釈します。テキスト項目には、MoneyFormat システム変数で定義された通貨記号を含んでいる必要があります。MoneyDecimalSep は、複数の異なる地域設定から受け取るデータソースを処理する際に特に有用です。

次の例は、MoneyDecimalSep システム変数の考えられる使用方法です。

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

この関数は、次の関数とよく併用されています。

関連する関数

関数	相互作用
MoneyFormat	テキスト項目を解釈する際、MoneyFormat 記号は解釈の一環として使用されます。数字書式については、チャートオブジェクトでは MoneyFormat 書式が Qlik Sense によって使用されます。
MoneyThousandSep	テキスト項目を解釈する際、MoneyThousandSep 関数も遵守する必要があります。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 - MoneyDecimalSep ドット (.) 表記

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされた日付のデータセット。
- ドット「.」が小数点の記号として使用されているテキスト書式の金額項目がある提供されたデータ。各レコードには「$」記号がプレフィックスとして付いています。ただし、最後のレコードのみは「£」記号が付きます。

MoneyFormat システム変数は、既定通貨としてドル「$」を定義します。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:amount。

次のメジャーを追加します。

- isNum(amount)
- sum(amount)

下記の結果をレビューし、すべてのドル「$」金額の適切な解釈を提示します。

結果テーブル

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

上記の結果テーブルでは、[amount] 項目がすべてのドル ($) プレフィックスが付いた値を正しく解釈し、ポンド (£) プレフィックスが付いた amount が金額に変換されていないことがわかります。

例 2 - MoneyDecimalSep カンマ (,) 表記

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされるデータセット。
- カンマ「,」が小数点の記号として使用されているテキスト書式の金額項目がある提供されたデータ。また、各レコードにはプレフィックスとして「$」記号が付いていますが、最後のレコードは誤ってドット小数点の記号が使われています。

MoneyFormat システム変数は、既定通貨としてドル「$」を定義します。

ロードスクリプト

```
SET MoneyThousandSep='.';
SET MoneyDecimalSep=',';
```

```
SET MoneyFormat='$###0.00;-$###0.00';

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

結果

結果の段落テキスト。

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:amount。

次のメジャーを追加します。

- isNum(amount)
- sum(amount)

下記の結果をレビューしてください。小数点の記号にドット「.」が使用されている金額以外、すべての値は正しく解釈されています。この場合、代わりにカンマを使用すべきでした。

結果テーブル

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

amount	=isNum(amount)	=Sum(amount)
$249,36	-1	$249.36
$545,36	-1	$545.36
$2.814,32	-1	$2814.32

MoneyFormat

システム変数は、数値のプレフィックスとして通貨記号が付くテキストから数値への自動変換のために Qlik が使用する書式パターンを定義します。また、Number Formatting プロパティが「金額」に設定されたメジャーがチャートオブジェクトに表示される方法も定義されます。

MoneyFormat システム変数で書式パターンの一部として定義された記号が、地域設定によって設定された通貨記号の代わりに使用されます。

既定により、Qlik Sense ではテーブル チャート内で数字とテキストが異なって表示されます。数字は右揃え、テキストは左揃えです。これにより、テキストから数字への変換の問題を容易に見つけることができます。Qlik Sense の結果を表示するこのページのいずれのテーブルもこの書式を使用します。

構文:

MoneyFormat

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

この書式は、数値項目の Number Formatting プロパティが Money に設定されたときに、チャートオブジェクトに表示されます。さらに、数値テキスト項目が Qlik Sense によって解釈される際、テキスト項目の通貨記号が MoneyFormat 変数で定義された記号と一致する場合、Qlik Sense がこの項目を金額値として解釈します。

この関数は、次の関数とよく併用されています。

関連する関数

関数	相互作用
MoneyDecimalSep (page 239)	数字書式については、オブジェクトの項目書式に MoneyDecimalSep が使用されます。
MoneyThousandSep (page 247)	数字書式については、オブジェクトの項目書式に MoneyThousandSep が使用されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 - MoneyFormat

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、Transactions という名前のテーブルに読み込まれるデータセットが含まれています。既定の MoneyFormat 変数定義が使用されます。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- amount

このメジャーを追加します。

```
=Sum(amount)
```

[数値書式] で、**[通貨]** を選択して、Sum(amount) を金額値として構成します。

結果テーブル

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

既定の MoneyFormat 定義が使用されます。これは次のようになります: $###0.00;-$###0.00。結果テーブルでは、[amount] 項目の書式には通貨記号と小数点が表示され、小数点以下桁数が含まれます。

例 2 - 千単位区切り記号と入力書式混在可能な MoneyFormat

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 混在入力書式データセット。これは千単位区切り記号と小数点の記号が混在した状態で Transactions というテーブルにロードされます。
- MoneyFormat 定義は、千単位区切り記号としてカンマを使うよう変更されています。
- データ行の 1 つが、誤ってカンマの千単位区切り記号で区切られています。この金額がテキストのままであり、数値に解釈されていないことに注意してください。

ロードスクリプト

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

Transactions:
```

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- amount

このメジャーを追加します:

```
=Sum(amount)
```

[数値書式] で、**[通貨]** を選択して、Sum(amount) を金額値として構成します。

結果テーブル

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

スクリプトの開始時に、MoneyFormat システム変数は千単位区切り記号としてカンマを使用するよう変更されます。Qlik Sense テーブルで、書式はこの区切り記号を使っていることがわかります。さらに、誤った区切り記号が使われた行が誤って解釈され、テキストのままになっています。金額の合計に含まれていないのはそのためです。

MoneyThousandSep

定義した桁区切り記号が地域設定によって設定された通貨の桁区切り記号の代わりに使用されます。

既定により、Qlik Sense ではテーブル チャート内で数字とテキストが異なって表示されます。数字は右揃え、テキストは左揃えです。これにより、テキストから数字への変換の問題を容易に見つけることができます。Qlik Sense の結果を表示するこのページのいずれのテーブルもこの書式を使用します。

構文:

MoneyThousandSep

Qlik Sense アプリケーションは、この書式に準拠したテキスト項目を貨幣価値として解釈します。テキスト項目には、MoneyFormat システム変数で定義された通貨記号を含んでいる必要があります。MoneyThousandSep は、複数の異なる地域設定から受け取るデータソースを処理する際に特に有用です。

次の例は、MoneyThousandSep システム変数の考えられる使用方法です。

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

この関数は、次の関数とよく併用されています。

関連する関数

関数	相互作用
MoneyFormat	テキスト項目を解釈する際、MoneyFormat 記号は解釈の一環として使用されます。数字書式については、チャート オブジェクトでは MoneyFormat 書式が Qlik Sense によって使用されます。
MoneyDecimalSep	テキスト項目を解釈する際、MoneyDecimalSep 関数も遵守する必要があります。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 - MoneyThousandSep カンマ (,) 表記

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされるデータセット。
- カンマが千単位区切り記号として使用されているテキスト書式の金額項目がある提供されたデータ。また、各レコードには「$」記号がプレフィックスとして付いています。

MoneyFormat システム変数は、既定通貨としてドル「$」を定義します。

ロード スクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:amount。

次のメジャーを追加します。

- isNum(amount)
- sum(amount)

下記の結果をレビューしてください。テーブルでは、千単位区切り記号としてカンマ「,」記号が使用されたすべての値が正しく解釈されていることがわかります。

[amount] 項目のすべての値は正しく解釈されていますが、千単位区切り記号としてドット「.」を使用している値は例外です。

結果テーブル

amount	=isNum(amount)	=Sum(amount)
合計	0	$161645330590.00
$3.454.343.566	0	$0.00
$249,475,336	-1	$249475336.00
$545,883,436	-1	$545883436.00
$7,873,578,754	-1	$7873578754.00
$10,000,000,441	-1	$10000000441.00
$21,237,492,432	-1	$21237492432.00
$24,313,369,837	-1	$24313369837.00
$24,33,884,663	-1	$24313884663.00
$35,545,828,255	-1	$35545828255.00
$37,565,817,436	-1	$37565817436.00

例 2 - MoneyThousandSep ドット (.) 表記

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされるデータセット。
- ドット「.」が千単位区切り記号として使用されているテキスト書式の金額項目がある提供されたデータ。また、各レコードには「$」記号がプレフィックスとして付いています。

MoneyFormat システム変数は、既定通貨としてドル「$」を定義します。

ロードスクリプト

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

Transactions:
Load
```

```
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,'$10.000.000.441'
01/02/2022,2,'$21.237.492.432'
01/03/2022,3,'$249.475.336'
01/04/2022,4,'$24.313.369.837'
01/05/2022,5,'$7.873.578.754'
01/06/2022,6,'$24.313.884.663'
01/07/2022,7,'$545.883.436'
01/08/2022,8,'$35.545.828.255'
01/09/2022,9,'$37.565.817.436'
01/10/2022,10,'$3,454,343,566'
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:amount。

次のメジャーを追加します。

- isNum(amount)
- sum(amount)

下記の結果をレビューしてください。千単位区切り記号としてドット「.」表記が使用されているすべての金額が正しく解釈されています。

[amount] 項目のすべての値は正しく解釈されていますが、千単位区切り記号としてカンマ「,」を使用している値は例外です。

結果テーブル

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

MonthNames

定義した書式が地域設定の月名の表記規則の代わりに使用されます。

構文:

MonthNames

変数を変更する場合、個々の値を区切るために ; を使用する必要があります。

関数の例

例	結果
`Set MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';`	MonthNames 関数を使用すると、月名が英語および省略形で定義されます。
`Set MonthNames='Enero;Feb;Marzo;Abr;Mayo;Jun;Jul;Agosto;Set;Oct;Nov;Dic';`	MonthNames 関数を使用すると、月名がスペイン語および省略形で定義されます。

MonthNames 関数は、次の関数と組み合わせて使用できます。

関連する関数

関数	相互作用
month (page 898)	MonthNames で定義された値を項目値として返すスクリプト関数
Date (page 1210)	提供された形式設定引数に基づいて MonthNames で定義された値を項目値として返すスクリプト関数。
LongMonthNames (page 235)	MonthNames の長い形式の値

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – システム変数の既定

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされた日付のデータセット。
- date 項目。
- 既定の MonthNames 定義。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- monthname

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

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

既定の MonthNames 定義が使用されます。ロードスクリプトでは、 Month 関数が date 項目と共に指定された引数として使用されます。

結果テーブルでは、この Month 関数の出力は MonthNames 定義の形式で月を表示します。

例 2 – システム変数の変更

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされた日付のデータセット。
- date 項目。
- スペイン語で省略された月を使用するように変更された MonthNames 変数。

ロードスクリプト

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

Transactions:
LOAD
date,
month(date) as month,
id,
amount
INLINE
[
date,id,amount
```

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- monthname

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

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

ロードスクリプトでは、まず MonthNames 変数が変更され、スペイン語の省略形で月が一覧表示されます。Month 関数は、date 項目と共に指定された引数として使用されます。

結果テーブルでは、この Month 関数の出力は MonthNames 定義の形式で月を表示します。

MonthNames 変数の言語がこの例のように変更されても、LongMonthNames 変数では引き続き英語の月名が表示されることを覚えておくことが重要です。アプリケーションで両方の変数が使用されている場合は、LongMonthNames 変数を変更する必要があります。

例 3 – 日付関数

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions という名前のテーブルにロードされた日付のデータセット。
- date 項目。
- 既定の MonthNames 定義。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- monthname

このメジャーを作成します。

```
=sum(amount)
```

結果テーブル

日付	monthname	sum(amount)
01/01/2022	Jan	1000.45
01/02/2022	Jan	2123.34
01/03/2022	Jan	4124.35
01/04/2022	Jan	2431.36

日付	monthname	sum(amount)
01/05/2022	Jan	4787.78
01/06/2022	Jan	2431.84
01/07/2022	Jan	2854.83
01/08/2022	Jan	3554.28
01/09/2022	Jan	3756.17
01/10/2022	Jan	3454.35

既定の MonthNames 定義が使用されます。ロードスクリプトでは、Date 関数が date 項目と共に最初の引数として使用されます。2 番目の引数は です。MMM

この形式 Qlik Sense を使用すると、最初の引数の値が、変数 MonthNamesに設定された、対応する月名に変換されます。結果テーブルでは、作成した項目 month の項目値にこれが表示されます。

NumericalAbbreviation

数値の省略形を使用して、数字のスケールプレフィックスに使用する省略形を設定します。例えば、メガや 100 万 (106) には M、^{6}マイクロ (10-6) には μ^{-6}).

構文:

NumericalAbbreviation

NumericalAbbreviation 変数は、セミコロンで区切られた省略形定義ペアのリストが含まれた文字列に対して設定します。各省略形定義ペアに、スケール (10 進法での指数) と省略形がコロンで区切られて含まれている必要があります。例えば、100 万の場合、6:M となります。

既定の設定は「'3:k;6:M;9:G;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y'」です。

この設定により、千のプレフィックスが t に、10 億のプレフィックスが B に変わります。これは、t$、M$、B$ などの省略形が必要となる財務アプリケーションで役立ちます。

```
Set NumericalAbbreviation='3:t;6:M;9:B;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y';
```

ReferenceDay

この設定は、第 1 週を定義する基準日として 1 月のどの日を設定するかを定義します。つまりこの設定は、第 1 週うち何日間が 1 月内の日付でなければならないかを規定します。

構文:

ReferenceDay

ReferenceDay は、年の最初の週に含まれる日数を設定します。ReferenceDay は 1 と7 の間の任意の値に設定できます。1-7 の範囲外の値は、週の中間点 (4) として解釈されます。これは、ReferenceDay が 4に設定されているという状態に相当します。

ReferenceDay 設定の値を選択しない場合、以下の ReferenceDay 値テーブルにあるように、既定値には ReferenceDay=0 が表示され、週の中間点 (4) として解釈されます。

ReferenceDay 関数は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

変数	相互作用
BrokenWeeks (page 214)	Qlik Sense アプリが分割されない週で動作している場合、ReferenceDay 変数の設定が適用されます。ただし、分割された週が使用されている場合、第 1 週は 1 月 1 日に開始され、FirstWeekDay 変数の設定と連動して終了し、ReferenceDay フラグは無視されます。
FirstWeekDay (page 228)	週の最初として使用する曜日を定義する整数です。

Qlik Sense では、ReferenceDay に次の値を設定できます。

ReferenceDay 値

値	基準日
0 (既定)	1 月 4 日
1	1 月 1 日
2	1 月 2 日
3	1 月 3 日
4	1 月 4 日
5	1 月 5 日
6	1 月 6 日
7	1 月 7 日

次の例では、ReferenceDay = 3 は 1 月 3 日を基準日として定義しています。

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

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

週数と週番号のISO設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

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

US設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

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

例 1 – 既定値; ReferenceDay=0 を使用したロードスクリプト

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 0 に設定された ReferenceDay 変数。
- 0 に設定された BrokenWeeks 変数は、アプリに分割していない週を強制します。
- 2019 年末から2020 年初頭までの日付のデータセット。

ロードスクリプト

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

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

```
01/08/2020,5000
01/09/2020,9000
01/10/2020,5000
01/11/2020,7000
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week
- weekday

結果テーブル

日付	週	weekday
12/27/2019	52	金
12/28/2019	52	土
12/29/2019	1	日
12/30/2019	1	月
12/31/2019	1	火
01/01/2020	1	水
01/02/2020	1	木
01/03/2020	1	金
01/04/2020	1	土
01/05/2020	2	日
01/06/2020	2	月
01/07/2020	2	火
01/08/2020	2	水
01/09/2020	2	木
01/10/2020	2	金
01/11/2020	2	土

第 52 週は 12 月 28 日土曜日に終了します。ReferenceDay では 1 月 4 日を第 1 週に含める必要があるため、第 1 週は 12 月 29 日に始まり、1 月 4 日の土曜日に終了します。

例 - ReferenceDay 変数を 5 に設定

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 5 に設定された ReferenceDay 変数。
- 0 に設定された BrokenWeeks 変数は、アプリに分割していない週を強制します。
- 2019 年末から 2020 年初頭までの日付のデータセット。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week
- weekday

結果テーブル

日付	週	weekday
12/27/2019	52	金
12/28/2019	52	土

日付	週	weekday
12/29/2019	53	日
12/30/2019	53	月
12/31/2019	53	火
01/01/2020	53	水
01/02/2020	53	木
01/03/2020	53	金
01/04/2020	53	土
01/05/2020	1	日
01/06/2020	1	月
01/07/2020	1	火
01/08/2020	1	水
01/09/2020	1	木
01/10/2020	1	金
01/11/2020	1	土

第 52 週は 12 月 28 日土曜日に終了します。BrokenWeeks 変数は、アプリに分割していない週を強制します。5 の基準日の値では、1 月 5 日を第 1 週に含める必要があります。

ただし、これは前年の第 52 週の終了から 8 日後です。したがって、第 53 週は 12 月 29 日に開始し、1 月 4 日に終了します。第 1 週は 1 月 5 日の日曜日に開始します。

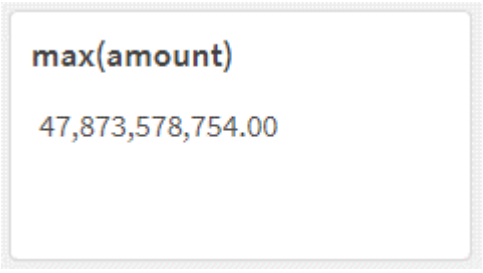
ThousandSep

定義した桁区切り記号がオペレーティング システム (地域設定) の桁区切り記号の代わりに使用されます。

構文:

ThousandSep

ThousandSep 変数を使用している Qlik Sense オブジェクト (千単位区切り記号付き)



Qlik Sense アプリは、この書式に準拠したテキスト項目を数値として解釈します。この書式は、数値項目の **数値書式** プロパティが **数値** に設定されたときに、チャート オブジェクトに表示されます。

ThousandSep は、複数の地域設定から受け取ったデータ ソースを処理する際に有用です。

*オブジェクトがすでに作成されてアプリケーションで書式設定された後に ThousandSep 変数が変更された場合、ユーザーは**数値書式**プロパティの**数値**を選択解除してから選択しなおすことにより、各関連項目の書式を設定しなおす必要があります。*

次の例は、ThousandSep システム変数の考えられる使用方法です。

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


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

これらのトピックは、この関数を使用するのに役立つかもしれません。

関連トピック

トピック	説明
DecimalSep (page 225)	テキスト項目を解釈する際、この関数により提供される小数点の記号設定もまた尊重される必要があります。数値書式については、必要に応じて **DecimalSep** が Qlik Sense によって使用されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 - 既定システムの変数

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされるデータセット。
- 既定の ThousandSep 変数定義の使用。

ロードスクリプト

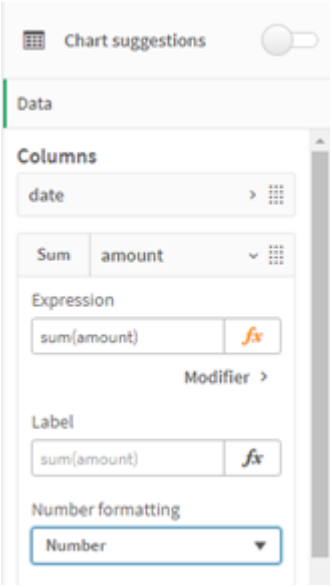
```
Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
01/01/2022,1,10000000441
01/02/2022,2,21237492432
01/03/2022,3,41249475336
01/04/2022,4,24313369837
01/05/2022,5,47873578754
01/06/2022,6,24313884663
01/07/2022,7,28545883436
01/08/2022,8,35545828255
01/09/2022,9,37565817436
01/10/2022,10,3454343566
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。
2. 次のメジャーを追加します。
   `=sum(amount)`
3. プロパティパネルの **[データ]** でメジャーを選択します。
4. **[数字の書式設定]** で、**[数値]** を選択します。

チャートメジャーに対する数値書式設定の調整



結果テーブル

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

この例では、カンマ書式 (「,」) が設定された既定の ThousandSep 定義が使用されます。結果テーブルで、金額項目の書式には千単位の間にカンマが表示されています。

例 2 – システム変数の変更

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセット。Transactions というテーブルにロードされます。
- スクリプトの最初の ThousandSep 定義で、千単位区切り記号として「*」文字が表示されています。これは極端な例であり、変数の機能を示すためのみに使用されています。

この例で使用されている変更は極端なものであり、一般的ではありませんが、変数の機能を示すために表示されています。

ロードスクリプト

```
SET ThousandSep='*';

Transactions:
Load
date,
id,
amount
Inline
[
date,id,amount
```

```
01/01/2022,1,10000000441
01/02/2022,2,21237492432
01/03/2022,3,41249475336
01/04/2022,4,24313369837
01/05/2022,5,47873578754
01/06/2022,6,24313884663
01/07/2022,7,28545883436
01/08/2022,8,35545828255
01/09/2022,9,37565817436
01/10/2022,10,3454343566
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。
2. 次のメジャーを追加します。
   `=sum(amount)`
3. プロパティパネルの **[データ]** でメジャーを選択します。
4. **[数字の書式設定]** で、**[カスタム]** を選択します。

結果テーブル

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

スクリプトの開始時に、`ThousandSep` システム変数は「*」に変更されます。結果テーブルで、金額項目の書式には千単位の間に「*」が表示されているのが確認できます。

例 3 – テキストの解釈

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされるデータセット。
- カンマが千単位区切り記号として使用されているテキスト書式の数値項目があるデータ。
- 既定の ThousandSep システム変数の使用。

ロードスクリプト

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

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。
2. 次のメジャーを追加します。
   =sum(amount)
3. プロパティパネルの **[データ]** でメジャーを選択します。
4. **[数字の書式設定]** で、**[数値]** を選択します。
5. 次のメジャーを追加して、金額項目が次の数値の値であるかどうかを評価します:
   =isnum(amount)

結果テーブル

日付	=sum(amount)	=isnum(amount)
01/01/2022	10,000,000,441.00	-1
01/02/2022	21,492,432.00	-1
01/03/2022	4,249,475,336.00	-1
01/04/2022	24,313,369,837.00	-1
01/05/2022	4,873,578,754.00	-1
01/06/2022	313,884,663.00	-1
01/07/2022	2,545,883,436.00	-1
01/08/2022	545,828,255.00	-1
01/09/2022	37,565,817,436.00	-1
01/10/2022	3*454*343*566.00	-1

データがロードされると、データが ThousandSep 変数に準拠しているため、Qlik Sense が金額項目を数値として解釈したことがわかります。これは、各入力を -1 または TRUE と評価する isnum() 関数によって示されています。

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

TimeFormat

定義した書式がオペレーティング システム (地域設定) の時刻書式の代わりに使用されます。

構文:

```
TimeFormat
```

```
Set TimeFormat='hh:mm:ss';
```

TimestampFormat

定義した書式がオペレーティング システム (地域設定) の日付と時刻の書式の代わりに使用されます。

構文:

```
TimestampFormat
```

次の例では、タイムスタンプ データとして *1983-12-14T13:15:30Z* を使用して、さまざまな **SET TimestampFormat** ステートメントの結果を表示します。使用される日付書式は **YYYYMMDD**、また時刻書式は **h:mm:ss TT** です。日付書式は **SET DateFormat** ステートメントによって、時刻書式は **SET**

TimeFormat ステートメントによってデータ ロード スクリプトの先頭で指定されます。

結果

例	結果
`SET TimestampFormat='YYYYMMDD';`	19831214
`SET TimestampFormat='M/D/YY hh:mm:ss[.fff]';`	12/14/83 13:15:30
`SET TimestampFormat='DD/MM/YYYY hh:mm:ss[.fff]';`	14/12/1983 13:15:30
`SET TimestampFormat='DD/MM/YYYY hh:mm:ss[.fff] TT';`	14/12/1983 1:15:30 PM
`SET TimestampFormat='YYYY-MM-DD hh:mm:ss[.fff] TT';`	1983-12-14 01:15:30

例: ロード スクリプト

例: ロード スクリプト

最初のロード スクリプト *SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT'* が使用されます。2 番目のロード スクリプトで、タイムスタンプの書式は *SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]'* に変更されます。さまざまな結果は、時刻データ書式が異なる場合の **SET TimeFormat** ステートメントによる処理の違いを示しています。

以下のテーブルは、後続のロード スクリプトで使用されるデータ セットを示しています。テーブルの 2 番目の列は、データ セット内の各タイムスタンプの書式を示しています。最初の 5 つのタイムスタンプは ISO 8601 のルールに従っていますが、6 番目は従っていません。

データ セット

使用される時刻データとデータ セット内の各タイムスタンプの書式を示したテーブル。

transaction_timestamp	time data format
2018-08-30	YYYY-MM-DD
20180830T193614.857	YYYYMMDDhhmmss.sss
20180830T193614.857+0200	YYYYMMDDhhmmss.sss±hhmm
2018-09-16T12:30-02:00	YYYY-MM-DDhh:mm±hh:mm
2018-09-16T13:15:30Z	YYYY-MM-DDhh:mmZ
9/30/18 19:36:14	M/D/YY hh:mm:ss

データ ロード エディターで、新しいセクションを作成し、サンプル スクリプトを追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

ロード スクリプト

```
SET FirstWeekDay=0;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
```

```
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
SET DateFormat='YYYYMMDD';
SET TimestampFormat='DD/MM/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
*,
Timestamp(transaction_timestamp, 'YYYY-MM-DD hh:mm:ss[.fff]') as LogTimestamp
;

Load * Inline [
transaction_id, transaction_timestamp, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 2018-08-30, 12423.56, 23, 0,2038593, L, Red
3751, 20180830T193614.857, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180830T193614.857+0200, 15.75, 1, 0.22, 5646471, S, blue
3753, 2018-09-16T12:30-02:00, 1251, 7, 0, 3036491, l, Black
3754, 2018-09-16T13:15:30Z, 21484.21, 1356, 75, 049681, xs, Red
3755, 9/30/18 19:36:14, -59.18, 2, 0.333333333333333, 2038593, M, Blue
];
```

結果

ロード スクリプトで使用中の TimestampFormat データ型変数の結果を示している Qlik Sense テーブル。データ セットの最後のタイムスタンプは正しい日付を返しません。

transaction_id	transaction_timestamp	LogTimeStamp
3750	2018-08-30	2018-08-30 00:00:00
3751	20180830T193614.857	2018-08-30 19:36:14
3752	20180830T193614.857+0200	2018-08-30 17:36:14
3753	2018-09-16T12:30-02:00	2018-09-16 14:30:00
3754	2018-09-16T13:15:30Z	2018-09-16 13:15:30
3755	9/30/18 19:36:14	-

次のロード スクリプトでは、同じデータ セットを使用します。ただし、*SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]'* を使用して 6 番目のタイムスタンプの非 ISO 8601 書式に一致させます。

データ ロード エディターで、前のサンプル スクリプトを以下のスクリプトで置き換えて実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

ロード スクリプト

```
SET FirstWeekDay=0;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
SET DateFormat='YYYYMMDD';
SET TimestampFormat='MM/DD/YYYY hh:mm:ss[.fff]';
```

```
Transactions:
Load
*,
Timestamp(transaction_timestamp, 'YYYY-MM-DD hh:mm:ss[.fff]') as LogTimestamp
;

Load * Inline [
transaction_id, transaction_timestamp, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 2018-08-30, 12423.56, 23, 0,2038593, L, Red
3751, 20180830T193614.857, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180830T193614.857+0200, 15.75, 1, 0.22, 5646471, S, blue
3753, 2018-09-16T12:30-02:00, 1251, 7, 0, 3036491, l, Black
3754, 2018-09-16T13:15:30Z, 21484.21, 1356, 75, 049681, xs, Red
3755, 9/30/18 19:36:14, -59.18, 2, 0.333333333333333, 2038593, M, Blue
];
```

結果

ロードスクリプトで使用中の TimestampFormat データ型変数の結果を示している Qlik Sense テーブル。

transaction_id	transaction_timestamp	LogTimeStamp
3750	2018-08-30	2018-08-30 00:00:00
3751	20180830T193614.857	2018-08-30 19:36:14
3752	20180830T193614.857+0200	2018-08-30 17:36:14
3753	2018-09-16T12:30-02:00	2018-09-16 14:30:00
3754	2018-09-16T13:15:30Z	2018-09-16 13:15:30
3755	9/30/18 19:36:14	2018-09-16 19:36:14

4.9 Direct Discovery 変数

Direct Discovery システム変数

DirectCacheSeconds

ビジュアライゼーションの Direct Discovery クエリの結果のキャッシュ制限を設定できます。この時間制限に達すると、Qlik Sense は新たな Direct Discovery クエリが行われた際にキャッシュをクリアします。Qlik Sense は選択用にソースデータのクエリを実行し、指定した時間制限に沿ったキャッシュを再び作成します。選択の組み合わせの結果は、個別にキャッシュされます。つまり、キャッシュは各選択ごとに個別に更新されるため、1 つ目の選択はその項目のキャッシュのみを更新し、2 つ目の選択はその項目のキャッシュのみを更新します。1 つ目の選択で更新された項目が 2 つ目の選択に含まれる場合、キャッシュ制限に達していなければ、これらの項目のキャッシュが再び更新されることはありません。

Direct Discovery キャッシュは [**テーブル**] のビジュアライゼーションには適用されません。テーブルの選択は、データソースを毎回クエリします。

制限値は秒単位で設定する必要があります。既定のキャッシュ制限は 1800 秒 (30 分) です。

DirectCacheSeconds に使用する値は、**DIRECT QUERY** ステートメント実行時に設定される値です。実行時にこの値を変更することはできません。

```
SET DirectCacheSeconds=1800;
```

DirectConnectionMax
接続プーリング機能を使用することで、データベースへの非同期の同時呼び出しを行うことができます。ロードスクリプト構文では、プーリング機能を以下のように設定します。

```
SET DirectConnectionMax=10;
```
この数値設定は、シート更新時に Direct Discovery コードが使用するデータベース接続の最大数を指定します。デフォルト設定は 1 です。

この変数の扱いには注意が必要で、1 以上に設定すると Microsoft SQL Server に接続した際に問題が生じることが判明しています。

DirectUnicodeStrings
Direct Discovery は、一部のデータベース (特にSQL Server) で必要とされている場合、拡張された文字列リテラル (N'<extended string>') 向けに SQL 標準形式を使用して、拡張 Unicode データの選択をサポートできます。この構文は、スクリプト変数 **DirectUnicodeStrings** を用いることで Direct Discovery での使用が可能になります。

この変数を「true」に設定すると、ANSI 標準ワイド文字マーカーの "N" を文字列リテラルの前に使用できるようになります。ただし、一部のデータベースではサポートされていません。デフォルト設定は「false」です。

DirectDistinctSupport
DIMENSION 項目値が Qlik Sense オブジェクトで選択されている場合、ソースデータベースに対するクエリが生成されます。クエリでグループ化が要求されている場合、Direct Discovery は **DISTINCT** キーワードを用いて一意の値のみ選択します。ただし、**GROUP BY** キーワードを使用しなければならないデータベースも一部存在します。この場合、**DirectDistinctSupport** を 'false' に設定し、一意の値のクエリで **DISTINCT** ではなく **GROUP BY** キーワードを用いるようにします。

```
SET DirectDistinctSupport='false';
```
DirectDistinctSupport を true に設定すると、**DISTINCT** が使用されます。設定しなかった場合は、デフォルトの **DISTINCT** が使用されます。

DirectEnableSubquery
濃度の高いマルチテーブルシナリオでは、大きな IN 節を生成する代わりに SQL クエリで複数のサブクエリを生成することができます。これは、**DirectEnableSubquery** に 'true' を設定することで可能になります。この値の既定値は 'false' です。

***DirectEnableSubquery** が有効になっている場合、Direct Discovery モードではないテーブルをロードできません。*

```
SET DirectEnableSubquery='true';
```

Teradata クエリバンド変数

Teradata クエリバンドとは、会計、優先度設定、ワークロード管理を改善するために、エンタープライズ アプリケーションが、根底にあるTeradata データベースと連携して動作することを可能にする機能です。クエリバンドを使用すると、クエリ周辺のメタデータ(ユーザー認証情報など) をラップすることができます。

以下の2 つの変数が用意されており、そのいずれも評価の後にデータベースに送信される文字列です。

SQLSessionPrefix
この文字列は、データベースへの接続が作成されると送信されます。

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & Chr(39) & 'Who=' & OSuser() & ';' & Chr(39) & '
FOR SESSION;';
```

例えば、**OSuser()** から *WA\|sbt* が返された場合、この結果は `SET QUERY_BAND = 'Who=WA\sbt;' FOR SESSION;` に照らして評価され、接続の作成時にデータベースに送信されます。

SQLQueryPrefix
この文字列は、クエリごとに送信されます。

```
SET SQLSessionPrefix = 'SET QUERY_BAND = ' & Chr(39) & 'Who=' & OSuser() & ';' & Chr(39) & '
FOR TRANSACTION;';
```

Direct Discovery 文字変数

DirectFieldColumnDelimiter
コンマ以外の文字を項目区切り文字として使用する必要のあるデータベースについては、**Direct Query** ステートメントの項目区切り文字として使用する文字を設定できます。指定した文字は、単一引用符で囲んで **SET** ステートメントで使用します。

```
SET DirectFieldColumnDelimiter= '|'
```

DirectStringQuoteChar
生成されたクエリで文字列を引用する際に使用する文字を指定できます。既定は単一引用符です。指定した文字は、単一引用符で囲んで **SET** ステートメントで使用します。

```
SET DirectStringQuoteChar= '"';
```

DirectIdentifierQuoteStyle
生成されたクエリで使用する非 ANSI 引用符を指定できます。現時点で利用可能な非 ANSI 引用符は GoogleBQ のみとなっています。既定は、ANSIです。大文字、小文字、大文字と小文字の組み合わせ (ANSI, ansi, Ansi) を使用できます。

```
SET DirectIdentifierQuoteStyle="GoogleBQ";
```

例えば、ANSI 引用符は次のような **SELECT** ステートメントで使用します。

```
SELECT [Quarter] FROM [qvTest].[sales] GROUP BY [Quarter]
```

DirectIdentifierQuoteStyle が "GoogleBQ" に設定されている場合、**SELECT** ステートメントでは次のように引用符が使用されます。

```
SELECT [Quarter] FROM [qvTest.sales] GROUP BY [Quarter]
```

DirectIdentifierQuoteChar
生成されたクエリで文字列を引用する際に制御する文字を指定できます。1 文字 (二重引用符 1 つなど) または 2 文字 (角括弧 2 つなど) に設定可能です。既定は二重引用符です。

```
SET DirectIdentifierQuoteChar='[]';
SET DirectIdentifierQuoteChar='``';
SET DirectIdentifierQuoteChar=' ';
SET DirectIdentifierQuoteChar='""';
```

DirectTableBoxListThreshold
[**テーブル**] のビジュアライゼーションで Direct Discovery 項目が使用されている場合、表示される行数を制限するしきい値が設定されます。既定のしきい値は 1000 です。既定のしきい値は、ロードスクリプトの **DirectTableBoxListThreshold** 変数を設定することで変更できます。例:

```
SET DirectTableBoxListThreshold=5000;
```
しきい値の設定は、Direct Discovery 項目が含まれる [**テーブル**] のビジュアライゼーションにのみ適用されます。インメモリ項目だけが含まれる [**テーブル**] のビジュアライゼーションは、**DirectTableBoxListThreshold** 設定による制限を受けません。

選択数がしきい値を下回るまで、[**テーブル**] のビジュアライゼーションに項目は表示されません。

Direct Discovery データ型変換変数

DirectMoneyDecimalSep
Direct Discovery で生成したデータロード用 SQL ステートメントで使用されている通貨の小数点記号の代わりに、ここで定義した小数点記号が使用されます。この文字は、**DirectMoneyFormat** で使用されている文字と一致する必要があります。

既定値: '.'

```
Set DirectMoneyDecimalSep='.';
```

DirectMoneyFormat
Direct Discovery で生成したデータロード用 SQL ステートメントで使用されている通貨書式の代わりに、ここで定義した記号が使用されます。千の桁区切り記号を含めることはできません。

既定値: '#.0000'

```
Set DirectMoneyFormat='#.0000';
```

DirectTimeFormat
Direct Discovery で生成したデータロード用 SQL ステートメントで使用されている時間書式の代わりに、ここで定義した時間書式が使用されます。

```
Set DirectTimeFormat='hh:mm:ss';
```

DirectDateFormat
Direct Discovery で生成したデータロード用 SQL ステートメントで使用されている日付書式の代わりに、ここで定義した日付書式が使用されます。

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

DirectTimeStampFormat
Direct Discovery を使用して SQL ステートメントで生成したデータロード用 SQL ステートメントで使用されている日付および時間書式の代わりに、ここで定義した日付 •時間書式が使用されます。

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

4.10 エラー変数

エラー変数の値は、スクリプト実行後もすべて残ります。最初の変数 ErrorMode は、ユーザー入力によるもので、最後の 3 つはスクリプトのエラーに関する情報を含む Qlik Sense からのアウトプットです。

エラー変数の概要

各変数について、概要の後に詳細を説明します。構文にある変数名をクリックして、特定の変数の詳細にすぐアクセスすることもできます。

変数の詳細については、Qlik Sense オンライン ヘルプを参照してください。

ErrorMode
このエラー変数は、スクリプトの実行中にエラーが発生したときに、Qlik Sense によって実行されるアクションを定義します。

```
ErrorMode
```

ScriptError
このエラー変数は、最後に実行されたスクリプト ステートメントのエラー コードを返します。

```
ScriptError
```

ScriptErrorCount
このエラー変数は、現在のスクリプトの実行中にエラーを発生させたステートメントの総数を返します。この変数は、スクリプトの実行開始時に常に 0 にリセットされます。

```
ScriptErrorCount
```

ScriptErrorList
このエラー変数には、最後のスクリプトの実行中に発生したすべてのスクリプト エラーの連結リストが含まれます。各エラーは、改行文字 (LF) で区切られます。

```
ScriptErrorList
```

ErrorMode

このエラー変数は、スクリプトの実行中にエラーが発生したときに、Qlik Sense によって実行されるアクションを定義します。

構文:

```
ErrorMode
```

引数:

引数

引数	説明
ErrorMode=1	デフォルトの設定。スクリプトの実行が中止され、ユーザーのアクションが要求されます (バッチ モード以外)。
ErrorMode =0	Qlik Sense はエラーを無視し、スクリプトの次のステートメントから、スクリプトの実行を続行します。
ErrorMode =2	Qlik Sense は、エラーの発生直後に "Execution of script failed...(スクリプトの実行失敗)" というエラー メッセージを表示します。ユーザーに要求されるアクションはありません。

```
set ErrorMode=0;
```

ScriptError

このエラー変数は、最後に実行されたスクリプト ステートメントのエラー コードを返します。

構文:

```
ScriptError
```

この変数は、各スクリプト ステートメントが正常に実行されるたびに、0 にリセットされます。エラーが発生すると、変数は内部 Qlik Sense エラー コードに設定されます。エラー コードは、数値とテキスト値のデュアル値です。以下のようなエラー コードがあります。

スクリプト エラー コード

エラー コード	説明
0	エラーなし。デュアル値テキストは空白です。
1	一般的なエラー。
2	構文エラー。

エラーコード	説明
3	一般的な ODBC エラー。
4	一般的な OLE DB エラー。
5	一般的なカスタム データベース エラー。
6	一般エラー:「$XML」。
7	一般的な HTML エラー。
8	ファイルが見つかりません。
9	データベースが見つかりません。
10	テーブルが見つかりません。
11	項目が見つかりません。
12	ファイル形式が正しくありません。
16	セマンティック エラー。

```
set ErrorMode=0;

LOAD * from abc.qvf;

if ScriptError=8 then

exit script;

//no file;

end if
```

ScriptErrorCount

このエラー変数は、現在のスクリプトの実行中にエラーを発生させたステートメントの総数を返します。この変数は、スクリプトの実行開始時に常に 0 にリセットされます。

構文:

```
ScriptErrorCount
```

ScriptErrorList

このエラー変数には、最後のスクリプトの実行中に発生したすべてのスクリプトエラーの連結リストが含まれます。各エラーは、改行文字 (LF) で区切られます。

構文:

```
ScriptErrorList
```

5 スクリプト式

式は、**LOAD** ステートメントと**SELECT** ステートメントの両方で使用できます。ここで説明する構文と関数が適用されるのは、**LOAD** ステートメントで、**SELECT** ステートメントではありません。これは、Select ステートメントがQlik Sense ではなくODBC ドライバによって解釈されるためです。ただし、ほとんどの ODBC ドライバは、以下で説明する多くの関数を解釈できます。

数式は関数、項目、演算子を構文で組み合わせたものです。

Qlik Sense スクリプト内のすべての数式は、数値と文字列のいずれか適切なものを返します。論理関数と演算子は、False の場合は 0、True の場合は -1 を返します。数値から文字列、文字列から数値への変換は、黙示的に行われます。論理演算子と関数は、0 を False、それ以外のすべてを True と解釈します。

数式の一般的な構文は、次のとおりです。

一般的な構文

数式	項目	演算子
expression ::= (constant	constant	|
expression ::= (constant	fieldref	|
expression ::= (constant	operator1 expression	|
expression ::= (constant	expression operator2 expression	|
expression ::= (constant	function	|
expression ::= (constant	(expression)	)

ここで

- **constant** は、ストレート単一引用符で囲まれた文字列 (テキスト、日付、時刻) または数値です。定数は桁区切りなしに書かれ、小数点記号として小数点付きで書かれます。
- **fieldref** は、ロードされるテーブルの項目名です。
- **operator1** は、右側にある 1 つの数式に対して作用する単項演算子です。
- **operator2** は、両側にある 2 つの数式に対して作用する二項演算子です。
- **function ::= functionname(parameters)**

- **parameters ::= expression { , expression }**

パラメータの数と種類は任意ではなく、使用する関数によって異なります。

数式と関数は自由にネストでき、解釈可能な値を数式が返す限り、Qlik Sense はエラー メッセージを表示しません。

6 チャートの数式

チャート(ビジュアライゼーション) の数式は、関数、項目、数学演算子 (+ * / =)、その他のメジャーを組み合わせたものです。数式は、ビジュアライゼーションで確認可能な結果を生成するために、アプリでデータを処理する際に使用します。数式はメジャー以外にも使用できます。タイトルやサブタイトル、脚注、さらには軸などの数式によって、より動的で効果的なビジュアライゼーションを作成できます。

例えば、ビジュアライゼーションのタイトルを静的テキストではなく、選択内容によって結果が変わる数式から生成される動的テキストにすることもできます。

スクリプト関数とチャート関数の詳細については、スクリプト構文およびチャート関数 を参照してください。

6.1 集計範囲の定義

数式で集計の値を定義するために使用されるレコードは、通常 2 つの要素により決定されます。ビジュアライゼーションでの作業時は、次の 2 つの要素があります。

- 軸の値 (チャートの数式での集計の場合)
- 選択

この 2 つの要素によって集計範囲が決まります。選択、軸、またはその両方を無視して計算が必要となる場合があるとします。チャート関数では、TOTAL 修飾子、set 分析、またはその 2 つの組み合せを使用すると、これを実行できます。

集計: 方法と説明

方法	説明
TOTAL 修飾子	集計関数内で TOTAL 修飾子を利用すると、軸の値が無視されます。 その結果、可能性のあるすべての項目の値について集計が実行されます。 **TOTAL** 修飾子の後には、山括弧で囲んだ1つ以上の項目名のリストを続けることができます。これらの項目名は、チャート軸の変数のサブセットにする必要があります。この場合、リストされているものを除き、すべてのチャート軸の変数を無視して計算が行われます。つまり、リストされている軸項目の項目値の組み合わせごとに1つの値が返されます。また、現在、チャートの軸ではない項目もリストに含めることができます。これは、軸項目が固定されていない場合に、軸をグループ化する場合に役立ちます。グループ内の変数がすべてリストされている場合、この関数はドリルダウン レベルが変更されても機能します。
set 分析	集計内部での set 分析の使用は、選択に優先します。これにより、軸全体で分割されているすべての値について集計が行われます。

方法	説明
TOTAL 修飾子と set 分析	**TOTAL** 修飾子とset 分析の集計内部での使用は選択に優先し、軸が無視されます。

ALL 修飾子	**ALL** 修飾子を集計内部で使用する場合は、選択と軸は無視されます。これは {1} set 分析のステートメントと**TOTAL** 修飾子でも同様です。
	`=sum(All Sales)`
	`=sum({1} Total Sales)`

TOTAL 修飾子

以下は、TOTAL 修飾子を使用した相対的なシェアの計算方法を示した例です。Q2 を選択した場合、TOTAL を使用すると軸が無視され、すべての値の合計が計算されます。

例: 合計修飾子

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%

*数値をパーセントで表示するには、該当するメジャーのプロパティパネルを開き、**[数値書式]** から **[数値]** を選択し、**[書式]** から **[シンプル]** と **%** 書式の 1 つを選択します。*

set 分析

以下は、set 分析を使用した選択前のデータセット比較方法を示した例です。Q2 が選択されたと想定し、set 定義 {1} で set 分析を使用すると、あらゆる値の合計が計算され、選択は無視されますが、軸で分割されます。

例: set 分析

Year	Quarter	Sum(Amount)	Sum({1} Amount)	Sum(Amount)/Sum({1} Amount)
		3000	10800	27,8%
2012	Q1	0	1100	0%
2012	Q3	0	1400	0%
2012	Q4	0	1800	0%
2012	Q2	1700	1700	100%
2013	Q1	0	1000	0%

Year	Quarter	Sum(Amount)	Sum({1} Amount)	Sum(Amount)/Sum({1} Amount)
2013	Q3	0	1100	0%
2013	Q4	0	1400	0%
2013	Q2	1300	1300	100%

TOTAL 修飾子とset 分析

以下は、選択前に実施するset 分析とTOTAL 修飾子を組み合せたすべての軸のデータセット比較方法を示した例です。Q2 を選択した場合、set 定義 {1} とTOTAL 修飾子の set 分析を使用すると、項目選択と軸は無視され、あらゆる値の合計が計算されます。

例: TOTAL 修飾子とset 分析

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

例で使用されているデータ:

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

6.2 set 分析

アプリで選択するときは、データ内のレコードのサブセットを定義します。Sum()、Max()、Min()、Avg()、Count() などの集計関数は、このサブセットに基づいて計算されます。

つまり、選択内容によって集計の範囲が定義されます。計算が行われるレコードのセットを定義します。

set 分析は、現在の選択条件によって定義されたレコードのセットとは異なるスコープを定義する方法を提供します。この新しいスコープは、代替値選択と見なすこともできます。

これは、現在の選択を特定の値 (例えば、昨年の値や世界の市場シェア) と比較する場合に役立ちます。

set 数式

set 数式は中括弧で囲まれた集計関数内外で使用できます。

内部の set 数式

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

外部の set 数式

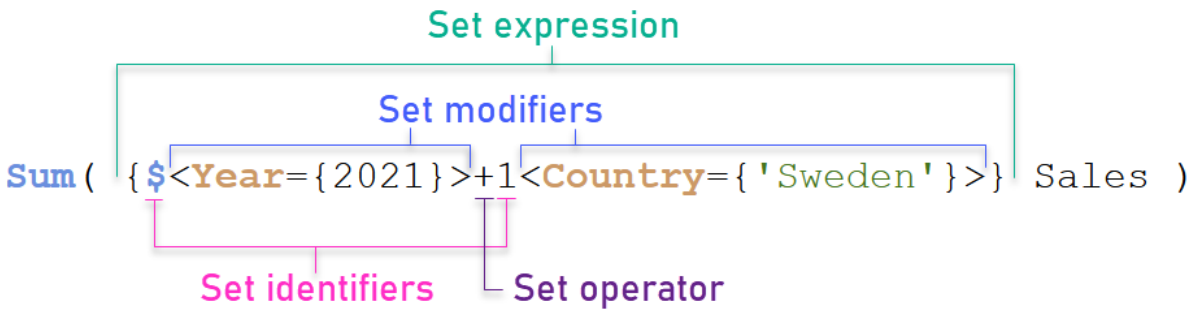
```
{<Year={2021}>} Sum(Sales) / Count(distinct Customer)
```

set 数式には、次の要素の組み合わせが含まれます。

- **識別子**set 識別子は他の場所で定義された選択を表します。また、データ内の特定のレコードセットを表します。これは、現在の選択、ブックマークからの選択、または並列ステートからの選択である可能性があります。単純な set 数式は、ドル記号 {$} など、現在の選択のすべてのレコードを意味する 1 つの識別子で構成されます。
  例: $、1、BookMark1、State2
- **演算子**set 演算子を使用して、異なる set 識別子間の和集合、差分、または共通部分を作成できます。このようにして、set 識別子によって定義された選択のサブセットまたはスーパーセットを作成できます。
  例: +、-、*、/
- **修飾子**set 修飾子を set 識別子に追加して、その選択を変更できます。修飾子は単独で使用することもでき、既定の識別子を変更します。修飾子は山括弧 <...> でくくる必要があります。
  例: <Year={2020}>、<Supplier={ACME}>

要素が組み合わされて、set 数式が形成されます。

set 数式の要素



例えば、上記の set 数式は、集計 Sum(Sales) から構築されます。

最初のオペランドは、現在の選択の 2021 年の売上を返します。これは、$ set 識別子と 2021 年の選択を含む修飾子によって示されます。2 番目のオペランドは Sweden に対して Sales を返し、1 set 識別子で示される現在の選択条件を無視します。

最後に、数式は、+ set 演算子で示されるように、2 つのセットオペランドのいずれかに属するレコードで構成されるセットを返します。

例

上記の set 数式要素を組み合わせた例は、次のトピックで利用できます。

Natural sets

通常、set 数式は、データモデル内のレコードのセットと、このデータのサブセットを定義する選択の両方を表します。この場合、セットは natural set と呼ばれます。

set 識別子は、set 修飾子の有無にかかわらず、常に natural set を表します。

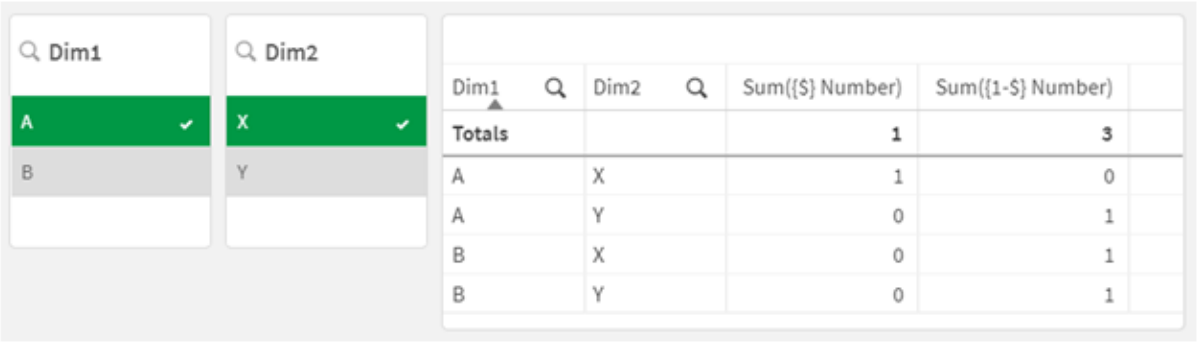
ただし、set 演算子を使用した set 数式もレコードのサブセットを表しますが、通常、項目値の選択を使用して記述することはできません。そのような数式は non-natural set です。

例えば、{1-$} によって与えられたセットは、常に選択によって定義されるとは限りません。したがって、それは natural set ではありません。これは、次のデータをロードしてテーブルに追加し、フィルター パネルを使用して選択することで表示できます。

```
Load * Inline
[Dim1, Dim2, Number
A, X, 1
A, Y, 1
B, X, 1
B, Y, 1];
```

Dim1 とDim2 を選択すると、次のテーブルに示すビューが表示されます。

natural とnon-natural set を持つテーブル



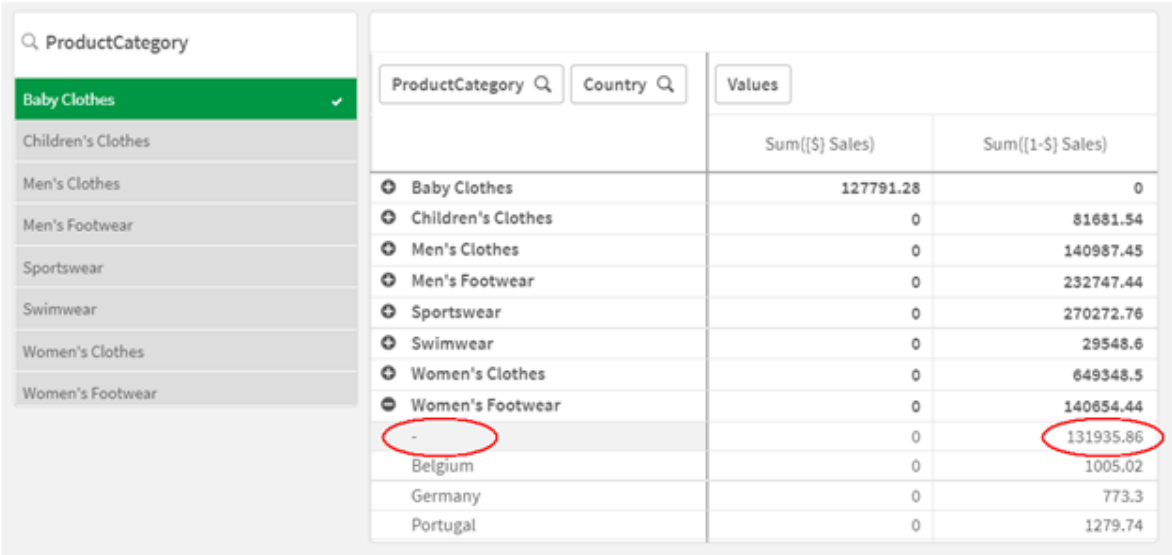
最初のメジャーの set 数式は natural set を使用します。これは、{$} で行われた選択に対応します。

2 番目のメジャーは異なります。{1-$} を使用します。このセットに対応する選択はできないため、non-natural set です。

この区別は、いくつかの結果をもたらします。

- set 修飾子は set 識別子にのみ適用できます。任意の set 数式に適用することはできません。例えば、次のような set 数式を使用することはできません。
  { (BM01 * BM02) <Field={x,y}> }
  ここで、通常の (丸) 括弧は、set 修飾子を適用する前に BM01 とBM02 の共通部分を評価する必要があることを意味します。その理由は、変更できる要素セットがないためです。
- P() および E() 要素関数内で non-natural set を使用することはできません。これらの関数は要素セットを返しますが、non-natural set から要素セットを推測することはできません。

- データモデルに多くのテーブルがある場合、non-natural set を使用するメジャーは、常に正しい軸の値に起因するとは限りません。例えば、次のチャートでは、除外された販売数の一部は正しい Country に起因しますが、他の数は NULL を Country として持っています。
  non-natural set のチャート



  割り当てが正しく行われるかどうかは、データモデルによって異なります。この場合、選択によって除外された国に関連する番号を割り当てることはできません。

識別子	説明
1	行われた選択に関係なく、アプリケーションに含まれるすべてのレコードセットを表しています。
$	現在選択されているレコードを表しています。set 数式 **{$}** は、set 数式を提示していない状態と同様です。
$1	以前の選択を表しています。$2 は以前の 1 つを除く選択を表し、以下同様に表します。
$_1	次 (将来) の選択を表しています。$_2 は次の 1 つを除く選択を表し、以下同様に表します。
BM01	任意のブックマーク ID またはブックマーク名を使用できます。
MyAltState	ステート名ごとに並列ステートで行った選択を参照できます。

例	結果
sum ({1} Sales)	アプリの sales の合計が返されます。選択は無視されますが、軸は無視されません。
sum ({$} Sales)	現在の選択の sales が返されます (sum(Sales) と同様)。
sum ({$1} Sales)	前の選択の sales が返されます。
sum ({BM01} Sales)	*BM01* という名前のブックマークの sales が返されます。

例	結果
sum({$<OrderDate = DeliveryDate>} Sales)	OrderDate = DeliveryDate の現在の選択の sales が返されます。
sum({1<Region = {US}>} Sales)	現在の選択を無視して、US 地域の sales が返されます。
sum({$<Region = >} Sales)	選択における *Region* での選択を除いた sales が返されます。
sum({<Region = >} Sales)	上記の例と同じ値が返されます。set 修飾子が省略されている場合は $ と見なされます。
sum({$<Year={2000}, Region={"U*"}>} Sales)	現在の選択条件での sales が返されますが、*Year* と *Region* で新たな選択が行われます。

set 識別子

set 識別子は、データ内のレコードのセット (すべてのデータまたはデータのサブセット) を表します。これは、選択によって定義されたレコードのセットです。これは、現在の選択、すべてのデータ (選択なし)、ブックマークからの選択、または並列ステートからの選択である可能性があります。

例 `Sum( {$<Year = {2009}>} Sales )` では、識別子はドル記号 `$` です。これは現在の選択条件を表します。また、すべての可能なレコードを表します。このセットは、set 数式の修飾子部分によって変更できます。`Year` の選択 `2009` が追加されます。

より複雑な set 数式では、2 つの識別子を演算子と一緒に使用して、2 つのレコードセットの和集合、差、または共通部分を形成できます。

次のテーブルには、いくつかの一般的な識別子が表示されています。

共通の識別子を含む例

識別子	説明
`1`	行われた選択に関係なく、アプリケーションに含まれるすべてのレコードセットを表しています。
`$`	既定のステートの現在選択されているレコードを表しています。set 数式 `{$}` は通常、set 数式を提示していない状態と同様です。
`$1`	既定の状態での前の選択を表します。`$2` は前の選択を表しますが、1 つというように続きます。
`$_1`	次の (前) 選択を表しています。`$_2` は次の 1 つを除く選択を表し、以下同様に表します。
`BM01`	任意のブックマーク ID またはブックマーク名を使用できます。
`AltState`	ステート名ごとに並列ステートを参照できます。
`AltState::BM01`	ブックマークにはすべてのステートの選択が含まれており、ブックマーク名を修飾することで特定のブックマークを参照できます。

次のテーブルは、さまざまな識別子を使用した例を示しています。

異なる識別子を含む例

例	結果
`Sum ({1} Sales)`	アプリの sales の合計が返されます。選択は無視されますが、軸は無視されません。
`Sum ({$} Sales)`	現在の選択の sales が返されます (`Sum(Sales)` と同様)。
`Sum ({$1} Sales)`	前の選択の sales が返されます。
`Sum ({BM01} Sales)`	`BM01` という名前のブックマークの sales が返されます。

set 演算子

set 演算子は、データセットを含めたり、除外したり、交差させたりするために使用されます。すべての演算子はオペランドとして set を使用し、結果として set を返します。

set 演算子は、次の 2 つの異なる状況で使用できます。

- データ内のレコードのセットを表す、セット識別子に対して set 演算を実行する。
- 要素セット、項目値、または set 修飾子内で set 演算を実行する。

次のテーブルは、set 数式で使用できる演算子を示しています。

演算子

演算子	説明
+	Union。この二項演算子は、2 つの SET オペランドのいずれかに属するレコードまたは要素を含むセットを返します。
-	Exclusion。この二項演算子は、2 つのうち最初の SET オペランドにのみ属するレコードまたは要素を含むセットを返します。また、単項演算子として使用する場合は、補集合を返します。
*	Intersection。この二項演算子は、両方の SET オペランドに属するレコードまたは要素を含むセットを返します。
/	対称差演算子 (XOR)。この二項演算子は、2 つの SET オペランドのどちらかに属している (両方には属していない) レコードまたは要素を含むセットを返します。

次のテーブルは、演算子を使用した例を示しています。

演算子の例

例	結果
`Sum ({1-$} Sales)`	現在の選択条件によって除外された、あらゆる sales を返します。
`Sum ({$*BM01} Sales)`	選択とブックマーク#160;`BM01` の共有部分における sales を返します。
`Sum ({-($+BM01)} Sales)`	選択とブックマーク `BM01` によって除外された sales を返します。
`Sum ({$<Year=`	現在の選択に関連する 2009 年の売上を返し、すべての年で国

例	結果
{2009}>+1<Country={'Sweden'}>} Sales)	Sweden に関連するデータセット全体を追加します。
Sum ({$<Country={"S*"}+{"*land"}>} Sales)	「S」で始まる国または「land」で終わる国の売上を返します。

set 修飾子

set 数式は計算の範囲を定義するために使用されます。set 数式の中心部分は、選択を指定する set 修飾子です。これは、ユーザーの選択、または set 識別子の選択を変更するために使用され、結果は計算の新しいスコープを定義します。

set 修飾子は、1 つまたは複数の項目名で構成されます。各項目名の後には、項目で実行する必要がある選択が続きます。修飾子は山括弧 (< >)でくくります: < >

例:

- Sum ({$<Year = {2015}>} Sales)
- Count ({1<Country = {Germany}>} distinct OrderID)
- Sum ({$<Year = {2015}, Country = {Germany}>} Sales)

要素セット

要素セットは、以下を使用して定義できます。

- 値のリスト
- 検索
- 別の項目への参照
- set 関数

要素セットの定義を省略した場合、set 修飾子はこの項目の選択をすべてクリアします。例:

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

例: 要素セットに基づくset 修飾子のチャートの数式

例 - チャートの数式

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
MyTable:
Load * Inline [
Country, Year, Sales
Argentina, 2014, 66295.03
Argentina, 2015, 140037.89
Austria, 2014, 54166.09
```

```
Austria, 2015, 182739.87
Belgium, 2014, 182766.87
Belgium, 2015, 178042.33
Brazil, 2014, 174492.67
Brazil, 2015, 2104.22
Canada, 2014, 101801.33
Canada, 2015, 40288.25
Denmark, 2014, 45273.25
Denmark, 2015, 106938.41
Finland, 2014, 107565.55
Finland, 2015, 30583.44
France, 2014, 115644.26
France, 2015, 30696.98
Germany, 2014, 8775.18
Germany, 2015, 77185.68
];
```

チャートの数式

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

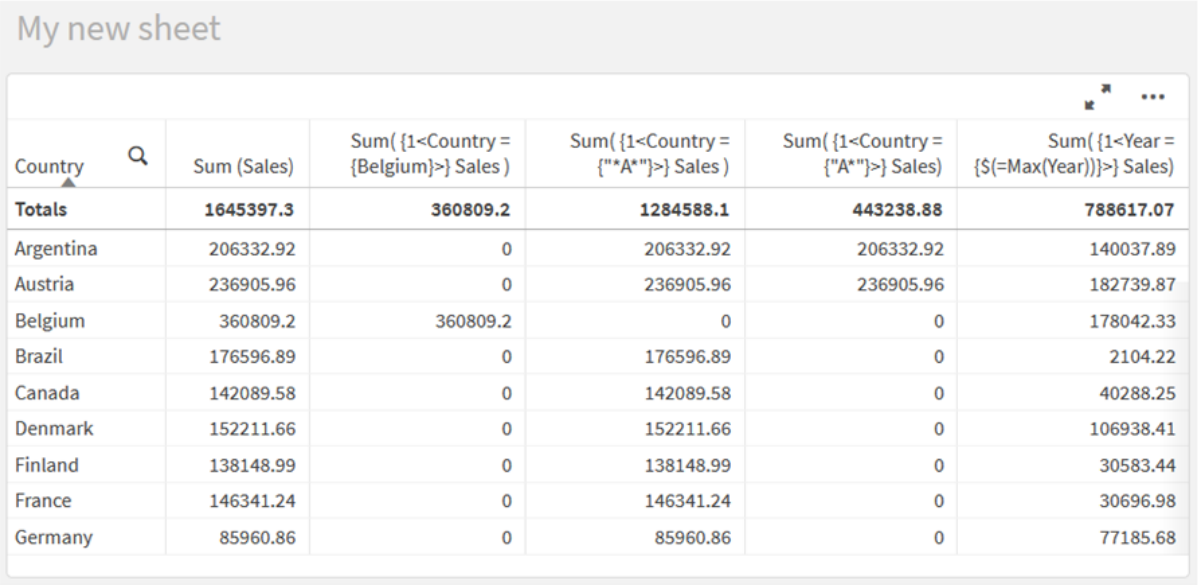
テーブル - 要素セットに基づくset修飾子

国	Sum(Sales)	Sum ({1<Country={Belgium}>} Sales)	Sum ({1<Country={"*A*"}>} Sales)	Sum ({1<Country={"A*"}>} Sales)	Sum ({1<Year={$(=Max(Year))}>} Sales)
合計	1645397.3	360809.2	1284588.1	443238.88	788617.07
アルゼンチン	206332.92	0	206332.92	206332.92	140037.89
オーストリア	236905.96	0	236905.96	236905.96	182739.87
ベルギー	360809.2	360809.2	0	0	178042.33
ブラジル	176596.89	0	176596.89	0	2104.22
カナダ	142089.58	0	142089.58	0	40288.25
デンマーク	152211.66	0	152211.66	0	106938.41
フィンランド	138148.99	0	138148.99	0	30583.44
フランス	146341.24	0	146341.24	0	30696.98
ドイツ	85960.86	0	85960.86	0	77185.68

説明

- 軸:
  - Country
- 数式:
  - Sum(Sales)
    set 数式なしで Sales を合計します。
  - Sum({1<Country={Belgium}>}Sales)
    Belgium を選択し、対応する Sales を合計します。
  - Sum({1<Country={"*A*"}>}Sales)
    A がある国をすべて選択し、対応する Sales を合計します。
  - Sum({1<Country={"A*"}>}Sales)
    A で始まるすべての国を選択し、対応する Sales を合計します。
  - Sum({1<Year={$(=Max(Year))}>}Sales)
    2015 である Max(Year) を計算し、対応する Sales を合計します。

要素セットに基づくset 修飾子



Country	Sum (Sales)	Sum({1<Country = {Belgium}>} Sales)	Sum({1<Country = {"*A*"}>} Sales)	Sum({1<Country = {"A*"}>} Sales)	Sum({1<Year = {$(=Max(Year))}>} Sales)
Totals	**1645397.3**	**360809.2**	**1284588.1**	**443238.88**	**788617.07**
Argentina	206332.92	0	206332.92	206332.92	140037.89
Austria	236905.96	0	236905.96	236905.96	182739.87
Belgium	360809.2	360809.2	0	0	178042.33
Brazil	176596.89	0	176596.89	0	2104.22
Canada	142089.58	0	142089.58	0	40288.25
Denmark	152211.66	0	152211.66	0	106938.41
Finland	138148.99	0	138148.99	0	30583.44
France	146341.24	0	146341.24	0	30696.98
Germany	85960.86	0	85960.86	0	77185.68

リストされた値

要素セットの最も一般的な例は、中括弧で囲まれた項目値のリストに基づく数式です。例:

- {$<Country = {Canada, Germany, Singapore}>}
- {$<Year = {2015, 2016}>}

内側の中括弧は要素セットを定義します。個々の値はコンマで区切られます。

引用符と大文字と小文字の区別

値にブランクまたは特殊文字が含まれている場合は、値を引用符で囲む必要があります。一重引用符は、単一の項目値と文字通りの大文字と小文字を区別して一致します。二重引用符は、1つまたは複数の項目値と大文字と小文字を区別しない一致を意味します。例:

- `<Country = {'New Zealand'}>`
  New Zealand のみに一致します。
- `<Country = {"New Zealand"}>`
  New Zealand、NEW ZEALAND、new zealand に一致します。

日付は引用符で囲み、問題の項目の日付形式を使用する必要があります。例:

- `<ISO_Date = {'2021-12-31'}>`
- `<US_Date = {'12/31/2021'}>`
- `<UK_Date = {'31/12/2021'}>`

二重引用符は角括弧またはアクサングラーブで置き換えることができます。

検索

要素セットは検索によって作成することもできます。例:

- `<Country = {"C*"}>`
- `<Ingredient = {"*garlic*"}>`
- `<Year = {">2015"}>`
- `<Date = {">12/31/2015"}>`

ワイルドカードはテキスト検索で使用できます。アスタリスク (*) は任意の数の文字を表し、疑問符 (?) は単一の文字を表します。関係演算子を使用して、数値検索を定義できます。

検索には常に二重引用符を使用する必要があります。検索値では、大文字と小文字が区別されます。

ドル展開

要素セット内で計算を使用する場合は、ドル展開が必要です。例えば、可能性のある最後の年のみを確認する場合は、次を使用できます:

```
<Year = {$(=Max(Year))}>
```

他の項目で選択された値

修飾子は、別の項目の選択された値に基づく場合もあります。例:

```
<OrderDate = DeliveryDate>
```

この修飾子は、DeliveryDate から選択値を取得し、OrderDate に適用します。200 以上の多くの固有値がある場合、この操作は CPU を集中して使用するため、行わないでください。

要素セット関数

要素セットは、設定された関数 P() (可能な値) および E() (除外された値) に基づくこともできます。

例えば、製品 Cap が販売されている国を選択する場合は、次を使用できます。

```
<Country = P({1<Product={Cap}>} Country)>
```

同様に、製品 Cap が販売されていない国を選択する場合は、次を使用できます。

```
<Country = E({1<Product={Cap}>} Country)>
```

検索を使った set 修飾子

set 修飾子を使用した検索により、要素セットを作成できます。

例:

- <Country = {"C*"}>
- <Year = {">2015"}>
- <Ingredient = {"*garlic*"}>

検索は常に二重引用符、角括弧、またはアクサングラーブで囲む必要があります。リテラル文字列 (一重引用符) と検索 (二重引用符) を組み合わせたリストを使用できます。例:

```
<Product = {'Nut', "*Bolt", Washer}>
```

テキスト検索

ワイルドカードやその他の記号は、テキスト検索で使用できます。

- アスタリスク (*) は任意の数の文字を表します。
- 疑問符 (?) は 1 つの文字を表します。
- 曲折アクセント記号 (^) は単語の始まりを示します。

例:

- <Country = {"C*", "*land"}>
  C で始まり land で終わるすべての国に一致します。
- <Country = {"*^z*"}>
  これは、New Zealand などの z で始まる単語を持つすべての国に一致します。

数値の検索

次の関係演算子を使用して数値の検索を行うことができます: >、>=、<、<=

数値の検索は、常にこれらの演算子の 1 つで始まります。例:

- <Year = {">2015"}>
  2016 年以降の年に一致します。
- <Date = {">=1/1/2015<1/1/2016"}>
  2015 年のすべての日付に一致します。2 つの日付の間の時間範囲を記述するための構文に注意してください。日付形式は、問題の項目の日付形式と一致する必要があります。

数式の検索

数式の検索を使用して、より高度な検索を行うことができます。次に、検索フィールドの項目値ごとに集計が評価されます。検索の数式が true を返すすべての値が選択されます。

数式検索は、必ず等号で開始します: =

例:

```
<Customer = {"=Sum(Sales)>1000"}>
```

これにより、売上高が1000 を超えるすべての顧客が返されます。Sum(Sales) は現在の選択条件に基づいて計算されます。つまり、Product 項目など、別の項目で選択した場合、選択した製品の販売条件を満たす顧客のみを取得できます。

条件を選択から独立させる場合は、検索文字列内で set 分析を使用する必要があります。例:

```
<Customer = {"=Sum({1} Sales)>1000"}>
```

等号の後の数式はブール値として解釈されます。これは、他の何かに評価された場合、ゼロ以外の数値は true として解釈され、ゼロおよび文字列は false として解釈されることを意味します。

引用符

検索文字列に空白または特殊文字が含まれている場合は、引用符を使用してください。一重引用符は、単一の項目値との文字通りの大文字と小文字を区別する一致を意味します。二重引用符は、複数の項目値に一致する可能性のある大文字と小文字を区別しない検索を意味します。

例:

- `<Country = {'New Zealand'}>`
  New Zealand のみに一致します。
- `<Country = {"New Zealand"}>`
  New Zealand、NEW ZEALAND、new zealand に一致します

二重引用符は角括弧またはアクサングラーブで置き換えることができます。

> *Qlik Sense の以前のバージョンでは、単一引用符と二重引用符の区別はなく、引用符で囲まれたすべての文字列が検索値として扱われていました。下位互換性を維持するため、旧バージョンの Qlik Sense で作成されたアプリは、前バージョンでの動作と同様に動作し続けます。Qlik Sense November 2017 以降で作成されたアプリは、2 種類の引用符の違いを認識します。*

例: 検索を使用した set 修飾子のチャートの数式

例 - チャートの数式

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
MyTable:
Load
Year(Date) as Year,
Date#(Date,'YYYY-MM-DD') as ISO_Date,
Date(Date#(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,
Country, Product, Amount
```

```
Inline
[Date, Country, Product, Amount
2018-02-20, Canada, Washer, 6
2018-07-08, Germany, Anchor bolt, 10
2018-07-14, Germany, Anchor bolt, 3
2018-08-31, France, Nut, 2
2018-09-02, Czech Republic, Bolt, 1
2019-02-11, Czech Republic, Bolt, 3
2019-07-31, Czech Republic, Washer, 6
2020-03-13, France, Anchor bolt, 1
2020-07-12, Canada, Anchor bolt, 8
2020-09-16, France, Washer, 1];
```

例 1: テキスト検索を使用したチャートの数式。

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

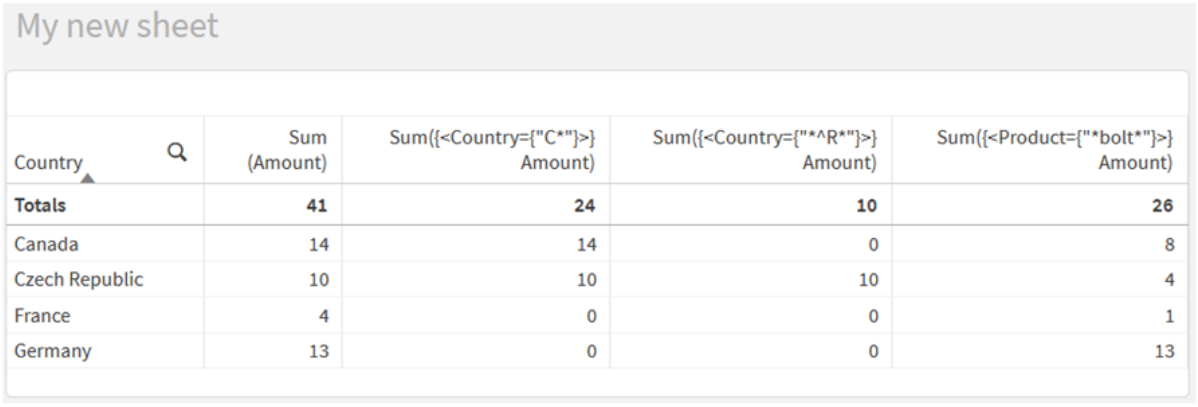
テーブル - テキスト検索を使った set 修飾子

国	Sum (Amount)	Sum({<Country={"C*"}>} Amount)	Sum({<Country={"*^R*"}>} Amount)	Sum({<Product={"*bolt*"}>} Amount)
合計	**41**	**24**	**10**	**26**
カナダ	14	14	0	8
チェコ共和国	10	10	10	4
フランス	4	0	0	1
ドイツ	13	0	0	13

説明

- 軸:
  - `Country`
- 数式:
  - `Sum(Amount)`
    set 数式なしで `Amount` を合計します。
  - `Sum({<Country={"C*"}>}Amount)`
    `Canada` や `Czech Republic` など、`C` で始まるすべての国について `Amount` を合計します。
  - `Sum({<Country={"*^R*"}>}Amount)`
    `Czech Republic` など、`R` で始まる単語があるすべての国について `Amount` を合計します。
  - `Sum({<Product={"*bolt*"}>}Amount)`
    `Bolt` や `Anchor bolt` など、文字列 `bolt` を含むすべての製品について `Amount` を合計します。

テキスト検索を使った set 修飾子



例 2: 数値の検索を使用したチャートの数式。

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

テーブル - 数値の検索を使用した set 修飾子

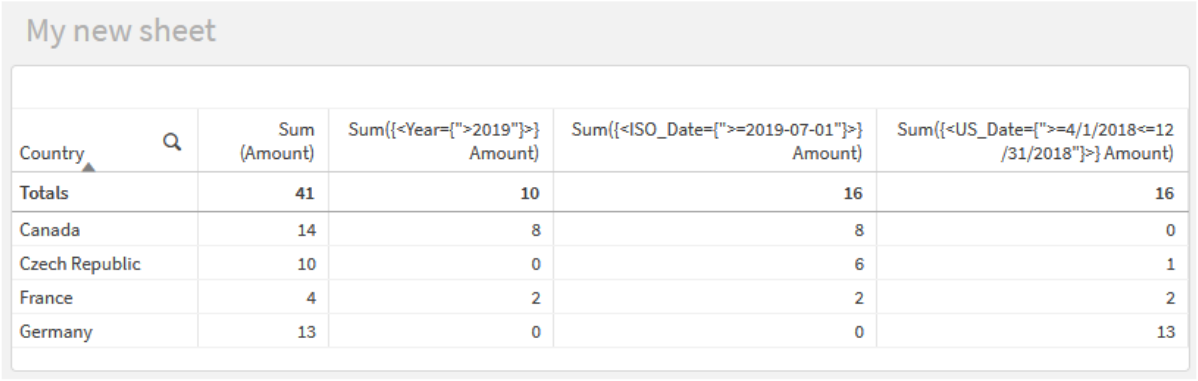
国	Sum (Amount)	Sum({<Year={">2019"}>} Amount)	Sum({<ISO_Date={">=2019-07-01"}>} Amount)	Sum({<US_Date={">=4/1/2018<=12/31/2018"}>} Amount)
合計	**41**	**10**	**16**	**16**
カナダ	14	8	8	0
チェコ共和国	10	0	6	1
フランス	4	2	2	2
ドイツ	13	0	0	13

説明

- 軸:
  - `Country`
- 数式:
  - `Sum(Amount)`
    set 数式なしで `Amount` を合計します。
  - `Sum({<Year={">2019"}>}Amount)`
    2019 以降のすべての年の場合は Sum `Amount` です。
  - `Sum({<ISO_Date={">=2019-07-01"}>}Amount)`
    2019-07-01 以降のすべての日付の場合は Sum `Amount` です。検索での日付の形式は、項目の形式と一致する必要があります。

- Sum({<US_Date={">=4/1/2018<=12/31/2018"}>}Amount)
  開始日と終了日を含む、4/1/2018 から 12/31/2018 までのすべての日付の場合は Sum Amount です。検索での日付の形式は、項目の形式と一致する必要があります。

数値の検索を使用した set 修飾子



例 3: 数式の検索を使用したチャートの数式

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

Table - Set modifiers with expression searches

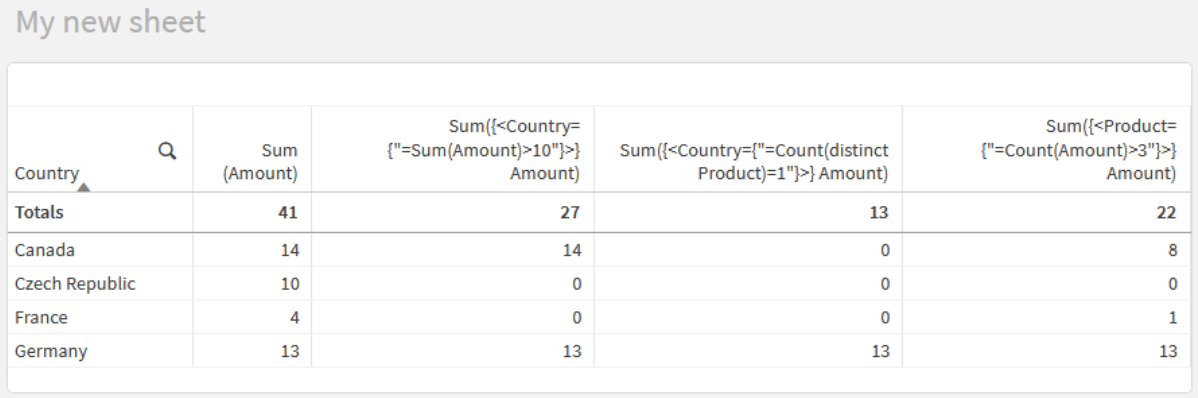
Country	Sum (Amount)	Sum({<Country={"=Sum (Amount)>10"}>} Amount)	Sum({<Country={"=Count(distinct Product)=1"}>} Amount)	Sum({<Product={"=Count (Amount)>3"}>} Amount)
Totals	**41**	**27**	**13**	**22**
Canada	14	14	0	8
Czech Republic	10	0	0	0
France	4	0	0	1
Germany	13	13	13	13

説明

- 軸:
  - Country
- 数式:
  - Sum(Amount)
    set 数式なしで Amount を合計します。
  - Sum({<Country={"=Sum(Amount)>10"}>}Amount)
    Amount の合計が 10 より大きいすべての国の場合は Sum Amount です。
  - Sum({<Country={"=Count(distinct Product)=1"}>}Amount)
    正確に 1 つの異なる製品に関連付けられているすべての国の場合は Sum Amount です。

- `Sum({<Product={"=Count(Amount)>3"}>}Amount)`
  データに 3 つ以上のトランザクションがあるすべての国の場合は Sum `Amount` です。

数式の検索を使用した set 修飾子



Country	Sum (Amount)	Sum({<Country={"=Sum(Amount)>10"}>} Amount)	Sum({<Country={"=Count(distinct Product)=1"}>} Amount)	Sum({<Product={"=Count(Amount)>3"}>} Amount)
Totals	**41**	**27**	**13**	**22**
Canada	14	14	0	8
Czech Republic	10	0	0	0
France	4	0	0	1
Germany	13	13	13	13

例	結果
sum({$–1<Product = {"*Internal*", "*Domestic*"}>} Sales)	現在の選択条件から製品 (Product) が文字列「Internal」もしくは「Domestic」を含む取引を除外した売上 (Sales) の合計が返されます。
sum({$<Customer = {"=Sum ({1<Year = {2007}>} Sales) > 1000000"}>} Sales)	現在の選択に Customer への新規選択 (2007 年の売上合計が 1,000,000 以上の顧客のみ) を追加した売上 (Sales) の合計が返されます。

ドル記号展開を使った set 修飾子

ドル記号展開は、数式が解析および評価される前に計算される構成です。次に、結果が `$(...)` の代わりに数式に挿入されます。数式の計算は、ドル展開の結果を使用して行われます。

数式エディタにはドル展開のプレビューが表示されるため、ドル記号展開が何に評価されるかを確認できます。

数式エディタでのドル記号展開プレビュー

要素セット内で計算を使用する場合は、ドル記号展開を使用します。

例えば、可能な最後の年のみを確認する場合は、次の構造を使用できます。

```
<Year = {$(=Max(Year))}>
```

Max(Year) が最初に計算され、結果が $(…) の代わりに数式に挿入されます。

ドル展開後の結果は、次のような数式になります。

```
<Year = {2021}>
```

ドル展開内の数式は、現在の選択条件に基づいて計算されます。これは、別の項目で選択した場合、数式の結果が影響を受けることを意味します。

計算を選択から独立させる場合は、ドル展開内で set 分析を使用します。例:

```
<Year = {$(=Max({1} Year))}>
```

文字列

ドル展開で文字列を作成する場合は、通常の見積もりルールが適用されます。例:

```
<Country = {'$(=FirstSortedValue(Country,Date)'}>
```

ドル展開後の結果は、次のような数式になります。

```
<Country = {'New Zealand'}>
```

引用符を使用しない場合、構文エラーが発生します。

数値

ドル展開で数値を作成する場合は、展開が項目と同じ書式設定になるようにします。これは、数式を書式設定関数でラップする必要がある場合があることを意味します。

例:

```
<Amount = {$(=Num(Max(Amount), '###0.00'))}>
```

ドル展開後の結果は、次のような数式になります。

```
<Amount = {12362.00}>
```

ハッシュを使用して、展開で常に小数点を使用し、千単位の区切り文字を使用しないようにします。例:

```
<Amount = {$(#=Max(Amount))}>
```

日付

ドル展開で日付を作成する場合は、拡張の書式設定が正しいことを確認してください。これは、数式を書式設定関数でラップする必要がある場合があることを意味します。

例:

```
<Date = {'$(=Date(Max(Date)))'}>
```

ドル展開後の結果は、次のような数式になります。

```
<Date = {'12/31/2015'}>
```

文字列の場合と同様に、正しい引用符を使用する必要があります。

一般的な使用例は、計算を先月 (または年) に制限することです。次に、AddMonths() 関数と組み合わせて数値の検索を使用できます。

例:

```
<Date = {">=$(=AddMonths(Today(),-1))"}>
```

ドル展開後の結果は、次のような数式になります。

```
<Date = {">=9/31/2021"}>
```

これにより、先月発生したすべてのイベントが選択されます。

例: ドル記号展開を使用した set 修飾子のチャートの数式

例 - チャートの数式

ロードスクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
Let vToday = Today();
MyTable:
Load
Year(Date) as Year,
Date#(Date,'YYYY-MM-DD') as ISO_Date,
Date(Date#(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,
Country, Product, Amount
Inline
[Date, Country, Product, Amount
2018-02-20, Canada, Washer, 6
2018-07-08, Germany, Anchor bolt, 10
2018-07-14, Germany, Anchor bolt, 3
2018-08-31, France, Nut, 2
2018-09-02, Czech Republic, Bolt, 1
2019-02-11, Czech Republic, Bolt, 3
2019-07-31, Czech Republic, Washer, 6
2020-03-13, France, Anchor bolt, 1
2020-07-12, Canada, Anchor bolt, 8
2021-10-15, France, Washer, 1];
```

ドル記号展開を使用したチャートの数式

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

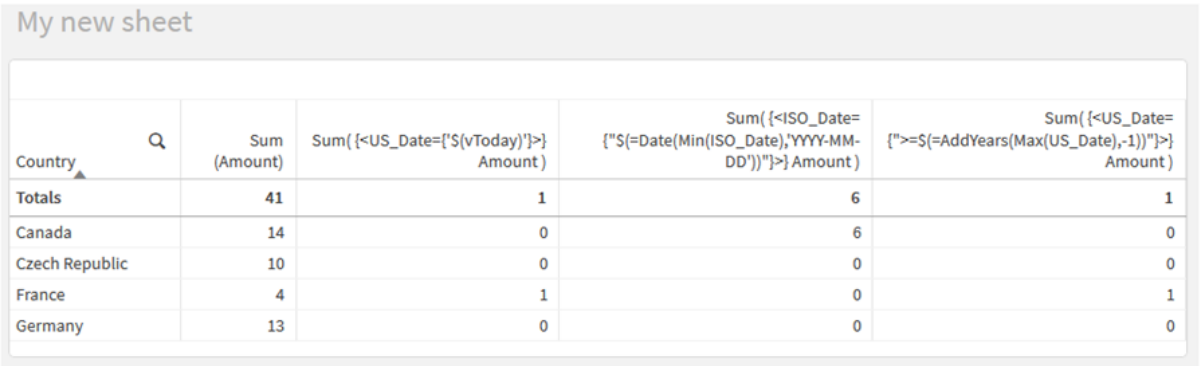
テーブル - ドル記号展開を使った set 修飾子

国	Sum (Amount)	Sum({<US_Date={'$(vToday)'}>} Amount)	Sum({<ISO_Date={"$(=Date(Min(ISO_Date),'YYYY-MM-DD'))"}>} Amount)	Sum({<US_Date={">=$(=AddYears(Max(US_Date),-1))"}>} Amount)
合計	**41**	**1**	**6**	**1**
カナダ	14	0	6	0
チェコ共和国	10	0	0	0
フランス	4	1	0	1
ドイツ	13	0	0	0

説明

- 軸:
  - `Country`
- 数式:
  - `Sum(Amount)`
    set 数式なしで `Amount` を合計します。
  - `Sum({<US_Date={'$(vToday)'}>}Amount)`
    `US_Date` が変数 `vToday` と同じであるすべてのレコードの場合は Sum `Amount` です。
  - `Sum({<ISO_Date={"$(=Date(Min(ISO_Date),'YYYY-MM-DD'))"}>}Amount)`
    `ISO_Date` が最初の (最小の) 可能な `ISO_Date` と同じであるすべてのレコードの場合は Sum `Amount` です。日付の形式が項目の形式と一致するようにするには、`Date()` 関数が必要です。
  - `Sum({<US_Date={">=$(=AddYears(Max(US_Date),-1))"}>}Amount)`
    最新の (最大の) 可能な `US_Date` の 1 年前または 1 年前の日付に `US_Date` があるすべてのレコードの場合は Sum `Amount` です。`AddYears()` 関数は、変数 `DateFormat` で指定された形式で日付を返します。これは、項目 `US_Date` の形式と一致する必要があります。

ドル記号展開を使った set 修飾子



例	結果
sum({$<Year = {$(#vLastYear)}>} Sales)	現在の選択に関連した、前年の sales が返されます。ここでは、対応する年を含む変数 vLastYear がドル記号展開に使用されています。
sum({$<Year = {$(#=Only(Year)-1)}>} Sales)	現在の選択に関連した、前年の sales が返されます。ここでは、前年を計算するためにドル記号展開が使用されています。

set 演算子を使った set 修飾子

set 演算子は、さまざまな要素セットを含めたり、除外したり、交差させたりするために使用されます。これらは、さまざまなメソッドを組み合わせて要素セットを定義します。

演算子は set 識別子に使用されるものと同じです。

演算子

演算子	説明
+	Union。この二項演算子は、2 つの SET オペランドのいずれかに属するレコードまたは要素を含むセットを返します。
-	Exclusion。この二項演算子は、2 つのうち最初の SET オペランドにのみ属するレコードまたは要素を含むセットを返します。また、単項演算子として使用する場合は、補集合を返します。
*	Intersection。この二項演算子は、両方の SET オペランドに属するレコードまたは要素を含むセットを返します。
/	対称差演算子 (XOR)。この二項演算子は、2 つの SET オペランドのどちらかに属している (両方には属していない) レコードまたは要素を含むセットを返します。

例えば、次の 2 つの修飾子は、同じ項目値のセットを定義します。

- `<Year = {1997, "20*"}>`
- `<Year = {1997} + {"20*"}>`

どちらの数式も 1997 年と 20 で始まる年を選択します。言い換えれば、これは 2 つの条件の和集合です。

set 演算子を使用すると、より複雑な定義も可能になります。例:

```
<Year = {1997, "20*"} - {2000}>
```

この数式は、上記と同じ年を選択しますが、さらに 2000 年を除外します。

例: set 演算子を使用した set 修飾子のチャートの数式

例 - チャートの数式

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
MyTable:
Load
Year(Date) as Year,
Date#(Date,'YYYY-MM-DD') as ISO_Date,
Date(Date#(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,
Country, Product, Amount
Inline
[Date, Country, Product, Amount
2018-02-20, Canada, Washer, 6
2018-07-08, Germany, Anchor bolt, 10
2018-07-14, Germany, Anchor bolt, 3
2018-08-31, France, Nut, 2
2018-09-02, Czech Republic, Bolt, 1
2019-02-11, Czech Republic, Bolt, 3
2019-07-31, Czech Republic, Washer, 6
2020-03-13, France, Anchor bolt, 1
2020-07-12, Canada, Anchor bolt, 8
2020-09-16, France, Washer, 1];
```

チャートの数式

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

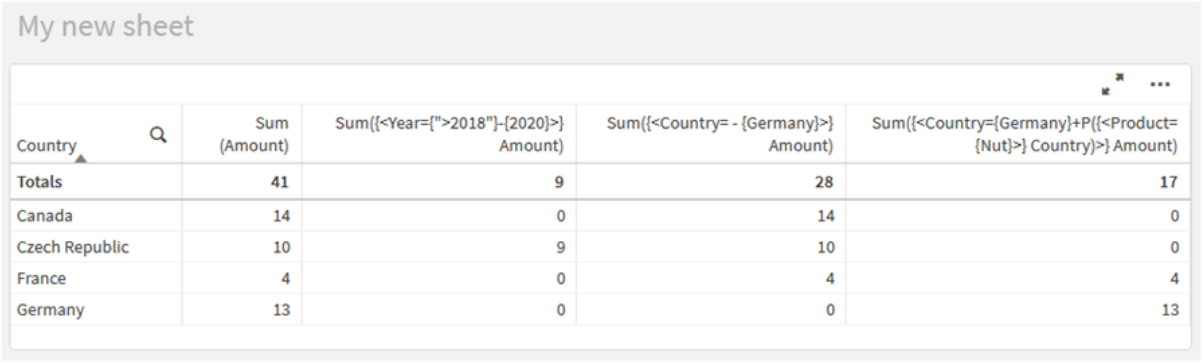
テーブル - set 演算子を使った set 修飾子

国	Sum (Amount)	Sum({<Year={">2018"}-{2020}>} Amount)	Sum ({<Country=-{Germany}>} Amount)	Sum({<Country={Germany}+P({<Product={Nut}>}Country)>} Amount)
合計	**41**	**9**	**28**	**17**
カナダ	14	0	14	0
チェコ共和国	10	9	10	0
フランス	4	0	4	4
ドイツ	13	0	0	13

説明

- 軸:
  - Country
- 数式:
  - Sum(Amount)
    set 数式なしで Amount を合計します。
  - Sum({<Year={">2018"}-{2020}>}Amount)
    2020 を除く2018 以降のすべての年の場合は Sum Amount です。
  - Sum({<Country=-{Germany}>}Amount)
    Germany を除くすべての国の場合は Sum Amount です。単項排他的演算子に注意してください。
  - Sum({<Country={Germany}+P({<Product={Nut}>}Country)>}Amount)
    Germany および製品 Nut に関連するすべての国の場合は Sum Amount です。

set 演算子を使った set 修飾子



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

例	結果
sum({$<Product = Product + {OurProduct1} – {OurProduct2} >} Sales)	現在の選択に製品 (Product) の OurProduct1 を追加し、OurProduct2 を削除した sales が返されます。
sum({$<Year = Year + ({“20*”,1997} – {2000}) >} Sales)	現在の選択に対する sales が返されますが、「Year」項目で追加の選択が行われます (1997 および 2000 以外の 20 から始まるすべての Year)。 2000 が現在の選択に含まれる場合、修飾子の後にもこの値が含まれることに注意してください。
sum({$<Year = (Year + {“20*”,1997}) – {2000} >} Sales)	上記とほぼ同じ結果を返しますが、ここでは現在の選択に 2000 が含まれていても変更後は削除されます。この例が示すように、優先順位を定義するために括弧が必要となる場合があります。
sum({$<Year = {“*”} – {2000}, Product = {“*bearing*”} >} Sales)	現在の選択条件に、[Year] における新規選択: 2000を除くすべての年、および、文字列 bearing を含む製品のみを追加した売上が返されます。

暗黙の set 演算子を使った set 修飾子

set 修飾子に選択を書き込む標準的な方法は、等号を使用することです。例:

```
Year = {">2015"}
```

set 修飾子の等号の右側の数式は、要素セットと呼ばれます。これは、一連の個別の項目値、つまり選択を定義します。

この表記は、項目内の現在の選択を無視し、新しい選択を定義します。したがって、set 識別子にこの項目の選択が含まれている場合、古い選択は要素セットの選択に置き換えられます。

項目の現在の選択に基づいて選択する場合は、別の数式を使用する必要があります

例えば、古い選択を尊重し、年が 2015 年以降であるという要件を追加する場合は、次のように記述できます。

```
Year = Year * {">2015"}
```

アスタリスクは交差を定義する set 演算子であるため、Year での現在の選択条件と、年が 2015 年以降という追加要件との間の交差を取得します。これを書く別の方法は次のとおりです。

```
Year *= {">2015"}
```

つまり、代入演算子 (*=) は暗黙的に交差を定義します。

同様に、暗黙の和集合、除外、対称差は、次を使用して定義できます: +=、-=、/=

例: 暗黙の set 演算子を使用した set 修飾子のチャートの数式

例 - チャートの数式

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
MyTable:
Load
Year(Date) as Year,
Date#(Date,'YYYY-MM-DD') as ISO_Date,
Date(Date#(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,
Country, Product, Amount
Inline
[Date, Country, Product, Amount
2018-02-20, Canada, Washer, 6
2018-07-08, Germany, Anchor bolt, 10
2018-07-14, Germany, Anchor bolt, 3
2018-08-31, France, Nut, 2
2018-09-02, Czech Republic, Bolt, 1
2019-02-11, Czech Republic, Bolt, 3
2019-07-31, Czech Republic, Washer, 6
```

```
2020-03-13, France, Anchor bolt, 1
2020-07-12, Canada, Anchor bolt, 8
2020-09-16, France, Washer, 1];
```

暗黙の set 演算子を使用したチャートの数式

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

国のリストからCanada とCzech Republic を選択します。

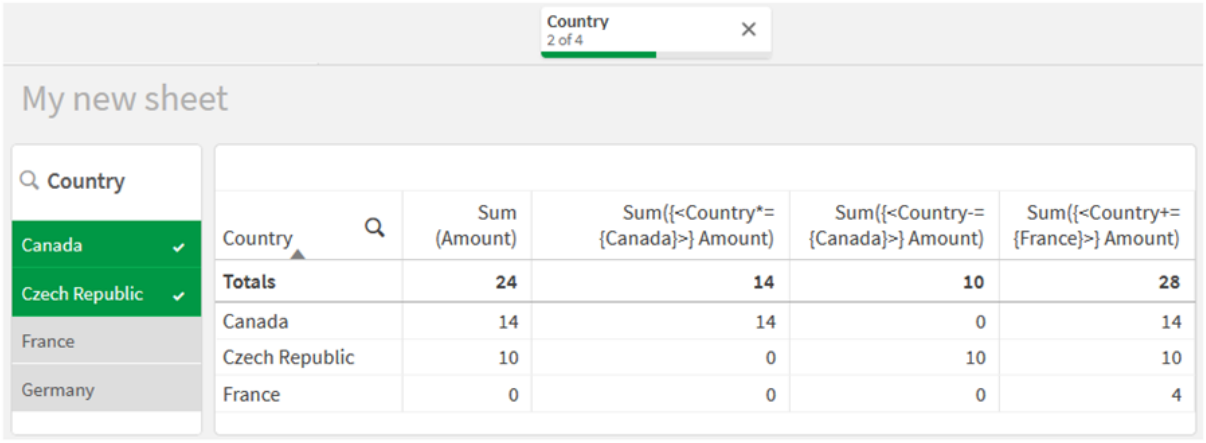
テーブル - 暗黙の set 演算子を使用したチャートの数式

国	Sum (Amount)	Sum({<Country*= {Canada}>} Amount)	Sum({<Country-= {Canada}>} Amount)	Sum({<Country+= {France}>} Amount)
合計	**24**	**14**	**10**	**28**
カナダ	14	14	0	14
チェコ共和国	10	0	10	10
フランス	0	0	0	4

説明

- 軸:
  - Country
- 数式:
  - Sum(Amount)
    現在の選択条件の場合は Sum Amount です。Canada とCzech Republic のみがゼロ以外の値を持つことに注意してください。
  - Sum({<Country*={Canada}>}Amount)
    現在の選択条件の合計 Amount は、Country が Canada であるという要件と交差しています。Canada がユーザー選択の一部でない場合、set 数式は空のセットを返し、列はすべての行で 0 になります。
  - Sum({<Country-={Canada}>}Amount)
    現在の選択条件では Sum Amount ですが、最初に Country の選択からCanada を除外します。Canada がユーザー選択の一部でない場合、set 数式は数値を変更しません。
  - Sum({<Country+={France}>}Amount)
    現在の選択条件では Sum Amount ですが、最初に Country の選択に France を追加します。France が既にユーザー選択の一部である場合、set 数式は数値を変更しません。

暗黙の set 演算子を使った set 修飾子



例	結果
sum({$<Product += {OurProduct1, OurProduct2} >} Sales)	現在の選択に対して、製品 (Product) の OurProduct1 と OurProduct2 を追加した sales が返されます。
sum({$<Year += {"20*",1997} – {2000} >} Sales)	現在の選択に対して sales が返されますが、黙示的な union を用いて年が追加されます (1997 と、2000 以外の 20 から始まるすべての Year)。 2000 が現在の選択に含まれる場合、修飾子の後にもこの値が含まれることに注意してください。これは、<Year=Year + ({"20*",1997}–{2000})> と同様です。
sum({$<Product *= {OurProduct1} >} Sales)	現在の選択のうち、Product の現在の選択値と OurProduct1 の共通部分における sales の合計が返されます。

set 関数を使用した set 修飾子

ネストされた set 定義を使用して、項目値のセットを定義する必要がある場合があります。例えば、製品を選択せずに、特定の製品を購入したすべての顧客を選択したい場合があります。

このような場合は、要素セット関数 P() および E() を使用してください。これらは、それぞれ、項目の可能な値と除外された値の要素セットを返します。角括弧内で、問題の項目と、スコープを定義する set 数式を指定できます。例:

```
P({1<Year = {2021}>} Customer)
```

これにより、2021 年にトランザクションが発生した一連の顧客が返されます。その後、これを set 修飾子で使用できます。例:

```
Sum({<Customer = P({1<Year = {2021}>} Customer)>} Amount)
```

この set 数式はこれらの顧客を選択しますが、選択を 2021 年に制限するものではありません。

これらの関数は、他の数式では使用できません。

さらに、要素セット関数内で使用できるのは natural set のみです。つまり、単一の選択によって定義されたレコードのセットです。

例えば、{1-$} によるセットは選択を通して定義されたものとは限らず、そのため natural set ではありません。これらの関数を non-natural set で使用すると、予期しない結果が返されます。

例: set 関数を使用した set 修飾子のチャートの数式

例 - チャートの数式

ロードスクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
MyTable:
Load
Year(Date) as Year,
Date#(Date,'YYYY-MM-DD') as ISO_Date,
Date(Date#(Date,'YYYY-MM-DD'),'M/D/YYYY') as US_Date,
Country, Product, Amount
Inline
[Date, Country, Product, Amount
2018-02-20, Canada, Washer, 6
2018-07-08, Germany, Anchor bolt, 10
2018-07-14, Germany, Anchor bolt, 3
2018-08-31, France, Nut, 2
2018-09-02, Czech Republic, Bolt, 1
2019-02-11, Czech Republic, Bolt, 3
2019-07-31, Czech Republic, Washer, 6
2020-03-13, France, Anchor bolt, 1
2020-07-12, Canada, Anchor bolt, 8
2020-09-16, France, Washer, 1];
```

チャートの数式

次のチャートの数を使用して、Qlik Sense シートにテーブルを作成します。

テーブル - set 関数を使用した set 修飾子

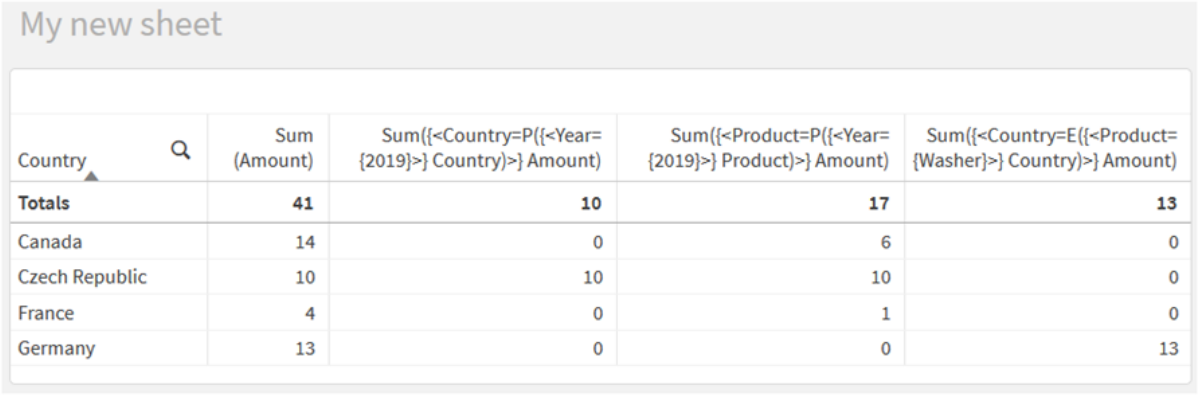
国	Sum (Amount)	Sum({<Country=P ({<Year= {2019}>}Country)>} Amount)	Sum({<Product=P ({<Year= {2019}>}Product)>} Amount)	Sum({<Country=E ({<Product= {Washer}>}Country)>} Amount)
合計	**41**	**10**	**17**	**13**
カナダ	14	0	6	0
チェコ共和国	10	10	10	0

国	Sum (Amount)	Sum({<Country=P ({<Year= {2019}>}Country)>} Amount)	Sum({<Product=P ({<Year= {2019}>}Product)>} Amount)	Sum({<Country=E ({<Product= {Washer}>}Country)>} Amount)
合計	**41**	**10**	**17**	**13**
フランス	4	0	1	0
ドイツ	13	0	0	13

説明

- 軸:
  - `Country`
- 数式:
  - `Sum(Amount)`
    set 数式なしで `Amount` を合計します。
  - `Sum({<Country=P({<Year={2019}>} Country)>} Amount)`
    2019 年に関連する国の場合は Sum `Amount` です。ただし、計算は `2019` に限定されません。
  - `Sum({<Product=P({<Year={2019}>} Product)>} Amount)`
    2019 年に関連する製品の場合は Sum `Amount` です。ただし、計算は `2019` に限定されません。
  - `Sum({<Country=E({<Product={Washer}>} Country)>} Amount)`
    製品 `Washer` に関連付けられていない国の場合は Sum `Amount` です。

set 関数を使用した set 修飾子



例	結果
sum({$<Customer = P({1<Product= {'Shoe'}>} Customer)>} Sales)	現在の選択に対する売上が返されますが、製品「Shoe」を購入したことのある顧客のみが対象となります。要素関数 P( ) はここで、絞込まれた顧客 (「Shoe」を項目 Product で選択することで黙示的に定義) のリストを返します。

例	結果
sum({$<Customer = P({1<Product= {'Shoe'}>})>} Sales)	上記と同様です。要素関数内の項目が省略されると、外部代入で指定された項目に可能な値が返されます。
sum({$<Customer = P({1<Product= {'Shoe'}>} Supplier)>} Sales)	現在の選択条件に対する売上高が返されますが、製品「Shoe」を供給したことのある顧客、つまり、サプライヤでもある顧客のみが対象となります。要素関数 P( ) はここで、絞込まれたサプライヤ (「Shoe」を項目 Product で選択することで黙示的に定義) のリストを返します。次に、サプライヤのリストは項目 Customer の選択に使用されます。
sum({$<Customer = E({1<Product= {'Shoe'}>})>} Sales)	現在の選択に対する sales が返されますが、製品「Shoe」を購入したことのない顧客のみが対象となります。ここで関数 E( ) は、「Shoe」を Product 項目で選択することによって除外された Customer のリストを返します。

内部と外部の set 数式

set 数式は中括弧で囲まれた集計関数内外で使用できます。

集計関数内で set 数式を使用すると、次のようになります。

内部の set 数式

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

複数の集計を行う式があり、すべての集計関数で同じ set 数式を書くことを回避する場合、集計関数の外で set 数式を使用します。

外部の set 数式を使用する場合、スコープの始めに配置する必要があります。

外部の set 数式

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

集計関数の外で set 数式を使用する場合、既存のマスター メジャーに適用することもできます。

マスター メジャーに適用された外部の set 数式

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

集計関数の外で使用される set 数式は、括弧で囲まれていない限り、式全体に影響を及ぼします。以下の語彙範囲の例では、set 数式が括弧内の集計にのみ適用されます。

語彙範囲

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

ルール

語彙範囲
set 数式は、括弧で囲まれていない限り、数式全体に影響します。その場合、括弧が語彙範囲を定義します。

位置
set 数式は、語彙範囲の最初に配置する必要があります。

コンテキスト
コンテキストは、数式に関連性のある選択です。従来、コンテキストは常に現在の選択状態を既定としています。しかしオブジェクトが代替ステートに設定されている場合、コンテキストは現在の選択状態の代替ステートとなります。

外部の set 数式の形式でコンテキストを定義することもできます。

継承
内部の set 数式は、外部の set 数式よりも優先されます。内部の set 数式に set 識別子がある場合、コンテキストが置換されます。そうでない場合、コンテキストと set 数式がマージされます。

- `{$<SetExpression>}` - 外部の set 数式を上書きする
- `{<SetExpression>}` - 外部の set 数式とマージされる

要素セットの割り当て
要素セットの割り当ては、2 つの選択肢がどのようにマージされるかを決定します。通常の等号が使用される場合、内部の set 数式の選択が優先されます。そうでない場合、暗黙の set 演算子が使用されます。

- `{<Field={value}>}` - この内部選択肢は "`Field`" の任意の外部選択肢を置換します。
- `{<Field+={value}>}` - この内部選択肢は、union 演算子を使用して "`Field`" の外部選択肢とマージされます。
- `{<Field*={value}>}` - この内部選択肢は、intersection 演算子を使用して "`Field`" の外部選択肢とマージされます。

複数のステップの継承
継承は複数のステップで発生する可能性があります。例:

- 現在の選択 → `Sum(Amount)`
  集計関数はコンテキストを使用します。ここでは現在の選択です。
- 現在の選択 → `{<Set1>} Sum(Amount)`
  `Set1` は現在の選択から継承され、結果は集計関数のコンテキストとなります。
- 現在の選択 → `{<Set1>} ({<Set2>} Sum(Amount))`
  `Set2` は `Set1` から継承され、結果は集計関数のコンテキストとなります。

Aggr() 関数
`Aggr()` 関数は、2 つの独立した集計を持つネストされた集計を作成します。下の例では、`Count()` が `Dim` の各値に対して計算され、算出された配列は `Sum()` 関数を使用して集計されます。

Sum(Aggr(Count(X),Dim))
Count() は内部集計であり、Sum() は外部集計です。

- 内部集計は、外部集計からどのコンテキストも継承しません。
- 内部集計は Aggr() 関数からコンテキストを継承しますが、これには set 数式が含まれている可能性があります。
- Aggr() 関数と外部集計関数の両方とも、外部の set 数式からコンテキストを継承します。

チュートリアル - set 数式の作成

Qlik Sense で、データ分析をサポートする set 数式を構築できます。このコンテキストでは、分析は set 分析と呼ばれることがよくあります。set 分析は、アプリの現在の選択によって定義されたレコードのセットとは異なるスコープを定義する方法を提供します。

学習内容

このチュートリアルでは、set 修飾子、識別子、演算子を使用して set 数式を作成するためのデータの数式とチャートの数式を提供します。

このチュートリアルを完了する必要がある対象者

このチュートリアルは、スクリプトエディタとチャート数式の操作に慣れているアプリ開発者を対象としています。

始める前に必要な準備

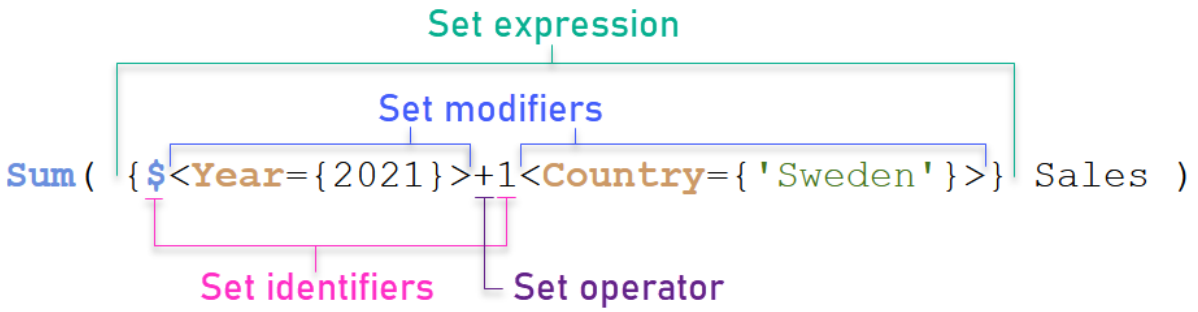
データのロードとアプリの作成を可能にする Qlik Sense Enterprise プロフェッショナル アクセス割り当て。

- set 分析 パート1: 初心者向けの概要
- set 分析 パート2

set 数式の要素

set 数式は、Sum()、Max()、Min()、Avg()、Count() などの集計関数で囲まれています。set 数式は要素と呼ばれる構成要素から構成されます。これらの要素は、set 修飾子、識別子、演算子です。

set 数式の要素



例えば、上記の set 数式は、集計 Sum(Sales) から構築されます。set 数式は外側の中括弧 { } で囲まれています。

数式の 1 番目のオペランドは次のとおりです: $<Year={2021}>

このオペランドは、現在の選択条件の 2021 年の売上を返します。修飾子 <Year={2021}> には、2021 年の選択が含まれます。$ set 識別子は set 数式が現在の選択条件に基づいていることを示します。

数式の 2 番目のオペランドは次のとおりです: 1<Country={'Sweden'}>

このオペランドは Sweden に対して Sales を返します。修飾子 <Country={'Sweden'}> には、国 Sweden の選択が含まれます。1 set 識別子はアプリで行われた選択が無視されることを示します。

最後に、+ set 演算子は、数式が 2 つのセット オペランドのいずれかに属するレコードで構成されるセットを返すことを示します。

set 数式チュートリアルの作成

このチュートリアルに示す set 数式を作成するには、次の手順を実行します。

新しいアプリを作成してデータをロード

次の手順を実行します。

1. 新しいアプリを作成します。
2. [**スクリプト エディタ**] をクリックします。または、ナビゲーション バーの [**準備**] > [**データ ロード エディタ**] をクリックします。
3. [**データ ロード エディタ**] で新しいセクションを作成します。
4. 次のデータをコピーして、新しいセクションに貼り付けます。*set 数式 チュートリアル データ (page 318)*
5. [**データのロード**] をクリックします。データはインライン ロードとしてロードされます。

修飾子を使用して set 数式を作成する

set 修飾子は、1 つまたは複数の項目名で構成されます。各項目名の後には、項目で実行する必要がある選択が続きます。修飾子は山括弧 (< >)でくくります。例えば、この set 数式では次のようになります。

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

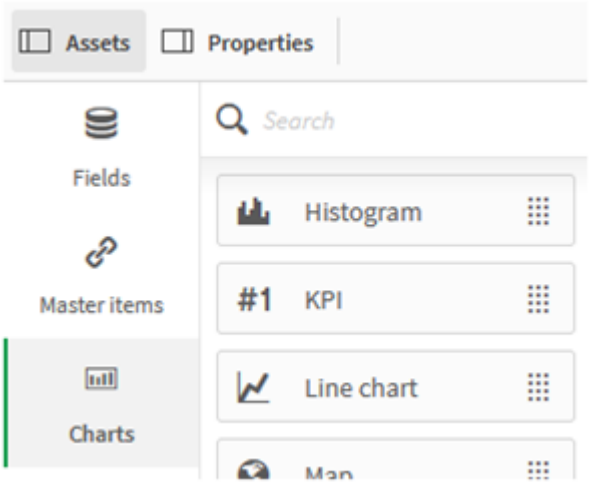
修飾子は次のとおりです。

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

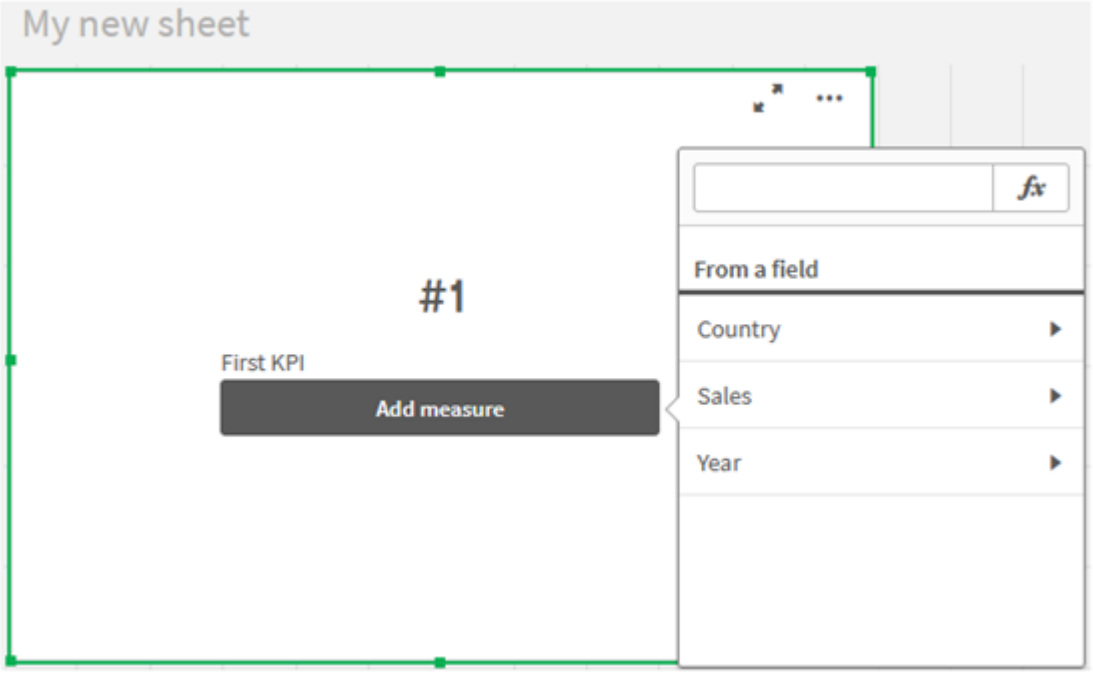
この修飾子は 2015 年のデータが選択されることを指定します。修飾子が囲まれている中括弧は set 数式を示します。

次の手順を実行します。

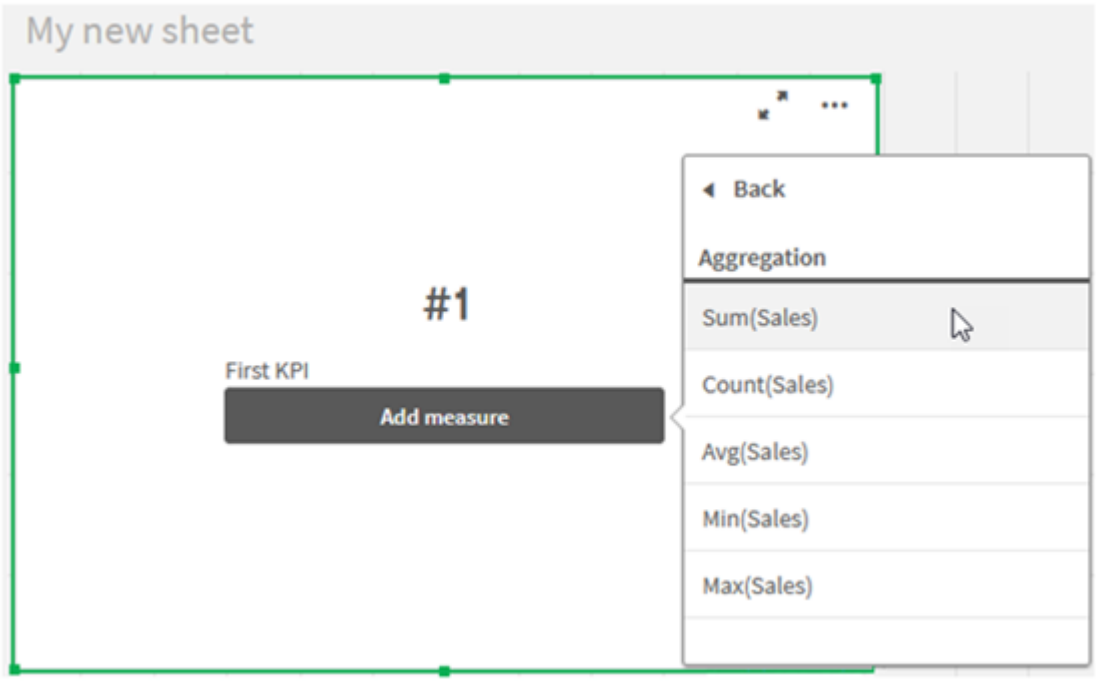
1. シートで、ナビゲーション バーから**アセット** パネルを開き、[**チャート**] をクリックします。



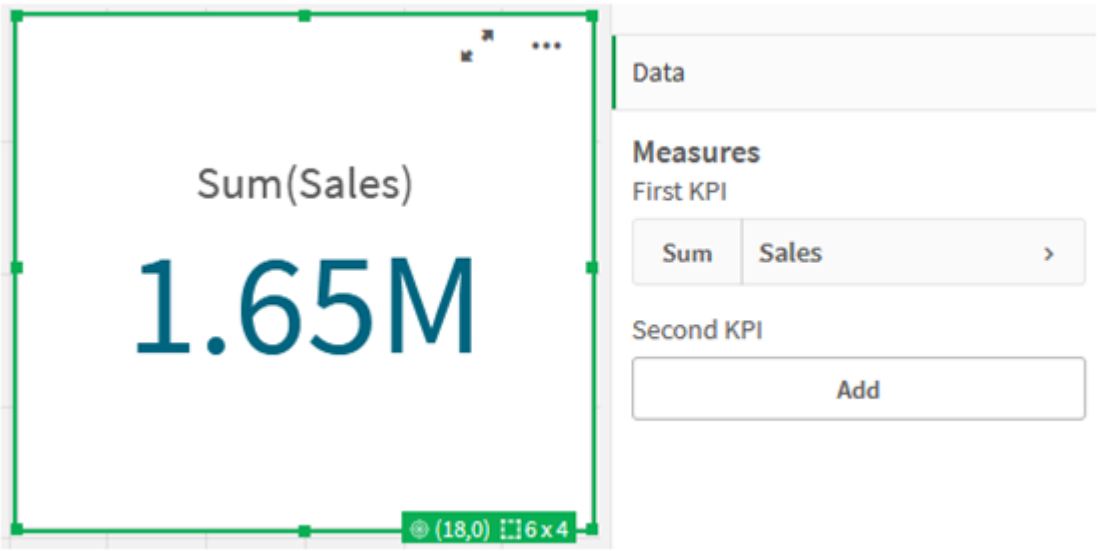
2. **KPI** をシートにドラッグして、[**メジャーを追加**] をクリックします。



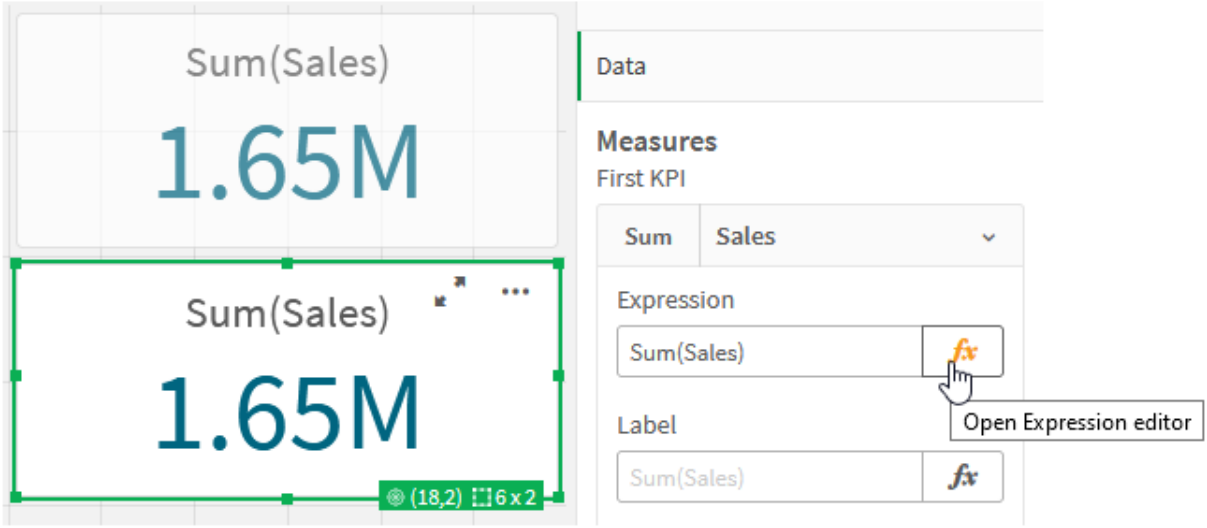
3. [Sales] をクリックして、集計 に [Sum(Sales)] を選択します。



[KPI] は、すべての年の売上高の合計を示しています。



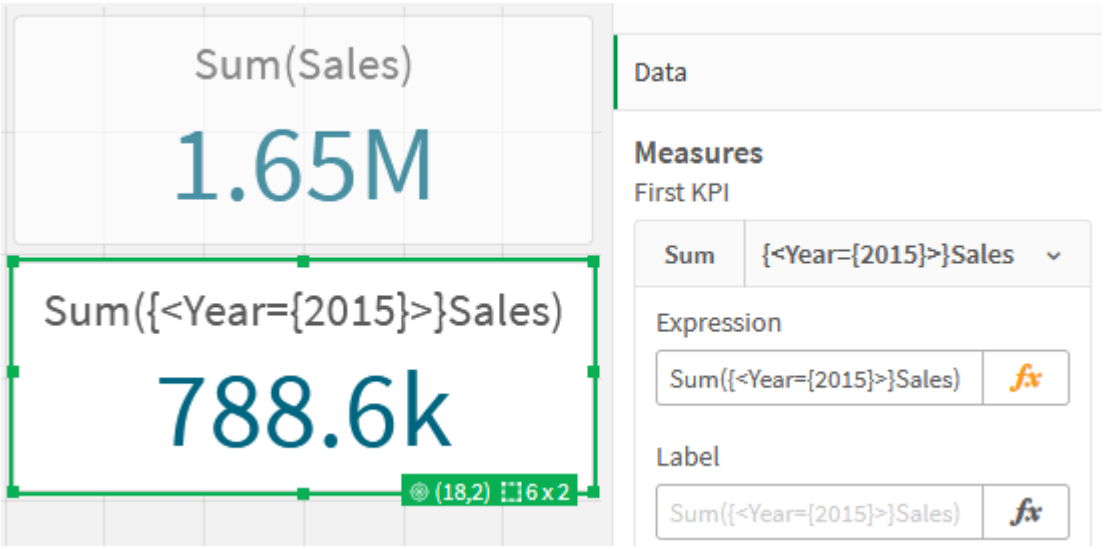
4. [KPI] をコピー & ペーストして、新しい [KPI] を作成します。
5. 新しい [KPI] をクリックして、[**メジャー**] の下の [**売上**] をクリックしてから、[**数式エディタを開く**] をクリックします。



数式エディタが開き、集計 `Sum(Sales)` が表示されます。

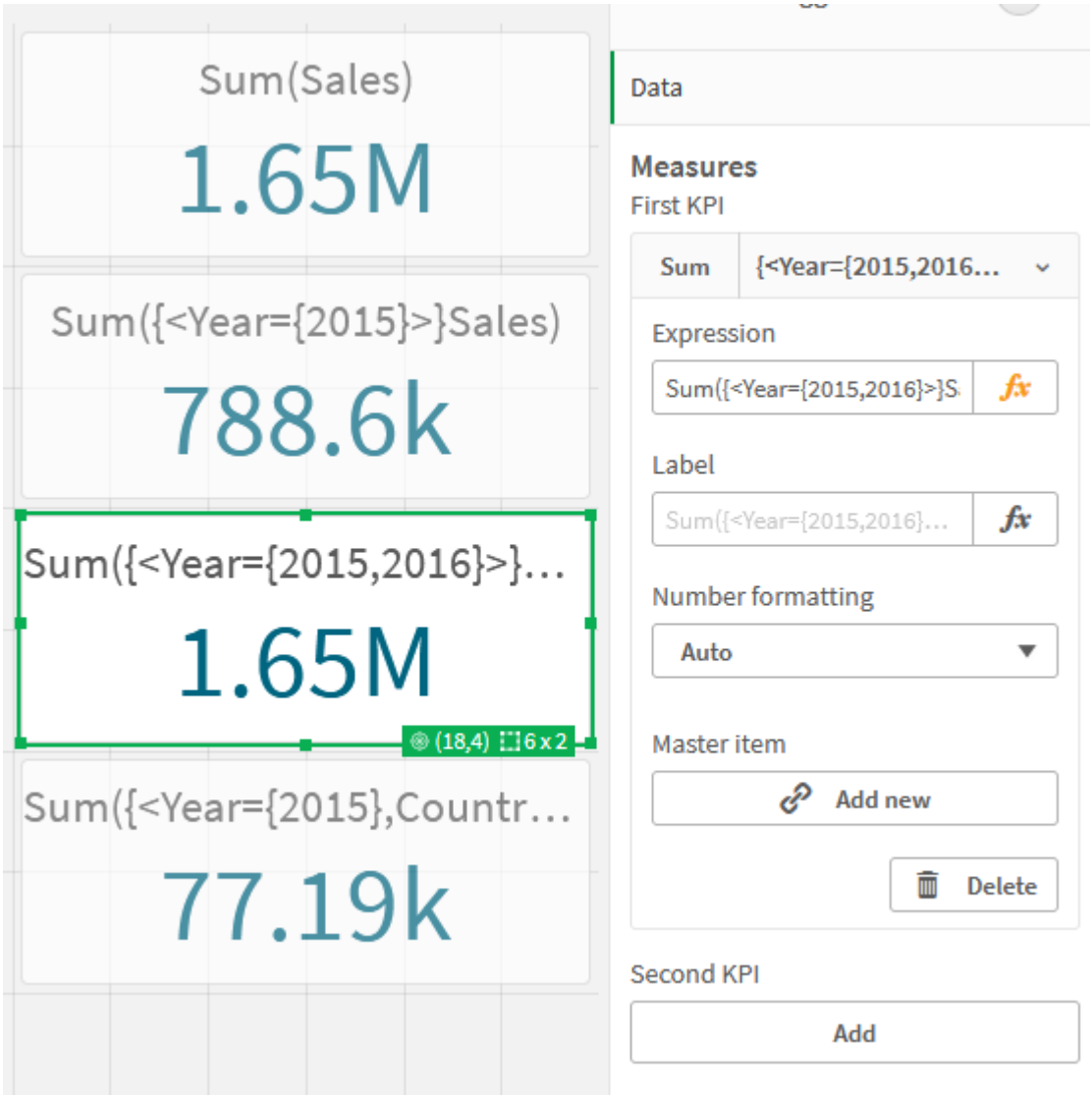
6. 数式エディターで、2015 年のみの [Sales] を合計する数式を作成します。
   i. set 数式を示すために中括弧を追加します: `Sum({}Sales)`
   i. set 修飾子を示すために山括弧を追加します: `Sum({<>}Sales)`
   ii. 山括弧内に、選択する項目 (この場合、項目は `Year`) を追加し、その後に等号を追加します。次に、2015 年を別の中括弧で囲みます。結果の set 修飾子は次のとおりです: `{<Year={2015}>}`。
   全体の数式は次のとおりです。
   `Sum({<Year={2015}>}Sales)`

iii. [**適用**] をクリックすると、式が保存され、数式エディタが閉じます。2015 年の Sales の合計は KPI に表示されます。



7. 以下の式でさらに 2 つの KPI を作成します。
   `Sum({<Year={2015,2016}>}Sales)`
   上記の修飾子は `<Year={2015,2016}>` です。数式は 2015 年と2016 年の Sales の合計を返します。
   `Sum({<Year={2015},Country={'Germany'}>} Sales)`
   上記の修飾子は `<Year={2015}, Country={'Germany'}>` です。この数式は 2015 年の [Sales] の合計を返します。2015 年は [Germany] と交差します。

set 数式を使用した KPI



set 識別子の追加

上記の set 数式は、識別子が使用されていないため、現在の選択条件をベースとして使用します。次に、識別子を追加して、選択されたときの動作を指定します。

次の手順を実行します。

シート上で、次の set 数式を構築またはコピーします。

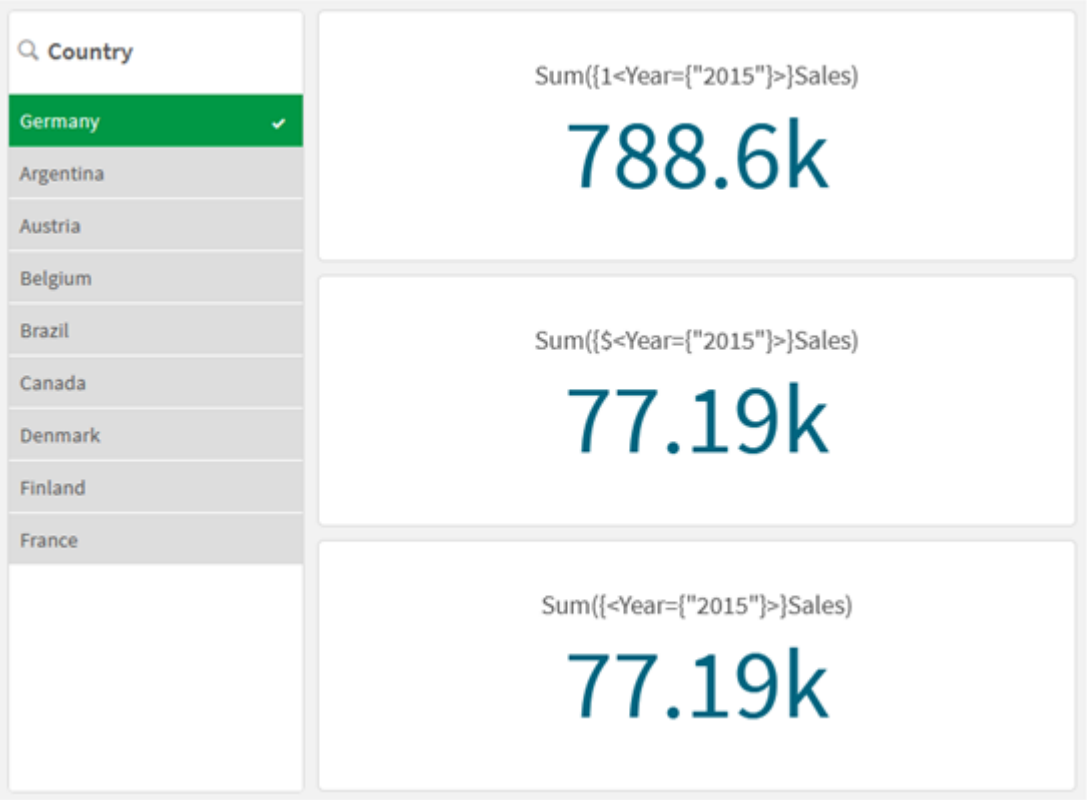
```
Sum({$<Year={"2015"}>}Sales)
```

$ 識別子は、データで選択された現在の選択条件に基づいて set 数式を作成します。これは、識別子が使用されていない場合の既定と同じ動作です。

```
Sum({1<Year={"2015"}>}Sales)
```

1 修飾子により、2015 年の Sum(Sales) の集約では現在の選択が無視されます。ユーザーが他の選択を行っても、集計の値は変更されません。例えば、以下で Germany が選択された場合、2015 年の集約合計は、変化しません。

set 修飾子と識別子を使用した KPI



演算子の追加

set 演算子は、データセットを含めたり、除外したり、交差させたりするために使用されます。すべての演算子はオペランドとして set を使用し、結果として set を返します。

set 演算子は、次の 2 つの異なる状況で使用できます。

- データ内のレコードのセットを表す、セット識別子に対して set 演算を実行する。
- 要素セット、項目値、または set 修飾子内で set 演算を実行する。

次の手順を実行します。

シート上で、次の set 数式を構築またはコピーします。

```
Sum({$<Year={2015}>+1<Country={'Germany'}>}Sales)
```

プラス記号 (+) 演算子は、2015 とGermany のデータセットの和集合を生成します。上記の set 識別子で説明したように、ドル記号 ($) 識別子は、現在の選択が最初のオペランド <Year={2015}> に使用されることを意味します。1 の識別子は、2 番目のオペランド、<Country={'Germany'}> の選択が無視されることを意味します。

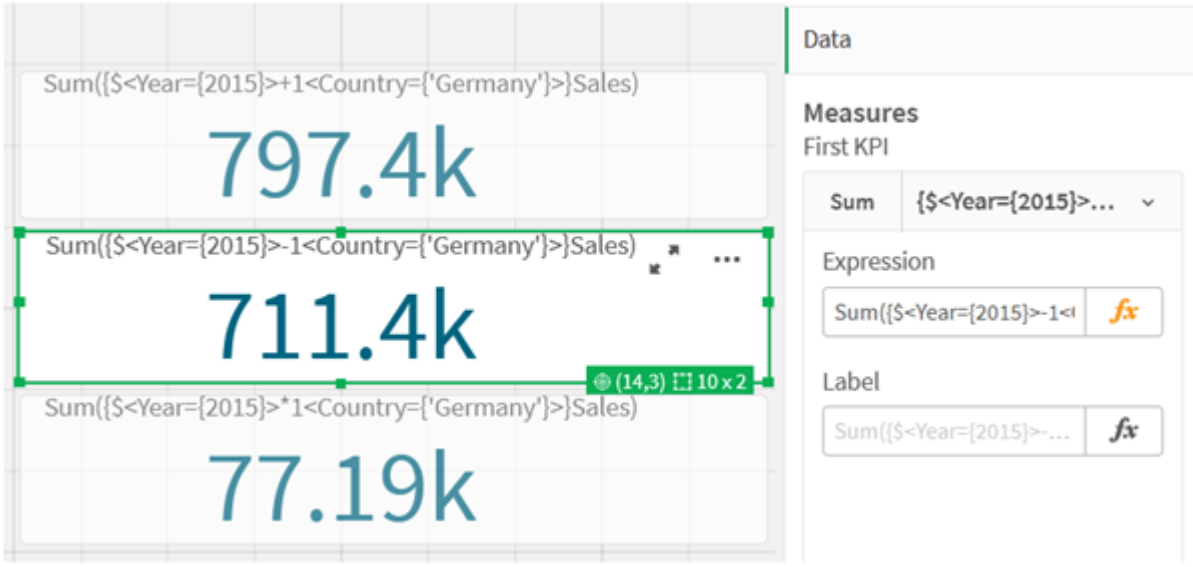
プラス記号 (+) 演算子を使用した KPI

または、マイナス記号 (-) を使用して、Germany ではなく2015 に属するレコードで構成されるデータセットを返します。または、アスタリスク (*) を使用して、両方のセットに属するレコードで構成されるセットを返します。

```
Sum({$<Year={2015}>-1<Country={'Germany'}>}Sales)
```

```
Sum({$<Year={2015}>*1<Country={'Germany'}>}Sales)
```

演算子を使用した KPI



set 数式チュートリアル データ

ロード スクリプト

以下のデータをインライン ロードでロードして、チュートリアルのチャートの数式を作成します。

```
//Create table SalesByCountry
SalesByCountry:
Load * Inline [
Country, Year, Sales
Argentina, 2016, 66295.03
Argentina, 2015, 140037.89
Austria, 2016, 54166.09
Austria, 2015, 182739.87
```

```
Belgium, 2016, 182766.87
Belgium, 2015, 178042.33
Brazil, 2016, 174492.67
Brazil, 2015, 2104.22
Canada, 2016, 101801.33
Canada, 2015, 40288.25
Denmark, 2016, 45273.25
Denmark, 2015, 106938.41
Finland, 2016, 107565.55
Finland, 2015, 30583.44
France, 2016, 115644.26
France, 2015, 30696.98
Germany, 2016, 8775.18
Germany, 2015, 77185.68
];
```

set 数式の構文

完全な構文 (優先を定義する標準の括弧のオプション使用は除く) は、BNF (Backus-Naur Formalism: バッカスナウル形式) で記述します。

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

6.3 チャート式用の一般的な構文

次の構文はチャートの数式に使用できる一般的な書式で、多数のオプションのパラメータがあります。

```
expression ::= ( constant | expressionname | operator1 expression | expression operator2
expression | function | aggregation function | (expression ) )
```

ここで

constant は、ストレート単一引用符で囲まれた文字列 (テキスト、日付、時刻) または数値です。定数は 3 桁区切りの記号を使用せず、小数点に 10 進小数点を使用します。

expressionname は、同じチャートに含まれるもう1 つの式の名前 (ラベル) です。

operator1 は、右側にある1 つの数式に対して作用する単項演算子です。

operator2 は、両側にある 2 つの数式に対して作用する二項演算子です。

```
function ::= functionname ( parameters )
parameters ::= expression { , expression }
```

パラメータの数と種類は任意ではなく、使用する関数によって異なります。

```
aggregationfunction ::= aggregationfunctionname ( parameters2 )
parameters2 ::= aggrexpression { , aggrexpression }
```
パラメータの数と種類は任意ではなく、使用する関数によって異なります。

6.4 集計関数の一般的な構文

次の構文は集計に使用できる一般的な書式で、多数のオプションのパラメータがあります。

```
aggrexpression ::= ( fieldref | operator1 aggrexpression | aggrexpression operator2
aggrexpression | functioninaggr | ( aggrexpression ) )
```
fieldref は項目名です。

```
functionaggr ::= functionname ( parameters2 )
```
このように、**fieldref** が常にちょうど 1 つの集計関数で囲まれており、数式が解釈可能な値を返す限り、数式と関数は自由にネストが可能であり、Qlik Sense にエラー メッセージが表示されることはありません。

7 演算子

このセクションでは、Qlik Sense で利用できる演算子について説明します。2 種類の演算子があります。

- 単項演算子 (単一オペランドのみを使用します)
- 二項演算子 (2 つのオペランドを使用します)

ほとんどの演算子は、二項演算子です。

次の演算子を定義できます。

- ビット演算子
- 論理演算子
- 数値演算子
- 関係演算子
- 文字列演算子

7.1 ビット演算子

すべてのビット演算子は、オペランドを符号付き整数 (32 ビット) に変換し (切り捨て)、同じ方法で結果を返します。すべての演算は、ビット単位で行われます。オペランドを数値として解釈できない場合、演算は NULL を返します。

ビット演算子

演算子	氏名	説明
bitnot	ビット反転。	単項演算子。この演算は、ビットごとに行われるオペランドの論理否定を返します。 `bitnot 17` は、-18 を返します
bitand	ビット単位の論理積。	この演算は、ビットごとに行われるオペランドの論理積を返します。 `17 bitand 7` は、1 を返します
bitor	ビット単位の論理和。	この演算は、ビットごとに行われるオペランドの論理和を返します。 `17 bitor 7` は、23 を返します

演算子	氏名	説明
bitxor	ビット単位の排他的論理和。	この演算は、ビットごとに行われるオペランドの排他的論理和を返します。 `17 bitxor 7` は、22 を返します
>>	ビット右シフト。	演算は、右シフトした最初のオペランドを返します。手順数は、2 番目のオペランドで定義されます。 `8 >> 2` は、2 を返します
<<	ビット左シフト。	演算は、左シフトした最初のオペランドを返します。手順数は、2 番目のオペランドで定義されます。 `8 << 2` は、32 を返します

7.2 論理演算子

すべての論理演算子は、オペランドを論理的に解釈し、結果として True (-1) または False (0) を返します。

論理演算子

演算子	説明
not	論理否定。いくつかの単項演算子の 1 つ。この演算は、オペランドの論理否定を返します。
and	論理積 (and)。この演算は、オペランドの論理積を返します。
or	論理和 (or)。この演算は、オペランドの論理和を返します。
Xor	排他的論理和。この演算は、オペランドの排他的論理和を返します。つまり、論理和と似ていますが、両方のオペランドが True の場合の結果が False になるという違いがあります。

7.3 数値演算子

すべての数値演算子はオペランドの数値を使用し、結果として数値を返します。

数値演算子

演算子	説明
+	正の数値 (単項演算子) または加算を表す記号。この二項演算子は、2 つのオペランドの和を返します。
-	負の数値 (単項演算子) または減算を表す記号。この単項演算子は -1 倍したオペランドを返し、二項演算子は 2 つのオペランドの差を返します。
*	乗算。この演算子は、2 つのオペランドの積を返します。
/	除算。この演算子は、2 つのオペランドの割合を返します。

7.4 関係演算子

すべての関係演算子はオペランドの値を比較し、結果としてTrue (-1) または False (0) を返します。すべての関係演算子が二項演算子です。

関係演算子

演算子	説明
<	未満。両方のオペランドが数値と解釈できる場合、数値比較が行われます。この演算は、比較の評価の論理値を返します。
<=	以下。両方のオペランドが数値と解釈できる場合、数値比較が行われます。この演算は、比較の評価の論理値を返します。
>	超。両方のオペランドが数値と解釈できる場合、数値比較が行われます。この演算は、比較の評価の論理値を返します。
>=	以上。両方のオペランドが数値と解釈できる場合、数値比較が行われます。この演算は、比較の評価の論理値を返します。
=	等しい。両方のオペランドが数値と解釈できる場合、数値比較が行われます。この演算は、比較の評価の論理値を返します。
<>	等しくない。両方のオペランドが数値と解釈できる場合、数値比較が行われます。この演算は、比較の評価の論理値を返します。

演算子	説明
precedes	**<** 演算子とは異なり、比較の前に引数値の数値解釈は行われません。演算子の左側の値がテキスト表現で、それが文字列の比較で右側の値のテキスト表現の前に来る場合は True を返します。 '1 ' precedes ' 2' は FALSE を返します ' 1' precedes ' 2' は TRUE を返します これは空白 (' ') の ASCII 値が数字の ASCII 値より小さいためです。 これを次のものと比較します。 '1 ' < ' 2' は TRUE を返します ' 1' < ' 2' は TRUE を返します
follows	**>** 演算子とは異なり、比較の前に引数値の数値解釈は行われません。演算子の左側の値がテキスト表現で、それが文字列の比較で右側の値のテキスト表現の後に来る場合は True を返します。 ' 2' follows '1' を返します FALSE ' 2' follows ' 1' は TRUE を返します これは空白 (' ') の ASCII 値が数字の ASCII 値より小さいためです。 これを次のものと比較します。 ' 2' > ' 1' は TRUE を返します ' 2' > '1 ' は TRUE を返します

7.5 文字列演算子

2 つの文字列演算子があります。そのうちの 1 つはオペランドの文字列値を使用し、結果として文字列を返します。もう1 つはオペランドを比較し、一致したかどうかを示す論理値を返します。

&

文字列連結。この演算は、2 つのオペランド文字列を順に連結したテキスト文字列を返します。

'abc' & 'xyz' は「abcxyz」を返します

like

ワイルドカード文字列を使用した文字列比較。演算子の前の文字列が演算子の後の文字列と一致した場合、この演算は論理値 True (-1) を返します。2 番目の文字列には、ワイルドカード文字 * (任意の数の任意の文字) または ? (1 つの任意の文字) が含まれることがあります。

`'abc' like 'a*'` は、True (-1) を返します

`'abcd' like 'a?c*'` は、True (-1) を返します

`'abc' like 'a??bc'` は、False (0) を返します

8 スクリプトおよびチャート関数

データロードスクリプトとチャートの数式の関数を使用して、データを変換および集計します。

関数の多くはデータロードスクリプトとチャート式の両方で同じように使用できますが、次のようないくつかの例外があります。

- データロードスクリプトでのみ使用できる関数もあり、それらはスクリプト関数として表されています。
- チャートの数式でのみ使用できる関数もあり、それらはチャート関数として表されています。
- データロードスクリプトおよびチャートの数式の両方で使用できる関数もありますが、パラメータおよびアプリケーションは異なります。それらは、スクリプト関数またはチャート関数として個別のトピックで説明されています。

8.1 サーバー側の拡張 (SSE) での分析接続

分析接続によって有効化される関数は、分析接続を構成して Qlik Sense を起動した場合にのみ表示されます。

分析接続は QMC で構成します。ガイド『Qlik Sense サイトの管理』のトピック「分析接続の作成」を参照してください。

Qlik Sense Desktop で分析接続を構成するには、*Settings.ini* ファイルを編集します。ガイド『Qlik Sense Desktop』のトピック「Configuring analytic connections in Qlik Sense Desktop」(「 における分析接続の構成」) を参照してください。

8.2 集計関数

集計関数と呼ばれる関数のファミリは、複数の項目値を入力として取得し、グループごとに結果を 1 つ返す関数で構成されます。グループ化はチャート軸やスクリプトステートメント内の **group by** 条件によって定義されます。

集計関数には、**Sum()**、**Count()**、**Min()**、**Max()** などがあります。

ほとんどの集計関数は、データロードスクリプトとチャートの数式の両方で使用できますが、構文が異なります。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

エンティティに名前を付けるときは、同じ名前を複数の項目、変数、またはメジャーに割り当てないでください。同じ名前のエンティティ間の競合を解決するには、厳密な優先順位があります。この順序は、これらのエンティティが使用されるすべてのオブジェクトまたはコンテキストに反映されます。この優先順位次のとおりです。

- 集計の内部では、項目は変数よりも優先されます。メジャーラベルは、集計内では関係がなく、優先されません。

- 集計の外部では、メジャー ラベルは変数よりも優先され、変数は項目名よりも優先されます。
- さらに、集計の外部では、ラベルが実際には計算されたものである場合を除き、メジャー ラベルを参照することによりメジャーを再使用することができます。この状況では、自己参照のリスクを低減するためにメジャーの重要性は低下し、この場合、名前は メジャーラベルとして常に最初に解釈され、2 番目に項目名、3 番目に変数名として解釈されます。

データロード スクリプトでの集計関数の使用

集計関数は **LOAD** および **SELECT** ステートメントの内部でのみ使用できます。

チャートの数式での集計関数の使用

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

集計関数は、選択内容によって定義されたレコード セットを集計します。ただし、代替のレコード セットは、Set 分析で Set 数式を用いることで定義できます。

集計の計算方法

集約は、特定のテーブルのレコードをループし、そのテーブル内のレコードを集約します。例えば、**Count** (<Field>) は、<Field> が存在するテーブル内のレコードの数をカウントします。固有の項目値のみを集計する場合は、**Count** (**distinct** <Field>) などの **distinct** 句を使用する必要があります。

集計関数に異なるテーブルの項目が含まれている場合、集計関数は、構成フィールドのテーブルの外積のレコードをループします。これにはパフォーマンスの低下があります。このため、特に大量のデータがある場合は、このような集計を回避する必要があります。

キー項目の集約

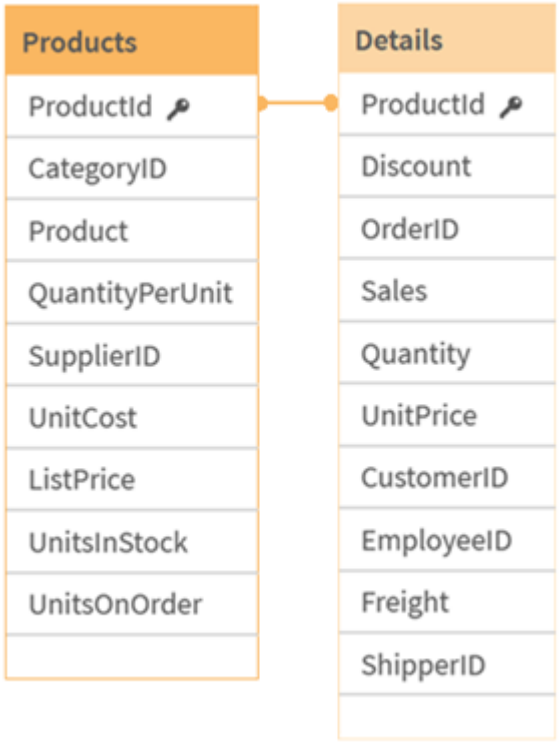
集計の計算方法は、集計に使用するテーブルが明確でないため、キー項目を集計できないことを意味します。例えば、項目 <Key> が 2 つのテーブルをリンクしている場合、**Count**(<Key>) が最初のテーブルまたは 2 番目のテーブルのレコード数を返す必要があるかどうかは明確ではありません。

ただし、**distinct** 句を使用する場合、集計は明確に定義されており、計算できます。

したがって、**distinct** 句を指定せずに集計関数内でキー項目を使用すると、Qlik Sense は意味のない数値を返します。解決策は、**distinct** 句を使用するか、キーのコピー (1 つのテーブルにのみ存在するコピー) を使用することです。

次のテーブルに対する例では、テーブル間のキーは ProductID です。

[Products (商品)] テーブルと [Details (詳細)] テーブル間の ProductID キー



Count(ProductID) は Products テーブル (製品ごとに1つのレコードしかない– ProductID が主キー)、または Details テーブル (製品ごとに複数のレコードがあるものが多い) でカウントされます。個別の製品の数をカウントする場合は、Count(distinct ProductID) を使用する必要があります。特定のテーブルの行数をカウントする場合は、キーを使用しないでください。

基本的な集計関数

基本的な集計関数の概要

基本的な集計関数は、最も一般的な集計関数のグループです。

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

データロードスクリプトの基本的な集計関数

FirstSortedValue

FirstSortedValue() は、**value** で指定した数式から値を返します。これは、単価が最も低い製品の名前など、**sort_weight** 引数のソート結果に対応します。**rank** では、ソート順の n 番目の値を指定できます。指定された **sort_weight** で複数の値が同じ **rank** を共有している場合、この関数は NULL を返します。保存された値は、**group by** 条件で定義されたレコードの数だけ反復処理されます。**group by** 条件が定義されていない場合は、すべてのデータセットから集計されます。

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

Max
Max() は、**group by** 句で定義された数式の集計データの最大値を算出します。**rank** n を指定することで、n 番目に高い値を探し出すことができます。

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

Min
Min() は、**group by** 句で定義された数式の集計データの最小値を算出します。**rank** n を指定することで、n 番目に低い値を探し出すことができます。

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

Mode
Mode() は、**group by** 句で定義された数式の集計データで最も頻繁に登場する値 (モード値) を返します。**Mode()** 関数は数値だけでなく、テキスト値を返すこともあります。

```
Mode (expression )
```

Only
Only()は、集計データに絞込結果が 1 つしか存在しない場合に値を返します。レコードに値が 1 つしか含まれていない場合はその値を、値が 2 つ以上含まれている場合は NULL を返します。**group by** 句を使用して複数のレコードを評価します。**Only()** 関数は数値およびテキスト値を返すこともあります。

```
Only (expression )
```

Sum
Sum() は、**group by** 句で定義された数式の集計値の合計を計算します。

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

チャート式の基本的な集計関数

チャート集計関数は、チャートの数式に含まれる項目にのみ使用できます。1 つの集計関数の引数式に、別の集計関数を含めることはできません。

FirstSortedValue
FirstSortedValue() は、**value** で指定した数式から値を返します。これは、単価が最も低い製品の名前など、**sort_weight** 引数のソート結果に対応します。**rank** では、ソート順の n 番目の値を指定できます。指定された **sort_weight** で複数の値が同じ **rank** を共有している場合、この関数は NULL を返します。

```
FirstSortedValue - チャート関数([{SetExpression}] [DISTINCT] [TOTAL [<fld
{,fld}>]] value, sort_weight [,rank])
```

Max
Max() は集計データの最高値を検出します。**rank** n を指定することで、n 番目に高い値を探し出すことができます。

```
Max - チャート関数Max() は集計データの最高値を検出します。rank n を指定することで、n 番目に高い値を探し出すことができます。 FirstSortedValueおよび rangemax 関数は、Max 関数と機能がよく似ています。必要に応じて、これらの解説も参照してください。 Max([{SetExpression}] [TOTAL
[<fld {,fld}>]] expr [,rank]) 数値 引数引数説明exprメジャーの対象となるデータが含まれている数式または項目。rankrank のデフォルト値は 1 で、これは最大値に相当します。rank を 2 に指定すると、2 番目に高い値が返されます。rank を 3 に指定すると 3 番目に高い値が返され、以下同様に
```

指定した順位に相当する値が返されます。SetExpressionデフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコード セットを定義することも可能です。 TOTAL関数の引数の前に TOTAL の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 TOTAL [<fld {.fld}>] (ここで、TOTAL 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。 データ CustomerProductUnitSalesUnitPrice AstridaAA416AstridaAA1015AstridaBB99BetacabBB510BetacabCC220BetacabDD-25CanutilityAA815CanutilityCC-19例と結果例結果Max(UnitSales)10 (UnitSales の最大値)注文の値は、(UnitSales) で販売された個数に単価を乗じて計算されます。Max (UnitSales*UnitPrice)150 ((UnitSales)*(UnitPrice) で算出されるあらゆる計算結果の最大値)Max(UnitSales, 2)9 (2 番目に大きい値)Max(TOTAL UnitSales)10 (TOTAL 修飾子は、最大値が特定され、チャート軸が無視されたことを意味するため)Customer が軸に設定されているチャートで TOTAL 修飾子を使用すると、各顧客の最大 UnitSales ではなく、全データセットにおける最大値が返されます。 Customer B を選択します。Max({1} TOTAL UnitSales)10 (選択に関係なく、Set Analysis の数式 {1} では ALL として評価されるレコード セットが定義されるため)例で使用されているデータ:ProductData:LOAD * inline [Customer|Product|UnitSales|UnitPriceAstrida|AA|4|16Astrida|AA|10|15Astrida|BB|9|9Betacab|BB|5|10Betacab|CC|2|20Betacab|DD||25Canutility|AA|8|15Canutility|CC||19] (delimiter is '|'); FirstSortedValue RangeMax ([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr [,rank])

Min

Min() は、集計データの最低値を検出します。**rank** n を指定することで、n 番目に低い値を探し出すことができます。

```
Min - チャート関数([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr [,rank])
```

Mode

Mode() は、集計データで最も頻繁に登場する値 (モード値) を返します。**Mode()** 関数は、テキスト値と同様に数値も処理できます。

```
Mode - チャート関数 ({[SetExpression] [TOTAL [<fld {,fld}>]]} expr)
```

Only

Only()は、集計データに絞込結果が 1 つしか存在しない場合に値を返します。たとえば、単価 = 9 の製品を検索した場合、単価が 9 の製品が複数あると、NULL が返されます。

```
Only - チャート関数([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr)
```

Sum

Sum() は、集計データ全体の数式や項目による値の合計を計算します。

```
Sum - チャート関数([{SetExpression}] [DISTINCT] [TOTAL [<fld {,fld}>]] expr])
```

FirstSortedValue

FirstSortedValue() は、**value** で指定した数式から値を返します。これは、単価が最も低い製品の名前など、**sort_weight** 引数のソート結果に対応します。**rank** では、ソート順の n 番目の値を指定できます。指定された **sort_weight** で複数の値が同じ **rank** を共有している場合、この関数は NULL を返します。保存された値は、**group by** 条件で定義されたレコードの数だけ反復処理されます。**group by** 条件が定義されていない場合は、すべてのデータセットから集計されます。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
value Expression	この関数は数式 **value** の値を特定します。これは **sort_weight** のソート結果に対応します。
sort-weight Expression	ソート対象となるデータが含まれている数式です。**sort_weight** の最初の値 (最小値) が返され、そこから **value** 数式の対応する値が特定されます。**sort_weight** の前にマイナス記号を付けると、最後にソートされた値 (最大値) が返されます。
rank Expression	**rank** "n" に 1 よりも大きな値を入力するとで、n 番目のソート値を取得できます。
distinct	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

スクリプトの例

例	結果
Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|CustomerID Astrida|AA|1|10|1 Astrida|AA|7|18|1 Astrida|BB|4|9|1 Astrida|CC|6|2|1 Betacab|AA|5|4|2 Betacab|BB|2|5|2 Betacab|DD|12|25|2 Canutility|AA|3|8|3 Canutility|CC|13|19|3 Divadip|AA|9|16|4 Divadip|AA|10|16|4 Divadip|DD|11|10|4] (delimiter is '|'); FirstSortedValue: LOAD Customer,FirstSortedValue(Product, UnitSales) as MyProductWithSmallestOrderByCustomer Resident Temp Group By Customer;	Customer MyProductWithSmallestOrderByCustomer Astrida CC Betacab AA Canutility AA Divadip DD この関数は UnitSales を最小値から最大値の順にソートし、Customer の値のうち UnitSales の最小値を持つものを最小値から順に検索します。 これは、CC が customer Astrida の最小の注文 (値は UnitSales: 2) に対応するためです。AA は customer Betacab の最小注文 (4)、AA は customer Canutility の最小注文 (8)、DD は customer Divadip. の最小注文 (10) に対応します。
Temp テーブルが前の例のようにロードされた場合: LOAD Customer,FirstSortedValue(Product, -UnitSales) as MyProductWithLargestOrderByCustomer Resident Temp Group By Customer;	Customer MyProductWithLargestOrderByCustomer Astrida AA Betacab DD Canutility CC Divadip - sort_weight 引数の前にマイナス記号を付けると、最大値から順にソートされます。 AA は customer Astrida の最大の注文 (値は UnitSales:18)、DD は customer Betacab の最大の注文 (12)、CC は customer Canutility の最大の注文 (13) に対応します。customer Divadip の最大の注文は 2 つの同じ値 (16) を持つため、結果は NULL となります。
Temp テーブルが前の例のようにロードされた場合: LOAD Customer,FirstSortedValue(distinct Product, -UnitSales) as MyProductWithSmallestOrderByCustomer Resident Temp Group By Customer;	Customer MyProductWithLargestOrderByCustomer Astrida AA Betacab DD Canutility CC Divadip AA distinct 修飾子が使用されている点を除くと、前の例と同じ結果になります。この場合は Divadip の結果が重複して無視され、NULL 以外の値を返すことができるようになります。

FirstSortedValue - チャート関数

FirstSortedValue() は、**value** で指定した数式から値を返します。これは、単価が最も低い製品の名前など、**sort_weight** 引数のソート結果に対応します。**rank** では、ソート順の n 番目の値を指定できます。指定された **sort_weight** で複数の値が同じ **rank** を共有している場合、この関数は NULL を返します。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
value	出力フィールド。この関数は数式 **value** の値を特定します。これは **sort_weight** のソート結果に対応します。
sort_weight	入力フィールド。ソート対象となるデータが含まれている数式です。**sort_weight** の最初の値 (最小値) が返され、そこから **value** 数式の対応する値が特定されます。**sort_weight** の前にマイナス記号を付けると、最後にソートされた値 (最大値) が返されます。
rank	**rank** "n" に 1 よりも大きな値を入力するとで、n 番目のソート値を取得できます。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

データ

Customer	Product	UnitSales	UnitPrice
Astrida	AA	4	16
Astrida	AA	10	15

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

例と結果

例	結果
`firstsortedvalue (Product, UnitPrice)`	BB (最小 `UnitPrice` (9) の `Product`)。
`firstsortedvalue (Product, UnitPrice, 2)`	BB (`UnitPrice` (10) が 2 番目に低い `Product`)。
`firstsortedvalue (Customer, -UnitPrice, 2)`	Betacab (`UnitPrice` (20) が 2 番目に高い `Product` に関連付けられている `Customer`)。
`firstsortedvalue (Customer, UnitPrice, 3)`	NULL (`UnitPrice` (15) の rank が同じ (最下位から 3 番目)`Customer` の値が 2 つ (Astrida と Canutility) あるため)。 予期しない NULL が返ってこないよう、distinct 修飾子を使います。
`firstsortedvalue (Customer, -UnitPrice*UnitSales, 2)`	Canutility (`UnitPrice` と `UnitSales` (120) を掛けた販売注文値が 2 番目に高い `Customer`)。

例で使用されているデータ:

```
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() は、**group by** 句で定義された数式の集計データの最大値を算出します。**rank** n を指定することで、n 番目に高い値を探し出すことができます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。
rank Expression	**rank** のデフォルト値は 1 で、これは最大値に相当します。**rank** を 2 に指定すると、2 番目に高い値が返されます。**rank** を 3 に指定すると 3 番目に高い値が返され、以下同様に指定した順位に相当する値が返されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');

Max:
LOAD Customer, Max(UnitSales) as MyMax Resident Temp Group By Customer;
```

結果のテーブル

Customer	MyMax
Astrida	18
Betacab	5
Canutility	8

Temp テーブルが前の例のようにロードされた場合:

```
LOAD Customer, Max(UnitSales,2) as MyMaxRank2 Resident Temp Group By Customer;
```

結果のテーブル

Customer	MyMaxRank2
Astrida	10
Betacab	4
Canutility	-

Max - チャート関数

Max() は集計データの最高値を検出します。**rank** n を指定することで、n 番目に高い値を探し出すことができます。

> ***FirstSortedValue*** *および* ***rangemax*** *関数は、****Max*** *関数と機能がよく似ています。必要に応じて、これらの解説も参照してください。*

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
rank	**rank** のデフォルト値は 1 で、これは最大値に相当します。**rank** を 2 に指定すると、2 番目に高い値が返されます。**rank** を 3 に指定すると3 番目に高い値が返され、以下同様に指定した順位に相当する値が返されます。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

データ

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

例と結果

例	結果
`Max(UnitSales)`	10 (`UnitSales` の最大値)
注文の値は、(`UnitSales`) で販売された個数に単価を乗じて計算されます。 `Max (UnitSales*UnitPrice)`	150 ((`UnitSales`)*(`UnitPrice`) で算出されるあらゆる計算結果の最大値)
`Max(UnitSales, 2)`	9 (2 番目に大きい値)
`Max(TOTAL UnitSales)`	10 (TOTAL 修飾子は、最大値が特定され、チャート軸が無視されたことを意味するため)Customer が軸に設定されているチャートで TOTAL 修飾子を使用すると、各顧客の最大 UnitSales ではなく、全データセットにおける最大値が返されます。
`Customer B` を選択します。 `Max({1} TOTAL UnitSales)`	10 (選択に関係なく、Set Analysis の数式 {1} では ALL として評価されるレコードセットが定義されるため)

例で使用されているデータ:

```
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 '|');
```

参照先:

- *FirstSortedValue - チャート関数 (page 333)*
- *RangeMax (page 1320)*

Min

Min() は、**group by** 句で定義された数式の集計データの最小値を算出します。**rank** n を指定することで、n 番目に低い値を探し出すことができます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。
rank Expression	**rank** のデフォルト値は 1 で、これは最小値に相当します。**rank** を 2 と指定すると、2 番目に低い値が返されます。**rank** が 3 のときは、3 番目に低い値が返され、以下同様に値が返されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|CustomerID
Astrida|AA|1|10|1
Astrida|AA|7|18|1
Astrida|BB|4|9|1
Astrida|CC|6|2|1
Betacab|AA|5|4|2
```

```
Betacab|BB|2|5|2
Betacab|DD
Canutility|DD|3|8
Canutility|CC
] (delimiter is '|');
Min:
LOAD Customer, Min(UnitSales) as MyMin Resident Temp Group By Customer;
```

結果のテーブル

Customer	MyMin
Astrida	2
Betacab	4
Canutility	8

Temp テーブルが前の例のようにロードされた場合:

```
LOAD Customer, Min(UnitSales,2) as MyMinRank2 Resident Temp Group By Customer;
```

結果のテーブル

Customer	MyMinRank2
Astrida	9
Betacab	5
Canutility	-

Min - チャート関数

Min() は、集計データの最低値を検出します。**rank** n を指定することで、n 番目に低い値を探し出すことができます。

FirstSortedValue *および* ***rangemin*** *関数は、* ***Min*** *関数と機能がよく似ています。必要に応じて、これらの解説も参照してください。*

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。

引数	説明
rank	**rank** のデフォルト値は 1 で、これは最小値に相当します。**rank** を 2 と指定すると、2 番目に低い値が返されます。**rank** が 3 のときは、3 番目に低い値が返され、以下同様に値が返されます。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {.fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

データ

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() 関数は、数式で与えられた値の配列から NULL 以外の値を返します。この例では、データに NULL 値があるため、数式で評価されている NULL 以外の最初の値が返されます。

例と結果

例	結果
`Min(UnitSales)`	2 (NULL 以外で最小の `UnitSales` の値)

例	結果
注文の値は、(UnitSales) で販売された個数に単価を乗じて計算されます。 Min (UnitSales*UnitPrice)	40 (NULL 以外の最小値。((UnitSales)*(UnitPrice) で算出)
Min(UnitSales, 2)	4 (NULL 値の後に続く2 番目に低い値)
Min(TOTAL UnitSales)	2 (TOTAL 修飾子は、最小値が特定され、チャート軸が無視されたことを意味するため)Customer が軸に設定されているチャートで TOTAL 修飾子を使用すると、各顧客の最小 UnitSales ではなく、全データセットにおける最小値が返されます。
Customer B を選択します。 Min({1} TOTAL UnitSales)	2 (Customer B の選択に依存しない) Set Analysis 式 {1} は、選択に関係なく、ALL として評価されるレコードセットを定義します。

例で使用されているデータ:

```
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 '|');
```

参照先:

- *FirstSortedValue - チャート関数 (page 333)*
- *RangeMin (page 1324)*

Mode

Mode() は、**group by** 句で定義された数式の集計データで最も頻繁に登場する値 (モード値) を返します。**Mode()** 関数は数値だけでなく、テキスト値を返すこともあります。

構文:

```
Mode ( expr)
```

戻り値データ型: dual

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。

制限事項:

複数の値が同じ頻度で現れる場合は、NULL が返されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

スクリプトの例

例	結果
Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|CustomerID Astrida|AA|1|10|1 Astrida|AA|7|18|1 Astrida|BB|4|9|1 Astrida|CC|6|2|1 Betacab|AA|5|4|2 Betacab|BB|2|5|2 Betacab|DD Canutility|DD|3|8 Canutility|CC] (delimiter is '|'); Mode: LOAD Customer, Mode(Product) as MyMostOftenSoldProduct Resident Temp Group By Customer;	MyMostOftenSoldProduct AA AA は複数回売れた唯一の製品であるためです。

Mode - チャート関数

Mode() は、集計データで最も頻繁に登場する値 (モード値) を返します。**Mode()** 関数は、テキスト値と同様に数値も処理できます。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {.fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

データ

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

例と結果

例	結果
`Mode(UnitPrice) Customer A` を選択します。	15 (`UnitSales` で最も一般的に発生する値) NULL (-) を返します。1 つの値が他の値よりも頻繁に生じることはありません。

例	結果
Mode(Product) Customer A を選択します。	AA (Product で最も一般的に生じる値) NULL (-) を返します。1 つの値が他の値よりも頻繁に生じることはありません。
Mode (TOTAL UnitPrice)	15 (TOTAL 修飾子は、最も一般的に生じる値が 15 で、チャート軸も無視されることを意味するため)
Customer B を選択します。 Mode({1} TOTAL UnitPrice)	15 (選択に関係なく、Set Analysis の数式 {1} では ALL として評価されるレコードセットが定義されるため)

例で使用されているデータ:

```
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 '|');
```

参照先:

- *Avg - チャート関数 (page 400)*
- *Median - チャート関数 (page 438)*

Only

Only()は、集計データに絞込結果が 1 つしか存在しない場合に値を返します。レコードに値が 1 つしか含まれていない場合はその値を、値が 2 つ以上含まれている場合は NULL を返します。**group by** 句を使用して複数のレコードを評価します。**Only()** 関数は数値およびテキスト値を返すこともあります。

構文:

```
Only ( expr )
```

戻り値データ型: dual

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

```
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;
```

結果のテーブル

Customer	MyUniqIDCheck
Astrida	1 customer Astrida だけが CustomerID を含む完全なレコードを持っているためです。

Only - チャート関数

Only()は、集計データに絞込結果が 1 つしか存在しない場合に値を返します。たとえば、単価 = 9 の製品を検索した場合、単価が 9 の製品が複数あると、NULL が返されます。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。

引数	説明
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

Only() は、サンプル データに可能な値が複数存在する場合に NULL を返すよう設定するために使用します。

例と結果:

データ

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

例と結果

例	結果
`Only({<UnitPrice={9}>} Product)`	BB (`Product` の中で「9」という`UnitPrice` がある唯一の項目)
`Only({<Product={DD}>} Customer)`	Betacab、それは「DD」と呼ばれる `Product` を販売している唯一の `Customer` だからです。
`Only({<UnitPrice={20}>} UnitSales)`	`UnitSales` の数。`UnitSales` の値は 1 つしかないため、`UnitPrice` が 20 の場合は 2 になる ((`UnitPrice`) =20)
`Only({<UnitPrice={15}>} UnitSales)`	NULL (`UnitPrice` = 15 の `UnitSales` 値が 2 つあるため)

例で使用されているデータ:

```
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() は、**group by** 句で定義された数式の集計値の合計を計算します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。
expr Expression	メジャーの対象となるデータが含まれている数式または項目。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

```
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;
```

結果のテーブル

Customer	MySum
Astrida	39
Betacab	9
Canutility	8

Sum - チャート関数

Sum() は、集計データ全体の数式や項目による値の合計を計算します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。 *DISTINCT 修飾子はサポート対象ですが、一部のデータが省略されていても総額が表示されていると誤解を招く可能性があるため、使用する場合は細心の注意が必要です。*
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

データ

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

例と結果

例	結果
`Sum(UnitSales)`	38. (`UnitSales` の値の合計)
`Sum(UnitSales*UnitPrice)`	505. (`UnitPrice` に `UnitSales` の集計値を乗じた合計)
`Sum (TOTAL UnitSales*UnitPrice)`	TOTAL 修飾子はチャート軸に関わらず、合計が 505 であることを意味するため、テーブルのすべての行と合計が 505 になります。
`Customer B` を選択します。 `Sum({1} TOTAL UnitSales*UnitPrice)`	505 (選択に関係なく、Set Analysis の数式 {1} では ALL として評価されるレコードセットが定義されるため)

例で使用されているデータ:

```
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 '|');
```

カウンタ集計関数

カウンタ集計関数は、データロードスクリプトに含まれる多数のレコード、またはチャート軸の多数の値の数式について、多様なカウントタイプを返します。

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

データロードスクリプトのカウンタ集計関数

Count
Count() は、**group by** 句で定義された数式で集計される値の数を返します。

```
Count ([distinct ] expression | * )
```

MissingCount
MissingCount() は、**group by** 句で定義された数式で集計される欠損値の数を返します。

```
MissingCount ([ distinct ] expression)
```

NullCount
NullCount() は、**group by** 句で定義された数式で集計されるNULL値の数を返します。

```
NullCount ([ distinct ] expression)
```

NumericCount
NumericCount() は、**group by** 句で定義された数式で見つかる数値の数を返します。

```
NumericCount ([ distinct ] expression)
```

TextCount
TextCount() は、**group by** 句で定義された数式で集計される数値以外の項目値の数を返します。

```
TextCount ([ distinct ] expression)
```

チャート式のカウンタ集計関数

チャートで使用可能なカウンタ集計関数は、次のとおりです。

Count
Count() は、各チャート軸に含まれる値、テキスト、数値の数の集計に使われます。

```
Count - チャート関数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]} expr)
```

MissingCount
MissingCount() は、各チャート軸の欠損値の数の集計に使われます。欠損値は、いずれも数値ではありません。

```
MissingCount - チャート関数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]
expr)
```

NullCount
NullCount() は、各チャート軸の NULL 値の数の集計に使われます。

```
NullCount - チャート関数({[SetExpression][DISTINCT] [TOTAL [<fld {,fld}>]]}
expr)
```

NumericCount
NumericCount() は、各チャート軸に含まれる数値の数を集計します。

```
NumericCount - チャート関数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]}
expr)
```

TextCount
TextCount() は、各チャート軸に含まれる数値以外の項目値の数の集計に使われます。

```
TextCount - チャート関数({[SetExpression] [DISTINCT] [TOTAL [<fld {,fld}>]]}
expr)
```

Count

Count() は、**group by** 句で定義された数式で集計される値の数を返します。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

スクリプトの例

例	結果
Temp: LOAD * inline [Customer|Product|OrderNumber|UnitSales|UnitPrice Astrida|AA|1|4|16 Astrida|AA|7|10|15 Astrida|BB|4|9|9 Betacab|CC|6|5|10 Betacab|AA|5|2|20 Betacab|BB|1|25| 25 Canutility|AA|3|8|15 Canutility|CC|||19 Divadip|CC|2|4|16 Divadip|DD|3|1|25] (delimiter is '|'); Count1: LOAD Customer,Count(OrderNumber) as OrdersByCustomer Resident Temp Group By Customer;	Customer OrdersByCustomer Astrida 3 Betacab 3 Canutility 2 Divadip 2 軸 Customer がシートのテーブルに含まれている限り、OrdersByCustomer の結果は 3, 2 です。
Temp テーブルが前の例のようにロードされた場合: LOAD Count(OrderNumber) as TotalOrderNumber Resident Temp;	TotalOrderNumber 10
Temp テーブルが最初の例のようにロードされた場合: LOAD Count(distinct OrderNumber) as TotalOrderNumber Resident Temp;	TotalOrderNumber 8 OrderNumber の 2 つの値は同一の値であるため、1、および null 値が 1 つになります。

Count - チャート関数

Count() は、各チャート軸に含まれる値、テキスト、数値の数の集計に使われます。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。

引数	説明
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {.fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD	1	25	25
Canutility	AA	3	8	15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

以下の例では、別途記載されていない限り、すべての顧客が選択されているものとします。

例と結果

例	結果
`Count(OrderNumber)`	10 (OrderNumber の値を持つことができる項目が 10 個あり、空のレコードを含むすべてのレコードがカウントされるため) *「0」は値と見なされ、空白のセルとはなりません。ただし、メジャーの軸に対する集計結果が 0 の場合、この軸はチャートには含まれません。*
`Count(Customer)`	10 (Count では、あらゆる項目の発生回数が評価されるため)
`Count(DISTINCT [Customer])`	4 (Distinct 修飾子を使用すると、Count では固有の発生のみが評価されるため)
顧客 Canutility が選択されている場合 `Count(OrderNumber)/Count({1} TOTAL OrderNumber)`	0.2 (この数式では、選択した顧客の注文数が全顧客の注文の割合として返されるため。この場合は、2/10)
顧客 Astrida と Canutility が選択されている場合 `Count(TOTAL <Product> OrderNumber)`	5 (選択した顧客の製品注文数であり、空のセルがカウントされるため)

例で使用されているデータ:

```
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() は、**group by** 句で定義された数式で集計される欠損値の数を返します。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

スクリプトの例

例	結果

`Temp:`
`LOAD * inline [`
`Customer|Product|OrderNumber|UnitSales|UnitPrice`
`Astrida|AA|1|4|16`
`Astrida|AA|7|10|15`
`Astrida|BB|4|9|9`
`Betacab|CC|6|5|10`
`Betacab|AA|5|2|20`
`Betacab|BB||| 25`
`Canutility|AA|||15`
`Canutility|CC| ||19`
`Divadip|CC|2|4|16`
`Divadip|DD|3|1|25`
`] (delimiter is '|');`
`MissCount1:`
 	
`LOAD Customer,MissingCount(OrderNumber) as`
`MissingOrdersByCustomer Resident Temp Group By Customer;`
 	
`Load MissingCount(OrderNumber) as TotalMissingCount Resident`
`Temp;`
	`Customer`
	`MissingOrdersByCustomer`
	
	`Astrida 0`
	
	`Betacab 1`
	
	`Canutility 2`
	
	`Divadip 0`
	
	2 番目のステートメントの結果:
	
	`TotalMissingCount`
	
	`3`
	この軸を有するテーブル。

Temp テーブルが前の例のようにロードされた場合:
 	
`LOAD MissingCount(distinct OrderNumber) as`
`TotalMissingCountDistinct Resident Temp;`
	`TotalMissingCountDistinct`
	
	`1`
	OrderNumber 欠損値が 1 つのみのことが理由。

MissingCount - チャート関数

MissingCount() は、各チャート軸の欠損値の数の集計に使われます。欠損値は、いずれも数値ではありません。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	9
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15

Customer	Product	OrderNumber	UnitSales	Unit Price
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

例と結果

例	結果
`MissingCount([OrderNumber])`	3 (10 個の OrderNumber 項目のうち、空白になっているものが 3 つあるため)。 *「0」は値と見なされ、空白のセルとはなりません。ただし、メジャーの軸に対する集計結果が 0 の場合、この軸はチャートには含まれません。*
`MissingCount ([OrderNumber])/MissingCount ({1} Total [OrderNumber])`	この数式では、選択した顧客の未完了注文数が全顧客の未完了注文数の割合として返されます。全顧客の OrderNumber のうち、合計 3 つの欠損値があるため、Product の欠損値を持つ各 Customer については、1/3 が返されます。

例で使用されているデータ:

```
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() は、**group by** 句で定義された数式で集計される NULL 値の数を返します。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

スクリプトの例

例	結果

`Set NULLINTERPRET = NULL;`
`Temp:`
`LOAD * inline [`
`Customer|Product|OrderNumber|UnitSales|CustomerID`
`Astrida|AA|1|10|1`
`Astrida|AA|7|18|1`
`Astrida|BB|4|9|1`
`Astrida|CC|6|2|1`
`Betacab|AA|5|4|2`
`Betacab|BB|2|5|2`
`Betacab|DD|||`
`Canutility|AA|3|8|`
`Canutility|CC|NULL||`
`] (delimiter is '|');`
`Set NULLINTERPRET=;`
`NullCount1:`
 	
`LOAD Customer,NullCount(OrderNumber) as NullOrdersByCustomer Resident Temp Group By Customer;`
 	
`LOAD NullCount(OrderNumber) as TotalNullCount Resident Temp;`
	`Customer NullOrdersByCustomer`
	
	`Astrida 0`
	
	`Betacab 0`
	
	`Canutility 1`
	
	2 番目のステートメントの結果:
	`TotalNullCount`
	
	`1`
	その軸を有するテーブルで、null 値を含むレコードは 1 つだけです。

NullCount - チャート関数

NullCount() は、各チャート軸の NULL 値の数の集計に使われます。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
set_expression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {.fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

例と結果

例	結果
NullCount ([OrderNumber])	1 (インライン **LOAD** ステートメントで NullInterpret を使用して Null 値を導入しているため)

例で使用されているデータ:

```
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() は、**group by** 句で定義された数式で見つかる数値の数を返します。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

スクリプトの例

例	結果
`LOAD NumericCount(OrderNumber) as TotalNumericCount Resident Temp;`	2 番目のステートメントの結果: TotalNumericCount 7 この軸を有するテーブル。
Temp テーブルが前の例のようにロードされた場合: `LOAD NumericCount(distinct OrderNumber) as TotalNumericCountDistinct Resident Temp;`	TotalNumericCountDistinct 6 別の OrderNumber と重複している OrderNumber が 1 つあるため、重複を除いた結果は 6 になります。

```
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;
```

結果のテーブル

Customer	NumericCountByCustomer
Astrida	3
Betacab	2
Canutility	0
Divadip	2

NumericCount - チャート関数

NumericCount() は、各チャート軸に含まれる数値の数を集計します。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
set_expression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。

引数	説明
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

以下の例では、別途記載されていない限り、すべての顧客が選択されているものとします。

例と結果

例	結果
`NumericCount ([OrderNumber])`	7 (10 個の OrderNumber 項目のうち 3 つが空のため)。 *「0」は値と見なされ、空白のセルとはなりません。ただし、メジャーの軸に対する集計結果が 0 の場合、この軸はチャートには含まれません。*
`NumericCount ([Product])`	0 (すべての製品名がテキストのため)。通常、この関数を使用して、テキスト項目に数値が含まれているかチェックできます。

例	結果
NumericCount (DISTINCT [OrderNumber])/Count (DISTINCT [OrderNumber)]	固有の数値注文番号をすべてカウントし、数値と非数値の注文番号の合計数で除算します。項目値がすべて数字の場合は 1 になります。通常、この関数を使用してすべての項目値が数字になっているかチェックできます。この例では、一意の数値および非数値が 8 つあり、OrderNumber に一意の数値が 7 つあるため、0.875 が返されます。

例で使用されているデータ:

```
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() は、**group by** 句で定義された数式で集計される数値以外の項目値の数を返します。

構文:

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

戻り値データ型: integer

引数:

引数

引数	説明
expr Expression	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

```
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;
```

結果のテーブル

Customer	ProductTextCount
Astrida	3
Betacab	3
Canutility	2
Divadip	2

```
LOAD Customer,TextCount(OrderNumber) as OrderNumberTextCount Resident Temp Group By Customer;
```

結果のテーブル

Customer	OrderNumberTextCount
Astrida	0
Betacab	1
Canutility	2
Divadip	0

TextCount - チャート関数

TextCount() は、各チャート軸に含まれる数値以外の項目値の数の集計に使われます。

構文:

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

戻り値データ型: 整数

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く)を使用して、合計絞込値のサブセットを作成できます。

例と結果:

Data

Customer	Product	OrderNumber	UnitSales	Unit Price
Astrida	AA	1	4	16
Astrida	AA	7	10	15
Astrida	BB	4	9	1
Betacab	BB	6	5	10
Betacab	CC	5	2	20
Betacab	DD			25
Canutility	AA			15
Canutility	CC			19
Divadip	AA	2	4	16
Divadip	DD	3		25

例と結果

例	結果
TextCount ([Product])	10 (10 個の Product 項目がすべてテキストのため)。 *「0」は値と見なされ、空白のセルとはなりません。ただし、メジャーの軸に対する集計結果が 0 の場合、この軸はチャートには含まれません。空白のセルは非テキストと見なされ、TextCount ではカウントされません。*
TextCount ([OrderNumber])	3 (空のセルがカウントされるため)。通常、この関数を使用して、数字の項目にテキストが指定されていないか、またはゼロ以外の値になっていないか検証します。
TextCount (DISTINCT [Product])/Count ([Product)]	Product (4) の異なるテキスト値の数をすべてカウントし、Product (10) の合計値で割ります。結果は 0.4 です。

例で使用されているデータ:

```
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 '|');
```

財務集計関数

このセクションでは、支払とキャッシュフローに関する財務業務向けの集計関数について説明します。

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

データロードスクリプトの財務集計関数

IRR

IRR() は、group by 句で定義されたレコードで反復処理される数式の数で表される一連のキャッシュフローから集計された内部収益率を返します。

```
IRR (expression)
```

XIRR

XIRR() は、group by 句で定義されたレコードで反復処理される **pmt** と **date** のペア数値で表されたキャッシュフロー計算書 (不定期の場合もあります) の集計された内部収益率 (年次) を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

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

NPV

NPV() スクリプト関数は、期間順の割引率と複数の値を取得します。これらの計算では、インフロー (収入) はプラス、アウトフロー (将来の支払い) はマイナスの値であると仮定しています。これらは、各期間の終わりに発生します。

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

XNPV

XNPV() は、**pmt** と **date** のペア数値で表されるキャッシュフロー計算書 (不定期の場合もあります) の集計された正味現在価値を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

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

チャート式の財務集計関数

チャートで使用可能な財務集計関数は、次のとおりです。

IRR

IRR() は、チャート軸に対して反復する **value** による数式の数値で表される一連のキャッシュフローについて、集計された内部収益率を返します。

```
 IRR - チャート関数[TOTAL [<fld {,fld}>]] value)
```

NPV

NPV() は、チャート軸に対して反復処理される **value** 内の数として表される期間あたりの **discount_rate**、将来の支払い (負の値)、および収入 (正の値) に基づく投資について、集計された正味現在価値を返します。支払いと収入は、各期末に発生するとみなされます。

```
NPV - チャート関数([TOTAL [<fld {,fld}>]] discount_rate, value)
```

XIRR

XIRR() は、チャート軸で反復処理された **pmt** と **date** の数値ペアで表されるキャッシュフロー計算書 (不定期の場合もあります) の内部収益率 (年次) を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

```
XIRR - チャート関数([TOTAL [<fld {,fld}>]] pmt, date)
```

XNPV

XNPV() は、チャート軸で反復処理された **pmt** と **date** の数値ペアで表されるキャッシュフロー計算書 (不定期の場合もあります) の正味現在価値を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

```
XNPV - チャート関数([TOTAL [<fld{,fld}>]] discount_rate, pmt, date)
```

IRR

IRR() は、group by 句で定義されたレコードで反復処理される数式の数で表される一連のキャッシュフローから集計された内部収益率を返します。

これらのキャッシュフローは、年金のように均等である必要はありませんが、毎月または毎年のように、定期的に発生しなければなりません。内部収益率は、定期的に発生する支払い (負の値) と収入 (正の値) からなる投資の利率です。この関数の計算には、正の値と負の値が少なくとも 1 つずつ必要です。

この関数は、内部利益率 (IRR) を計算するためにニュートン法の簡素化されたバージョンを使用します。

構文:

```
IRR(value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	メジャーの対象となるデータが含まれている数式または項目。

制限事項:

テキスト値、NULL 値、欠損値は無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

例と結果:

例と結果

例	年	IRR2013
Cashflow: LOAD 2013 as Year, * inline [Date|Discount|Payments 2013-01-01|0.1|-10000 2013-03-01|0.1|3000 2013-10-30|0.1|4200 2014-02-01|0.2|6800] (delimiter is '|'); Cashflow1: LOAD Year,IRR(Payments) as IRR2013 Resident Cashflow Group By Year;	2013	0.1634

IRR - チャート関数

IRR() は、チャート軸に対して反復する **value** による数式の数値で表される一連のキャッシュフローについて、集計された内部収益率を返します。

これらのキャッシュフローは、年金のように均等である必要はありませんが、毎月または毎年のように、定期的に発生しなければなりません。内部収益率は、定期的に発生する支払い (負の値) と収入 (正の値) からなる投資の利率です。この関数の計算には、正の値と負の値が少なくとも 1 つずつ必要です。

この関数は、内部利益率 (IRR) を計算するためにニュートン法の簡素化されたバージョンを使用します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	メジャーの対象となるデータが含まれている数式または項目。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

テキスト値、NULL 値、欠損値は無視されます。

例と結果:

例と結果

例	結果
`IRR (Payments)`	0.1634 支払いは定期的に発生するものとします (毎月など)。 *XIRR の例では、Date 項目で支払日が指定されている限り、支払いは定期的でなくても構いません。*

例で使用されているデータ:

```
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 '|');
```

参照先:

- *XIRR - チャート関数 (page 382)*
- *Aggr - チャート関数 (page 542)*

NPV

NPV() スクリプト関数は、期間順の割引率と複数の値を取得します。これらの計算では、インフロー (収入) はプラス、アウトフロー (将来の支払い) はマイナスの値であると仮定しています。これらは、各期間の終わりに発生します。

正味現在価値 (NPV) は、将来のキャッシュフローの現行合計価値を計算するのに使用されます。NPV を計算するには、各期間に対して将来のキャッシュフローを推測して、正しい割引率を決定する必要があります。**NPV()** スクリプト関数は、期間順の割引率と複数の値を取得します。これらの計算では、インフロー (収入) はプラス、アウトフロー (将来の支払い) はマイナスの値であると仮定しています。これらは、各期間の終わりに発生します。

構文:

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

戻り値データ型: 数値 既定では、結果は通貨としてフォーマットされます。

正味現在価値を計算する式:

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

次のような前提です:

- R_t = 単一期間 t 中の正味キャッシュインフロー/アウトフロー
- i = 代替投資で得られた割引率または利益率
- t = タイマー期間数

引数

引数	説明
discount_rate	**discount_rate** は、適用された割引のパーセントです。 値 0.1 は、10% の割引率を示します。
value	この項目は、複数の期間を期間順に並べた値を保持します。最初の値は、期間 1 のキャッシュフローというように見なされます。

制限事項:

NPV() 関数には次の制限事項があります:

- テキスト値、NULL 値、欠損値は無視されます。
- キャッシュフロー値は、期間を昇順に並べる必要があります。

使用に適しているケース

NPV() は、プロジェクトの収益性をチェックし、他のメジャーを派生させるために使用される財務関数です。この関数は、キャッシュフローが生データとして存在する場合に有用です。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 - 単一の支払い (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 1 件のプロジェクトと1 期間のそのキャッシュフローのデータセットで、CashFlow という名前のテーブルにロードされます。
- CashFlow テーブルからの resident load で、NPV という名前のプロジェクトの NPV 項目を計算するのに使用されます。

- ハードコードされた割引率 10% で、これは NPV 計算で使用されます。
- Group By ステートメント、これはプロジェクトのすべての支払いをまとめるのに使用されます。

ロード スクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- PrjId
- NPV

結果テーブル

PrjId	NPV
1	$909.09

1 回の支払いで期末に 1000 ドルを受け取る場合、期間あたりの割引率を 10% とすると、NPV は 1000 ドルを (1 + 割引率) で除算した値になります。有効な NPV は 909.09 ドルと同値です。

例 2 – 複数支払い (スクリプト)

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 1 件のプロジェクトと複数期間のそのキャッシュフローのデータセットで、CashFlow という名前のテーブルにロードされます。

- CashFlow テーブルからの resident load で、NPV という名前のプロジェクトの NPV 項目を計算するのに使用されます。
- ハードコードされた割引率 10% (0.1) で、これは NPV 計算で使用されます。
- Group By ステートメント、これはプロジェクトのすべての支払いをまとめるのに使用されます。

ロード スクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- PrjId
- NPV

結果テーブル

PrjId	NPV
1	$1735.54

支払いで 2 期間の末に 1000 ドルを受け取る場合、期間あたりの割引率を 10% とすると、有効な NPV は 1735.54 ドルになります。

例 3 – 複数支払い (スクリプト)

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 2 つのプロジェクトの割引率で、Project というテーブルにロードされます。
- プロジェクトID と期間 ID 別の、各プロジェクトに対する複数の期間のキャッシュフロー。この期間 ID は、データを順序付けしない場合に、レコードを順序付けするのに使用できます。
- 臨時テーブル tmpNPV を作成するための、NoConcatenate、Resident loads、および Left Join 関数の組み合わせ。テーブルは、Project とCashFlow テーブルを組み合わせて 1 つのフラットテーブルにします。このテーブルでは、各期間に対して割引率が繰り返されます。
- tmpNPV テーブルからの resident load で、NPV という名前の各プロジェクトの NPV 項目を計算するのに使用されます。
- 各プロジェクトに関連付けられた単一値の割引率。これは、only() 関数を使って取得され、各プロジェクトに対する NPV 計算で使用されます。
- Group By ステートメント、これはプロジェクトID 別に各プロジェクトのすべての支払いをまとめるのに使用されます。

データモデルに合計または重複データがロードされないようにするため、tmpNPV テーブルがスクリプトの終わりに削除されます。

ロードスクリプト

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

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

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

NPV:
Load
       PrjId,
       NPV(Only(Discount_Rate),Values) as NPV //Discount Rate will be 10% for Project 1 and 15% for
```

```
Project 2
Resident tmpNPV
Group By PrjId;

Drop table tmpNPV;
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- PrjId
- NPV

結果テーブル

PrjId	NPV
1	$2486.85
2	$2042.12

プロジェクトID 1では、期間あたりの割引率 10% で、3 期間の末に支払額 1000 ドルを受け取ると予想されます。そのため、有効な NPV は 2486.85 です。

プロジェクトID 2では、割引率 15% で、4 期間に 500 ドルの支払いを 2 回と、さらに 1000 ドルの支払いを 2 回受け取ると予想されます。そのため、有効な NPV は 2042.12 です。

例 4 – プロジェクトの収益率の例 (例)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2 つのプロジェクトの割引率と初期投資額 (期間 0) で、Project というテーブルにロードされます。
- プロジェクトID と期間 ID 別の、各プロジェクトに対する複数の期間のキャッシュフロー。この期間 ID は、データを順序付けしない場合に、レコードを順序付けするのに使用できます。
- 臨時テーブル tmpNPV を作成するための、NoConcatenate、Resident loads、および Left Join 関数の組み合わせ。テーブルは、Project と CashFlow テーブルを組み合わせて 1 つのフラットテーブルにします。このテーブルでは、各期間に対して割引率が繰り返されます。
- 各プロジェクトに関連付けられた単一値の割引率で、これは、only() 関数を使って取得され、各プロジェクトに対する NPV 計算で使用されます。
- tmpNPV テーブルからの resident load で、NPV という名前の各プロジェクトの NPV 項目を計算するのに使用されます。
- プロジェクトの収益性指数を算出するために、各プロジェクトの NPV を初期投資額で割る追加項目。

- group by ステートメント (プロジェクト ID 別にグループ化) は、各プロジェクトのすべての支払いをまとめるのに使用されます。

データモデルに合計または重複データがロードされないようにするため、tmpNPV テーブルがスクリプトの終わりに削除されます。

ロード スクリプト

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

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

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

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

Drop table tmpNPV;
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- PrjId
- NPV

次のメジャーを作成します:

```
=only(Profitability_Index)
```

結果テーブル

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

プロジェクトID 1の有効NPVは87039.82ドルで、初期投資額は100000ドルです。そのため、収益性指数は0.87となります。これは1より小さいため、プロジェクトには収益性がありません。

プロジェクトID 2の有効NPVは123513.71ドルで、初期投資額は100000ドルです。そのため、収益性指数は1.24となります。これは1より大きいため、プロジェクトは収益性があるということです。

NPV - チャート関数

NPV() は、チャート軸に対して反復処理される**value**内の数として表される期間あたりの**discount_rate**、将来の支払い(負の値)、および収入(正の値)に基づく投資について、集計された正味現在価値を返します。支払いと収入は、各期末に発生するとみなされます。

構文:

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

戻り値データ型: 数値 既定では、結果は通貨としてフォーマットされます。

引数:

引数

引数	説明
discount_rate	**discount_rate**は、適用された割引のパーセントです。
value	メジャーの対象となるデータが含まれている数式または項目。

引数	説明
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {.fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。 **TOTAL** 修飾子の後には、山括弧で囲んだ 1 つ以上の項目名のリストを続けることができます。これらの項目名は、チャート軸の変数のサブセットにする必要があります。この場合、リストされているものを除き、すべてのチャート軸の変数を無視して計算が行われます。つまり、リストされている軸項目の項目値の組み合わせごとに 1 つの値が返されます。また、現在、チャートの軸ではない項目もリストに含めることができます。これは、軸項目が固定されていない場合に、軸をグループ化する場合に役立ちます。グループ内の変数がすべてリストされている場合、この関数はドリルダウンレベルが変更されても機能します。

制限事項:

discount_rate および **value** の内部集計に **TOTAL** 修飾子が含まれる場合を除き、これらに集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

テキスト値、NULL 値、欠損値は無視されます。

例と結果:

例と結果

例	結果
`NPV(Discount, Payments)`	-$540.12

例で使用されているデータ:

```
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 '|');
```

参照先:

- *XNPV - チャート関数 (page 391)*
- *Aggr - チャート関数 (page 542)*

XIRR

XIRR() は、group by 句で定義されたレコードで反復処理される **pmt** と **date** のペア数値で表されたキャッシュフロー計算書 (不定期の場合もあります) の集計された内部収益率 (年次) を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

Qlik の XIRR 機能 (**XIRR()** および **RangeXIRR()** 関数) は、次の方程式を使用して `Rate` 値を解き、正しい XIRR 値を決定します。

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

この方程式は、ニュートン法の簡素化されたバージョンを使用して解かれます。

構文:

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

戻り値データ型: 数値

引数

引数	説明
pmt	支払い。**date** で指定された支払いスケジュールに対応するキャッシュフローを含む式またはフィールド。
date	**pmt** で指定された支払いキャッシュフローに対応する数式または項目で、支払日が含まれます。

この関数を使用する場合は、次の制限が適用されます。

- 一対のデータのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべての対となるデータが無視されます。
- この関数には、少なくとも 1 つの有効なマイナスの支払いと1 つの有効なプラスの支払い (対応する有効日付付き) が必要です。これらの支払いが入力されない場合、NULL 値が返されます。

これらのトピックは、この関数を使用するのに役立つかもしれません。

- *XNPV (page 385)*: この関数は、キャッシュフローのスケジュールの集計正味現行価値を計算するために使用します。
- *RangeXIRR (page 1343)*: **RangeXIRR()** は **XIRR()** 関数と同等の範囲関数です。

Qlik Sense Client-Managed のバージョンが異なると、この関数で使用される基になるアルゴリズムが異なります。アルゴリズムの最近のアップデートについて詳しくは、サポート記事「XIRR 関数の修正とアップデート」を参照してください。

例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 一連のキャッシュフローのトランザクションデータ。
- これらのキャッシュフローの内部年次履歴率を計算するための **XIRR()** 関数の使用。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- Year
- XIRR2013

結果テーブル

年	XIRR2013
2013	0.5385

XIRR 戻り値の解釈

XIRR 機能は、通常、投資の分析に使用され、当初は流出 (マイナス) の支払いがあり、その後、小さな収入 (プラス) の支払いが連続して発生します。マイナスの支払い1件とプラスの支払い1件を持つ簡単な例:

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

最初に100を支払うと、ちょうど1年後に110が戻ってきます。これは、1年あたり10%の利益率を示します。`XIRR(Payments, Date)` は 0.1 の値を戻します。

XIRR 機能の戻り値はプラスとマイナスのどちらでもかまいません。投資の場合、マイナスの結果は投資に損失が出ていることを示します。損益の金額は、支払いフィールドに対して合計集計を行うだけで計算できます。

上記の例では、1 年間融資を行っています。利益率は利率とみなすことができます。また、自分が取引の相手側 (例えば、貸し手ではなく借り手) になった場合も、XIRR の機能を利用できます。

この例を考慮します:

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

これは、最初の例と同じですが、逆になっています。ここでは、1 年間に 100 を借り、10% の利息を付けて返済します。この例では、XIRR 計算は 0.1 (10%) を返します。これは、最初の例と同じ値です。

最初の例では 10 の利益を得て、2 番目の例では 10 の損失が生じましたが、XIRR 機能の戻り値は両方の例でプラスになっていることに注意してください。これは、取引のどちら側にいても、XIRR 機能が取引の隠れ利息を計算するたmです。

複数ソリューションの制限

Qlik の XIRR 機能は、次のソリューションにより定義され、Rate 値が解決されます。

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

この方程式は、複数の解を持つことができる場合があります。これは「複数 IRR 問題」と呼ばれ、非正規のキャッシュフローの流れ (非従来型キャッシュフローとも呼ばれる) によって引き起こされます。次のロードスクリプトにはこの例が表示されます。

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

この例では、マイナスの解が 1 つとプラスの解が 1 つあります (Rate = -0.3 と Rate = 0.8)。**XIRR()** は 0.8 を返します。

Qlik の XIRR 機能が解を検索すると、Rate = 0 から開始され、解が見つかるまで段階的にレートが増加します。プラスの解が複数ある場合、発生した最初の解が返されます。正のソリューションが見つからない場合、Rate をゼロにリセットし、マイナスの方向の解の検索を開始します。

なお「正常」なキャッシュフローストリームは、解が 1 つしかないことが保証されています。「正常」なキャッシュフローの流れとは、同じ符号 (プラスまたはマイナス) を持つすべての支払いが連続したグループであることを意味します。

参照先:

- *XNPV (page 385)*

RangeXIRR (page 1343)
XIRR 関数の修正およびアップデート

XIRR - チャート関数

XIRR() は、チャート軸で反復処理された **pmt** と **date** の数値ペアで表されるキャッシュフロー計算書 (不定期の場合もあります) の内部収益率 (年次) を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

Qlik の XIRR 機能 (**XIRR()** および **RangeXIRR()** 関数) は、次の方程式を使用して `Rate` 値を解き、正しい XIRR 値を決定します。

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

この方程式は、ニュートン法の簡素化されたバージョンを使用して解かれます。

構文:

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

戻り値データ型: 数値

引数

引数	説明
pmt	支払い。**date** で指定された支払いスケジュールに対応するキャッシュフローを含む式またはフィールド。
date	**pmt** で指定された支払いキャッシュフローに対応する数式または項目で、支払日が含まれます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

この関数を使用する場合は、次の制限が適用されます。

- **pmt** および **date** の内部集計に **TOTAL** 修飾子が含まれる場合を除き、これらに集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。
- データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。
- この関数には、少なくとも 1 つの有効なマイナスの支払いと1 つの有効なプラスの支払い (対応する有効日付付き) が必要です。これらの支払いが入力されない場合、NULL 値が返されます。

これらのトピックは、この関数を使用するのに役立つかもしれません。

- *XNPV - チャート関数 (page 391)*: この関数は、キャッシュフローのスケジュールの集計正味現行価値を計算するために使用します。
- *RangeXIRR (page 1343)*: **RangeXIRR()** は **XIRR()** 関数と同等の範囲関数です。

Qlik Sense Client-Managed のバージョンが異なると、この関数で使用される基になるアルゴリズムが異なります。アルゴリズムの最近のアップデートについて詳しくは、サポート記事「XIRR 関数の修正とアップデート」を参照してください。

例

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- キャッシュフロー取引を含むデータセット。
- 情報は、Cashflow というテーブルに保存されます。

ロードスクリプト

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

結果

次の手順を実行します。

データをロードしてシートを開きます。新しいテーブルを作成し、メジャーとして次の計算を追加します。

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

結果テーブル

=XIRR(Payments, Date)
0.5385

XIRR 戻り値の解釈

XIRR 機能は、通常、投資の分析に使用され、当初は流出 (マイナス) の支払いがあり、その後、小さな収入 (プラス) の支払いが連続して発生します。マイナスの支払い 1 件とプラスの支払い 1 件を持つ簡単な例:

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

最初に 100 を支払うと、ちょうど 1 年後に 110 が戻ってきます。これは、1 年あたり10% の利益率を示します。XIRR(Payments, Date) は 0.1 の値を戻します。

XIRR 機能の戻り値はプラスとマイナスのどちらでもかまいません。投資の場合、マイナスの結果は投資に損失が出ていることを示します。損益の金額は、支払いフィールドに対して合計集計を行うだけで計算できます。

上記の例では、1 年間融資を行っています。利益率は利率とみなすことができます。また、自分が取引の相手側 (例えば、貸し手ではなく借り手) になった場合も、XIRR の機能を利用できます。

この例を考慮します:

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

これは、最初の例と同じですが、逆になっています。ここでは、1 年間に 100 を借り、10% の利息を付けて返済します。この例では、XIRR 計算は 0.1 (10%) を返します。これは、最初の例と同じ値です。

最初の例では 10 の利益を得て、2 番目の例では 10 の損失が生じましたが、XIRR 機能の戻り値は両方の例でプラスになっていることに注意してください。これは、取引のどちら側にいても、XIRR 機能が取引の隠れ利息を計算するたmです。

複数ソリューションの制限

Qlik の XIRR 機能は、次のソリューションにより定義され、Rate 値が解決されます。

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

この方程式は、複数の解を持つことができる場合があります。これは「複数 IRR 問題」と呼ばれ、非正規のキャッシュフローの流れ (非従来型キャッシュフローとも呼ばれる) によって引き起こされます。次のロードスクリプトにはこの例が表示されます。

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

この例では、マイナスの解が 1 つとプラスの解が 1 つあります (Rate = -0.3 と Rate = 0.8)。**XIRR()** は 0.8 を返します。

Qlik の XIRR 機能が解を検索すると、Rate = 0 から開始され、解が見つかるまで段階的にレートが増加します。プラスの解が複数ある場合、発生した最初の解が返されます。正のソリューションが見つからない場合、Rate をゼロにリセットし、マイナスの方向の解の検索を開始します。

なお「正常」なキャッシュフローストリームは、解が 1 つしかないことが保証されています。「正常」なキャッシュフローの流れとは、同じ符号 (プラスまたはマイナス) を持つすべての支払いが連続したグループであることを意味します。

参照先:

- *IRR - チャート関数 (page 369)*
- *Aggr - チャート関数 (page 542)*
- XIRR 関数の修正およびアップデート

XNPV

XNPV() は、**pmt** と**date** のペア数値で表されるキャッシュフロー計算書 (不定期の場合もあります) の集計された正味現在価値を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

構文:

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

戻り値データ型: 数値

> *既定では、結果は通貨としてフォーマットされます。*

XNPV を計算するための式は次の通りです:

XNPV 集計式

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

次のような前提です:

- P_i = 単一期間 i 中の正味キャッシュインフロー/アウトフロー
- d_1 = 最初の支払日
- d_i = $i^{番目}$の支払日
- rate = 割引率

正味現在価値 (NPV) は、所定の割引率を使って将来のキャッシュフローの現行合計価値を計算するのに使用されます。XNPV を計算するには、対応する日付で詳細のキャッシュフローを推定する必要があります。この後、支払いのたびに、支払い日付に基づいて複利割引率を適用します。

一連の支払いに対して XNPV 集計を実行することは、それらの支払いに対して Sum 集計を行うことと似ています。違いは、各金額が、選択した割引率 (利息と類似) と将来のいつまで支払うかにに従って修正 (または「割引」) されることです。**discount_rate** パラメーターをゼロに設定して XNPV を実行すると、XNPV が Sum 演算子と同等になります (支払いは合計される前に修正されます)。一般的に、**discount_rate** が設定される値がゼロに近いほど、XNPV 結果が Sum 集計の結果に類似するようになります。

引数

引数	説明
discount_rate	**discount_rate** は、支払いが割引されるべき年率です。 値 0.1 は、10% の割引率を示します。
pmt	支払い。**date** で指定された支払いスケジュールに対応するキャッシュフローを含む式またはフィールド。プラスは入金、マイナスは出金とみなされます。 ***XNPV()*** *は、開始日に必ず発生するため、初期キャッシュフローを割引しません。その後の支払いは、年 365 日の日割り計算で割り引かれます。これは、最初の支払いも割り引かれる* ***NPV()*** *とは異なります。*
date	**pmt** で指定された支払いキャッシュフローに対応する数式または項目で、支払日が含まれます。最初の値は、将来のキャッシュフローのオフセットを計算するための開始日として使用されます。

この関数を使用する場合は、次の制限が適用されます。

- データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

使用に適しているケース

- XNPV() は、財務モデリングにおいて、投資機会の正味現在価値 (NPV) を計算するのに使用されます。
- 精度が高いため、あらゆるタイプの財務モデルについて XNPV のほうが NPV より好まれています。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 - 単一の支払い (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 1件のプロジェクトと1年のそのキャッシュフローのデータセットで、CashFlowという名前のテーブルに入っています。計算の開始日は2022年7月1日に設定されており、正味キャッシュフローは0です。1年後、1000ドルのキャッシュフローが発生します。
- CashFlowテーブルからのresident loadで、XNPVという名前のプロジェクトのXNPV項目を計算するのに使用されます。
- ハードコードされた割引率10% (0.1)で、これはXNPV計算で使用されます。
- Group Byステートメントは、プロジェクトのすべての支払いをまとめるのに使用されます。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- PrjId
- XNPV

結果テーブル

PrjId	XNPV
1	$909.09

式によると、1番目のレコードのXNPV値は0、2番目のレコードのXNPV値は909.09ドルです。そのため、XNPVの合計は909.09ドルとなります。

例 2 – 複数支払い (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 1 件のプロジェクトと1 年のそのキャッシュフローのデータセットで、CashFlow という名前のテーブルに入っています。
- CashFlow テーブルからの resident load で、XNPV という名前のプロジェクトの XNPV 項目を計算するのに使用されます。
- ハードコードされた割引率 10% (0.1) で、これは XNPV 計算で使用されます。
- Group By ステートメントは、プロジェクトのすべての支払いをまとめるのに使用されます。

ロードスクリプト

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

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

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- PrjId
- XNPV

結果テーブル

PrjId	XNPV
1	$1322.21

この例では、1 年目の終わりに 1000 ドル、2 年目の終わりに 500 ドルの支払いを受けています。期間あたりの割引率が 10% であるため、有効な XNPV は 1322.21 となります。

計算の基準日を指すのは最初の行のデータのみであることに注意してください。その他の行については、データパラメータが経過期間を計算するのに使用されるため、順序は重要ではありません。

例 3 – 複数支払いと不定期のキャッシュフロー (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- `Project` というテーブルの 2 つのプロジェクトの割引率。
- プロジェクト ID と日付別の、各プロジェクトに対する複数の期間のキャッシュフロー。`Dates` 項目は、割引率がキャッシュフローに割り当てられる期間を計算するのに使用されます。最初のレコード (最初のキャッシュフローと日付) を除けば、レコードの順序は重要ではなく、変更しても計算には影響しません。
- `NoConcatenate`、Resident loads、および `Left Join` 関数、臨時テーブルの組み合わせを使って、`tmpNPV` が作成されます。これは、1 つのフラットテーブルに `Project` と `CashFlow` テーブルのレコードを組み合わせます。このテーブルでは、各キャッシュフローに対して割引率が繰り返されます。
- `tmpNPV` テーブルからの resident load で、`XNPV` という名前の各プロジェクトの `XNPV` 項目を計算するのに使用されます。
- 各プロジェクトに関連付けられた単一値の割引率で、これは、`only()` 関数を使って取得され、各プロジェクトに対する `XNPV` 計算で使用されます。
- `Group By` ステートメント (プロジェクト ID 別にグループ化) は、各プロジェクトのすべての支払いと対応する日付をまとめるのに使用されます。
- データモデルに合計または重複データがロードされないようにするため、`tmpXNPV` テーブルがスクリプトの終わりに削除されます。

ロードスクリプト

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

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

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

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

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

Drop table tmpXNPV;
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- PrjId
- XNPV

結果テーブル

PrjId	XNPV
1	$1735.54
2	$278.36

プロジェクトID 1は、2021年7月1日に初期キャッシュフローが0ドルでした。期間あたりの割引率10%で、2年連続年度末に1000ドルの支払いを受け取ることになっています。そのため、有効なXNPVは1735.54です。

プロジェクトID 2は、2021年7月1日に初期キャッシュフローが1000ドルでした(そのためマイナス記号となっています)。2年後、500ドルが支払われることになっています。3年後、500ドルがさらに支払われます。最後に、2024年7月1日に1000ドルの支払いが見込まれています。割引率が15%であるため、有効なXNPVは278.36となります。

参照先:

- *Drop table (page 150)*
- *group by (page 160)*
- *Join (page 72)*
- *Max (page 334)*

NoConcatenate (page 90)
NPV - チャート関数 (page 377)
Only (page 344)

XNPV - チャート関数

XNPV() は、チャート軸で反復処理された **pmt** と **date** の数値ペアで表されるキャッシュフロー計算書 (不定期の場合もあります) の正味現在価値を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

構文:

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

戻り値データ型: 数値

既定では、結果は通貨としてフォーマットされます。

XNPV を計算するための式は次の通りです:

XNPV 集計式

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

次のような前提です:

- P_i = 単一期間 i 中の正味キャッシュインフロー/アウトフロー
- d_1 = 最初の支払日
- d_i = i番目の支払日
- rate = 割引率

正味現在価値 (NPV) は、所定の割引率を使って将来のキャッシュフローの現行合計価値を計算するのに使用されます。XNPV を計算するには、対応する日付で詳細のキャッシュフローを推定する必要があります。この後、支払いのたびに、支払い日付に基づいて複利割引率を適用します。

一連の支払いに対して XNPV 集計を実行することは、それらの支払いに対して Sum 集計を行うことと似ています。違いは、各金額が、選択した割引率 (利息と類似) と将来のいつまで支払うかにに従って修正 (または「割引」) されることです。**discount_rate** パラメーターをゼロに設定して XNPV を実行すると、XNPV が Sum 演算子と同等になります (支払いは合計される前に修正されます)。一般的に、**discount_rate** が設定される値がゼロに近いほど、XNPV 結果が Sum 集計の結果に類似するようになります。

引数

引数	説明
discount_rate	**discount_rate** は、支払いが割引されるべき年率です。 値 0.1 は、10% の割引率を示します。

引数	説明
pmt	支払い。**date** で指定された支払いスケジュールに対応するキャッシュフローを含む式またはフィールド。プラスは入金、マイナスは出金とみなされます。 ***XNPV()*** *は、開始日に必ず発生するため、初期キャッシュフローを割引しません。その後の支払いは、年 365 日の日割り計算で割り引かれます。これは、最初の支払いも割り引かれる* ***NPV()*** *とは異なります。*
date	**pmt** で指定された支払いキャッシュフローに対応する数式または項目で、支払日が含まれます。最初の値は、将来のキャッシュフローのオフセットを計算するための開始日として使用されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {.fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

この関数を使用する場合は、次の制限が適用されます。

- **discount_rate**、**pmt**、**date** の内部集計に **TOTAL** または **ALL** 修飾子が含まれるという場合を除き、これらに集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。
- データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

使用に適しているケース

- XNPV() は、財務モデリングにおいて、投資機会の正味現在価値 (NPV) を計算するのに使用されます。
- 精度が高いため、あらゆるタイプの財務モデルについて XNPV のほうが NPV より好まれています。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- キャッシュフロー取引を含むデータセット。
- 情報は、Cashflow というテーブルに保存されます。

ロードスクリプト

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

結果

次の手順を実行します。

データをロードしてシートを開きます。新しいテーブルを作成し、メジャーとして次の計算を追加します。

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

結果テーブル

=XNPV(0.09, Payments, Date)
$3062.49

参照先:

- *NPV - チャート関数 (page 377)*
- *Aggr - チャート関数 (page 542)*

統計集計関数

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

データロードスクリプトの統計集計関数

スクリプトで使用可能な統計集計関数は次のとおりです。

Avg

Avg()は、**group by**節で定義されたレコードの数式内の集計データの平均値を算出します。

```
Avg  ([distinct] expression)
```

Correl

Correl() は、**group by** 句で定義されたレコードで反復処理されるx-expression とy-expression のペア数値で表される一連の座標の、集計された相関係数を返します。

```
Correl (x-expression, y-expression)
```

Fractile

Fractile() は、**group by** 句で定義されたレコードの数式内の集計データの包括的 フラクタイル (分位値) に対応する値を算出します。

```
Fractile (expression, fractile)
```

FractileExc

FractileExc() は、**group by** 句で定義されたレコードの数式内の集計データの排他的 フラクタイル (分位値) に対応する値を算出します。

```
FractileExc (expression, fractile)
```

Kurtosis

Kurtosis() は、**group by** 句で定義されたレコードの数式内のデータの尖度を返します。

```
Kurtosis  ([distinct ] expression )
```

LINEST_B

LINEST_B() は、**group by** 句で定義されたレコードで反復処理されるx-expression とy-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計 b 値 (y 切片) を返します。

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

LINEST_df

LINEST_DF() は、**group by** 句で定義されたレコードで反復処理されるx-expression とy-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された自由度を返します。

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

LINEST_f

このスクリプト関数は、**group by** 節で定義された複数のレコードで反復処理されたx-expression とy-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計されたF 統計量 ($r^2/(1-r^2)$) を返します。

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

LINEST_m

LINEST_M() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された m 値 (傾き) を返します。

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

LINEST_r2

LINEST_R2() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された r^2 値 (決定係数) を返します。

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

LINEST_seb

LINEST_SEB() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された b 値の標準誤差を返します。

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

LINEST_sem

LINEST_SEM() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された m 値の標準誤差を返します。

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

LINEST_sey

LINEST_SEY() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された y 予測値の標準誤差を返します。

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

LINEST_ssreg

LINEST_SSREG() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された回帰変動を返します。

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

Linest_ssresid

LINEST_SSRESID() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された残差変動を返します。

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

Median

Median() は、**group by** 句で定義されたレコードの数式の集計された中央値を返します。

```
Median (expression)
```

Skew
Skew() は、**group by** 句で定義されたレコードの数式の歪度を返します。

```
Skew ([ distinct] expression)
```

Stdev
Stdev() は、**group by** 句で定義されたレコードの、数式によって得られた値の標準偏差を返します。

```
Stdev ([distinct] expression)
```

Sterr
Sterr() は、**group by** 句で定義されたレコードで反復処理される数式で表される一連の値に対して、集計標準誤差 (stdev/sqrt(n)) を返します。

```
Sterr  ([distinct] expression)
```

STEYX
STEYX() は、**group by** 句で定義された複数のレコードで反復処理されるx-expression とy-expression のペア数値で表される一連の座標について、回帰における各 X 値に対するy 予測値の集計された標準誤差を返します。

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

チャート式の統計集計関数

チャートで使用可能な統計集計関数は、次のとおりです。

Avg
Avg() は、チャート軸で反復処理された数式または項目の集計された平均を返します。

```
Avg - チャート関数({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]} expr)
```

Correl
Correl() は、2 つのデータセットの集計相関係数を返します。相関関数はデータセット間の関係を表すメジャーとして、チャート軸に対して反復処理される (x,y) 値のペアに対して集計されます。

```
Correl - チャート関数({[SetExpression] [TOTAL [<fld {, fld}>]]} value1, value2 )
```

Fractile
Fractile() は、チャート軸に対して反復処理された数式で指定された範囲において、集計データの包括的フラクタイル (分位値) に相当する値を返します。

```
Fractile - チャート関数({[SetExpression] [TOTAL [<fld {, fld}>]]} expr,
fraction)
```

FractileExc
FractileExc() は、チャート軸に対して反復処理された数式で指定された範囲において、集計データの排他的フラクタイル (分位値) に相当する値を返します。

```
FractileExc - チャート関数({[SetExpression] [TOTAL [<fld {, fld}>]]} expr,
fraction)
```

Kurtosis
Kurtosis() は、チャート軸で反復処理される数式または項目のデータを集計し、データ範囲の尖度を返します。

```
Kurtosis - チャート関数({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]]}
expr)
```

LINEST_b
LINEST_B() は、チャート軸で反復処理された数式 **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された b 値 (y 切片) を返します。

```
LINEST_R2 - チャート関数({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_
value[, y0_const[, x0_const]])
```

LINEST_df
LINEST_DF() は、チャート軸で反復処理された数式 **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された自由度を返します。

```
LINEST_DF - チャート関数({[SetExpression] [TOTAL [<fld{, fld}>]]} y_value, x_
value [, y0_const [, x0_const]])
```

LINEST_f
LINEST_F() は、チャート軸で反復処理された数式 **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の F 統計値 (r2/(1-r2)) を返します。

```
LINEST_F - チャート関数({[SetExpression] [TOTAL[<fld{, fld}>]]} y_value, x_value
[, y0_const [, x0_const]])
```

LINEST_m
LINEST_M() は数値チャート関数で、チャート軸で反復処理された数式 **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される直線回帰の集計された m 値 (傾き) を返します。

```
LINEST_M - チャート関数({[SetExpression] [TOTAL[<fld{, fld}>]]} y_value, x_value
[, y0_const [, x0_const]])
```

LINEST_r2
LINEST_R2() は、チャート軸で反復処理された **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された r2 値 (決定係数) を返します。

```
LINEST_R2 - チャート関数({[SetExpression] [TOTAL [<fld{ ,fld}>]] }y_value, x_
value[, y0_const[, x0_const]])
```

LINEST_seb
LINEST_SEB() は、チャート軸で反復処理された **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された b 値の標準誤差を返します。

LINEST_SEB - チャート関数({[SetExpression] [**TOTAL** [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]]**)**

LINEST_sem

LINEST_SEM() は、チャート軸で反復処理された**x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計されたm 値の標準誤差を返します。

LINEST_SEM - チャート関数([{set_expression}][**distinct**] [**total** [<fld {,fld}>]] y-expression, x-expression [, y0 [, x0]] **)**

LINEST_sey

LINEST_SEY() は、チャート軸で反復処理された**x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計されたy 予測値の標準誤差を返します。

LINEST_SEY - チャート関数({[SetExpression] [**TOTAL** [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]]**)**

LINEST_ssreg

LINEST_SSREG() は、チャート軸で反復処理された**x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された回帰変動を返します。

LINEST_SSREG - チャート関数({[SetExpression] [**TOTAL** [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]]**)**

LINEST_ssresid

LINEST_SSRESID()は、チャート軸で反復処理された数式 **x_value**と**y_value**の数値ペアで表される一連の座標に対して、方程式 y=mx+bで定義される線形回帰の残差変動を返します。

LINEST_SSRESID - チャート関数LINEST_SSRESID()は、チャート軸で反復処理された数式 x_value と y_valueの数値ペアで表される一連の座標に対して、方程式 y=mx+bで定義される線形回帰の残差変動を返します。 LINEST_SSRESID([{SetExpression}] [DISTINCT] [TOTAL [<fld{, fld}>]] y_value, x_value[, y0_const[, x0_const]])数値 引数引数説明y_valueメジャー対象である y 値の範囲が含まれている数式または項目です。x_valueメジャー対象である x 値の範囲が含まれている数式または項目です。y0, x0オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 と x0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 と x0 が記述されている場合は、データ ペアが 1 組あれば計算できます。 SetExpressionデフォルトでは、集計関数は選択されたレコード セットに対して集計を行います。Set 分析数式でレコード セットを定義することも可能です。 DISTINCT関数の引数の前に DISTINCT という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。 TOTAL関数の引数の前に TOTAL の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 TOTAL [<fld {.fld}>] (ここで、TOTAL 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 内部集計に TOTAL 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。 ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 Aggr を使用します。 データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべて

のデータペアが無視されます。 **An example of how to use linest functionsavg**
({[SetExpression] [**TOTAL** [<fld{ ,fld}>]] }y_value, x_value[, y0_const[, x0_const]]**)**

Median
Median() は、チャート軸で反復処理された数式の値を集計し、その範囲の中央値を返します。

Median - チャート関数({[SetExpression] [**TOTAL** [<fld{, fld}>]]} expr**)**

MutualInfo
MutualInfo は、2 つの項目間または **Aggr()** の集計値間の相互情報量 (MI) を計算します。

MutualInfo - チャート関数{[SetExpression] [**DISTINCT**] [**TOTAL** target, driver [, datatype [, breakdownbyvalue [, samplesize]]]**)**

Skew
Skew() は、チャート軸で反復処理された数式または項目の集計された歪度を返します。

Skew - チャート関数{[SetExpression] [**DISTINCT**] [**TOTAL** [<fld{ ,fld}>]]} expr**)**

Stdev
Stdev() は、チャート軸で反復処理された数式または項目のデータを集計し、データ範囲の標準偏差を返します。

Stdev - チャート関数({[SetExpression] [**DISTINCT**] [**TOTAL** [<fld{, fld}>]]} expr**)**

Sterr
Sterr() は、チャート軸で反復処理された数式の集計値の範囲に対して、平均値の標準誤差 (stdev/sqrt (n)) を返します。

Sterr - チャート関数({[SetExpression] [**DISTINCT**] [**TOTAL**[<fld{, fld}>]]} expr**)**

STEYX
STEYX() は、数式 **y_value** と**x_value** 数値ペアで表される一連の座標について、線形回帰の各 x 値に対して予想される y 値の集計された標準誤差を返します。

STEYX - チャート関数{[SetExpression] [**TOTAL** [<fld{, fld}>]]} y_value, x_value**)**

Avg

Avg()は、**group by**節で定義されたレコードの数式内の集計データの平均値を算出します。

構文:
Avg([**DISTINCT**] expr**)**

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
DISTINCT	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果
Temp: crosstable (Month, Sales) load * inline [Customer|Jan|Feb|Mar||Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec Astrida|46|60|70|13|78|20|45|65|78|12|78|22 Betacab|65|56|22|79|12|56|45|24|32|78|55|15 Canutility|77|68|34|91|24|68|57|36|44|90|67|27 Divadip|36|44|90|67|27|57|68|47|90|80|94] (delimiter is '|'); Avg1: LOAD Customer, Avg(Sales) as MyAverageSalesByCustomer Resident Temp Group By Customer;	Customer MyAverageSalesByCustomer Astrida 48.916667 Betacab 44.916667 Canutility 56.916667 Divadip 63.083333 これは、メジャーを含むテーブルを作成することで、シートで確認できます。 Sum(Sales)/12
Temp テーブルが前の例のようにロードされた場合: LOAD Customer,Avg(DISTINCT Sales) as MyAvgSalesDistinct Resident Temp Group By Customer;	Customer MyAverageSalesByCustomer Astrida 43.1 Betacab 43.909091 Canutility 55.909091 Divadip 61 ユニーク値のみカウントされています。合計を重複しない値の数で割ります。

Avg - チャート関数

Avg() は、チャート軸で反復処理された数式または項目の集計された平均を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

例と結果:

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

関数の例

例	結果
Avg(Sales)	軸 Customer およびメジャー Avg([Sales]) を含むテーブルで [合計] が表示されている場合、結果は 2566 になります。
Avg([TOTAL (Sales))	Customer のあらゆる値で 53.458333 になります。これは、TOTAL 修飾子を使うと軸が無視されるためです。
Avg(DISTINCT (Sales))	合計 51.862069 になります。これは、Distinct 修飾子を使うと、各 Customer で Sales の固有の値のみが評価されるためです。

例で使用されているデータ:

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

参照先:

Aggr - チャート関数 (page 542)

Correl

Correl() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標の、集計された相関係数を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value1, value2	相関係数を測定する2 つのサンプル セットを含む数式または項目。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果
Salary: Load *, 1 as Grp; LOAD * inline ["Employee name"|Gender|Age|Salary Aiden Charles|Male|20|25000 Brenda Davies|Male|25|32000 Charlotte Edberg|Female|45|56000 Daroush Ferrara|Male|31|29000 Eunice Goldblum|Female|31|32000 Freddy Halvorsen|Male|25|26000 Gauri Indu|Female|36|46000 Harry Jones|Male|38|40000 Ian Underwood|Male|40|45000 Jackie Kingsley|Female|23|28000] (delimiter is '|'); Correl1: LOAD Grp, Correl(Age,Salary) as Correl_Salary Resident Salary Group By Grp;	Correl_Salary 軸を持つテーブルでは、このデータロードスクリプトの Correl() 計算は、0.9270611 と表示されます。

Correl - チャート関数

Correl() は、2 つのデータセットの集計相関係数を返します。相関関数はデータセット間の関係を表すメジャーとして、チャート軸に対して反復処理される (x,y) 値のペアに対して集計されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value1, value2	相関係数を測定する2つのサンプル セットを含む数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL値、不明な値があると、すべてのデータペアが無視されます。

例と結果:

関数の例

例	結果
`Correl(Age, Salary)`	軸 `Employee name` およびメジャー `Correl(Age, Salary)` を含むテーブルでは、結果は0.9270611になります。結果は合計セルにのみ表示されます。
`Correl(TOTAL Age, Salary))`	0.927. これ以降は、読みやさの観点から小数点以下3桁で結果を示しています。 軸 Gender のフィルター パネルを作成し、そこから選択する場合、Female を選択すると0.951となり、Male を選択すると0.939となります。これは、他の Gender 値に属する結果がすべて除外されるためです。
`Correl({1} TOTAL Age, Salary))`	0.927. 選択に不依存。これは、set 数式 {1} により選択と軸が無視されるためです。

例	結果
Correl(TOTAL <Gender> Age, Salary))	合計セルの結果は 0.927、Male のすべての値は 0.939、Female のすべての値は 0.951 となります。これは、フィルター パネルで Gender に基づいて選択した結果に対応しています。

例で使用されているデータ:

```
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 '|');
```

参照先:

- *Aggr - チャート関数 (page 542)*
- *Avg - チャート関数 (page 400)*
- *RangeCorrel (page 1312)*

Fractile

Fractile() は、**group by** 句で定義されたレコードの数式内の集計データの包括的フラクタイル (分位値) に対応する値を算出します。

排他的フラクタイルの計算には、FractileExc (page 410) を使用します。

構文:

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

戻り値データ型: 数値

この関数は、rank = fraction * (N-1) + 1 で定義されたランクに対応する値を返します。N は expr の値の数です。rank が整数以外の場合は、最も近い 2 つの値の間で補間します。

引数:

引数

引数	説明
expr	フラクタイルを計算する際には、数式またはデータを含んだ項目を使用します。
fraction	計算対象となる分位数 (変位値) に相当する値 (0 ~ 1 の範囲内)。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果
Table1: Crosstable (Type, Value) Load recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Fractile1: LOAD Type, Fractile(Value,0.75) as MyFractile Resident Table1 Group By Type;	Type、MyFractile、Fractile() の軸を持つテーブルでは、このデータロードスクリプトの計算の結果は、次のようになります。 Type MyFractile Comparison 27.5 Observation 36

Fractile - チャート関数

Fractile() は、チャート軸に対して反復処理された数式で指定された範囲において、集計データの包括的フラクタイル (分位値) に相当する値を返します。

排他的フラクタイルの計算には、FractileExc - チャート関数 (page 412) を使用します。

構文:

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

戻り値データ型: 数値

この関数は、rank = fraction * (N-1) + 1 で定義されたランクに対応する値を返します。N は expr の値の数です。rank が整数以外の場合は、最も近い 2 つの値の間で補間します。

引数:

引数

引数	説明
expr	フラクタイルを計算する際には、数式またはデータを含んだ項目を使用します。
fraction	計算対象となる分位数 (変位値) に相当する値 (0 ~ 1 の範囲内)。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

例と結果:

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

関数の例

例	結果
`Fractile (Sales, 0.75)`	軸 `Customer` およびメジャー `Fractile([Sales])` を含むテーブルで [合計] が表示されている場合、結果は 71.75 になります。これは、`Sales` の値の分布にあるポイントで、値の 75% がこれを下回ります。

例	結果
Fractile (TOTAL Sales, 0.75))	Customer のあらゆる値で 71.75 になります。これは、TOTAL 修飾子を使うと軸が無視されるためです。
Fractile (DISTINCT Sales, 0.75)	合計 70 になります。これは、DISTINCT 修飾子を使うと、各 Customer で Sales の固有の値のみが評価されるためです。

例で使用されているデータ:

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

参照先:

- *Aggr - チャート関数 (page 542)*

FractileExc

FractileExc() は、**group by** 句で定義されたレコードの数式内の集計データの排他的フラクタイル (分位値) に対応する値を算出します。

包括的フラクタイルの計算には、Fractile (page 406) を使用します。

構文:

```
FractileExc(expr, fraction)
```

戻り値データ型: 数値

この関数は、rank = fraction * (N+1) で定義されたランクに対応する値を返します。N は expr の値の数です。rank が整数以外の場合は、最も近い 2 つの値の間で補間します。

引数:

引数

引数	説明
expr	フラクタイルを計算する際には、数式またはデータを含んだ項目を使用します。
fraction	計算対象となる分位数 (変位値) に相当する値 (0 ~ 1 の範囲内)。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果
Table1: Crosstable (Type, Value) Load recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Fractile1: LOAD Type, FractileExc(Value,0.75) as MyFractile Resident Table1 Group By Type;	Type、MyFractile、FractileExc() の軸を持つテーブルでは、このデータロードスクリプトの計算の結果は、次のようになります。 Type MyFractile Comparison 28.5 Observation 38

FractileExc - チャート関数

FractileExc() は、チャート軸に対して反復処理された数式で指定された範囲において、集計データの排他的フラクタイル (分位値) に相当する値を返します。

包括的フラクタイルの計算には、Fractile - チャート関数 (page 408) を使用します。

構文:

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

戻り値データ型: 数値

この関数は、rank = fraction * (N+1) で定義されたランクに対応する値を返します。N は expr の値の数です。rank が整数以外の場合は、最も近い 2 つの値の間で補間します。

引数:

引数

引数	説明
expr	フラクタイルを計算する際には、数式またはデータを含んだ項目を使用します。
fraction	計算対象となる分位数 (変位値) に相当する値 (0 ~ 1 の範囲内)。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

例と結果:

Example table

Customer	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Astrida	46	60	70	13	78	20	45	65	78	12	78	22
Betacab	65	56	22	79	12	56	45	24	32	78	55	15
Canutility	77	68	34	91	24	68	57	36	44	90	67	27
Divadip	57	36	44	90	67	27	57	68	47	90	80	94

関数の例

例	結果
`FractileExc (Sales, 0.75)`	軸 `Customer` およびメジャー `FractileExc([Sales])` を含むテーブルで **[合計]** が表示されている場合、結果は 75.25 になります。これは、`Sales` の値の分布にあるポイントで、値の 75% がこれを下回ります。

例	結果
FractileExc (TOTAL Sales, 0.75))	Customer のあらゆる値で 75.25 になります。これは、TOTAL 修飾子を使うと軸が無視されるためです。
FractileExc (DISTINCT Sales, 0.75)	合計 73.50 になります。これは、DISTINCT 修飾子を使うと、各 Customer で Sales の固有の値のみが評価されるためです。

例で使用されているデータ:

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

参照先:

Aggr - チャート関数 (page 542)

Kurtosis

Kurtosis() は、**group by** 句で定義されたレコードの数式内のデータの尖度を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果
Table1: Crosstable (Type, Value) Load recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Kurtosis1: LOAD Type, Kurtosis(Value) as MyKurtosis1, Kurtosis(DISTINCT Value) as MyKurtosis2 Resident Table1 Group By Type;	Type、MyKurtosis1、MyKurtosis2 の軸を持つテーブルでは、このデータロードスクリプトの Kurtosis() 計算の結果は、次のようになります。 Type MyKurtosis1 MyKurtosis2 Comparison -1.1612957 -1.4982366 Observation -1.1148768 -0.93540144

Kurtosis - チャート関数

Kurtosis() は、チャート軸で反復処理される数式または項目のデータを集計し、データ範囲の尖度を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

例と結果:

Example table

Type	Value																			
Comparison	2	27	38	31	1	19	1	34	3	1	2	3	2	1	2	1	3	29	37	2
Observation	35	40	12	15	21	14	46	10	28	48	16	30	32	48	31	22	12	39	19	25

関数の例

例	結果
Kurtosis (Value)	軸 Type およびメジャー Kurtosis(Value) を含むテーブルで [**合計**] が表示されている場合、数字の書式設定が有効桁数 3 に設定され、結果は 1.252 になります。Comparison は 1.161、Observation は 1.115 です。
Kurtosis (TOTAL Value))	Type のあらゆる値で 1.252 になります。これは、TOTAL 修飾子を使うと軸が無視されるためです。

例で使用されているデータ:

```
Table1:
Crosstable (Type, Value)
Load recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

参照先:

- *Avg - チャート関数 (page 400)*

LINEST_B

LINEST_B() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計 b 値 (y 切片) を返します。

構文:

```
LINEST_B (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 と x0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 と x0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_B - チャート関数

LINEST_B() は、チャート軸で反復処理された数式 **x_value** と **y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された b 値 (y 切片) を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0_const, x0_const	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_DF

LINEST_DF() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された自由度を返します。

構文:

```
LINEST_DF (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 と x0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 と x0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_DF - チャート関数

LINEST_DF() は、チャート軸で反復処理された数式 **x_value** と **y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された自由度を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_F

このスクリプト関数は、**group by** 節で定義された複数のレコードで反復処理された x-expression とy-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された F 統計量 ($r^2/(1-r^2)$) を返します。

構文:

```
LINEST_F (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_F - チャート関数

LINEST_F() は、チャート軸で反復処理された数式 **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の F 統計値 (r2/(1-r2)) を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_M

LINEST_M() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された m 値 (傾き) を返します。

構文:

```
LINEST_M (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_M - チャート関数

LINEST_M() は数値チャート関数で、チャート軸で反復処理された数式 **x_value** と**y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される直線回帰の集計された m 値 (傾き) を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_R2

LINEST_R2() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された r^2 値 (決定係数) を返します。

構文:

```
LINEST_R2 (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 と x0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 と x0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_R2 - チャート関数

LINEST_R2() は、チャート軸で反復処理された **x_value** と **y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された r2 値 (決定係数) を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_SEB

LINEST_SEB() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された b 値の標準誤差を返します。

構文:

```
LINEST_SEB (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 と x0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 と x0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_SEB - チャート関数

LINEST_SEB() は、チャート軸で反復処理された **x_value** と **y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された b 値の標準誤差を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_SEM

LINEST_SEM() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された m 値の標準誤差を返します。

構文:

```
LINEST_SEM (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 と x0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 と x0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_SEM - チャート関数

LINEST_SEM() は、チャート軸で反復処理された **x_value** と **y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された m 値の標準誤差を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_SEY

LINEST_SEY() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された y 予測値の標準誤差を返します。

構文:

```
LINEST_SEY (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数	説明
y_value	メジャー対象であるy値の範囲が含まれている数式または項目です。
x_value	メジャー対象であるx値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_SEY - チャート関数

LINEST_SEY() は、チャート軸で反復処理された **x_value** と **y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計されたy予測値の標準誤差を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象であるy値の範囲が含まれている数式または項目です。
x_value	メジャー対象であるx値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_SSREG

LINEST_SSREG() は、**group by** 句で定義されたレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された回帰変動を返します。

構文:

```
LINEST_SSREG (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 と x0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 と x0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 と x0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_SSREG - チャート関数

LINEST_SSREG() は、チャート軸で反復処理された **x_value** と **y_value** の数値ペアで表される一連の座標に対して、方程式 y=mx+b で定義される線形回帰の集計された回帰変動を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

LINEST_SSRESID

LINEST_SSRESID() は、**group by** 句で定義されたレコードで反復処理される x-expression とy-expression のペア数値で表される一連の座標に対して、数式 y=mx+b で定義される直線回帰の集計された残差変動を返します。

構文:

```
LINEST_SSRESID (y_value, x_value[, y0 [, x0 ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象であるy値の範囲が含まれている数式または項目です。
x_value	メジャー対象であるx値の範囲が含まれている数式または項目です。
y(0), x(0)	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*

LINEST_SSRESID - チャート関数

LINEST_SSRESID()は、チャート軸で反復処理された数式 **x_value**と**y_value**の数値ペアで表される一連の座標に対して、方程式 y=mx+bで定義される線形回帰の残差変動を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象であるy値の範囲が含まれている数式または項目です。
x_value	メジャー対象であるx値の範囲が含まれている数式または項目です。

引数	説明
y0, x0	オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。 *y0 とx0 の両方が記述されていない限り、この関数の計算には少なくとも 2 つの有効なデータペアが必要です。y0 とx0 が記述されている場合は、データペアが 1 組あれば計算できます。*
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

オプション値 y0 を記述することにより、y 軸上にある特定の点に回帰線を通すことができます。y0 とx0 の両方を記述すると、1 つの固定座標に回帰線を通すことができます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

参照先:

- *linest 関数の使用例 (page 459)*
- *Avg - チャート関数 (page 400)*

Median

Median() は、**group by** 句で定義されたレコードの数式の集計された中央値を返します。

構文:

```
Median (expr)
```

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。

例: 中央値を使用したスクリプトの数式

例 - スクリプト式

ロード スクリプト

この例のデータ ロード エディタで以下のインラインデータとスクリプト式をロードします。

```
Table 1:
Load RecNo() as RowNo, Letter, Number Inline
[Letter, Number
A,1
A,3
A,4
A,9
B,2
B,8
B,9];

Median:
LOAD Letter,
Median(Number) as MyMedian
Resident Table1 Group By Letter;
```

ビジュアライゼーションの作成

Qlik Sense シートに [**Letter**] と [**MyMedian**] を軸としたテーブルのビジュアライゼーションを作成します。

結果

Letter	MyMedian
A	3.5
B	8

説明

数値が最小から最大の順にソートされている場合、中央値は「中間」の数値と見なされます。データセットに偶数の値がある場合、関数は 2 つの中間値の平均を返します。この例では、中央値は **A** と **B** の値の各セットに対して計算されます。これはそれぞれ 3.5 と 8 です。

Median - チャート関数

Median() は、チャート軸で反復処理された数式の値を集計し、その範囲の中央値を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

例: 中央値を使用したチャートの数式

例 - チャートの数式

ロードスクリプト

以下のデータをインラインデータとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
Load RecNo() as RowNo, Letter, Number Inline
[Letter, Number
A,1
A,3
A,4
A,9
B,2
B,8
B,9];
```

ビジュアライゼーションの作成
Letter を軸として使用して、Qlik Sense シートにテーブルのビジュアライゼーションを作成します。

チャートの数式
次の数式をメジャーとしてテーブルに追加します。

```
Median(Number)
```

結果

Letter	Median(Number)
Totals	4
A	3.5
B	8

説明
数値が最小から最大の順にソートされている場合、中央値は「中間」の数値と見なされます。データセットに偶数の値がある場合、関数は 2 つの中間値の平均を返します。この例では、中央値は **A** と **B** の値の各セットに対して計算されます。これはそれぞれ 3.5 と 8 です。

Totals の中央値は、4 に等しいすべての値から計算されます。

参照先:

- *Avg - チャート関数 (page 400)*

MutualInfo - チャート関数

MutualInfo は、2 つの項目間または **Aggr()** の集計値間の相互情報量 (MI) を計算します。

MutualInfo は、2 つのデータセットについて集計された相互情報を返します。これにより、項目と潜在的な要因との間で主要因分析が可能になります。相関情報は、データセット間の関係を表すメジャーとして、チャート軸に対して反復処理される (x,y) ペア値に対して集計されます。相関情報は 0 ~ 1 の間で測定され、パーセンタイル値として書式設定できます。**MutualInfo** は、選択または set 数式によって定義されます。

MutualInfo を使用すると、さまざまな種類の MI 分析ができます。

- ペアワイズ MI: ドライバー項目とターゲット項目間の MI を計算します。
- 値別のドライバー内訳: MI は、ドライバーおよびターゲット項目で個別の項目値ごとに計算されます。
- 機能選択: グリッドチャートで **MutualInfo** を使用して、MI に基づいてすべての項目が互いに比較されるマトリックスを生成します。

MutualInfo は、相互に情報を共有する項目間の因果関係を必ずしも示すものではありません。2 つの項目は、相互に情報を共有していますが、お互いに同じ要因ではない場合があります。例えば、アイスクリームの売り上げと外の気温を比較した場合、**MutualInfo** はこの 2 つの間の相互情報を示します。外の気温がアイスクリームの売上をの要因になっているのか(その可能性は高い) 、アイスクリームの売上が外気温の要因になっているのか(その可能性は低い) を示すものではありません。

相互情報を計算する場合、関連付けは異なるテーブルの項目の値間の対応と頻度に影響します。

同じ項目や選択でも、返される値が若干異なる場合があります。これは、各 **MutualInfo** 呼び出しがランダムに選択されたサンプルで動作し、**MutualInfo** アルゴリズムに固有のランダム性があるためです。

MutualInfo は **Aggr()** 関数に適用できます。

構文:

```
MutualInfo({SetExpression}] [DISTINCT] [TOTAL] field1, field2 , datatype [, breakdownbyvalue [, samplesize ]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
field1, field2	相互情報を測定する 2 つのサンプル セットを含む数式または項目。
datatype	ターゲットおよびドライバーに含まれるデータ型は、 discrete:discrete の場合は 1 または 'dd' continuous:continuous の場合は 2 または 'cc' continuous:discrete の場合は 3 または 'cd' discrete:continuous の場合は 4 または 'dc' データタイプは大文字と小文字を区別しません。
breakdownbyvalue	ドライバー内の値に対応する静的値。入力すると、当該値に対する MI 貢献度が計算されます。**ValueList()** または **ValueLoop()** を使用できます。**Null()** を追加すると、ドライバー内の値すべてに対する MI 全体が計算されます。 値による内訳には、離散データを含むドライバーが必要です。
samplesize	ターゲットおよびドライバーからサンプリングするための値の数。サンプリングはランダムに行われます。**MutualInfo** には、最低 80 サンプル サイズが必要です。**MutualInfo** はリソースを集中的に使用するため、**MutualInfo** は既定で最大 10,000 のデータペアのみをサンプルします。サンプル サイズでさらに多くのデータペアをサンプルするように指定できます。**MutualInfo** がタイムアウトになる場合は、サンプル サイズを縮小してください。

引数	説明
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

関数の例

例	結果
`mutualinfo(Age, Salary, 1)`	軸 `Employee name` およびメジャー `mutualinfo(Age, Salary, 1)` を含むテーブルでは、結果は 0.99820986 になります。結果は合計セルにのみ表示されます。
`mutualinfo (TOTAL Age, Salary, 1, null (), 81)`	軸 Gender でフィルターパネルを作成して選択した場合、Female を選択した時は結果が 0.99805677 になり、Male を選択した時は 0.99847373 になります。これは、他の Gender 値に属する結果がすべて除外されるためです。
`mutualinfo (TOTAL Age, Gender, 1, ValueLoop(25,35))`	0.68196996.Gender からいずれかの値を選択すると、これは 0 になります。
`mutualinfo({1} TOTAL Age, Salary, 1, null ())`	0.99820986. これは選択に依存しません。set 数式 {1} は、すべての選択と軸を無視します。

例で使用されているデータ:

```
Salary:

LOAD * inline [

"Employee name"|Age|Gender|Salary
```

```
Aiden Charles|20|Male|25000

Ann Lindquist|69|Female|58000

Anna Johansen|37|Female|36000

Anna Karlsson|42|Female|23000

Antonio Garcia|20|Male|61000

Benjamin Smith|42|Male|27000

Bill Yang|49|Male|50000

Binh Protzmann|69|Male|21000

Bob Park|51|Male|54000

Brenda Davies|25|Male|32000

Celine Gagnon|48|Female|38000

Cezar Sandu|50|Male|46000

Charles Ingvar Jönsson|27|Male|58000

Charlotte Edberg|45|Female|56000

Cindy Lynn|69|Female|28000

Clark Wayne|63|Male|31000

Daroush Ferrara|31|Male|29000

David Cooper|37|Male|64000

David Leg|58|Male|57000

Eunice Goldblum|31|Female|32000

Freddy Halvorsen|25|Male|26000

Gauri Indu|36|Female|46000

George van Zaant|59|Male|47000

Glenn Brown|58|Male|40000

Harry Jones|38|Male|40000

Helen Brolin|52|Female|66000

Hiroshi Ito|24|Male|42000
```

```
Ian Underwood|40|Male|45000

Ingrid Hendrix|63|Female|27000

Ira Baumel|39|Female|39000

Jackie Kingsley|23|Female|28000

Jennica Williams|36|Female|48000

Jerry Tessel|31|Male|57000

Jim Bond|50|Male|58000

Joan Callins|60|Female|65000

Joan Cleaves|25|Female|61000

Joe Cheng|61|Male|41000

John Doe|36|Male|59000

John Lemon|43|Male|21000

Karen Helmkey|54|Female|25000

Karl Berger|38|Male|68000

Karl Straubaum|30|Male|40000

Kaya Alpan|32|Female|60000

Kenneth Finley|21|Male|25000

Leif Shine|63|Male|70000

Lennart Skoglund|63|Male|24000

Leona Korhonen|46|Female|50000

Lina André|50|Female|65000

Louis Presley|29|Male|36000

Luke Langston|50|Male|63000

Marcus Salvatori|31|Male|46000

Marie Simon|57|Female|23000

Mario Rossi|39|Male|62000

Markus Danzig|26|Male|48000
```

```
Michael Carlen|21|Male|45000

Michelle Tyson|44|Female|69000

Mike Ashkenaz|45|Male|68000

Miro Ito|40|Male|39000

Nina Mihn|62|Female|57000

Olivia Nguyen|35|Female|51000

Olivier Simenon|44|Male|31000

Östen Ärlig|68|Male|57000

Pamala Garcia|69|Female|29000

Paolo Romano|34|Male|45000

Pat Taylor|67|Female|69000

Paul Dupont|34|Male|38000

Peter Smith|56|Male|53000

Pierre Clouseau|21|Male|37000

Preben Jørgensen|35|Male|38000

Rey Jones|65|Female|20000

Ricardo Gucci|55|Male|65000

Richard Ranieri|30|Male|64000

Rob Carsson|46|Male|54000

Rolf Wesenlund|25|Male|51000

Ronaldo Costa|64|Male|39000

Sabrina Richards|57|Female|40000

Sato Hiromu|35|Male|21000

Sehoon Daw|57|Male|24000

Stefan Lind|67|Male|35000

Steve Cioazzi|58|Male|23000

Sunil Gupta|45|Male|40000
```

```
Sven Svensson|45|Male|55000

Tom Lindwall|46|Male|24000

Tomas Nilsson|27|Male|22000

Trinity Rizzo|52|Female|48000

Vanessa Lambert|54|Female|27000

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

Skew

Skew() は、**group by** 句で定義されたレコードの数式の歪度を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
DISTINCT	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、軸として Type および *MySkew* を使用して、ストレートテーブルを構築します。

結果のデータ

例	結果
Table1: Crosstable (Type, Value) Load recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Skew1: LOAD Type, Skew(Value) as MySkew Resident Table1 Group By Type;	Skew() 計算の結果は次のとおりです。 • Type は MySkew • Comparison は 0.86414768 • Observation は 0.32625351

Skew - チャート関数

Skew() は、チャート軸で反復処理された数式または項目の集計された歪度を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。

引数	説明
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

例と結果:

アプリにスクリプト例を追加して実行します。軸として `Type` 、メジャーとして `Skew(Value)` を使用して、ストレートテーブルを作成します。

テーブルのプロパティで、`Totals` を有効にしてください。

例	結果

`Table1:`
`Crosstable (Type, Value)`
`Load recno() as ID, * inline [`
`Observation|Comparison`
`35|2`
`40|27`
`12|38`
`15|31`
`21|1`
`14|19`
`46|1`
`10|34`
`28|3`
`48|1`
`16|2`
`30|3`
`32|2`
`48|1`
`31|2`
`22|1`
`12|3`
`39|29`
`19|37`
`25|2 ] (delimiter is '|');`
	Skew(Value) 計算の結果は次のとおりです。
	
	• `Total` は 0.23522195
	• `Comparison` は 0.86414768
	• `Observation` は 0.32625351

参照先:

- *Avg - チャート関数 (page 400)*

Stdev

Stdev() は、**group by** 句で定義されたレコードの、数式によって得られた値の標準偏差を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、軸として `Type` および `MyStdev` を使用して、ストレートテーブルを構築します。

結果のデータ

例	結果
Table1: Crosstable (Type, Value) Load recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Stdev1: LOAD Type, Stdev(Value) as MyStdev Resident Table1 Group By Type;	Stdev() 計算の結果は次のとおりです。 • Type は MyStdev • Comparison は 14.61245 • Observation は 12.507997

Stdev - チャート関数

Stdev() は、チャート軸で反復処理された数式または項目のデータを集計し、データ範囲の標準偏差を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。

引数	説明
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

例と結果:

アプリにスクリプト例を追加して実行します。軸として`Type` 、メジャーとして `Stdev(Value)` を使用して、ストレートテーブルを作成します。

テーブルのプロパティで、`Totals` を有効にしてください。

例	結果

`Stdev(Value)`
`Table1:`
`Crosstable (Type, Value)`
`Load recno() as ID, * inline [`
`Observation|Comparison`
`35|2`
`40|27`
`12|38`
`15|31`
`21|1`
`14|19`
`46|1`
`10|34`
`28|3`
`48|1`
`16|2`
`30|3`
`32|2`
`48|1`
`31|2`
`22|1`
`12|3`
`39|29`
`19|37`
`25|2 ] (delimiter is '|');`
	Stdev(Value) 計算の結果は次のとおりです。
	
	• `Total` は 15.47529
	• `Comparison` は 14.61245
	• `Observation` は 12.507997

参照先:

- *Avg - チャート関数 (page 400)*
- *STEYX - チャート関数 (page 457)*

Sterr

Sterr() は、**group by** 句で定義されたレコードで反復処理される数式で表される一連の値に対して、集計標準誤差 (stdev/sqrt(n)) を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

制限事項:

テキスト値、NULL 値、欠損値は無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果
Table1: Crosstable (Type, Value) Load recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|'); Sterr1: LOAD Type, Sterr(Value) as MySterr Resident Table1 Group By Type;	Type および MySterr の軸を持つテーブルでは、このデータロードスクリプトの Sterr() 計算の結果は、次のようになります。 Type MySterr Comparison 3.2674431 Observation 2.7968733

Sterr - チャート関数

Sterr() は、チャート軸で反復処理された数式の集計値の範囲に対して、平均値の標準誤差 (stdev/sqrt (n)) を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。

引数	説明
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {.fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

テキスト値、NULL 値、欠損値は無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。軸として`Type` 、メジャーとして`Sterr(Value)` を使用して、ストレートテーブルを作成します。

テーブルのプロパティで、`Totals` を有効にしてください。

例	結果
Table1: Crosstable (Type, Value) Load recno() as ID, * inline [Observation|Comparison 35|2 40|27 12|38 15|31 21|1 14|19 46|1 10|34 28|3 48|1 16|2 30|3 32|2 48|1 31|2 22|1 12|3 39|29 19|37 25|2] (delimiter is '|');	Sterr(Value) 計算の結果は次のとおりです。 • Total は 2.4468583 • Comparison は 3.2674431 • Observation は 2.7968733

参照先:

- *Avg - チャート関数 (page 400)*
- *STEYX - チャート関数 (page 457)*

STEYX

STEYX() は、**group by** 句で定義された複数のレコードで反復処理される x-expression と y-expression のペア数値で表される一連の座標について、回帰における各 X 値に対する y 予測値の集計された標準誤差を返します。

構文:

```
STEYX (y_value, x_value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象である y 値の範囲が含まれている数式または項目です。
x_value	メジャー対象である x 値の範囲が含まれている数式または項目です。

制限事項:

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果
Trend: Load *, 1 as Grp; LOAD * inline [Month|KnownY|KnownX Jan|2|6 Feb|3|5 Mar|9|11 Apr|6|7 May|8|5 Jun|7|4 Jul|5|5 Aug|10|8 Sep|9|10 Oct|12|14 Nov|15|17 Dec|14|16] (delimiter is '|'); STEYX1: LOAD Grp, STEYX(KnownY, KnownX) as MySTEYX Resident Trend Group By Grp;	MySTEYX 軸を持つテーブルでは、このデータロードスクリプトの STEYX() 計算の結果は、2.0714764 になります。

STEYX - チャート関数

STEYX() は、数式 **y_value** と**x_value** 数値ペアで表される一連の座標について、線形回帰の各 x 値に対して予想される y 値の集計された標準誤差を返します。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
y_value	メジャー対象となる既知の y 値の範囲が含まれている数式および項目。
x_value	メジャー対象となる既知の x 値の範囲が含まれている数式および項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

内部集計に **TOTAL** 修飾子が含まれない限り、集計関数のパラメーターに他の集計関数を含めることはできません。ネストされた集計関数が必要な場合、指定された軸と組み合わせて高度な関数 **Aggr** を使用します。

データペアのどちらか、または両方にテキスト値、NULL 値、不明な値があると、すべてのデータペアが無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。軸として `KnownY` および `KnownX` 、メジャーとして `Steyx (KnownY,KnownX)` を使用して、ストレートテーブルを作成します。

テーブルのプロパティで、`Totals` を有効にしてください。

例	結果
Trend: LOAD * inline [Month|KnownY|KnownX Jan|2|6 Feb|3|5 Mar|9|11 Apr|6|7 May|8|5 Jun|7|4 Jul|5|5 Aug|10|8 Sep|9|10 Oct|12|14 Nov|15|17 Dec|14|16] (delimiter is '|');	STEYX(KnownY,KnownX) 計算の結果は 2.071 (数字の形式が小数点 3 桁に設定されている場合) です。

参照先:

- *Avg - チャート関数 (page 400)*
- *Sterr - チャート関数 (page 453)*

linest 関数の使用例

linest 関数は、直線回帰分析に関連した値の計算に使用します。このセクションでは、サンプル データを使って、Qlik Sense で使用可能な linest 関数の値を特定するためのビジュアライゼーションの作成方法を説明します。linest 関数は、データ ロード スクリプトおよびチャートの数式で使用できます。

構文と引数については、各 linest チャート関数およびスクリプト関数のトピックを参照してください。

例で使用されているデータとスクリプトの数式

下記の linest() 例をデータ ロード エディタで以下のインラインデータとスクリプトの数式をロードします。

```
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;
```

例 1: linest を使用したスクリプトの数式

例: スクリプトの数式

データ ロード スクリプトの計算からビジュアライゼーションを作成

以下の項目を列とし、Qlik Sense シートにテーブル ビジュアライゼーションを作成します。

- Linest_B
- Linest_DF
- Linest_F
- Linest_M

- Linest_R2
- Linest_SEB
- Linest_SEM
- Linest_SEY
- Linest_SSREG
- Linest_SSRESID

結果

データロードスクリプトで行われたlinest計算の結果が含まれるテーブルは、次のようになります。

結果テーブル

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

結果テーブル

Linest_SEM	Linest_SEY	Linest_SSREG	Linest_SSRESID
1.887	48.666	49235.014	42631.186

例2: linestを使用したチャートの数式

例: チャートの数式

以下の項目を軸とし、Qlik Senseシートにテーブル ビジュアライゼーションを作成します。

```
ValueList('Linest_b', 'Linest_df','Linest_f', 'Linest_m','Linest_r2','Linest_SEB','Linest_
SEM','Linest_SEY','Linest_SSREG','Linest_SSRESID')
```

この数式は、linest関数の名前を持つ軸のラベルを作成するために合成軸関数を使用します。ラベルを**Linest functions**に変更してスペースを節約することも可能です。

次の数式をメジャーとしてテーブルに追加します。

```
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) )
```

この数式は、合成軸内の対応する名前の各linest関数の結果値を表示します。`Linest_b(Y,X)`の結果は**linest_b**の隣に表示されます。

結果

結果テーブル

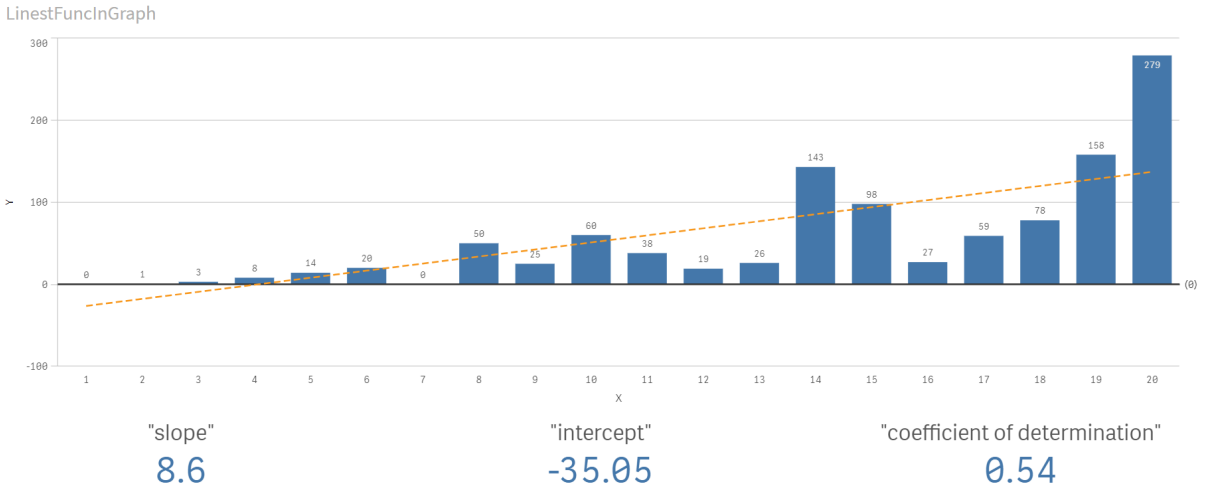
Linest functions	Linest function results
Linest_b	-35.047
Linest_df	18
Linest_f	20.788
Linest_m	8.605
Linest_r2	0.536
Linest_SEB	22.607
Linest_SEM	1.887
Linest_SEY	48.666
Linest_SSREG	49235.014
Linest_SSRESID	42631.186

例 3: linest を使用したチャートの数式

例: チャートの数式

1. **X** を軸、**Y** をメジャーとして、Qlik Sense シートにバーチャートのビジュアライゼーションを作成します。
2. Y メジャーに線形トレンドラインを追加します。
3. シートに KPI ビジュアライゼーションを追加します。
    1. KPI のラベルとして*傾き*を追加します。
    2. KPI の数式として `sum(Linest_M)` を追加します。
4. シートに 2 番目の KPI ビジュアライゼーションを追加します。
    1. KPI のラベルとして*切片*を追加します。
    2. KPI の数式として `Sum(Linest_B)` を追加します。
5. シートに 3 番目の KPI ビジュアライゼーションを追加します。
    1. KPI のラベルとして*決定係数*を追加します。
    2. KPI の数式として `Sum(Linest_R2)` を追加します。

結果



説明

バーチャートは、X および Y データのプロットを示しています。関連する linest() 関数は、トレンドラインが基づく線形回帰方程式の値、つまりy = m * x + b を提供します。この方程式は、「最小二乗」法を使用して、データに最適な線を表す配列を返すことにより、直線 (トレンドライン) を計算します。

KPI は、線形回帰方程式の変数である傾きの linest() 関数 **sum(Linest_M)** とY 切片の **sum(Linest_B)** の結果、および決定係数の対応する集計 R2 値を表示します。

統計検定関数

統計検定関数は、データロードスクリプトとチャートの数式の両方で使用できますが、構文が異なります。

カイ二乗検定関数

通常は質的変数の調査に使用します。一元度数表で観測度数と期待度数を比較したり、分割表で 2 つの変数の関係を調べることができます。

t 検定関数

t 検定関数は、2 つの母平均の統計学的検討に使用されます。2 サンプル t 検定は、2 つの標本が異なるものかどうかを調べます。これは、一般に、2 つの正規分布の分散が不明であり、かつ実験で小さな標本サイズが使用される場合に使用されます。

z 検定関数

2 つの母平均の統計学的検討を行います。2 サンプル z 検定は、2 つの標本が異なるものかどうかを調べます。これは、一般に、2 つの正規分布の分散が既知であり、かつ実験で大きな標本サイズが使用される場合に使用されます。

カイ二乗検定関数

通常は質的変数の調査に使用します。一元度数表で観測度数と期待度数を比較したり、分割表で 2 つの変数の関係を調べることができます。Chi-squared test functions are used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more groups. Often a histogram is used, and the different bins are compared to an expected distribution.

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

Chi2Test_chi2
Chi2Test_chi2() は、1 つまたは 2 つの一連の値に対して集計されたカイ ^{2} 乗検定の値を返します。

```
Chi2Test_chi2() は、1 つまたは 2 つの一連の値に対して集計されたカイ 2 乗検定の値を返します。
(col, row, actual_value[, expected_value])
```

Chi2Test_df
Chi2Test_df() は、1 つまたは 2 つの一連の値に対して集計されたカイ二乗検定の df 値 (自由度) を返します。

```
Chi2Test_df() は、1 つまたは 2 つの一連の値に対して集計されたカイ二乗検定の df 値 (自由度)
を返します。(col, row, actual_value[, expected_value])
```

Chi2Test_p
Chi2Test_p() は、1 つまたは 2 つの一連の値に対して集計されたカイ二乗検定の p 値 (有意性) を返します。

```
Chi2Test_p - チャート関数(col, row, actual_value[, expected_value])
```

参照先:

- *t 検定関数 (page 467)*
- *z 検定関数 (page 500)*

Chi2Test_chi2

Chi2Test_chi2() は、1 つまたは 2 つの一連の値に対して集計されたカイ ^{2} 乗検定の値を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

すべて Qlik Sense chi^2 検定の関数には、同じ引数が含まれています。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
col, row	検定対象の値マトリックスにおいて指定されている列と行を指します。
actual_value	指定した **col** および **row** でのデータの観測値です。
expected_value	指定した **col** および **row** での分布予想値です。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
Chi2Test_chi2( Grp, Grade, Count )
Chi2Test_chi2( Gender, Description, Observed, Expected )
```

参照先:

- *チャートでの chi2-test 関数の使用例 (page 514)*
- *データロードスクリプトでの chi2-test 関数の使用例 (page 518)*

Chi2Test_df

Chi2Test_df() は、1 つまたは 2 つの一連の値に対して集計されたカイ二乗検定の df 値 (自由度) を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

すべて Qlik Sense chi^2 検定の関数には、同じ引数が含まれています。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
col, row	検定対象の値マトリックスにおいて指定されている列と行を指します。
actual_value	指定した **col** および **row** でのデータの観測値です。
expected_value	指定した **col** および **row** での分布予想値です。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
Chi2Test_df( Grp, Grade, Count )
Chi2Test_df( Gender, Description, Observed, Expected )
```

参照先:

- *チャートでの chi2-test 関数の使用例 (page 514)*
- *データロードスクリプトでの chi2-test 関数の使用例 (page 518)*

Chi2Test_p - チャート関数

Chi2Test_p() は、1 つまたは 2 つの一連の値に対して集計されたカイ二乗検定の p 値 (有意性) を返します。検定は、指定された **col** と **row** マトリックスの変動を検定する **actual_value** の値を用いて、または **actual_value** の値を **expected_value** の対応値と比較することで実行されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

すべて Qlik Sense chi^2 検定の関数には、同じ引数が含まれています。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
col, row	検定対象の値マトリックスにおいて指定されている列と行を指します。
actual_value	指定した **col** および **row** でのデータの観測値です。
expected_value	指定した **col** および **row** での分布予想値です。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
Chi2Test_p( Grp, Grade, Count )
Chi2Test_p( Gender, Description, Observed, Expected )
```

参照先:

- *チャートでの chi2-test 関数の使用例 (page 514)*
- *データロードスクリプトでの chi2-test 関数の使用例 (page 518)*

t 検定関数

t 検定関数は、2 つの母平均の統計学的検討に使用されます。2 サンプル t 検定は、2 つの標本が異なるものかどうかを調べます。これは、一般に、2 つの正規分布の分散が不明であり、かつ実験で小さな標本サイズが使用される場合に使用されます。

以下のセクションでは、t 検定統計関数は、各関数タイプに適用される標本のスチューデント検定に基づいてグループ化されています。

標準的な t-test レポートの作成 (page 520)

2 つの独立標本による t 検定

次の関数は、2 つの独立標本のスチューデント t 検定に適用されます。

ttest_conf

TTest_conf は、2 つの独立した一連の値に対して集計された t検定信頼区間値を返します。

TTest_conf は、2 つの独立した一連の値に対して集計された t 検定信頼区間値を返します。 (grp, value [, sig[, eq_var]]**)**

ttest_df

TTest_df() は、2 つの独立した一連の値に対して集計されたスチューデント t 検定値 (自由度) を返します。

TTest_df() は、2 つの独立した一連の値に対して集計されたスチューデント t 検定値 (自由度) を返します。 (grp, value [, eq_var)

ttest_dif
TTest_dif() は数値関数で、2 つの独立した一連の値に対して集計されたスチューデントt 検定の平均の差を返します。

TTest_dif() は数値関数で、2 つの独立した一連の値に対して集計されたスチューデント t 検定の平均の差を返します。 (grp, value)

ttest_lower
TTest_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。

TTest_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。 (grp, value [, sig[, eq_var]])

ttest_sig
TTest_sig() は、2 つの独立した一連の値に対して集計されたスチューデントt 検定の両側有意水準を返します。

TTest_sig() は、2 つの独立した一連の値に対して集計されたスチューデント t 検定の両側有意水準を返します。 (grp, value [, eq_var])

ttest_sterr
TTest_sterr() は、2 つの独立した一連の値値に対して集計されたスチューデントt 検定の平均の差の標準誤差を返します。

TTest_sterr() は、2 つの独立した一連の値値に対して集計されたスチューデント t 検定の平均の差の標準誤差を返します。 (grp, value [, eq_var])

ttest_t
TTest_t() は、2 つの独立した一連の値に対して集計されたt 値を返します。

TTest_t() は、2 つの独立した一連の値に対して集計された t 値を返します。 (grp, value [, eq_var])

ttest_upper
TTest_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。

TTest_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。 (grp, value [, sig [, eq_var]])

2 つの独立加重標本によるt 検定
次の関数は、入力データ系列が加重 2 段組で与えられる、2 つの独立標本のスチューデントt 検定に適用されます。

ttestw_conf
TTestw_conf() は、2 つの独立した一連の値に対して集計されたt 値を返します。

TTestw_conf() は、2 つの独立した一連の値に対して集計された t 値を返します。 (weight, grp, value [, sig[, eq_var]])

ttestw_df
TTestw_df() は、2 つの独立した一連の値に対して集計されたスチューデントt 検定の df 値 (自由度) を返します。

TTestw_df() は、2 つの独立した一連の値に対して集計されたスチューデント t 検定の df 値 (自由度) を返します。 (weight, grp, value [, eq_var])

ttestw_dif
TTestw_dif() は、2 つの独立した一連の値に対して集計されたスチューデントt 検定の平均の差を返します。

TTestw_dif() は、2 つの独立した一連の値に対して集計されたスチューデント t 検定の平均の差を返します。 (weight, grp, value)

ttestw_lower
TTestw_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。

TTestw_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。 (weight, grp, value [, sig[, eq_var]])

ttestw_sig
TTestw_sig() は、2 つの独立した一連の値に対して集計されたスチューデントt 検定の両側有意水準を返します。

TTestw_sig() は、2 つの独立した一連の値に対して集計されたスチューデント t 検定の両側有意水準を返します。 (weight, grp, value [, eq_var])

ttestw_sterr
TTestw_sterr() は、2 つの独立した一連の値値に対して集計されたスチューデントt 検定の平均の差の標準誤差を返します。

TTestw_sterr() は、2 つの独立した一連の値値に対して集計されたスチューデント t 検定の平均の差の標準誤差を返します。 (weight, grp, value [, eq_var])

ttestw_t
TTestw_t() は、2 つの独立した一連の値に対して集計されたt 値を返します。

TTestw_t() は、2 つの独立した一連の値に対して集計された t 値を返します。 (weight, grp, value [, eq_var])

ttestw_upper
TTestw_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。

TTestw_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。 (weight, grp, value [, sig [, eq_var]])

1 つの標本によるt 検定
次の関数は、1 標本のスチューデントt 検定に適用されます。

ttest1_conf
TTest1_conf() は、一連の値に対して集計された信頼区間値を返します。

TTest1_conf() は、一連の値に対して集計された信頼区間値を返します。 (value [, sig])

ttest1_df
TTest1_df() は、一連の値に対して集計されたスチューデントt 検定の df 値 (自由度) を返します。

TTest1_df() は、一連の値に対して集計されたスチューデント t 検定の df 値 (自由度) を返します。 (value)

ttest1_dif
TTest1_dif()は、一連の値に対して集計されたスチューデントt 検定の平均の差を返します。

TTest1_dif()は、一連の値に対して集計されたスチューデント t 検定の平均の差を返します。 (value)

ttest1_lower
TTest1_lower()は、一連の値に対して集計された信頼区間の下限値を返します。

TTest1_lower()は、一連の値に対して集計された信頼区間の下限値を返します。 (value [, sig])

ttest1_sig
TTest1_sig()は、一連の値に対して集計されたスチューデントt 検定の両側有意水準を返します。

TTest1_sig()は、一連の値に対して集計されたスチューデント t 検定の両側有意水準を返します。 (value)

ttest1_sterr
TTest1_sterr() は、一連の値に対して集計されたスチューデントt 検定の平均の差の標準誤差を返します。

TTest1_sterr() は、一連の値に対して集計されたスチューデント t 検定の平均の差の標準誤差を返します。 (value)

ttest1_t
TTest1_t() は、一連の値に対して集計されたt 値を返します。

TTest1_t() は、一連の値に対して集計された t 値を返します。 (value)

ttest1_upper
TTest1_upper() は、一連の値に対して集計された信頼区間の上限値を返します。

TTest1_upper() は、一連の値に対して集計された信頼区間の上限値を返します。 (value [, sig])

1 つの加重標本によるt 検定

次の関数は、入力データ系列が加重 2 段組で与えられる、1 標本のスチューデントt 検定に適用されます。

ttest1w_conf
TTest1w_conf() は、一連の値に対して集計された信頼区間値を返す **numeric** 関数です。

TTest1w_conf() は、一連の値に対して集計された信頼区間値を返す numeric 関数です。 (weight, value [, sig])

ttest1w_df
TTest1w_df() は、一連の値に対して集計されたスチューデントt 検定の df 値 (自由度) を返します。

TTest1w_df() は、一連の値に対して集計されたスチューデント t 検定の df 値 (自由度) を返します。 (weight, value)

ttest1w_dif

TTest1w_dif()は、一連の値に対して集計されたスチューデントt検定の平均の差を返します。

TTest1w_dif()は、一連の値に対して集計されたスチューデント t 検定の平均の差を返します。 (weight, value)

ttest1w_lower

TTest1w_lower()は、一連の値に対して集計された信頼区間の下限値を返します。

TTest1w_lower()は、一連の値に対して集計された信頼区間の下限値を返します。 (weight, value [, sig])

ttest1w_sig

TTest1w_sig()は、一連の値に対して集計されたスチューデントt検定の両側有意水準を返します。

TTest1w_sig()は、一連の値に対して集計されたスチューデント t 検定の両側有意水準を返します。 (weight, value)

ttest1w_sterr

TTest1w_sterr() は、一連の値に対して集計されたスチューデントt検定の平均の差の標準誤差を返します。

TTest1w_sterr() は、一連の値に対して集計されたスチューデント t 検定の平均の差の標準誤差を返します。 (weight, value)

ttest1w_t

TTest1w_t() は、一連の値に対して集計されたt値を返します。

TTest1w_t() は、一連の値に対して集計された t 値を返します。 (weight, value)

ttest1w_upper

TTest1w_upper() は、一連の値に対して集計された信頼区間の上限値を返します。

TTest1w_upper() は、一連の値に対して集計された信頼区間の上限値を返します。 (weight, value [, sig])

TTest_conf

TTest_conf は、2 つの独立した一連の値に対して集計された t検定信頼区間値を返します。

この関数は、独立したサンプル スチューデントのt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_conf( Group, Value )
TTest_conf( Group, Value, Sig, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest_df

TTest_df() は、2 つの独立した一連の値に対して集計されたスチューデントt 検定値 (自由度) を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_df( Group, Value )
TTest_df( Group, Value, false )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest_dif

TTest_dif() は数値関数で、2 つの独立した一連の値に対して集計されたスチューデントt 検定の平均の差を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_dif( Group, Value )
TTest_dif( Group, Value, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest_lower

TTest_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_lower( Group, Value )
TTest_lower( Group, Value, Sig, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest_sig

TTest_sig() は、2 つの独立した一連の値に対して集計されたスチューデント t 検定の両側有意水準を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_sig( Group, Value )
TTest_sig( Group, Value, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest_sterr

TTest_sterr() は、2 つの独立した一連の値値に対して集計されたスチューデントt 検定の平均の差の標準誤差を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_sterr( Group, Value )
TTest_sterr( Group, Value, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest_t

TTest_t() は、2 つの独立した一連の値に対して集計された t 値を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_t( Group, Value, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest_upper

TTest_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest_upper( Group, Value )
TTest_upper( Group, Value, sig, false )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTestw_conf

TTestw_conf() は、2 つの独立した一連の値に対して集計された t 値を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、2 つの独立したサンプル スチューデント t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って 1 回または複数回カウントされます。
grp	2 つのサンプルグループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_conf( Weight, Group, Value )
TTestw_conf( Weight, Group, Value, sig, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTestw_df

TTestw_df() は、2 つの独立した一連の値に対して集計されたスチューデントt 検定の df 値 (自由度) を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、2 つの独立したサンプル スチューデントt 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。
grp	2つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
value	評価対象の標本値です。サンプル値は、**group** で指定された2つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_df( Weight, Group, Value )
TTestw_df( Weight, Group, Value, false )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTestw_dif

TTestw_dif() は、2つの独立した一連の値に対して集計されたスチューデントt検定の平均の差を返します。

この関数は、加重2段組に入力データ系列が与えられている状態にある、2つの独立したサンプル スチューデントt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

```
TTestw_dif (weight, grp, value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロード スクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロード スクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_dif( Weight, Group, Value )
TTestw_dif( Weight, Group, Value, false )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTestw_lower

TTestw_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、2 つの独立したサンプル スチューデントt 検定に適用されます。

関数がデータロード スクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。
grp	2つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
value	評価対象の標本値です。サンプル値は、**group** で指定された2つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_lower( Weight, Group, Value )
TTestw_lower( Weight, Group, Value, sig, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTestw_sig

TTestw_sig() は、2つの独立した一連の値に対して集計されたスチューデントt検定の両側有意水準を返します。

この関数は、加重2段組に入力データ系列が与えられている状態にある、2つの独立したサンプル スチューデントt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロード スクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロード スクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_sig( Weight, Group, Value )
TTestw_sig( Weight, Group, Value, false )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTestw_sterr

TTestw_sterr() は、2 つの独立した一連の値値に対して集計されたスチューデントt検定の平均の差の標準誤差を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、2 つの独立したサンプル スチューデントt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロード スクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロード スクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_sterr( Weight, Group, Value )
TTestw_sterr( Weight, Group, Value, false )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTestw_t

TTestw_t() は、2 つの独立した一連の値に対して集計された t 値を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、2 つの独立したサンプル スチューデント t 検定に適用されます。

関数がデータ ロード スクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って 1 回または複数回カウントされます。
grp	2 つのサンプルグループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_t( Weight, Group, Value )
TTestw_t( Weight, Group, Value, false )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTestw_upper

TTestw_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、2 つの独立したサンプル スチューデントt 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
weight	**value** の各値は、**weight** に対応する加重値に従って 1 回または複数回カウントされます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロード スクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロード スクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTestw_upper( Weight, Group, Value )
TTestw_upper( Weight, Group, Value, sig, false )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest1_conf

TTest1_conf() は、一連の値に対して集計された信頼区間値を返します。

この関数は、1 サンプル スチューデント t 検定に適用されます。

関数がデータ ロード スクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_conf( value )
TTest1_conf( value, 0.005 )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1_df

TTest1_df() は、一連の値に対して集計されたスチューデントt検定の df 値 (自由度) を返します。

この関数は、1 サンプル スチューデントt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

```
TTest1_df (value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_df( Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1_dif

TTest1_dif()は、一連の値に対して集計されたスチューデントt検定の平均の差を返します。

この関数は、1 サンプル スチューデントt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

```
TTest1_dif (value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_dif( Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1_lower

TTest1_lower()は、一連の値に対して集計された信頼区間の下限値を返します。

この関数は、1 サンプル スチューデントt 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_lower( Value )
TTest1_lower( Value, 0.005 )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1_sig

TTest1_sig()は、一連の値に対して集計されたスチューデントt 検定の両側有意水準を返します。

この関数は、1 サンプル スチューデントt 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

```
TTest1_sig (value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_sig( Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1_sterr

TTest1_sterr() は、一連の値に対して集計されたスチューデントt検定の平均の差の標準誤差を返します。

この関数は、1 サンプル スチューデントt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

```
TTest1_sterr (value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_sterr( value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1_t

TTest1_t() は、一連の値に対して集計された t 値を返します。

この関数は、1 サンプル スチューデント t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

```
TTest1_t (value)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_t( Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1_upper

TTest1_upper() は、一連の値に対して集計された信頼区間の上限値を返します。

この関数は、1 サンプル スチューデント t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1_upper( Value )
TTest1_upper( Value, 0.005 )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest1w_conf

TTest1w_conf() は、一連の値に対して集計された信頼区間値を返す **numeric** 関数です。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、1 サンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_conf( Weight, Value )
TTest1w_conf( Weight, Value, 0.005 )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1w_df

TTest1w_df() は、一連の値に対して集計されたスチューデントt検定のdf値 (自由度) を返します。

この関数は、加重2段組に入力データ系列が与えられている状態にある、1サンプル スチューデントのt検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_df( Weight, Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1w_dif

TTest1w_dif()は、一連の値に対して集計されたスチューデント t 検定の平均の差を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、1 サンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_dif( Weight, Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1w_lower

TTest1w_lower()は、一連の値に対して集計された信頼区間の下限値を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、1 サンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

```
TTest1w_lower (weight, value [, sig ])
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って 1 回または複数回カウントされます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_lower( Weight, Value )
TTest1w_lower( Weight, Value, 0.005 )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1w_sig

TTest1w_sig()は、一連の値に対して集計されたスチューデントt検定の両側有意水準を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、1 サンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って 1 回または複数回カウントされます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_sig( Weight, Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1w_sterr

TTest1w_sterr() は、一連の値に対して集計されたスチューデントt検定の平均の差の標準誤差を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、1 サンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って 1 回または複数回カウントされます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_sterr( Weight, Value )
```

参照先:

標準的な t-test レポートの作成 (page 520)

TTest1w_t

TTest1w_t() は、一連の値に対して集計された t 値を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、1 サンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_t( Weight, Value )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

TTest1w_upper

TTest1w_upper() は、一連の値に対して集計された信頼区間の上限値を返します。

この関数は、加重 2 段組に入力データ系列が与えられている状態にある、1 サンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価されるサンプルです。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。

引数	説明
weight	**value** の各値は、**weight** に対応する加重値に従って1回または複数回カウントされます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
TTest1w_upper( Weight, Value )
TTest1w_upper( Weight, Value, 0.005 )
```

参照先:

- *標準的な t-test レポートの作成 (page 520)*

z 検定関数

2 つの母平均の統計学的検討を行います。2 サンプル z 検定は、2 つの標本が異なるものかどうかを調べます。これは、一般に、2 つの正規分布の分散が既知であり、かつ実験で大きな標本サイズが使用される場合に使用されます。

z 検定統計関数は、関数に適用される入力データ系列のタイプに基づいてグループ化されています。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

z-test 関数の使用例 (page 524)

1 段組形式の関数

次の関数は、シンプルな入力データ系列を含む z 検定に適用されます。

ztest_conf
ZTest_conf() は、一連の値に対して集計された z 値を返します。

ZTest_conf() は、一連の値に対して集計された z 値を返します。 (value [, sigma [, sig])

ztest_dif
ZTest_dif() は、一連の値に対して集計された z 検定の平均の差を返します。

ZTest_dif() は、一連の値に対して集計された z 検定の平均の差を返します。 (value [, sigma])

ztest_sig
ZTest_sig() は、一連の値に対して集計されたz検定の両側有意水準を返します。

ZTest_sig() は、一連の値に対して集計された z 検定の両側有意水準を返します。 (value [, sigma])

ztest_sterr
ZTest_sterr() は、一連の値に対して集計されたz検定の平均の差の標準誤差を返します。

ZTest_sterr() は、一連の値に対して集計された z 検定の平均の差の標準誤差を返します。 (value [, sigma])

ztest_z
ZTest_z() は、一連の値に対して集計されたz値を返します。

ZTest_z() は、一連の値に対して集計された z 値を返します。 (value [, sigma])

ztest_lower
ZTest_lower() は、2つの独立した一連の値に対して集計された信頼区間の下限値を返します。

ZTest_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。 (grp, value [, sig [, eq_var]])

ztest_upper
ZTest_upper() は、2つの独立した一連の値に対して集計された信頼区間の上限値を返します。

ZTest_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。 (grp, value [, sig [, eq_var]])

加重 2 段組形式の関数

次の関数は、入力データ系列が加重2段組で与えられるz検定に適用されます。

ztestw_conf
ZTestw_conf() は、一連の値に対して集計されたz信頼区間値を返します。

ZTestw_conf() は、一連の値に対して集計された z 信頼区間値を返します。 (weight, value [, sigma [, sig]])

ztestw_dif
ZTestw_dif() は、一連の値に対して集計されたz検定の平均の差を返します。

ZTestw_dif() は、一連の値に対して集計された z 検定の平均の差を返します。 (weight, value [, sigma])

ztestw_lower
ZTestw_lower() は、2つの独立した一連の値に対して集計された信頼区間の下限値を返します。

ZTestw_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。 (weight, value [, sigma])

ztestw_sig
ZTestw_sig() は、一連の値に対して集計されたz検定の両側有意水準を返します。

ZTestw_sig() は、一連の値に対して集計された z 検定の両側有意水準を返します。 (weight, value [, sigma])

ztestw_sterr

ZTestw_sterr() は、一連の値に対して集計されたz検定の平均の差の標準誤差を返します。

ZTestw_sterr() は、一連の値に対して集計された z 検定の平均の差の標準誤差を返します。 (weight, value [, sigma])

ztestw_upper

ZTestw_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。

ZTestw_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。 (weight, value [, sigma])

ztestw_z

ZTestw_z() は、一連の値に対して集計されたz値を返します。

ZTestw_z() は、一連の値に対して集計された z 値を返します。 (weight, value [, sigma])

ZTest_z

ZTest_z() は、一連の値に対して集計されたz値を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。母平均 0 と仮定されます。他の平均について検定する場合は、標本値からその平均値を減算します。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTest_z( Value-TestValue )
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTest_sig

ZTest_sig() は、一連の値に対して集計された z 検定の両側有意水準を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。母平均 0 と仮定されます。他の平均について検定する場合は、標本値からその平均値を減算します。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTest_sig(Value-TestValue)
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTest_dif

ZTest_dif() は、一連の値に対して集計された z 検定の平均の差を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。母平均 0 と仮定されます。他の平均について検定する場合は、標本値からその平均値を減算します。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTest_dif(Value-TestValue)
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTest_sterr

ZTest_sterr() は、一連の値に対して集計された z 検定の平均の差の標準誤差を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。母平均 0 と仮定されます。他の平均について検定する場合は、標本値からその平均値を減算します。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTest_sterr(Value-TestValue)
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTest_conf

ZTest_conf() は、一連の値に対して集計された z 値を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。母平均 0 と仮定されます。他の平均について検定する場合は、標本値からその平均値を減算します。

引数	説明
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTest_conf(Value-TestValue)
```

参照先:

z-test 関数の使用例 (page 524)

ZTest_lower

ZTest_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

引数	説明
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTest_lower( Group, Value )
ZTest_lower( Group, Value, sig, false )
```

参照先:

z-test 関数の使用例 (page 524)

ZTest_upper

ZTest_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロード スクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロード スクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロード スクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。

引数	説明
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTest_upper( Group, Value )
ZTest_upper( Group, Value, sig, false )
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTestw_z

ZTestw_z() は、一連の値に対して集計されたz値を返します。

この関数は、入力データ系列が加重 2 段組で与えられるz検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	値は、**value** によって返される必要があります。標本平均 0 と仮定されます。他の平均について検定する場合は、標本値からその値を減算します。
weight	**value** の各サンプル値は、**weight** に対応する重みに従って、1 回または複数回カウントされます。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTestw_z( Weight, Value-TestValue)
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTestw_sig

ZTestw_sig() は、一連の値に対して集計された z 検定の両側有意水準を返します。

この関数は、入力データ系列が加重 2 段組で与えられる z 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	値は、**value** によって返される必要があります。標本平均 0 と仮定されます。他の平均について検定する場合は、標本値からその値を減算します。
weight	**value** の各サンプル値は、**weight** に対応する重みに従って、1 回または複数回カウントされます。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTestw_sig( Weight, Value-TestValue)
```

参照先:

z-test 関数の使用例 (page 524)

ZTestw_dif

ZTestw_dif() は、一連の値に対して集計された z 検定の平均の差を返します。

この関数は、入力データ系列が加重 2 段組で与えられる z 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	値は、**value** によって返される必要があります。標本平均 0 と仮定されます。他の平均について検定する場合は、標本値からその値を減算します。
weight	**value** の各サンプル値は、**weight** に対応する重みに従って、1 回または複数回カウントされます。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTestw_dif( Weight, Value-TestValue)
```

参照先:

z-test 関数の使用例 (page 524)

ZTestw_sterr

ZTestw_sterr() は、一連の値に対して集計された z 検定の平均の差の標準誤差を返します。

この関数は、入力データ系列が加重 2 段組で与えられる z 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	値は、**value** によって返される必要があります。標本平均 0 と仮定されます。他の平均について検定する場合は、標本値からその値を減算します。
weight	**value** の各サンプル値は、**weight** に対応する重みに従って、1 回または複数回カウントされます。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTestw_sterr( Weight, Value-TestValue)
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTestw_conf

ZTestw_conf() は、一連の値に対して集計された z 信頼区間値を返します。

この関数は、入力データ系列が加重 2 段組で与えられる z 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。母平均 0 と仮定されます。他の平均について検定する場合は、標本値からその平均値を減算します。
weight	**value** の各サンプル値は、**weight** に対応する重みに従って、1 回または複数回カウントされます。
sigma	標準偏差がわかっている場合は、**sigma** に記述します。**sigma** の記述を省略すると、実際のサンプル標準偏差が使用されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTestw_conf( Weight, Value-TestValue)
```

参照先:

- *z-test 関数の使用例 (page 524)*

ZTestw_lower

ZTestw_lower() は、2 つの独立した一連の値に対して集計された信頼区間の下限値を返します。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTestw_lower( Group, Value )
ZTestw_lower( Group, Value, sig, false )
```

参照先:

z-test 関数の使用例 (page 524)

ZTestw_upper

ZTestw_upper() は、2 つの独立した一連の値に対して集計された信頼区間の上限値を返します。

この関数は、独立したサンプル スチューデントの t 検定に適用されます。

関数がデータロードスクリプトでが使用される場合、値は group by 句で定義されたレコードで反復処理されます。

関数がチャート式で使用される場合、値はチャート軸に対して反復処理されます。

構文:

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

戻り値データ型: 数値

引数:

引数

引数	説明
value	評価対象の標本値です。サンプル値は、**group** で指定された 2 つの値に従って、論理的にグループ化する必要があります。サンプル値の項目名がロードスクリプトに入力されていない場合、**Value** という名前が自動的に付与されます。
grp	2 つのサンプル グループの名前が含まれている項目です。グループの項目名がロードスクリプトに入力されていない場合、**Type** という名前が自動的に付与されます。
sig	両側有意水準は、**sig** で指定します。指定されない場合、**sig** は 0.025 に設定され、その結果として信頼区間は 95% になります。
eq_var	**eq_var** が False (0) に指定されている場合、2 つのサンプルは個別に分散していると解釈されます。**eq_var** が True (1) に指定されている場合、サンプルは均等に分散していると解釈されます。

制限事項:

数式にテキスト値、NULL 値、および欠損値が含まれていると、この関数は NULL を返します。

```
ZTestw_upper( Group, Value )
ZTestw_upper( Group, Value, sig, false )
```

参照先:

- *z-test 関数の使用例 (page 524)*

統計検定関数の例

このセクションでは、チャートとデータロードスクリプトで使用される統計検定関数の例を紹介します。

チャートでの chi2-test 関数の使用例

chi2-test 関数は、カイ二乗統計分析に関連した値の計算に使用します。

このセクションでは、サンプル データを用いて Qlik Sense で使用可能なカイ二乗分布検定関数の値を特定するためのビジュアライゼーションの作成方法を説明します。構文と引数については、各 chi2-test チャート関数のトピックを参照してください。

サンプル データのロード

3 つの異なる統計サンプルをスクリプトにロードする方法を説明するために、3 組のサンプル データを使用します。

次の手順を実行します。

1. 新しいアプリを作成します。
   データロードエディタで、以下を入力します。
   ```
   // Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the top of the script.
   ```
2.
   ```
   Sample_1:

   LOAD * inline [

   Grp,Grade,Count

   I,A,15

   I,B,7

   I,C,9

   I,D,20

   I,E,26

   I,F,19

   II,A,10

   II,B,11

   II,C,7

   II,D,15

   II,E,21

   II,F,16

   ];

   // 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. をクリックしてデータをロードします。

chi2-testチャート関数 ビジュアライゼーションの作成

サンプル 1

次の手順を実行します。

1. データ ロード エディターでをクリックしてアプリ ビューに進み、作成したシートをクリックします。
   シート ビューが表示されます。
2. [**シートを編集**] をクリックしてシートを編集します。
3. [**チャート**] からテーブルを追加し、[**項目**] からGrp、Grade、Count を軸として追加します。
   このテーブルにはサンプル データが表示されています。
4. 以下の数式を軸として使用する別のテーブルを追加します。
   `ValueList('p','df','Chi2')`
   その際、3 つの chi2-test 関数の名前を持つ軸のラベルを作成するために合成軸関数を使用します。
   次の数式をメジャーとしてテーブルに追加します。
   `IF(ValueList('p','df','Chi2')='p',Chi2Test_p(Grp,Grade,Count),`
5.
   ```
   IF(ValueList('p','df','Chi2')='df',Chi2Test_df(Grp,Grade,Count),

   Chi2Test_Chi2(Grp,Grade,Count)))
   ```
   これにより、各 chi2-test 関数の結果値が関連する合成軸の横のテーブルに表示されます。
6. メジャーの [**数値形式**] を [**数値**] に設定し、[**3**] を [**有効桁数**] に入力します。

> *メジャーの数式では、次の数式を代用することも可能です。`Pick(Match(ValueList('p','df','Chi2'),'p','df','Chi2'),Chi2Test_p(Grp,Grade,Count),Chi2Test_df(Grp,Grade,Count),Chi2Test_Chi2(Grp,Grade,Count))`*

結果:

サンプル 1 の chi2-test 関数の結果テーブルには次の値が含まれます。

結果テーブル

p	df	Chi2
0.820	5	2.21

サンプル 2

次の手順を実行します。

1. サンプル 1 の例で編集したシートに [**チャート**] からテーブルを追加し、[**項目**] からSex、Opinion、OpCount を軸として追加します。
2. サンプル 1 の結果テーブルを [**コピー**] と [**貼り付け**] コマンドでコピーします。メジャー内の数式を編集し、3 つの chi2-test 関数すべての引数をサンプル 2 データで使用されている名前 (例えば、`Chi2Test_p(Sex,Opinion,OpCount)`) に置き換えます。

結果:

サンプル 2 の chi2-test 関数の結果テーブルには次の値が含まれます。

結果テーブル

p	df	Chi2
0.000309	2	16.2

サンプル 3

次の手順を実行します。

1. サンプル 1 および 2 と同じ方法でテーブルをさらに 2 つ作成します。軸テーブルで、以下の項目を軸として使用します。Gender、Description、Actual、および Expected です。
2. 結果テーブルで、サンプル 3 データで使用した項目名を使用します。例: `Chi2Test_p (Gender,Description,Actual,Expected)`

結果:

サンプル 3 の chi2-test 関数の結果テーブルには次の値が含まれます。

結果テーブル

p	df	Chi2
0.000308	2	16.2

データロードスクリプトでの chi2-test 関数の使用例

chi2-test 関数は、カイ二乗統計分析に関連した値の計算に使用します。このセクションでは、Qlik Sense で使用可能なカイ二乗分布検定関数のデータロードスクリプトでの使用方法を説明します。構文と引数については、各 chi2-test スクリプト関数のトピックを参照してください。

この例では、2 グループ (I と II) の学生の成績 (A から F) 別の人数を含むテーブルを使用します。

Data table

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

サンプル データのロード

次の手順を実行します。

1. 新しいアプリを作成します。

   データロードエディタで、以下を入力します。

   ```
   // Sample_1 data is pre-aggregated... Note: make sure you set your DecimalSep='.' at the top of the script.
   ```
2.

   ```
   Sample_1:
   ```

```
LOAD * inline [

Grp,Grade,Count

I,A,15

I,B,7

I,C,9

I,D,20

I,E,26

I,F,19

II,A,10

II,B,11

II,C,7

II,D,15

II,E,21

II,F,16

];
```

3. をクリックして データをロードします。

サンプル データがロードされ ます。

chi2-test 関数の値のロード

chi2-test の値を Grp でグループ化 された新 しいテーブルのサンプル データに基づいて、ロードします。

次の手順を実行します。

1. データ ロード エディタで、スクリプトの最後に以下の記述を追加します。

```
// 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. をクリックして データをロードします。

chi2-test の値が、Chi2_table という名前のテーブルにロードされます。

結果

chi2-test の結果の値を、**[プレビュー]** のデータモデル ビューアで表示できます。次のように表示されます。

Results

Grp	chi2	df	p
I	16.00	5	0.007
II	9.40	5	0.094

標準的な t-test レポートの作成

標準的なスチューデントt-test レポートには、**Group Statistics** と**Independent Samples Test** の結果を含むテーブルを掲載できます。

以下のセクションでは、Qlik Senset-test 関数を 2 つの独立したサンプル グループ、Observation と Comparison に適用してテーブルを構築します。これらのサンプルに対応するテーブルは以下のようになります。

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

Type	conf	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval (Lower)	95% Confidence Interval (Upper)
Equal Variance not Assumed	0	3.534	37.116717335823	0.001	15.2	4.30101	6.48625	23.9137
Equal Variance Assumed	8.706939	3.534	38	0.001	15.2	4.30101	6.49306	23.9069

サンプル データのロード

次の手順を実行します。

1. 新しいシートでアプリを新規作成します。
2. データ ロード エディタに以下を入力します。

```
Table1:
Crosstable (Type, Value)
Load recno() as ID, * inline [
Observation|Comparison
35|2
40|27
12|38
15|31
21|1
14|19
46|1
10|34
28|3
48|1
16|2
30|3
32|2
48|1
31|2
22|1
12|3
39|29
19|37
25|2 ] (delimiter is '|');
```

   このロード スクリプトでは、**crosstable** に 3 つの引数が必要となるため、**recno()** が含まれています。よって、**recno()** により追加の引数 (この場合は各行の ID) が返されます。これが存在しない場合、**Comparison** サンプル値はロードされません。
3. をクリックして データをロードします。

Group statistics テーブルの作成

次の手順を実行します。

1. データ ロード エディターで をクリックしてアプリ ビューに進み、作成したシートをクリックします。
   これでシート ビューが表示されます。
2. [**シートを編集**] をクリックしてシートを編集します。
3. [**チャート**] からテーブルを追加し、[**項目**] から Type を軸としてテーブルに追加します。

4. 次の数式をメジャーとして追加します。

数式の例

ラベル	式
N	Count(Value)
Mean	Avg(Value)
Standard Deviation	Stdev(Value)
Standard Error Mean	Sterr(Value)

5. [ソート] をクリックして Type をソート リストの先頭にくるようにします。

結果:

これらのサンプルの Group statistics テーブルは以下のようになります。

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 テーブルの作成

次の手順を実行します。

1. [シートを編集] をクリックしてシートを編集します。
2. [チャート] から、軸として、次の式を持つテーブルをテーブル `=ValueList (Dual('Equal Variance not Assumed', 0), Dual('Equal Variance Assumed', 1))` に追加して、ラベル「タイプ」を付けます。

3. 次の数式をメジャーとして追加します:

数式の例

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

結果:

Independent Sample Test

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

Type	conf	t	df	Sig. (2-tailed)	Mean Difference	Standard Error Difference	95% Confidence Interval (Lower)	95% Confidence Interval (Upper)
Equal Variance Assumed	8.706939	3.534	38	0.001	15.2	4.30101	6.49306	23.9069

z-test 関数の使用例

z-test 関数は、分散が既知であり、通常 30 を超える大型データサンプルの z-test 統計分析に関連した値を特定する際に使用します。

このセクションでは、サンプル データを使って、Qlik Sense で使用可能な z-test 関数の値を特定するためのビジュアライゼーションの作成方法を説明します。構文と引数については、各 z-test チャート関数のトピックを参照してください。

サンプル データのロード

ここで使用するサンプル データは、t-test 関数の例で使用したものと同じです。このサンプル データは通常 z 検定分析には小さすぎますが、ここでは Qlik Sense で異なる z-test 関数の使用を説明する目的で使用します。

次の手順を実行します。

1. 新しいシートでアプリを新規作成します。

   t-test 関数用のアプリを作成したことがある場合、そのアプリを使ってこれらの関数の新しいシートを作成することもできます。

2. データ ロード エディタで、以下を入力します。

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

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

このロード スクリプトでは、**crosstable** に 3 つの引数が必要となるため、**recno()** が含まれています。よって、**recno()** により追加の引数 (この場合は各行の ID) が返されます。これが存在しない場合、**Comparison** サンプル値はロードされません。

3. をクリックしてデータをロードします。

z-test テーブルの作成

次の手順を実行します。

1. データ ロード エディタで をクリックしてアプリ ビューに進み、上記で作成したシートをクリックします。シート ビューが表示されます。
2. [**シートを編集**] をクリックしてシートを編集します。
3. [**チャート**] からテーブルを追加し、[**項目**] から Type を軸として追加します。
4. 次の数式をメジャーとしてテーブルに追加します。

数式の例

ラベル	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)

*意味のある値を表示するために、必要に応じてメジャーの数値形式を調整します。大部分のメジャーの数値形式を、**[自動]** ではなく **[数値] > [シンプル]** に設定すると、テーブルの読み取りが容易になります。ただし、ZTest Sig の場合は、例えば数値書式: **[カスタム]** を使用して、書式パターンを **#.######** に調整します。*

結果:

サンプル データの z-test 関数の結果テーブルには次の値が含まれます。

z-test 結果テーブル

Type	ZTest Conf	ZTest Dif	ZTest Sig	ZTest Sterr	ZTest Z
Comparison	6.40	11.95	0.000123	3.27	3.66
Observation	5.48	27.15	0.000000	2.80	9.71

z-testw テーブルの作成

z-testw 関数は、入力データ系列が加重 2 段組で発生した場合に使用します。この数式には、weight 引数の値が必要です。

ここで示す例では全体的に値 2 が使用されていますが、各観測値の weight を値として定義する数式を使用することもできます。

次の手順を実行します。

1. データロードエディタで をクリックしてアプリビューに進み、上記で作成したシートをクリックします。シートビューが表示されます。
2. [**シートを編集**] をクリックしてシートを編集します。
3. [**チャート**] からテーブルを追加し、[**項目**] から Type を軸として追加します。
4. 次の数式をメジャーとしてテーブルに追加します。

数式の例

ラベル	式
ZTestw Conf	ZTestw_conf(2,Value)
ZTestw Dif	ZTestw_dif(2,Value)
ZTestw Sig	ZTestw_sig(2,Value)
ZTestw Sterr	ZTestw_sterr(2,Value)
ZTestw Z	ZTestw_z(2,Value)

z-test 関数例と同じ数字形式を使用します。

結果:

z-testw 関数の結果テーブルには次の値が含まれます。

z-testw 結果テーブル

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

文字列集計関数

このセクションでは、文字列関連の集計関数について説明します。

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

データロードスクリプトの文字列集計関数

Concat

Concat() は、文字列値を組み合わせるために使用します。このスクリプト関数は、**group by** 句で定義されたレコードで反復処理される数式の値をすべて集計した文字列連結を返します。

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

FirstValue

FirstValue() は、数式で定義され、**group by** 句でソートされたレコードから最初にロードされた値を返します。

この関数は、スクリプト関数としてのみ使用できます。

```
FirstValue  (expression)
```

LastValue

LastValue() は、数式で定義され、**group by** 句でソートされたレコードから最後にロードされた値を返します。

この関数は、スクリプト関数としてのみ使用できます。

```
LastValue  (expression)
```

MaxString

MaxString() は、数式に含まれる文字列の値を検索し、**group by** 句で定義されたとおり、複数のレコードについて、アルファベット順で最後のテキスト値を返します。

```
MaxString (expression )
```

MinString

MinString() は、数式に含まれる文字列の値を検索し、**group by** 句で定義されたとおり、複数のレコードについて、アルファベット順で最初のテキスト値を返します。

```
MinString (expression )
```

チャートの文字列集計関数

次のチャート関数は、チャートの文字列を集計する際に使用できます

Concat

Concat() は、文字列値を組み合わせるために使用します。この関数では、それぞれの軸に対して評価された数式に含まれるあらゆる値の文字列連結が返されます。

```
Concat - チャート関数({[SetExpression] [DISTINCT] [TOTAL [<fld{, fld}>]] string
[, delimiter[, sort_weight]])
```

MaxString
MaxString() は、数式または項目に含まれる文字列の値を検索し、アルファベット順で最後のテキスト値を返します。

```
MaxString - チャート関数({[SetExpression] [TOTAL [<fld{, fld}>]]} expr)
```

MinString
MinString() は、数式または項目に含まれる文字列の値を検索し、アルファベット順で最初のテキスト値を返します。

```
MinString - チャート関数({[SetExpression] [TOTAL [<fld {, fld}>]]} expr)
```

Concat

Concat() は、文字列値を組み合わせるために使用します。このスクリプト関数は、**group by** 句で定義されたレコードで反復処理される数式の値をすべて集計した文字列連結を返します。

構文:

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

戻り値データ型: string

引数:

処理される文字列が含まれている数式および項目。

引数

引数	説明
string	処理される文字列が含まれている数式および項目。
delimiter	各値は、delimiter の文字列によって区切られます。
sort-weight	連結の順序は、軸 **sort-weight** の値によって決定されます。最小値に対応する文字列が連結の最初に表示されます。.
distinct	数式の前に **distinct** がある場合、重複はすべて無視されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

例と結果

例	結果	シートに追加された結果
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,Concat(Team) as TeamConcat1 Resident TeamData Group By SalesGroup;	SalesGroup East West	TeamConcat1 AlphaBetaDeltaGammaGamma EpsilonEtaThetaZeta
TeamData テーブルが前の例のようにロードされた場合: LOAD SalesGroup,Concat(distinct Team,'-') as TeamConcat2 Resident TeamData Group By SalesGroup;	SalesGroup East West	TeamConcat2 Alpha-Beta-Delta-Gamma Epsilon-Eta-Theta-Zeta
TeamData テーブルが前の例のようにロードされた場合。**sort-weight** 引数が追加されているので、結果は Amount 軸の値によって並べ替えられます。 LOAD SalesGroup,Concat(distinct Team,'-',Amount) as TeamConcat2 Resident TeamData Group By SalesGroup;	SalesGroup East West	TeamConcat2 Delta-Beta-Gamma-Alpha Eta-Epsilon-Zeta-Theta

Concat - チャート関数

Concat() は、文字列値を組み合わせるために使用します。この関数では、それぞれの軸に対して評価された数式に含まれるあらゆる値の文字列連結が返されます。

構文:

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

戻り値データ型: string

引数:

引数

引数	説明
string	処理される文字列が含まれている数式および項目。
delimiter	各値は、delimiter の文字列によって区切られます。
sort-weight	連結の順序は、軸 **sort-weight** の値によって決定されます。最小値に対応する文字列が連結の最初に表示されます。.
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	関数の引数の前に **DISTINCT** という用語が付いている場合、関数の引数の評価から生じる重複は無視されます。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

Results table

SalesGroup	Amount	Concat(Team)	Concat(TOTAL <SalesGroup> Team)
East	25000	Alpha	AlphaBetaDeltaGammaGamma
East	20000	BetaGammaGamma	AlphaBetaDeltaGammaGamma
East	14000	Delta	AlphaBetaDeltaGammaGamma
West	17000	Epsilon	EpsilonEtaThetaZeta
West	14000	Eta	EpsilonEtaThetaZeta
West	23000	Theta	EpsilonEtaThetaZeta
West	19000	Zeta	EpsilonEtaThetaZeta

関数の例

例	結果
Concat(Team)	テーブルは、軸 SalesGroup とAmount、およびメジャー Concat(Team) のバリエーションで構成されています。テーブル結果を無視すると、SalesGroup の 2 つの値にまたがる Team の 8 つの値のデータがある場合でも、テーブルでは複数の Team 文字列値を連結するメジャー Concat(Team) の結果のみが、軸 Amount 20000 を含む行になる点に注意が必要です。これにより、BetaGammaGamma が得られます。これは、入力データで Amount 20000 の値が 3 つあるためです。SalesGroup とAmount の各組み合わせの Team の値は 1 つしかないため、メジャーが軸全体にある場合は、その他すべての結果は連結されません。
Concat (DISTINCT Team,', ')	Beta, Gamma (DISTINCT 修飾子を使用すると、重複した Gamma の結果が無視されるため。また、区切り記号の引数がコンマとスペースで定義されているため)。
Concat (TOTAL <SalesGroup> Team)	TOTAL 修飾子が使用されている場合、Team のあらゆる値の文字列値がすべて連結されます。<SalesGroup> の項目選択が指定されている場合は、SalesGroup 軸の 2 つの値に結果が分割されます。SalesGroupEast の結果は AlphaBetaDeltaGammaGamma になります。SalesGroupWest の結果は EpsilonEtaThetaZeta になります。
Concat (TOTAL <SalesGroup> Team,';', Amount)	**sort-weight**:Amount の引数を追加すると、結果は Amount 軸の値によって順位が付けられます。結果は、DeltaBetaGammaGammaAlpha および EtaEpsilonZEtaTheta になります。

例で使用されているデータ:

```
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() は、数式で定義され、**group by** 句でソートされたレコードから最初にロードされた値を返します。

この関数は、スクリプト関数としてのみ使用できます。

構文:

```
FirstValue ( expr)
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。

制限事項:

テキスト値が見つからない場合は、NULL が返されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果	シート上の結果
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); FirstValue1: LOAD SalesGroup,FirstValue(Team) as FirstTeamLoaded Resident TeamData Group By SalesGroup;	SalesGroup East West	FirstTeamLoaded Gamma Zeta

LastValue

LastValue() は、数式で定義され、**group by** 句でソートされたレコードから最後にロードされた値を返します。

この関数は、スクリプト関数としてのみ使用できます。

構文:

```
LastValue ( expr )
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。

制限事項:

テキスト値が見つからない場合は、NULL が返されます。

例と結果:

アプリにスクリプト例を追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

下図の結果列と同じ外観にするには、プロパティパネルで [ソート] セクションの設定を [自動] から [カスタム] に変更し、数値とアルファベットのソート順の選択を解除します。

例	結果	カスタム ソートの結果
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); LastValue1: LOAD SalesGroup,LastValue(Team) as LastTeamLoaded Resident TeamData Group By SalesGroup;	SalesGroup East West	LastTeamLoaded Beta Theta

MaxString

MaxString() は、数式に含まれる文字列の値を検索し、**group by** 句で定義されたとおり、複数のレコードについて、アルファベット順で最後のテキスト値を返します。

構文:

```
MaxString ( expr )
```

戻り値データ型: dual

引数:

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。

制限事項:

テキスト値が見つからない場合は、NULL が返されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

例	結果	
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); Concat1: LOAD SalesGroup,MaxString(Team) as MaxString1 Resident TeamData Group By SalesGroup;	SalesGroup East West	MaxString1 Gamma Zeta
TeamData テーブルが前の例のようにロードされ、データロードスクリプトに次の SET ステートメントが含まれています。 SET DateFormat='DD/MM/YYYY';': LOAD SalesGroup,MaxString(Date) as MaxString2 Resident TeamData Group By SalesGroup;	SalesGroup East West	MaxString2 01/11/2013 01/12/2013

MaxString - チャート関数

MaxString() は、数式または項目に含まれる文字列の値を検索し、アルファベット順で最後のテキスト値を返します。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set分析数式でレコードセットを定義することも可能です。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

制限事項:

数式に文字列表現の値が含まれていない場合は、NULL が返されます。

例と結果:

結果テーブル

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

関数の例

例	結果
MaxString (Team)	軸 Amount には 20000 の値が 3 つあります。2 つは Gamma (異なる日付)、1 つは Beta です。このため、メジャー MaxString (Team) の結果はソートされた文字列の最大値である Gamma になります。
MaxString (Date)	2013/11/01 は、軸 Amount に関連付けられている 3 つの Date 値の最大値です。ここでは、スクリプトに SET ステートメント SET DateFormat='YYYY-MM-DD';' が含まれているものとします。

例で使用されているデータ:

```
TeamData:
LOAD * inline [
SalesGroup|Team|Date|Amount
East|Gamma|01/05/2013|20000
East|Gamma|02/05/2013|20000
West|Zeta|01/06/2013|19000
East|Alpha|01/07/2013|25000
East|Delta|01/08/2013|14000
West|Epsilon|01/09/2013|17000
West|Eta|01/10/2013|14000
East|Beta|01/11/2013|20000
West|Theta|01/12/2013|23000
] (delimiter is '|');
```

MinString

MinString() は、数式に含まれる文字列の値を検索し、**group by** 句で定義されたとおり、複数のレコードについて、アルファベット順で最初のテキスト値を返します。

構文:

```
MinString ( expr )
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。

制限事項:

テキスト値が見つからない場合は、NULL が返されます。

例と結果:

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

結果のデータ

例	結果	
TeamData: LOAD * inline [SalesGroup|Team|Date|Amount East|Gamma|01/05/2013|20000 East|Gamma|02/05/2013|20000 West|Zeta|01/06/2013|19000 East|Alpha|01/07/2013|25000 East|Delta|01/08/2013|14000 West|Epsilon|01/09/2013|17000 West|Eta|01/10/2013|14000 East|Beta|01/11/2013|20000 West|Theta|01/12/2013|23000] (delimiter is '|'); Concat1: LOAD SalesGroup,MinString(Team) as MinString1 Resident TeamData Group By SalesGroup;	SalesGroup East West	MinString1 Alpha Epsilon
TeamData テーブルが前の例のようにロードされ、データロードスクリプトに次の SET ステートメントが含まれています。 SET DateFormat='DD/MM/YYYY';': LOAD SalesGroup,MinString(Date) as MinString2 Resident TeamData Group By SalesGroup;	SalesGroup East West	MinString2 01/05/2013 01/06/2013

MinString - チャート関数

MinString() は、数式または項目に含まれる文字列の値を検索し、アルファベット順で最初のテキスト値を返します。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
TOTAL	関数の引数の前に **TOTAL** の文字が配置されている場合、現在の軸の値に関連しているものだけでなく、現在の選択範囲内にあるすべての可能な値に対して計算が実行されます。つまりチャート軸は無視されます。 **TOTAL [<fld {,fld}>]** (ここで、**TOTAL** 修飾子の後には、1 つまたは複数の項目名のリストがチャート軸変数のサブセットとして続く) を使用して、合計絞込値のサブセットを作成できます。

例と結果:

サンプル データ

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

関数の例

例	結果
`MinString (Team)`	軸 Amount には 20000 の値が 3 つあります。2 つは Gamma (異なる日付)、1 つは Beta です。このため、メジャー MinString (Team) の結果はソートされた文字列の開始値である Beta になります。
`MinString (Date)`	2013/11/01 は、軸 Amount に関連付けられている 3 つの値で最も早い Date の値です。ここでは、スクリプトに SET ステートメント `SET DateFormat='YYYY-MM-DD';` が含まれているものとします。

例で使用されているデータ:

```
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 '|');
```

合成軸関数

合成軸は、データモデルの項目から直接作成するのではなく、合成軸関数で生成した値を基にアプリで作成します。合成軸関数で生成された値をチャートで計算済みの軸として使用すると、合成軸が作成されます。合成軸を使用すると、データから取得した値のある軸 (つまり動的軸) でチャートを作成できます。

合成軸は選択に影響されません。

チャートで使用可能な合成集計関数は、次のとおりです。

ValueList

ValueList() は、計算軸で使用される場合、合成軸を形成するリストされた値のセットを返します。

ValueList - チャート関数 (v1 {, Expression}**)**

ValueLoop

ValueLoop() は、計算軸で使用される場合、合成軸を形成する反復処理された値のセットを返します。

ValueLoop - チャート関数(from [, to [, step]]**)**

ValueList - チャート関数

ValueList() は、計算軸で使用される場合、合成軸を形成するリストされた値のセットを返します。

***ValueList** 関数を用いて作成された合成軸を持つチャートでは、チャートの数式に同じパラメータを持つ **ValueList** 関数を再記述することで、特定の数式のセルに対応する軸の値を参照できます。もちろん、この関数はレイアウト内ならどこででも使用できますが、合成軸に対して使用する場合を除き、この関数は集計関数内でのみ有効になります。*

合成軸は選択に影響されません。

構文:

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

戻り値データ型: dual

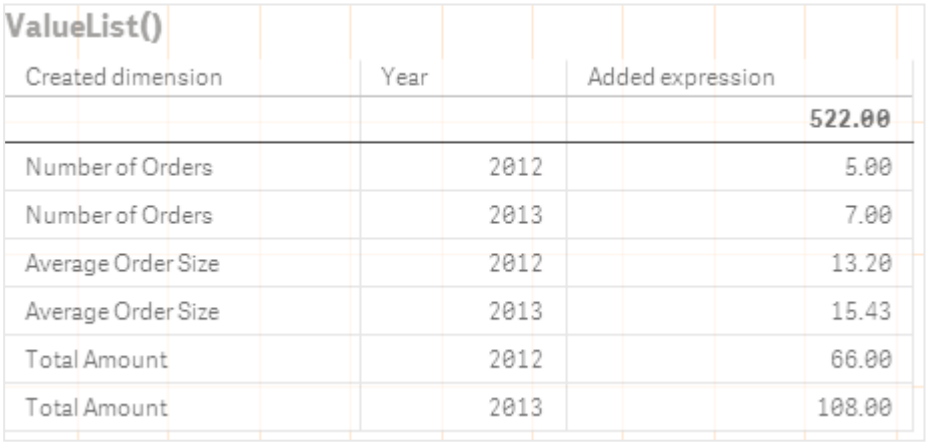
引数:

引数

引数	説明
v1	静的な値 (通常は文字列、ただし数値も可)。
{,...}	オプションの静的値リスト

例と結果:

関数の例

例	結果
`ValueList ('Number of Orders', 'Average Order Size', 'Total Amount')`	例えば、テーブルで軸の作成に使用すると、テーブルの行ラベルとして 3 つの文字列値が生じます。これらの値は数式で参照できます。
`=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) ))`	この数式では、作成された軸から値を取得し、ネストされた IF ステートメントで 3 つの集計関数の入力として参照します。 

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

例で使用されているデータ:

```
SalesPeople:
LOAD * INLINE [
SaleID|SalesPerson|Amount|Year
1|1|12|2013
2|1|23|2013
3|1|17|2013
4|2|9|2013
5|2|14|2013
6|2|29|2013
```

```
7|2|4|2013
8|1|15|2012
9|1|16|2012
10|2|11|2012
11|2|17|2012
12|2|7|2012
] (delimiter is '|');
```

ValueLoop - チャート関数

ValueLoop() は、計算軸で使用される場合、合成軸を形成する反復処理された値のセットを返します。生成される値は、**from** 値から始まり、**to** 値で終了します (step 増分の中間値を含む)。

> ***ValueLoop** 関数を用いて作成された合成軸を持つチャートでは、チャートの数式に同じパラメータを持つ **ValueLoop** 関数を再記述することで、特定の数式のセルに対応する軸の値を参照できます。もちろん、この関数はレイアウト内ならどこででも使用できますが、合成軸に対して使用する場合を除き、この関数は集計関数内でのみ有効になります。*

> *合成軸は選択に影響されません。*

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
from	生成する一連の値の開始値。
to	生成する一連の値の終了値。
step	値の増分量。

例と結果:

関数の例

例	結果
`ValueLoop (1, 10)`	これにより、数字付きラベルのような目的に使用できるテーブルの軸などが作成されます。この例では、値は 1 から10 になります。これらの値は数式で参照できます。
`ValueLoop (2, 10,2)`	この例では、引数 step が 2 になっているため、値は 2、4、6、8、10 になります。

ネストされた集計関数

別の集計結果に対して集計処理を行わなければならない場合、ネスト集計と呼ばれる演算を行います。

ほとんどのチャートの数式で集計をネストすることはできません。ただし、内部集計関数で **TOTAL** 修飾子を使用すると、集計をネストできます。

100 レベル以下のネストが可能です。

TOTAL 修飾子を含むネストされた集計関数

項目 **Sales** で、前年の **OrderDate** と等しい取引のみを集計することにします。前年のデータは、集計関数 **Max(TOTAL** Year(OrderDate)**)** で取得できます。

このような場合は、次の集計関数を使用します。

```
Sum(If(Year(OrderDate)=Max(TOTAL Year(OrderDate)), Sales))
```

Qlik Sense では、このタイプのネストに **TOTAL** 修飾子を含める必要があります。望ましい比較のために必要です。これらは非常によく使われるタイプのネストで、データの入手に適しています。

参照先:

- *Aggr - チャート関数 (page 542)*

8.3 Aggr - チャート関数

Aggr() は、指定された軸上で計算された数式の値の配列を返します。たとえば、顧客別、地域別 sales の最大値です。

Aggr 関数はネストされた集計に使用され、最初のパラメーター (内部集計) は軸の値ごとに 1 回計算されます。軸は、2 番目のパラメーター (および後続のパラメーター) で指定されます。

さらに、**Aggr** 関数は外部の集計関数で囲む必要があり、**Aggr** の結果の配列をネストされる集計への入力として使用します。

構文:

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

戻り値データ型: デュアル

引数:

引数

引数	説明
expr	集計関数で構成される数式。デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。

引数	説明
StructuredParameter	StructuredParameter は、軸と(Dimension(Sort-type, Ordering)) 形式のソート基準 (オプション) で構成されています。 軸は単一項目であり、数式にはできません。軸を使用して、Aggr 数式で計算される値の配列が決定されます。 ソート基準が含まれている場合は、軸について計算された Aggr 関数により、作成された値の配列がソートされます。これは、Aggr 関数を囲んでいる数式の結果にソート順序が影響する場合に重要です。 ソート基準の使用法について詳しくは、「構造化パラメータの軸へのソート基準の追加」を参照してください。
SetExpression	デフォルトでは、集計関数は選択されたレコードセットに対して集計を行います。Set 分析数式でレコードセットを定義することも可能です。
DISTINCT	expression 引数の前に **distinct** 修飾子が配置されている場合、あるいは修飾子がまったく使用されていない場合は、軸の値の組み合わせごとに 1 つの戻り値のみ生成されます。これは正常な集計方法で、これらの異なる組み合わせがそれぞれチャートの 1 行に反映されます。
NODISTINCT	expression 引数の前に **nodistinct** 修飾子が配置されている場合、軸の値の組み合わせは、いずれも基底のデータ構造に基づいて、複数の戻り値を生成する可能性があります。軸が 1 本だけの場合、**aggr** 関数は、ソースデータの行数と同じ数の要素を含む配列を返します。

Sum、**Min**、**Avg** などの基本的な集計関数では数値が 1 つ返されるのに対し、Aggr()関数は、一時的な段階の結果セット(仮想テーブル) を作成することと比較することができ、その結果セットで別の集計を行うことができます。例えば、**Aggr()** ステートメントで顧客別の売上を合計して平均売上値を計算し、それから加算された結果の平均値を計算することができます:**Avg(TOTAL Aggr(Sum(Sales),Customer))**。

複数のレベルでネストされたチャート集計を作成する場合は、計算軸で Aggr() 関数を使用してください。

制限事項:

Aggr() 関数の各軸は、単一の項目でなければならず、数式 (計算軸) にすることはできません。

構造化パラメータの軸へのソート基準の追加

基本形式において、Aggr 関数構文の引数 StructuredParameter は単一軸です。数式Aggr(Sum(Sales, Month)) により、各月の総売上高の値が返されます。ただし、別の集計関数によって囲まれている場合は、ソート基準が使用される場合を除いて、予期しない結果になる可能性があります。これは、軸によって、数値やアルファベットなどの異なる基準でソートされるからです。

Aggr 関数の StructuredParameter 引数には、数式での軸のソート基準を指定できます。この方法により、Aggr 関数によって作成される仮想テーブルにソート順序を適用します。

引数 StructuredParameter の構文は次のとおりです。

```
(FieldName, (Sort-type, Ordering))
```

構造化パラメータはネストすることができます。

```
(FieldName, (FieldName2, (Sort-type, Ordering)))
```

ソート タイプは、NUMERIC、TEXT、FREQUENCY、または LOAD_ORDER です。

各ソート タイプに関連付けられる順序 タイプは次のとおりです。

許可される順序 タイプ

ソート タイプ	許可される順序 タイプ
NUMERIC	ASCENDING、DESCENDING、または REVERSE
TEXT	ASCENDING、A2Z、DESCENDING、REVERSE、または Z2A
FREQUENCY	DESCENDING、REVERSE 、または ASCENDING
LOAD_ORDER	ASCENDING、ORIGINAL、DESCENDING、または REVERSE

順序 タイプ REVERSE とDESCENDING は同じです。

ソート タイプ TEXT の場合、順序 タイプ ASCENDING とA2Z は同じであり、DESCENDING、REVERSE、Z2A は同じです。

ソート タイプ LOAD_ORDER の場合、順序 タイプ ASCENDING とORIGINAL は同じです。

例: Aggr を使用したチャートの数式

例 - チャートの数式

チャートの数式例 1

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
ProductData:

LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|CC|2|20
Betacab|DD|25|25
Canutility|AA|8|15
Canutility|CC|0|19
] (delimiter is '|');
```

チャートの数式

Qlik Sense シートで KPI のビジュアライゼーションを作成します。次の数式をメジャーとして KPI に追加します。

```
Avg(Aggr(Sum(UnitSales*UnitPrice), Customer))
```

結果

376.7

説明

数式 `Aggr(Sum(UnitSales*UnitPrice), Customer)` は、**Customer** 別の売り上げの合計値で、3 つの **Customer** の値として、295、715、120 の値の配列を返します。

実質的には、それらの値を含む明示的なテーブルまたは列を作成することなく、値の一時的なリストを作成したことになります。

これらの値は **Avg()** 関数に使われ、売り上げの平均値として 376.7 という値が算出されます

チャートの数式例 2

ロードスクリプト

以下のデータをインラインデータとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
ProductData:

LOAD * inline [
Customer|Product|UnitSales|UnitPrice
Astrida|AA|4|16
Astrida|AA|10|15
Astrida|BB|10|15
Astrida|BB|9|9
Betacab|BB|5|10
Betacab|BB|7|12
Betacab|CC|2|22
Betacab|CC|4|20
Betacab|DD|25|25
Canutility|AA|8|15
Canutility|AA|5|11
Canutility|CC|0|19
] (delimiter is '|');
```

チャートの数式

Customer、**Product**、**UnitPrice**、**UnitSales** を軸としたテーブルの可視化を Qlik Sense シートを作成します。次の数式をメジャーとしてテーブルに追加します。

```
Aggr(NODISTINCT Max(UnitPrice), Customer, Product)
```

結果

Customer	Product	UnitPrice	UnitSales	Aggr(NODISTINCT Max(UnitPrice), Customer, Product)
Astrida	AA	15	10	16
Astrida	AA	16	4	16
Astrida	BB	9	9	15
Astrida	BB	15	10	15
Betacab	BB	10	5	12
Betacab	BB	12	7	12
Betacab	CC	20	4	22
Betacab	CC	22	2	22
Betacab	DD	25	25	25
Canutility	AA	11	5	15
Canutility	AA	15	8	15
Canutility	CC	19	0	19

説明

値の配列: 16、16、15、15、12、12、22、22、25、15、15、19。**nodistinct** 修飾子は、配列にはソースデータの各行の要素が1つ含まれており、各要素が **Customer** と **Product** それぞれの最大 **UnitPrice** になっていることを意味します。

チャートの数式例 3

ロードスクリプト

以下のデータをインラインデータとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
Set vNumberOfOrders = 1000;

OrderLines:
Load
       RowNo() as OrderLineID,
       OrderID,
       OrderDate,
       Round((Year(OrderDate)-2005)*1000*Rand()*Rand()*Rand1) as Sales
       While Rand()<=0.5 or IterNo()=1;
Load * Where OrderDate<=Today();
Load
       Rand() as Rand1,
       Date(MakeDate(2013)+Floor((365*4+1)*Rand())) as OrderDate,
```

```
        RecNo() as OrderID
        Autogenerate vNumberOfOrders;

Calendar:
Load distinct
        Year(OrderDate) as Year,
        Month(OrderDate) as Month,
        OrderDate
        Resident OrderLines;
```

チャートの数式

Qlik Sense シートに [**Year**] と [**Month**] を軸としたテーブルのビジュアライゼーションを作成します。次の数式をメジャーとしてテーブルに追加します。

- `Sum(Sales)`
- `Sum(Aggr( Rangesum(Above(Sum(Sales),0,12)), (Year, (Numeric, Ascending)), (Month, (Numeric, Ascending)) ))` は表の中で Structured Aggr() とラベル表示されています。

結果

Year	Month	Sum(Sales)	Structured Aggr()
2013	Jan	53495	53495
2013	Feb	48580	102075
2013	Mar	25651	127726
2013	Apr	36585	164311
2013	May	61211	225522
2013	Jun	23689	249211
2013	Jul	42311	291522
2013	Aug	41913	333435
2013	Sep	28886	362361
2013	Oct	25977	388298
2013	Nov	44455	432753
2013	Dec	64144	496897
2014	Jan	67775	67775

説明

この例では、各年の 12 か月間の集計値を年代順に昇順で表示しているため、**Aggr()** 式の構造化パラメータ (数値、昇順) の部分を使用しています。構造化パラメータとして、(1) **年** (数値) と (2) **月** (数値) でソートされている、**年** と **月** の 2 つの特定の軸が必要です。この 2 つの軸は、テーブルやチャートのビジュアライゼーションに

使用する必要があります。これは、**Aggr()** 関数の軸リストが、ビジュアライゼーションに使用されるオブジェクトの軸に対応するために必要です。

これらのメジャー間の違いは、テーブルまたは独立した折れ線グラフで比較できます。

- `Sum(Aggr( Rangesum(Above(Sum(Sales),0,12)), (Year), (Month) ))`
- `Sum(Aggr( Rangesum(Above(Sum(Sales),0,12)), (Year, (Numeric, Ascending)), (Month, (Numeric, Ascending)) ))`

後者の式のみが、目的とする集計値の蓄積を行うことは、明らかです。

参照先:

基本的な集計関数 (page 328)

8.4 カラー関数

これらの関数は、チャートオブジェクトのカラープロパティを設定および評価する数式やデータロードスクリプトで使用します。

Qlik Senseは、下位互換性の理由からカラー関数 ***Color()***、***qliktechblue***、***qliktechgray*** *に対応していますが、これらの使用はお勧めしません。*

ARGB

ARGB()は、チャートオブジェクトのカラープロパティを設定または評価する数式で使用されます。色は **alpha** のアルファ係数 (不透明度) を使用した、赤の要素 **r**、緑の要素 **g**、青の要素**b** によって定義されます。

```
ARGB (alpha, r, g, b)
```

HSL

HSL()は、チャートオブジェクトのカラープロパティを設定、または評価する数式で使用されます。色は、**hue**、**saturation**、**luminosity**の 0 ~ 1 の値で定義されます。

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

RGB

RGB() は、赤の成コンポーネントr、緑のコンポーネントg、青のコンポーネントb の 3 つのパラメータで定義された色のカラーコードに対応する整数を返します。これらのコンポーネントは、0～255 の整数値である必要があります。この関数を数式で使用して、チャートオブジェクトのカラープロパティを設定または評価できます。

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

Colormix1

Colormix1() は、0 ~ 1 の間の値を基準に 2 色グラデーションの ARGB カラー表現を返す数式で使用されます。

```
Colormix1 (Value , ColorZero , ColorOne)
```

Value は、0 と1 の間の実数です。

- Value = 0 の場合、ColorZero が返されます。
- Value = 1 の場合、ColorOne が返されます。
- 0 < Value < 1 のとき、相当する中間の陰影を持つ色を返します。

ColorZero は、色を間隔の下端に関連付ける有効な RGB カラー表現です。

ColorOne は、色を間隔の上端に関連付ける有効な RGB カラー表現です。

```
Colormix1(0.5, red(), blue())
```
の戻り値:

```
ARGB(255,64,0,64) (purple)
```

Colormix2

Colormix2() は、-1 ~ 1 の間の値を基準に 2 色グラデーションの ARGB カラー表現を返す数式で使用されます。中心位置 (0) に中間色を指定することもできます。

```
Colormix2 (Value ,ColorMinusOne , ColorOne[ , ColorZero])
```
Value は、-1 と1 の間の実数です。

- Value = -1 のとき、1 つ目の色を返します。
- Value = 1 のとき、2 つ目の色を返します。
- -1 < Value< 1 の場合、適切なカラー ミックスが返されます。

ColorMinusOne は、色を間隔の下端に関連付ける有効な RGB カラー表現です。

ColorOne は、色を間隔の上端に関連付ける有効な RGB カラー表現です。

ColorZero は、色を間隔の中間に関連付ける有効なオプションの RGB カラー表現です。

SysColor

SysColor() は、Windows システム色 nr の ARGB カラー表現を返します。nr は、Windows API 関数 **GetSysColor(nr)** へのパラメータに相当します。

```
SysColor (nr)
```

ColorMapHue

ColorMapHue() は、カラーマップからの色の ARGB 値を返します。カラーマップは、HSV カラーモデルの色相要素によって異なります。カラー マップは赤から始まり、黄、緑、シアン、青、マゼンタを通り、赤に戻ります。x は 0 ~ 1 の値で指定される必要があります。

```
ColorMapHue (x)
```

ColorMapJet

ColorMapJet() は、カラー マップからの色のARGB 値を返します。カラー マップは青から始まり、シアン、黄、オレンジを通って赤に戻ります。x は 0 ~ 1 の値で指定される必要があります。

```
ColorMapJet (x)
```

定義済みのカラー関数

次の関数は、定義済みの色の数式で使用できます。各関数は、RGB カラー表現を返します。

任意で、アルファ係数のパラメータを指定できます。その場合、ARGB カラー表現が返されます。アルファ係数 0 は完全な透明に相当し、255 は完全な不透明色に相当します。アルファの値が入力されていない場合、255 と見なされます。

定義済みのカラー関数

カラー関数	RGB 値
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)

例と結果:

例と結果

例	結果
`Blue()`	RGB(0,0,128)
`Blue(128)`	ARGB(128,0,0,128)

ARGB

ARGB()は、チャートオブジェクトのカラープロパティを設定または評価する数式で使用されます。色は **alpha** のアルファ係数 (不透明度) を使用した、赤の要素 **r**、緑の要素 **g**、青の要素 **b** によって定義されます。

構文:

```
ARGB(alpha, r, g, b)
```

戻り値データ型: dual

引数:

引数

引数	説明
alpha	範囲 0 ~ 255 の透過性値。0 は完全な透明で、255 は完全な不透明色です。
r, g, b	赤、緑および青の成分値。成分値は 0 が最小値、255 が最大値になります。

すべての引数は、0 から 255 の範囲の整数で分解できる数式でなければなりません。

数値コンポーネントと書式が 16 進法で解釈されている場合、色成分の値はより分かりやすくなります。たとえば、薄緑色の数値は 4 278 255 360 となり、16 進法では FF00FF00 となります。最初の 2 桁 'FF' (255) は、**alpha** チャネルを示します。次の 2 桁 '00' は **red** の量、その次の 2 桁 'FF' は **green** の量、最後の 2 桁 '00' は **blue** の量を示します.

RGB

RGB() は、赤の成コンポーネントr、緑のコンポーネントg、青のコンポーネントb の 3 つのパラメータで定義された色のカラー コードに対応する整数を返します。これらのコンポーネントは、0~255 の整数値である必要があります。この関数を数式で使用して、チャート オブジェクトのカラープロパティを設定または評価できます。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
r, g, b	赤、緑および青の成分値。成分値は 0 が最小値、255 が最大値になります。

すべての引数は、0 から 255 の範囲の整数で分解できる数式でなければなりません。

数値コンポーネントと書式が 16 進法で解釈されている場合、色成分の値はより分かりやすくなります。たとえば、薄緑色の数値は 4 278 255 360 となり、16 進法では FF00FF00 となります。最初の 2 桁 'FF' (255) は、**alpha** チャネルを示します。関数 **RGB** と**HSL** では、常に 'FF' (不透明) です。次の 2 桁 '00' は **red** の量、その次の 2 桁 'FF' は **green** の量、最後の 2 桁 '00' は **blue** の量を示します.

例: チャートの数式
この例では、カスタム カラーをチャートに適用します。

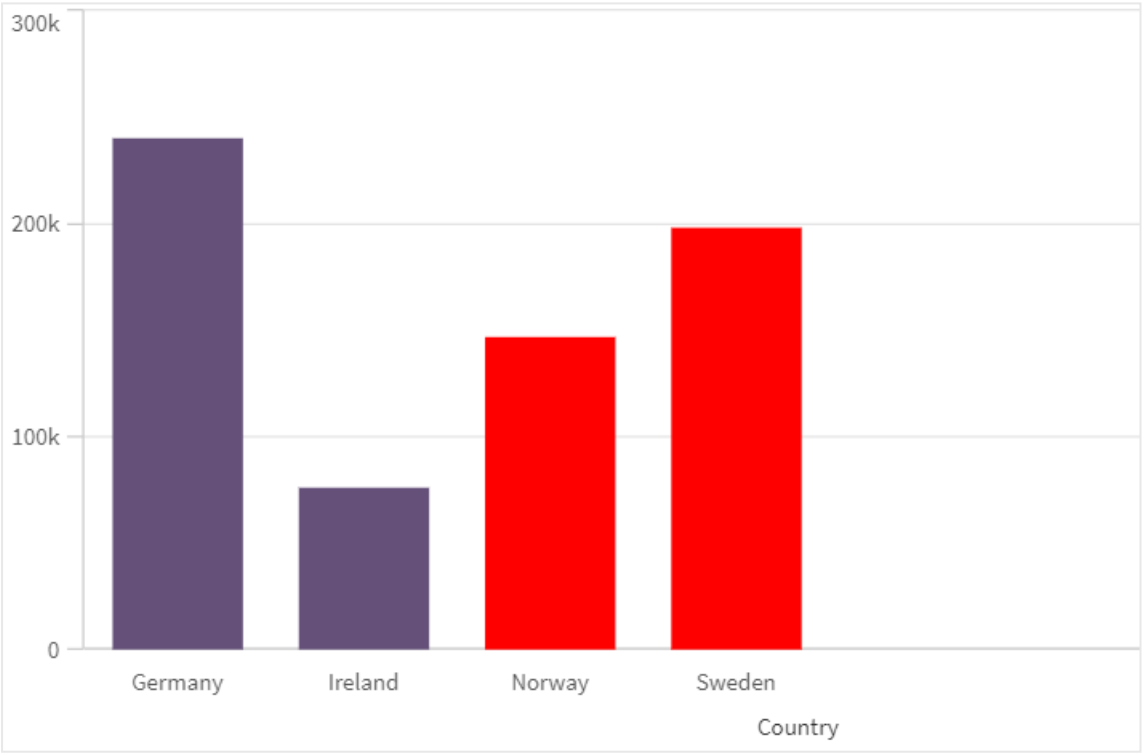
この例で使用されているデータ:

```
ProductSales:
Load * Inline
[Country,Sales,Budget
Sweden,100000,50000
Germany, 125000, 175000
Norway, 74850, 68500
Ireland, 45000, 48000
Sweden,98000,50000
Germany, 115000, 175000
Norway, 71850, 68500
Ireland, 31000, 48000
] (delimiter is ',');
```

[色と凡例] プロパティパネルに次の数式を入力します。

```
If (Sum(Sales)>Sum(Budget),RGB(255,0,0),RGB(100,80,120))
```

結果:



例: ロードスクリプト
次の例は、16 進形式の値と同等の RGB 値を表示します。

```
Load
Text(R & G & B) as Text,
RGB(R,G,B)      as Color;
Load
```

```
Num#(R,'(HEX)') as R,
Num#(G,'(HEX)') as G,
Num#(B,'(HEX)') as B
Inline
[R,G,B
01,02,03
AA,BB,CC];
```

結果:

テキスト	色
010203	RGB(1,2,3)
AABBCC	RGB(170,187,204)

HSL

HSL()は、チャートオブジェクトのカラープロパティを設定、または評価する数式で使用されます。色は、**hue**、**saturation**、**luminosity**の 0 ~ 1 の値で定義されます。

構文:

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

戻り値データ型: dual

引数:

引数

引数	説明
hue, saturation, luminosity	hue および saturation、luminosity コンポーネントの値は、0 から1 の範囲です。

すべての引数は、0 から1 の範囲の整数で分解できる数式でなければなりません。

数値コンポーネントと書式が16 進法で解釈されている場合、色成分のRGB 値はより分かりやすくなります。たとえば、薄緑色の数値は 4 278 255 360 となり、16 進法では FF00FF00 および RGB (0,255,0) となります。これは、HSL (80/240, 240/240, 120/240) と同等で、(0.33, 1, 0.5) の HSL 値です。

8.5 条件分岐関数

条件分岐関数は条件を評価し、条件値に応じて異なる答えを返します。関数は、データロードスクリプトおよびチャートの数式で使用できます。

条件分岐関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

alt

alt 関数は、有効な数値表現を持つ最初のパラメータを返します。一致が見つからない場合は、最後のパラメータを返します。任意の数のパラメータを使用できます。

```
alt (expr1[ , expr2 , expr3 , ...] , else)
```

class

class 関数は、class の間隔に最初のパラメータを割り当てます。結果は dual 値であり、a<=x<bがテキスト値として含まれています (aとbはビンの上限と下限で、下限は数値で示されます)。

```
class (expression, interval [ , label [ , offset ]])
```

coalesce

合体関数は、有効な non-NULL 表現を持つ最初のパラメータを返します。任意の数のパラメータを使用できます。

```
coalesce(expr1[ , expr2 , expr3 , ...])
```

if

if 関数は、与えられた条件が True または False のどちらに評価されるかによって異なる値を返します。

```
if (condition , then , else)
```

match

match 関数は、最初のパラメータを後続のすべてのパラメータと比較し、一致する数式の数値の位置を返します。比較では大文字と小文字が区別されます。

```
match ( str, expr1 [ , expr2,...exprN ])
```

mixmatch

mixmatch 関数は、最初のパラメーターとそれに続くすべてのパラメーターを比較し、一致した数式の数値の場所を返します。比較では大文字と小文字は区別されず、日本語のひらがなとカタカナも区別されません。

```
mixmatch ( str, expr1 [ , expr2,...exprN ])
```

pick

pick 関数は、リストの *n* 番目の数式を返します。

```
pick (n, expr1[ , expr2,...exprN])
```

wildmatch

wildmatch 関数は、最初のパラメーターとそれに続くすべてのパラメーターを比較し、一致した数式の数を返します。比較文字列でワイルドカード文字 (***** および **?**) を使用できます。***** は任意の文字のシーケンスと一致します。**?** は任意の 1 文字と一致します。比較では大文字と小文字は区別されず、日本語のひらがなとカタカナも区別されません。

```
wildmatch ( str, expr1 [ , expr2,...exprN ])
```

alt

alt 関数は、有効な数値表現を持つ最初のパラメータを返します。一致が見つからない場合は、最後のパラメータを返します。任意の数のパラメータを使用できます。

構文:

```
alt(expr1[ , expr2 , expr3 , ...] , else)
```

引数:

引数

引数	説明
expr1	有効な数値表現を確認する最初の数式。
expr2	有効な数値表現を確認する2番目の数式。
expr3	有効な数値表現を確認する3番目の数式。
else	以前のパラメータに有効な数値表現が存在しない場合に返す値。

alt 関数は、数値または日付の変換関数で使用されます。このように、Qlik Sense は、優先順順位に従って異なる日付形式をテストできます。また、数式での NULL 値の処理にも使用できます。

例

例	結果
alt(date#(dat , 'YYYY/MM/DD'), date#(dat , 'MM/DD/YYYY'), date#(dat , 'MM/DD/YY'), 'No valid date')	この数式は、項目の日付が指定された3つの日付形式のいずれかに従っているかどうかをテストします。従っている場合は、元の文字列と日付の有効な数値表現を含むデュアル値を返します。どの形式にも従っていない場合、'No valid date' というテキストが返されます (有効な数値表現は含まれません)。
alt(Sales,0) + alt(Margin,0)	この数式は、項目 Sales とMargin を追加して、すべての欠損値 (NULL) を 0 で置き換えます。

class

class 関数は、class の間隔に最初のパラメータを割り当てます。結果は dual 値であり、a<=x<bがテキスト値として含まれています (aとbはビンの上限と下限で、下限は数値で示されます)。

構文:

```
class(expression, interval [ , label [ , offset ]])
```

引数:

引数

引数	説明
interval	ビン幅を特定する数値。
label	結果のテキストの 'x' を置き換えることができる任意の文字列。
offset	デフォルトの分類開始点から、オフセットとして使用される数値。デフォルトの開始点は、通常 0 です。

例

例	結果
`class( var,10 )` で、`var = 23`	の戻り値: '20<=x<30'
`class( var,5,'value' )` で、`var = 23`	の戻り値: '20<= value <25'
`class( var,10,'x',5 )` で、`var = 23`	の戻り値: '15<=x<25'

例 - class を使用したロードスクリプト

例: ロードスクリプト

ロードスクリプト

この例では、人々の名前と年齢を含むテーブルをロードします。10 歳単位での年齢グループにより、各人を分類する項目を追加します。元のソーステーブルは次のようになります。

結果

Name	Age
John	25
Karen	42
Yoshi	53

年齢グループ分類項目を追加するために、**class** 関数を使用する先行する load ステートメントを追加できます。

データロードエディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
LOAD *,
class(Age, 10, 'age') As Agegroup;
```

```
LOAD * INLINE
[ Age, Name
25, John
42, Karen
53, Yoshi];
```

結果

結果

Name	Age	Agegroup
John	25	20 <= age < 30
Karen	42	40 <= age < 50
Yoshi	53	50 <= age < 60

coalesce

合体関数は、有効な non-NULL 表現を持つ最初のパラメータを返します。任意の数のパラメータを使用できます。

構文:

```
coalesce(expr1[ , expr2 , expr3 , ...])
```

引数:

引数

引数	説明
expr1	有効な NULL 以外の表現を確認する最初の数式。
expr2	有効な NULL 以外表現を確認する 2 番目の数式。
expr3	有効な NULL 以外表現を確認する 3 番目の数式。

例

例	結果
	この数式は、項目のすべての NULL 値を「N/A」に変更します。
Coalesce(ProductDescription, ProductName, ProductCode, 'no description available')	この数式は、一部の項目に製品の値がない場合に備えて、3 つの異なる製品説明項目から選択します。null 以外の値を持つ、指定された順序の最初の項目が返されます。どの項目にも値が含まれていない場合、結果は「説明がありません」になります。

例	結果
`Coalesce(TextBetween(FileName, '"', '"'), FileName)`	この数式は、項目 [*FileName*] から潜在的な囲み引用符を削除します。指定された [*FileName*] が引用符で囲まれている場合、これらは削除され、囲まれた引用符で囲まれていない [*FileName*] が返されます。[*TextBetween*] 関数が区切り文字を見つけられない場合、null を返します。これは、[**Coalesce**] が拒否し、代わりに生の [*FileName*] を返します。

if

if 関数は、与えられた条件が True または False のどちらに評価されるかによって異なる値を返します。

構文:

```
if(condition , then [, else])
```

引数

引数	説明
condition	論理的に解釈された数式。
then	数式は任意の型に指定できます。*condition* が True の場合、if 関数は *then* 数式の値を返します。
else	数式は任意の型に指定できます。*condition* が False の場合、if 関数は *else* 数式の値を返します。 このパラメータはオプションです。*condition* が False の場合、else を指定しないとNULL が返されます。

例

例	結果
`if( Amount>= 0, 'OK', 'Alarm' )`	この数式は、Amount が正の数 (0 以上) かどうかテストし、正の数であれば 'OK' を返します。Amount が 0 未満であれば、'Alarm' を返します。

例 - if を使用したロードスクリプト

例: ロードスクリプト

ロードスクリプト

If は、他のメソッドやオブジェクト (変数を含む) とともにロードスクリプトで使用できます。たとえば、変数 *threshold* を設定し、このしきい値に基づいてデータモデルに項目を含める場合、以下を実行できます。

データロードエディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange
3752, 20180916, 5.75, 1, 5646471, S, blue
3753, 20180922, 125.00, 7, 3036491, l, Black
3754, 20180922, 484.21, 13, 049681, xs, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
];

set threshold = 100;

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

Transaction_Buckets:
Load
   transaction_id,
   If(transaction_amount > $(threshold),'Greater than $(threshold)','Less than $(threshold)')
as [Compared to Threshold]
Resident Transactions;
```

結果

ロードスクリプトで *if* 関数を使用した結果の出力を示す Qlik Sense のテーブル。

transaction_id	しきい値と比較
3750	100 未満
3751	100 超
3752	100 未満
3753	100 超
3754	100 超
3756	100 未満
3757	100 超

例 - if を使用したチャートの数式

例: チャートの数式

チャートの数式 1

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。データをロードした後、以下のチャートの数式の例を Qlik Sense テーブルに作成します。

```
MyTable:
LOAD * inline [Date, Location, Incidents
1/3/2016, Beijing, 0
1/3/2016, Boston, 12
1/3/2016, Stockholm, 3
1/3/2016, Toronto, 0
1/4/2016, Beijing, 0
1/4/2016, Boston, 8];
```

Qlik Sense のテーブルは、チャートの数式における *if* 関数の例を示しています。

日付	場所	Incidents	if(Incidents>=10, 'Critical', 'Ok')	if(Incidents>=10, 'Critical', If(Incidents>=1 and Incidents<10, 'Warning', 'Ok'))
1/3/2016	Beijing	0	Ok	Ok
1/3/2016	Boston	12	Critical	Critical
1/3/2016	Stockholm	3	Ok	Warning
1/3/2016	Toronto	0	Ok	Ok
1/4/2016	Beijing	0	Ok	Ok
1/4/2016	Boston	8	Ok	警告

チャートの数式 2

新しいアプリで、データ ロード エディターの新しいタブに次のスクリプトを追加してから、データをロードします。次に、以下のチャートの数式を使用してテーブルを作成できます。

```
SET FirstWeekDay=0;
Load
Date(MakeDate(2022)+RecNo()-1) as Date
Autogenerate 14;
```

Qlik Sense のテーブルは、チャートの数式における *if* 関数の例を示しています。

Date	WeekDay(Date)	If(WeekDay (Date)>=5,'WeekEnd','Normal Day')
1/1/2022	Sat	WeekEnd
1/2/2022	日	WeekEnd
1/3/2022	月	Normal Day
1/4/2022	火	Normal Day
1/5/2022	水	Normal Day
1/6/2022	Thu	Normal Day
1/7/2022	Fri	Normal Day
1/8/2022	Sat	WeekEnd
1/9/2022	日	WeekEnd
1/10/2022	月	Normal Day
1/11/2022	火	Normal Day
1/12/2022	水	Normal Day
1/13/2022	Thu	Normal Day
1/14/2022	Fri	Normal Day

match

match 関数は、最初のパラメータを後続のすべてのパラメータと比較し、一致する数式の数値の位置を返します。比較では大文字と小文字が区別されます。

構文:

```
match( str, expr1 [ , expr2,...exprN ])
```

*大文字と小文字を区別せずに比較する場合は、**mixmatch** 関数を使用します。大文字と小文字を区別せずにワイルドカードを使用して比較する場合は、**wildmatch** 関数を使用します。*

例:match を使用したロードスクリプト

例: ロードスクリプト

ロードスクリプト

match を使用してデータのサブセットをロードできます。たとえば、関数内の数式の数値を返すことができます。次に、その数値に基づいて、ロードされるデータを制限できます。一致がない場合、Match は 0 を返します。よって、この例に適合しないすべての数式は 0 を返し、WHERE ステートメントによるデータロードから除外されま

す。

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange
3752, 20180916, 5.75, 1, 5646471, S, blue
3753, 20180922, 125.00, 7, 3036491, l, Black
3754, 20180922, 484.21, 13, 049681, xs, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
];

/*
Create new table called Transaction_Buckets
Create new fields called Customer, and Color code - Blue and Black
Load Transactions table.
Match returns 1 for 'Blue', 2 for 'Black'.
Does not return a value for 'blue' because match is case sensitive.
Only values that returned numeric value greater than 0
are loaded by WHERE statment into Transactions_Buckets table.
*/

Transaction_Buckets:
Load
 customer_id,
 customer_id as [Customer],
 color_code as [Color Code Blue and Black]
Resident Transactions
Where match(color_code,'Blue','Black') > 0;
```

結果

ロード スクリプトで match 関数を使用した結果の出力を示す Qlik Sense テーブル

Color Code Blue and Black	Customer
ブラック	203521
ブラック	3036491
青	2038593

例 - match を使用したチャートの数式

例: チャートの数式

チャートの数式 1

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。データをロードした後、以下のチャートの数式の例を Qlik Sense テーブルに作成します。

```
MyTable:
Load * inline [Cities, Count
Toronto, 123
Toronto, 234
Toronto, 231
Boston, 32
Boston, 23
Boston, 1341
Beijing, 234
Beijing, 45
Beijing, 235
Stockholm, 938
Stockholm, 39
Stockholm, 189
zurich, 2342
zurich, 9033
zurich, 0039];
```

以下のテーブルの最初の数式は Stockholm の場合に 0 を返します。これは、**match** 関数の数式のリストに「Stockholm」が含まれていないからです。**match** 比較では大文字と小文字が区別されるため、「Zurich」の場合には 0 も返します。

Qlik Sense のテーブルは、チャートの数式における *match* 関数の例を示しています

Cities	**match(Cities,'Toronto','Boston','Beijing','Zurich')**	**match(Cities,'Toronto','Boston','Beijing','Stockholm','Zurich')**
Beijing	3	3
Boston	2	2
Stockholm	0	4
Toronto	1	1
zurich	0	5

チャートの数式 2

match を使用して、1 つの数式のカスタム ソートを実行できます。

既定では、データに応じて、列が数値またはアルファベット順にソートされます。

既定のソート順の例を示した Qlik Sense テーブル

Cities
Beijing
Boston
Stockholm
Toronto
zurich

順序を変更するには、以下のステップを実行します。

1. **プロパティ**パネルで、グラフの **[ソート]** セクションを開きます。
2. カスタム ソートを適用する列の自動ソートをオフにします。
3. **[数値によるソート]** と **[アルファベット順でソート]** の選択を解除します。
4. **[数式によるソート]** を選択し、次に似た数式を入力します。
   `=match( Cities, 'Toronto','Boston','Beijing','Stockholm','zurich')`
   Cities 列のソート順序が変更されます。

match 関数を使用したソート順の変更例を示した Qlik Sense テーブル

Cities
Toronto
Boston
Beijing
Stockholm
zurich

返される数値を表示することもできます。

match 関数から返される数値の例を示した Qlik Sense テーブル

Cities	**Cities & ' - ' & match (Cities, 'Toronto','Boston', 'Beijing','Stockholm','zurich')**
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

mixmatch

mixmatch 関数は、最初のパラメーターとそれに続くすべてのパラメーターを比較し、一致した数式の数値の場所を返します。比較では大文字と小文字は区別されず、日本語のひらがなとカタカナも区別されません。

構文:

```
mixmatch( str, expr1 [ , expr2,...exprN ])
```

代わりに大文字と小文字を区別して比較する場合は、**match** 関数を使用します。大文字と小文字を区別せずにワイルドカードを使用して比較する場合は、**wildmatch** 関数を使用します。

例 - mixmatch を使用したロードスクリプト

例: ロードスクリプト

ロードスクリプト

mixmatch を使用してデータのサブセットをロードできます。たとえば、関数内の数式の数値を返すことができます。次に、その数値に基づいて、ロードされるデータを制限できます。一致がない場合、Mixmatch は 0 を返します。よって、この例に適合しないすべての数式は 0 を返し、WHERE ステートメントによるデータロードから除外されます。

データロードエディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Load * Inline [ transaction_id, transaction_date, transaction_amount, transaction_quantity,
customer_id, size, color_code 3750, 20180830, 23.56, 2, 2038593, L, Red 3751, 20180907,
556.31, 6, 203521, m, orange 3752, 20180916, 5.75, 1, 5646471, S, blue 3753, 20180922, 125.00,
7, 3036491, l, Black 3754, 20180922, 484.21, 13, 049681, xs, Red 3756, 20180922, 59.18, 2,
2038593, M, Blue 3757, 20180923, 177.42, 21, 203521, XL, Black ];  /* Create new table called
Transaction_Buckets Create new fields called Customer, and Color code - Black, Blue, blue Load
Transactions table. Mixmatch returns 1 for 'Black', 2 for 'Blue'.  Also returns 3 for 'blue'
because mixmatch is not case sensitive. Only values that returned numeric value greater than 0
 are loaded by WHERE statement into Transactions_Buckets table. */  Transaction_Buckets: Load
  customer_id,     customer_id as [Customer],    color_code as [Color Code - Black, Blue,
blue] Resident Transactions Where mixmatch(color_code,'Black','Blue') > 0;
```

結果

ロードスクリプトで mixmatch 関数を使用した結果の出力を示す Qlik Sense のテーブル。

Color Code Black, Blue, blue	Customer
ブラック	203521
ブラック	3036491
青	2038593
blue	5646471

例 - mixmatch を使用したチャートの数式

例: チャートの数式
データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。データをロードした後、以下のチャートの数式の例を Qlik Sense テーブルに作成します。

チャートの数式 1

```
MyTable: Load * inline [Cities, Count Toronto, 123 Toronto, 234 Toronto, 231 Boston, 32
Boston, 23 Boston, 1341 Beijing, 234 Beijing, 45 Beijing, 235 Stockholm, 938 Stockholm, 39
Stockholm, 189 zurich, 2342 zurich, 9033 zurich, 0039];
```

以下のテーブルの最初の数式は Stockholm の場合に 0 を返します。これは、**mixmatch** 関数の数式のリストに「Stockholm」が含まれていないからです。**mixmatch** 比較では大文字と小文字が区別されないため、「Zurich」の場合には 4 を返します。

Qlik Sense のテーブルは、チャートの数式における *mixmatch* 関数の例を示しています

Cities	**mixmatch(Cities,'Toronto','Boston','Beijing','Zurich')**	**mixmatch(Cities,'Toronto','Boston','Beijing','Stockholm','Zurich')**
Beijing	3	3
Boston	2	2
Stockholm	0	4
Toronto	1	1
zurich	4	5

チャートの数式 2

mixmatch を使用して、1 つの数式のカスタム ソートを実行できます。

既定では、データに応じて、列がアルファベットまたは数値順にソートされます。

既定のソート順の例を示した Qlik Sense テーブル

Cities
Beijing
Boston
Stockholm
Toronto
zurich

順序を変更するには、以下のステップを実行します。

1. **プロパティ**パネルで、グラフの **[ソート]** セクションを開きます。
2. カスタム ソートを適用する列の自動ソートをオフにします。
3. **[数値によるソート]** と **[アルファベット順でソート]** の選択を解除します。
4. **[数式によるソート]** を選択し、次の数式を入力します。
   `=mixmatch( Cities, 'Toronto','Boston','Beijing','Stockholm','Zurich')`
   Cities 列のソート順序が変更されます。

mixmatch 関数を使用したソート順の変更例を示した Qlik Sense テーブル。

Cities
Toronto
Boston
Beijing
Stockholm
zurich

返される数値を表示することもできます。

mixmatch 関数から返される数値の例を示した Qlik Sense テーブル。

Cities	**Cities & ' - ' & mixmatch (Cities, 'Toronto','Boston', 'Beijing','Stockholm','Zurich')**
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

pick

pick 関数は、リストの *n* 番目の数式を返します。

構文:

```
pick(n, expr1[ , expr2,...exprN])
```

引数:

引数

引数	説明
n	*n* は 1 から N の間の整数です。

例

例	結果
pick(N, 'A','B',4, 6)	'B' を返します (N = 2 の場合) 4 を返します (N = 3 の場合)

wildmatch

wildmatch 関数は、最初のパラメーターとそれに続くすべてのパラメーターを比較し、一致した数式の数を返します。比較文字列でワイルドカード文字 (***** および **?**) を使用できます。***** は任意の文字のシーケンスと一致します。**?** は任意の 1 文字と一致します。比較では大文字と小文字は区別されず、日本語のひらがなとカタカナも区別されません。

構文:

```
wildmatch( str, expr1 [ , expr2,...exprN ])
```

ワイルドカードを使わずに比較する場合は、**match** 関数または **mixmatch** 関数を使用します。

例:wildmatch を使用したロードスクリプト

例: ロードスクリプト

ロードスクリプト

wildmatch を使用してデータのサブセットをロードできます。たとえば、関数内の数式の数値を返すことができます。次に、その数値に基づいて、ロードされるデータを制限できます。一致がない場合、Wildmatch は 0 を返します。よって、この例に適合しないすべての数式は 0 を返し、WHERE ステートメントによるデータロードから除外されます。

データロードエディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions: Load * Inline [ transaction_id, transaction_date, transaction_amount,
transaction_quantity, customer_id, size, color_code 3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, m, orange 3752, 20180916, 5.75, 1, 5646471, S, blue 3753,
20180922, 125.00, 7, 3036491, l, Black 3754, 20180922, 484.21, 13, 049681, xs, Red 3756,
20180922, 59.18, 2, 2038593, M, Blue 3757, 20180923, 177.42, 21, 203521, XL, Black ];   /*
Create new table called Transaction_Buckets Create new fields called Customer, and Color code
- Black, Blue, blue, red Load Transactions table. Wildmatch returns 1 for 'Black', 'Blue', and
'blue', and 2 for 'Red'.  Only values that returned numeric value greater than 0  are loaded
by WHERE statement into Transactions_Buckets table. */   Transaction_Buckets: Load
customer_id,    customer_id as [Customer],    color_code as [Color Code Black, Blue, blue,
Red] Resident Transactions Where wildmatch(color_code,'Bl*','R??') > 0;
```

結果

ロード スクリプトで *wildmatch* 関数を使用した結果の出力を示す Qlik Sense のテーブル

Color Code Black, Blue, blue, Red	Customer
ブラック	203521
ブラック	3036491
青	2038593
blue	5646471
赤色	049681
赤色	2038593

例:wildmatch を使用したチャートの数式

例: チャートの数式

チャートの数式 1

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。データをロードした後、以下のチャートの数式の例を Qlik Sense テーブルに作成します。

```
MyTable: Load * inline [Cities, Count Toronto, 123 Toronto, 234 Toronto, 231 Boston, 32
Boston, 23 Boston, 1341 Beijing, 234 Beijing, 45 Beijing, 235 Stockholm, 938 Stockholm, 39
Stockholm, 189 zurich, 2342 zurich, 9033 zurich, 0039];
```

以下のテーブルの最初の数式は Stockholm の場合に 0 を返します。これは、**wildmatch** 関数の数式のリストに「Stockholm」が含まれていないからです。また、? は任意の 1 文字に適合するため、「Boston」の場合にも 0 を返します。

Qlik Sense のテーブルは、チャートの数式における *wildmatch* 関数の例を示しています

Cities	wildmatch(Cities,'Tor*','?ton','Beijing','*urich')	wildmatch(Cities,'Tor*','???ton','Beijing','Stockholm','*urich')
Beijing	3	3
Boston	0	2
Stockholm	0	4
Toronto	1	1
zurich	4	5

チャートの数式 2

wildmatch を使用して、1 つの数式のカスタム ソートを実行できます。

既定では、データに応じて、列が数値またはアルファベット順にソートされます。

既定のソート順の例を示した Qlik Sense テーブル

Cities
Beijing
Boston
Stockholm
Toronto
zurich

順序を変更するには、以下のステップを実行します。

1. **プロパティ**パネルで、グラフの **[ソート]** セクションを開きます。
2. カスタム ソートを適用する列の自動ソートをオフにします。
3. **[数値によるソート]** と **[アルファベット順でソート]** の選択を解除します。
4. **[数式によるソート]** を選択し、次に似た数式を入力します。
   ```
   =wildmatch( Cities, 'Tor*','???ton','Beijing','Stockholm','*urich')
   ```
   Cities 列のソート順序が変更されます。

wildmatch 関数を使用したソート順の変更例を示した Qlik Sense テーブル。

Cities
Toronto
Boston
Beijing
Stockholm
zurich

返される数値を表示することもできます。

wildmatch 関数から返される数値の例を示した Qlik Sense テーブル

Cities	**Cities & ' - ' & wildmatch (Cities, 'Tor*','???ton','Beijing','Stockholm','*urich')**
Toronto	Toronto - 1
Boston	Boston - 2
Beijing	Beijing - 3
Stockholm	Stockholm - 4
zurich	zurich - 5

8.6 カウンタ関数

このセクションでは、データロードスクリプトで **LOAD** ステートメント評価中のレコードカウンタに関連する関数について説明します。チャート数式で使用される唯一の関数は、**RowNo()** です。

一部のカウンタ関数はパラメータを取りませんが、末尾の括弧は必要です。

カウンタ関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

autonumber

このスクリプト関数は、スクリプトの実行中に発生する *expression* の個々の評価値について、一意の整数値を返します。この関数は、複合キーのコンパクトメモリ表示を作成する場合などに使用します。

```
autonumber (expression[ , AutoID])
```

autonumberhash128

このスクリプト関数は、複合入力式の値の 128 ビットハッシュ値を計算し、スクリプトの実行中に発生する個々のハッシュ値について一意の整数値を返します。この関数は、複合キーのコンパクトメモリ表示を作成する場合などに使用します。

```
autonumberhash128 (expression {, expression})
```

autonumberhash256

このスクリプト関数は、複合入力式の値の 256 ビットハッシュ値を計算し、スクリプトの実行中に発生する個々のハッシュ値について一意の整数値を返します。この関数は、複合キーのコンパクトメモリ表示を作成する場合などに使用します。

```
autonumberhash256 (expression {, expression})
```

IterNo

このスクリプト関数は、**while** 節を含む **LOAD** ステートメントで、単一のレコードが評価された回数を示す整数を返します。最初の反復の値は 1 です。**IterNo** 関数は、**while** 節と共に使用される場合にのみ有効となります。

```
IterNo ( )
```

RecNo

このスクリプト関数は、現在のテーブルで読み取られている行番号を整数で返します。最初のレコードの番号は 1 です。

```
RecNo ( )
```

RowNo - script function

この関数は、結果として得られる Qlik Sense の内部テーブルの現在の行の位置を整数で返します。最初の行は 1 です。

```
RowNo ( )
```

RowNo - chart function
RowNo() は、テーブルの現在の列セグメント内の現在行の数を返します。ビットマップチャートの場合、**RowNo()** はストレートテーブルに相当するセグメントに含まれる現在の行の数を返します。

```
RowNo - チャート関数([TOTAL])
```

autonumber

このスクリプト関数は、スクリプトの実行中に発生する *expression* の個々の評価値について、一意の整数値を返します。この関数は、複合キーのコンパクトメモリ表示を作成する場合などに使用します。

> ***autonumber***キーは、テーブルが読み込まれた順番で生成されるため、同じデータロードで生成された場合のみ結合できます。ソースデータのソート処理から独立してデータロード間で恒久的に維持されるキーを使用する必要がある場合は、***hash128*** 関数、***hash160*** 関数、***hash256*** 関数を使用する必要があります。

構文:

```
autonumber(expression[ , AutoID])
```

引数:

引数	説明
AutoID	**autonumber** 関数が1つのスクリプト内の複数のキーで使用されている場合に、複数のカウンタインスタンスを作成するには、オプションのパラメータ *AutoID* を使用して各カウンタに名前を付けます。

複合キーの作成

この例では、メモリを保護するために、**autonumber** 関数を使用して複合キーを作成します。この例は、デモのために、簡略化したものになっていますが、この方法が効果的なのは、多数の行が含まれるテーブルで使用した場合です。

データの例

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

インライン データを使用して、ソース データをロードします。次に、Region 項目、Year 項目、Month 項目から複合キーを作成する、先行する LOAD を追加します。

```
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
];
```

この結果、テーブルは次のようになります。

結果テーブル

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

この例では、別のテーブルにリンクする必要がある場合、たとえば、文字列 "North2014May" の代わりに RYMkey の 1 を参照できます。

同様の方法で、Costs のソース テーブルをロードします。Region 項目、Year 項目、Month 項目は、**autonumber** 関数を使用しテーブルをリンクしてすでに複合キーを作成しているため、合成キーの作成を避けるために先行する LOAD から除外されます。

```
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
];
```

テーブル ビジュアライゼーションをシートに追加し、Region 項目、Year 項目、Month 項目、および Sales と Costs の Sum メジャーを追加できるようになりました。テーブルは次のようになります。

結果テーブル

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

autonumberhash128

このスクリプト関数は、複合入力式の値の 128 ビット ハッシュ値を計算し、スクリプトの実行中に発生する個々のハッシュ値について一意の整数値を返します。この関数は、複合キーのコンパクト メモリ表示を作成する場合などに使用します。

***autonumberhash128** キーは、テーブルが読み込まれた順番で生成されるため、同じデータ ロードで生成された場合のみ結合できます。ソース データのソート処理から独立してデータ ロード間で恒久的に維持されるキーを使用する必要がある場合は、**hash128** 関数、**hash160** 関数、**hash256** 関数を使用する必要があります。*

構文:

```
autonumberhash128(expression {, expression})
```

複合キーの作成

この例では、メモリを保護するために、**autonumberhash128** 関数を使用して複合キーを作成します。この例は、デモのために、簡略化したものになっていますが、この方法が効果的なのは、多数の行が含まれるテーブルで使用した場合です。

データの例

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

インライン データを使用して、ソース データをロードします。次に、Region 項目、Year 項目、Month 項目から複合キーを作成する、先行する LOAD を追加します。

```
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
];
```

この結果、テーブルは次のようになります。

結果テーブル

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

この例では、別のテーブルにリンクする必要がある場合、たとえば、文字列 "North2014May" の代わりに RYMkey の 1 を参照できます。

同様の方法で、Costs のソース テーブルをロードします。Region 項目、Year 項目、Month 項目は、**autonumberhash128** 関数を使用しテーブルをリンクしてすでに複合キーを作成しているため、合成キーの作成を避けるために先行する LOAD から除外されます。

```
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
];
```

テーブル ビジュアライゼーションをシートに追加し、Region 項目、Year 項目、Month 項目、および Sales と Costs の Sum メジャーを追加できるようになりました。テーブルは次のようになります。

結果テーブル

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

autonumberhash256

このスクリプト関数は、複合入力式の値の 256 ビット ハッシュ値を計算し、スクリプトの実行中に発生する個々のハッシュ値について一意の整数値を返します。この関数は、複合キーのコンパクト メモリ表示を作成する場合などに使用します。

> ***autonumberhash256** キーは、テーブルが読み込まれた順番で生成されるため、同じデータ ロードで生成された場合のみ結合できます。ソース データのソート処理から独立してデータ ロード間で恒久的に維持されるキーを使用する必要がある場合は、**hash128** 関数、**hash160** 関数、**hash256** 関数を使用する必要があります。*

構文:

```
autonumberhash256(expression {, expression})
```

複合キーの作成

この例では、メモリを保護するために、**autonumberhash256** 関数を使用して複合キーを作成します。この例は、デモのために、簡略化したものになっていますが、この方法が効果的なのは、多数の行が含まれるテーブルで使用した場合です。

テーブルの例

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

インライン データを使用して、ソース データをロードします。次に、Region 項目、Year 項目、Month 項目から複合キーを作成する、先行する LOAD を追加します。

```
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
];
```

この結果、テーブルは次のようになります。

結果テーブル

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

この例では、別のテーブルにリンクする必要がある場合、たとえば、文字列 "North2014May" の代わりに RYMkey の 1 を参照できます。

同様の方法で、Costs のソース テーブルをロードします。Region 項目、Year 項目、Month 項目は、**autonumberhash256** 関数を使用しテーブルをリンクしてすでに複合キーを作成しているため、合成キーの作成を避けるために先行する LOAD から除外されます。

```
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
];
```

テーブル ビジュアライゼーションをシートに追加し、Region 項目、Year 項目、Month 項目、および Sales と Costs の Sum メジャーを追加できるようになりました。テーブルは次のようになります。

結果テーブル

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

IterNo

このスクリプト関数は、**while** 節を含む **LOAD** ステートメントで、単一のレコードが評価された回数を示す整数を返します。最初の反復の値は 1 です。**IterNo** 関数は、**while** 節と共に使用される場合にのみ有効となります。

構文:

```
IterNo( )
```

例と結果:

```
LOAD
   IterNo() as Day,
   Date( StartDate + IterNo() - 1 ) as Date
   While StartDate + IterNo() - 1 <= EndDate;

LOAD * INLINE
[StartDate, EndDate
2014-01-22, 2014-01-26
];
```

この **LOAD** ステートメントは、**StartDate** と**EndDate** によって定義される範囲内で、日付ごとにレコードを1つ生成します。

この結果、テーブルは次のようになります。

結果テーブル

Day	Date
1	2014-01-22

Day	Date
2	2014-01-23
3	2014-01-24
4	2014-01-25
5	2014-01-26

RecNo

このスクリプト関数は、現在のテーブルで読み取られている行番号を整数で返します。最初のレコードの番号は 1 です。

構文:

```
RecNo( )
```

生成される Qlik Sense テーブルの行をカウントする **RowNo( )** とは対照的に、**RecNo( )** は、生データテーブルのレコードをカウントし、生データテーブルが別のテーブルと連結された場合はリセットされます。

データ ロード スクリプト

生データテーブルのロード:

```
Tab1:
LOAD * INLINE
[A, B
1, aa
2,cc
3,ee];

Tab2:
LOAD * INLINE
[C, D
5, xx
4,yy
6,zz];
```

選択した行のレコードと行番号のロード:

```
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;
```

結果の Qlik Sense 内部テーブル:

結果テーブル

A	B	RecNo( )	RowNo( )
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

RowNo

この関数は、結果として得られる Qlik Sense の内部テーブルの現在の行の位置を整数で返します。最初の行は 1 です。

構文:

```
RowNo( [TOTAL])
```

生データテーブルのレコード数をカウントする **RecNo( )** とは対照的に、**RowNo( )** 関数は、**where** 節で除外されたレコードはカウントせず、生データテーブルが別のテーブルに連結された場合でもリセットされません。

*先行する LOAD (同じテーブルから読み取りを行う、スタックされた複数の **LOAD** ステートメント) を使用する場合は、**RowNo( )** のみを最上部の **LOAD** ステートメントで使用できます。**RowNo( )** を後続の **LOAD** ステートメントで使用すると、0 が返されます。*

データ ロード スクリプト

生データテーブルのロード:

```
Tab1:
LOAD * INLINE
[A, B
1, aa
2,cc
3,ee];

Tab2:
LOAD * INLINE
[C, D
5, xx
4,yy
6,zz];
```

選択した行のレコードと行番号のロード:

```
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;
```

結果の Qlik Sense 内部テーブル:

結果テーブル

A	B	RecNo( )	RowNo( )
1	aa	1	1
3	ee	3	2
4	yy	2	3
6	zz	3	4

RowNo - チャート関数

RowNo() は、テーブルの現在の列セグメント内の現在行の数を返します。ビットマップチャートの場合、**RowNo()** はストレートテーブルに相当するセグメントに含まれる現在の行の数を返します。

テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除くすべての軸列の現在行と同じ値を持つ行だけが含まれます。

列セグメント

	Region	Country	Population	Rank(Population)
Column segment #1	Americas	Mexico	128,932,753	2
	Americas	Canada	37,742,154	3
	Americas	United States of America	331,002,651	1
Column segment #2	Europe	Sweden	10,099,265	4
	Europe	United Kingdom	67,886,011	2
	Europe	France	65,273,511	3
	Europe	Germany	83,783,942	1

チャートの式いずれかにこのチャート関数が使用されている場合、チャートのy値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

構文:

```
RowNo([TOTAL])
```

戻り値データ型: 整数

引数:

引数	説明
TOTAL	テーブルが1軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

例: RowNo を使用したチャートの数式

例 - チャートの数式

ロードスクリプト

以下のデータをインラインデータとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
Temp:
LOAD * inline [
Customer|Product|OrderNumber|UnitSales|UnitPrice
Astrida|AA|1|4|16
Astrida|AA|7|10|15
Astrida|BB|4|9|9
Betacab|CC|6|5|10
Betacab|AA|5|2|20
Betacab|BB|1|25| 25
Canutility|AA|3|8|15
Canutility|CC|5|4|19
Divadip|CC|2|4|16
Divadip|DD|3|1|25
] (delimiter is '|');
```

チャートの数式

Qlik Sense シートに **Customer** と**UnitSales** を軸としたテーブルのビジュアライゼーションを作成します。[**セグメント内の行**] および **Row Number** というラベルの付いたメジャーとして RowNo( ) および RowNo(TOTAL) をそれぞれ追加します。次の数式をメジャーとしてテーブルに追加します。

```
If( RowNo( )=1, 0, UnitSales / Above( UnitSales ))
```

結果

Customer	UnitSales	Row in Segment	Row Number	If(RowNo( )=1, 0, UnitSales / Above(UnitSales))
Astrida	4	1	1	0
Astrida	9	2	2	2.25
Astrida	10	3	3	1.1111111111111
Betacab	2	1	4	0
Betacab	5	2	5	2.5
Betacab	25	3	6	5
Canutility	4	1	7	0
Canutility	8	2	8	2
Divadip	1	1	9	0
Divadip	4	2	10	4

説明

Row in Segment 列には、顧客 Astrida の UnitSales の値が含まれている列セグメントの結果 1、2、3 が表示されます。行番号は、次の列セグメントBetacab でも再度 1 から始まります。

Row Number 列は RowNo() の TOTAL 引数のために軸を無視し、テーブルの行をカウントします。

この数式は、各列セグメントの 1 行目に 0 を返すため、列には次のように表示されます。

0、2.25、1.1111111、0、2.5、5、0、2、0、4。

参照先:

- *Above - チャート関数 (page 1252)*

8.7 日付および時刻関数

Qlik Sense の日付および時刻関数は、日付と時間の値を転送、変換するために使用されます。すべての関数は、データロードスクリプトおよびチャートの数式の両方で使用できます。

関数は、1899 年 12 月 30 日からの経過日数と等しい日時のシリアル値に基づいています。整数値は日付を表し、小数値はその日付の時刻を表します。

Qlik Sense はパラメータの数値を使用するため、日付や時刻として書式設定されていない場合でも、数値はパラメータとして有効です。パラメータが文字列の場合など、数値ではない場合、Qlik Sense は、日付と時刻の環境変数に従って、その文字列の解釈を試みます。

パラメータで使用されている時刻書式が、環境変数で設定されている書式に対応していない場合、Qlik Sense は正しく解釈することができません。この問題を解決するには、設定を変更するか、変換関数を使用します。

各関数の例は、日付と時刻のデフォルト書式である hh:mm:ss および YYYY-MM-DD (ISO 8601) を使用していると仮定して記載しています。

日付関数または時刻関数で日付と時刻を処理する場合、Qlik Sense は、日付関数または時刻関数に地理的位置が含まれている場合を除き、サマータイム時間のパラメータを無視します。

たとえば、`ConvertToLocalTime( filetime('Time.qvd'), 'Paris')` はサマータイム時間のパラメータを使用しますが、`ConvertToLocalTime(filetime('Time.qvd'), 'GMT-01:00')` はサマータイム時間のパラメータを使用しません。

日付と時刻の関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

時刻の整数式

second

この関数は、**expression** の小数部が標準的な数値の解釈に従って時間と判断される場合に、秒を表す整数を返します。

```
second (expression)
```

minute

この関数は、**expression** の小数部が標準的な数値の解釈に従って時間と判断される場合に、分を表す整数を返します。

```
minute (expression)
```

hour

この関数は、**expression** の小数部が標準的な数値の解釈に従って時間と判断される場合に、時間を表す整数を返します。

```
hour (expression)
```

day
この関数は、**expression** の小数部が標準的な数値の解釈に従って日付と判断される場合に、日付を表す整数を返します。

```
day (expression)
```

week
この関数は、ISO 8601 に従って、週番号を表す整数を返します。週番号は標準的な数値の解釈に従って、数式の日付の解釈により計算されます。

```
week (expression)
```

month
この関数は、環境変数 **MonthNames** および 1 から 12 までの整数で定義されている月名を持つデュアル値を返します。月は標準的な数値の解釈に従って、数式の日付の解釈により計算されます。

```
month (expression)
```

year
この関数は、**expression** が標準的な数値の解釈に従って日付と判断される場合に、年を表す整数を返します。

```
year (expression)
```

weekyear
この関数は、環境変数に基づいた週番号が含まれる年を返します。週番号の範囲は、1 からおよそ 52 となります。

```
weekyear (expression)
```

weekday
この関数は、以下を持つデュアル値を返します。

- 環境変数 **DayNames** で定義される日の名前。
- 曜日に相当する 0 から 6 までの整数。

```
weekday (date)
```

タイムスタンプ関数

now
この関数は、現在の時刻のタイムスタンプを返します。この関数は、**TimeStamp** システム変数形式の値を返します。既定の **timer_mode** 値は 1 です。

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

today
この関数は、現在の日付を返します。この関数は、`DateFormat` システム変数形式の値を返します。

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

LocalTime

この関数は、指定されたタイム ゾーンの現在の時刻のタイムスタンプを返します。

```
localtime ([timezone [, ignoreDST ]])
```

make 関数

makedate

この関数は、年 **YYYY**、月 **MM**、日 **DD** から算出された日付を返します。

```
makedate (YYYY [ , MM [ , DD ] ])
```

makeweekdate

この関数は、年 、週番号、曜日 から算出された日付を返します。

```
makeweekdate (YYYY [ , WW [ , D ] ])
```

maketime

この関数は、時間 **hh**、分 **mm**、秒 **ss** から算出された時間を返します。

```
maketime (hh [ , mm [ , ss [ .fff ] ] ])
```

その他の日付関数

AddMonths

この関数は、**startdate** 後の **n** か月後の日付、または **n** が負の場合には **startdate** の **n** か月前の日付を返します。

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

AddYears

この関数は、**startdate** の **n** 年の日付、または **n** が負の場合には **startdate** の **n** 年前の日付を返します。

```
addyears (startdate, n)
```

yeartodate

この関数は、入力したタイムスタンプがスクリプトが最後にロードされた日付の年に該当するかどうかを算出し、該当する場合は True を返し、該当しない場合は False を返します。

```
yeartodate (date [ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

timezone 関数

timezone

この関数を使うと、Qlik エンジンが実行されているコンピュータで定義された通りのタイム ゾーンが返されます。

```
timezone ( )
```

GMT

この関数は、現在の Greenwich Mean Time を返します。これは地域設定から導かれます。

```
GMT ( )
```

UTC
現在の Coordinated Universal Time を返します。

```
UTC ( )
```

daylightsaving
Windows の定義に基づき、現在の夏時間調整を返します。

```
daylightsaving ( )
```

converttolocaltime
UTC または GMT のタイムスタンプをデュアル値として現地時間に変換します。世界中の任意の都市、場所、タイム ゾーンを指定できます。

```
converttolocaltime (timestamp [, place [, ignore_dst=false]])
```

時刻設定関数

setdateyear
この関数は入力として **timestamp** と **year** を取得し、入力で指定された **year** で **timestamp** を更新します。

```
setdateyear  (timestamp, year)
```

setdateyearmonth
この関数は入力として **timestamp** と **month** 、**year** を取得し、入力で指定された **year** と **month** で **timestamp** を更新します。

```
setdateyearmonth  (timestamp, year, month)
```

in... 関数

inyear
この関数は、**timestamp** が **base_date** を含む年の範囲内にある場合、True を返します。

```
inyear (date, basedate , shift [, first_month_of_year = 1])
```

inyeartodate
この関数は、**timestamp** が **base_date** のミリ秒単位まで正確に **base_date** を含む年の範囲内にある場合、True を返します。

```
inyeartodate  (date, basedate , shift [, first_month_of_year = 1])
```

inquarter
この関数は、**timestamp** が **base_date** を含む四半期に含まれる場合、True を返します。

```
inquarter  (date, basedate , shift [, first_month_of_year = 1])
```

inquartertodate
この関数は、**timestamp** が **base_date** のミリ秒単位まで正確に **base_date** を含む四半期の範囲内にある場合、True を返します。

```
inquartertodate  (date, basedate , shift [, first_month_of_year = 1])
```

inmonth

この関数は、**timestamp** が **base_date** を含む月にある場合、True を返します。

```
inmonth (date, basedate , shift)
```

inmonthtodate

basedate の最後のミリ秒まで **basedate** を含む月に **date** がある場合に True を返します。

```
inmonthtodate  (date, basedate , shift)
```

inmonths

この関数は、タイムスタンプが基準日と同じ月、隔月、四半期、4 か月、または半年に該当するかどうかを確認します。タイムスタンプがその前後の期間に該当するか確認することもできます。

```
inmonths (n, date, basedate , shift [, first_month_of_year = 1])
```

inmonthstodate

この関数は、タイムスタンプが、base_date の最後のミリ秒までの月、2 か月、四半期、4 か月、半年のいずれかの期間の範囲内か確認します。タイムスタンプがその前後の期間に該当するか確認することもできます。

```
inmonthstodate  (n, date, basedate , shift [, first_month_of_year = 1])
```

inweek

この関数は、**timestamp** が **base_date** を含む週にある場合、True を返します。

```
inweek (date, basedate , shift [, weekstart])
```

inweektodate

この関数は、**timestamp** が **base_date** のミリ秒単位まで正確に **base_date** を含む週の範囲内にある場合、True を返します。

```
inweektodate  (date, basedate , shift [, weekstart])
```

inlunarweek

この関数は、**timestamp** が **base_date** を含む週周期の範囲内かどうかを判断します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて、各週は正確に 7 日構成となります。

```
inlunarweek  (date, basedate , shift [, weekstart])
```

inlunarweektodate

この関数は、**timestamp** が **base_date** の最後のミリ秒までの週周期の範囲内か確認します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて正確に 7 日構成となります。

```
inlunarweektodate  (date, basedate , shift [, weekstart])
```

inday

この関数は、**base_timestamp** を含む日に **timestamp** が含まれている場合、True を返します。

```
inday (timestamp, basetimestamp , shift [, daystart])
```

indaytotime
この関数は、**timestamp** が **base_timestamp** のミリ秒単位まで正確に **base_timestamp** を含む日の範囲内にある場合、True を返します。

```
indaytotime (timestamp, basetimestamp , shift [, daystart])
```

start ... end 関数

yearstart
この関数は、**date**を含む年の最初の日の開始に対応するタイムスタンプを返します。デフォルトの出力形式は、スクリプトに設定されている**DateFormat** です。

```
yearstart ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearend
この関数は、**date** を含む年の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている**DateFormat** です。

```
yearend ( date [, shift = 0 [, first_month_of_year = 1]])
```

yearname
この関数は、**date** を含む年の初日の最初のミリ秒のタイムスタンプに対応する数値を基底として、4 桁の年の表示値を返します。

```
yearname (date [, shift = 0 [, first_month_of_year = 1]] )
```

quarterstart
この関数は、**date** を含む四半期の最初のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている**DateFormat** です。

```
quarterstart (date [, shift = 0 [, first_month_of_year = 1]])
```

quarterend
この関数は、**date** を含む四半期の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている**DateFormat** です。

```
quarterend (date [, shift = 0 [, first_month_of_year = 1]])
```

quartername
この関数は、四半期の初日の最初のミリ秒のタイムスタンプに対応する値を基底として、四半期の月数 (**MonthNames** スクリプト変数に従った書式) および年の表示値を返します。

```
quartername (date [, shift = 0 [, first_month_of_year = 1]])
```

monthstart
この関数は、**date** を含む月の初日の最初のミリ秒のタイムスタンプに対応する値を返します。デフォルトの出力形式は、スクリプトに設定されている**DateFormat** です。

```
monthstart (date [, shift = 0])
```

monthend

この関数は、date を含む月の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている DateFormat です。

```
monthend (date [, shift = 0])
```

monthname

この関数は、月の初日の最初のミリ秒のタイムスタンプに対応する基底の数値を持つ、月 (**MonthNames** スクリプト変数に従った書式) および年の表示値を返します。

```
monthname (date [, shift = 0])
```

monthsstart

この関数は、ベース日付を含む月、2 か月、四半期、4 か月、半年のいずれかの期間の最初のミリ秒のタイムスタンプに相当する値を返します。その前後の期間のタイムスタンプを取得することもできます。既定の出力形式は、スクリプトに設定されている **DateFormat** です。

```
monthsstart (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsend

この関数は、ベース日付を含む月、2 か月、四半期、4 か月、半年のいずれかの期間の最後のミリ秒のタイムスタンプに相当する値を返します。その前後の期間のタイムスタンプを取得することもできます。

```
monthsend (n, date [, shift = 0 [, first_month_of_year = 1]])
```

monthsname

この関数は、期間の月の範囲 (**MonthNames** スクリプト変数に従った書式で表示) および年を表す表示値を返します。基底値は、ベース日付を含む月、2 か月、四半期、4 か月、半年のいずれかの期間の最初のミリ秒のタイムスタンプに相当する値です。

```
monthsname (n, date [, shift = 0 [, first_month_of_year = 1]])
```

weekstart

この関数は、**date** を含む暦週の初日の最初のミリ秒のタイムスタンプに対応する値を返します。デフォルトの出力形式は、スクリプトに設定されている **DateFormat** です。

```
weekstart (date [, shift = 0 [,weekoffset = 0]])
```

weekend

この関数は、**date** を含む暦週の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている **DateFormat** です。

```
weekend (date [, shift = 0 [,weekoffset = 0]])
```

weekname

この関数は、**date** を含む週の初日の最初のミリ秒のタイムスタンプに対応する数値を基底として、年と週番号を表示する値を返します。

```
weekname (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekstart
この関数は、**date** を含む週周期の初日の最初のミリ秒のタイムスタンプに相当する値を返します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて正確に 7 日構成となります。

```
lunarweekstart (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekend
この関数は、**date** を含む週周期の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて正確に 7 日構成となります。

```
lunarweekend (date [, shift = 0 [,weekoffset = 0]])
```

lunarweekname
この関数は、**date** を含む週周期の初日の最初のミリ秒のタイムスタンプに対応する年と週周期番号を表示する表示値を返します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて正確に 7 日構成となります。

```
lunarweekname (date [, shift = 0 [,weekoffset = 0]])
```

daystart
この関数は、**time** 引数に含まれる日の最初のミリ秒で、タイムスタンプに対応する値を返します。デフォルトの出力形式は、スクリプトに設定されている **TimestampFormat** です。

```
daystart (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayend
この関数は、**time** を含む日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている **TimestampFormat** です。

```
dayend (timestamp [, shift = 0 [, dayoffset = 0]])
```

dayname
この関数は、**time** を含む日の最初のミリ秒のタイムスタンプに対応する数値を基底として、日付を表示する値を返します。

```
dayname (timestamp [, shift = 0 [, dayoffset = 0]])
```

日付連番関数

age
age 関数は、**date_of_birth** に生まれた人の **timestamp** 時点での年齢 (満年齢) を返します。

```
age (timestamp, date_of_birth)
```

networkdays
networkdays 関数は、オプションで指定された **holiday** を考慮した上で、**start_date** と **end_date** の間の当日を含む作業日数 (月 ~ 金曜日) を返します。

```
networkdays  (start:date, end_date {, holiday})
```

firstworkdate

firstworkdate 関数は、**end_date** までに **no_of_workdays** (月 ~ 金曜日) の日数に達するように、オプションで指定された休日を考慮した最遅開始日を返します。**end_date** および **holiday** は有効な日付またはタイムスタンプでなければなりません。

```
firstworkdate (end_date, no_of_workdays {, holiday} )
```

lastworkdate

lastworkdate 関数は、オプションで指定された **holiday** を考慮した上で、**start_date** に開始した場合に **no_of_workdays** (月 ~ 金曜日) の日数に達する最早終了日を返します。**start_date** と**holiday** は、有効な日付またはタイムスタンプでなければなりません。

```
lastworkdate (start_date, no_of_workdays {, holiday})
```

daynumberofyear

この関数は、タイムスタンプの年の日番号を計算します。計算は、年の初日の最初のミリ秒から行われますが、最初の月を補正することもできます。

```
daynumberofyear (date[,firstmonth])
```

daynumberofquarter

この関数は、タイムスタンプの四半期の日番号を計算します。この機能はマスター カレンダーを作成するときに使用します。

```
daynumberofquarter (date[,firstmonth])
```

addmonths

この関数は、**startdate** 後の **n** か月後の日付、または**n** が負の場合には **startdate** の **n** か月前の日付を返します。

構文:

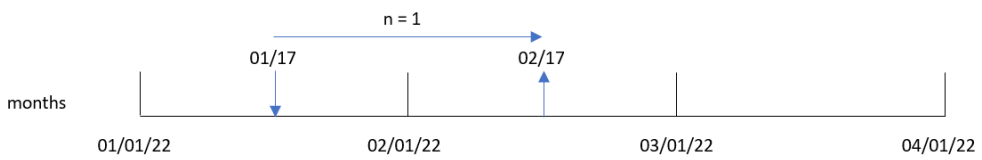
```
AddMonths(startdate, n , [ , mode])
```

戻り値データ型: dual

addmonths() 関数は、startdate に対して定義された月数 n を加算または減算し、結果の日付を返します。

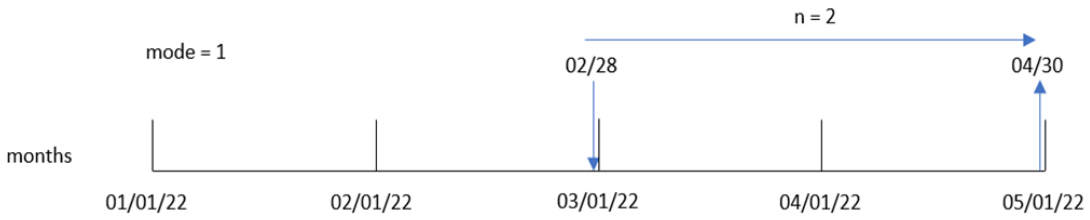
mode 引数は、月の 28 日以降の startdate 値に影響します。mode 引数を 1 に設定することにより、addmonths() 関数は月末までの相対距離が startdate と等しい日付を返します。

addmonths() 関数の図の例



例えば、2 月 28 日は月の最終日です。mode が 1 である addmonths() 関数が 2 か月後の日付を返すのに使用される場合、関数は 4 月の月末日である 4 月 30 日を返します。

mode=1 がある addmonths() 関数の例の図



引数

引数	説明
startdate	タイムスタンプの開始日。例: '2012-10-12'
n	正または負の整数の月数。
mode	該当する月が、その月の始めを基準として追加されるのか、または終わりを基準として追加されるのかを指定します。既定のモードは 0 で、月の始めを基準として追加されます。月の終わりを基準として追加する場合は、モードを 1 に設定します。モードが 1 に設定されていて、入力日付が 28 日以降の場合、関数では開始日付の月の終わりに到達するまでの残りの日数を確認します。返される日付では、月の終わりに達するまでと同じ日数が設定されます。

使用に適しているケース

addmonths() 関数は、特定期間の所定の月数前後の日付を見つけるための式に一般的に使用されます。

例えば、addmonths() 関数は携帯電話契約の終了日付を特定するのに使用できます。

関数の例

例	結果
addmonths ('01/29/2003' ,3)	「04/29/2003」を返します
addmonths ('01/29/2003',3,0)	「04/29/2003」を返します
addmonths ('01/29/2003',3,1)	「04/28/2003」を返します
addmonths ('01/29/2003',1,0)	「02/28/2003」を返します
addmonths ('01/29/2003',1,1)	「02/26/2003」を返します
addmonths ('02/28/2003',1,0)	「03/28/2003」を返します
addmonths ('02/28/2003',1,1)	「03/31/2003」を返します
addmonths ('01/29/2003',-3)	「10/29/2002」を返します

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクション発生の 2 か月後の日付を返す項目 [two_months_later] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        addmonths(date,2) as two_months_later
        ;
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
```

```
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- two_months_later

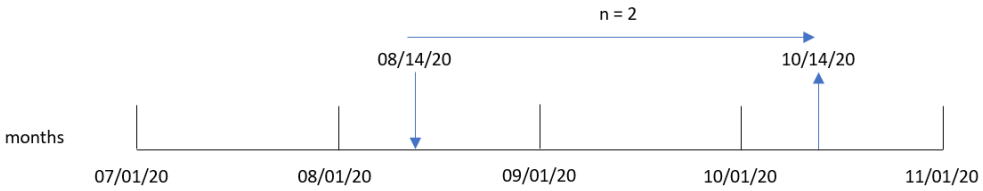
結果テーブル

日付	two_months_later
01/10/2020	03/10/2020
02/28/2020	04/28/2020
04/09/2020	06/09/2020
04/16/2020	06/16/2020
05/21/2020	07/21/2020
08/14/2020	10/14/2020
10/07/2020	12/07/2020
12/05/2020	02/05/2021
01/22/2021	03/22/2021
02/03/2021	04/03/2021
03/17/2021	05/17/2021
04/23/2021	06/23/2021
05/04/2021	07/04/2021
06/30/2021	08/30/2021
07/26/2021	09/26/2021
12/27/2021	02/27/2022
02/02/2022	04/02/2022

日付	two_months_later
02/26/2022	04/26/2022
03/07/2022	05/07/2022
03/11/2022	05/11/2022

[two_months_later] 項目は、addmonths() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は、評価される日付を識別します。2 番目の引数は、startdate に対して加算または減算する月数です。この場合、値 2 が入力されます。

addmonths() 関数の図、追加の引数がない例



トランザクション 8193 は 8 月 14 日に発生しました。そのため、addmonths() 関数は [two_months_later] 項目に対して 2020 年 10 月 14 日を返します。

例 2 – 相対的月末

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連の月末トランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクション発生の 2 か月前の相対的月末日付を返す項目 [relative_two_months_prior] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        addmonths(date,-2,1) as relative_two_months_prior
        ;
Load
*
Inline
```

```
[
id,date,amount
8188,'01/28/2022',37.23
8189,'01/31/2022',57.54
8190,'02/28/2022',17.17
8191,'04/29/2022',88.27
8192,'04/30/2022',57.42
8193,'05/31/2022',53.80
8194,'08/14/2022',82.06
8195,'10/07/2022',40.39
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

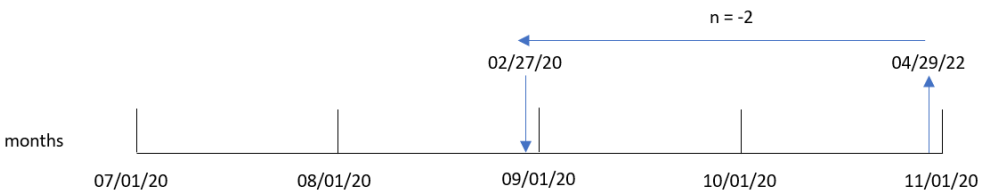
- date
- relative_two_months_prior

結果テーブル

日付	relative_two_months_prior
01/28/2022	11/27/2021
01/31/2022	11/30/2021
02/28/2022	12/31/2021
04/29/2022	02/27/2022
04/30/2022	02/28/2022
05/31/2022	03/31/2022
08/14/2022	06/14/2022
10/07/2022	08/07/2022

[relative_two_months_prior] 項目は、addmonths() 関数を使用して、先行 Load ステートメントで作成されます。提供される最初の引数は、評価される日付を識別します。2 番目の引数は、startdate に対して加算または減算する月数です。この場合、値 -2 が入力されます。最後の引数 (値は 1) はモードで、28 以上のすべての日付について、この関数が相対的月末の日付を計算するように強制します。

n=-2 の例がある addmonths() 関数の図



トランザクション8191は2022年4月29日に発生します。当初、2か月前なら2月に設定されていました。次に、関数の第3引数でモードを1に、日付の値を27日よりも後に設定し、相対的月末の値を計算します。この関数は、29日が4月の月末日から2日早い日付であることを識別し、2月の月末日から2日早い日付である27日を返します。

例 3 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクション発生の2か月後の日付を返す計算は、チャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

次のメジャーを作成します:

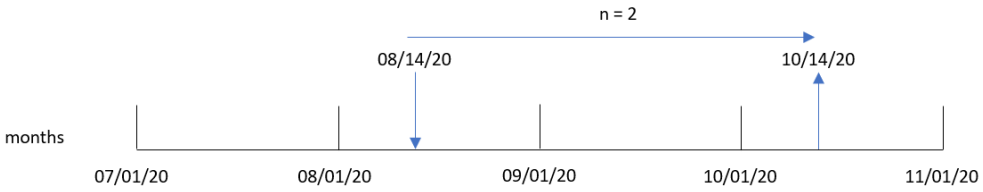
```
=addmonths(date,2)
```

結果テーブル

日付	=addmonths(date,2)
01/10/2020	03/10/2020
02/28/2020	04/28/2020
04/09/2020	06/09/2020
04/16/2020	06/16/2020
05/21/2020	07/21/2020
08/14/2020	10/14/2020
10/07/2020	12/07/2020
12/05/2020	02/05/2021
01/22/2021	03/22/2021
02/03/2021	04/03/2021
03/17/2021	05/17/2021
04/23/2021	06/23/2021
05/04/2021	07/04/2021
06/30/2021	08/30/2021
07/26/2021	09/26/2021
12/27/2021	02/27/2022
02/02/2022	04/02/2022
02/26/2022	04/26/2022
03/07/2022	05/07/2022
03/11/2022	05/11/2022

two_months_later メジャーは、addmonths() 関数を使用してチャートオブジェクトに作成されます。提供される最初の引数は、評価される日付を識別します。2 番目の引数は、startdate に対して加算または減算する月数です。この場合、値 2 が入力されます。

addmonths() 関数の図、チャートオブジェクトの例



トランザクション 8193 は 8 月 14 日に発生しました。そのため、addmonths() 関数は [two_months_later] 項目に対して 2020 年 10 月 14 日を返します。

例 4 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Mobile_Plans」というテーブルにロードされるデータセット。
- 契約 ID、開始日付、契約期間、月額料金の情報。

エンドユーザーは、契約 ID ごとに各電話契約の終了日を表示するチャートオブジェクトを希望しています。

ロードスクリプト

```
Mobile_Plans:
Load
*
Inline
[
contract_id,start_date,contract_length,monthly_fee
8188,'01/13/2020',18,37.23
8189,'02/26/2020',24,17.17
8190,'03/27/2020',36,88.27
8191,'04/16/2020',24,57.42
8192,'05/21/2020',24,53.80
8193,'08/14/2020',12,82.06
8194,'10/07/2020',18,40.39
8195,'12/05/2020',12,87.21
8196,'01/22/2021',12,95.93
8197,'02/03/2021',18,45.89
8198,'03/17/2021',24,36.23
8199,'04/23/2021',24,25.66
8200,'05/04/2021',12,82.77
8201,'06/30/2021',12,69.98
8202,'07/26/2021',12,76.11
8203,'12/27/2021',36,25.12
8204,'06/06/2022',24,46.23
```

```
8205,'07/18/2022',12,84.21
8206,'11/14/2022',12,96.24
8207,'12/12/2022',18,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- contract_id
- start_date
- contract_length

次のメジャーを作成して、各契約の終了日付を計算します。

```
=addmonths(start_date,contract_length, 0)
```

結果テーブル

contract_id	start_date	contract_length	=addmonths(start_date,contract_length,0)
8188	01/13/2020	18	07/13/2021
8189	02/26/2020	24	02/26/2022
8190	03/27/2020	36	03/27/2023
8191	04/16/2020	24	04/16/2022
8192	05/21/2020	24	05/21/2022
8193	08/14/2020	12	08/14/2021
8194	10/07/2020	18	04/07/2022
8195	12/05/2020	12	12/05/2021
8196	01/22/2021	12	01/22/2022
8197	02/03/2021	18	08/03/2022
8198	03/17/2021	24	03/17/2023
8199	04/23/2021	24	04/23/2023
8200	05/04/2021	12	05/04/2022
8201	06/30/2021	12	06/30/2022
8202	07/26/2021	12	07/26/2022
8203	12/27/2021	36	12/27/2024
8204	06/06/2022	24	06/06/2024
8205	07/18/2022	12	07/18/2023
8206	11/14/2022	12	11/14/2023
8207	12/12/2022	18	06/12/2024

addyears

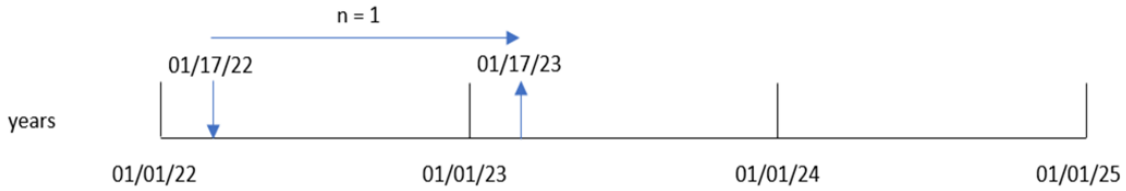
この関数は、**startdate** の **n** 年の日付、または**n** が負の場合には **startdate** の **n** 年前の日付を返します。

構文:

```
AddYears(startdate, n)
```

戻り値データ型: dual

addyears() 関数の図の例



addyears() 関数は、startdate に対して定義された年数 n を加算または減算します。次に結果の日付を返します。

引数

引数	説明
startdate	タイムスタンプの開始日。例: '2012-10-12'
n	正または負の整数の年数。

関数の例

例	結果
addyears ('01/29/2010',3)	「01/29/2013」を返します
addyears ('01/29/2010',-1)	「01/29/2009」を返します

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 簡単な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクション発生の 2 年後の日付を返す項目 [two_years_later] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        addyears(date,2) as two_years_later
        ;
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

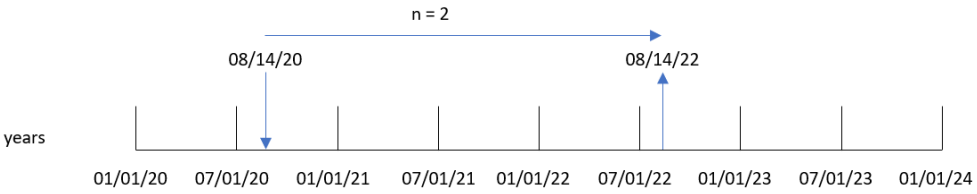
- date
- two_years_later

結果テーブル

日付	two_years_later
01/10/2020	01/10/2022
02/28/2020	02/28/2022
04/09/2020	04/09/2022
04/16/2020	04/16/2022
05/21/2020	05/21/2022
08/14/2020	08/14/2022
10/07/2020	10/07/2022
12/05/2020	12/05/2022
01/22/2021	01/22/2023
02/03/2021	02/03/2023
03/17/2021	03/17/2023
04/23/2021	04/23/2023
05/04/2021	05/04/2023
06/30/2021	06/30/2023
07/26/2021	07/26/2023
12/27/2021	12/27/2023
02/02/2022	02/02/2024
02/26/2022	02/26/2024
03/07/2022	03/07/2024
03/11/2022	03/11/2024

[two_years_later] 項目は、addyears() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は、評価される日付を識別します。2 番目の引数は、開始日付に対して加算または減算する年数です。この場合、値 2 が入力されます。

addyears() 関数の図、基本的な例



トランザクション 8193 は 2020 年 8 月 14 日に発生しました。そのため、addyears() 関数は [two_years_later] 項目に対して 2022 年 8 月 14 日を返します。

例 2 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。

チャート オブジェクトで、トランザクション発生時から1年前の日付を返すメジャー prior_year_date を作成します。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/10/2020',37.23
8189,'02/28/2020',17.17
8190,'04/09/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
```

```
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'02/02/2022',46.23
8205,'02/26/2022',84.21
8206,'03/07/2022',96.24
8207,'03/11/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

次のメジャーを作成して、各トランザクションから1年前の日付を計算します。

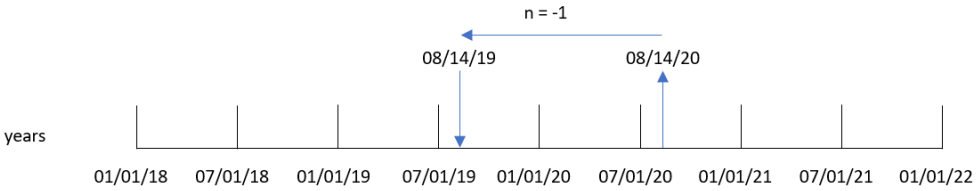
```
=addyears(date,-1)
```

結果テーブル

日付	=addyears(date,-1)
01/10/2020	01/10/2019
02/28/2020	02/28/2019
04/09/2020	04/09/2019
04/16/2020	04/16/2019
05/21/2020	05/21/2019
08/14/2020	08/14/2019
10/07/2020	10/07/2019
12/05/2020	12/05/2019
01/22/2021	01/22/2020
02/03/2021	02/03/2020
03/17/2021	03/17/2020
04/23/2021	04/23/2020
05/04/2021	05/04/2020
06/30/2021	06/30/2020
07/26/2021	07/26/2020
12/27/2021	12/27/2020
02/02/2022	02/02/2021
02/26/2022	02/26/2021
03/07/2022	03/07/2021
03/11/2022	03/11/2021

one_year_prior メジャーは、addyears() 関数を使用してチャートオブジェクトに作成されます。提供される最初の引数は、評価される日付を識別します。2 番目の引数は、 startdate に対して加算または減算する年数です。この場合、値 -1 が入力されます。

addyears() 関数の図、チャートオブジェクトの例



トランザクション 8193 は 8 月 14 日に発生しました。そのため、addyears() 関数は [one_year_prior] 項目に対して 2019 年 8 月 14 日を返します。

例 3 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Warranties」というテーブルにロードされるデータセット。
- 製品 ID、購入日付、保証期間、購入価格の情報。

エンドユーザーは、製品 ID ごとに各製品の保証終了日を表示するチャートオブジェクトを希望しています。

ロードスクリプト

```
Warranties:
Load
*
Inline
[
product_id,purchase_date,warranty_length,purchase_price
8188,'01/13/2020',4,32000
8189,'02/26/2020',2,28000
8190,'03/27/2020',3,41000
8191,'04/16/2020',4,17000
8192,'05/21/2020',2,25000
8193,'08/14/2020',1,59000
8194,'10/07/2020',2,12000
8195,'12/05/2020',3,12000
8196,'01/22/2021',4,24000
8197,'02/03/2021',1,50000
8198,'03/17/2021',2,80000
8199,'04/23/2021',3,10000
8200,'05/04/2021',4,30000
```

```
8201,'06/30/2021',3,30000
8202,'07/26/2021',4,20000
8203,'12/27/2021',4,10000
8204,'06/06/2022',2,25000
8205,'07/18/2022',1,32000
8206,'11/14/2022',1,30000
8207,'12/12/2022',4,22000
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- product_id
- purchase_date
- warranty_length

次のメジャーを作成して、各保証期間の終了日付を計算します。

```
=addyears(purchase_date,warranty_length)
```

結果テーブル

product_id	purchase_date	warranty_length	=addyears(purchase_date,warranty_length)
8188	01/13/2020	4	01/13/2024
8189	02/26/2020	2	02/26/2022
8190	03/27/2020	3	03/27/2023
8191	04/16/2020	4	04/16/2024
8192	05/21/2020	2	05/21/2022
8193	08/14/2020	1	08/14/2021
8194	10/07/2020	2	10/07/2022
8195	12/05/2020	3	12/05/2023
8196	01/22/2021	4	01/22/2025
8197	02/03/2021	1	02/03/2022
8198	03/17/2021	2	03/17/2023
8199	04/23/2021	3	04/23/2024
8200	05/04/2021	4	05/04/2025
8201	06/30/2021	3	06/30/2024
8202	07/26/2021	4	07/26/2025
8203	12/27/2021	4	12/27/2025

product_id	purchase_ date	warranty_ length	=addyears(purchase_date,warranty_ length)
8204	06/06/2022	2	06/06/2024
8205	07/18/2022	1	07/18/2023
8206	11/14/2022	1	11/14/2023
8207	12/12/2022	4	12/12/2026

age

age 関数は、**date_of_birth** に生まれた人の **timestamp** 時点での年齢 (満年齢) を返します。

構文:

```
age(timestamp, date_of_birth)
```

数式に使用できます。

戻り値データ型: 数値

引数:

引数

引数	説明
timestamp	満年齢を計算するためのタイムスタンプまたは計算結果がタイムスタンプになる数式。
date_of_birth	年齢を計算する人の生年月日。数式に使用できます。

例と結果:
これらの例は、日付書式 **DD/MM/YYYY** を使用しています。日付書式は、データロードスクリプト上部の **SET DateFormat** ステートメントで指定されています。必要に応じて、書式を変更してください。

スクリプトの例

例	結果
age('25/01/2014', '29/10/2012')	1 を返します。
age('29/10/2014', '29/10/2012')	2 を返します。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Employees:
LOAD * INLINE [
Member|DateOfBirth
John|28/03/1989
Linda|10/12/1990
```

```
Steve|5/2/1992
Birg|31/3/1993
Raj|19/5/1994
Prita|15/9/1994
Su|11/12/1994
Goran|2/3/1995
Sunny|14/5/1996
Ajoa|13/6/1996
Daphne|7/7/1998
Biffy|4/8/2000
] (delimiter is |);
AgeTable:
Load *,
age('20/08/2015', DateOfBirth) As Age
Resident Employees;
Drop table Employees;
```

結果テーブルには、テーブルの各レコードに対する age の戻り値が表示されます。

結果テーブル

Member	DateOfBirth	Age
John	28/03/1989	26
Linda	10/12/1990	24
Steve	5/2/1992	23
Birg	31/3/1993	22
Raj	19/5/1994	21
Prita	15/9/1994	20
Su	11/12/1994	20
Goran	2/3/1995	20
Sunny	14/5/1996	19
Ajoa	13/6/1996	19
Daphne	7/7/1998	17
Biffy	4/8/2000	15

converttolocaltime

UTC または GMT のタイムスタンプをデュアル値として現地時間に変換します。世界中の任意の都市、場所、タイム ゾーンを指定できます。

構文:

```
ConvertToLocalTime(timestamp [, place [, ignore_dst=false]])
```

戻り値データ型： dual

引数

引数	説明
timestamp	変換するタイムスタンプまたは計算結果がタイムスタンプになる数式。
place	下記の有効な場所とタイム ゾーンの表に示された場所またはタイム ゾーン。あるいは、GMT または UTC を使用して現地時間を定義できます。次の値とタイム オフセットの範囲が有効です。 • GMT • GMT-12:00 - GMT-01:00 • GMT+01:00 - GMT+14:00 • UTC • UTC-12:00 - UTC-01:00 • UTC+01:00 - UTC+14:00 *DST オフセットを使用する場合 (つまり、False を評価する **ignore_dst** 引数値を指定する場合)、**place** 引数に GMT オフセットではなく場所を指定する必要があります。これは、夏時間に合わせて調整するには、GMT オフセットによって提供される経度情報に加えて、緯度情報も必要となるためです。詳しくは「GMT オフセットを DST と組み合わせて使用する (page 613)」を参照してください。* *標準タイム オフセットのみ使用できます。GMT-04:27 など、任意のタイム オフセットを使用することはできません。*
ignore_dst	この引数が True と評価される場合、DST (夏時間) は無視されます。True と評価される有効な引数値には、-1 や True() などがあります。 この引数が False と評価された場合、タイムスタンプは夏時間に合わせて調整されます。False と評価される有効な引数値には、0 や False() などがあります。 **ignore_dst** 引数値が無効な場合、関数は **ignore_dst** 値が True と評価されるかのように数式を評価します。**ignore_dst** 引数値が指定されていない場合、関数は **ignore_dst** 値が False と評価されるかのように数式を評価します。

有効な場所とタイム ゾーン

A-C	D-K	L-R	S-Z
Abu Dhabi	Darwin	La Paz	Samoa
Adelaide	Dhaka	Lima	Santiago

A-C	D-K	L-R	S-Z
Alaska	Eastern Time (US & Canada)	Lisbon	Sapporo
Amsterdam	Edinburgh	Ljubljana	Sarajevo
Arizona	Ekaterinburg	London	Saskatchewan
Astana	Fiji	Madrid	Seoul
Athens	Georgetown	Magadan	Singapore
Atlantic Time (Canada)	Greenland	Mazatlan	Skopje
Auckland	Greenwich Mean Time : Dublin	Melbourne	Sofia
Azores	Guadalajara	Mexico City	Solomon Is.
Baghdad	Guam	Mid-Atlantic	Sri Jayawardenepura
Baku	Hanoi	Minsk	St. Petersburg
Bangkok	Harare	Monrovia	Stockholm
Beijing	Hawaii	Monterrey	Sydney
Belgrade	Helsinki	Moscow	Taipei
Berlin	Hobart	Mountain Time (US & Canada)	Tallinn
Bern	Hong Kong	Mumbai	Tashkent
Bogota	Indiana (East)	Muscat	Tbilisi
Brasilia	International Date Line West	Nairobi	Tehran
Bratislava	Irkutsk	New Caledonia	Tokyo
Brisbane	Islamabad	New Delhi	Urumqi
Brussels	Istanbul	Newfoundland	Warsaw
Bucharest	Jakarta	Novosibirsk	Wellington
Budapest	Jerusalem	Nuku'alofa	West Central Africa
Buenos Aires	Kabul	Osaka	Vienna
Cairo	Kamchatka	Pacific Time (US & Canada)	Vilnius
Canberra	Karachi	Paris	Vladivostok
Cape Verde Is.	Kathmandu	Perth	Volgograd

A-C	D-K	L-R	S-Z
Caracas	Kolkata	Port Moresby	Yakutsk
Casablanca	Krasnoyarsk	Prague	Yerevan
Central America	Kuala Lumpur	Pretoria	Zagreb
Central Time (US & Canada)	Kuwait	Quito	-
Chennai	Kyiv	Riga	-
Chihuahua	-	Riyadh	-
Chongqing	-	Rome	-
Copenhagen	-	-	-

例と結果:

スクリプトの例

例	結果
`ConvertToLocalTime('2023-08-14 08:39:47','Paris')`	「2023-08-14 10:39:47」と、対応するタイムスタンプの内部表現を返します。
`ConvertToLocalTime(UTC(), 'Stockholm')`	夏時間の調整をして、ストックホルムの時間を返します。
`ConvertToLocalTime(UTC(), 'Stockholm', -1)`	夏時間の調整なしで、ストックホルムの時間を返します。
`ConvertToLocalTime(UTC(), 'GMT-05:00')`	北米東海岸 (ニューヨークなど) の時刻を返します。場所ではなくGMT オフセットが指定されているため、夏時間の調整は行われません。
`ConvertToLocalTime(UTC(), 'New York', -1)`	夏時間の調整なしで、北米東海岸 (ニューヨーク) の時刻を返します。
`ConvertToLocalTime(UTC(), 'New York', True())`	夏時間の調整なしで、北米東海岸 (ニューヨーク) の時刻を返します。
`ConvertToLocalTime(UTC(), 'New York', 0)`	夏時間の調整をして、北米東海岸 (ニューヨーク) の時刻を返します。
`ConvertToLocalTime(UTC(), 'New York', False())`	夏時間の調整をして、北米東海岸 (ニューヨーク) の時刻を返します。

GMT オフセットを DST と組み合わせて使用する

Qlik Sense で International Components for Unicode (ICU) ライブラリを実装した後、GMT (グリニッジ標準時) オフセットを DST (夏時間) と組み合わせて使用するには、追加の緯度情報が必要になります。

GMT は経度 (東西) オフセットであるのに対し、DST は緯度 (南北) オフセットです。例えば、ヘルシンキ (フィンランド) とヨハネスブルグ (南アフリカ) は同じ GMT+02:00 オフセットを共有しますが、同じ DST オフセットは共有しません。つまり、現地の DST 条件に関する完全な情報を得るために、GMT オフセットに加えて、DST オフセットでも現地のタイム ゾーンの緯度位置に関する情報 (地理的タイム ゾーン入力) が必要となります。

day

この関数は、**expression** の小数部が標準的な数値の解釈に従って日付と判断される場合に、日付を表す整数を返します。

この関数は、特定の日付の日を返します。これは通常、カレンダーの軸の一部として日付項目を算出するために使用します。

構文:

```
day(expression)
```

戻り値データ型: 整数

関数の例

例	結果
day(1971-10-12)	12 を返します
day(35648)	6 を返します (35648 = 1997-08-06 のため)

例 1 – DateFormat データセット (スクリプト)

ロード スクリプトと結果

概要

データ ロード エディター を開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています:

- Master_Calendar という名前の日付のデータセット。DateFormat システム変数は、DD/MM/YYYY に設定されています。
- day() 関数を使用して day_of_month という名前の追加項目を作成する、先行する LOAD。
- date() 関数を使用して完全な月名を表示する、long_date という名前の追加項目。

ロード スクリプト

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

Master_Calendar:
Load
        date,
        date(date,'dd-MMMM-YYYY') as long_date,
        day(date) as day_of_month
Inline
```

```
[
date
03/11/2022
03/12/2022
03/13/2022
03/14/2022
03/15/2022
03/16/2022
03/17/2022
03/18/2022
03/19/2022
03/20/2022
03/21/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- long_date
- day_of_month

結果テーブル

日付	long_date	day_of_month
03/11/2022	11-March- 2022	11
03/12/2022	12-March- 2022	12
03/13/2022	13-March- 2022	13
03/14/2022	14-March- 2022	14
03/15/2022	15-March- 2022	15
03/16/2022	16-March- 2022	16
03/17/2022	17-March- 2022	17
03/18/2022	18-March- 2022	18
03/19/2022	19-March- 2022	19
03/20/2022	20-March- 2022	20
03/21/2022	21-March- 2022	21

該当月の日付は、スクリプトの day() 関数により正常に評価されています。

例 2 – ANSI 日付 (スクリプト)

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- Master_Calendar という名前の日付のデータセット。DateFormat システム変数 DD/MM/YYYY が使用されます。ただし、データセットに含まれる日付は、ANSI 標準日付形式です。
- date() 関数を使用して、day_of_month という名前の追加項目を作成する先行するロード。
- date() 関数を使用して日付を完全な月名で表示する、long_date という名前の追加項目。

ロードスクリプト

```
SET DateFormat='DD/MM/YYYY';
Master_Calendar:
Load
       date,
       date(date,'dd-MMMM-YYYY') as long_date,
       day(date) as day_of_month

Inline
[
date
2022-03-11
2022-03-12
2022-03-13
2022-03-14
2022-03-15
2022-03-16
2022-03-17
2022-03-18
2022-03-19
2022-03-20
2022-03-21
 ];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- long_date
- day_of_month

結果テーブル

日付	long_date	day_of_month
03/11/2022	11-March- 2022	11
03/12/2022	12-March- 2022	12
03/13/2022	13-March- 2022	13
03/14/2022	14-March- 2022	14
03/15/2022	15-March- 2022	15
03/16/2022	16-March- 2022	16
03/17/2022	17-March- 2022	17
03/18/2022	18-March- 2022	18
03/19/2022	19-March- 2022	19
03/20/2022	20-March- 2022	20
03/21/2022	21-March- 2022	21

該当月の日付は、スクリプトの day() 関数により正常に評価されています。

例 3 – 形式設定のない日付 (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- Master_Calendar という名前の日付のデータセット。 DateFormat システム変数 DD/MM/YYYY が使用されます。
- day() 関数を使用して、day_of_month という名前の追加項目を作成する先行するロード。
- unformatted_date という名前の、形式設定がない元の日付。
- date() を使用して数字表記の日付を書式設定された日付項目に変換する、long_date という名前の追加項目。

ロードスクリプト

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

Master_Calendar:
Load
      unformatted_date,
      date(unformatted_date,'dd-MMMM-YYYY') as long_date,
      day(date) as day_of_month
```

```
Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- unformatted_date
- long_date
- day_of_month

結果テーブル

unformatted_date	long_date	day_of_month
44868	03-November- 2022	3
44898	03-December- 2022	3
44928	02-January- 2023	2
44958	01-February- 2023	1
44988	03-March- 2023	3
45008	23-March- 2023	23
45018	02-April- 2023	2
45038	22-April- 2023	22
45048	02-May- 2023	2
45068	22-May- 2023	22
45078	01-June- 2023	1

該当月の日付は、スクリプトの day() 関数により正常に評価されています。

例 4 – 失効月の計算 (チャート)

ロードスクリプトとチャートの数式

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 3 月に注文があった Orders という名前のデータセット。テーブルには 3 項目が含まれています。
  - ID
  - order_date
  - amount

ロードスクリプト

```
Orders:
Load
        id,
        order_date,
        amount
Inline
[
id,order_date,amount
1,03/01/2022,231.24
2,03/02/2022,567.28
3,03/03/2022,364.28
4,03/04/2022,575.76
5,03/05/2022,638.68
6,03/06/2022,785.38
7,03/07/2022,967.46
8,03/08/2022,287.67
9,03/09/2022,764.45
10,03/10/2022,875.43
11,03/11/2022,957.35
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:order_date。

納品日を計算するには、メジャー =day(order_date+5) を作成します。

結果テーブル

order_date	=day(order_date+5)
03/11/2022	16

order_date	=day(order_date+5)
03/12/2022	17
03/13/2022	18
03/14/2022	19
03/15/2022	20
03/16/2022	21
03/17/2022	22
03/18/2022	23
03/19/2022	24
03/20/2022	25
03/21/2022	26

day() 関数は、5 日間という配達期間に基づき、3 月 11 日にあった注文は 16 日に配達されると正しく確定します。

dayend

この関数は、**time** を含む日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている **TimestampFormat** です。

構文:

```
DayEnd(time[, [period_no[, day_start]])
```

使用に適しているケース

dayend() 関数は、ユーザーがまだ発生していない日の端数を計算に使用する場合に、数式の一部として一般的に使用されます。例えば、日中にまだ発生する総費用を計算します。

戻り値データ型: dual

引数

引数	説明
time	評価するタイムスタンプ。
period_no	**period_no** は整数または計算結果が整数になる数式で、値 0 は **time** を含む日を示します。**period_no** の値が負の場合は過去の日を、正の場合は将来の日を示します。
day_start	1 日の開始時刻を深夜 0 時以外に設定する場合は、**day_start** に 1 日未満の長さを補正値として指定します。例えば、0.125 は午前 3 時を意味します。 つまり、オフセットを作るには、開始時刻を 24 時間で割り算してください。たとえば、1 日が午前 7:00 に始まる場合は、分数 7/24 を使用します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
dayend('01/25/2013 16:45:00')	01/25/2013 23:59:59 を返します。PM
dayend('01/25/2013 16:45:00', -1)	01/24/2013 23:59:59 を返します。PM
dayend('01/25/2013 16:45:00', 0, 0.5)	01/26/2013 11:59:59 を返します。PM

例 1 - 基本的なスクリプト

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 日付のリストを含むデータセットは、「Calendar」という名前のテーブルにロードされます。
- 既定の DateFormat システム変数 (MM/DD/YYYY)。
- dayend() 関数を使用して、追加の項目「EOD_timestamp」を作成するための先行する LOAD。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Calendar:
    Load
      date,
      dayend(date) as EOD_timestamp
      ;
Load
date
Inline
[
date
03/11/2022 1:47:15 AM
```

```
03/12/2022 4:34:58 AM
03/13/2022 5:15:55 AM
03/14/2022 9:25:14 AM
03/15/2022 10:06:54 AM
03/16/2022 10:44:42 AM
03/17/2022 11:33:30 AM
03/18/2022 12:58:14 PM
03/19/2022 4:23:12 PM
03/20/2022 6:42:15 PM
03/21/2022 7:41:16 PM
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- EOD_timestamp

結果テーブル

日付	EOD_timestamp
03/11/2022 1:47:15 AM	3/11/2022 11:59:59 PM
03/12/2022 4:34:58 AM	3/12/2022 11:59:59 PM
03/13/2022 5:15:55 AM	3/13/2022 11:59:59 PM
03/14/2022 9:25:14 AM	3/14/2022 11:59:59 PM
03/15/2022 10:06:54 AM	3/15/2022 11:59:59 PM
03/16/2022 10:44:42 AM	3/16/2022 11:59:59 PM
03/17/2022 11:33:30 AM	3/17/2022 11:59:59 PM
03/18/2022 12:58:14 PM	3/18/2022 11:59:59 PM
03/19/2022 4:23:12 PM	3/19/2022 11:59:59 PM
03/20/2022 6:42:15 PM	3/20/2022 11:59:59 PM
03/21/2022 7:41:16 PM	3/21/2022 11:59:59 PM

上のテーブルからわかるように、データセット内の日付ごとに1日の終わりのタイムスタンプが生成されます。タイムスタンプはシステム変数 `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT` の形式です。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

サービス予約を含むデータセットを「Services」という名前のテーブルにロードします。

データセットには次の項目が含まれています。

- service_id
- service_date
- amount

テーブルに 2 つの新しい項目を作成します。

- deposit_due_date: デポジットを受け取るべき日付。これは、service_date の 3 日前の 1 日の終わりです。
- final_payment_due_date: 最終的な支払いを受け取るべき日付。これは、service_date の 7 日後の 1 日の終わりです。

上記の 2 つの項目は、dayend() 関数を使用して先行するロードで作成され、最初の2つのパラメータtime と period_no を提供します。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Services:
   Load
     *,
     dayend(service_date,-3) as deposit_due_date,
     dayend(service_date,7) as final_payment_due_date
     ;
Load
service_id,
service_date,
amount
Inline
[
service_id, service_date,amount
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- service_date
- deposit_due_date
- final_payment_due_date

結果テーブル

service_date	deposit_due_date	final_payment_due_date
03/11/2022 9:25:14 AM	3/8/2022 11:59:59 PM	3/18/2022 11:59:59 PM
03/12/2022 10:06:54 AM	3/9/2022 11:59:59 PM	3/19/2022 11:59:59 PM
03/13/2022 10:44:42 AM	3/10/2022 11:59:59 PM	3/20/2022 11:59:59 PM
03/14/2022 11:33:30 AM	3/11/2022 11:59:59 PM	3/21/2022 11:59:59 PM
03/15/2022 12:58:14 PM	3/12/2022 11:59:59 PM	3/22/2022 11:59:59 PM
03/16/2022 4:23:12 PM	3/13/2022 11:59:59 PM	3/23/2022 11:59:59 PM
03/17/2022 6:42:15 PM	3/14/2022 11:59:59 PM	3/24/2022 11:59:59 PM
03/18/2022 7:41:16 PM	3/15/2022 11:59:59 PM	3/25/2022 11:59:59 PM
03/19/2022 8:14:15 PM	3/16/2022 11:59:59 PM	3/26/2022 11:59:59 PM
03/20/2022 9:23:51 PM	3/17/2022 11:59:59 PM	3/27/2022 11:59:59 PM
03/21/2022 10:04:41 PM	3/18/2022 11:59:59 PM	3/28/2022 11:59:59 PM

新しい項目の値は TimestampFormat M/D/YYYY h:mm:ss[.fff] TT にあります。関数 dayend() が使用されたため、タイムスタンプ値はすべてその日の最後のミリ秒です。

dayend() 関数で渡された 2 番目の引数が負であるため、デポジットの期日の値はサービス日の 3 日前です。

dayend() 関数で渡された 2 番目の引数が正であるため、最終的な支払期日の値はサービス日の 7 日後です。

例 3 – day_start script

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

この例で使用されているデータセットとシナリオは、前の例と同じです。

前の例のように、2つの新しい項目を作成します。

- deposit_due_date: デポジットを受け取るべき日付。これは、service_date の 3 日前の 1 日の終わりです。
- final_payment_due_date: 最終的な支払いを受け取るべき日付。これは、service_date の 7 日後の 1 日の終わりです。

ただし、あなたの会社は、営業日が 5 PM に始まり、翌日の 5 PM に終わるというポリシーの下で運営したいと考えています。これで、会社はそれらの営業時間内に発生するトランザクションを監視できます。

これらの要件を達成するために、dayend() 関数を使用して先行するロードで作成され、3 つの引数 time、period_no、day_start をすべて使用します。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Services:
   Load
     *,
     dayend(service_date,-3,17/24) as deposit_due_date,
     dayend(service_date,7,17/24) as final_payment_due_date
     ;
Load
service_id,
service_date,
amount
Inline
[
service_id, service_date,amount
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- service_date
- deposit_due_date
- final_payment_due_date

結果テーブル

service_date	deposit_due_date	final_payment_due_date
03/11/2022 9:25:14 AM	3/8/2022 4:59:59 PM	3/18/2022 4:59:59 PM
03/12/2022 10:06:54 AM	3/9/2022 4:59:59 PM	3/19/2022 4:59:59 PM

service_date	deposit_due_date	final_payment_due_date
03/13/2022 10:44:42 AM	3/10/2022 4:59:59 PM	3/20/2022 4:59:59 PM
03/14/2022 11:33:30 AM	3/11/2022 4:59:59 PM	3/21/2022 4:59:59 PM
03/15/2022 12:58:14 PM	3/12/2022 4:59:59 PM	3/22/2022 4:59:59 PM
03/16/2022 4:23:12 PM	3/13/2022 4:59:59 PM	3/23/2022 4:59:59 PM
03/17/2022 6:42:15 PM	3/14/2022 4:59:59 PM	3/24/2022 4:59:59 PM
03/18/2022 7:41:16 PM	3/15/2022 4:59:59 PM	3/25/2022 4:59:59 PM
03/19/2022 8:14:15 PM	3/16/2022 4:59:59 PM	3/26/2022 4:59:59 PM
03/20/2022 9:23:51 PM	3/17/2022 4:59:59 PM	3/27/2022 4:59:59 PM
03/21/2022 10:04:41 PM	3/18/2022 4:59:59 PM	3/28/2022 4:59:59 PM

日付は例 2 と同じままですが、dayend() 関数に渡された 3 番目の引数 day_start の値が 17/24 であるため、日付のタイムスタンプは 5:00 PM より前の最後のミリ秒になりました。

例 4 – チャートの例

ロード スクリプトとチャートの数式

概要

データ ロード エディター を開き、以下のロード スクリプトを新しいタブに追加します。

この例で使用されているデータセットとシナリオは、前の 2 つの例と同じです。あなたの会社は、営業日が 5:00 PM に始まり、翌日の 5:00 PM に終わるというポリシーの下で運営したいと考えています。

前の例のように、2つの新しい項目を作成します。

- deposit_due_date: デポジットを受け取るべき日付。これは、service_date の 3 日前の 1 日の終わりです。
- final_payment_due_date: 最終的な支払いを受け取るべき日付。これは、service_date の 7 日後の 1 日の終わりです。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Services:
Load
service_id,
service_date,
amount
Inline
[
service_id, service_date,amount
1,03/11/2022 9:25:14 AM,231.24
2,03/12/2022 10:06:54 AM,567.28
```

```
3,03/13/2022 10:44:42 AM,364.28
4,03/14/2022 11:33:30 AM,575.76
5,03/15/2022 12:58:14 PM,638.68
6,03/16/2022 4:23:12 PM,785.38
7,03/17/2022 6:42:15 PM,967.46
8,03/18/2022 7:41:16 PM,287.67
9,03/19/2022 8:14:15 PM,764.45
10,03/20/2022 9:23:51 PM,875.43
11,03/21/2022 10:04:41 PM,957.35
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

service_date をクリックします。

deposit_due_date 項目を作成するには、このメジャーを作成します。

=dayend(service_date,-3,17/24)。

次に、final_payment_due_date 項目を作成するには、次のメジャーを作成します:

=dayend(service_date,7,17/24)。

結果テーブル

service_date	=dayend(service_date,-3,17/24)	=dayend(service_date,7,17/24
03/11/2022	3/8/2022 16:59:59 PM	3/18/2022 16:59:59 PM
03/12/2022	3/9/2022 16:59:59 PM	3/19/2022 16:59:59 PM
03/13/2022	3/10/2022 16:59:59 PM	3/20/2022 16:59:59 PM
03/14/2022	3/11/2022 16:59:59 PM	3/21/2022 16:59:59 PM
03/15/2022	3/12/2022 16:59:59 PM	3/22/2022 16:59:59 PM
03/16/2022	3/13/2022 16:59:59 PM	3/23/2022 16:59:59 PM
03/17/2022	3/14/2022 16:59:59 PM	3/24/2022 16:59:59 PM
03/18/2022	3/15/2022 16:59:59 PM	3/25/2022 16:59:59 PM
03/19/2022	3/16/2022 16:59:59 PM	3/26/2022 16:59:59 PM
03/20/2022	3/17/2022 16:59:59 PM	3/27/2022 16:59:59 PM
03/21/2022	3/18/2022 16:59:59 PM	3/28/2022 16:59:59 PM

新しい項目の値は TimestampFormat M/D/YYYY h:mm:ss[.fff] TT にあります。関数 dayend() が使用されたため、タイムスタンプ値はすべてその日の最後のミリ秒です。

dayend() 関数で渡された 2 番目の引数が負であるため、支払期日の値はサービス日の 3 日前です。

dayend() 関数で渡された 2 番目の引数が正であるため、最終的な支払期日の値はサービス日の 7 日後です。

dayend() 関数に渡された 3 番目の引数 day_start の値が 17/24 であるため、日付のタイムスタンプは 5:00 PM より前の最後のミリ秒です。

引数

引数	説明
time	評価するタイムスタンプ。
period_no	**period_no** は整数または計算結果が整数になる数式で、値 0 は **time** を含む日を示します。**period_no** の値が負の場合は過去の日を、正の場合は将来の日を示します。
day_start	1 日の開始時刻を深夜 0 時以外に設定する場合は、**day_start** に 1 日未満の長さを補正値として指定します。例えば、0.125 は午前 3 時を意味します。

daylightsaving

Windows の定義に基づき、現在の夏時間調整を返します。

構文:

```
DaylightSaving( )
```

戻り値データ型: dual

```
daylightsaving( )
```

dayname

この関数は、**time** を含む日の最初のミリ秒のタイムスタンプに対応する数値を基底として、日付を表示する値を返します。

構文:

```
DayName(time[, period_no [, day_start]])
```

戻り値データ型: dual

引数:

引数

引数	説明
time	評価するタイムスタンプ。
period_no	**period_no** は整数または計算結果が整数になる数式で、値 0 は **time** を含む日を示します。**period_no** の値が負の場合は過去の日を、正の場合は将来の日を示します。
day_start	1 日の開始時刻を深夜 0 時以外に設定する場合は、**day_start** に 1 日未満の長さを補正値として指定します。例えば、0.125 は午前 3 時を意味します。

例と結果:
これらの例は、日付書式 **DD/MM/YYYY** を使用しています。日付書式は、データロードスクリプト上部の **SET DateFormat** ステートメントで指定されています。必要に応じて、書式を変更してください。

スクリプトの例

例	結果
dayname('25/01/2013 16:45:00')	25/01/2013 を返します。
dayname('25/01/2013 16:45:00', -1)	24/01/2013 を返します。
dayname('25/01/2013 16:45:00', 0, 0.5)	25/01/2013 を返します。 タイムスタンプ全体を表示すると、25/01/2013 12:00:00.000. に相当する元の値が表示されます。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

この例では、日の名前は、テーブルの各請求書日付の翌日の開始時刻を示すタイムスタンプから作成されます。

```
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;
```

結果テーブルには、元の日付と、dayname() 関数の戻り値の列が含まれています。プロパティパネルで書式を指定すると、タイムスタンプ全体を表示できます。

結果テーブル

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

この関数は、タイムスタンプの四半期の日番号を計算します。この機能はマスター カレンダーを作成するときに使用します。

構文:

```
DayNumberOfQuarter(timestamp[,start_month])
```

戻り値データ型: integer

引数

引数	説明
timestamp	評価する日付またはタイムスタンプ。
start_month	**start_month** を 2 から12 の間で指定することで (省略した場合は 1)、年の開始時点を任意の月の初日に移動することができます。例えば、会計年度を3 月 1 日から開始する場合には、**start_month** = 3 と指定します。

これらの例は、日付書式 **DD/MM/YYYY** を使用しています。日付書式は、データロードスクリプト上部の **SET DateFormat** ステートメントで指定されています。必要に応じて、書式を変更してください。

関数の例

例	結果
`DayNumberOfQuarter('12/09/2014')`	現四半期の日番号、74 を返します。
`DayNumberOfQuarter ('12/09/2014',3)`	現四半期の日番号、12 を返します。 この場合、第 1 四半期は 3 月から始まります (start_monthに 3 が指定されているため)。これは、現四半期が第 3 四半期で、9 月 1 日に始まったことを意味します。

例 1 – 年の開始が1月 (スクリプト)

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- `Calendar` という名前のテーブルにロードされる日付のリストを含む単純なデータセット。既定の `DateFormat` システム変数 MM/DD/YYYY が使用されます。
- `DayNumberOfQuarter()` 関数を使用して `DayNrQtr` という名前の追加項目を作成する、先行する LOAD。

日付を除いて、関数に追加のパラメータは提供されません。

ロードスクリプト

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

Calendar:
Load
        date,
        DayNumberOfQuarter(date) as DayNrQtr
```

```
        ;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- daynrqtr

結果テーブル

日付	daynrqtr
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
02/28/2022	59
03/01/2022	61
03/31/2022	91
04/01/2022	1

DayNumberOfQuarter() 関数に 2 番目の引数が渡されなかったため、年の最初の日は 1 月 1 日です

1 月 1 日は四半期の 1 日目であり、2 月 1 日は四半期の 32 日目です。3 月 31 日は四半期の 91 日で最終日であり、4 月 1 日は第 2 四半期の 1 日目です。

例 2 – 年の開始が 2 月 (スクリプト)

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 最初の例と同じデータセット。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- 2 月 1 日から始まる start_month 引数。これにより、会計年度が 2 月 1 日に設定されます。

ロードスクリプト

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

Calendar:
Load
       date,
       DayNumberOfQuarter(date,2) as DayNrQtr
       ;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- daynrqtr

結果テーブル

日付	daynrqtr
01/01/2022	62

日付	daynrqtr
01/10/2022	71
01/31/2022	92
02/01/2022	1
02/10/2022	10
02/28/2022	28
03/01/2022	30
03/31/2022	60
04/01/2022	61

DayNumberOfQuarter() 関数に渡された 2 番目の引数が 2 であったため、年の最初の日は 2 月 1 日です。

今年の第 1 四半期は 2 月から 4 月まで、第 4 四半期は 11 月から 1 月までです。これは結果テーブルに示されています。2 月 1 日は四半期の 1 日目であり、1 月 31 日は四半期の 92 日目で最後の日です。

例 3 – 年の開始が 1 月 (チャート)

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 最初の例と同じデータセット。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。四半期の日付の値は、チャートオブジェクトのメジャーを介して計算されます。

ロードスクリプト

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

Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
```

```
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: date。

次のメジャーを作成します:

```
=daynumberofquarter(date)
```

結果テーブル

日付	=daynumberofquarter(date)
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
02/28/2022	59
03/01/2022	61
03/31/2022	91
04/01/2022	1

DayNumberOfQuarter() 関数に 2 番目の引数が渡されなかったため、年の最初の日は 1 月 1 日です

1 月 1 日は四半期の 1 日目であり、2 月 1 日は四半期の 32 日目です。3 月 31 日は四半期の 91 日で最終日であり、4 月 1 日は第 2 四半期の 1 日目です。

例 4 – 年の開始が 2 月 (チャート)

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 最初の例と同じデータセット。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- 会計年度は 2 月 1 日から1月 31 日までです。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。四半期の日付の値は、チャート オブジェクトのメジャーを介して計算されます。

ロード スクリプト

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

Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
02/28/2022
03/01/2022
03/31/2022
04/01/2022
];
```

チャート オブジェクト

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `date`。

次のメジャーを作成します:

```
=daynumberofquarter(date,2)
```

結果

結果テーブル

日付	=daynumberofquarter(date,2)
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	1
02/10/2022	10
02/28/2022	28
03/01/2022	30
03/31/2022	60
04/01/2022	61

`DayNumberOfQuarter()` 関数に渡された 2 番目の引数が 2 であったため、年の最初の日は 1 月 1 日です。

今年の第 1 四半期は 2 月から4 月まで、第 4 四半期は 11 月から1 月までです。これは、2 月 1 日が四半期の1 日目であり、1 月 31 日が四半期の 92 日目で最後の日である結果テーブルで証明されています。

daynumberofyear

この関数は、タイムスタンプの年の日番号を計算します。計算は、年の初日の最初のミリ秒から行われますが、最初の月を補正することもできます。

構文:

```
DayNumberOfYear(timestamp[,start_month])
```

戻り値データ型: integer

引数

引数	説明
timestamp	評価する日付またはタイムスタンプ。
start_month	**start_month** を 2 から12 の間で指定することで (省略した場合は 1)、年の開始時点を任意の月の初日に移動することができます。例えば、会計年度を3 月 1 日から開始する場合には、**start_month** = 3 と指定します。

これらの例は、日付書式 **DD/MM/YYYY** を使用しています。日付書式は、データロードスクリプト上部の **SET DateFormat** ステートメントで指定されています。必要に応じて、書式を変更してください。

関数の例

例	結果
`DayNumberOfYear( '12/09/2014' )`	年の初めからカウントした日番号である256 を返します。
`DayNumberOfYear( '12/09/2014',3 )`	3 月 1 日からカウントした日番号である196 を返します。

例 1 – 年の開始が 1 月 (スクリプト)

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- `Calendar` という名前のテーブルにロードされる日付のリストを含む単純なデータセット。既定の `DateFormat` システム変数 MM/DD/YYYY が使用されます。
- `DayNumberOfYear()` 関数を使用して `daynryear` という名前の追加項目を作成する、先行するLOAD。

日付を除いて、関数に追加のパラメータは提供されません。

ロード スクリプト

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

Calendar:
Load
       date,
       DayNumberOfYear(date) as daynryear
       ;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- daynryear

結果テーブル

日付	daynryear
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
06/30/2022	182
07/26/2022	208
10/31/2022	305
11/01/2022	306
12/31/2022	366

DayNumberOfYear() 関数に 2 番目の引数が渡されなかったため、年の最初の日は 1 月 1 日です。

1 月 1 日は四半期の 1 日目であり、2 月 1 日は年の 32 日目です。6 月 30 日はその年の 182 日目であり、12 月 31 日はその年の 366 日目で最終日です。

例 2 – 年の開始が 11 月 (スクリプト)

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 最初の例と同じデータセット。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます
- 11 月 1 日から始まる start_month 引数。これにより、会計年度が 11 月 1 日に設定されます。

ロードスクリプト

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

Calendar:
Load
       date,
       DayNumberOfYear(date,11) as daynryear
       ;
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- daynryear

結果テーブル

日付	daynryear
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	93
02/10/2022	102
06/30/2022	243
07/26/2022	269
10/31/2022	366
11/01/2022	1
12/31/2022	61

DayNumberOfYear() 関数に渡された 2 番目の引数が 11 であったため、年の最初の日は 11 月 1 日です。

1 月 1 日は四半期の 1 日目であり、2 月 1 日は年の 32 日目です。6 月 30 日はその年の 182 日目であり、12 月 31 日はその年の 366 日目で最終日です。

例 3 – 年の開始が 1 月 (チャート)

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 最初の例と同じデータセット。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。四半期の日付の値は、チャートオブジェクトのメジャーを介して計算されます。

ロードスクリプト

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

Calendar:
Load
date
Inline
[
date
01/01/2022
```

```
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: date。

次のメジャーを作成します:

```
=daynumberofyear(date)
```

結果テーブル

日付	=daynumberofyear(date)
01/01/2022	1
01/10/2022	10
01/31/2022	31
02/01/2022	32
02/10/2022	41
06/30/2022	182
07/26/2022	208
10/31/2022	305
11/01/2022	306
12/31/2022	366

DayNumberOfYear() 関数に 2 番目の引数が渡されなかったため、年の最初の日は 1 月 1 日です

1 月 1 日は年の 1 日目であり、2 月 1 日は年の 32 日目です。6 月 30 日はその年の 182 日目であり、12 月 31 日はその年の 366 日目で最終日です。

例 4 – 年の開始が 11 月 (チャート)

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 最初の例と同じデータセット。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- 会計年度は 11 月 1 日から10 月 31 日までです。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。年の日付の値は、チャートオブジェクトのメジャーを介して計算されます。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
Calendar:
Load
date
Inline
[
date
01/01/2022
01/10/2022
01/31/2022
02/01/2022
02/10/2022
06/30/2022
07/26/2022
10/31/2022
11/01/2022
12/31/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: date。

次のメジャーを作成します:

```
=daynumberofyear(date)
```

結果テーブル

日付	=daynumberofyear(date,11)
01/01/2022	62
01/10/2022	71
01/31/2022	92
02/01/2022	93
02/10/2022	102
06/30/2022	243
07/26/2022	269
10/31/2022	366

日付	=daynumberofyear(date,11)
11/01/2022	1
12/31/2022	61

DayNumberOfYear() 関数に渡された 2 番目の引数が 11 であったため、年の最初の日は 11 月 1 日です。

会計年度は 11 月から 10 月までです。これは結果テーブルに示されています。11 月 1 日は年の 1 日目であり、10 月 31 日は年の 366 日目で最後の日です。

daystart

この関数は、**time** 引数に含まれる日の最初のミリ秒で、タイムスタンプに対応する値を返します。デフォルトの出力形式は、スクリプトに設定されている **TimestampFormat** です。

構文:

```
DayStart(time[, [period_no[, day_start]])
```

戻り値データ型: dual

引数

引数	説明
time	評価するタイムスタンプ。
period_no	**period_no** は整数または計算結果が整数になる数式で、値 0 は **time** を含む日を示します。**period_no** の値が負の場合は過去の日を、正の場合は将来の日を示します。
day_start	1 日の開始時刻を深夜 0 時以外に設定する場合は、**day_start** に 1 日未満の長さを補正値として指定します。例えば、0.125 は午前 3 時を意味します。 つまり、オフセットを作るには、開始時刻を 24 時間で割り算してください。たとえば、1 日が午前 7:00 に始まる場合は、分数 7/24 を使用します。

使用に適しているケース

daystart() 関数は、ユーザーがこれまで経過した日の端数を計算に使用する場合に、数式の一部として一般的に使用されます。例えば、その日これまでに従業員が稼いだ合計賃金の計算に使用できます。

これらの例は、タイムスタンプ形式 'M/D/YYYY h:mm:ss[.fff] TT' を使用しています。タイムスタンプ形式は、データロードスクリプト上部の SET TimeStamp ステートメントで指定されています。必要に応じて、書式を変更してください。

関数の例

例	結果
daystart('01/25/2013 4:45:00 PM')	1/25/2013 12:00:00 AM を返します。
daystart('1/25/2013 4:45:00 PM', -1)	1/24/2013 12:00:00 AM を返します。
daystart('1/25/2013 16:45:00',0,0.5)	1/25/2013 12:00:00 PM を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 簡単な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Calendar という名前のテーブルにロードされる日付のリストを含む単純なデータセット。
- 既定の TimeStampFormat システム変数 (M/D/YYYY h:mm:ss[.fff] TT) が使用されます。
- daystart() 関数を使用して、SOD_timestamp という名前の追加項目を作成する先行する LOAD。

日付を除いて、関数に追加のパラメータは提供されません。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Calendar:
    Load
        date,
        daystart(date) as SOD_timestamp
        ;
Load
date
Inline
[
date
03/11/2022 1:47:15 AM
03/12/2022 4:34:58 AM
03/13/2022 5:15:55 AM
03/14/2022 9:25:14 AM
03/15/2022 10:06:54 AM
03/16/2022 10:44:42 AM
03/17/2022 11:33:30 AM
```

```
03/18/2022 12:58:14 PM
03/19/2022 4:23:12 PM
03/20/2022 6:42:15 PM
03/21/2022 7:41:16 PM
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- SOD_timestamp

結果テーブル

日付	SOD_timestamp
03/11/2022 1:47:15 AM	3/11/2022 12:00:00 AM
03/12/2022 4:34:58 AM	3/12/2022 12:00:00 AM
03/13/2022 5:15:55 AM	3/13/2022 12:00:00 AM
03/14/2022 9:25:14 AM	3/14/2022 12:00:00 AM
03/15/2022 10:06:54 AM	3/15/2022 12:00:00 AM
03/16/2022 10:44:42 AM	3/16/2022 12:00:00 AM
03/17/2022 11:33:30 AM	3/17/2022 12:00:00 AM
03/18/2022 12:58:14 PM	3/18/2022 12:00:00 AM
03/19/2022 4:23:12 PM	3/19/2022 12:00:00 AM
03/20/2022 6:42:15 PM	3/20/2022 12:00:00 AM
03/21/2022 7:41:16 PM	3/21/2022 12:00:00 AM

As can be seen in the table above, the end of day timestamp is generated for each date in our dataset. タイムスタンプはシステム変数 `TimestampFormat M/D/YYYY h:mm:ss[.fff] TT` の形式です。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- `Fines` という名前のテーブルにロードされる駐車違反料金を含むデータセット。データセットには次の項目が含まれています。

        - id
        - due_date
        - number_plate
        - amount
    - daystart() 関数を使った先行 Load および 3 つのパラメータの供給: time、period_no、および day_start。この先行 Load は、次の 2 つの新しい日付項目を作成します。
        - [early_repayment_period] 日付項目、支払期日から7 日前から開始。
        - [late_penalty_period] 日付項目、支払期日から14 日後から開始。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Fines:
    Load
        *,
        daystart(due_date,-7) as early_repayment_period,
        daystart(due_date,14) as late_penalty_period
        ;
Load
*
Inline
[
id, due_date, number_plate,amount
1,02/11/2022, 573RJG,50.00
2,03/25/2022, SC41854,50.00
3,04/14/2022, 8EHZ378,50.00
4,06/28/2022, 8HSS198,50.00
5,08/15/2022, 1221665,50.00
6,11/16/2022, EAK473,50.00
7,01/17/2023, KD6822,50.00
8,03/22/2023, 1GGLB,50.00
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- due_date
- early_repayment_period
- late_penalty_period

結果テーブル

due_date	early_repayment_period	late_penalty_period
02/11/2022 9:25:14 AM	2/4/2022 12:00:00 AM	2/25/2022 12:00:00 AM
03/25/2022 10:06:54 AM	3/18/2022 12:00:00 AM	4/8/2022 12:00:00 AM

due_date	early_repayment_period	late_penalty_period
04/14/2022 10:44:42 AM	4/7/2022 12:00:00 AM	4/28/2022 12:00:00 AM
06/28/2022 11:33:30 AM	6/21/2022 12:00:00 AM	7/12/2022 12:00:00 AM
08/15/2022 12:58:14 PM	8/8/2022 12:00:00 AM	8/29/2022 12:00:00 AM
11/16/2022 4:23:12 PM	11/9/2022 12:00:00 AM	11/30/2022 12:00:00 AM
01/17/2023 6:42:15 PM	1/10/2023 12:00:00 AM	1/31/2023 12:00:00 AM
03/22/2023 7:41:16 PM	3/15/2023 12:00:00 AM	4/5/2023 12:00:00 AM

新しい項目の値は TimestampFormat M/DD/YYYY tt にあります。関数 daystart() が使用されたため、タイムスタンプ値はすべてその日の最初のミリ秒です。

daystart() 関数で渡される2番目の引数が負であるため、早期再支払期間の値は期日の7日前です。

daystart() 関数で渡される2番目の引数が負であるため、遅延再支払期間の値は期日の14日後です。

例 3 – day_start

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 前の例と同じデータセットとシナリオ。
- 前の例と同じ先行 Load 。

この例では、業務日が毎日 7:00 AM に開始/終了するように設定されています。

ロードスクリプト

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

Fines:
    Load
        *,
        daystart(due_date,-7,7/24) as early_repayment_period,
        daystart(due_date,14, 7/24) as late_penalty_period
        ;
Load
*
Inline
[
id, due_date, number_plate,amount
1,02/11/2022, 573RJG,50.00
2,03/25/2022, SC41854,50.00
3,04/14/2022, 8EHZ378,50.00
```

```
4,06/28/2022, 8HSS198,50.00
5,08/15/2022, 1221665,50.00
6,11/16/2022, EAK473,50.00
7,01/17/2023, KD6822,50.00
8,03/22/2023, 1GGLB,50.00
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- due_date
- early_repayment_period
- late_penalty_period

結果テーブル

due_date	early_repayment_period	late_penalty_period
02/11/2022	2/3/2022 7:00:00 AM	2/24/2022 7:00:00 AM
03/25/2022	3/17/2022 7:00:00 AM	4/7/2022 7:00:00 AM
04/14/2022	4/6/2022 7:00:00 AM	4/27/2022 7:00:00 AM
06/28/2022	6/20/2022 7:00:00 AM	7/11/2022 7:00:00 AM
08/15/2022	8/7/2022 7:00:00 AM	8/28/2022 7:00:00 AM
11/16/2022	11/8/2022 7:00:00 AM	11/29/2022 7:00:00 AM
01/17/2023	1/9/2023 7:00:00 AM	1/30/2023 7:00:00 AM
03/22/2023	3/14/2023 7:00:00 AM	4/4/2023 7:00:00 AM

daystart() 関数に渡された引数 day_start の値が 7/24 であったため、日付のタイムスタンプは 7:00 AMとなります。これにより、日の始めが 7:00 AM に設定されます。

[due_date] 項目にタイムスタンプがないため、12:00 AM として処理されますが、これは日付が 7:00 AM で開始/終了するため前の日の一部のままです。そのため、2 月 11 日が期日の違反料金の早期再支払期間は 2 月 3 日の 7:00 AM に始まります。

例 4 – チャートオブジェクトの例

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

子の例では、前の例と同じデータセットとシナリオを使用しています。

ただし、元の Fines テーブルは、2 つの追加期日値がチャートオブジェクトで計算され、アプリケーションにロードされます。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Fines:
    Load
*
Inline
[
id, due_date, numer_plate,amount
1,02/11/2022 9:25:14 AM, 573RJG,50.00
2,03/25/2022 10:06:54 AM, SC41854,50.00
3,04/14/2022 10:44:42 AM, 8EHZ378,50.00
4,06/28/2022 11:33:30 AM, 8HSS198,50.00
5,08/15/2022 12:58:14 PM, 1221665,50.00
6,11/16/2022 4:23:12 PM, EAK473,50.00
7,01/17/2023 6:42:15 PM, KD6822,50.00
8,03/22/2023 7:41:16 PM, 1GGLB,50.00
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:due_date。
2. [early_repayment_period] 項目を作成するには、次のメジャーを作成します。
   =daystart(due_date,-7,7/24)
3. [late_penalty_period] 項目を作成するには、次のメジャーを作成します:
   =daystart(due_date,14,7/24)

結果テーブル

due_date	=daystart(due_date,-7,7/24)	=daystart(due_date,14,7/24)
02/11/2022 9:25:14 AM	2/4/2022 7:00:00 AM	2/25/2022 7:00:00 AM
03/25/2022 10:06:54 AM	3/18/2022 7:00:00 AM	4/8/2022 7:00:00 AM
04/14/2022 10:44:42 AM	4/7/2022 7:00:00 AM	4/28/2022 7:00:00 AM
06/28/2022 11:33:30 AM	6/21/2022 7:00:00 AM	7/12/2022 7:00:00 AM
08/15/2022 12:58:14 PM	8/8/2022 7:00:00 AM	8/29/2022 7:00:00 AM
11/16/2022 4:23:12 PM	11/9/2022 7:00:00 AM	11/30/2022 7:00:00 AM
01/17/2023 6:42:15 PM	1/10/2023 7:00:00 AM	1/31/2023 7:00:00 AM
03/22/2023 7:41:16 PM	3/15/2023 7:00:00 AM	4/5/2023 7:00:00 AM

新しい項目の値は TimestampFormat M/D/YYYY h:mm:ss[.fff] TT にあります。daystart() 関数が使用されたため、日付と時刻の値はすべてその日の最初のミリ秒に対応します。

daystart() 関数で渡される 2 番目の引数が負であるため、早期再支払期間の値は期日の 7 日前です。

daystart() 関数で渡される2番目の引数が正であるため、遅延再支払期間の値は期日の14日後です。

daystart() 関数に渡された引数 day_start の値が 7/24 であったため、日付のタイムスタンプは 7:00 AMとなります。

firstworkdate

firstworkdate 関数は、**end_date** までに **no_of_workdays** (月 ~ 金曜日) の日数に達するように、オプションで指定された休日を考慮した最遅開始日を返します。**end_date** および **holiday** は有効な日付またはタイムスタンプでなければなりません。

構文:

```
firstworkdate(end_date, no_of_workdays {, holiday} )
```

戻り値データ型: integer

引数:

引数

引数	説明
end_date	評価する終了日のタイムスタンプ。
no_of_workdays	作成する作業日数。
holiday	作業日から除外する休日期間。休日は文字列定数の日付として示されます。コンマで区切り、複数の休日を設定できます。 **'12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'**

例と結果:
これらの例は、日付書式 **DD/MM/YYYY** を使用しています。日付書式は、データロードスクリプト上部の **SET DateFormat** ステートメントで指定されています。必要に応じて、書式を変更してください。

スクリプトの例

例	結果
firstworkdate ('29/12/2014', 9)	'17/12/2014' を返します。
firstworkdate ('29/12/2014', 9, '25/12/2014', '26/12/2014')	2日間の休日期間を考慮したため、'15/12/2014' を返します。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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;
```

結果テーブルには、テーブルの各レコードに対するFirstWorkDateの戻り値が表示されます。

結果テーブル

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

この関数は、現在のGreenwich Mean Timeを返します。これは地域設定から導かれます。この関数は、TimestampFormatシステム変数形式の値を返します。

アプリがリロードされると、GMT関数を使用する任意のロードスクリプトテーブル、変数、またはチャートオブジェクトは、システム時計から得られる最新のグリニッジ標準時に合わせて調整されます。

構文:

```
GMT( )
```

戻り値データ型: dual

これらの例は、タイムスタンプ形式 M/D/YYYY h:mm:ss[.fff] TTを使用しています。日付書式は、データロードスクリプト上部のSET TimestampFormatステートメントで指定されています。必要に応じて、書式を変更してください。

関数の例

例	結果
GMT()	3/28/2022 2:47:36 PM

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 変数 (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。この例では、GMT 関数を使って現在のグリニッジ標準時刻をロードスクリプトの変数に設定します。

ロードスクリプト

```
LET vGMT = GMT();
```

結果

データをロードしてシートを作成します。[**テキストと画像**] チャートオブジェクトを使用して、テキストボックスを作成します。

テキストボックスにこのメジャーを追加します。

```
=vGMT
```

テキストボックスには、下記に類似した日付と時刻が記載されたテキスト行を含みます:

```
3/28/2022 2:47:36 PM
```

例 2 – 年の開始が 11 月 (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Overdue という名前のテーブルにロードされる、図書館延滞料を含むデータセット。既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- それぞれの本の延滞日数を計算する、days_overdue という新しい項目の作成。

ロード スクリプト

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

Overdue:
    Load
        *,
        Floor(GMT()-due_date) as days_overdue
        ;
Load
*
Inline
[
cust_id,book_id,due_date
1,4,01/01/2021,
2,24,01/10/2021,
6,173,01/31/2021,
31,281,02/01/2021,
86,265,02/10/2021,
52,465,06/30/2021,
26,537,07/26/2021,
92,275,10/31/2021,
27,455,11/01/2021,
27,46,12/31/2021
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- due_date
- book_id
- days_overdue

結果テーブル

due_date	book_id	days_overdue
01/01/2021	4	455
01/10/2021	24	446
01/31/2021	173	425
02/01/2021	281	424
02/10/2021	265	415
06/30/2021	465	275

due_date	book_id	days_overdue
07/26/2021	537	249
10/31/2021	275	152
11/01/2021	455	151
12/31/2021	46	91

[days_overdue] 項目の値は、GMT() 関数を使って、現在のグリニッジ標準時刻と元の期日の差を検出することにより計算します。日数のみを計算するには、結果は Floor() 関数を使って一番近い整数に丸められます。

例 3 – チャート オブジェクト (チャート)

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。ロード スクリプトには、最初の例と同じデータセットが含まれます。既定の DateFormat システム変数 MM/DD/YYYY が使用されます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。延滞日数の値は、チャート オブジェクトのメジャーを介して計算されます。

ロード スクリプト

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

Overdue:
Load
*
Inline
[
cust_id,book_id,due_date
1,4,01/01/2021,
2,24,01/10/2021,
6,173,01/31/2021,
31,281,02/01/2021,
86,265,02/10/2021,
52,465,06/30/2021,
26,537,07/26/2021,
92,275,10/31/2021,
27,455,11/01/2021,
27,46,12/31/2021
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- due_date
- book_id

次のメジャーを作成します:

```
=Floor(GMT() - due_date)
```

結果テーブル

due_date	book_id	=Floor(GMT()-due_date)
01/01/2021	4	455
01/10/2021	24	446
01/31/2021	173	425
02/01/2021	281	424
02/10/2021	265	415
06/30/2021	465	275
07/26/2021	537	249
10/31/2021	275	152
11/01/2021	455	151
12/31/2021	46	91

[days_overdue] 項目の値は、GMT() 関数を使って、現在のグリニッジ標準時刻と元の期日の差を検出することにより計算します。日数のみを計算するには、結果は Floor() 関数を使って一番近い整数に丸められます。

hour

この関数は、**expression** の小数部が標準的な数値の解釈に従って時間と判断される場合に、時間を表す整数を返します。

構文:

```
hour(expression)
```

戻り値データ型: 整数

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロード

エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
hour('09:14:36')	与えられたテキスト文字列は、TimestampFormat 変数で定義された日付と時刻形式と一致するため、暗黙のうちにタイムスタンプに変換されます。式は 9 を返します。
hour('0.5555')	式は 13 を返します (0.5555 = 13:19:55 のため)。

例 1 – 変数 (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- タイムスタンプによるトランザクションを含むデータセット
- 既定の TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT)

項目「hour」を作成し、購入がいつ行われたかを計算します。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
  Load
    *,
    hour(date) as hour
    ;
Load
*
Inline
[
id,date,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
```

```
9506,'2022-01-06 08:49:09',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- hour

結果テーブル

日付	時間
2022-01-01 22:10:22	22
2022-01-02 08:35:54	8
2022-01-03 05:40:49	5
2022-01-03 14:21:53	14
2022-01-04 18:49:38	18
2022-01-04 22:58:34	22
2022-01-05 19:04:57	19
2022-01-05 19:34:46	19
2022-01-06 08:49:09	8
2022-01-06 11:29:38	11

時間項目の値は、hour() 関数を使用し、先行するLOAD ステートメントの数式として日付を渡すことによって作成されます。

例 2 – チャート オブジェクト (チャート)

ロード スクリプトとチャートの数式

概要

データ ロード エディター を開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています:

- 最初の例と同じデータセット。
- 既定の TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT)。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。「hour」値は、チャート オブジェクトのメジャーを介して計算されます。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
*
Inline
[
id,date,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: date。

「hour」を計算するには、次のメジャーを作成します。

```
=hour(date)
```

結果テーブル

due_date	=hour(date)
2022-01-01 22:10:22	22
2022-01-02 08:35:54	8
2022-01-03 05:40:49	5
2022-01-03 14:21:53	14
2022-01-04 18:49:38	18
2022-01-04 22:58:34	22
2022-01-05 19:04:57	19
2022-01-05 19:34:46	19
2022-01-06 08:49:09	8
2022-01-06 11:29:38	11

「hour」の値は、hour() 関数を使用し、チャート オブジェクトのメジャーの数式として日付を渡すことによって作成されます。

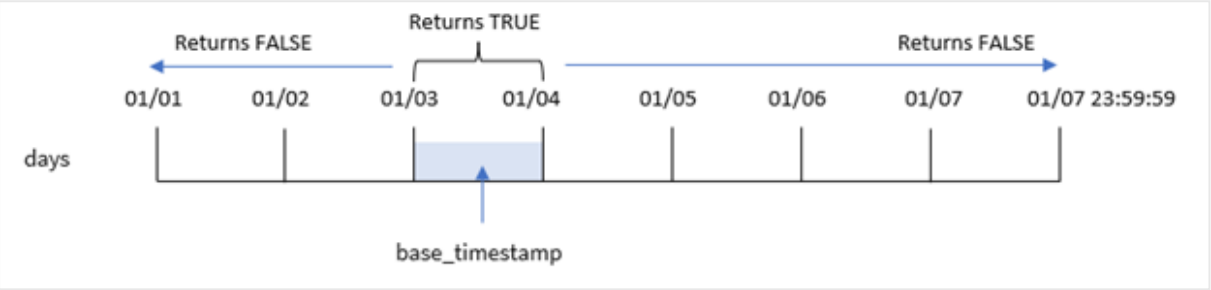
inday

この関数は、**base_timestamp** を含む日に **timestamp** が含まれている場合、True を返します。

構文:

```
InDay (timestamp, base_timestamp, period_no[, day_start])
```

inday 関数の図



inday() 関数は、base_timestamp 引数を使用して、タイムスタンプが該当する日を識別します。1 日の開始時刻は、既定では深夜です。ただし、inday() 関数の day_start 引数を使用して、1 日の開始時刻を変更できます。この日が定義されると、関数は指定されたタイムスタンプ値をその日と比較するときにブール値の結果を返します。

使用に適しているケース

inday() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。これは、評価された日付が問題のタイムスタンプの日に発生したかどうかに応じて、集計または計算を返します。

例えば、inday() 関数を使用して、特定の日に製造されたすべての機器を識別することができます。

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

引数

引数	説明
timestamp	base_timestamp と比較したい日付と時刻。
base_timestamp	タイムスタンプの評価に使用する日付と時刻。
period_no	日は period_no によって補正することができます。period_no は整数で、値 0 は base_timestamp を含む日を示します。period_no の値が負の場合は過去の日を、正の場合は将来の日を示します。
day_start	1 日の開始時刻を深夜 0 時以外に設定する場合は、day_start を使用して 1 日未満の長さを補正値として指定します。例えば、0.125 は午前 3 時を意味します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0)	True を返す
inday ('01/12/2006 12:23:00 PM', '01/13/2006 12:00:00 AM', 0)	False を返す
inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', -1)	False を返す
inday ('01/11/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', -1)	True を返す
inday ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0, 0.5)	False を返す
inday ('01/12/2006 11:23:00 AM', '01/12/2006 12:00:00 AM', 0, 0.5)	True を返す

例 1 – Load ステートメント (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされるタイムスタンプによるトランザクションを含むデータセット。
- TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT) 形式で提供される日付項目。
- in_day 項目として設定されている inday() 関数を含む先行する LOAD。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        inday(date,'01/05/2022 12:00:00 AM', 0) as in_day
        ;
```

```
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_day

結果テーブル

日付	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	-1
01/05/2022 7:04:57 PM	-1
01/06/2022 8:49:09 AM	0

項目は、関数を使用し、日付項目、1月 5日のハードコードされたタイムスタンプ、関数の引数として0の を渡すことにより、前の load ステートメントで作成されます。in_dayinday()period_no0

例 2 – period_no

ロードスクリプトと結果

概要

ロードスクリプトは、最初の例で使用されたものと同じデータセットとシナリオを使用します。

ただし、この例では、トランザクションの日付が1月 5 日の 2 日前に発生したかどうかを計算することがタスクです。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        inday(date,'01/05/2022 12:00:00 AM', -2) as in_day
        ;
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_day

結果テーブル

日付	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	-1
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	0
01/05/2022 11:29:38 AM	0

日付	in_day
01/05/2022 7:04:57 PM	0
01/06/2022 8:49:09 AM	0

この場合、inday() 関数のオフセット引数として -2 の period_no が使用されたため、この関数は、各トランザクションの日付が1月 3 日に発生したかどうかを判別します。これは、1 つのトランザクションが TRUE のブール結果を返す出力テーブルで確認できます。

例 3 – day_start

ロードスクリプトと結果

概要

ロードスクリプトは、前の例で使用されたものと同じデータセットとシナリオを使用します。

ただし、この例では、会社のポリシーでは、就業日は 7 時 AM に開始および終了します。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        inday(date,'01/05/2022 12:00:00 AM', 0, 7/24) as in_day
        ;
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_day

結果テーブル

日付	in_day
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	-1
01/04/2022 10:58:34 PM	-1
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	0
01/05/2022 7:04:57 PM	0
01/06/2022 8:49:09 AM	0

7 時 AM である 7/24 の start_day 引数が inday() 関数で使用されているため、この関数は、各トランザクションの日付が 1 月 4 日の 7 時 AM から 1 月 5 日の 7 時 AM までに発生したかどうかを判別します。

これは、1 月 4 日の 7 時 AM 以降に発生したトランザクションが TRUE のブール結果を返し、1 月 5 日の 7 時 AM 以降に発生したトランザクションが FALSE のブール結果を返す出力テーブルで確認できます。

例 4 – チャートオブジェクト

ロードスクリプトとチャートの数式

概要

ロードスクリプトは、前の例で使用されたものと同じデータセットとシナリオを使用します。

ただし、この例ではデータセットは変更されず、アプリケーションにロードされます。チャートオブジェクトにメジャーを作成することにより、1 月 5 日にトランザクションが発生するかどうかを判断するために計算します。

ロードスクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
```

```
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- date

トランザクションが1月 5 日に行われるかどうかを計算するには、次のメジャーを作成します。

```
=inday(date,'01/05/2022 12:00:00 AM',0)
```

結果テーブル

日付	inday(date,'01/05/2022 12:00:00 AM',0)
01/01/2022 7:34:46 PM	0
01/01/2022 10:10:22 PM	0
01/02/2022 8:35:54 AM	0
01/03/2022 2:21:53 PM	0
01/04/2022 6:49:38 PM	0
01/04/2022 10:58:34 PM	0
01/05/2022 5:40:49 AM	-1
01/05/2022 11:29:38 AM	-1
01/05/2022 7:04:57 PM	-1
01/06/2022 8:49:09 AM	0

例 5 – シナリオ

ロードスクリプトと結果

概要

この例では、機器のエラーにより、1月 5 日に製造された製品に欠陥があることが確認されています。エンドユーザーは、製造された製品のステータスが「不具合」または「不具合なし」であったこと、および1月 5 日に製造された製品のコストを日付別に表示するチャートオブジェクトを希望しています。

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「製品」というテーブルにロードされるデータセット。
- テーブルには次の項目が含まれています。

- 製品 ID
- 製造時間
- コスト

ロード スクリプト

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
9497,'01/01/2022 7:34:46 PM',13.24
9498,'01/01/2022 10:10:22 PM',31.43
9499,'01/02/2022 8:35:54 AM',36.34
9500,'01/03/2022 2:21:53 PM',51.75
9501,'01/04/2022 6:49:38 PM',15.35
9502,'01/04/2022 10:58:34 PM',74.34
9503,'01/05/2022 5:40:49 AM',73.53
9504,'01/05/2022 11:29:38 AM',50.00
9505,'01/05/2022 7:04:57 PM',47.25
9506,'01/06/2022 8:49:09 AM',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

```
=dayname(manufacture_date)
```

次のメジャーを作成します:

- `=if(only(InDay(manufacture_date,makedate(2022,01,05),0)),'Defective','Faultless')`
- `=sum(cost_price)`

メジャーの [**数値書式**] を [**通貨**] に設定します。

[**スタイル**] で [**合計**] をオフにします。

結果テーブル

dayname (manufacture_ date)	=if(only(InDay(manufacture_date,makedate (2022,01,05),0)),'Defective','Faultless')	=sum (cost_ price)
01/01/2022	不具合なし	44.67
01/02/2022	不具合なし	36.34
01/03/2022	不具合なし	51.75
01/04/2022	不具合なし	89.69

dayname (manufacture_ date)	=if(only(InDay(manufacture_date,makedate (2022,01,05),0)),'Defective','Faultless')	=sum (cost_ price)
01/05/2022	不具合	170.78
01/06/2022	不具合なし	74.23

inday() 関数は、各製品の製造日を評価するときにブール値を返します。1 月 5 日に製造された製品の場合、inday() 関数はブール値 TRUE を返し、製品を「不具合」としてマークします。FALSE の値を返し、その日に製造されなかった製品については、その製品に「不具合なし」のマークが付けられます。

indaytotime

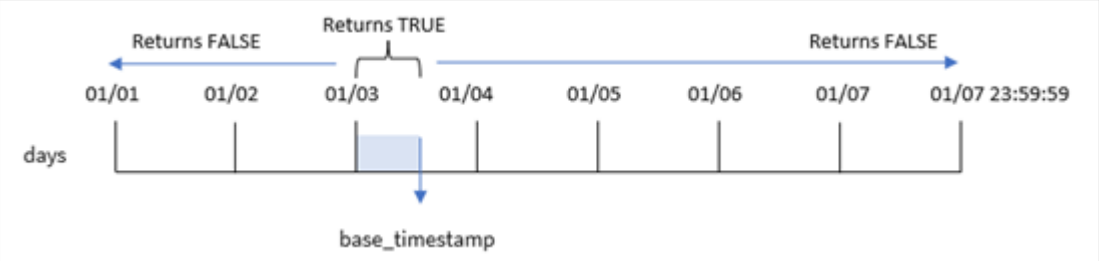
この関数は、**timestamp** が **base_timestamp** のミリ秒単位まで正確に **base_timestamp** を含む日の範囲内にある場合、True を返します。

構文:

```
InDayToTime (timestamp, base_timestamp, period_no[, day_start])
```

indaytotime() 関数は、その日のセグメント中にタイムスタンプ値が発生するタイミングに応じてブール値の結果を返します。このセグメントの開始境界は 1 日の始まりであり、既定では深夜に設定されています。1 日の始まりは、indaytotime() 関数の day_start 引数によって変更できます。日付セグメントの終了境界は、関数の base_timestamp 引数によって決定されます。

indaytotime 関数の図。



使用に適しているケース

indaytotime() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。indaytotime() 関数は、ベース タイムスタンプの時刻までの、日のセグメントでタイムスタンプが発生したかどうかに応じて、集計または計算を返します。

例えば、indaytotime() 関数を使用して、今日までに行われたショーのチケット販売の合計を表示できます。

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

引数

引数	説明
timestamp	base_timestamp と比較したい日付と時刻。
base_timestamp	タイムスタンプの評価に使用する日付と時刻。
period_no	日は period_no によって補正することができます。period_no は整数で、値 0 は base_timestamp を含む日を示します。period_no の値が負の場合は過去の日を、正の場合は将来の日を示します。
day_start	(オプション) 1 日の開始時刻を深夜 0 時以外に設定する場合は、day_start を使用して 1 日未満の長さを補正値として指定します。例えば、午前 3 時を表すには 0.125 を使用します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
indaytotime ('01/12/2006 12:23:00 PM', '01/12/2006 11:59:00 PM', 0)	True を返す
indaytotime ('01/12/2006 12:23:00 PM', '01/12/2006 12:00:00 AM', 0)	False を返す
indaytotime '01/11/2006 12:23:00 PM', '01/12/2006 11:59:00 PM', -1)	True を返す

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 1 月 4 日から 5 日までの一連のトランザクションを含むデータセットが、「トランザクション」と呼ばれるテーブルにロードされます。

- TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT) 形式で提供される日付項目。
- 各トランザクションが 9:00 AM より前に行われるかどうかを決定する 'in_day_to_time' 項目として設定された indaytotime() 関数を含む、先行する LOAD。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        indaytotime(date,'01/05/2022 9:00:00 AM',0) as in_day_to_time
        ;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

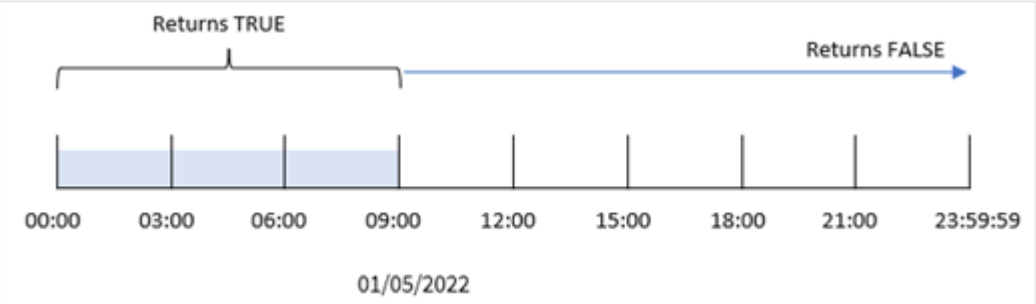
- date
- in_day_to_time

結果テーブル

日付	in_day_to_time
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0

日付	in_day_to_time
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	-1
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

例 1 9 時 AM の制限がある indaytotime 関数の図。



in_day_to_time field は、indaytotime() 関数を使用し、日付項目、1 月 5 日 9:00 AM のハードコードされたタイムスタンプ、関数の引数として 0 のオフセットを渡すことにより、前の load ステートメントで作成されます。1 月 5 日の 0 時 AM から 9 時 AM の間に発生するトランザクションはすべて TRUE を返します。

例 2 – period_no

ロード スクリプトと結果

概要

ロード スクリプトは、最初の例で使用されたものと同じデータセットとシナリオを使用します。

ただし、この例では、トランザクションの日付が 1 月 5 日の 9 時 AM の 1 日前に発生したかどうかを計算します。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
```

```
        *,
        indaytotime(date,'01/05/2022 9:00:00 AM', -1) as in_day_to_time
        ;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

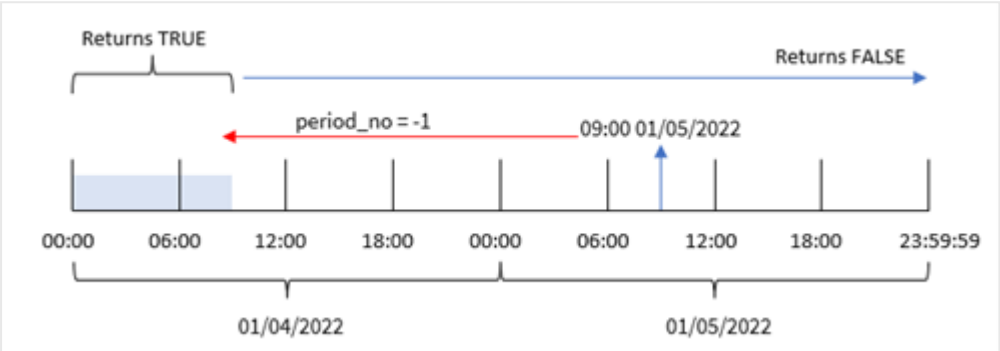
- date
- in_day_to_time

結果テーブル

日付	in_day_to_time
01/04/2022 3:41:54 AM	-1
01/04/2022 4:19:43 AM	-1
01/04/2022 04:53:47 AM	-1
01/04/2022 8:38:53 AM	-1
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	0
01/05/2022 8:44:36 AM	0

日付	in_day_to_time
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

例 2 1月 4 日からのトランザクションを使用した indaytotime 関数の図。



この例では、indaytotime() 関数のオフセット引数として -1 のオフセットが使用されたため、この関数は、各トランザクションの日付が 1 月 4 日の 9 時 AM より前に発生したかどうかを判別します。これは、トランザクションが TRUE のブール結果を返す出力テーブルで確認できます。

例 3 – day_start

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例では、会社のポリシーでは、就業日は 8 時 AM に開始および終了します。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        indaytotime(date,'01/05/2022 9:00:00 AM', 0,8/24) as in_day_to_time
        ;
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
```

```
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

結果

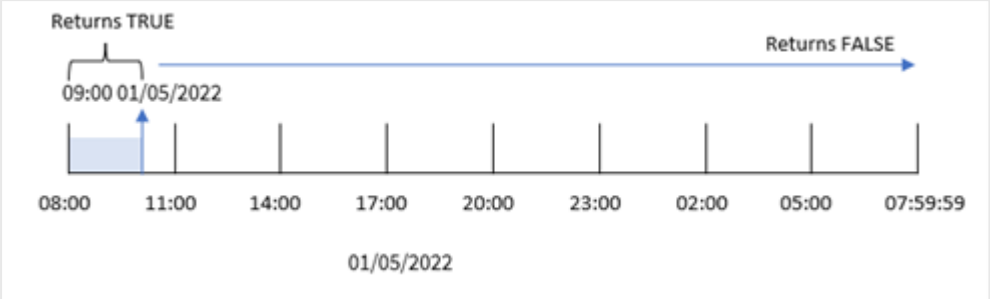
データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_day_to_time

結果テーブル

日付	in_day_to_time
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	0
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

例 3 8 時 AM から 9 時 AM までのトランザクションを使用した indaytotime 関数の図。



indaytotime() 関数では 8/24 の start_day 引数 (8:00 AM に相当) が使用されているため、毎日 8:00 AM に開始および終了します。したがって、indaytotime() 関数は、1 月 5 日の 8:00 AM から 9:00 AM の間に発生したトランザクションに対して TRUE のブール値の結果を返します。

例 4 – チャート オブジェクト

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例ではデータセットは変更されず、アプリケーションにロードされます。チャート オブジェクトにメジャーを作成することにより、1 月 5 日の 9 時 AM より前にトランザクションが発生するかどうかを判断するために計算します。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/04/2022 3:41:54 AM',25.66
8189,'01/04/2022 4:19:43 AM',87.21
8190,'01/04/2022 4:53:47 AM',53.80
8191,'01/04/2022 8:38:53 AM',69.98
8192,'01/04/2022 10:37:52 AM',57.42
8193,'01/04/2022 1:54:10 PM',45.89
8194,'01/04/2022 5:53:23 PM',82.77
8195,'01/04/2022 8:13:26 PM',36.23
8196,'01/04/2022 10:00:49 PM',76.11
8197,'01/05/2022 7:45:37 AM',82.06
8198,'01/05/2022 8:44:36 AM',17.17
8199,'01/05/2022 11:26:08 AM',40.39
8200,'01/05/2022 6:43:08 PM',37.23
8201,'01/05/2022 10:54:10 PM',88.27
8202,'01/05/2022 11:09:09 PM',95.93
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

date。

トランザクションが1月 5 日の9:00 AM より前に行われるかどうかを判断するには、次のメジャーを作成します。

```
=indaytotime(date,'01/05/2022 9:00:00 AM',0)
```

結果テーブル

日付	=indaytotime(date,'01/05/2022 9:00:00 AM',0)
01/04/2022 3:41:54 AM	0
01/04/2022 4:19:43 AM	0
01/04/2022 04:53:47 AM	0
01/04/2022 8:38:53 AM	0
01/04/2022 10:37:52 AM	0
01/04/2022 1:54:10 PM	0
01/04/2022 5:53:23 PM	0
01/04/2022 8:13:26 PM	0
01/04/2022 10:00:49 PM	0
01/05/2022 7:45:37 AM	-1
01/05/2022 8:44:36 AM	-1
01/05/2022 11:26:08 AM	0
01/05/2022 6:43:08 PM	0
01/05/2022 10:54:10 PM	0
01/05/2022 11:09:09 PM	0

in_day_to_time メジャーは、indaytotime() 関数を使用し、日付項目、1月 5 日 9:00 AM のハードコードされたタイムスタンプ、関数の引数として 0 のオフセットを渡すことにより、チャートオブジェクトで作成されます。1月 5 日の 0 時 AM から9 時 AM の間に発生するトランザクションはすべて TRUE を返します。これは、結果テーブルで検証されます。

例 5 – シナリオ

ロードスクリプトと結果

概要

この例では、地元の映画館のチケット販売を含むデータセットが、Ticket_Sales というテーブルにロードされます。今日は 2022 年 5 月 3 日、11:00 AM です。

ユーザーは、KPI チャート オブジェクトに、今日までに行われたすべてのショーから得られた収益を表示したいと考えています。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Ticket_Sales:
Load
*
Inline
[
sale ID, show time, ticket price
1,05/01/2022 09:30:00 AM,10.50
2,05/03/2022 05:30:00 PM,21.00
3,05/03/2022 09:30:00 AM,10.50
4,05/03/2022 09:30:00 AM,31.50
5,05/03/2022 09:30:00 AM,10.50
6,05/03/2022 12:00:00 PM,42.00
7,05/03/2022 12:00:00 PM,10.50
8,05/03/2022 05:30:00 PM,42.00
9,05/03/2022 08:00:00 PM,31.50
10,05/04/2022 10:30:00 AM,31.50
11,05/04/2022 12:00:00 PM,10.50
12,05/04/2022 05:30:00 PM,10.50
13,05/05/2022 05:30:00 PM,21.00
14,05/06/2022 12:00:00 PM,21.00
15,05/07/2022 09:30:00 AM,42.00
16,05/07/2022 10:30:00 AM,42.00
17,05/07/2022 10:30:00 AM,10.50
18,05/07/2022 05:30:00 PM,10.50
19,05/08/2022 05:30:00 PM,21.00
20,05/11/2022 09:30:00 AM,10.50
];
```

結果

以下を実行します。

1. KPI オブジェクトを作成します。
2. indaytotime() 関数を使用して、今日までに行われたショーのすべてのチケット販売の合計を表示するメジャーを作成します。

```
=sum(if(indaytotime([show time],'05/03/2022 11:00:00 AM',0),[ticket price],0))
```

3. KPI オブジェクトのラベル「現在の収益」を作成します。
4. メジャーの **[数値書式]** を **[通貨]** に設定します。

2022 年 5 月 3 日の 11:00 AM までのチケット販売の合計は 52.50 ドルです。

indaytotime () 関数は、各チケット販売の表示時間を現在の時刻 (05/03/2022 11:00:00 AM」) と比較するときにブール値を返します。5 月 3 日の 11:00 AM より前のショーの場合、indaytotime () 関数はブール値 TRUE を返し、そのチケット価格は合計に含まれます。

inlunarweek

この関数は、**timestamp** が **base_date** を含む週周期の範囲内かどうかを判断します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて、各週は正確に 7 日構成となります。

構文:

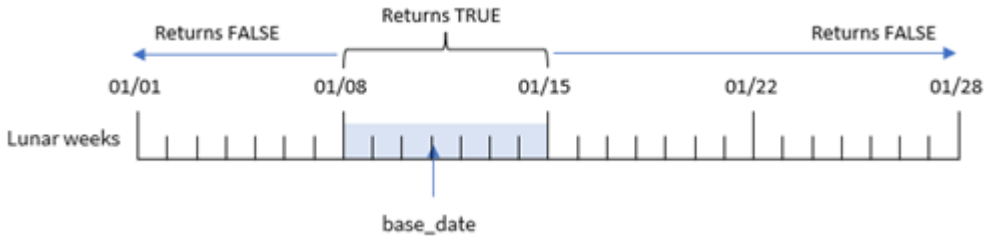
```
InLunarWeek (timestamp, base_date, period_no[, first_week_day])
```

戻り値データ型: ブール値

> *Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。*

inlunarweek() 関数は、base_date がどの旧暦の週に当たるかを決定します。次に、base_date と同じ旧暦の週の間に各タイムスタンプ値が発生することが決定したら、ブール値を返します。

inlunarweek() 関数の図



使用に適しているケース

inlunarweek() 関数はブール値の結果を返します。通常、このタイプの関数は IF 式の条件として使用されます。これにより、評価される日付が問題の旧暦の週に発生したかどうかに応じて、集計または計算を返します。

例えば、inlunarweek() 関数を使用して、特定の旧暦の週に製造されたすべての機器を識別することができます。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	週周期の評価に使用する日付。
period_no	週周期は **period_no** によって補正することができます。period_no は整数で、値 0 は **base_date** を含む週周期を示します。**period_no** の値が負の場合は過去の週周期を、正の場合は将来の週周期を示します。
first_week_day	0 未満または 0 よりも大きい補正値。日数または 1 日未満の長さ、またはその両方を指定して、年の開始時点を変更できます。

関数の例

例	結果
inlunarweek ('01/12/2013', '01/14/2013', 0)	timestamp の値 01/12/2013 が 01/08/2013 ~ 01/14/2013 の週に当たるため、TRUE を返します。
inlunarweek ('01/12/2013', '01/07/2013', 0)	base_date 01/07/2013 が 01/01/2013 ~ 01/07/2013 と定義された旧暦の週に当たるため、FALSE を返します。
inlunarweek ('01/12/2013', '01/14/2013', -1)	FALSE を返します。period_noの値に-1 が指定されており、前の週である01/01/2013 から01/07/2013 に週がシフトしています。
inlunarweek ('01/07/2013', 01/14/2013', -1)	TRUE を返します。上の例と異なり、timestamp は過去へシフトするとタイムスタンプがその週の範囲内になります。
inlunarweek ('01/11/2006', '01/08/2006', 0, 3)	FALSE を返します。first_week_day に値 3 を指定すると、年の初めが 01/04/2013 から計算されます。そのため、base_date の値は最初の週に当たり、timestamp の値は 01/11/2013 ~ 01/17/2013 の週に当たります。

inlunarweek() 関数は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
lunarweekname (page 847)	この関数は、入力された日付が属する年の旧暦の週番号を決定するために使用されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、1 月のトランザクションのデータセット。
- 日付項目は DateFormat システム変数形式 (MM/DD/YYYY) で提供されています。

トランザクションが 1 月 10 日と同じ旧暦の週に発生するかどうかを決定する項目 in_lunar_week を作成します。

ロードスクリプト

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

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 0) as in_lunar_week
        ;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

結果

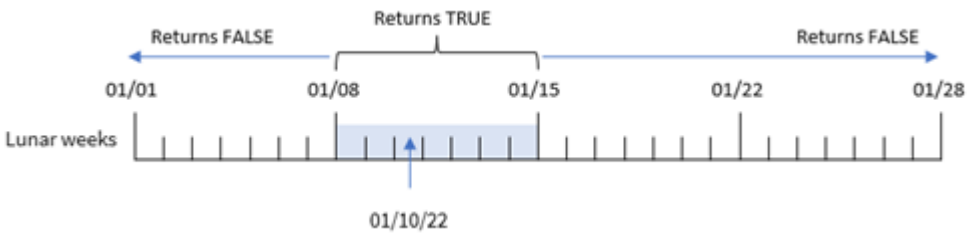
データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_lunar_week

結果テーブル

日付	in_lunar_week
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	-1
1/14/2022	-1
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	0
1/23/2022	0

inlunarweek() 関数、基本的な例



in_lunar_week 項目は、inlunarweek() 関数を使用し、関数の引数として次の項目を渡すことにより、先行するLOAD ステートメントで作成されます。

- date 項目
- base_date としてハードコード化された1月 10 日の日付
- 0 の period_no

旧暦の週は 1 月 1 日から始まるため、1 月 10 日は 1 月 8 日に始まり1月 14 日に終わる旧暦の週に当たります。そのため、それら2 つの日付の間に発生するトランザクションはブール値 TRUE を返します。これは、結果テーブルで検証されます。

例 2 – period_no

例と結果:

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 日付項目は DateFormat システム変数形式 (MM/DD/YYYY) で提供されています。

ただし、この例では、トランザクションが1月 10 日より旧暦 2 週間後に発生するかどうかを決定する項目 2_lunar_weeks_later を作成します。

ロードスクリプト

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

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 2) as [2_lunar_weeks_later]
        ;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
```

```
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

結果

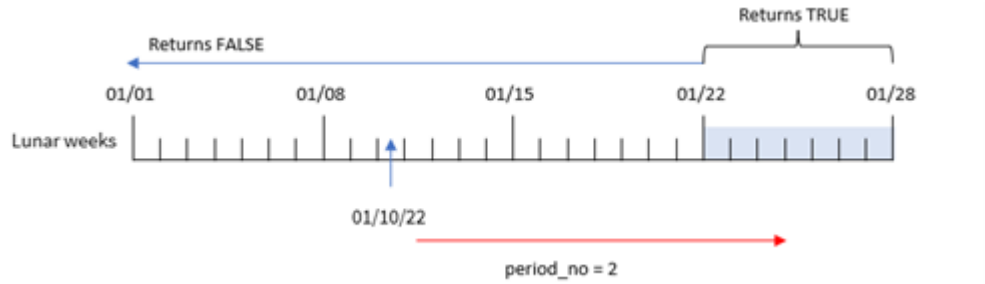
データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- 2_lunar_weeks_later

結果テーブル

日付	2_lunar_weeks_later
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	0
1/9/2022	0
1/10/2022	0
1/11/2022	0
1/12/2022	0
1/13/2022	0
1/14/2022	0
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0
1/22/2022	-1
1/23/2022	-1

inlunarweek() 関数、period_no example



この例では、period_no の 2 が inlunarweek() 関数のオフセット引数として使用されるため、関数がトランザクションを突き合わせる旧暦の週として 1 月 22 日で始まる週を定義しします。したがって、1 月 22 日 ~ 1 月 28 日に発生したトランザクションは、TRUEのブール値の結果を返します。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトは、最初の例と同じデータセットとシナリオを使用します。ただし、この例では、旧暦の週が 1 月 6 日に始まるよう設定しています。

- 最初の例と同じデータセットとシナリオ。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- first_week_day 引数 5。これにより、旧暦の週が 1 月 5 日に始まります。

ロードスクリプト

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

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 0,5) as in_lunar_week
        ;
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
```

```
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

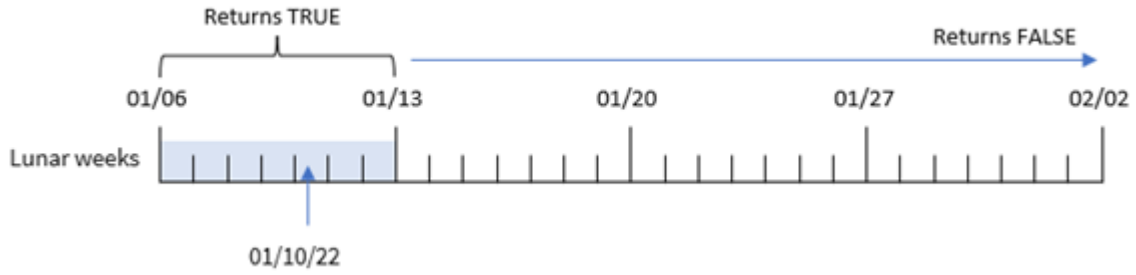
- date
- in_lunar_week

結果テーブル

日付	in_lunar_week
1/5/2022	0
1/6/2022	-1
1/7/2022	-1
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	0
1/14/2022	0
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/20/2022	0
1/21/2022	0

日付	in_lunar_week
1/22/2022	0
1/23/2022	0

inlunarweek() 関数、first_week_day 例



この例では、first_week_date 引数の 5 が inlunarweek() 関数で使用されているため、旧暦の週の始めが1月 6 日にオフセットされます。そのため 1 月 10 日は 1 月 6 日に始まり12 日に終わる旧暦の週に当たります。これら2 つの日付の間に発生するトランザクションは、ブール値 TRUE を返します。

例 4 – チャート オブジェクト

ロード スクリプトとチャートの数式:

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 日付項目は DateFormat システム変数形式 (MM/DD/YYYY) で提供されています。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが 1 月 10 日と同じ旧暦の週に発生したかどうかを判断する計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
```

```
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `date`。

トランザクションが 1 月 10 日を含む週に発生するかどうかを計算するには、次のメジャーを作成します。

```
= inlunarweek(date,'01/10/2022', 0)
```

結果テーブル

日付	=inlunarweek(date,'01/10/2022', 0)
1/5/2022	0
1/6/2022	0
1/7/2022	0
1/8/2022	-1
1/9/2022	-1
1/10/2022	-1
1/11/2022	-1
1/12/2022	-1
1/13/2022	-1
1/14/2022	-1
1/15/2022	0
1/16/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0

日付	=inlunarweek(date,'01/10/2022', 0)
1/20/2022	0
1/21/2022	0
1/22/2022	0
1/23/2022	0

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- 製品 ID、製造年月日、原価を含む情報。

機器のエラーにより、1 月 12 日を含む旧暦の週に製造された製品に欠陥があることが確認されています。エンドユーザーは、製造された製品のステータスが「不具合」または「不具合なし」であったこと、およびその月に製造された製品のコストを旧暦の週別に表示するチャートオブジェクトを希望しています。

ロードスクリプト

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

Transactions:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8183,'1/5/2022',42.32
8184,'1/6/2022',68.22
8185,'1/7/2022',15.25
8186,'1/8/2022',25.26
8187,'1/9/2022',37.23
8188,'1/10/2022',37.23
8189,'1/11/2022',17.17
8190,'1/12/2022',88.27
8191,'1/13/2022',57.42
8192,'1/14/2022',53.80
8193,'1/15/2022',82.06
8194,'1/16/2022',87.21
8195,'1/17/2022',95.93
8196,'1/18/2022',45.89
8197,'1/19/2022',36.23
8198,'1/20/2022',25.66
```

```
8199,'1/21/2022',82.77
8200,'1/22/2022',69.98
8201,'1/23/2022',76.11
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 月名を表示する軸を作成します。
   `=lunarweekname(manufacture_date)`
3. `inlunarweek()` 関数を使って、不具合のある製品とない製品を特定するメジャーを作成します。
   `=if(only(inlunarweek(manufacture_date,makedate(2022,01,12),0)), 'Defective','Faultless')`
4. 製品の `cost_price` を合計するメジャーを作成します。
   `=sum(cost_price)`
5. メジャーの **[数値書式]** を **[通貨]** に設定します。
6. **[スタイル]** で **[合計]** をオフにします。

結果テーブル

lunarweekname (manufacture_date)	=if(only(inlunarweek(manufacture_date,makedate(2022,01,12),0)), 'Defective','Faultless')	sum(cost_price)
2022/01	不具合なし	$125.79
2022/02	不具合	$316.38
2022/03	不具合なし	$455.75
2022/04	不具合なし	$146.09

`inlunarweek()` 関数は、各製品の製造日を評価するときにブール値を返します。1月10日を含む旧暦の週に製造された製品の場合、`inlunarweek()` 関数はブール値 `TRUE` を返し、製品を「不具合」としてマークします。`FALSE` の値を返し、その週に製造されなかった製品については、その製品に「不具合なし」のマークが付けられます。

inlunarweektodate

この関数は、**timestamp** が **base_date** の最後のミリ秒までの週周期の範囲内か確認します。Qlik Sense の旧暦の週は、1月1日を週の初日として数えるよう定義され、1年の最終週を除いて正確に7日構成となります。

構文:

```
InLunarWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

戻り値データ型： ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

inlunarweektodate() 関数の図の例

inlunarweektodate() 関数は、旧暦の週の終了点として機能します。一方、inlunarweek() 関数は、base_date がどの旧暦の週に当たるかを決定します。例えば、base_date が 1 月 5 日で、1 月 1 日 ~ 1 月 5 日のタイムスタンプはブール値の結果 TRUE を返しますが、1 月 6 日と 7 日以降はブール値 FALSE を返します。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	週周期の評価に使用する日付。
period_no	週周期は **period_no** によって補正することができます。period_no は整数で、値 0 は **base_date** を含む週周期を示します。**period_no** の値が負の場合は過去の週周期を、正の場合は将来の週周期を示します。
first_week_day	0 未満または 0 よりも大きい補正値。日数または 1 日未満の長さ、またはその両方を指定して、年の開始時点を変更できます。

使用に適しているケース

inlunarweektodate() 関数はブール値の結果を返します。通常、このタイプの関数は IF 式の条件として使用されます。inlunarweektodate() 関数は、評価された日付が対象の週の特定のセグメントに発生したかどうかに応じて、集計または計算を返すようにユーザーが希望する場合に使用されます。

例えば、inlunarweektodate() 関数を使用して、特定の日付までの週に製造されたすべての機器を識別することができます。

関数の例

例	結果
`inlunarweektodate ('01/12/2013', '01/13/2013', 0)`	timestamp の値 01/12/2013 が 01/08/2013 ~ 01/13/2013 の週の一部に当たるため、TRUE を返します。

例	結果
inlunarweektodate ('01/12/2013', '01/11/2013', 0)	2 つの日付が 01/12/2012 の同じ旧暦の週にありますが、timestamp の値が base_date の値より後であるため、FALSE を返します。
inlunarweektodate ('01/12/2006', '01/05/2006', 1)	TRUE を返します。period_no に 1 を指定すると base_date を 1 週間先にシフトするので、timestamp の値がその週周期の範囲内になります。

inlunarweektodate() 関数は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
lunarweekname (page 847)	この関数は、入力された日付が属する年の旧暦の週番号を決定するために使用されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、1 月の一連のトランザクションを含むデータセット。既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- 1 月 10 日と同じ旧暦の週に発生するトランザクションを決定する項目 in_lunar_week_to_date を作成します。

ロードスクリプト

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

```
Transactions:
    Load
        *,
        inlunarweektodate(date,'01/10/2022', 0) as in_lunar_week_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_lunar_week_to_date

結果テーブル

日付	in_lunar_week_to_date
1/1/2022	0
1/4/2022	0
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0

日付	in_lunar_week_to_date
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 関数、追加引数なし

in_lunar_week_to_date 項目は、inlunarweektodate() 関数を使用し、base_date と0 を関数の引数として1 月 10 日のハードコード化された日付 date を渡すことにより、先行する LOAD ステートメントで作成されます。

旧暦の週は 1 月 1 日から始まるため、1 月 10 日は 1 月 8 日に始まる旧暦の週に当たり、inlunarweektodate() 関数を使用しているため、その旧暦の週は 1 月 10 日に終わります。そのため、それら2 つの日付の間に発生するトランザクションはブール値 TRUE を返します。これは、結果テーブルで検証されます。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例のタスクは、トランザクションが 1 月 10 日より旧暦 2 週間後に発生するかどうかを決定する項目 2_lunar_weeks_later を作成することです。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inlunarweektodate(date,'01/10/2022', 2) as [2_lunar_weeks_later]
        ;
Load
*
Inline
```

```
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

結果

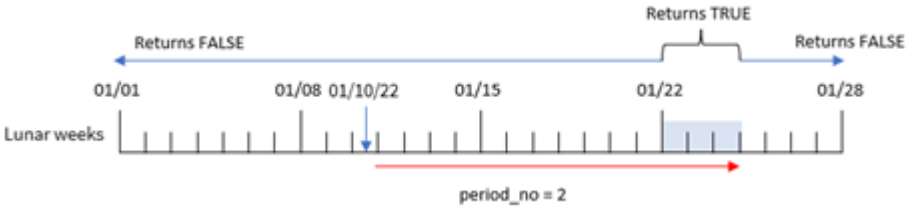
データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- 2_lunar_weeks_later

結果テーブル

日付	2_lunar_weeks_later
1/1/2022	0
1/4/2022	0
1/10/2022	0
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	-1
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 関数、period_no example



この例では、inlunarweektodate() 関数が 1 月 10 日までの旧暦の週が 3 日 (1 月 8、9、10 日) と等しいかどうかを決定します。period_no 2 がオフセット引数として使用されるため、この旧暦の週は 14 日間ずれます。そのため、これは 1 月 22、23、および 24 日を含む 3 日の旧暦の週を定義します。1 月 22 日 ~ 1 月 24 日に発生したトランザクションは、TRUEのブール値の結果を返します。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- first_week_date 引数 3。これにより、旧暦の週が 1 月 3 日に始まります。

ロードスクリプト

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

Transactions:
    Load
        *,
        inlunarweek(date,'01/10/2022', 0,3) as in_lunar_week_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
```

```
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_lunar_week_to_date

結果テーブル

日付	in_lunar_week_to_date
1/1/2022	0
1/4/2022	-1
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 関数、first_week_day 例

この例では、the first_week_date 引数 3 が inlunarweek() 関数で使用されているため、最初の旧暦の週は 1 月 3 日 ~ 1 月 10 日となります。1 月 10 日も base_date に当たるため、これら 2 つの日付の間に発生するトランザクションは、ブール値 TRUE を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが 1 月 10 日までの旧暦の週に発生したかどうかを判断する計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/17/2022',17.17
8190,'1/26/2022',88.27
8191,'1/12/2022',57.42
8192,'1/19/2022',53.80
8193,'1/21/2022',82.06
8194,'1/1/2022',40.39
8195,'1/27/2022',87.21
8196,'1/11/2022',95.93
8197,'1/29/2022',45.89
8198,'1/31/2022',36.23
8199,'1/18/2022',25.66
```

```
8200,'1/23/2022',82.77
8201,'1/15/2022',69.98
8202,'1/4/2022',76.11
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

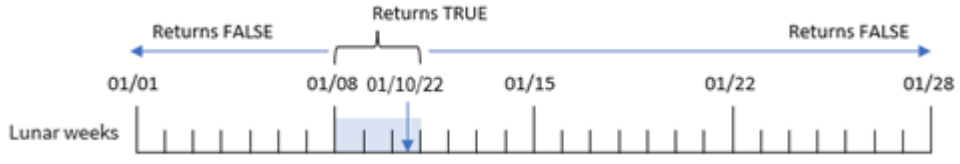
次のメジャーを作成します:

```
=inlunarweektodate(date,'01/10/2022', 0)
```

結果テーブル

日付	=inlunarweektodate(date,'01/10/2022', 0)
1/1/2022	0
1/4/2022	0
1/10/2022	-1
1/11/2022	0
1/12/2022	0
1/15/2022	0
1/17/2022	0
1/18/2022	0
1/19/2022	0
1/21/2022	0
1/23/2022	0
1/26/2022	0
1/27/2022	0
1/29/2022	0
1/31/2022	0

inlunarweektodate() 関数、チャートオブジェクトの例



in_lunar_week_to_date メジャーは、inlunarweektodate() 関数を使用し、日付項目、1 月 10 日のハードコードされた日付を base_date として、0 のオフセットを関数の引数として渡すことにより、チャートオブジェクトで作成されます。

旧暦の週は 1 月 1 日から始まるため、1 月 10 日は 1 月 8 日に始まる旧暦の週に当たります。さらに、inlunarweektodate() 関数を使用しているため、その旧暦の週は 1 月 10 日に終わります。そのため、それら 2 つの日付の間に発生するトランザクションはブール値 TRUE を返します。これは、結果テーブルで検証されます。

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- 製品 ID、製造年月日、原価を含む情報。

機器のエラーにより、1 月 12 日の旧暦の週に製造された製品に欠陥があることが確認されています。この問題は 1 月 13 日に解決されました。エンドユーザーは、製造された製品のステータスが「不具合」または「不具合なし」であったこと、およびその週に製造された製品のコストを週別に表示するチャートオブジェクトを希望しています。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';

Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'01/02/2022 12:22:06',37.23
8189,'01/05/2022 01:02:30',17.17
8190,'01/06/2022 15:36:20',88.27
8191,'01/08/2022 10:58:35',57.42
8192,'01/09/2022 08:53:32',53.80
8193,'01/10/2022 21:13:01',82.06
8194,'01/11/2022 00:57:13',40.39
8195,'01/12/2022 09:26:02',87.21
8196,'01/13/2022 15:05:09',95.93
8197,'01/14/2022 18:44:57',45.89
8198,'01/15/2022 06:10:46',36.23
8199,'01/16/2022 06:39:27',25.66
8200,'01/17/2022 10:44:16',82.77
8201,'01/18/2022 18:48:17',69.98
8202,'01/26/2022 04:36:03',76.11
8203,'01/27/2022 08:07:49',25.12
```

```
8204,'01/28/2022 12:24:29',46.23
8205,'01/30/2022 11:56:56',84.21
8206,'01/30/2022 14:40:19',96.24
8207,'01/31/2022 05:28:21',67.67
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 週名を表示する軸を作成します。
   `=weekname(manufacture_date)`
3. 次に、不具合のある製品とない製品を特定する、inlunarweektodate() 関数を使った軸を作成します。
   `=if(inlunarweektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')`
4. 製品の cost_price を合計するメジャーを作成します。
   `=sum(cost_price)`
5. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

=lunarweekname (manufacture_date)	=if(InLunarWeekToDate(manufacture_date,makedate (2022,01,12),0),'Defective','Faultless')	=Sum(cost_price)
2022/01	不具合なし	$142.67
2022/02	不具合	$320.88
2022/02	不具合なし	$141.82
2022/03	不具合なし	$214.64
2022/04	不具合なし	$147.46
2022/05	不具合なし	$248.12

inlunarweektodate() 関数は、各製品の製造日を評価するときにブール値を返します。TRUE のブール値を返すものについては、製品を 'Defective' とマークします。FALSE の値を返し、1 月 12 日までの旧暦の週に製造されていない製品については、製品を 'Faultless' とマークします。

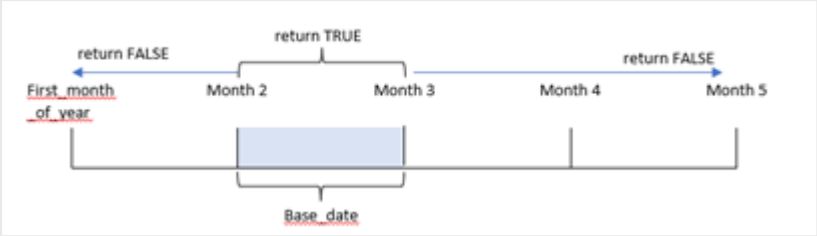
inmonth

この関数は、**timestamp** が **base_date** を含む月にある場合、True を返します。

構文:

```
InMonth (timestamp, base_date, period_no)
```

indaytotime 関数の図。



つまり、inmonth() 関数は、一連の日付がこの月に該当するかどうかを判断し、月を識別する base_date に基づいてブール値を返します。

使用に適しているケース

inmonth() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。これは、対象の日付を含み、日付がその月に発生したかどうかに応じて、集計または計算を返します。

例えば、inmonth() 関数を使用して、特定の月に製造されたすべての機器を識別することができます。

戻り値データ型： ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

引数

引数	説明
日付と時刻	base_date と比較する日付。
base_date	月の評価に使用する日付。base_date はある月の任意の日であることに注意してください。
period_no	月は period_no によって補正することができます。period_no は整数で、値 0 は base_date を含む月を示します。period_no の値が負の場合は過去の月を、正の場合は将来の月を示します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
inmonth ('25/01/2013', '01/01/2013', 0)	True を返す
inmonth('25/01/2013', '23/04/2013', 0)	False を返す
inmonth ('25/01/2013', '01/01/2013', -1)	False を返す
inmonth ('25/12/2012', '17/01/2013', -1)	True を返す

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2022 年上半期の一連の取引を含むデータセット。
- トランザクションが 4 月に発生したかどうかを判断する追加の変数「in_month」を使用した先行する LOAD。

ロードスクリプト

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

Transactions:
  Load
    *,
    inmonth(date,'04/01/2022', 0) as in_month
    ;
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
8197,'4/15/2022',45.89
8198,'4/25/2022',36.23
8199,'5/20/2022',25.66
```

```
8200,'5/22/2022',82.77
8201,'6/19/2022',69.98
8202,'6/22/2022',76.11
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_month

関数の例

日付	in_month
1/10/2022	0
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	0
2/2/2022	0
2/20/2022	0
4/11/2022	-1
4/13/2022	-1
4/15/2022	-1
4/25/2022	-1
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

「in_month」項目は、inmonth() 関数を使用し、日付項目、ハードコードされた 4 月 1 日、関数の引数として 0 の base_date と period_no を渡すことにより、先行する LOAD ステートメントで作成されます。

base_date は、TRUE のブール結果を返す月を識別します。したがって、4 月に発生したすべてのトランザクションは TRUE を返し、結果テーブルで検証されます。

例 2 – period_no

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例では、トランザクションが 4 月の 2 か月前に発生したかどうかを判断する項目「2_months_prior」を作成します。

ロード スクリプト

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

Transactions:
Load
   *,
   inmonth(date,'04/01/2022', -2) as [2_months_prior]
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
8197,'4/15/2022',45.89
8198,'4/25/2022',36.23
8199,'5/20/2022',25.66
8200,'5/22/2022',82.77
8201,'6/19/2022',69.98
8202,'6/22/2022',76.11
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- 2_months_prior

関数の例

日付	2_months_prior
1/10/2022	0
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	-1
2/2/2022	-1
2/20/2022	-1

日付	2_months_prior
4/11/2022	0
4/13/2022	0
4/15/2022	0
4/25/2022	0
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

inmonth() 関数の period_no 引数として -2 を使用すると、[base_date 引数で定義された月が 2 か月前にシフトされます。この例では、定義された月を 4 月から 2 月に変更します。

したがって、2 月に行われるすべてのトランザクションは、ブール値の結果 TRUE を返します。

例 3 – チャート オブジェクト

ロード スクリプトとチャートの数式

概要

前の例と同じデータセットとシナリオが使用されます。

ただし、この例ではデータセットは変更されず、アプリケーションにロードされます。トランザクションが 4 月に発生したかどうかを判断する計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/10/2022',37.23
8189,'1/14/2022',17.17
8190,'1/20/2022',88.27
8191,'1/22/2022',57.42
8192,'2/1/2022',53.80
8193,'2/2/2022',82.06
8194,'2/20/2022',40.39
8195,'4/11/2022',87.21
8196,'4/13/2022',95.93
8197,'4/15/2022',45.89
8198,'4/25/2022',36.23
8199,'5/20/2022',25.66
```

```
8200,'5/22/2022',82.77
8201,'6/19/2022',69.98
8202,'6/22/2022',76.11
];
```

チャート オブジェクト

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

```
date
```

トランザクションが 4 月に行われるかどうかを計算するには、次のメジャーを作成します。

```
=inmonth(date,'04/01/2022', 0)
```

結果

関数の例

日付	=inmonth(date,'04/01/2022', 0)
1/10/2022	0
1/14/2022	0
1/20/2022	0
1/22/2022	0
2/1/2022	0
2/2/2022	0
2/20/2022	0
4/11/2022	-1
4/13/2022	-1
4/15/2022	-1
4/25/2022	-1
5/20/2022	0
5/22/2022	0
6/19/2022	0
6/22/2022	0

例 4 – シナリオ

ロード スクリプトと結果

概要

この例では、「Products」という名前のテーブルにデータセットがロードされます。テーブルには次の項目が含まれています。

- 製品 ID
- 製造日付
- コスト

2022 年 7 月に製造された商品が、設備の不具合により不良品となっていました。この問題は 2022 年 7 月 27 日に解決されました。

エンドユーザーは、製造された製品のステータスが「不具合」(ブール値が TRUE) または「不具合なし」(ブール値が FALSE) であったこと、およびその月に製造された製品のコストを表示するチャートを希望しています。

ロードスクリプト

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

```
=monthname(manufacture_date)
```

次のメジャーを作成します

- `=sum(cost_price)`
- `=if(only(inmonth(manufacture_date,makedate(2022,07,01),0)),'Defective','Faultless')`

1. メジャーの **[数値書式]** を **[通貨]** に設定します。
2. **[スタイル]** で **[合計]** をオフにします。

結果テーブル

monthname (manufacture_date)	=if(only(inmonth(manufacture_date,makedate (2022,07,01),0)),'Defective','Faultless')	sum(cost_ price)
2022 年 1 月	不具合なし	$54.40
2022 年 2 月	不具合なし	$145.69
2022 年 3 月	不具合なし	$53.80
2022 年 4 月	不具合なし	$82.06
2022 年 5 月	不具合なし	$127.60
2022 年 6 月	不具合なし	$141.82
2022 年 7 月	不具合	$214.64
2022 年 8 月	不具合なし	$147.46
2022 年 9 月	不具合なし	$84.21
2022 年 10 月	不具合なし	$163.91

inmonth() 関数は、各製品の製造日を評価するときにブール値を返します。2022 年 7 月に製造された製品の場合、inmonth() 関数はブール値 TRUE を返し、製品を「不具合」としてマークします。FALSE の値を返し、7 月に製造されなかった製品については、その製品に「不具合なし」のマークが付けられます。

inmonths

この関数は、タイムスタンプが基準日と同じ月、隔月、四半期、4 か月、または半年に該当するかどうかを確認します。タイムスタンプがその前後の期間に該当するか確認することもできます。

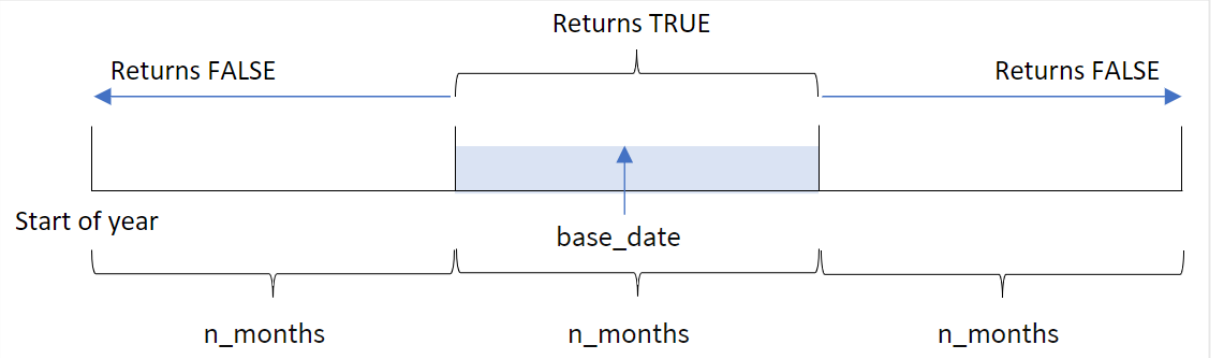
構文:

```
InMonths(n_months, timestamp, base_date, period_no [, first_month_of_year])
```

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

inmonths() 関数の図



inmonths() 関数は、指定された n_months 引数に基づいて年をセグメントに分割します。次に、評価する各タイムスタンプが base_date 引数と同じセグメントに当たるかどうかを評価します。ただし、period_no 引数が入力されると、関数はタイムスタンプが base_date の前と後のどちらの期間に入るかを決定します。

次の年のセグメントは、n_month 引数として関数で使用できます。

n_month 引数

期間	月数
月	1
隔月	2
四半期	3
4 か月	4
半年	6

使用に適しているケース

inmonths() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。inmonths() 関数を使用することにより、評価する期間を選択できます。例えば、ユーザーが特定の期間のその月、四半期、6 か月に製造された製品を特定できるようにします。

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

引数

引数	説明
n_months	期間を定義する月数。整数、または計算結果が整数になる数式で次のうちのいずれかでなければならない。1 (inmonth() 関数と同機能)、2 (2 か月)、3 (inquarter() 関数と同機能)、4 (4 か月)、6 (半年)。
timestamp	**base_date** と比較する日付。
base_date	期間の評価に使用する日付。
period_no	期間は、**period_no**、整数、計算結果が整数になる数式を使用して補正できます。値 0 は **base_date** を含む期間を示します。**period_no** の値が負の場合は過去の期間を、正の場合は将来の期間を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

次の値を使用して、first_month_of_year 引数に年の最初の月を設定できます。

first_month_of_year values

月	値
February	2
3 月	3
April	4
May	5
June	6
7 月	7
8 月	8
September	9
10 月	10
November	11
12 月	12

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
`inmonths(4, '01/25/2013', '04/25/2013', 0)`	TRUE を返します。日付と時刻の値 25/01/2013 は、01/01/2013 ~ 30/04/2013 までの 4 か月の期間内です。この期間には、base_date の値 25/04/2013 も含まれています。
`inmonths(4, '05/25/2013', '04/25/2013', 0)`	FALSE を返します。25/05/2013 は、前述の例にある期間の範囲外です。
`inmonths(4, '11/25/2012', '02/01/2013', -1 )`	TRUE を返します。period_no の値が -1 のため、検索期間を 4 か月前 (n-months の値) にずらして、01/09/2012 ~ 31/12/2012 までの期間にします。

例	結果
inmonths(4, '05/25/2006', '03/01/2006', 0, 3)	TRUE を返します。first_month_of_year の値が 3 に設定されているため、検索期間は 01/01/2006 ~ 30/04/2006 ではなく01/03/2006 ~ 30/07/2006 になります。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- トランザクションが 2022 年 5 月 15 日と同じ四半期に発生したかどうかを判断する追加の変数「in_months」を使用した先行する LOAD。

ロードスクリプト

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

Transactions:
    Load
       *,
       inmonths(3,date,'05/15/2022', 0) as in_months
       ;
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
```

```
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

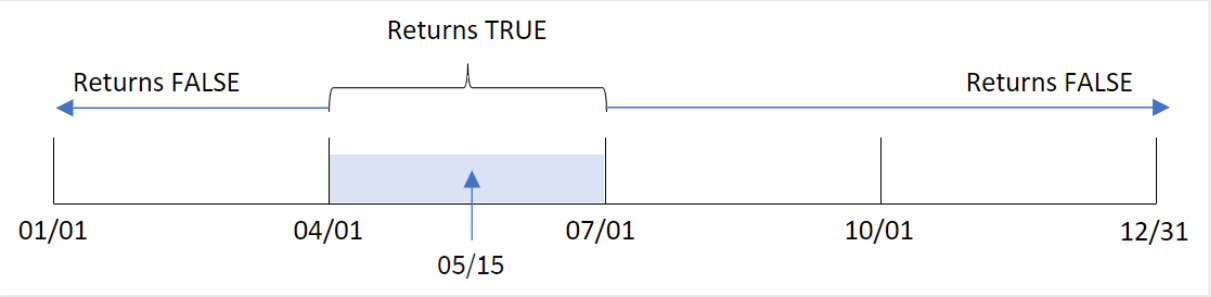
- date
- in_months

結果テーブル

日付	in_months
2/19/2022	0
3/7/2022	0
3/30/2022	0
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

[in_months] 項目は、inmonths() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は 3 で、年を四半期のセグメントに分割します。2 番目の引数は、評価される項目 (この例では日付項目) を識別します。3 番目の引数は 5 月 15 日のハード化された日付 (base_date) であり、0 の period_no が最終的な引数です。

四半期セグメントの inmonths() 関数の図



5 月は年の第 2 四半期に当たります。したがって、4 月 1 日 ~ 6 月 30 日に発生したトランザクションは、のブール値の結果を返します。これは、結果テーブルで検証されます。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- トランザクションが 2022 年 5 月 15 日より前の四半期に発生したかどうかを判断する追加の変数「previous_quarter」を使用した先行 Load。

ロードスクリプト

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

Transactions:
    Load
       *,
       inmonths(3,date,'05/15/2022', -1) as previous_quarter
       ;
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
```

```
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_quarter

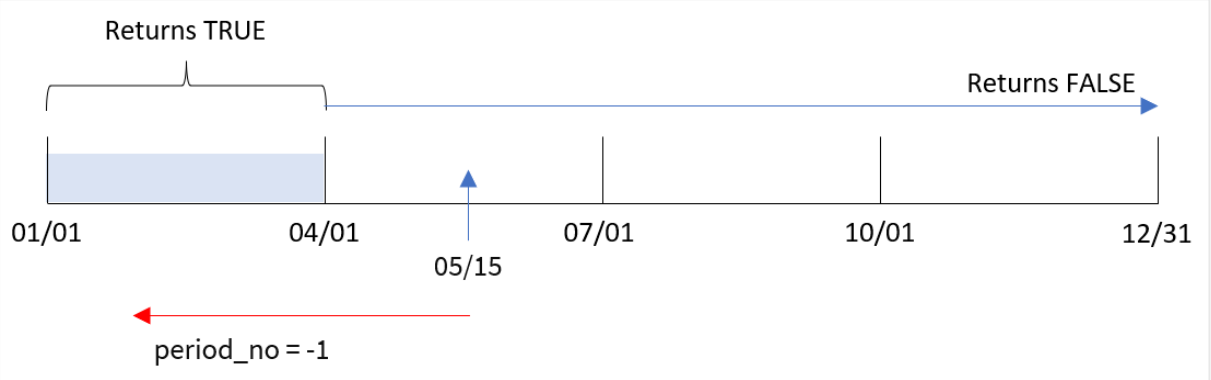
結果テーブル

日付	前の四半期
2/19/2022	-1
3/7/2022	-1
3/30/2022	-1
4/5/2022	0
4/16/2022	0
5/1/2022	0
5/7/2022	0
5/22/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0

日付	前の四半期
10/14/2022	0
10/29/2022	0

関数は、inmonths() で -1 を period_no 引数として使用することにより、トランザクションがその年の第 1 四半期に発生したかどうかを評価します。5 月 15 日は base_date であり、年の第 2 四半期に当たります (4 ~ 6 月)。

period_no が -1 に設定された四半期セグメントの inmonths() 関数の図



したがって、1 月 ~ 3 月に発生したトランザクションは、TRUE のブール値の結果を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- トランザクションが 2022 年 5 月 15 日と同じ四半期に発生したかどうかを判断する追加の変数「in_months」を使用した先行 Load。

この例では、組織ポリシーでは 3 月が会計期間の開始月に定められています。

ロードスクリプト

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

Transactions:
    Load
       *,
       inmonths(3,date,'05/15/2022', 0, 3) as in_months
       ;
Load
*
```

```
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_months

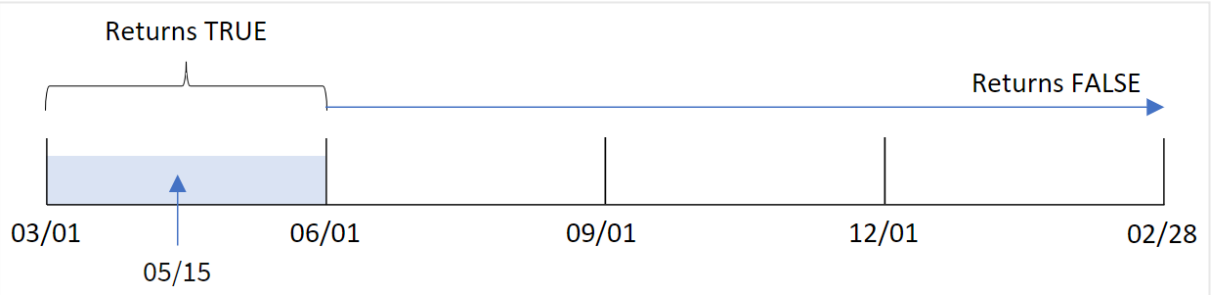
結果テーブル

日付	in_months
2/19/2022	0
3/7/2022	-1
3/30/2022	-1
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1
5/22/2022	-1
6/15/2022	0
6/26/2022	0

日付	in_months
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

inmonths() 関数の first_month_of_year 引数に 3 を使用することにより、関数は 3 月 1 日に年度を開始します。inmonths() 関数は、次にその年度を四半期に分割します。3 月 ~ 5 月、6 月 ~ 8 月、9 月 ~ 11 月、12 月 ~ 2 月。従って、5 月 15 日はその年の第 1 四半期 (3 月 ~ 5 月) に当たります。

3 月が年の最初の月に設定された inmonths() 関数の図



これらの月に発生したトランザクションは、ブール値結果 TRUE を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例ではデータセットは変更されず、アプリケーションにロードされます。トランザクションが 2022 年 5 月 15 日と同じ四半期に発生したかどうかを決定する計算は、アプリのチャートのメジャーとして作成されます。

ロード スクリプト

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

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,'2/19/2022',37.23
8189,'3/7/2022',17.17
8190,'3/30/2022',88.27
8191,'4/5/2022',57.42
8192,'4/16/2022',53.80
8193,'5/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/22/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- date

5 月 15 日までの同じ四半期にトランザクションが発生したかどうかを計算するには、次のメジャーを作成します。

```
=inmonths(3,date,'05/15/2022', 0)
```

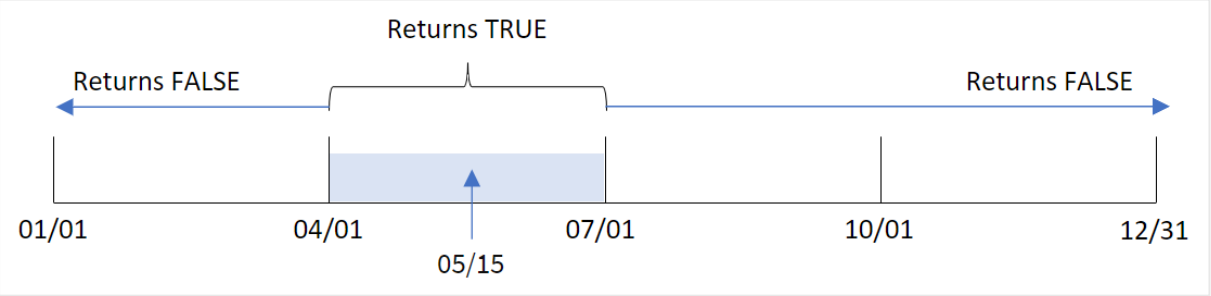
結果テーブル

日付	=inmonths(3,date,'05/15/2022', 0)
2/19/2022	0
3/7/2022	0
3/30/2022	0
4/5/2022	-1
4/16/2022	-1
5/1/2022	-1
5/7/2022	-1

日付	=inmonths(3,date,'05/15/2022', 0)
5/22/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

[in_months] 項目は、inmonths() 関数を使用することにより、チャートに作成されます。提供される最初の引数は 3 で、年を四半期のセグメントに分割します。2 番目の引数は、評価される項目 (この例では日付項目) を識別します。3 番目の引数は 5 月 15 日のハード化された日付 (base_date) であり、0 の period_no が最終的な引数です。

四半期セグメントの inmonths() 関数の図



5 月は年の第 2 四半期に当たります。したがって、4 月 1 日 ~ 6 月 30 日に発生したトランザクションは、のブール値の結果を返します。これは、結果テーブルで検証されます。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- テーブルには次の項目が含まれています。
    - 製品 ID
    - 製品の種類
    - 製造日付
    - コスト

エンドユーザーは、2021 年の最初のセグメントに製造された製品のコストを製品タイプ別に表示するチャートを希望しています。ユーザーはセグメントの長さを定義したいと考えています。

ロードスクリプト

```
SET vPeriod = 1;

Products:
Load
*
Inline
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'2/19/2022',37.23
8189,product D,'3/7/2022',17.17
8190,product C,'3/30/2022',88.27
8191,product B,'4/5/2022',57.42
8192,product D,'4/16/2022',53.80
8193,product D,'5/1/2022',82.06
8194,product A,'5/7/2022',40.39
8195,product B,'5/22/2022',87.21
8196,product C,'6/15/2022',95.93
8197,product B,'6/26/2022',45.89
8198,product C,'7/9/2022',36.23
8199,product D,'7/22/2022',25.66
8200,product D,'7/23/2022',82.77
8201,product A,'7/27/2022',69.98
8202,product A,'8/2/2022',76.11
8203,product B,'8/8/2022',25.12
8204,product B,'8/19/2022',46.23
8205,product B,'9/26/2022',84.21
8206,product C,'10/14/2022',96.24
8207,product D,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。

ロードスクリプトの開始時には、変数入力コントロールに関連付けられる変数 (vPeriod) が作成されます。

以下を実行します。

1. アセット パネルで、[**カスタム オブジェクト**] をクリックします。
2. [**Qlik ダッシュボード バンドル**] を選択し、**変数入力** オブジェクトを作成します。
3. チャート オブジェクトのタイトルを入力します。
4. [**変数**] で、[名前] に [**vPeriod**] を選択し、オブジェクトを [**ドロップ ダウン**] として表示するように設定します。
5. [**値**] で、[**ダイナミック**] 値をクリックします。以下を入力します。
   `='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.`
6. 新しいテーブルをシートに追加します。
7. プロパティ パネルの [**データ**] から、軸として `product_type` を追加します。
8. 次の数式をメジャーとして追加します:
   `=sum(if(inmonths($(vPeriod),manufacture_date,makedate(2022,01,01),0),cost_price,0))`
9. メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

product_type	=sum(if(inmonths($(vPeriod),manufacture_date,makedate (2022,01,01),0),cost_price,0))
製品 A	$88.27
製品 B	$37.23
製品 C	$17.17
製品 D	$0.00

`inmonths()` 関数は、ユーザー入力を引数として使用し、年の開始セグメントのサイズを定義します。関数は各製品の製造日付を `inmonths()` 関数の第 3 引数として渡します。`inmonths()` 関数で 1 月 1 日を第 3 引数にすると、製造日が開始セグメントに当たる製品は TRUE を返すため、sum 関数でそれらの製品のコストを加算することができます。

inmonthstodate

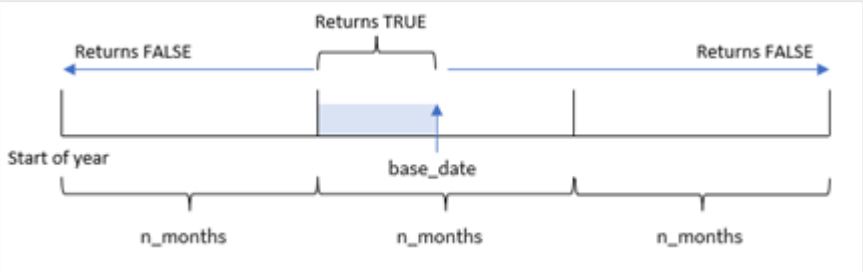
この関数は、タイムスタンプが、`base_date` の最後のミリ秒までの月、2 か月、四半期、4 か月、半年のいずれかの期間の範囲内か確認します。タイムスタンプがその前後の期間に該当するか確認することもできます。

構文:

```
InMonths (n_months, timestamp, base_date, period_no[, first_month_of_year ])
```

戻り値データ型： ブール値

inmonthstodate 関数の図。



引数

引数	説明
n_months	期間を定義する月数。整数、または計算結果が整数になる数式で次のうちのいずれかでなければならない。1 (inmonth() 関数と同機能)、2 (2 か月)、3 (inquarter() 関数と同機能)、4 (4 か月)、6 (半年)。
timestamp	**base_date** と比較する日付。
base_date	期間の評価に使用する日付。
period_no	期間は、**period_no**、整数、計算結果が整数になる数式を使用して補正できます。値 0 は **base_date** を含む期間を示します。**period_no** の値が負の場合は過去の期間を、正の場合は将来の期間を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

inmonthstodate() 関数では、base_date がそれが属する特定の年度セグメントの終了点として機能します。

例えば、1 年を三分割して、base_date を 5 月 15 日とした場合、1 月 ~ 4 月のタイムスタンプは、ブール値として FALSE を返します。5 月 1 日 ~ 5 月 15 日の日付は TRUE を返します。年のそれ以外の日付は FALSE を返します。

inmonthstodate 関数のブール値結果範囲の図。

次の年のセグメントは、n_month 引数として関数で使用できます。

n_month 引数

期間	月数
月	1
隔月	2
四半期	3
3 か月ごと	4
半年	6

使用に適しているケース

inmonthstodate() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。inmonthstodate() 関数を使用することにより、評価する期間を選択できます。例えば、ユーザーが特定の期間までの月、四半期、6 か月に製造された製品を特定できるようにする入力変数を提供します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
inmonthstodate(4, '01/25/2013', '04/25/2013', 0)	timestamp の値 01/25/2013 は、01/01/2013 から 04/25/2013 末までに渡る 4 か月の範囲内であるため、True を返します。この期間には、base_date、04/25/2013 の値 も含まれています。
inmonthstodate(4, '04/26/2013', '04/25/2006', 0)	04/26/2013 は、上の例の期間の範囲外であるため、False を返します。
inmonthstodate(4, '09/25/2005', '02/01/2006', -1)	period_no の値 -1が、比較対象の期間が 4 か月 (n-months の値) ずれて、検索期間が 01/09/2005 ~ 02/01/2006 になるため、True を返します。
inmonthstodate(4, '04/25/2006', '06/01/2006', 0, 3)	first_month_of_year の値に 3 が設定されており、比較対象の期間が 05/01/2006 ~ 06/01/2006 ではなく、03/01/2006 ~ 06/01/2006 になっているため、True を返します。

例 1 – 追加の引数なし

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数 ((MM/DD/YYYY)) 形式の日付項目。
- 次を含む先行する LOAD ステートメント:
    - 項目 [in_months_to_date] として設定された inmonthstodate() 関数。これは、2022 年 5 月 15 日までの四半期に発生したトランザクションを決定します。

ロード スクリプト

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

Transactions:
    Load
    *,
    inmonthstodate(3,date,'05/15/2022', 0) as in_months_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_months_to_date

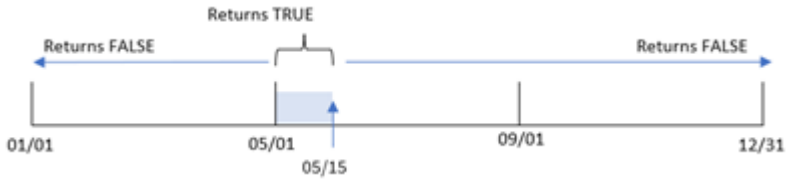
結果テーブル

日付	in_months_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

[in_months_to_date] 項目は、inmonthstodate() 関数を使用して、先行 Load ステートメントで作成されます。

提供される最初の引数は 3 で、年を四半期のセグメントに分割します。2 番目の引数は、評価される項目を識別します。第 3 引数は、5 月 15 日のハードコードされた日付です。これは base_date で、セグメントの終了境界を定義します。0 の period_no が最終引数です。

追加の引数がない inmonthstodate 関数の図。



4 月 1 日 ~ 5 月 15 日に発生したトランザクションは、TRUE のブール値の結果を返します。その期間外のトランザクション日付は FALSE を返します。

例 2 – period_no

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例のタスクは、トランザクションが 5 月 15 日より四半期前に発生するかどうかを決定する項目 [previous_qtr_to_date] を作成します。

ロードスクリプト

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

Transactions:
    Load
    *,
    inmonthstodate(3,date,'05/15/2022', -1) as previous_qtr_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
```

```
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_qtr_to_date

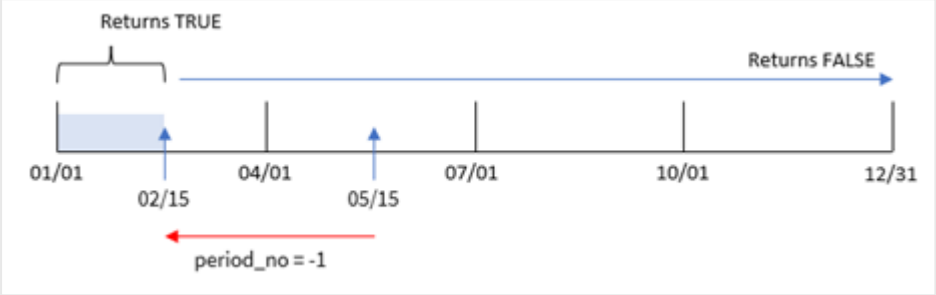
結果テーブル

日付	previous_qtr_to_date
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

inmonthstodate() 関数で -1 を period_no 引数として使用することにより、関数は、比較対象年度の境界を四半期ずらします。

5 月 15 日はその年の第 2 四半期に分類されるため、セグメントは最初は 4 月 1 日 ~ 5 月 15 日に相当します。period_no 引数は、このセグメントを過去に 3 か月ずらします。日付境界が 1 月 1 日から 2 月 15 日になります。

period_no が -1 に設定された inmonthstodate 関数の図。



したがって、1 月 1 日 ~ 2 月 15 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

この例では、組織ポリシーでは 3 月が会計期間の開始月に定められています。

2022 年 5 月 15 日まで同四半期に発生したトランザクションを決定する項目 [in_months_to_date] を作成します。

ロードスクリプト

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

Transactions:
    Load
    *,
    inmonthstodate(3,date,'05/15/2022', 0,3) as in_months_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
```

```
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_months_to_date

結果テーブル

日付	previous_qtr_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

日付	previous_qtr_to_date
9/26/2022	0
10/14/2022	0
10/29/2022	0

inmonthstodate() 関数の first_month_of_year 引数に 3 を使用することにより、関数は 3 月 1 日に年度を開始し、次に入力された第 1 引数に基づいてその年度を四半期に分割します。したがって、四半期セグメントは次のとおりです。

- 3 月 ~ 5 月
- 6 月 ~ 8 月
- 9 月 ~ 11 月
- 12 月 ~ 2 月

次に、5 月 15 日の base_date は、5 月 15 日に終了境界を設定することにより、3 月 ~ 5 月の四半期をセグメント化します。

3 月が年の最初の月に設定された inmonthstodate 関数の図。

したがって、3 月 1 日 ~ 5 月 15 日に発生したトランザクションは TRUE のブール値の結果を返しますが、これらの境界外の日付のトランザクションは FALSE の値を返します。

例 4 – チャートの例

ロードスクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

この例では、データセットは変更されず、アプリにロードされます。タスクは、アプリのチャートのメジャーとして、トランザクションが 5 月 15 日と同じ四半期に発生したかどうかを判断する計算を作成することです。

ロードスクリプト

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

Transactions:
Load
*
```

```
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

```
date
```

5 月 15 日までの同じ四半期にトランザクションが発生したかどうかを計算するには、次のメジャーを作成します。

```
=inmonthstodate(3,date,'05/15/2022', 0)
```

結果テーブル

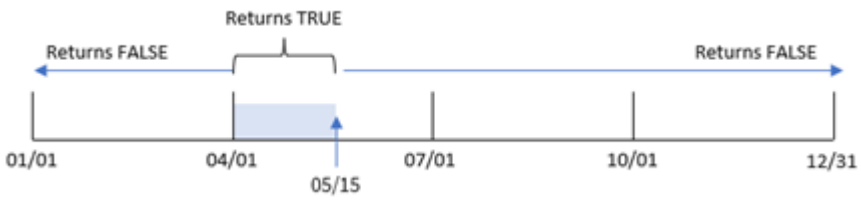
日付	=inmonthstodate(3,date,'05/15/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0

日付	=inmonthstodate(3,date,'05/15/2022', 0)
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

'in_months_to_date メジャーは、inmonthstodate() 関数を使用してチャートに作成されます。

提供される最初の引数は 3 で、年を四半期のセグメントに分割します。2 番目の引数は、評価される項目を識別します。第 3 引数は、5 月 15 日のハードコードされた日付です。これは base_date で、セグメントの終了境界を定義します。0 の period_no が最終引数です。

四半期セグメントの inmonthstodate 関数の図。



4 月 1 日 ~ 5 月 15 日に発生したトランザクションは、TRUE のブール値の結果を返します。そのセグメント外のトランザクション日付は FALSE を返します。

例 5 – シナリオ

ロードスクリプトと結果

概要

この例では、「Sales」という名前のテーブルにデータセットがロードされます。テーブルには次の項目が含まれています。

- 製品 ID
- 製品の種類

- 販売日付
- 販売価格

エンドユーザーは、2022 年 12 月 24 日までの期間に販売された製品の売上を、製品タイプ別に表示するグラフを希望しています。ユーザーは期間の長さを定義したいと考えています。

ロード スクリプト

```
SET vPeriod = 1;

Products:
Load
*
Inline
[
product_id,product_type,sales_date,sales_price
8188,product A,'9/19/2022',37.23
8189,product D,'10/27/2022',17.17
8190,product C,'10/30/2022',88.27
8191,product B,'10/31/2022',57.42
8192,product D,'11/16/2022',53.80
8193,product D,'11/28/2022',82.06
8194,product A,'12/2/2022',40.39
8195,product B,'12/5/2022',87.21
8196,product C,'12/15/2022',95.93
8197,product B,'12/16/2022',45.89
8198,product C,'12/19/2022',36.23
8199,product D,'12/22/2022',25.66
8200,product D,'12/23/2022',82.77
8201,product A,'12/24/2022',69.98
8202,product A,'12/24/2022',76.11
8203,product B,'12/26/2022',25.12
8204,product B,'12/27/2022',46.23
8205,product B,'12/27/2022',84.21
8206,product C,'12/28/2022',96.24
8207,product D,'12/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。

ロード スクリプトの開始時には、変数入力コントロールに関連付けられる変数 (vPeriod) が作成されます。

以下を実行します。

1. アセット パネルで、[**カスタム オブジェクト**] をクリックします。
2. [**Qlik ダッシュボード バンドル**] を選択して、シートに [**変数入力**] を追加します。
3. チャートのタイトルを入力します。
4. [**変数**] で、[名前] に [**vPeriod**] を選択し、オブジェクトを [**ドロップ ダウン**] として表示するように設定します。

5. [**値**] で、[**ダイナミック**] 値をクリックします。以下を入力します。
   `='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.`
6. 新しいテーブルをシートに追加します。
7. プロパティパネルの [**データ**] から、軸として `product_type` を追加します。
8. 次の数式をメジャーとして追加します:
   `=sum(if(inmonthstodate($(vPeriod),sales_date,makedate(2022,12,24),0),sales_price,0))`
9. メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

product_type	=sum(if(inmonthstodate($(vPeriod),sales_date,makedate (2022,12,24),0),sales_price,0))
製品 A	$186.48
製品 B	$190.52
製品 C	$220.43
製品 D	$261.46

`inmonthstodate()` 関数は、ユーザー入力を引数として使用し、年の開始セグメントのサイズを定義します。

関数は各製品の販売日付を `inmonthstodate()` 関数の第 3 引数として渡します。`inmonthstodate()` 関数で 12 月 24 日を第 3 引数として使用することにより、12 月 24 日までの定義済み期間に発生した販売日付を持つ製品は TRUE のブール値を返します。sum 関数はこれらの製品の売上を合計します。

inmonthtodate

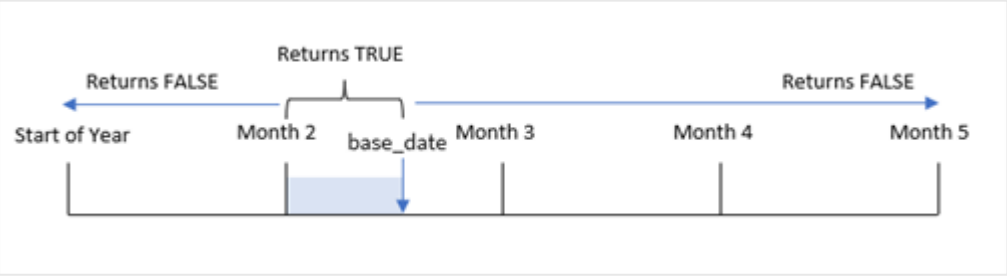
basedate の最後のミリ秒まで **basedate** を含む月に **date** がある場合に True を返します。

構文:

```
InMonthToDate (timestamp, base_date, period_no)
```

戻り値データ型: ブール値

inmonthtodate 関数の図。



`inmonthtodate()` 関数は、選択した月をセグメントとして特定します。開始境界は月の始めです。終了境界は、月の後半の日付として設定できます。次に、日付セットがこのセグメントに当てはまるかどうかを決定し、ブール値 TRUE または FALSE を返します。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	月の評価に使用する日付。
period_no	月は **period_no** によって補正することができます。**period_no** は整数で、値 0 は **base_date** を含む月を示します。**period_no** の値が負の場合は過去の月を、正の場合は将来の月を示します。

使用に適しているケース

inmonthtodate() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。inmonthtodate() 関数は、対象の日付を含み、日付がその月に発生したかどうかに応じて、集計または計算を返します。

例えば、inmonthtodate() 関数を使用して、特定の日付を含む月に製造されたすべての機器を識別することができます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
inmonthtodate ('01/25/2013', '25/01/2013', 0)	True を返します
inmonthtodate ('01/25/2013', '24/01/2013', 0)	False を返します
inmonthtodate ('01/25/2013', '28/02/2013', -1)	True を返します

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数 (MM/DD/YYYY) 形式で提供される日付項目。
- 次を含む先行する LOAD ステートメント:
    - 項目 [in_month_to_date] として設定されている inmonthtodate() 関数。これは、2022 年の 7 月 1 日 ~ 7 月 26 日に発生したトランザクションを決定します。

ロード スクリプト

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

Transactions:
    Load
    *,
    inmonthtodate(date,'07/26/2022', 0) as in_month_to_date
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_month_to_date

結果テーブル

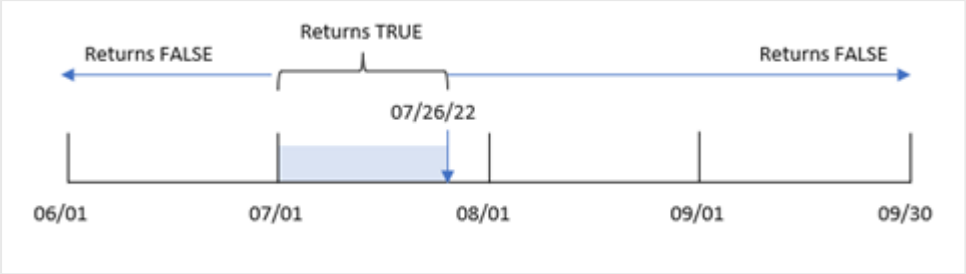
日付	in_month_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	-1
7/22/2022	-1
7/23/2022	-1
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

[in_month_to_date] 項目は、inmonthtodate() 関数を使用して、前の Load ステートメントで作成されます。

最初の引数は、評価される項目を識別します。第 2 引数はハードコード化された日付 7 月 26 日で、これは base_date です。この base_date 引数は、どの月がセグメント化されるか、そのセグメントの終了境界を特定します。

0 の period_no は最後の引数です。これは、関数がセグメント化された月の前後の月を比較していないことを意味します。

追加の引数がない inmonthtodate 関数の図。



その結果、7 月 1 日 ~ 7 月 26 日に発生したトランザクションは、ブール値結果 TRUE を返します。7 月 26 日より後の 7 月の日付に発生したトランザクションはブール値 FALSE を返し、その年の他の月の日付も同様です。

例 2 – period_no

ロード スクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

この例のタスクは、トランザクションが 7 月 1 日 ~ 7 月 26 日より6 か月前に発生するかどうかを決定する項目 [six_months_prior] を作成することです。

ロード スクリプト

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

Transactions:
    Load
    *,
    inmonthtodate(date,'07/26/2022', -6) as six_months_prior
    ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
```

```
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

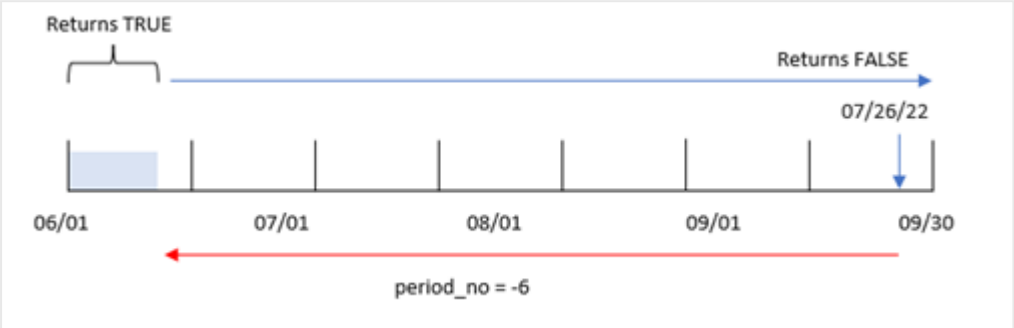
- date
- six_months_prior

結果テーブル

日付	six_months_prior
1/7/2022	-1
1/19/2022	-1
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

inmonthtodate() 関数で -6 を period_no 引数として使用することにより、比較対象年度の境界を 6 か月ずらします。最初、月セグメントは 7 月 1 日 ~ 7 月 26 日と同値です。次に period_no はマイナス 6 か月このセグメントをオフセットし、日付境界が 1 月 1 日 ~ 1 月 26 日にずれます。

period_no が -6 に設定された inmonthtodate 関数の図。



その結果、1 月 1 日 ~ 1 月 26 日に発生したトランザクションは、ブール値結果 TRUE を返します。

例 3 – チャートの例

ロードスクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

この例では、データセットは変更されず、アプリにロードされます。タスクは、アプリのチャートのメジャーとして、トランザクションが 7 月 1 日から 7 月 26 日の間に発生したかどうかを判断する計算を作成することです。

ロードスクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
```

```
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

```
date
```

7 月 1 日 ~ 7 月 26 日にトランザクションが発生したかどうかを計算するには、次のメジャーを作成します。

```
=inmonthtodate(date,'07/26/2022', 0)
```

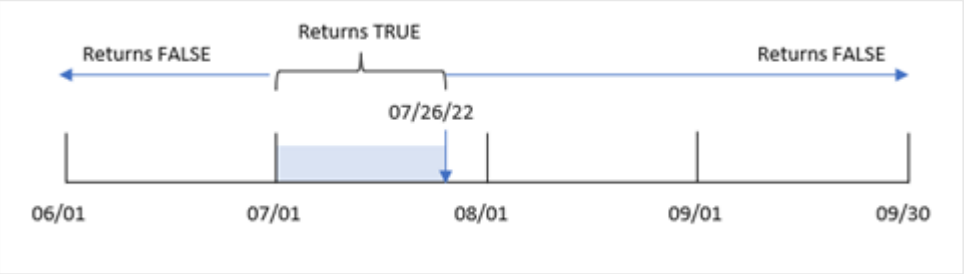
結果テーブル

日付	=inmonthtodate(date,'07/26/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	-1
7/22/2022	-1
7/23/2022	-1
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

[in_month_to_date] 項目 メジャーは、inmonthtodate() 関数を使用することにより、チャートに作成されます。

最初の引数は、評価される項目を識別します。第 2 引数はハードコード化された日付 7 月 26 日で、これは base_date です。この base_date 引数は、どの月がセグメント化されるか、そのセグメントの終了境界を特定します。period_no の 0 が最終引数です。つまり、関数がセグメント化された月の前後の月を比較していないことを意味します。

追加の引数がない inmonthtodate 関数の図。



その結果、7 月 1 日 ~ 7 月 26 日に発生したトランザクションは、ブール値結果 TRUE を返します。7 月 26 日より後の 7 月の日付に発生したトランザクションはブール値 FALSE を返し、その年の他の月の日付も同様です。

例 4 – シナリオ

ロード スクリプトと結果

概要

この例では、「Products」という名前のテーブルにデータセットがロードされます。テーブルには次の項目が含まれています。

- 製品 ID
- 製造日付
- コスト

2022 年 7 月に製造された商品が、設備の不具合により不良品となっていました。この問題は 2022 年 7 月 27 日に解決されました。

エンド ユーザーは、製造された製品のステータスが「不具合」(ブール値が TRUE) または「不具合なし」(ブール値が FALSE) であったこと、およびその月に製造された製品のコストを表示するチャートを希望しています。

ロード スクリプト

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
```

```
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- `=monthname(manufacture_date)`
- `=if(Inmonthtodate(manufacture_date,makedate(2022,07,26),0),'Defective','Faultless')`

製品の合計コストを計算するには、メジャーを作成します。

```
=sum(cost_price)
```

メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

monthname (manufacture_date)	if(Inmonthtodate(manufacture_date,makedate (2022,07,26),0),'Defective','Faultless')	Sum(cost_ price)
2022年1月	不具合なし	$54.40
2022年2月	不具合なし	$145.69
2022年3月	不具合なし	$53.80
2022年4月	不具合なし	$82.06
2022年5月	不具合なし	$127.60
2022年6月	不具合なし	$141.82
2022年7月	不具合	$144.66
2022年7月	不具合なし	$69.98
2022年8月	不具合なし	$147.46
2022年9月	不具合なし	$84.21

monthname (manufacture_date)	if(Inmonthtodate(manufacture_date,makedate (2022,07,26),0),'Defective','Faultless')	Sum(cost_ price)
2022 年 10 月	不具合なし	$163.91

inmonthtodate() 関数は、各製品の製造日を評価するときにブール値を返します。

ブール値 TRUE を返す日付については、製品が「不具合あり」とマークされています。FALSE の値を返し、7 月 26 日までの月に製造されなかった製品については、その製品に「不具合なし」のマークが付けられます。

inquarter

この関数は、**timestamp** が **base_date** を含む四半期に含まれる場合、True を返します。

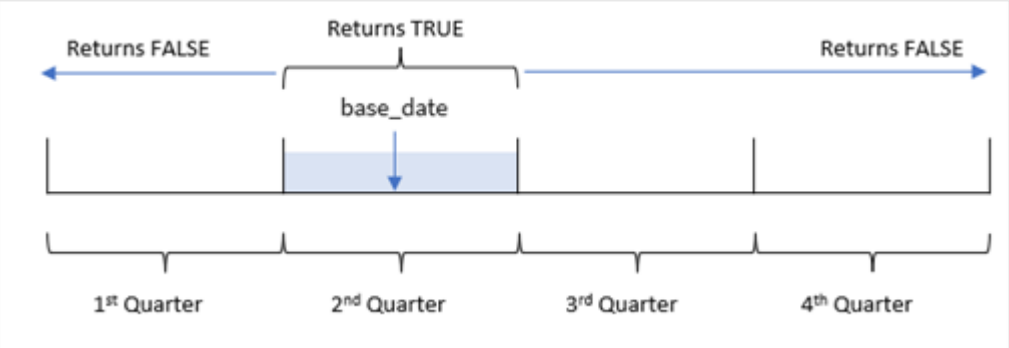
構文:

```
InQuarter (timestamp, base_date, period_no[, first_month_of_year])
```

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

inquarter() 関数範囲の図



言い換えると、inquarter() 関数は 1 月 1 日 ~ 12 月 31 日の年を 4 つの四半期に等分します。first_month_of_year 引数を使ってアプリの開始時としてどの月を設定するかを変更することができ、その引数に基づいて四半期が変化します。base_date、関数は関数の比較対象としてどの四半期を使用すべきかを特定します。最後に、日付値をその四半期セグメントと比較すると、関数はブール値の結果を返します。

使用に適しているケース

inquarter() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。これは、選択した四半期でその日付が発生したかどうかに応じて、集計または計算を返します。

例えば、inquarter() 関数は、機器が製造された日付に基づいて、その四半期セグメントに製造されたすべての機器を特定するために使用できます。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	四半期の評価に使用する日付。
period_no	四半期は **period_no** によって補正することができます。**period_no** は整数で、値 0 は **base_date** を含む四半期を示します。**period_no** の値が負の場合は過去の四半期を、正の場合は将来の四半期を示します。
first_month_of_year	事業年度が1月以外の月に始まる場合は、**first_month_of_year** で 2 から12 の間の値を指定します。

次の値を使用して、first_month_of_year 引数に年の最初の月を設定できます。

first_month_of_year values

月	値
February	2
3 月	3
April	4
May	5
June	6
7 月	7
8 月	8
September	9
10 月	10
November	11
12 月	12

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
inquarter ('01/25/2013', '01/01/2013', 0)	TRUE を返します
inquarter ('01/25/2013', '04/01/2013', 0)	FALSE を返します
inquarter ('01/25/2013', '01/01/2013', -1)	FALSE を返します
inquarter ('12/25/2012', '01/01/2013', -1)	TRUE を返します
inquarter ('01/25/2013', '03/01/2013', 0, 3)	FALSE を返します
inquarter ('03/25/2013', '03/01/2013', 0, 3)	TRUE を返します

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- inquarter() 関数が [in_quarter] 項目として設定された先行する LOAD で、2022 年 5 月 15 日と同じ四半期に発生したトランザクションを決定します。

ロードスクリプト

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

Transactions:
    Load
       *,
       inquarter (date,'05/15/2022', 0) as in_quarter
       ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
```

```
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_quarter

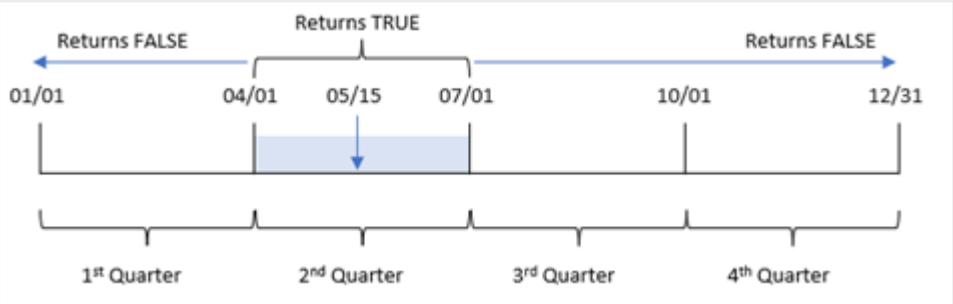
結果テーブル

日付	in_quarter
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0

日付	in_quarter
10/14/2022	0
10/29/2022	0

[in_quarter] 項目は、inquarter() 関数を使用して、前の Load ステートメントで作成されます。最初の引数は、評価される項目を識別します。第 2 引数は 5 月 15 日のハードコード化された日付であり、比較対象として定義 sる3四半期を特定します。0 の period_no は最後の引数であり、inquarter() 関数がセグメント化された四半期の前後の四半期を比較していないことを意味します。

基準日が 5 月 15 日の inquarter() 関数の図



4 月 1 日 ~ 6 月 30 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- inquarter() 関数が [previous_quarter] 項目として設定された先行 load で、2022 年 5 月 15 日の前の四半期に発生したトランザクションを決定します。

ロードスクリプト

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

Transactions:
    Load
       *,
       inquarter (date,'05/15/2022', -1) as previous_qtr
       ;
Load
*
Inline
[
```

```
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_qtr

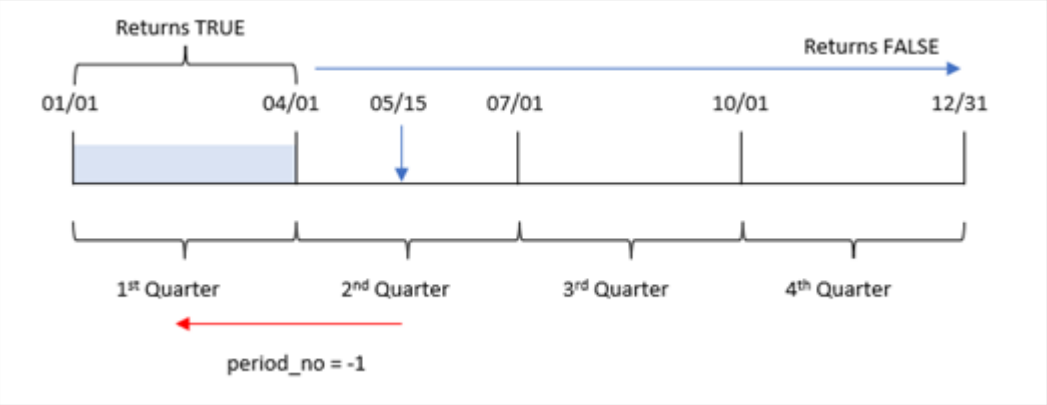
結果テーブル

日付	previous_qtr
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	-1
3/16/2022	-1
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0

日付	previous_qtr
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

inquarter() 関数で -1 を period_no 引数として使用することにより、比較対象四半期の境界を1四半期ずらします。5 月 15 日はその年の第 2 四半期に分類されるため、セグメントは最初は 4 月 1 日 ~ 6 月 30 日に相当します。period_no は、このセグメントを 3 か月前にオフセットし、日付の境界を1月 1 日 ~ 3 月 30 日にします。

基準日が 5 月 15 日の inquarter() 関数の図



したがって、1 月 1 日 ~ 3 月 30 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- inquarter() 関数が [in_quarter] 項目として設定された先行 load で、2022 年 5 月 15 日と同じ四半期に発生したトランザクションを決定します。

ただしこの例では、組織ポリシーでは 3 月が会計期間の開始月に定められています。

ロード スクリプト

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

Transactions:
    Load
       *,
       inquarter (date,'05/15/2022', 0, 3) as in_quarter
       ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

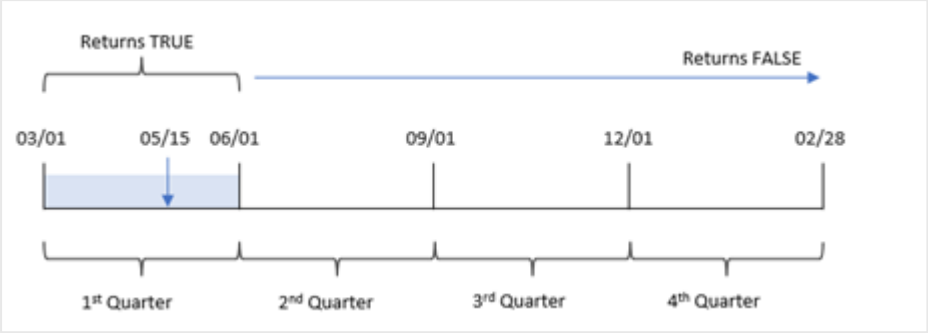
- date
- previous_qtr

結果テーブル

日付	previous_qtr
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

`inquarter()` 関数の `first_month_of_year` 引数に `3` を使用することにより、関数は 3 月 1 日に年度を開始し、その後その年度を四半期に分割します。四半期別セグメントはしたがって、3 ~ 5月、6 ~ 8月、9 ~ 11月、12 ~ 2月になります。5 月 15 日の `base_date` は 3 ~ 5月四半期を関数の比較対象四半期として設定しています。

3 月が年の最初の月に設定された inquarter() 関数の図



したがって、3 月 1 日 ~ 5 月 31 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- inquarter() 関数が [in_quarter] 項目として設定された先行 load で、2022 年 5 月 15 日と同じ四半期に発生したトランザクションを決定します。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
```

```
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- date

5 月 15 日までの同じ四半期にトランザクションが発生したかどうかを計算するには、次のメジャーを作成します。

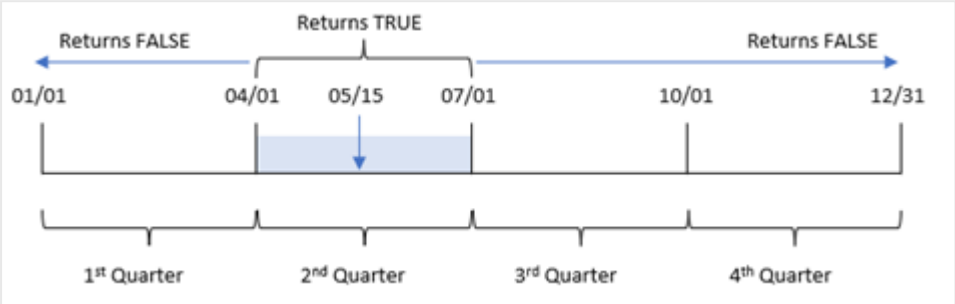
```
=inquarter(date,'05/15/2022', 0)
```

結果テーブル

日付	in_quarter
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	-1
6/15/2022	-1
6/26/2022	-1
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

in_quarter メジャーは、inquarter() 関数を使用してチャートに作成されます。最初の引数は、評価される項目を識別します。第 2 引数は 5 月 15 日のハードコード化された日付であり、比較対象として定義 sる四半期を特定します。0 の period_no は最後の引数であり、inquarter() 関数がセグメント化された四半期の前後の四半期を比較していないことを意味します。

基準日が 5 月 15 日の inquarter() 関数の図



4 月 1 日 ~ 6 月 15 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- テーブルには次の項目が含まれています。
    - 製品 ID
    - 製品の種類
    - 製造日付
    - コスト

機器のエラーにより、2022 年 5 月 15 日の四半期に製造された製品に欠陥があることが確認されています。エンドユーザーは、製造された製品のステータスが「不具合」または「不具合なし」であったこと、およびその四半期に製造された製品のコストを表示するチャートを希望しています。

ロードスクリプト

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
```

```
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

```
=quartername(manufacture_date)
```

次のメジャーを作成します:

- `inquarter()` 関数を使って、不具合のある製品とない製品を特定する `=if(only(InQuarter(manufacture_date,makedate(2022,05,15),0)),'Defective','Faultless')`。
- 各製品の合計コストを示す `=sum(cost_price)`。

次の手順を実行します。

1. メジャーの **[数値書式]** を **[通貨]** に設定します。
2. **[スタイル]** で **[合計]** をオフにします。

結果テーブル

quartername (manufacture_date)	=if(only(InQuarter(manufacture_date,makedate (2022,05,15),0)),'Defective','Faultless')	Sum (cost_ price)
2022 年 1 月 ~ 3 月	不具合なし	253.89
2022 年 4 月 ~ 6 月	不具合	351.48
2022 年 7 月 ~ 9 月	不具合なし	446.31
2022 年 10 月 ~ 12 月	不具合なし	163.91

`inquarter()` 関数は、各製品の製造日を評価するときにブール値を返します。5 月 15 日を含む四半期に製造された製品の場合、`inquarter()` 関数はブール値 TRUE を返し、製品を「不具合」としてマークします。FALSE の値を返し、その四半期に製造されなかった製品については、その製品に「不具合なし」のマークが付けられます。

inquartertodate

この関数は、**timestamp** が **base_date** のミリ秒単位まで正確に **base_date** を含む四半期の範囲内にある場合、True を返します。

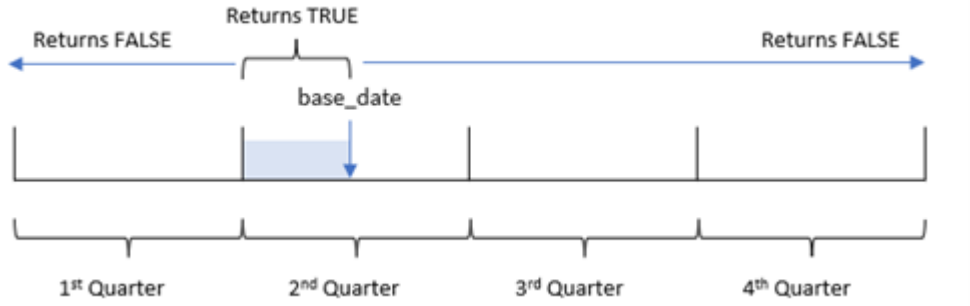
構文:

```
InQuarterToDate (timestamp, base_date, period_no [, first_month_of_year])
```

戻り値データ型: ブール値

> *Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。*

inquartertodate 関数の図



inquartertodate() 関数は、1 年を 1 月 1 日 ~ 12 月 31 日 (またはユーザーが定義する年の開始日とそれに対応する終了日) の間の 4 つの等しい四半期に分割します。base_date を使用して、関数は特定の四半期を分割し、base_date はその四半期セグメントの四半期と最大許容日付の両方を識別します。最後に、指定された日付値をそのセグメントと比較すると、関数はブール値の結果を返します。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	四半期の評価に使用する日付。
period_no	四半期は **period_no** によって補正することができます。**period_no** は整数で、値 0 は **base_date** を含む四半期を示します。**period_no** の値が負の場合は過去の四半期を、正の場合は将来の四半期を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

使用に適しているケース

inquartertodate() 関数はブール値の結果を返します。通常、このタイプの関数は if 式の条件として使用されます。inquartertodate() 関数は、評価された日付が問題の日付を含む四半期に発生したかどうかに応じて、集計または計算を返します。

例えば、inquartertodate() 関数を使用して、特定の日付を含む四半期に製造されたすべての機器を識別することができます。

関数の例

例	結果
inquartertodate ('01/25/2013', '03/25/2013', 0)	timestamp の値 01/25/2013 が、base_date の値 03/25/2013 が入っている 01/01/2013 ~ 03/25/2013 の 3 か月に当たるため、TRUE を返します。
inquartertodate ('04/26/2013', '03/25/2013', 0)	04/26/2013 は、前の例の期間の範囲外であるため、FALSE を返します。
inquartertodate ('02/25/2013', '06/09/2013', -1)	period_noの値 -1 が検索期間を1期間 (3 か月 = 1 年の 1 四半期) 戻るため、TRUE を返します。これにより、検索期間は 01/01/2013 ~ 03/09/2013 となります。
inquartertodate ('03/25/2006', '04/15/2006', 0, 2)	first_month_of_year の値が 2 に設定され、検索期間が 04/01/2006 ~ 04/15/2006 ではなく02/01/2006 ~ 04/15/2006 になるため、TRUE が返されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- 2022 年 5 月 15 日までの四半期に発生したトランザクションを決定する項目 [in_quarter_to_date] の作成。

ロード スクリプト

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

Transactions:
    Load
        *,
        inquartertodate(date,'05/15/2022', 0) as in_quarter_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_quarter_to_date

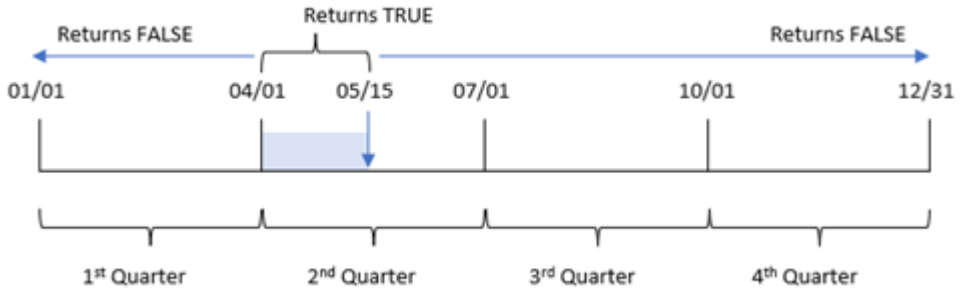
結果テーブル

日付	in_quarter_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0

日付	in_quarter_to_date
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

[in_quarter_to_date] 項目は、inquartertodate() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は、評価される項目を識別します。2 番目の引数は、5 月 15 日のハードコードされた日付です。これは base_date で、セグメント化する四半期を識別し、そのセグメントの終了境界を定義します。period_no の 0 は最後の引数です。つまり、関数がセグメント化された四半期の前後の四半期を比較していないということです。

inquartertodate 関数の図、追加の引数なし



4 月 1 日 ~ 5 月 15 日に発生したトランザクションは、TRUE のブール値の結果を返します。5 月 16 日以降の取引日は FALSE を返し、4 月 1 日より前の取引も同様です。

例 2 – period_no

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 2022 年 5 月 15 日に終了する四半期セグメント前の四半期全体で発生したトランザクションを決定する項目 [previous_qtr_to_date] の作成。

ロード スクリプト

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

Transactions:
    Load
        *,
        inquartertodate(date,'05/15/2022', -1) as previous_qtr_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

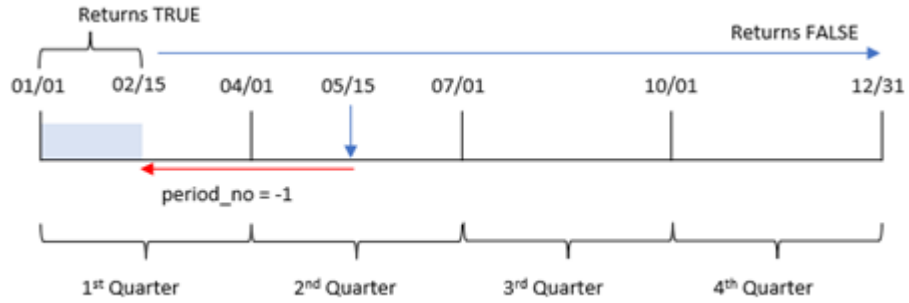
- date
- previous_qtr_to_date

結果テーブル

日付	previous_qtr_to_date
1/7/2022	-1
1/19/2022	-1
2/5/2022	-1
2/28/2022	0
3/16/2022	0
4/1/2022	0
5/7/2022	0
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

period_no 値 -1 は、inquartertodate () 関数が入力四半期セグメントを前の四半期と比較することを示します。5 月 15 日はその年の第 2 四半期に分類されるため、セグメントは最初は 4 月 1 日 ~ 5 月 15 日に相当します。period_no は、このセグメントを 3 か月前にオフセットし、日付の境界を 1 月 1 日 ~ 2 月 15 日にします。

inquartertodate 関数の図、period_no の例



したがって、1 月 1 日 ~ 2 月 15 日に発生したトランザクションは、TRUEのブール値の結果を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 2022 年 5 月 15 日まで同四半期に発生したトランザクションを決定する項目 [in_quarter_to_date] の作成。

この例では、3 月を会計年度の最初の月として設定します。

ロードスクリプト

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

Transactions:
    Load
        *,
        inquartertodate(date,'05/15/2022', 0,3) as in_quarter_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
```

```
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_quarter_to_date

結果テーブル

日付	in_quarter_to_date
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	-1
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0

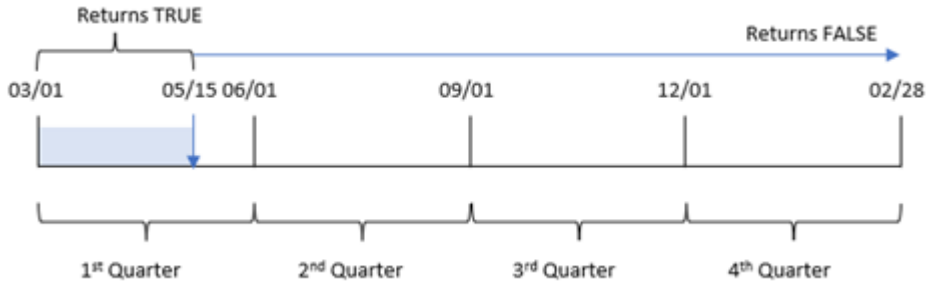
日付	in_quarter_to_date
9/26/2022	0
10/14/2022	0
10/29/2022	0

inquartertodate() 関数の first_month_of_year 引数に 3 を使用することにより、関数は 3 月 1 日に年度を開始し、その後その年度を四半期に分割します。したがって、四半期セグメントは次のとおりです。

- 3 月 ~ 5 月
- 6 月 ~ 8 月
- 9 月 ~ 11 月
- 12 月 ~ 2 月

次に、5 月 15 日の base_date は、5 月 15 日に終了境界を設定することにより、3 月 ~ 5 月の四半期をセグメント化します。

inquartertodate 関数の図、first_month_of_year の例



したがって、3 月 1 日 ~ 5 月 15 日に発生したトランザクションは TRUE のブール値の結果を返しますが、これらの境界外の日付のトランザクションは FALSE の値を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。5 月 15 日と同じ四半期に発生したトランザクションを決定する計算は、チャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
```

```
Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

次のメジャーを作成します:

```
=inquartertodate(date,'05/15/2022', 0)
```

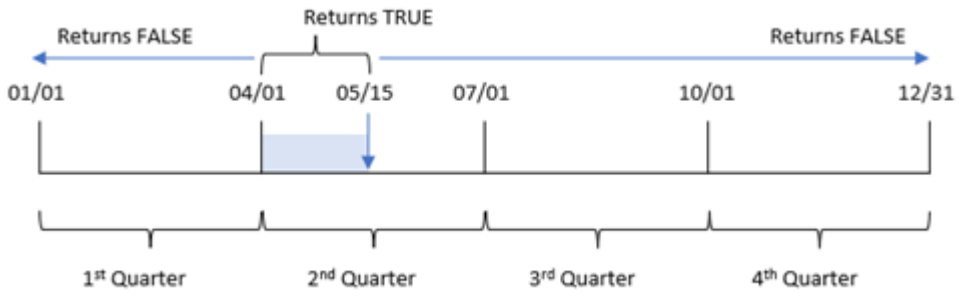
結果テーブル

日付	=inquartertodate(date,'05/15/2022', 0)
1/7/2022	0
1/19/2022	0
2/5/2022	0
2/28/2022	0
3/16/2022	0
4/1/2022	-1
5/7/2022	-1
5/16/2022	0
6/15/2022	0

日付	=inquartertodate(date,'05/15/2022', 0)
6/26/2022	0
7/9/2022	0
7/22/2022	0
7/23/2022	0
7/27/2022	0
8/2/2022	0
8/8/2022	0
8/19/2022	0
9/26/2022	0
10/14/2022	0
10/29/2022	0

in_quarter_to_date メジャーは、inquartertodate() 関数を使用してチャートオブジェクトに作成されます。最初の引数は、評価対象の日付項目です。2 番目の引数は、5 月 15 日のハードコードされた日付です。これは base_date で、セグメント化する四半期を識別し、そのセグメントの終了境界を定義します。period_no の 0 は最後の引数です。これは、関数がセグメント化された四半期の前後の四半期を比較していないことを意味します。

inquartertodate 関数の図、チャートオブジェクトの例



4 月 1 日 ~ 5 月 15 日に発生したトランザクションは、TRUE のブール値の結果を返します。5 月 16 日以降のトランザクションは FALSE を返し、4 月 1 日より前のトランザクションも同様です。

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- 製品 ID、製造年月日、原価に関する情報。

2022 年 5 月 15 日、製造工程で 1 件の設備エラーが特定され、解決されました。この日付までにその四半期に製造された製品は不良品になります。エンドユーザーは、製造された製品のステータスが「不具合」または「不具合なし」であったこと、およびその四半期までに製造された製品のコストを日付別に表示するチャートオブジェクトを希望しています。

ロードスクリプト

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。四半期名を表示する軸を作成します。
   `=quartername(manufacture_date)`
2. 次に、不具合のある製品とない製品を特定する軸を作成します。
   `=if(inquartertodate(manufacture_date,makedate(2022,05,15),0),'Defective','Faultless')`
3. 製品の cost_price を合計するメジャーを作成します。
   `=sum(cost_price)`
4. メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

quartername (manufacture_date)	if(inquartertodate(manufacture_date,makedate (2022,05,15),0),'Defective','Faultless')	Sum(cost_ price)
2022 年 1 月 ~ 3 月	不具合なし	$253.89
2022 年 4 月 ~ 6 月	不具合なし	$229.03
2022 年 4 月 ~ 6 月	不具合	$122.45
2022 年 7 月 ~ 9 月	不具合なし	$446.31
2022 年 10 月 ~ 12 月	不具合なし	$163.91

inquartertodate() 関数は、各製品の製造日を評価するときにブール値を返します。TRUE のブール値を返すものについては、製品を 'Defective' とマークします。FALSE の値を返し、5 月 15 日を含む四半期に製造されていない製品については、製品を 'Faultless' とマークします。

inweek

この関数は、**timestamp** が **base_date** を含む週にある場合、True を返します。

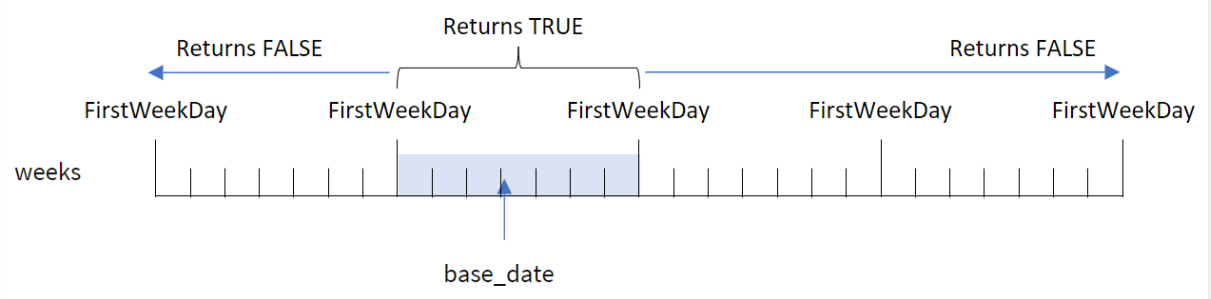
構文:

```
InWeek (timestamp, base_date, period_no[, first_week_day])
```

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

inweek() 関数範囲の図



inweek() 関数は、base_date 引数を使用して、日付が含まれる 7 日間を特定します。週の始めの曜日は、FirstWeekDay システム変数に基づいています。ただし、inweek() 関数で first_week_day 引数を使用して、週の最初の曜日を変更することもできます。

選択した週が定義されると、指定された日付値をその週セグメントと比較する際、関数はブール値の結果を返します。

使用に適しているケース

inweek() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。inweek() 関数は、評価する日付が base_date 引数の選択した日付を含む週に入っているかによって集計または計算が返されます。

例えば、inweek() 関数を使用して、特定の週に製造されたすべての機器を識別することができます。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	週の評価に使用する日付。
period_no	週は **period_no** によって補正することができます。**period_no** は整数で、値 0 は **base_date** を含む週を示します。**period_no** の値が負の場合は過去の週を、正の場合は将来の週を示します。
first_week_day	既定では、週の最初の曜日は日曜日 (FirstWeekDay システム変数で決定) で、土曜日と日曜日の間の午前 0 時に始まります。**first_week_day** パラメータは **FirstWeekDay** 変数に取って代わります。別の曜日から始まる週を指定するには、0 ~ 6 でフラグを指定します。

first_week_day values

毎日	値
月曜日	0
火曜日	1
水曜日	2
木曜日	3
金曜日	4
土曜日	5
日曜日	6

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
`inweek ('01/12/2006', '01/14/2006', 0)`	TRUE を返します
`inweek ('01/12/2006', '01/20/2006', 0 )`	FALSE を返します
`inweek ('01/12/2006', '01/14/2006', -1 )`	FALSE を返します
`inweek ('01/07/2006', '01/14/2006', -1)`	TRUE を返します
`inweek ('01/12/2006', '01/09/2006', 0, 3)`	`first_week_day` が 3 (木曜日) に指定されており、これにより01/12/2006 が、01/09/2006 を含む週の次の週の開始日となるため、FALSE を返します。

これらのトピックは、この関数を使用するのに役立つかもしれません。

関連トピック

トピック	既定 フラグ / 値	説明
FirstWeekDay (page 228)	6 / 日曜日	各週の開始日を定義します。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「`Transactions`」というテーブルにロードされる、2022 年 1 月の一連のトランザクションを含むデータセット。
- 6 (日曜日) に設定された `FirstWeekDay` システム変数。
- 次を含む、先行する LOAD:

- 2022 年 1 月 14 日の週に発生したトランザクションを決定する項目 [in_week] として設定された inweek() 関数。
- 2022 年 1 月 14 日の週にweek_day発生したトランザクションを決定する項目 [] として設定された weekday() 関数。

ロード スクリプト

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       weekday(date) as week_day,
       inweek(date,'01/14/2022', 0) as in_week
       ;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week_day
- in_week

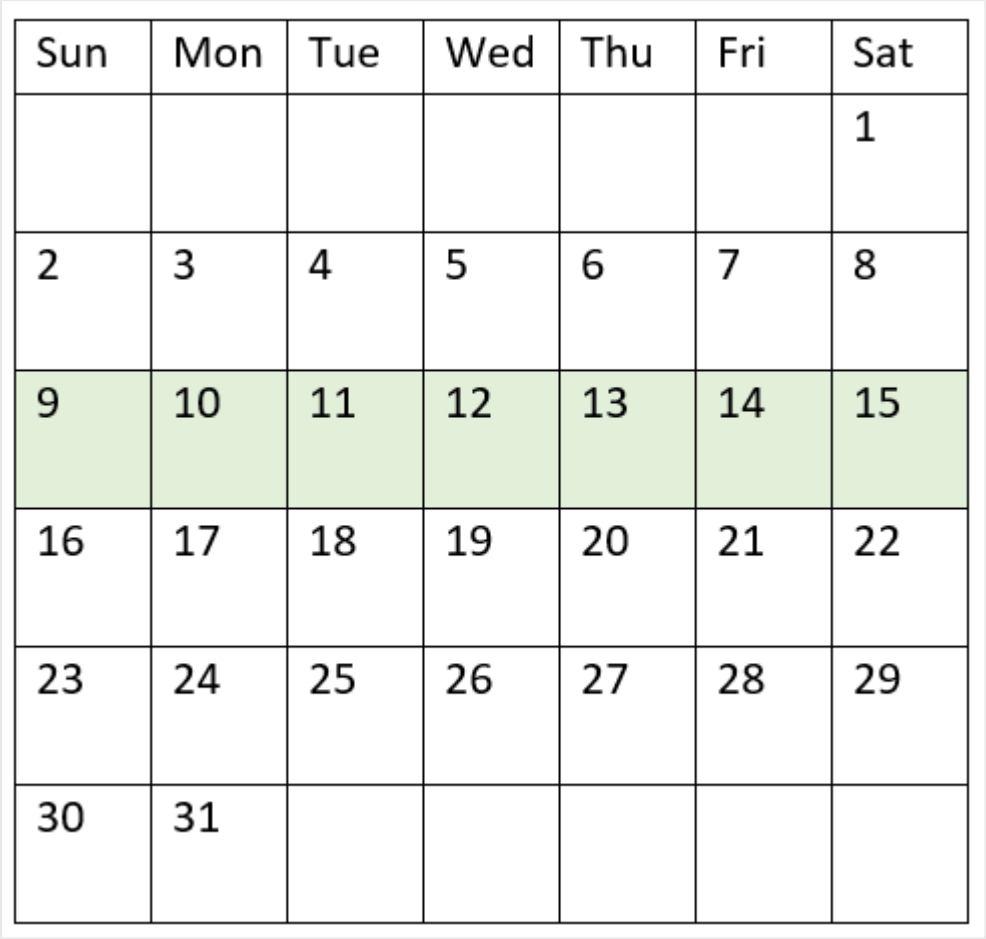
結果テーブル

日付	week_day	in_week
01/02/2022	日	0
01/05/2022	水	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	日	-1
01/10/2022	月	-1
01/11/2022	火	-1
01/12/2022	水	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	日	0
01/17/2022	月	0
01/18/2022	火	0
01/26/2022	水	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	日	0
01/31/2022	月	0

[in_week] 項目は、inweek() 関数を使用して、前の Load ステートメントで作成されます。最初の引数は、評価される項目を識別します。第 2 引数はハードコード化された日付 1 月 14 日で、これは base_date です。base_date 引数は FirstWeekDay システム変数と連携して、比較対象の週を特定します。0 の period_no (関数がセグメント化された週の前後の週を比較していないことを意味する) は、最後の引数です。

システム変数は、週が日曜日に始まり、土曜日に終わることを決定します。FirstWeekDay したがって、1 月は下図のように週に分割され、1 月 9 日 ~ 14 日の日付が inweek() 計算の有効期間となります。

inweek() 引数の範囲がハイライトされたカレンダーの図



Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1 月 9 日 ~ 14 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2022 年の一連のトランザクションを含む同じデータセットが、「Transactions」というテーブルにロードされます。
- 6 (日曜日) に設定された FirstWeekDay システム変数。
- 次を含む、先行する LOAD:
    - 2022 年 1 月 14 日の週の 1 週間前に発生したトランザクションを決定する項目 [prev_week] として設定された inweek () 関数。

- 2022 年 1 月 14 日の週にweek_day発生したトランザクションを決定する項目 [] として設定された weekday() 関数。

ロード スクリプト

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       weekday(date) as week_day,
       inweek(date,'01/14/2022', -1) as prev_week
       ;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week_day
- prev_week

結果テーブル

日付	week_day	prev_week
01/02/2022	日	-1
01/05/2022	水	-1
01/06/2022	Thu	-1
01/08/2022	Sat	-1
01/09/2022	日	0
01/10/2022	月	0
01/11/2022	火	0
01/12/2022	水	0
01/13/2022	Thu	0
01/14/2022	Fri	0
01/15/2022	Sat	0
01/16/2022	日	0
01/17/2022	月	0
01/18/2022	火	0
01/26/2022	水	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	日	0
01/31/2022	月	0

`inweek()` 関数で `-1` を `period_no` 引数として使用することにより、比較対象の週の境界を7日間ずらします。`0` の `period_no` では、週は1月 9 ~ 15 日となります。この例では、`-1` の `period_no` はこのセグメントの開始と終了境界を1週間戻します。日付境界が1月 2 日から1月 8 日になります。

inweek() 引数の範囲がハイライトされたカレンダーの図

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

したがって、1 月 2 日 ~ 1 月 8 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2022 年の一連のトランザクションを含む同じデータセットが、「Transactions」というテーブルにロードされます。
- 6 (日曜日) に設定された FirstWeekDay システム変数。
- 次を含む、先行する LOAD:
    - 2022 年 1 月 14 日の週にin_week発生したトランザクションを決定する項目 [] として設定された inweek() 関数。

- 2022 年 1 月 14 日の週にweek_day発生したトランザクションを決定する項目 [] として設定された weekday() 関数。

ロード スクリプト

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
    Load
       *,
       weekday(date) as week_day,
       inweek(date,'01/14/2022', 0, 0) as in_week
       ;
Load
*
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week_day
- in_week

結果テーブル

日付	week_day	in_week
01/02/2022	日	0
01/05/2022	水	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	日	0
01/10/2022	月	-1
01/11/2022	火	-1
01/12/2022	水	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	日	-1
01/17/2022	月	0
01/18/2022	火	0
01/26/2022	水	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	日	0
01/31/2022	月	0

`inweek()` 関数の `first_week_day` 引数として `0` を使用することにより、関数引数が `FirstWeekDay` システム変数に取って代わり、月曜日を週の初日に設定します。

inweek() 引数の範囲がハイライトされたカレンダーの図

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

したがって、1 月 10 日と16 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例ではデータセットは変更されず、アプリケーションにロードされます。結果テーブルにメジャーを作成して、2022 年 1 月 14 日の週に発生したトランザクションを判断します。

ロード スクリプト

```
SET FirstWeekDay=6;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
```

```
Inline
[
id,date,amount
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- date

次のメジャーを作成します:

- =inweek (date,'01/14/2022',0)、トランザクションが1月 14 日と同じ週に発生したかどうかを計算します。
- =weekday(date)、各日付に対応する曜日を示します。

結果テーブル

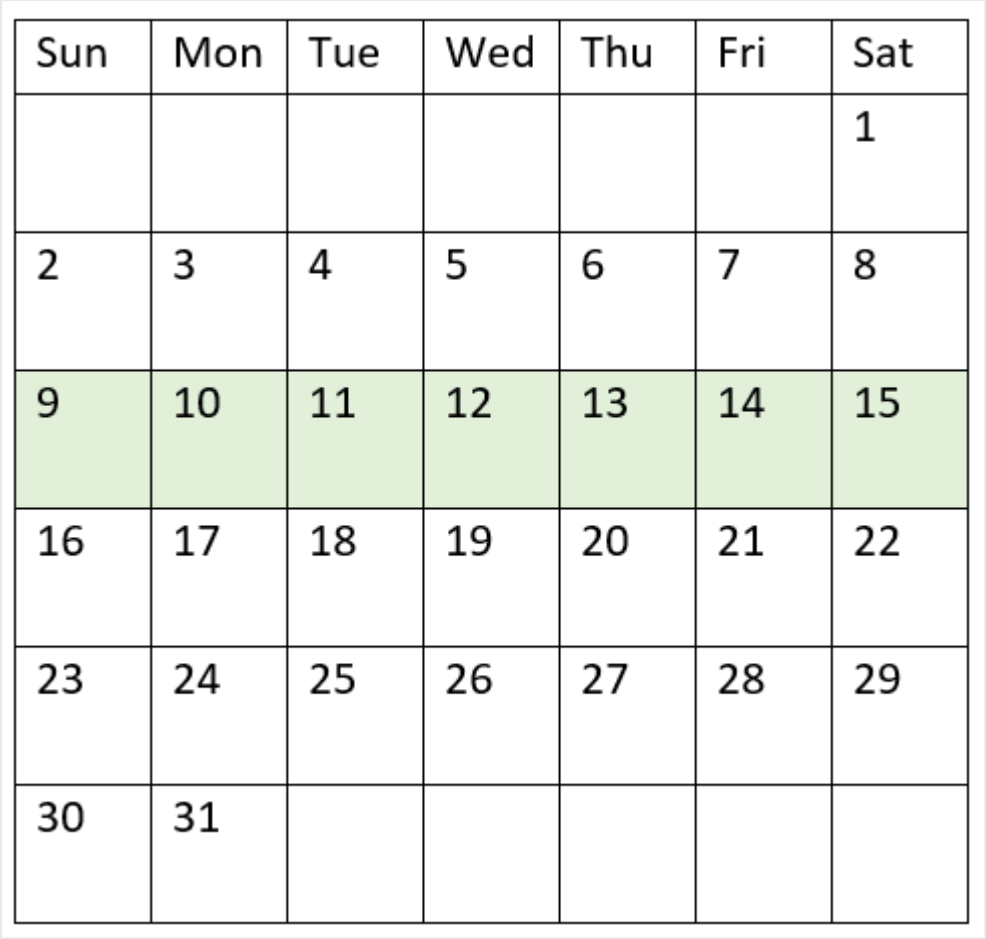
日付	week_day	=inweek (date,'01/14/2022',0)
01/02/2022	日	0
01/05/2022	水	0
01/06/2022	Thu	0
01/08/2022	Sat	0
01/09/2022	日	-1
01/10/2022	月	-1
01/11/2022	火	-1

日付	week_day	=inweek (date,'01/14/2022',0)
01/12/2022	水	-1
01/13/2022	Thu	-1
01/14/2022	Fri	-1
01/15/2022	Sat	-1
01/16/2022	日	0
01/17/2022	月	0
01/18/2022	火	0
01/26/2022	水	0
01/27/2022	Thu	0
01/28/2022	Fri	0
01/29/2022	Sat	0
01/30/2022	日	0
01/31/2022	月	0

in_week メジャーは、inweek() 関数を使用してチャートに作成されます。最初の引数は、評価される項目を識別します。第 2 引数はハードコード化された日付 1 月 14 日で、これは base_date です。base_date 引数は FirstWeekDay システム変数と連携して、比較対象の週を特定します。0 の period_no が最終引数です。

システム変数は、週が日曜日に始まり、土曜日に終わることを決定します。FirstWeekDay したがって、1 月は下図のように週に分割され、1 月 9 日 ~ 14 日の日付が inweek() 計算の有効期間となります。

inweek() 引数の範囲がハイライトされたカレンダーの図



Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1 月 9 日 ~ 14 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- テーブルには次の項目が含まれています。
  - 製品 ID
  - 製品の種類
  - 製造日付
  - コスト

機器のエラーにより、1 月 12 日の週に製造された製品に欠陥があることが確認されています。エンドユーザーは、製造された製品のステータスが「不具合」または「不具合なし」であったこと、その週に製造された製品のコストを週別に表示するチャートを希望しています。

ロードスクリプト

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'01/02/2022',37.23
8189,'01/05/2022',17.17
8190,'01/06/2022',88.27
8191,'01/08/2022',57.42
8192,'01/09/2022',53.80
8193,'01/10/2022',82.06
8194,'01/11/2022',40.39
8195,'01/12/2022',87.21
8196,'01/13/2022',95.93
8197,'01/14/2022',45.89
8198,'01/15/2022',36.23
8199,'01/16/2022',25.66
8200,'01/17/2022',82.77
8201,'01/18/2022',69.98
8202,'01/26/2022',76.11
8203,'01/27/2022',25.12
8204,'01/28/2022',46.23
8205,'01/29/2022',84.21
8206,'01/30/2022',96.24
8207,'01/31/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- `=weekname(manufacture_date)`

次のメジャーを作成します:

- `inweek()` 関数を使って、不具合のある製品とない製品を特定する `=if(only(inweek(manufacture_date,makedate(2022,01,12),0)),'Defective','Faultless')`。
- 各製品の合計コストを示す `=sum(cost_price)`。

次の手順を実行します。

1. メジャーの **[数値書式]** を **[通貨]** に設定します。
2. **[スタイル]** で **[合計]** をオフにします。

結果テーブル

weekname (manufacture_date)	=if(only(inweek(manufacture_date,makedate (2022,01,12),0)), 'Defective','Faultless')	Sum(cost_ price)
2022/02	不具合なし	200.09
2022/03	不具合	441.51
2022/04	不具合なし	178.41
2022/05	不具合なし	231.67
2022/06	不具合なし	163.91

inweek() 関数は、各製品の製造日を評価するときにブール値を返します。1 月 12 日の週に製造された製品の場合、inweek() 関数はブール値 TRUE を返し、製品を「不具合」としてマークします。 の値を返し、その週に製造されなかった製品については、その製品に「不具合なし」のマークが付けられます。

inweektodate

この関数は、**timestamp** が **base_date** のミリ秒単位まで正確に **base_date** を含む週の範囲内にある場合、True を返します。

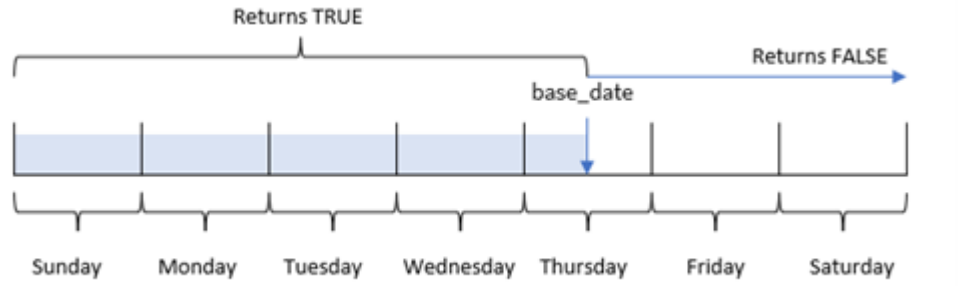
構文:

```
InWeekToDate (timestamp, base_date, period_no [, first_week_day])
```

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

inweektodate 関数の図



inweektodate() 関数は、base_date パラメータを使用して、FirstWeekDay システム変数 (またはユーザー定義の first_week_day パラメータ) に基づいて、週セグメントの最大境界日と、それに対応する週の開始日を識別します。この週セグメントが定義されると、指定された日付値をそのセグメントと比較する際、関数はブール値の結果を返します。

使用に適しているケース

inweektodate() 関数はブール値の結果を返します。通常、このタイプの関数は if 式の条件として使用されます。これにより、評価された日付が特定の日付を含む問題の週に発生したかどうかに応じて、集計または計算を返します。

たとえば、inweektodate() 関数を使用して、指定された週の特定の日付までのすべての売上を計算できます。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	週の評価に使用する日付。
period_no	週は **period_no** によって補正することができます。**period_no** は整数で、値 0 は **base_date** を含む週を示します。**period_no** の値が負の場合は過去の週を、正の場合は将来の週を示します。
first_week_day	既定では、週の最初の曜日は日曜日 (FirstWeekDay システム変数で決定) で、土曜日と日曜日の間の午前 0 時に始まります。**first_week_day** パラメータは **FirstWeekDay** 変数に取って代わります。別の曜日から始まる週を指定するには、0 ~ 6 でフラグを指定します。 月曜日で始まり日曜日で終わる週の場合、月曜日は 0、火曜日は 1、水曜日は 2、木曜日は 3、金曜日は 4、土曜日は 5、日曜日は 6 のフラグを使用します。

関数の例

例	相互作用
inweektodate ('01/12/2006', '01/12/2006', 0)	TRUE を返します。
inweektodate ('01/12/2006', '01/11/2006', 0)	FALSE を返します。
inweektodate ('01/12/2006', '01/18/2006', -1)	FALSE を返します。 period_noに -1 が指定されているため、timestampと比較する日付は 01/11/2006 になります。
inweektodate ('01/11/2006', '01/12/2006', 0, 3)	FALSE を返します。first_week_day に 3 (木曜日) が指定されているので、01/12/2006 は 01/12/2006 を含む週の次の週の初日になります。

この関数を使用する際に便利なトピックには次のようなものがあります。

関連トピック

トピック	既定 フラグ / 値	説明
FirstWeekDay (page 228)	6 / 日曜日	各週の開始日を定義します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年 1 月の一連のトランザクションを含むデータセット。
- TimestampFormat='M/D/YYYY h:mm:ss[.fff]' 形式で提供されるデータ項目。
- 2022 年 1 月 14 日までの週に発生したトランザクションを決定する項目 [in_week_to_date] の作成。
- weekday() 関数を使用した、weekday という名前の追加項目の作成。この新しい項目は、各日付に対応する曜日を示すために作成されます。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';
SET FirstWeekDay=6;
Transactions:
    Load
        *,
        weekday(date) as week_day,
        inweektodate(date,'01/14/2022', 0) as in_week_to_date
        ;
Load
*
Inline
[
id,date,amount
```

```
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week_day
- in_week_to_date

結果テーブル

日付	week_day	in_week_to_date
2022-01-02 12:22:06	日	0
2022-01-05 01:02:30	水	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	日	-1
2022-01-10 21:13:01	月	-1
2022-01-11 00:57:13	火	-1
2022-01-12 09:26:02	水	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1
2022-01-15 06:10:46	Sat	0

日付	week_day	in_week_to_date
2022-01-16 06:39:27	日	0
2022-01-17 10:44:16	月	0
2022-01-18 18:48:17	火	0
2022-01-26 04:36:03	水	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	日	0
2022-01-30 14:40:19	日	0
2022-01-31 05:28:21	月	0

[in_week_to_date] 項目は、inweektodate() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は、評価される項目を識別します。2 番目の引数は、1月 14 日のハードコード化された日付です。これは base_date で、セグメント化する週を識別し、そのセグメントの終了境界を定義します。period_no の 0 は最後の引数です。これは、関数がセグメント化された週の前後の週を比較していないことを意味します。

システム変数は、週が日曜日に始まり、土曜日に終わることを決定します。FirstWeekDay したがって、1 月は下図のように週に分割され、1 月 9 日 ~ 14 日の日付が inweekdodate() 計算の有効期間となります。

TRUE のブール結果を返す取引日を示すカレンダー図

Sun	Mon	Tue	Wed	Thur	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1 月 9 日 ~ 14 日に発生したトランザクションは、TRUE のブール値の結果を返します。日付の前後のトランザクションはブール値の結果 FALSE を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 2022 年 1 月 14 日に終了する週セグメント前の週全体で発生したトランザクションを決定する項目 [prev_week_to_date] の作成。
- weekday() 関数を使用した、weekday という名前の追加項目の作成。これは、各日付に対応する曜日を示すためものです。

ロードスクリプト

```
SET FirstWeekDay=6;
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';
Transactions:
    Load
        *,
        weekday(date) as week_day,
        inweektodate(date,'01/14/2022', -1) as prev_week_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week_day
- prev_week_to_date

結果テーブル

日付	week_day	prev_week_to_date
2022-01-02 12:22:06	日	-1
2022-01-05 01:02:30	水	-1
2022-01-06 15:36:20	Thu	-1
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	日	0
2022-01-10 21:13:01	月	0
2022-01-11 00:57:13	火	0
2022-01-12 09:26:02	水	0
2022-01-13 15:05:09	Thu	0
2022-01-14 18:44:57	Fri	0
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	日	0
2022-01-17 10:44:16	月	0
2022-01-18 18:48:17	火	0
2022-01-26 04:36:03	水	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	日	0
2022-01-30 14:40:19	日	0
2022-01-31 05:28:21	月	0

period_no 値 -1 は、inweektodate () 関数が入力週セグメントを前の週と比較することを示します。週セグメントは、最初は 1 月 9 日 ~ 1 月 14 日に相当します。period_no は、このセグメントの開始境界と終了境界の両方を 1 週間前にオフセットし、日付境界を 1 月 2 日 ~ 1 月 7 日にします。

TRUE のブール結果を返すトランザクション日付を示すカレンダー図

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

したがって、1 月 2 日 ~ 8 日 (1 月 8 日自体は含まれない) に発生したトランザクションは、TRUEのブール値の結果を返します。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 2022 年 1 月 14 日までの週に発生したトランザクションを決定する項目 [in_week_to_date] の作成。
- weekday() 関数を使用した、weekday という名前の追加項目の作成。これは、各日付に対応する曜日を示すためものです。

この例では、週の初日として月曜日を使用しています。

ロードスクリプト

```
SET FirstWeekDay=6;
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff]';

Transactions:
    Load
        *,
        weekday(date) as week_day,
        inweektodate(date,'01/14/2022', 0, 0) as in_week_to_date
        ;
Load
*
Inline
```

```
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- week_day
- in_week_to_date

結果テーブル

日付	week_day	in_week_to_date
2022-01-02 12:22:06	日	0
2022-01-05 01:02:30	水	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	日	0
2022-01-10 21:13:01	月	-1
2022-01-11 00:57:13	火	-1
2022-01-12 09:26:02	水	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1

日付	week_day	in_week_to_date
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	日	0
2022-01-17 10:44:16	月	0
2022-01-18 18:48:17	火	0
2022-01-26 04:36:03	水	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	日	0
2022-01-30 14:40:19	日	0
2022-01-31 05:28:21	月	0

inweektodate() 関数の first_week_day 引数として 0 を使用することにより、関数引数が FirstWeekDay システム変数に取って代わり、月曜日を週の初日に設定します。

TRUE のブール結果を返す取引日を示すカレンダー図

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	17
24	25	26	27	28	29	30
31						

したがって、1 月 10 日 ~ 14 日に発生したトランザクションは TRUE のブール値の結果を返しますが、これらの境界外の日付のトランザクションは FALSE の値を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。1 月 14 日までの週に四半期に発生したトランザクションを決定する計算は、チャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `date`。
2. 1 月 14 日までの同週にトランザクションが発生したかどうかを計算するには、次のメジャーを作成します。
   `=inweektodate(date,'01/14/2022',0)`
3. どの曜日がどの日付に対応するかを表示するには、追加のメジャーを作成します。
   `=weekday(date)`

結果テーブル

日付	week_day	in_week_to_date
2022-01-02 12:22:06	日	0

日付	week_day	in_week_to_date
2022-01-05 01:02:30	水	0
2022-01-06 15:36:20	Thu	0
2022-01-08 10:58:35	Sat	0
2022-01-09 08:53:32	日	-1
2022-01-10 21:13:01	月	-1
2022-01-11 00:57:13	火	-1
2022-01-12 09:26:02	水	-1
2022-01-13 15:05:09	Thu	-1
2022-01-14 18:44:57	Fri	-1
2022-01-15 06:10:46	Sat	0
2022-01-16 06:39:27	日	0
2022-01-17 10:44:16	月	0
2022-01-18 18:48:17	火	0
2022-01-26 04:36:03	水	0
2022-01-27 08:07:49	Thu	0
2022-01-28 12:24:29	Fri	0
2022-01-30 11:56:56	日	0
2022-01-30 14:40:19	日	0
2022-01-31 05:28:21	月	0

[in_week_to_date] 項目は、inweektodate() 関数を使用してチャートオブジェクトに作成されます。提供される最初の引数は、評価される項目を識別します。2 番目の引数は、1月 14 日のハードコード化された日付です。これは base_date で、セグメント化する週を識別し、そのセグメントの終了境界を定義します。period_no の 0 は最後の引数です。これは、関数がセグメント化された週の前後の週を比較していないことを意味します。

システム変数は、週が日曜日に始まり、土曜日に終わることを決定します。FirstWeekDay したがって、1 月は下図のように週に分割され、1 月 9 日 ~ 14 日の日付が inweekdodate() 計算の有効期間となります。

TRUE のブール結果を返す取引日を示すカレンダー図

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

1 月 9 日 ~ 14 日に発生したトランザクションは、TRUE のブール値の結果を返します。日付の前後のトランザクションはブール値の結果 FALSE を返します。

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- 製品 ID、製造年月日、原価に関する情報。

機器のエラーにより、1 月 12 日の週に製造された製品に欠陥があることが確認されています。この問題は 1 月 13 日に解決されました。エンドユーザーは、製造された製品のステータスが「不具合」または「不具合なし」であったこと、およびその週に製造された製品のコストを週別に表示するチャートオブジェクトを希望しています。

ロードスクリプト

```
Products:
Load
*
Inline
[
product_id,manufacture_date,cost_price
8188,'2022-01-02 12:22:06',37.23
8189,'2022-01-05 01:02:30',17.17
8190,'2022-01-06 15:36:20',88.27
8191,'2022-01-08 10:58:35',57.42
8192,'2022-01-09 08:53:32',53.80
8193,'2022-01-10 21:13:01',82.06
```

```
8194,'2022-01-11 00:57:13',40.39
8195,'2022-01-12 09:26:02',87.21
8196,'2022-01-13 15:05:09',95.93
8197,'2022-01-14 18:44:57',45.89
8198,'2022-01-15 06:10:46',36.23
8199,'2022-01-16 06:39:27',25.66
8200,'2022-01-17 10:44:16',82.77
8201,'2022-01-18 18:48:17',69.98
8202,'2022-01-26 04:36:03',76.11
8203,'2022-01-27 08:07:49',25.12
8204,'2022-01-28 12:24:29',46.23
8205,'2022-01-30 11:56:56',84.21
8206,'2022-01-30 14:40:19',96.24
8207,'2022-01-31 05:28:21',67.67
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。週名を表示する軸を作成します。
   `=weekname(manufacture_date)`
2. 次に、不具合のある製品とない製品を特定する軸を作成します。
   `=if(inWeektodate(manufacture_date,makedate(2022,01,12),0),'Defective','Faultless')`
3. 製品の `cost_price` を合計するメジャーを作成します。
   `=sum(cost_price)`
4. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

weekname (manufacture_date)	if(inweektodate(manufacture_date,makedate (2022,01,12),0),'Defective','Faultless')	Sum(cost_ price)
2022/02	不具合なし	$200.09
2022/03	不具合	$263.46
2022/03	不具合なし	$178.05
2022/04	不具合なし	$178.41
2022/05	不具合なし	$147.46
2022/06	不具合なし	$248.12

`inweektodate()` 関数は、各製品の製造日を評価するときにブール値を返します。`TRUE` のブール値を返すものについては、製品を `'Defective'` とマークします。`FALSE` の値を返し、1 月 12 日までの週に製造されていない製品については、製品を `'Faultless'` とマークします。

inyear

この関数は、**timestamp** が **base_date** を含む年の範囲内にある場合、True を返します。

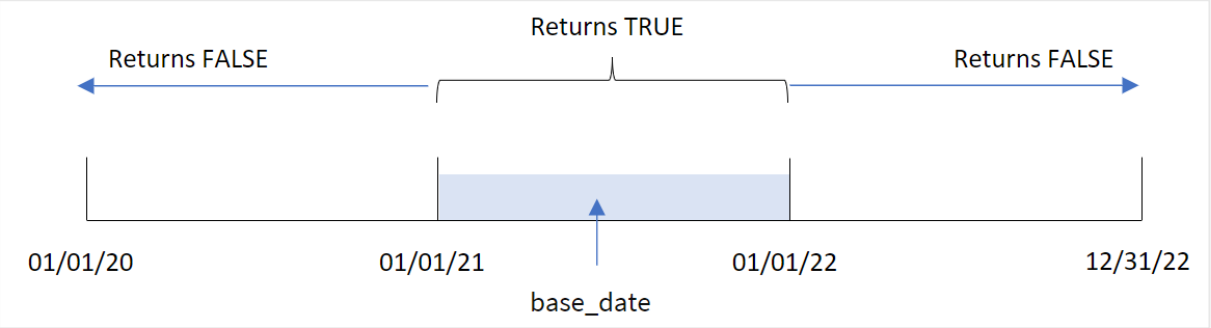
構文:

```
InYear (timestamp, base_date, period_no [, first_month_of_year])
```

戻り値データ型： ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

inyear() 関数範囲の図



選択した日付値を base_date により定義された年と比較する際、inyear() 関数はブール値の結果を返します。

使用に適しているケース

inyear() 関数はブール値の結果を返します。通常、このタイプの関数は if expression の条件として使用されます。これは、評価された日付が問題の年に発生したかどうかに応じて、集計または計算を返します。例えば、inyear() 関数は定義した年に発生したすべての販売を特定するのに使用できます。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	年の評価に使用する日付。
period_no	年は **period_no** によって補正することができます。**period_no** は整数で、値 0 は **base_date** を含む年を示します。**period_no** の値が負の場合は過去の年を、正の場合は将来の年を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

次の値を使用して、first_month_of_year 引数に年の最初の月を設定できます。

first_month_of_year values

月	値
February	2
3 月	3
April	4

月	値
May	5
June	6
7 月	7
8 月	8
September	9
10 月	10
November	11
12 月	12

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
`inyear ('01/25/2013', '01/01/2013', 0 )`	TRUE を返します
`inyear ('01/25/2012', '01/01/2013', 0)`	FALSE を返します
`inyear ('01/25/2013', '01/01/2013', -1)`	FALSE を返します
`inyear ('01/25/2012', '01/01/2013', -1 )`	TRUE を返します
`inyear ('01/25/2013', '01/01/2013', 0, 3)`	TRUE を返します `first_month_of_year` の `base_date` は、タイムスタンプが01/03/2012 から 28/02/2013 までの範囲内である必要がありことを指定します。
`inyear ('03/25/2013', '07/01/2013', 0, 3 )`	TRUE を返します

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- inyear() 関数が [in_year] 項目として設定された先行する LOAD で、2021 年 7 月 26 日と同じ年に発生したトランザクションを決定します。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inyear(date,'07/26/2021', 0) as in_year
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

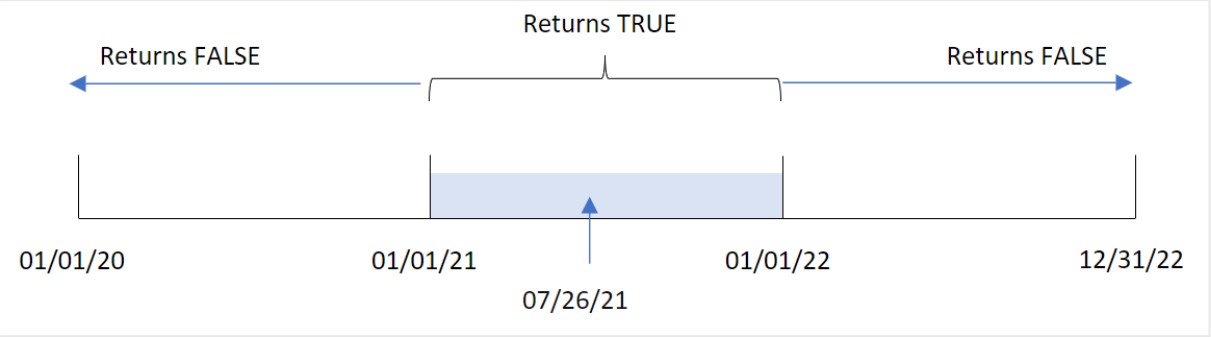
- date
- in_year

結果テーブル

日付	in_year
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

[in_year] 項目は、inyear() 関数を使用して、前の Load ステートメントで作成されます。最初の引数は、評価される項目を識別します。2 番目の引数は、2021 年 7 月 26 日のハードコード化された日付です。これは base_date で、その比較年を定義します。0 の period_no は最後の引数です。つまり、inyear() 関数がセグメント化された年の前後の年を比較しないという意味です。

基準日が 7 月 26 日の inyear() 関数の範囲の図



2021 年に発生したトランザクションは、ブール値結果 TRUE を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- inyear() 関数が [previous_year] 項目として設定された先行 load で、2021 年 7 月 26 日を含む年の前の年に発生したトランザクションを決定します。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inyear(date,'07/26/2021', -1) as previous_year
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
```

```
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_year

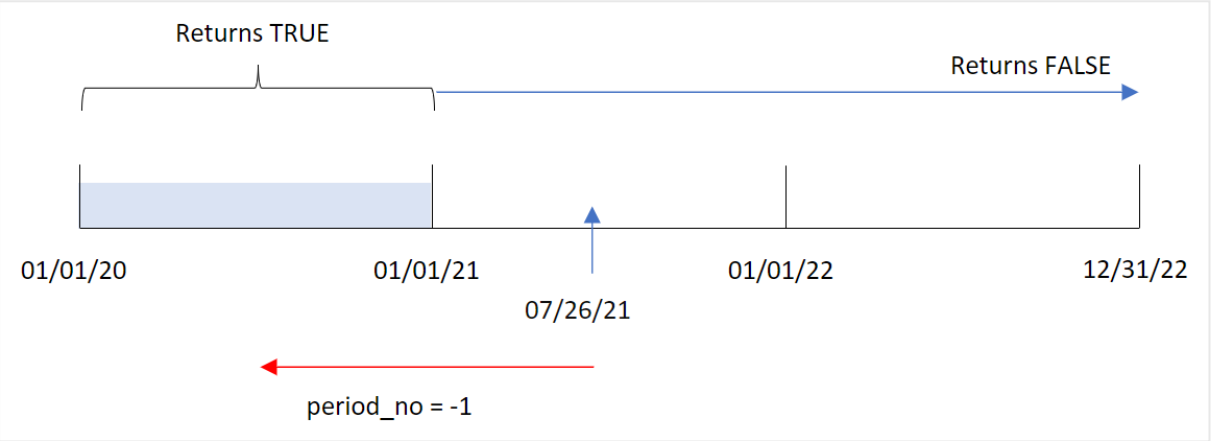
結果テーブル

日付	previous_year
01/13/2020	-1
02/26/2020	-1
03/27/2020	-1
04/16/2020	-1
05/21/2020	-1
08/14/2020	-1
10/07/2020	-1
12/05/2020	-1
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
06/06/2022	0
07/18/2022	0

日付	previous_year
11/14/2022	0
12/12/2022	0

-1 を inyear() の period_no 引数として使用することにより、比較年の境界が丸 1 年戻ります。元々比較年として特定されていたのは 2021 年です。period_no は、比較年を 1 年オフセットするため、2020 年が比較年となります。

period_no argument が -1 に設定された inyear() 関数の範囲の図



そのため、2020 年に発生したトランザクションは、ブール値結果 TRUE を返します。

Example 3 - first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- inyear() 関数が [in_year] 項目として設定された先行 load で、2021 年 7 月 26 日と同じ年に発生したトランザクションを決定します。

ただしこの例では、組織ポリシーでは 3 月が会計期間の開始月に定められています。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
Transactions:
    Load
        *,
        inyear(date,'07/26/2021', 0, 3) as in_year
```

```
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_year

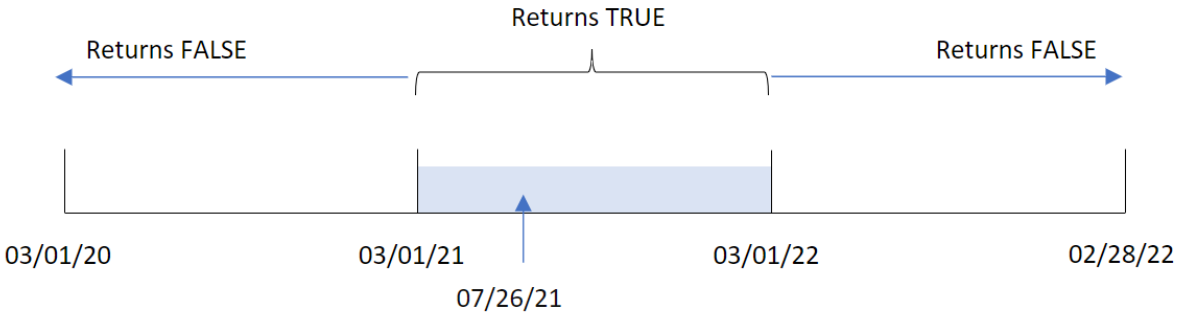
結果テーブル

日付	in_year
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0

日付	in_year
01/22/2021	0
02/03/2021	0
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

3 を inyear() 関数の first_month_of_year 引数として使用することで、年は 3 月 1 日に開始され、2 月 終わりに終了します。

3 月が年の最初の月に設定された inyear() 関数の範囲の図



したがって、2021 年 3 月 1 日 ~ 2022 年 3 月 1 日に発生したトランザクションは、TRUE のブール値の結果を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例ではデータセットは変更されず、アプリケーションにロードされます。トランザクションが 2021 年 7 月 26 日の年と同年に発生したかどうかを判断する計算は、アプリケーションのチャートオブジェクトのメジャーとして作成されます。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- date

2021 年 7 月 26 日と同年にトランザクションが発生したかどうかを計算するには、次のメジャーを作成します:

- =inyear(date,'07/26/2021', 0)

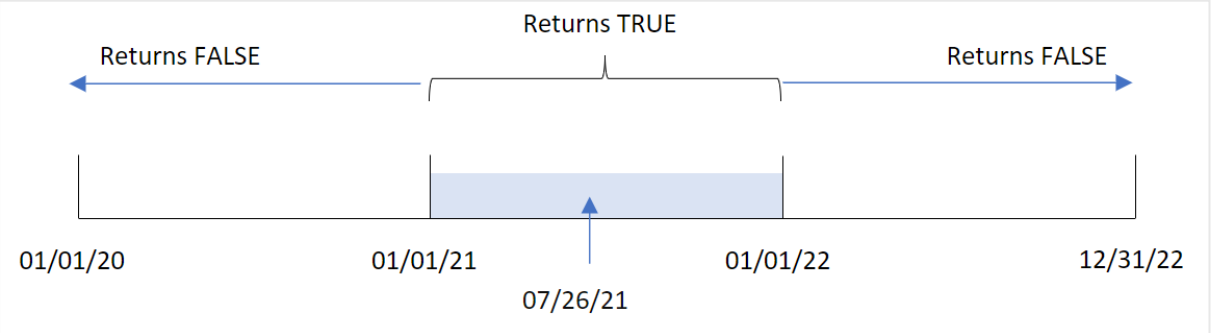
結果テーブル

日付	=inyear(date,'07/26/2021',0)
01/13/2020	0
02/26/2020	0

日付	=inyear(date,'07/26/2021',0)
03/27/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
12/27/2021	-1
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

[in_year] 項目は、inyear() 関数を使用することにより、チャートに作成されます。最初の引数は、評価される項目を識別します。2 番目の引数は、2021 年 7 月 26 日のハードコード化された日付です。これは base_date で、その比較年を定義します。0 の period_no は最後の引数です。つまり、inyear() 関数がセグメント化された年の前後の年を比較しないという意味です。

基準日が 7 月 27 日の inyear() 関数の範囲の図



2021 年に発生したトランザクションは、ブール値結果 TRUE を返します。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- テーブルには次の項目が含まれています。
    - 製品 ID
    - 製品の種類
    - 製造日付
    - コスト

エンドユーザーは、2021 年に製造された製品のコストを製品タイプ別に表示するチャートオブジェクトを望んでいます。

ロードスクリプト

```
Products:
Load
*
Inline
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'01/13/2020',37.23
8189,product B,'02/26/2020',17.17
8190,product B,'03/27/2020',88.27
8191,product C,'04/16/2020',57.42
8192,product D,'05/21/2020',53.80
8193,product D,'08/14/2020',82.06
8194,product C,'10/07/2020',40.39
8195,product B,'12/05/2020',87.21
8196,product A,'01/22/2021',95.93
8197,product B,'02/03/2021',45.89
8198,product C,'03/17/2021',36.23
8199,product C,'04/23/2021',25.66
8200,product B,'05/04/2021',82.77
8201,product D,'06/30/2021',69.98
8202,product D,'07/26/2021',76.11
8203,product D,'12/27/2021',25.12
8204,product C,'06/06/2022',46.23
8205,product C,'07/18/2022',84.21
8206,product A,'11/14/2022',96.24
8207,product B,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

- product_type

7 月 27 日より前、2021 年に製造された各製品の合計を計算するメジャーを作成します:

- =sum(if(InYear(manufacture_date,makedate(2021,01,01),0),cost_price,0))

次の手順を実行します。

1. メジャーの **[数値書式]** を **[通貨]** に設定します。
2. **[スタイル]** で **[合計]** をオフにします。

結果テーブル

product_type	=sum(if(InYear(manufacture_date,makedate(2021,01,01),0),cost_price,0))
製品 A	$95.93
製品 B	$128.66
製品 C	$61.89
製品 D	$171.21

inyear() 関数は、各製品の製造日を評価するときにブール値を返します。2021 年に製造された製品の場合、inyear() 関数はブール値 cost_price を返し、. の合計を算出します。

inyeartodate

この関数は、**timestamp** が **base_date** のミリ秒単位まで正確に **base_date** を含む年の範囲内にある場合、True を返します。

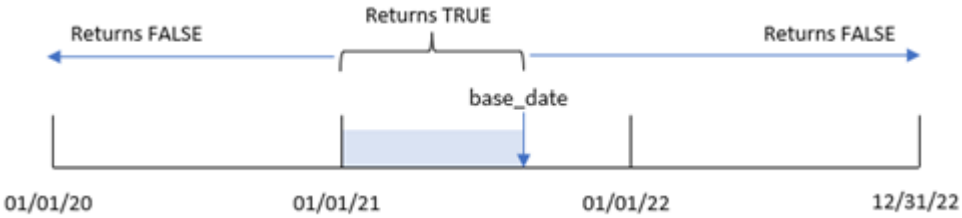
構文:

```
InYearToDate (timestamp, base_date, period_no[, first_month_of_year])
```

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

inyeartodate 関数の図



inyeartodate() 関数は base_date でその年度の特定の部分を分割し、その年度セグメントの最大許容日付の両方を識別します。関数は、日付項目または値がこのセグメントに該当するかどうかを評価し、ブール値の結果を返します。

引数

引数	説明
timestamp	**base_date** と比較する日付。
base_date	年の評価に使用する日付。
period_no	年は **period_no** によって補正することができます。**period_no** は整数で、値 0 は **base_date** を含む年を示します。**period_no** の値が負の場合は過去の年を、正の場合は将来の年を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

使用に適しているケース

inyeartodate() 関数はブール値の結果を返します。通常、このタイプの関数は if 式の条件として使用されます。これにより、評価される日付が問題の日付を含む年度に発生したかどうかに応じて、集計または計算を返します。

例えば、inyeartodate() 関数を使用して、特定の日付を含む年度に製造されたすべての機器を識別することができます。

これらの例は、日付書式 DD/MM/YYYY を使用しています。日付書式は、データロードスクリプト上部の SET DateFormat ステートメントで指定されています。必要に応じて、書式を変更してください。

関数の例

例	結果
inyeartodate ('01/25/2013', '02/01/2013', 0)	TRUE を返します。
inyeartodate ('01/25/2012', '01/01/2013', 0)	FALSE を返します。

例	結果
inyeartodate ('01/25/2012', '02/01/2013', -1)	TRUE を返します。
inyeartodate ('11/25/2012', '01/31/2013', 0, 4)	TRUE を返します。 timestamp の値は、4 番目の月から始まる会計年度かつ base_date の値までの範囲内です。
inyeartodate ('3/31/2013', '01/31/2013', 0, 4)	FALSE を返します。 上の例との違いは、timestamp の値が会計年度内ではあるものの base_date の値の後になることです。そのため、範囲外になります。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- 2021 年 7 月 26 日までの年に発生したトランザクションを決定する項目 [in_year_to_date] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        inyeartodate(date,'07/26/2021', 0) as in_year_to_date
```

```
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- in_year_to_date

結果テーブル

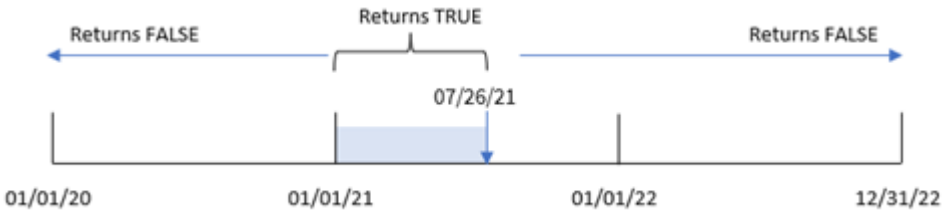
日付	in_year_to_date
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0

日付	in_year_to_date
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

[in_year_to_date] 項目は、inyeartodate() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は、評価される項目を識別します。

2 番目の引数は、2021 年 7 月 26 日のハードコード化された日付です。これは base_date で、その年セグメントの終了境界を定義します。period_no の 0 は最後の引数です。つまり、関数がセグメント化された年の前後の年を比較していないということです。

inyeartodate 関数の図、追加の引数なし



1 月 1 日 ~ 7 月 26 日に発生したトランザクションは、TRUE のブール値の結果を返します。2021 年より前で 2021 年 7 月 26 より後のトランザクション日付は FALSE を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 2021 年 7 月 26 日に終了する週セグメント前の年全体で発生したトランザクションを決定する項目 [previous_year_to_date] の作成。

ロード スクリプト

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

Transactions:
    Load
        *,
        inyeartodate(date,'07/26/2021', -1) as previous_year_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_year_to_date

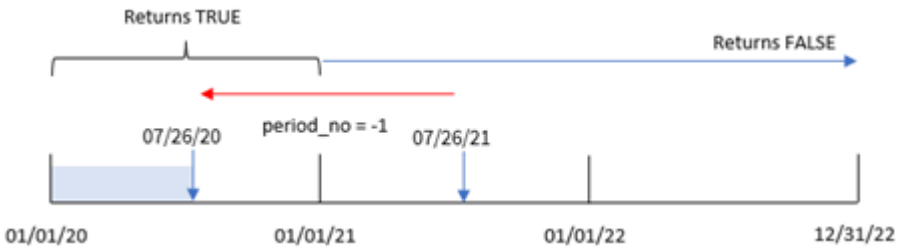
結果テーブル

日付	previous_year_to_date
01/13/2020	-1

日付	previous_year_to_date
02/26/2020	-1
03/27/2020	-1
04/16/2020	-1
05/21/2020	-1
06/14/2020	-1
08/07/2020	0
09/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

period_no 値 -1 は、inyeartodate () 関数が入力年セグメントを前の年と比較することを示します。入力日が 2021 年 7 月 26 日の場合、2021 年 1 月 1 日 ~ 2021 年 7 月 26 日のセグメントが最初に年初来として識別されました。period_no は、このセグメントを 1 年前にオフセットし、日付の境界を 2020 年 1 月 1 日 ~ 7 月 26 日にします。

inyeartodate 関数の図、period_no の例



したがって、1 月 1 日 ~ 7 月 26 日に発生したトランザクションは、TRUEのブール値の結果を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 2021年7月26日までの同年に発生したトランザクションを決定する項目 [in_year_to_date] の作成。

この例では、3月を会計年度の最初の月として設定します。

ロードスクリプト

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

Transactions:
    Load
        *,
        inyeartodate(date,'07/26/2021', 0,3) as in_year_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

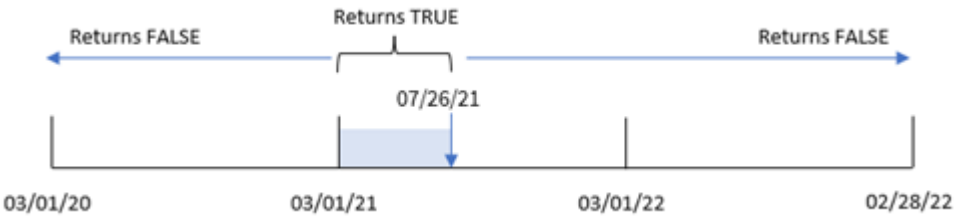
- date
- in_year_to_date

結果テーブル

日付	in_year_to_date
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

inyeartodate() 関数で first_month_of_year 引数に 3 を使用すると、関数は 3 月 1 日に年度を開始します。2021 年 7 月 26 日の base_date は、その年のセグメントの終了日を設定します。

inyeartodate 関数の図、first_month_of_year の例



したがって、2021 年 3 月 1 日 ~ 7 月 26 日に発生したトランザクションは TRUE のブール値の結果を返しますが、これらの境界外の日付のトランザクションは FALSE の値を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが 2021 年 7 月 26 日までの年と同年に発生したかどうかを判断する計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'06/14/2020',82.06
8194,'08/07/2020',40.39
8195,'09/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'07/27/2021',25.12
8204,'06/06/2022',46.23
```

```
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

次のメジャーを作成します:

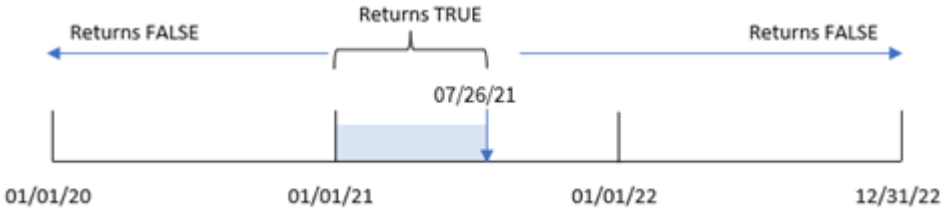
```
=inyeartodate(date,'07/26/2021', 0)
```

結果テーブル

日付	=inyeartodate(date,'07/26/2021', 0)
01/13/2020	0
02/26/2020	0
03/27/2020	0
04/16/2020	0
05/21/2020	0
06/14/2020	0
08/07/2020	0
09/05/2020	0
01/22/2021	-1
02/03/2021	-1
03/17/2021	-1
04/23/2021	-1
05/04/2021	-1
06/30/2021	-1
07/26/2021	-1
07/27/2021	0
06/06/2022	0
07/18/2022	0
11/14/2022	0
12/12/2022	0

in_year_to_date メジャーは、inyeartodate() 関数を使用してチャートオブジェクトに作成されます。提供される最初の引数は、評価される項目を識別します。2 番目の引数は、2021 年 7 月 26 日のハードコード化された日付です。これは base_date で、その比較年セグメントの終了境界を定義します。period_no の 0 は最後の引数です。つまり、関数がセグメント化された年の前後の年を比較していないということです。

inyeartodate 関数の図、チャートオブジェクトの例



2021 年 1 月 1 日 ~ 7 月 26 日に発生したトランザクションは、TRUE のブール値の結果を返します。2021 年より前で 2021 年 7 月 26 より後のトランザクション日付は FALSE を返します。

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Products」というテーブルにロードされるデータセット。
- 製品 ID、製品タイプ、製造年月日、原価に関する情報。

エンドユーザーは、2021 年に製造された製品の 7 月 26 日までのコストを製品タイプ別に表示するチャートオブジェクトを望んでいます。

ロードスクリプト

```
Products:
Load
*
Inline
[
product_id,product_type,manufacture_date,cost_price
8188,product A,'01/13/2020',37.23
8189,product B,'02/26/2020',17.17
8190,product B,'03/27/2020',88.27
8191,product C,'04/16/2020',57.42
8192,product D,'05/21/2020',53.80
8193,product D,'08/14/2020',82.06
8194,product C,'10/07/2020',40.39
8195,product B,'12/05/2020',87.21
8196,product A,'01/22/2021',95.93
8197,product B,'02/03/2021',45.89
```

```
8198,product C,'03/17/2021',36.23
8199,product C,'04/23/2021',25.66
8200,product B,'05/04/2021',82.77
8201,product D,'06/30/2021',69.98
8202,product D,'07/26/2021',76.11
8203,product D,'12/27/2021',25.12
8204,product C,'06/06/2022',46.23
8205,product C,'07/18/2022',84.21
8206,product A,'11/14/2022',96.24
8207,product B,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:product_type。

7 月 27 日より前、2021 年に製造された各製品の合計を計算するメジャーを作成します。

```
=sum(if(inyeartodate(manufacture_date,makedate(2021,07,26),0),cost_price,0))
```

メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

product_type	**=sum(if(inyeartodate(manufacture_date,makedate (2021,07,26),0),cost_price,0))**
製品 A	$95.93
製品 B	$128.66
製品 C	$61.89
製品 D	$146.09

inyeartodate() 関数は、各製品の製造日を評価するときにブール値を返します。7 月 27 日まで、2021 年に製造された製品の場合、inyeartodate() 関数はブール値 TRUE を返し、cost_price. の合計を算出します。

製品 D は、2021 年 7 月 26 日より後に製造された唯一の製品でもあります。product_ID 8203 のエントリは、12 月 27 日に製造され、コストは $25.12 でした。したがって、このコストはチャートオブジェクトの製品 D の合計には含まれません。

lastworkdate

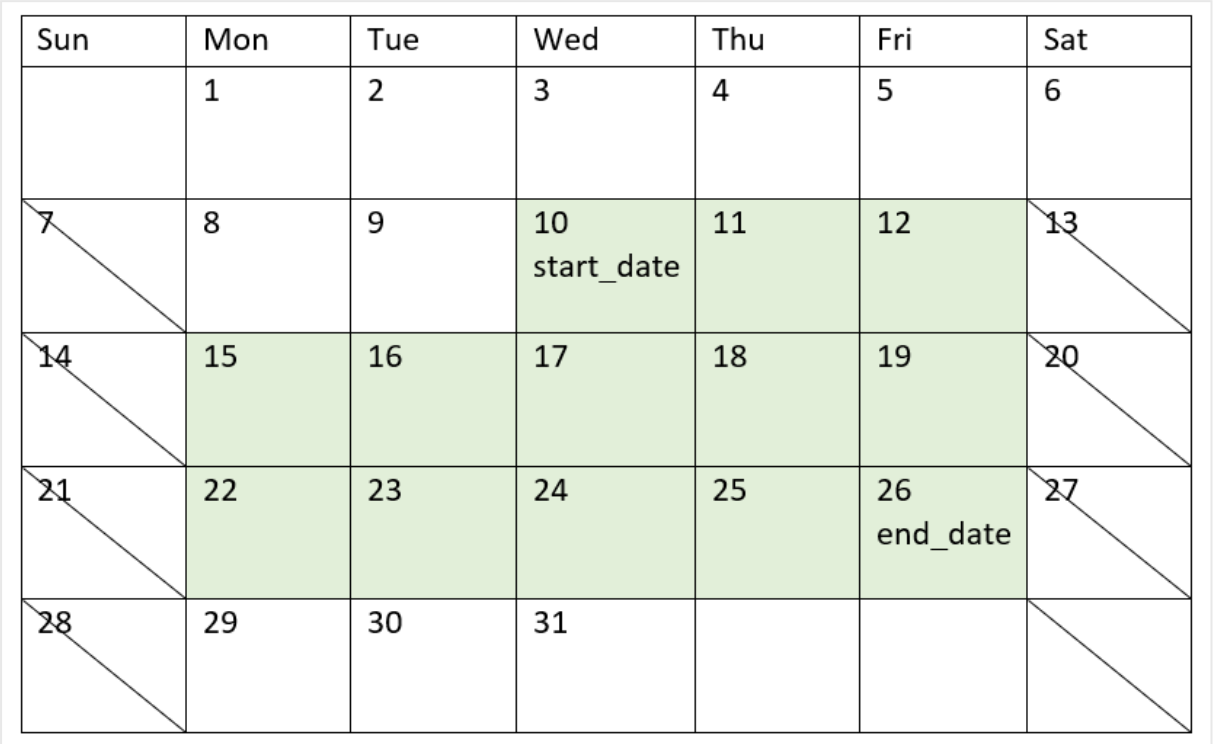
lastworkdate 関数は、オプションで指定された **holiday** を考慮した上で、**start_date** に開始した場合に **no_of_workdays** (月 ~ 金曜日) の日数に達する最早終了日を返します。**start_date** と **holiday** は、有効な日付またはタイムスタンプでなければなりません。

構文:

```
lastworkdate(start_date, no_of_workdays {, holiday})
```

戻り値データ型： 整数

lastworkdate() 関数の使用方法を示すカレンダー



Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10 start_date	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26 end_date	27
28	29	30	31			

制限事項

開始日が月曜日で終了日が金曜日の業務週以外に関与する地域やシナリオに対して lastworkdate() 関数を変更する方法はありません。

休日パラメータは文字列定数である必要があります。数式は使用できません。

使用に適しているケース

lastworkdate() 関数は、ユーザーが、プロジェクトが開始する日時とその期間発生する休日に基づいて、プロジェクトまたは課題に要請された終了日を計算するための数式の一部としてよく使用されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

引数

引数	説明
start_date	評価する開始日。
no_of_workdays	作成する作業日数。
holiday	作業日から除外する休日期間。休日は文字列定数の日付として示されます。コンマで区切り、複数の休日を設定できます。 **'12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'**

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- プロジェクトID、プロジェクト開始日、およびプロジェクトに必要な推定工数、日数を含むデータセット。「Projects」というテーブルにロードされるデータセット。
- 項目 [end_date] として設定され、各プロジェクトが終了するスケジュールを特定する lastworkdate() 関数を含む、先行する LOAD。

ロードスクリプト

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

Projects:
    Load
       *,
       LastWorkDate(start_date,effort) as end_date
       ;
Load
id,
start_date,
effort
Inline
[
id,start_date,effort
1,01/01/2022,14
```

```
2,02/10/2022,17
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- effort
- end_date

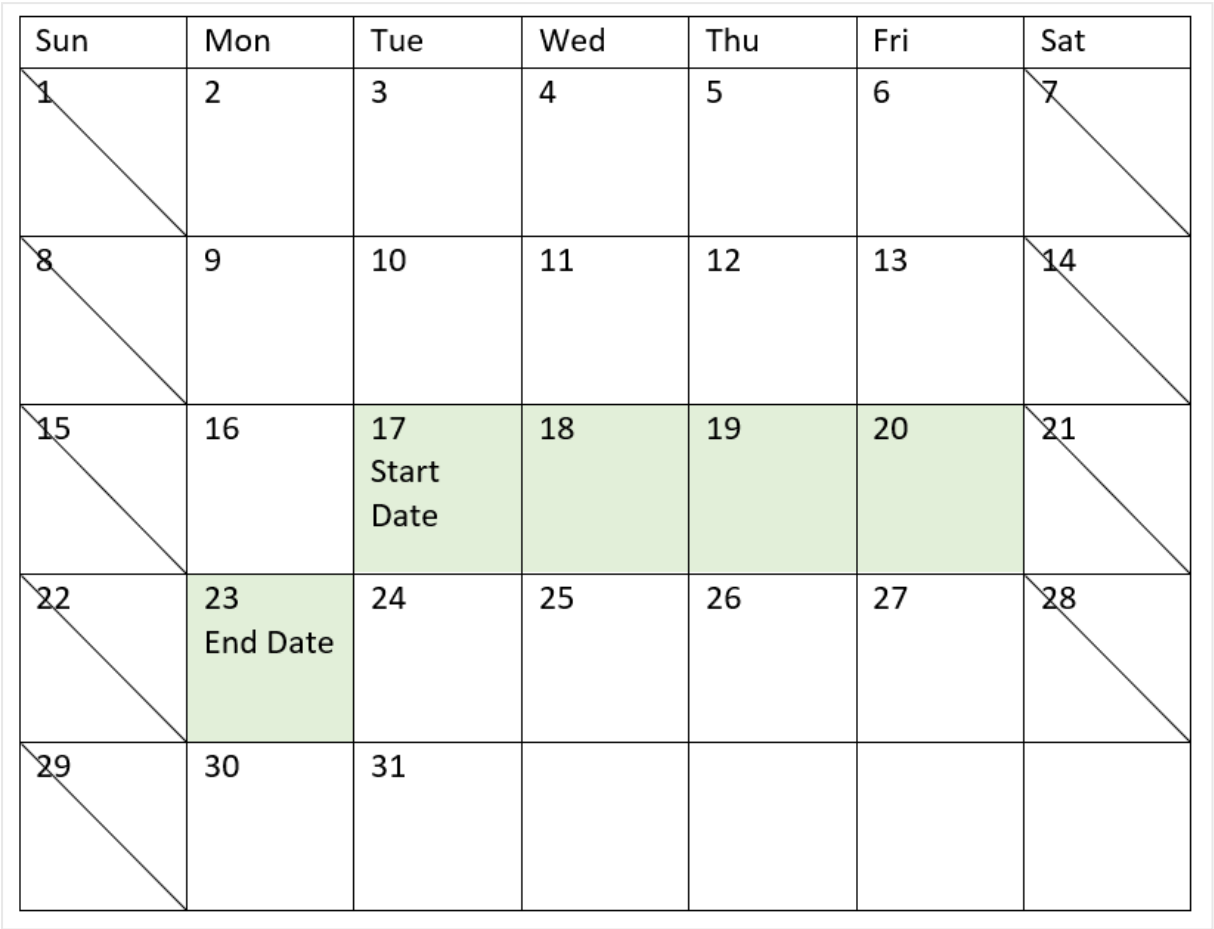
結果テーブル

ID	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/23/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

スケジュールされた休日がないため、関数は定義された業務日数 (月 ~ 金) を開始日に加えて、最も早い終了日を算出します。

次のカレンダーには、プロジェクト 3 の開始と終了日が表示されています (業務日は緑色でハイライトされている)。

プロジェクト3の開始と終了日を示すカレンダー



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18	19	20	21
22	23 End Date	24	25	26	27	28
29	30	31				

例 2 - 単一の休日

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- プロジェクトID、プロジェクト開始日、およびプロジェクトに必要な推定工数、日数を含むデータセット。「Projects」というテーブルにロードされるデータセット。
- 項目 [end_date] として設定され、各プロジェクトが終了するスケジュールを特定するlastworkdate() 関数を含む先行ロード。

ただし、2022年5月18日に休日が1日スケジュールされています。先行するLOADのlastworkdate() 関数には、各プロジェクトが終了する予定の日時を特定する第3引数に休日が含まれます。

ロード スクリプト

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

Projects:
    Load
       *,
       LastWorkDate(start_date,effort, '05/18/2022') as end_date
       ;
Load
id,
start_date,
effort
Inline
[
id,start_date,effort
1,01/01/2022,14
2,02/10/2022,17
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- effort
- end_date

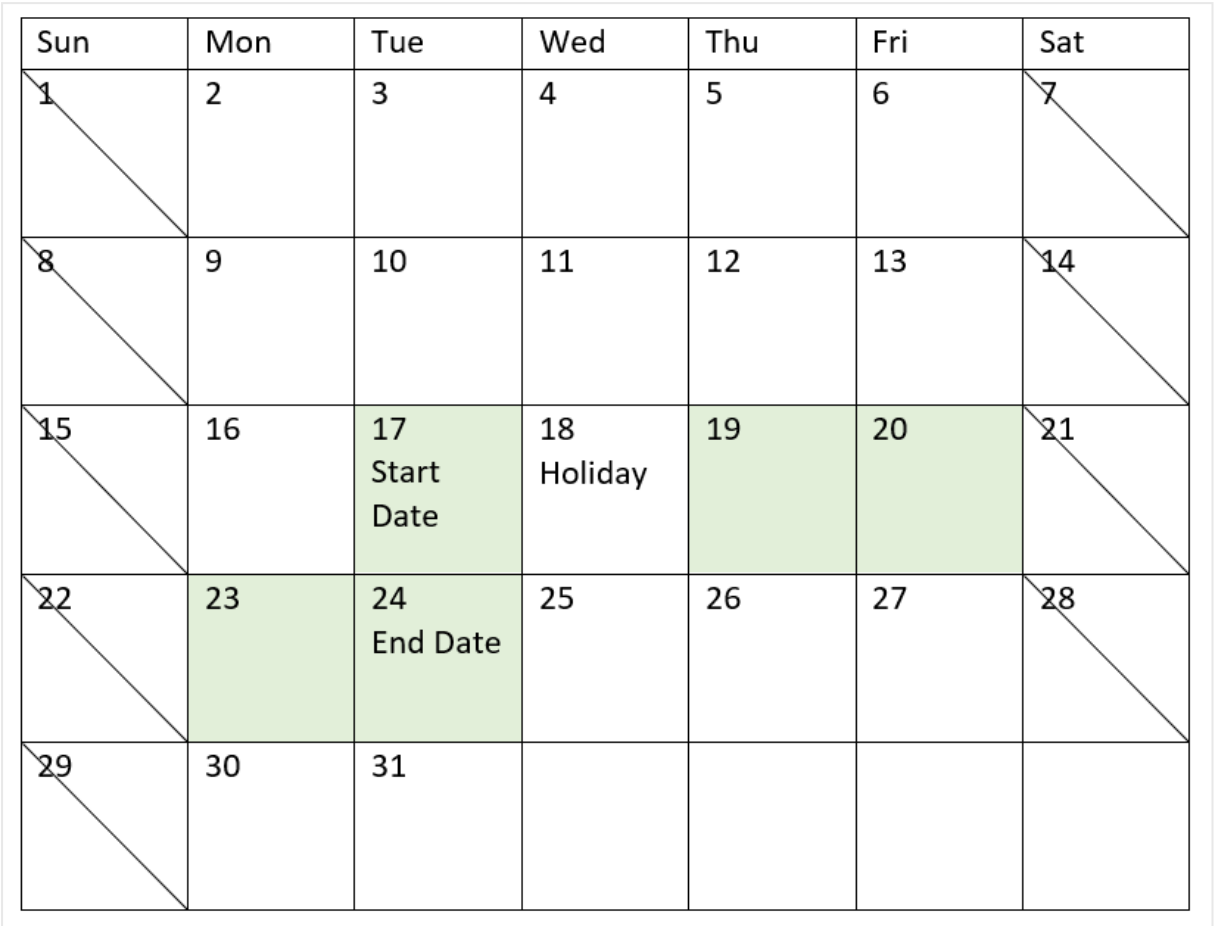
結果テーブル

ID	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/24/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

単一のスケジュールされた休日は、lastworkdate() 関数に第 3 引数として入力されます。結果として、休日が終了日までの業務日の1日に当たるため、プロジェクト3の終了日は1日後にずれます。

次のカレンダーは、プロジェクト3の開始/終了日と、休日によりプロジェクトの終了日が1日変更されることを示しています。

プロジェクト 3 の開始と終了日、休日が 5 月 18 日であることを示すカレンダー



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18 Holiday	19	20	21
22	23	24 End Date	25	26	27	28
29	30	31				

例 3 - 複数の休日

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- プロジェクトID、プロジェクト開始日、およびプロジェクトに必要な推定工数、日数を含むデータセット。「Projects」というテーブルにロードされるデータセット。
- 項目 [end_date] として設定され、各プロジェクトが終了するスケジュールを特定する lastworkdate() 関数を含む先行ロード。

ただし、5 月 19 日、20 日、21 日、および 22 日に対して 3 日の休日がスケジュールされています。先行する LOAD の lastworkdate() 関数には、各プロジェクトが終了する予定の日時を特定する第 3引数に休日が含まれます。

ロード スクリプト

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

Projects:
    Load
       *,
       LastWorkDate(start_date,effort, '05/19/2022','05/20/2022','05/21/2022','05/22/2022') as
end_date
       ;
Load
id,
start_date,
effort
Inline
[
id,start_date,effort
1,01/01/2022,14
2,02/10/2022,17
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- effort
- end_date

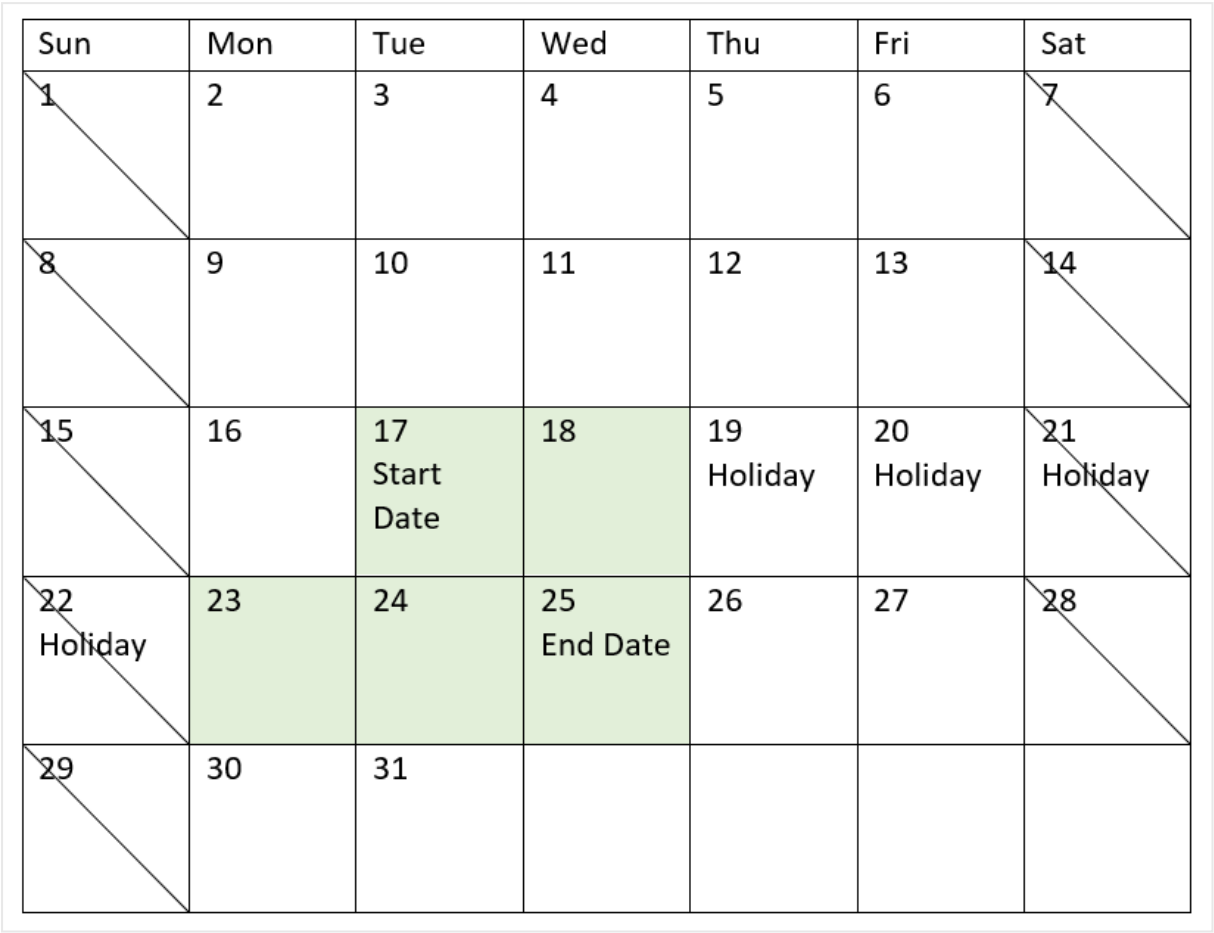
結果テーブル

ID	start_date	effort	end_date
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/25/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

4 日の休日は、開始日後の `lastworkdate()` 関数の引数と業務日数のリストとして入力されます。

次のカレンダーは、プロジェクト 3 の開始/終了日と、休日によりプロジェクトの終了日が 3 日変更されることを示しています。

プロジェクト 3 の開始と終了日、休日が 5 月 19 ~ 22 日であることを示すカレンダー



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18	19 Holiday	20 Holiday	21 Holiday
22 Holiday	23	24	25 End Date	26	27	28
29	30	31				

例 4 - 単一の休日 (チャート)

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例ではデータセットは変更されず、アプリにロードされます。[end_date] 項目は、チャートのメジャーとして計算されます。

ロード スクリプト

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

Projects:
Load
id,
start_date,
effort
Inline
[
```

```
id,start_date,effort
1,01/01/2022,14
2,02/10/2022,17
3,05/17/2022,5
4,06/01/2022,12
5,08/10/2022,26
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- effort

end_date を計算するには、次のメジャーを作成します:

- =LastWorkDate(start_date,effort,'05/18/2022')

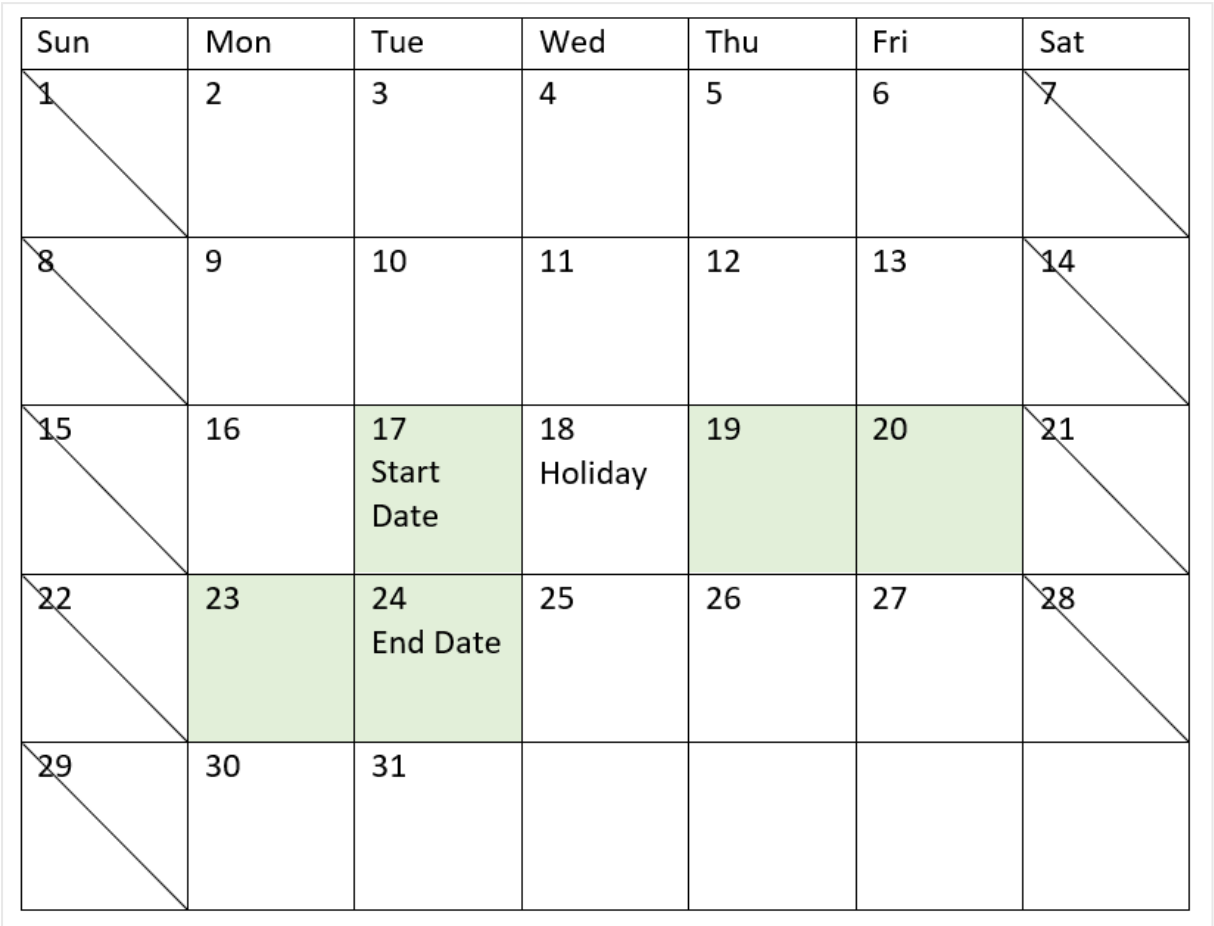
結果テーブル

ID	start_date	effort	=LastWorkDate(start_date,effort,'05/18/2022')
1	01/01/2022	14	01/20/2022
2	02/10/2022	17	03/04/2022
3	05/17/2022	5	05/23/2022
4	06/01/2022	12	06/16/2022
5	08/10/2022	26	09/14/2022

単一のスケジュールされた休日は、チャートのメジャーとして入力されます。結果として、休日が終了日までの勤務日の1日に当たるため、プロジェクト3の終了日は1日後にずれます。

次のカレンダーは、プロジェクト3の開始/終了日と、休日によりプロジェクトの終了日が1日変更されることを示しています。

プロジェクト 3 の開始と終了日、休日が 5 月 18 日であることを示すカレンダー



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17 Start Date	18 Holiday	19	20	21
22	23	24 End Date	25	26	27	28
29	30	31				

localtime

この関数は、指定されたタイム ゾーンの現在の時刻のタイムスタンプを返します。

構文:

```
LocalTime([timezone [, ignoreDST ]])
```

戻り値データ型： dual

引数

引数	説明
timezone	**timezone**は、**Windows Control Panel** の **Date and Time** で **Time Zone** にリストされているいずれかの場所を含む文字列、あるいは 'GMT+hh:mm' 形式の文字列として指定されます。受け入れ可能な場所とタイムゾーンのリストについても下の表に示されています。 タイムゾーンが指定されていない場合は、現地時間が返されます。 *DST オフセットを使用する場合 (つまり、False を評価する **ignoreDST** 引数値を指定する場合)、**place** 引数に GMT オフセットではなく場所を指定する必要があります。これは、夏時間に合わせて調整するには、GMT オフセットによって提供される経度情報に加えて、緯度情報も必要となるためです。詳細については、「GMT オフセットを DST と組み合わせて使用する (page 835)」を参照してください。*
ignoreDST	この引数が True と評価される場合、DST (夏時間) は無視されます。True と評価される有効な引数値には、-1 や True() などがあります。 この引数が False と評価された場合、タイムスタンプは夏時間に合わせて調整されます。False と評価される有効な引数値には、0 や False() などがあります。 **ignoreDST** 引数値が無効な場合、関数は **ignore_dst** 値が True と評価されるかのように数式を評価します。**ignoreDST** 引数値が指定されていない場合、関数は **ignore_dst** 値が False と評価されるかのように数式を評価します。

有効な場所とタイム ゾーン

A-C	D-K	L-R	S-Z
Abu Dhabi	Darwin	La Paz	Samoa
Adelaide	Dhaka	Lima	Santiago
Alaska	Eastern Time (US & Canada)	Lisbon	Sapporo
Amsterdam	Edinburgh	Ljubljana	Sarajevo
Arizona	Ekaterinburg	London	Saskatchewan
Astana	Fiji	Madrid	Seoul
Athens	Georgetown	Magadan	Singapore
Atlantic Time (Canada)	Greenland	Mazatlan	Skopje

A-C	D-K	L-R	S-Z
Auckland	Greenwich Mean Time : Dublin	Melbourne	Sofia
Azores	Guadalajara	Mexico City	Solomon Is.
Baghdad	Guam	Mid-Atlantic	Sri Jayawardenepura
Baku	Hanoi	Minsk	St. Petersburg
Bangkok	Harare	Monrovia	Stockholm
Beijing	Hawaii	Monterrey	Sydney
Belgrade	Helsinki	Moscow	Taipei
Berlin	Hobart	Mountain Time (US & Canada)	Tallinn
Bern	Hong Kong	Mumbai	Tashkent
Bogota	Indiana (East)	Muscat	Tbilisi
Brasilia	International Date Line West	Nairobi	Tehran
Bratislava	Irkutsk	New Caledonia	Tokyo
Brisbane	Islamabad	New Delhi	Urumqi
Brussels	Istanbul	Newfoundland	Warsaw
Bucharest	Jakarta	Novosibirsk	Wellington
Budapest	Jerusalem	Nuku'alofa	West Central Africa
Buenos Aires	Kabul	Osaka	Vienna
Cairo	Kamchatka	Pacific Time (US & Canada)	Vilnius
Canberra	Karachi	Paris	Vladivostok
Cape Verde Is.	Kathmandu	Perth	Volgograd
Caracas	Kolkata	Port Moresby	Yakutsk
Casablanca	Krasnoyarsk	Prague	Yerevan
Central America	Kuala Lumpur	Pretoria	Zagreb
Central Time (US & Canada)	Kuwait	Quito	-
Chennai	Kyiv	Riga	-
Chihuahua	-	Riyadh	-

A-C	D-K	L-R	S-Z
Chongqing	-	Rome	-
Copenhagen	-	-	-

例と結果:

以下の例は、現地時間 2023-08-14 08:39:47 に呼び出される関数に基づいており、サーバーまたはデスクトップ環境の現地のタイム ゾーンは GMT-05:00 で、このリストの日付時点で夏時間が導入されています。

スクリプトの例

例	結果
`localtime ()`	現地時間 2023-08-14 08:39:47 を返します。
`localtime ('London')`	ロンドンの現地時間 2023-08-14 13:39:47 を返します。
`localtime ('GMT+02:00')`	GMT+02:00 のタイムゾーンの現地時間 2023-08-14 14:39:47 を返します。場所ではなくGMT オフセットが指定されているため、夏時間の調整は行われません。
`localtime ('Paris',-1)`	夏時間を無視したパリの現地時間 2023-08-14 13:39:47 を返します。
`localtime ('Paris',True())`	夏時間を無視したパリの現地時間 2023-08-14 13:39:47 を返します。
`localtime ('Paris',0)`	夏時間を考慮したパリの現地時間 2023-08-14 14:39:47 を返します。
`localtime ('Paris',False ())`	夏時間を考慮したパリの現地時間 2023-08-14 14:39:47 を返します。

GMT オフセットを DST と組み合わせて使用する

Qlik Sense で International Components for Unicode (ICU) ライブラリを実装した後、GMT (グリニッジ標準時) オフセットを DST (夏時間) と組み合わせて使用するには、追加の緯度情報が必要になります。

GMT は経度 (東西) オフセットであるのに対し、DST は緯度 (南北) オフセットです。例えば、ヘルシンキ (フィンランド) とヨハネスブルグ (南アフリカ) は同じ GMT+02:00 オフセットを共有しますが、同じ DST オフセットは共有しません。つまり、現地の DST 条件に関する完全な情報を得るために、GMT オフセットに加えて、DST オフセットでも現地のタイム ゾーンの緯度位置に関する情報 (地理的タイム ゾーン入力) が必要となります。

lunarweekend

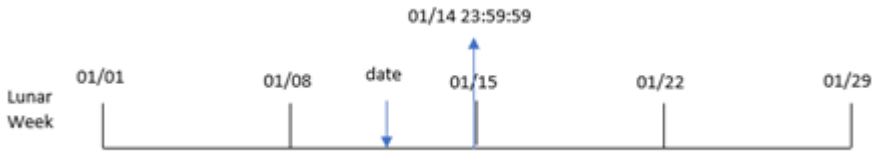
この関数は、**date** を含む週周期の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて正確に 7 日構成となります。

構文:

```
LunarweekEnd(date[, period_no[, first_week_day]])
```

戻り値データ型： dual

lunarweekend() 関数の図の例



lunarweekend() 関数は、date がどの旧暦の週に当たるかを決定します。次に、その週の最後のミリ秒のタイムスタンプを日付形式で返します。

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数または計算結果が整数になる数式で、値 0 は **date** を含む週周期を示します。**period_no** の値が負の場合は過去の週周期を、正の場合は将来の週周期を示します。
first_week_day	0 未満または 0 よりも大きい補正値。日数または 1 日未満の長さ、またはその両方を指定して、年の開始時点を変更できます。

使用に適しているケース

lunarweekend() 関数は、ユーザーがまだ発生していない週の端数を計算に使用する場合に、数式の一部として一般的に使用されます。weekend() 関数と異なり、各暦年の旧暦最終週は 12 月 31 日に終了します。例えば lunarweekend() 関数は、その週にまだ発生していない利息を計算するために使用することができます。

関数の例

例	結果
`lunarweekend('01/12/2013')`	01/14/2013 23:59:59 を返します。
`lunarweekend('01/12/2013', -1)`	01/07/2013 23:59:59 を返します。
`lunarweekend('01/12/2013', 0, 1)`	01/15/2013 23:59:59 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する旧暦の週の終わりのタイムスタンプを返す、項目 [end_of_week] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        lunarweekend(date) as end_of_week,
        timestamp(lunarweekend(date)) as end_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
```

```
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- end_of_week
- end_of_week_timestamp

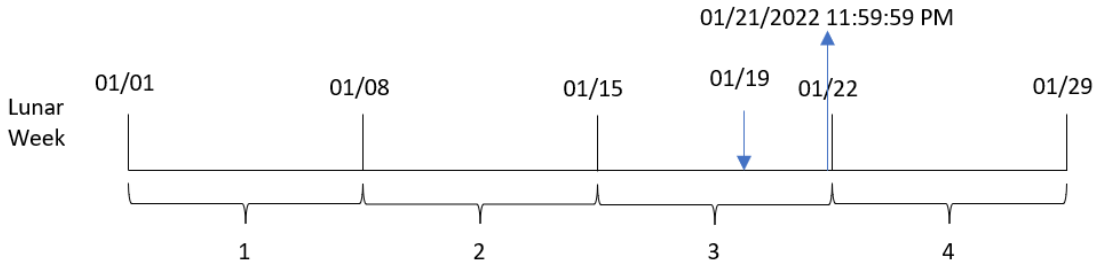
結果テーブル

日付	end_of_week	end_of_week_timestamp
1/7/2022	01/07/2022	1/7/2022 11:59:59 PM
1/19/2022	01/21/2022	1/21/2022 11:59:59 PM
2/5/2022	02/11/2022	2/11/2022 11:59:59 PM
2/28/2022	03/04/2022	3/4/2022 11:59:59 PM
3/16/2022	03/18/2022	3/18/2022 11:59:59 PM
4/1/2022	04/01/2022	4/1/2022 11:59:59 PM
5/7/2022	05/13/2022	5/13/2022 11:59:59 PM
5/16/2022	05/20/2022	5/20/2022 11:59:59 PM
6/15/2022	06/17/2022	6/17/2022 11:59:59 PM
6/26/2022	07/01/2022	7/1/2022 11:59:59 PM
7/9/2022	07/15/2022	7/15/2022 11:59:59 PM
7/22/2022	07/22/2022	7/22/2022 11:59:59 PM
7/23/2022	07/29/2022	7/29/2022 11:59:59 PM
7/27/2022	07/29/2022	7/29/2022 11:59:59 PM
8/2/2022	08/05/2022	8/5/2022 11:59:59 PM
8/8/2022	08/12/2022	8/12/2022 11:59:59 PM
8/19/2022	08/19/2022	8/19/2022 11:59:59 PM
9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
10/14/2022	10/14/2022	10/14/2022 11:59:59 PM
10/29/2022	11/04/2022	11/4/2022 11:59:59 PM

end_of_week 項目は、lunarweekend() 関数を使用し、関数の引数として date 項目を渡すことにより、先行する LOAD ステートメントで作成されます。

lunarweekend() 関数は、日付値がどの旧暦週に該当するかを識別し、その週の最後のミリ秒のタイムスタンプを返します。

lunarweekend() 関数の図、追加の引数がない例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekend() 関数は、旧暦の週が 1 月 15 日に開始することを特定します。そのため、トランザクションの end_of_week 値は、旧暦の週の最後のミリ秒である 1 月 21 日 11:59:59 PM を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の旧暦の週の終わりのタイムスタンプを返す、項目 [previous_lunar_week_end] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        lunarweekend(date,-1) as previous_lunar_week_end,
        timestamp(lunarweekend(date,-1)) as previous_lunar_week_end_timestamp
        ;
Load
*
Inline
[
id,date,amount
```

```
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_lunar_week_end
- previous_lunar_week_end_timestamp

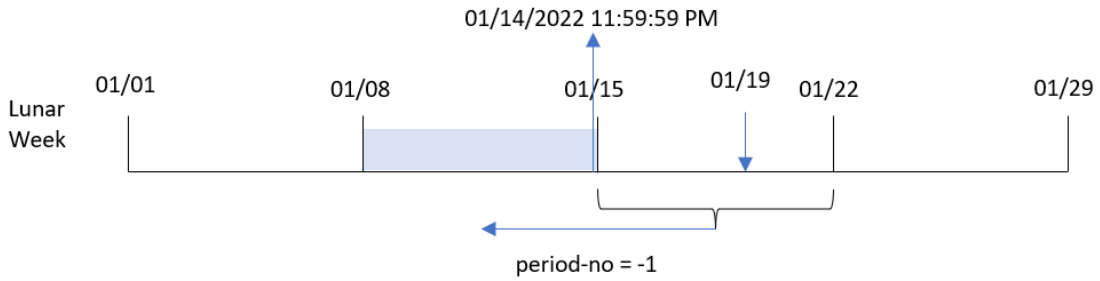
結果テーブル

日付	previous_lunar_week_end	previous_lunar_week_end_timestamp
1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
1/19/2022	01/14/2022	1/14/2022 11:59:59 PM
2/5/2022	02/04/2022	2/4/2022 11:59:59 PM
2/28/2022	02/25/2022	2/25/2022 11:59:59 PM
3/16/2022	03/11/2022	3/18/2022 11:59:59 PM
4/1/2022	03/25/2022	3/25/2022 11:59:59 PM
5/7/2022	05/06/2022	5/6/2022 11:59:59 PM
5/16/2022	05/13/2022	5/13/2022 11:59:59 PM
6/15/2022	06/10/2022	6/10/2022 11:59:59 PM
6/26/2022	06/24/2022	6/24/2022 11:59:59 PM
7/9/2022	07/08/2022	7/8/2022 11:59:59 PM

日付	previous_lunar_week_end	previous_lunar_week_end_timestamp
7/22/2022	07/15/2022	7/15/2022 11:59:59 PM
7/23/2022	07/22/2022	7/22/2022 11:59:59 PM
7/27/2022	07/22/2022	7/22/2022 11:59:59 PM
8/2/2022	07/29/2022	7/29/2022 11:59:59 PM
8/8/2022	08/05/2022	8/5/2022 11:59:59 PM
8/19/2022	08/12/2022	8/12/2022 11:59:59 PM
9/26/2022	09/23/2022	9/23/2022 11:59:59 PM
10/14/2022	10/07/2022	10/7/2022 11:59:59 PM
10/29/2022	10/28/2022	10/28/2022 11:59:59 PM

この例では、-1 の period_no が lunarweekend() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した旧暦の週を識別します。次に、1 週間前にずらして、旧暦のその週の最後のミリ秒を識別します。

lunarweekend() 関数の図、period_no の例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekend() 関数は、旧暦の週が 1 月 15 日に開始することを特定します。そのため、旧暦の前の週は 1 月 8 日に開始され、1 月 14 日の 11:59:59 PM に終了しました。これは、[previous_lunar_week_end] 項目に対して返される値です。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。この例では、旧暦の週が 1 月 5 日に始まるよう設定しています。

ロード スクリプト

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

Transactions:
    Load
        *,
        lunarweekend(date,0,4) as end_of_week,
timestamp(lunarweekend(date,0,4)) as end_of_week_timestamp
;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- end_of_week
- end_of_week_timestamp

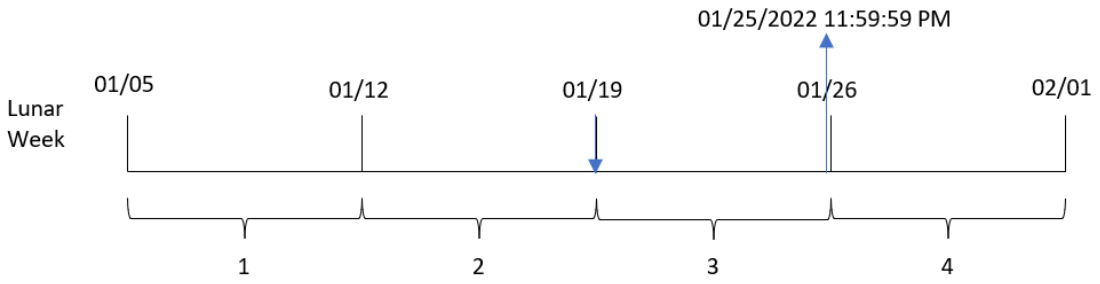
結果テーブル

日付	end_of_week	end_of_week_timestamp
1/7/2022	01/11/2022	1/11/2022 11:59:59 PM
1/19/2022	01/25/2022	1/25/2022 11:59:59 PM

日付	end_of_week	end_of_week_timestamp
2/5/2022	02/08/2022	2/8/2022 11:59:59 PM
2/28/2022	03/01/2022	3/1/2022 11:59:59 PM
3/16/2022	03/22/2022	3/22/2022 11:59:59 PM
4/1/2022	04/05/2022	4/5/2022 11:59:59 PM
5/7/2022	05/10/2022	5/10/2022 11:59:59 PM
5/16/2022	05/17/2022	5/17/2022 11:59:59 PM
6/15/2022	06/21/2022	6/21/2022 11:59:59 PM
6/26/2022	06/28/2022	6/28/2022 11:59:59 PM
7/9/2022	07/12/2022	7/12/2022 11:59:59 PM
7/22/2022	07/26/2022	7/26/2022 11:59:59 PM
7/23/2022	07/26/2022	7/26/2022 11:59:59 PM
7/27/2022	08/02/2022	8/2/2022 11:59:59 PM
8/2/2022	08/02/2022	8/2/2022 11:59:59 PM
8/8/2022	08/09/2022	8/9/2022 11:59:59 PM
8/19/2022	08/23/2022	8/23/2022 11:59:59 PM
9/26/2022	09/27/2022	9/27/2022 11:59:59 PM
10/14/2022	10/18/2022	10/18/2022 11:59:59 PM
10/29/2022	11/01/2022	11/1/2022 11:59:59 PM

このインスタンスでは、first_week_date 引数である 4 が lunarweekend() 関数で使用されるため、1 月 1 日から 1 月 5 日に年の初めがオフセットされます。

lunarweekend() 関数、first_week_day 例の図



トランザクション 8189 は 1 月 19 日に発生しました。旧暦の週が 1 月 5 日に始まる前、lunarweekend() 関数は 1 月 19 日を含む旧暦の週も 1 月 19 日に始まることを特定します。そのため、旧暦の週の終わりは 1 月 25 日 11:59:59 PM に当たります。これは、[end_of_week] 項目に対して返される値です。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した旧暦の週の終わりのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。

次のメジャーを追加します。

```
=lunarweekend(date)
```

```
=timestamp(lunarweekend(date))
```

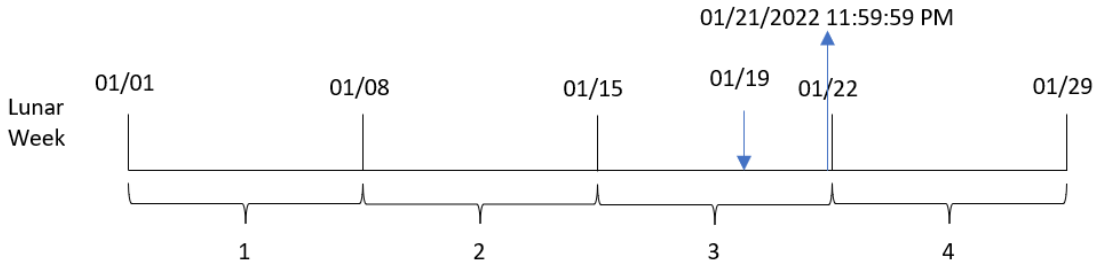
結果テーブル

日付	=lunarweekend(date)	=timestamp(lunarweekend(date))
1/7/2022	01/07/2022	1/7/2022 11:59:59 PM
1/19/2022	01/21/2022	1/21/2022 11:59:59 PM
2/5/2022	02/11/2022	2/11/2022 11:59:59 PM
2/28/2022	03/04/2022	3/4/2022 11:59:59 PM
3/16/2022	03/18/2022	3/18/2022 11:59:59 PM
4/1/2022	04/01/2022	4/1/2022 11:59:59 PM
5/7/2022	05/13/2022	5/13/2022 11:59:59 PM
5/16/2022	05/20/2022	5/20/2022 11:59:59 PM
6/15/2022	06/17/2022	6/17/2022 11:59:59 PM
6/26/2022	07/01/2022	7/1/2022 11:59:59 PM
7/9/2022	07/15/2022	7/15/2022 11:59:59 PM
7/22/2022	07/22/2022	7/22/2022 11:59:59 PM
7/23/2022	07/29/2022	7/29/2022 11:59:59 PM
7/27/2022	07/29/2022	7/29/2022 11:59:59 PM
8/2/2022	08/05/2022	8/5/2022 11:59:59 PM
8/8/2022	08/12/2022	8/12/2022 11:59:59 PM
8/19/2022	08/19/2022	8/19/2022 11:59:59 PM
9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
10/14/2022	10/14/2022	10/14/2022 11:59:59 PM
10/29/2022	11/04/2022	11/4/2022 11:59:59 PM

[end_of_week] メジャーは、lunarweekend() 関数を使用し、関数の引数として [date] 項目を渡すことにより、チャートオブジェクトで作成されます。

lunarweekend() 関数は、日付値がどの旧暦週に該当するかを識別し、その週の最後のミリ秒のタイムスタンプを返します。

lunarweekend() 関数の図、チャート オブジェクトの例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekend() 関数は、旧暦の週が 1 月 15 日に開始することを特定します。そのため、トランザクションの end_of_week 値は、旧暦の週の最後のミリ秒である 1 月 21 日 11:59:59 PM を返します。

例 5 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 「Employee_Expenses」というテーブルにロードされるデータセット。
- 従業員 ID、従業員名および各従業員の平均日次経費請求。

エンド ユーザーは、従業員 ID と従業員名別に、旧暦のその週の残りの期間にまだ発生する推定経費請求を表示するグラフ オブジェクトを求めています。

ロード スクリプト

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の項目を軸として追加します。
   - employee_id
   - employee_name
3. 次に、次のメジャーを作成して、累積利息を計算します。
   =(lunarweekend(today(1))-today(1))*avg_daily_claim
4. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

employee_id	employee_name	=(lunarweekend(today(1))-today(1))*avg_daily_claim
182	Mark	$75.00
183	Deryck	$62.50
184	Dexter	$62.50
185	Sydney	$135.00
186	Agatha	$90.00

lunarkweekend() 関数は、今日の日付を唯一の引数として使用することにより、現在の旧暦の週の終了日を返します。次に、旧暦の週の終了日から今日の日付を引くことによって、数式は今週の残りの日数を返します。

次に、この値に各従業員による1日あたりの平均経費請求額を乗算して、旧暦の週の残り期間に各従業員が行うと予想される請求の推定額を計算します。

lunarweekname

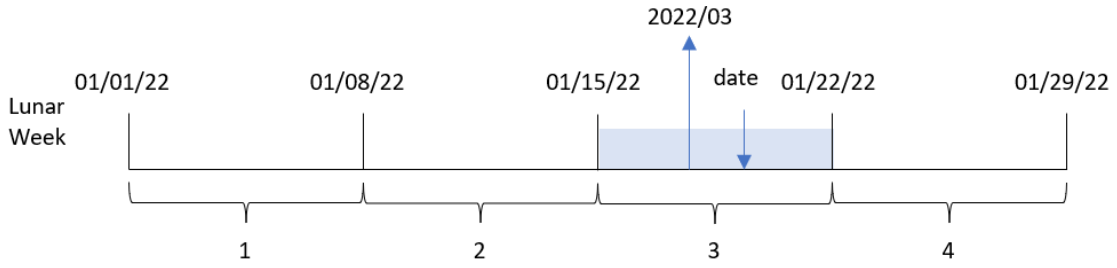
この関数は、**date** を含む週周期の初日の最初のミリ秒のタイムスタンプに対応する年と週周期番号を表示する表示値を返します。Qlik Sense の旧暦の週は、1月1日を週の初日として数えるよう定義され、1年の最終週を除いて正確に7日構成となります。

構文:

```
LunarWeekName(date [, period_no[, first_week_day]])
```

戻り値データ型： dual

lunarweekname() 関数の図の例



lunarweekname() 関数は、日付が1月1日からの週カウントから始まる旧暦のどの週に当てはまるかを決定します。次に、year/weekcount から派生した値を返します。

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数または計算結果が整数になる数式で、値 0 は **date** を含む週周期を示します。**period_no** の値が負の場合は過去の週周期を、正の場合は将来の週周期を示します。
first_week_day	0 未満または 0 よりも大きい補正値。日数または 1 日未満の長さ、またはその両方を指定して、年の開始時点を変更できます。

使用に適しているケース

lunarweekname() 関数は、集計を旧暦の週単位で比較する場合に便利です。例えば、関数は旧暦の週当たりの製品の合計売上を決定するために使用できます。旧暦の週は、年の最初の週に含まれるすべての値が、最短で1月1日からの値だけを含む場合に役立ちます。

これらの軸は、関数を使用してマスター カレンダー テーブルに項目を作成することにより、ロード スクリプトで作成できます。関数は、計算軸としてチャートで直接使用することもできます。

関数の例

例	結果
`lunarweekname('01/12/2013')`	2006/02 を返します。
`lunarweekname('01/12/2013', -1)`	2006/01 を返します。
`lunarweekname('01/12/2013', 0, 1)`	2006/02 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数がない日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する旧暦の週の年と週番号を返す、項目 [lunar_week_name] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        lunarweekname(date) as lunar_week_name
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
```

```
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- lunar_week_name

結果テーブル

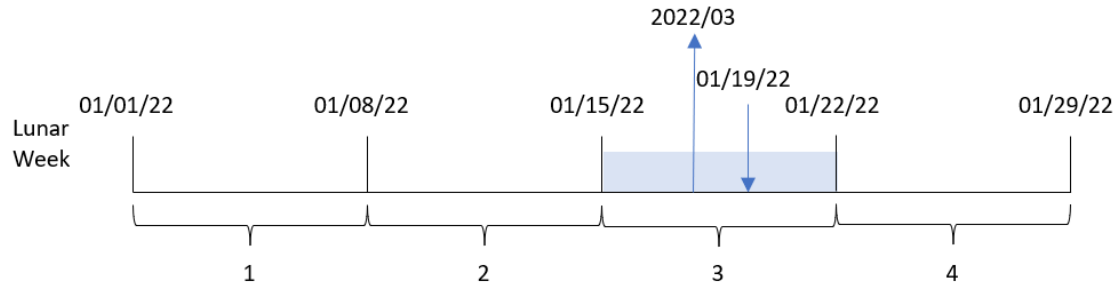
日付	lunar_week_name
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/06
2/28/2022	2022/09
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/19
5/16/2022	2022/20
6/15/2022	2022/24
6/26/2022	2022/26
7/9/2022	2022/28
7/22/2022	2022/29
7/23/2022	2022/30
7/27/2022	2022/30
8/2/2022	2022/31
8/8/2022	2022/32
8/19/2022	2022/33

日付	lunar_week_name
9/26/2022	2022/39
10/14/2022	2022/41
10/29/2022	2022/44

lunar_week_name 項目は、lunarweekname() 関数を使用し、関数の引数として date 項目を渡すことにより、先行する LOAD ステートメントで作成されます。

lunarweekname() 関数は、日付値がどの旧暦の週に該当するかを識別し、その日の年と週番号を返します。

lunarweekname() 関数の図、追加の引数がない例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekname() 関数は、この日付が 1 月 15 日に始まる旧暦の週 (年の旧暦第 3 の週) に当たることを特定します。そのため、トランザクションに対して返された lunar_week_name 値は 2022/03 です。

例 2 – period_no 引数を持つ日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の旧暦の週の年と週番号を返す、項目 [previous_lunar_week_name] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        lunarweekname(date,-1) as previous_lunar_week_name
        ;
```

```
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_lunar_week_name

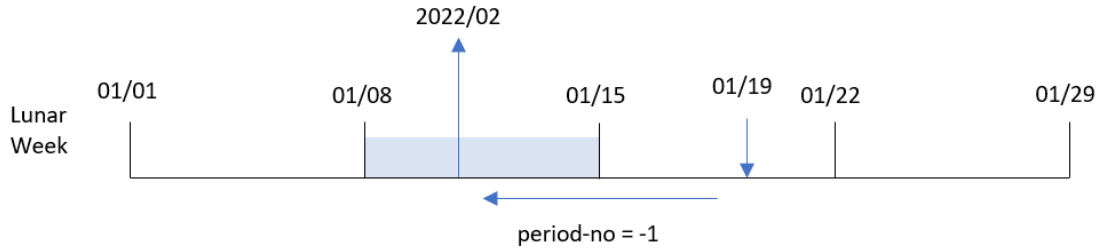
結果テーブル

日付	previous_lunar_week_name
1/7/2022	2021/52
1/19/2022	2022/02
2/5/2022	2022/05
2/28/2022	2022/08
3/16/2022	2022/10
4/1/2022	2022/12
5/7/2022	2022/18
5/16/2022	2022/19
6/15/2022	2022/23

日付	previous_lunar_week_name
6/26/2022	2022/25
7/9/2022	2022/27
7/22/2022	2022/28
7/23/2022	2022/29
7/27/2022	2022/29
8/2/2022	2022/30
8/8/2022	2022/31
8/19/2022	2022/32
9/26/2022	2022/38
10/14/2022	2022/40
10/29/2022	2022/43

この例では、-1 の period_no が lunarweekname() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した旧暦の週を識別します。次に、年と1週間前の番号を返します。

lunarweekname() 関数の図、period_no の例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekname() 関数は、このトランザクションが年の旧暦の第三週に発生したことを特定し、次に [previous_lunar_week_name] 項目に対して年と1週間前の値 2022/02 を返します。

例 3 – first_week_day 引数を持つ日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。この例では、旧暦の週が1月 5 日に始まるよう設定しています。

ロード スクリプト

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

Transactions:
    Load
        *,
        lunarweekname(date,0,4) as lunar_week_name
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

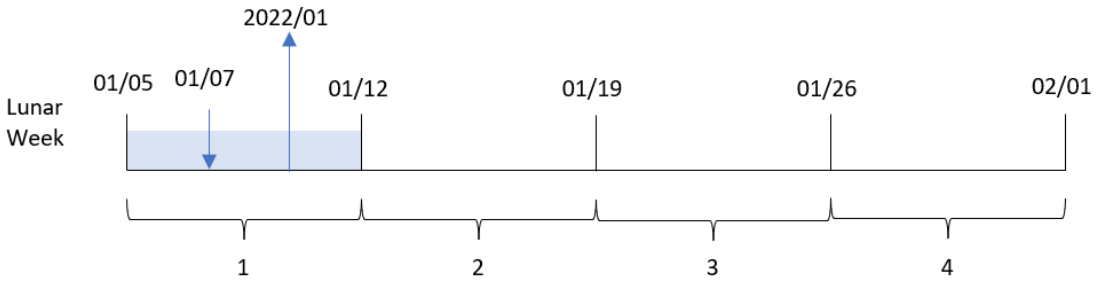
- date
- lunar_week_name

結果テーブル

日付	lunar_week_name
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/05
2/28/2022	2022/08

日付	lunar_week_name
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/18
5/16/2022	2022/19
6/15/2022	2022/24
6/26/2022	2022/25
7/9/2022	2022/27
7/22/2022	2022/29
7/23/2022	2022/29
7/27/2022	2022/30
8/2/2022	2022/30
8/8/2022	2022/31
8/19/2022	2022/33
9/26/2022	2022/38
10/14/2022	2022/41
10/29/2022	2022/43

lunarweekname() 関数、first_week_day 例の図



このインスタンスでは、first_week_date 引数である 4 が lunarweekname() 関数で使用されるため、1 月 1 日から 1 月 5 日に旧暦の週の初めがオフセットされます。

トランザクション 8188 は 1 月 7 日に発生しました。旧暦の週が 1 月 5 日に始まるため、lunarweekname() 関数は 1 月 7 日を含む旧暦の週が年の最初の旧暦の週であることを特定します。そのため、トランザクションに対して返された lunar_week_name 値は 2022/01 です。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した旧暦の週の週と年の番号を返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `date`。

トランザクションが発生する旧暦の週の開始日を計算するには、次のメジャーを作成します:

```
=lunarweekname(date)
```

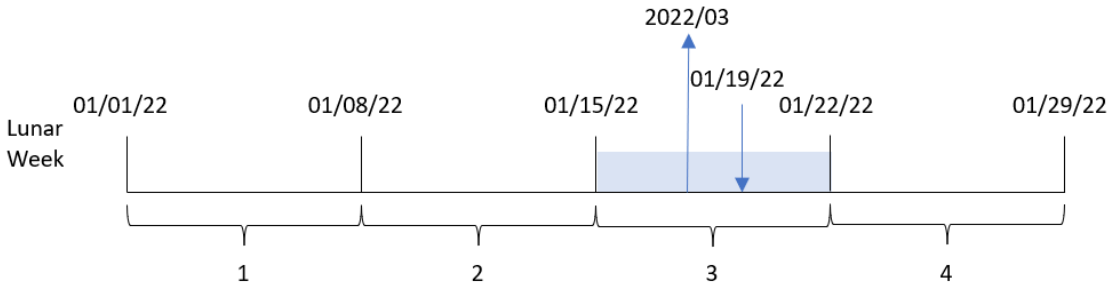
結果テーブル

日付	=lunarweekname(date)
1/7/2022	2022/01
1/19/2022	2022/03
2/5/2022	2022/06
2/28/2022	2022/09
3/16/2022	2022/11
4/1/2022	2022/13
5/7/2022	2022/19
5/16/2022	2022/20
6/15/2022	2022/24
6/26/2022	2022/26
7/9/2022	2022/28
7/22/2022	2022/29
7/23/2022	2022/30
7/27/2022	2022/30
8/2/2022	2022/31
8/8/2022	2022/32
8/19/2022	2022/33
9/26/2022	2022/39
10/14/2022	2022/41
10/29/2022	2022/44

[lunar_week_name] メジャーは、lunarweekname() 関数を使用し、関数の引数として [date] 項目を渡すことにより、チャートオブジェクトで作成されます。

lunarweekname() 関数は、日付値がどの旧暦の週に該当するかを識別し、その日の年と週番号を返します。

lunarweekname() 関数の図、チャート オブジェクトの例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekname() 関数は、この日付が 1 月 15 日に始まる旧暦の週 (年の旧暦第 3 の週) に当たることを特定します。そのため、トランザクションに対する lunar_week_name 値は 2022/03 です。

例 5 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。

エンド ユーザーは、現行年の週ごとの総売上高を示すチャート オブジェクトを求めています。第 1 週 (7 日間) は 1 月 1 日に始まる必要があります。これは、lunarweekname() 関数をチャートの計算軸として使用することにより、この軸がデータ モデルにない場合でも実現できます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
```

```
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の式を使用して計算軸を作成します。
   `=lunarweekname(date)`
3. 次の集計メジャーを使って、総売上を計算します。
   `=sum(amount)`
4. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

=lunarweekname(date)	=sum(amount)
2022/01	$17.17
2022/03	$37.23
2022/06	$57.42
2022/09	$88.27
2022/11	$53.80
2022/13	$82.06
2022/19	$40.39
2022/20	$87.21
2022/24	$95.93
2022/26	$45.89
2022/28	$36.23
2022/29	$25.66
2022/30	$152.75
2022/31	$76.11

=lunarweekname(date)	=sum(amount)
2022/32	$25.12
2022/33	$46.23
2022/39	$84.21
2022/41	$96.24
2022/44	$67.67

lunarweekstart

この関数は、**date** を含む週周期の初日の最初のミリ秒のタイムスタンプに相当する値を返します。Qlik Sense の旧暦の週は、1 月 1 日を週の初日として数えるよう定義され、1 年の最終週を除いて正確に 7 日構成となります。

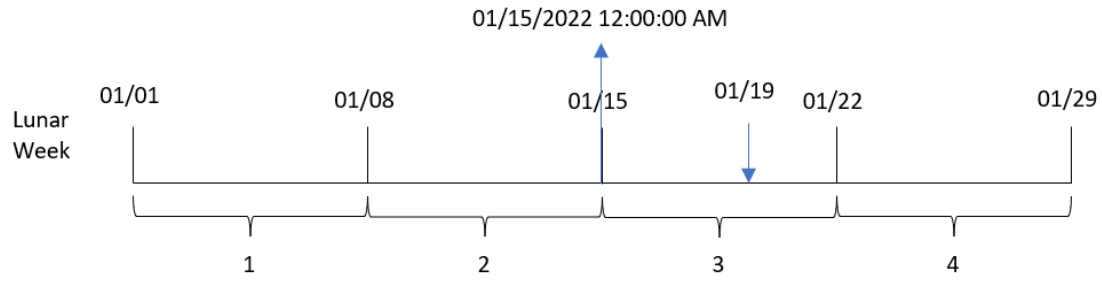
構文:

```
LunarweekStart(date[, period_no[, first_week_day]])
```

戻り値データ型: dual

`lunarweekstart()` 関数は、`date` がどの旧暦の週に当たるかを決定します。次に、その年の最初のミリ秒のタイムスタンプを日付形式で返します。

`lunarweekstart()` 関数の図の例



引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数または計算結果が整数になる数式で、値 0 は **date** を含む週周期を示します。**period_no** の値が負の場合は過去の週周期を、正の場合は将来の週周期を示します。
first_week_day	0 未満または 0 よりも大きい補正値。日数または 1 日未満の長さ、またはその両方を指定して、年の開始時点を変更できます。

使用に適しているケース

lunarweekstart() 関数は、ユーザーがこれまで経過した週の端数を計算に使用する場合に、数式の一部として一般的に使用されます。weekstart() 関数と異なり、各カレンダー年の始めに、週は 1 月 1 日に始まり、それに続く隔週は 7 日後ごとに始まります。lunarweekstart() 関数は FirstWeekDay システム変数による影響を受けません。

たとえば、lunarweekstart() は日付までの週に累積した利息を計算するのに使用できます。

関数の例

例	結果
lunarweekstart ('01/12/2013')	01/08/2013 を返します。
lunarweekstart ('01/12/2013', -1)	01/01/2013 を返します。
lunarweekstart ('01/12/2013', 0, 1)	first_week_day を 1 に設定するということは、年の初めが 01/02/2013 に変更されるということであるため、01/09/2013 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する旧暦の週の初めのタイムスタンプを返す、項目 [start_of_week] の作成。

ロード スクリプト

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

Transactions:
    Load
        *,
        lunarweekstart(date) as start_of_week,
        timestamp(lunarweekstart(date)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- start_of_week
- start_of_week_timestamp

結果テーブル

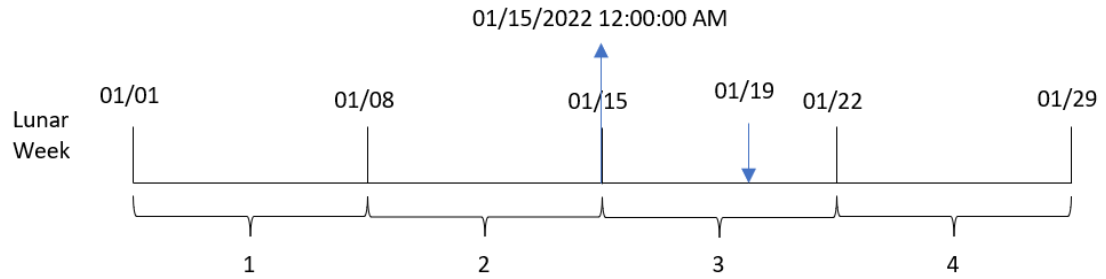
日付	start_of_week	start_of_week_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/15/2022	1/15/2022 12:00:00 AM

日付	start_of_week	start_of_week_timestamp
2/5/2022	02/05/2022	2/5/2022 12:00:00 AM
2/28/2022	02/26/2022	2/26/2022 12:00:00 AM
3/16/2022	03/12/2022	3/12/2022 12:00:00 AM
4/1/2022	03/26/2022	3/26/2022 12:00:00 AM
5/7/2022	05/07/2022	5/7/2022 12:00:00 AM
5/16/2022	05/14/2022	5/14/2022 12:00:00 AM
6/15/2022	06/11/2022	6/11/2022 12:00:00 AM
6/26/2022	06/25/2022	6/25/2022 12:00:00 AM
7/9/2022	07/09/2022	7/9/2022 12:00:00 AM
7/22/2022	07/16/2022	7/16/2022 12:00:00 AM
7/23/2022	07/23/2022	7/23/2022 12:00:00 AM
7/27/2022	07/23/2022	7/23/2022 12:00:00 AM
8/2/2022	07/30/2022	7/30/2022 12:00:00 AM
8/8/2022	08/06/2022	8/6/2022 12:00:00 AM
8/19/2022	08/13/2022	8/13/2022 12:00:00 AM
9/26/2022	09/24/2022	9/24/2022 12:00:00 AM
10/14/2022	10/08/2022	10/8/2022 12:00:00 AM
10/29/2022	10/29/2022	10/29/2022 12:00:00 AM

start_of_week 項目は、lunarweekstart() 関数を使用し、関数の引数として date 項目を渡すことにより、先行するLOAD ステートメントで作成されます。

lunarweekstart() 関数は、日付がどの旧暦の週に該当するかを識別し、その週の最初のミリ秒のタイムスタンプを返します。

lunarweekstart() 関数の図、追加の引数がない例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekstart() 関数は、旧暦の週が 1 月 15 日に開始することを特定します。そのため、トランザクションの start_of_week 値は、その日の最初のミリ秒である 1 月 15 日 12:00:00 AM を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の旧暦の週の始めのタイムスタンプを返す、項目 [previous_lunar_week_start] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        lunarweekstart(date,-1) as previous_lunar_week_start,
        timestamp(lunarweekstart(date,-1)) as previous_lunar_week_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
```

```
8207,10/29/2022,67.67
];
```

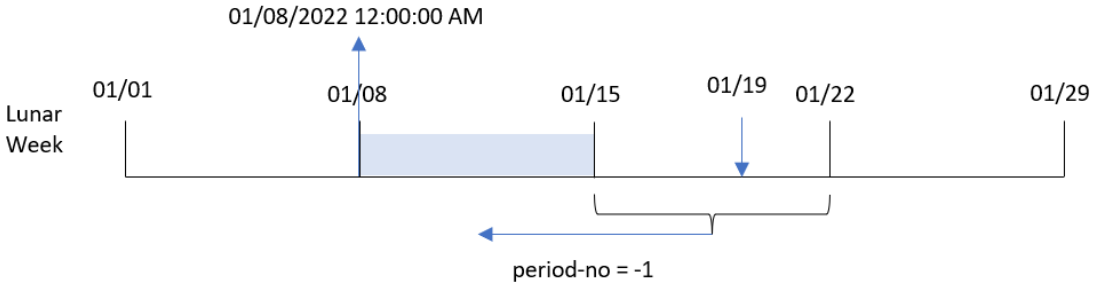
結果

結果 テーブル

日付	previous_lunar_week_start	previous_lunar_week_start_timestamp
1/7/2022	12/24/2021	12/24/2021 12:00:00 AM
1/19/2022	01/08/2022	1/8/2022 12:00:00 AM
2/5/2022	01/29/2022	1/29/2022 12:00:00 AM
2/28/2022	02/19/2022	2/19/2022 12:00:00 AM
3/16/2022	03/05/2022	3/5/2022 12:00:00 AM
4/1/2022	03/19/2022	3/19/2022 12:00:00 AM
5/7/2022	04/30/2022	4/30/2022 12:00:00 AM
5/16/2022	05/07/2022	5/7/2022 12:00:00 AM
6/15/2022	06/04/2022	6/4/2022 12:00:00 AM
6/26/2022	06/18/2022	6/18/2022 12:00:00 AM
7/9/2022	07/02/2022	7/2/2022 12:00:00 AM
7/22/2022	07/09/2022	7/9/2022 12:00:00 AM
7/23/2022	07/16/2022	7/16/2022 12:00:00 AM
7/27/2022	07/16/2022	7/16/2022 12:00:00 AM
8/2/2022	07/23/2022	7/23/2022 12:00:00 AM
8/8/2022	07/30/2022	7/30/2022 12:00:00 AM
8/19/2022	08/06/2022	8/6/2022 12:00:00 AM
9/26/2022	09/17/2022	9/17/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/22/2022	10/22/2022 12:00:00 AM

この例では、-1 の period_no が lunarweekstart() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した旧暦の週を識別します。次に、1 週間前にずらして、旧暦のその週の最初のミリ秒を識別します。

lunarweekstart() 関数の図、period_no の例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekstart() 関数は、旧暦の週が 1 月 15 日に開始することを特定します。そのため、旧暦の前の週は 1 月 8 日 12:00:00 AM に開始されます。これは、[previous_lunar_week_start] 項目に対して返される値です。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。この例では、旧暦の週が 1 月 5 日に始まるよう設定しています。

ロードスクリプト

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

Transactions:
    Load
        *,
        lunarweekstart(date,0,4) as start_of_week,
        timestamp(lunarweekstart(date,0,4)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- start_of_week
- start_of_week_timestamp

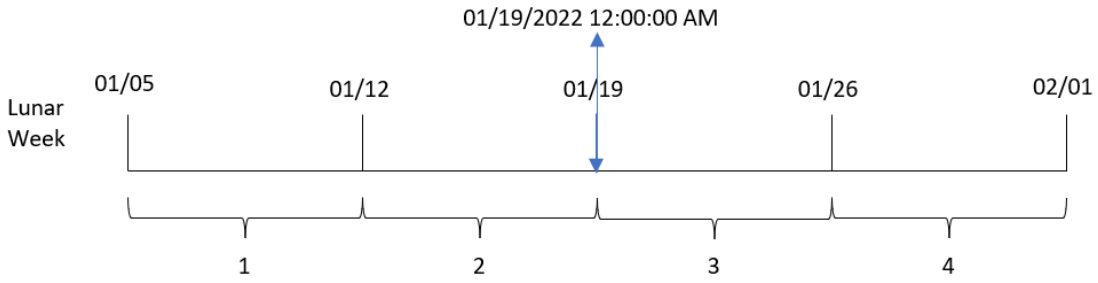
結果テーブル

日付	start_of_week	start_of_week_timestamp
1/7/2022	01/05/2022	1/5/2022 12:00:00 AM
1/19/2022	01/19/2022	1/19/2022 12:00:00 AM
2/5/2022	02/02/2022	2/2/2022 12:00:00 AM
2/28/2022	02/23/2022	2/23/2022 12:00:00 AM
3/16/2022	03/16/2022	3/16/2022 12:00:00 AM
4/1/2022	03/30/2022	3/30/2022 12:00:00 AM
5/7/2022	05/04/2022	5/4/2022 12:00:00 AM
5/16/2022	05/11/2022	5/11/2022 12:00:00 AM
6/15/2022	06/15/2022	6/15/2022 12:00:00 AM
6/26/2022	06/22/2022	6/22/2022 12:00:00 AM
7/9/2022	07/06/2022	7/6/2022 12:00:00 AM
7/22/2022	07/20/2022	7/20/2022 12:00:00 AM
7/23/2022	07/20/2022	7/20/2022 12:00:00 AM
7/27/2022	07/27/2022	7/27/2022 12:00:00 AM
8/2/2022	07/27/2022	7/27/2022 12:00:00 AM
8/8/2022	08/03/2022	8/3/2022 12:00:00 AM
8/19/2022	08/17/2022	8/17/2022 12:00:00 AM

日付	start_of_week	start_of_week_timestamp
9/26/2022	09/21/2022	9/21/2022 12:00:00 AM
10/14/2022	10/12/2022	10/12/2022 12:00:00 AM
10/29/2022	10/26/2022	10/26/2022 12:00:00 AM

このインスタンスでは、first_week_date 引数である 4 が lunarweekstart() 関数で使用されるため、1 月 1 日から 1 月 5 日に年の初めがオフセットされます。

lunarweekstart() 関数、first_week_day 例の図



トランザクション 8189 は 1 月 19 日に発生しました。旧暦の週が 1 月 5 日に始まる前、lunarweekstart() 関数は 1 月 19 日 12:00:00 AM を含む旧暦の週も 1 月 19 日に始まることを特定します。そのため、これは [start_of_week] 項目に返された値です。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した旧暦の週の始めのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
```

```
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。

次のメジャーを追加します。

```
=lunarweekstart(date)
```

```
=timestamp(lunarweekstart(date))
```

結果テーブル

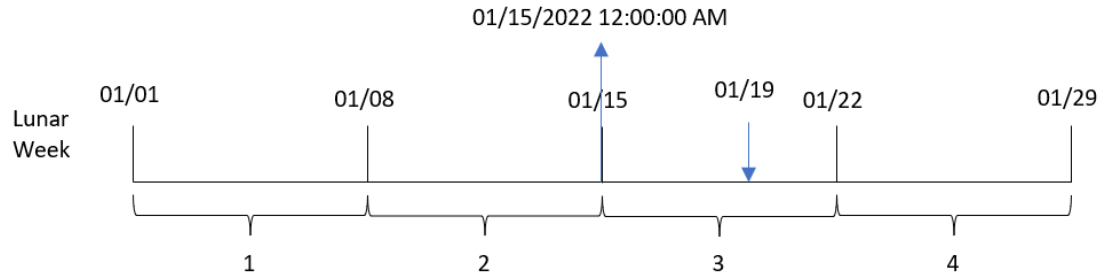
日付	=lunarweekstart(date)	=timestamp(lunarweekstart(date))
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/15/2022	1/15/2022 12:00:00 AM
2/5/2022	02/05/2022	2/5/2022 12:00:00 AM
2/28/2022	02/26/2022	2/26/2022 12:00:00 AM
3/16/2022	03/12/2022	3/12/2022 12:00:00 AM
4/1/2022	03/26/2022	3/26/2022 12:00:00 AM
5/7/2022	05/07/2022	5/7/2022 12:00:00 AM
5/16/2022	05/14/2022	5/14/2022 12:00:00 AM
6/15/2022	06/11/2022	6/11/2022 12:00:00 AM
6/26/2022	06/25/2022	6/25/2022 12:00:00 AM
7/9/2022	07/09/2022	7/9/2022 12:00:00 AM
7/22/2022	07/16/2022	7/16/2022 12:00:00 AM

日付	=lunarweekstart(date)	=timestamp(lunarweekstart(date))
7/23/2022	07/23/2022	7/23/2022 12:00:00 AM
7/27/2022	07/23/2022	7/23/2022 12:00:00 AM
8/2/2022	07/30/2022	7/30/2022 12:00:00 AM
8/8/2022	08/06/2022	8/6/2022 12:00:00 AM
8/19/2022	08/13/2022	8/13/2022 12:00:00 AM
9/26/2022	09/24/2022	9/24/2022 12:00:00 AM
10/14/2022	10/08/2022	10/8/2022 12:00:00 AM
10/29/2022	10/29/2022	10/29/2022 12:00:00 AM

start_of_week メジャーは、lunarweekstart() 関数を使用し、関数の引数として日付項目を渡すことにより、チャート オブジェクトで作成されます。

lunarweekstart() 関数は、日付値がどの旧暦週に該当するかを識別し、その週の最後のミリ秒のタイムスタンプを返します。

lunarweekstart() 関数の図、チャート オブジェクトの例



トランザクション 8189 は 1 月 19 日に発生しました。lunarweekstart() 関数は、旧暦の週が 1 月 15 日に開始することを特定します。そのため、そのトランザクションの start_of_week 値は、その日の最初のミリ秒である 1 月 15 日 12:00:00 AM です。

例 5 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Loans というテーブルにロードされる、一連のローン残高を含むデータセット。
- ローン ID、週の初めの残高、各ローンにかかる単利の年率で構成されるデータ。

エンドユーザーは、日付までの1週間の各ローンで発生した現在の利息をローンID別に表示するチャートオブジェクトを求めています。

ロードスクリプト

```
Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の項目を軸として追加します。
   - loan_id
   - start_balance
3. 次に、次のメジャーを作成して、累積利息を計算します。
   `=start_balance*(rate*(today(1)-lunarweekstart(today(1)))/365)`
4. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

loan_id	start_balance	=start_balance*(rate*(today(1)-lunarweekstart (today(1)))/365)
8188	$10000.00	$15.07
8189	$15000.00	$128.84
8190	$17500.00	$63.29
8191	$21000.00	$107.59
8192	$90000.00	$1139.18

lunarweekstart() 関数は、今日の日付を唯一の引数として使用することにより、現在の年の開始日を返します。その結果を現在の日付から減算することにより、数式は今週経過した日数を返します。

次に、この値に利率を乗算して365で除算すると、この期間に発生する実効利率が返されます。次に、結果にローンの開始残高を掛けると、今週これまでに発生した利息を返されます。

makedate

この関数は、年 **YYYY**、月 **MM**、日 **DD** から算出された日付を返します。

構文:

```
MakeDate(YYYY [ , MM [ , DD ] ])
```

戻り値データ型: dual

引数

引数	説明
YYYY	年 (整数)。
MM	月 (整数)。月が指定されていない場合は、1 (1 月) と見なされます。
DD	日 (整数)。日が指定されていない場合は、1 (1 日) と見なされます。

使用に適しているケース

makedate() 関数は、カレンダーを生成するために、データ生成用のスクリプトで一般的に使用されます。これは、日付項目を直接日付として利用できないが、年、月、日コンポーネントを抽出するために何らかの変換が必要な場合にも使用できます。

これらの例は、日付書式 DD/MM/YYYY を使用しています。日付書式は、データロードスクリプト上部の SET DateFormat ステートメントで指定されています。必要に応じて、例の書式を変更してください。

関数の例

例	結果
makedate(2012)	01/01/2012 を返します。
makedate(12)	01/01/2012 を返します。
makedate(2012,12)	12/01/2012 を返します。
makedate(2012,2,14)	02/14/2012 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2018 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- MM/DD/YYYY 形式で日付を返す項目 [transaction_date] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        makedate(transaction_year, transaction_month, transaction_day) as transaction_date
        ;
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_day, transaction_amount,
transaction_quantity, customer_id
3750, 2018, 08, 30, 12423.56, 23, 2038593
3751, 2018, 09, 07, 5356.31, 6, 203521
3752, 2018, 09, 16, 15.75, 1, 5646471
3753, 2018, 09, 22, 1251, 7, 3036491
3754, 2018, 09, 22, 21484.21, 1356, 049681
3756, 2018, 09, 22, -59.18, 2, 2038593
3757, 2018, 09, 23, 3177.4, 21, 203521
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- transaction_year
- transaction_month
- transaction_day
- transaction_date

結果テーブル

transaction_year	transaction_month	transaction_day	transaction_date
2018	08	30	08/30/2018

transaction_year	transaction_month	transaction_day	transaction_date
2018	09	07	09/07/2018
2018	09	16	09/16/2018
2018	09	22	09/22/2018
2018	09	23	09/23/2018

transaction_date 項目は、makedate() 関数を使用して、関数の引数として年、月、日の項目を渡すことにより、先行するLOAD ステートメントで作成されます。

次に関数はこれらの値を組み合わせて日付項目に変換し、DateFormat システム変数の形式で結果を返します。

例 2 – 変更された DateFormat

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- DateFormat システム変数を変更しない、形式 DD/MM/YYYY での項目 transaction_date の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        date(makedate(transaction_year, transaction_month, transaction_day), 'DD/MM/YYYY') as
transaction_date
        ;
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_day, transaction_amount,
transaction_quantity, customer_id
3750, 2018, 08, 30, 12423.56, 23, 2038593
3751, 2018, 09, 07, 5356.31, 6, 203521
3752, 2018, 09, 16, 15.75, 1, 5646471
3753, 2018, 09, 22, 1251, 7, 3036491
3754, 2018, 09, 22, 21484.21, 1356, 049681
3756, 2018, 09, 22, -59.18, 2, 2038593
3757, 2018, 09, 23, 3177.4, 21, 203521
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- transaction_year
- transaction_month
- transaction_day
- transaction_date

結果テーブル

transaction_year	transaction_month	transaction_day	transaction_date
2018	08	30	30/08/2018
2018	09	07	07/09/2018
2018	09	16	16/09/2018
2018	09	22	22/09/2018
2018	09	23	23/09/2018

この場合、makedate() 関数は date() 関数内にネストされています。date() 関数の第 2 引数は、makedate() 関数の結果を必要な DD/MM/YYYY として設定します。

例 3 – チャートオブジェクトの例

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2018 年の一連のトランザクションを含むデータセット。
- 2 つの項目 year とmonth に提供されたトランザクション日付。

MM/DD/YYYY 形式で日付を返す、チャートオブジェクトメジャー transaction_date。

ロードスクリプト

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

Transactions:
Load * Inline [
transaction_id, transaction_year, transaction_month, transaction_amount, transaction_quantity,
customer_id
3750, 2018, 08, 12423.56, 23, 2038593
3751, 2018, 09, 5356.31, 6, 203521
3752, 2018, 09, 15.75, 1, 5646471
```

```
3753, 2018, 09, 1251, 7, 3036491
3754, 2018, 09, 21484.21, 1356, 049681
3756, 2018, 09, -59.18, 2, 2038593
3757, 2018, 09, 3177.4, 21, 203521
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- year
- month

transaction_date を決定するには、このメジャーを作成します:

```
=makedate(transaction_year,transaction_month)
```

結果テーブル

transaction_year	transaction_month	transaction_date
2018	08	08/01/2018
2018	09	09/01/2018

transaction_date メジャーは、makedate() 関数を使用し、関数の引数として年と月項目を渡すことにより、チャートオブジェクトで作成されます。

次に関数はこれらの値と、推定される日の値である 01 を組み合わせます。次にこれらの値が組み合わされて日付項目に変換され、DateFormat システム変数の形式で結果を返します。

例 4 – シナリオ

ロードスクリプトとチャートの数式

概要

暦年 2020 のカレンダーデータセットを作成します。

ロードスクリプト

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

Calendar:
       load
               *
    where year(date)=2022;
load
       date(recno()+makedate(2021,12,31)) as date
AutoGenerate 400;
```

結果

結果テーブル

日付
01/01/2022
01/02/2022
01/03/2022
01/04/2022
01/05/2022
01/06/2022
01/07/2022
01/08/2022
01/09/2022
01/10/2022
01/11/2022
01/12/2022
01/13/2022
01/14/2022
01/15/2022
01/16/2022
01/17/2022
01/18/2022
01/19/2022
01/20/2022
01/21/2022
01/22/2022
01/23/2022
01/24/2022
01/25/2022
さらに 340 行以上

makedate() 関数は、2021 年 12 月 31 日の日付の値を作成します。recno() 関数は、テーブルにロードされる現行レコードのレコード番号を 1 から順に割り当てます。そのため、最初のレコードの日付は 2022 年 1 月 1 日となります。その後の各 recno() は、この日付を 1 日ずつ増やします。この数式は、値を日付に変換する date

() 関数で囲まれています。この処理は autogenerate 関数により400 回繰り返されます。最後に、先行する LOAD を使うことにより、where 条件を使って 2022 年からのみ日付をロードすることができます。このスクリプトは、2022 年のすべての日付を含むカレンダーを生成します。

maketime

この関数は、時間 **hh**、分 **mm**、秒 **ss** から算出された時間を返します。

構文:

```
MakeTime(hh [ , mm [ , ss ] ])
```

戻り値データ型: dual

引数

引数	説明
hh	時間 (整数)。
mm	分 (整数)。 分が指定されていない場合は、00 として処理されます。
ss	秒 (整数)。 秒が指定されていない場合は、00 として処理されます。

使用に適しているケース

maketime() 関数は、時刻項目を生成するために、データ生成用のスクリプトで一般的に使用されます。時刻項目が入力テキストから派生する場合、そのコンポーネントを用いて時刻を構築するためにこの関数を使用できます。

これらの例は、時刻形式 h:mm:ss を使用しています。時刻書式は、データロードスクリプト上部の SET TimeFormat ステートメントで指定されています。必要に応じて、例の書式を変更してください。

関数の例

例	結果
maketime(22)	22:00:00 を返します。
maketime(22, 17)	22:17:00 を返します。
maketime(22,17,52)	22:17:52 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – maketime()

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、一連のトランザクションを含むデータセット。
- 3 つの項目 hours、minutes および seconds に提供されたトランザクション時刻。
- TimeFormat システム変数の形式で時刻を返す、項目 transaction_time の作成。

ロードスクリプト

```
SET TimeFormat='h:mm:ss TT';

Transactions:
    Load
        *,
        maketime(transaction_hour, transaction_minute, transaction_second) as transaction_time
        ;
Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_second, transaction_amount,
transaction_quantity, customer_id
3750, 18, 43, 30, 12423.56, 23, 2038593
3751, 6, 32, 07, 5356.31, 6, 203521
3752, 12, 09, 16, 15.75, 1, 5646471
3753, 21, 43, 41, 7, 3036491
3754, 17, 55, 22, 21484.21, 1356, 049681
3756, 2, 52, 22, -59.18, 2, 2038593
3757, 9, 25, 23, 3177.4, 21, 203521
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- transaction_hour
- transaction_minute
- transaction_second
- transaction_time

結果テーブル

transaction_hour	transaction_minute	transaction_second	transaction_time
2	52	22	2:52:22 AM
6	32	07	6:32:07 AM
9	25	23	9:25:23 AM
12	09	16	12:09:16 PM
17	55	22	5:55:22 PM
18	43	30	6:43:30 PM
21	43	41	9:43:41 PM

transaction_time 項目は、maketime() 関数を使用して、関数の引数として時間、分、秒の項目を渡すことにより、先行するLOAD ステートメントで作成されます。

次に関数はこれらの値を組み合わせて時刻項目に変換し、TimeFormat システム変数の時刻形式で結果を返します。

例 2 – time() 関数

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 項目 transaction_time の作成。これにより、TimeFormat システム変数を変更せずに、24 時間の時刻形式で結果を表示できます。

ロードスクリプト

```
SET TimeFormat='h:mm:ss TT';

Transactions:
    Load
        *,
        time(maketime(transaction_hour, transaction_minute, transaction_second),'h:mm:ss') as
transaction_time
        ;
Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_second, transaction_amount,
transaction_quantity, customer_id
3750, 18, 43, 30, 12423.56, 23, 2038593
3751, 6, 32, 07, 5356.31, 6, 203521
3752, 12, 09, 16, 15.75, 1, 5646471
3753, 21, 43, 41, 7, 3036491
```

```
3754, 17, 55, 22, 21484.21, 1356, 049681
3756, 2, 52, 22, -59.18, 2, 2038593
3757, 9, 25, 23, 3177.4, 21, 203521
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- transaction_hour
- transaction_minute
- transaction_second
- transaction_time

結果テーブル

transaction_hour	transaction_minute	transaction_second	transaction_time
2	52	22	2:52:22
6	32	07	6:32:07
9	25	23	9:25:23
12	09	16	12:09:16
17	55	22	17:55:22
18	43	30	18:43:30
21	43	41	21:43:41

この場合、maketime() 関数は time() 関数内にネストされています。time() 関数の第 2 引数は、maketime() 関数の結果を必要な h:mm:ss として設定します。

例 3 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、一連のトランザクションを含むデータセット。
- 2 つの項目 hours と minutes に提供されたトランザクション時刻。
- TimeFormat システム変数の形式で時刻を返す、項目 transaction_time の作成。

h:mm:ss TT 形式で時刻を返す、チャート オブジェクト メジャー transaction_time。

ロード スクリプト

```
SET TimeFormat='h:mm:ss TT';

Transactions:
Load * Inline [
transaction_id, transaction_hour, transaction_minute, transaction_amount, transaction_
quantity, customer_id
3750, 18, 43, 12423.56, 23, 2038593
3751, 6, 32, 5356.31, 6, 203521
3752, 12, 09, 15.75, 1, 5646471
3753, 21, 43, 7, 3036491
3754, 17, 55, 21484.21, 1356, 049681
3756, 2, 52, -59.18, 2, 2038593
3757, 9, 25, 3177.4, 21, 203521
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- transaction_hour
- transaction_minute

transaction_time を計算するには、次のメジャーを作成します:

```
=maketime(transaction_hour,transaction_minute)
```

結果テーブル

transaction_hour	transaction_minute	=maketime(transaction_hour, transaction_minute)
2	52	2:52:00 AM
6	32	6:32:00 AM
9	25	9:25:00 AM
12	09	12:09:00 PM
17	55	5:55:00 PM
18	43	6:43:00 PM
21	43	9:43:00 PM

transaction_time メジャーは、maketime() 関数を使用し、関数の引数として時間と分項目を渡すことにより、チャート オブジェクトで作成されます。

関数は次にこれらの値を組み合わせ、秒は 00 と推測されます。次にこれらの値が組み合わされて時刻項目に変換され、TimeFormat システム変数の形式で結果を返します。

例 4 – シナリオ

ロードスクリプトとチャートの数式

概要

8時間増分に分割された、2022年1月のカレンダーデータセットを作成します。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

tmpCalendar:
       load
               *
    where year(date)=2022;
load
       date(recno()+makedate(2021,12,31)) as date
AutoGenerate 31;

Left join(tmpCalendar)
load
       maketime((recno()-1)*8,00,00) as time
autogenerate 3;

Calendar:
load
       timestamp(date + time) as timestamp
resident tmpCalendar;

drop table tmpCalendar;
```

結果

結果テーブル

日付と時刻
1/1/2022 12:00:00 AM
1/1/2022 8:00:00 AM
1/1/2022 4:00:00 PM
1/2/2022 12:00:00 AM
1/2/2022 8:00:00 AM
1/2/2022 4:00:00 PM
1/3/2022 12:00:00 AM
1/3/2022 8:00:00 AM

日付と時刻
1/3/2022 4:00:00 PM
1/4/2022 12:00:00 AM
1/4/2022 8:00:00 AM
1/4/2022 4:00:00 PM
1/5/2022 12:00:00 AM
1/5/2022 8:00:00 AM
1/5/2022 4:00:00 PM
1/6/2022 12:00:00 AM
1/6/2022 8:00:00 AM
1/6/2022 4:00:00 PM
1/7/2022 12:00:00 AM
1/7/2022 8:00:00 AM
1/7/2022 4:00:00 PM
1/8/2022 12:00:00 AM
1/8/2022 8:00:00 AM
1/8/2022 4:00:00 PM
1/9/2022 12:00:00 AM
さらに 68 行以上

開始時の autogenerate 関数は、tmpCalendar と呼ばれるテーブルに 1 月の日付をすべて含むカレンダーを作成します。

3 つのレコードを含む 2 番目のテーブルが作成されます。各レコードについて、recno() - 1 (値 0、1、2) が取得され、結果が 8 で乗算されます。その結果、これは値 0、8、16 を生成します。これらの値は maketime() 関数で分と秒の値を 0 とする時刻パラメータとして使用されます。その結果、テーブルには 3 つの時刻項目 (12:00:00 AM、8:00:00 AM、および 4:00:00 PM) が含まれます。

このテーブルは、tmpCalendar テーブルと結合されます。2 つのテーブル間で結合するために一致する項目がないため、時刻行は各日付行に追加されます。その結果、各日付行が各時刻値ごとに 3 回繰り返されるようになります。

最後に、カレンダー テーブルが tmpCalendar テーブルの resident load から作成されます。日付と時刻項目は、timestamp() 関数で連結されて囲まれ、タイムスタンプ項目が作成されます。

次に、tmpCalendar テーブルが削除されます。

makeweekdate

この関数は、年 、週番号、曜日 から算出された日付を返します。

構文:

```
MakeWeekDate(weekyear [, week [, weekday [, first_week_day [, broken_weeks [, reference_day]]]]])
```

戻り値データ型: dual

makeweekdate() 関数はスクリプトとチャート関数の両方として使用できます。関数は、関数に渡されたパラメータに基づいて日付を計算します。

引数

引数	説明
weekyear	特定の日付に対してWeekYear() 関数によって定義された年、つまり週番号が属する年です。 *前年の12月に第1週がすでに開始されている場合など、週の年が暦年と異なる場合があります。*
week	特定の日付に対してWeek() 関数によって定義された週番号です。 週番号が指定されていない場合は、1として処理されます。
weekday	質問の日付に対してWeekDay() 関数によって定義された曜日です。0は週の最初の日、6は週の最後の日になります。 曜日が指定されていない場合は、0として処理されます。 *0は常に週の最初の日を意味し、6は常に最後の日を意味しますが、対応する曜日は **first_week_day** パラメーターによって決定されます。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。* 分離した週が使用され、パラメーターを組み合わせることができない場合、選択した年に属さない結果になる可能性があります。 MakeWeekDate(2021,1,0,6,1) この日は指定された週の最初の日 (日曜日) であるため、「2020年12月27日」を返します。2021年1月1日は金曜日でした。
first_week_day	週の開始日を指定します。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。 **first_week_day** には、0が月曜日、1が火曜日、2が水曜日、3が木曜日、4が金曜日、5が土曜日、6が日曜日の値を使用できます。 システム変数の詳細については、*FirstWeekDay (page 228)* を参照してください。

引数	説明
broken_weeks	**broken_weeks** が指定されていない場合は、変数 **BrokenWeeks** の値を使用して、週が分離しているかどうかを定義します。
reference_day	**reference_day** が指定されていない場合は、変数 **ReferenceDay** の値を使用して、第1週を定義する参照日として設定する1月の日を定義します。

使用に適しているケース

makeweekdate() 関数は、日付のリストを生成したり、入力データに年、週、曜日がある場合に日付を作成したりするために、データ生成用のスクリプトで一般的に使用されます。

次の例は下記の内容を推測します。

```
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;
```

関数の例

例	結果
makeweekdate(2014,6,6)	の戻り値: 02/09/2014
makeweekdate(2014,6,1)	の戻り値: 02/04/2014
makeweekdate(2014,6)	02/03/2014 を返します (曜日が 0 として処理されます)

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 日を含む

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Sales というテーブルの 2022 年の週次売上合計を含むデータセット。
- 3 つの項目 year、week および sales に提供されたトランザクション日付。
- makeweekdate() 関数を使って MM/DD/YYYY 形式でその週の金曜日の日付を返す、メジャー end_of_week を作成するのに使用される先行する LOAD。

返された日付が金曜日であることを証明するには、end_of_week 数式も曜日を示すため weekday() 関数で囲みます。

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Transactions:
    Load
        *,
        makeweekdate(transaction_year, transaction_week,4) as end_of_week,
        weekday(makeweekdate(transaction_year, transaction_week,4)) as week_day
        ;
Load * Inline [
transaction_year, transaction_week, sales
2022, 01, 10000
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- transaction_year
- transaction_week
- end_of_week
- week_day

結果テーブル

transaction_year	transaction_week	end_of_week	week_day
2022	01	01/07/2022	Fri
2022	02	01/14/2022	Fri
2022	03	01/21/2022	Fri
2022	04	01/28/2022	Fri

transaction_year	transaction_week	end_of_week	week_day
2022	05	02/04/2022	Fri
2022	06	02/11/2022	Fri
2022	07	02/18/2022	Fri

end_of_week 項目は、makeweekdate() 関数を使用して、先行する LOAD ステートメントで作成されます。transaction_year、transaction_week 項目は、年および週の引数として関数を使って渡されます。値 4 は、日引数に使用されます。

次に関数はこれらの値を組み合わせて日付項目に変換し、DateFormat システム変数の形式で結果を返します。

makeweekdate() 関数とその引数もまた weekday() 関数で囲まれ、week_day 項目を返します。さらに、上記のテーブルでは、week_day 項目ではこれらの日付が金曜日に当たることがわかります。

例 2 – 日を除外

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Sales というテーブルの 2022 年の週次売上合計を含むデータセット。
- 3 つの項目 year、week および sales に提供されたトランザクション日付。
- makeweekdate() 関数を使って、メジャー first_day_of_week を作成するのに使用される先行 load。これにより、MM/DD/YYYY 形式でその週の月曜日の日付が返されます。

返された日付が月曜日であることを証明するには、first_day_of_week 数式も曜日を示すため weekday() 関数で囲みます。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Transactions:
    Load
        *,
        makeweekdate(transaction_year, transaction_week) as first_day_of_week,
        weekday(makeweekdate(transaction_year, transaction_week)) as week_day
        ;
Load * Inline [
transaction_year, transaction_week, sales
2022, 01, 10000
```

```
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- transaction_year
- transaction_week
- first_day_of_week
- week_day

結果テーブル

transaction_year	transaction_week	first_day_of_week	week_day
2022	01	01/03/2022	月
2022	02	01/10/2022	月
2022	03	01/17/2022	月
2022	04	01/24/2022	月
2022	05	01/31/2022	月
2022	06	02/07/2022	月
2022	07	02/14/2022	月

[first_day_of_week] 項目は、makeweekdate() 関数を使用して、前の Load ステートメントで作成されます。transaction_year とtransaction_week パラメータは関数の引数として渡され、日パラメータはブランクのままとなります。

次に関数はこれらの値を組み合わせて日付項目に変換し、DateFormat システム変数の形式で結果を返します。

makeweekdate() 関数とその引数もまたweekday() 関数で囲まれ、week_day 項目を返します。上の表からわかるように、week_day フィールドはすべてのケースで月曜日を返します。これは、makeweekdate() 関数ではそのパラメーターが空白のままであり、既定は 0 (週の最初の曜日) で、週の最初の曜日が FirstWeekDay システム変数によって月曜日に設定されているためです。

例 3 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Sales というテーブルの 2022 年の週次売上合計を含むデータセット。
- 3 つの項目 year、week および sales に提供されたトランザクション日付。

この例では、最初の例から end_of_week 計算に相当するメジャーを作成するのに、チャート オブジェクトが使用されます。このメジャーは makeweekdate() 関数を使って、MM/DD/YYYY 形式でその週の金曜日の日付が返されます。

返された日付が金曜日であることを証明するため、曜日を返す 2 番目のメジャーが作成されます。

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Master_Calendar:
Load * Inline [
transaction_year, transaction_week, sales
2022, 01, 10000
2022, 02, 11250
2022, 03, 9830
2022, 04, 14010
2022, 05, 28402
2022, 06, 9992
2022, 07, 7292
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:
   - transaction_year
   - transaction_week
2. 最初の例の end_of_week 項目に相当する計算を実行するには、次のメジャーを作成します:
   `=makeweekdate(transaction_year,transaction_week,4)`
3. 各トランザクションの週の曜日を計算するには、次のメジャーを作成します:
   `=weekday(makeweekdate(transaction_year,transaction_week,4))`

結果テーブル

transaction_year	transaction_week	=makeweekdate(transaction_year,transaction_week,4)	=weekday(makeweekdate(transaction_year,transaction_week,4))
2022	01	01/07/2022	Fri
2022	02	01/14/2022	Fri
2022	03	01/21/2022	Fri
2022	04	01/28/2022	Fri
2022	05	02/04/2022	Fri
2022	06	02/11/2022	Fri
2022	07	02/18/2022	Fri

end_of_week に相当する項目が、makeweekdate() 関数を使用することにより、メジャーとしてチャート オブジェクトに作成されます。transaction_year および transaction_week 項目は、年および週の引数として渡されます。値 4 は、日引数に使用されます。

次に関数はこれらの値を組み合わせて日付項目に変換し、DateFormat システム変数の形式で結果を返します。

makeweekdate() 関数とその引数もまた weekday() 関数で囲まれ、最初の例からの week_day 項目の計算に相当する計算を返します。上記のテーブルで分かるように、右側の最後の列はこれらの日付が金曜日に当たることを示しています。

例 4 – シナリオ

ロード スクリプトとチャートの数式

概要

この例では、2022 年のすべての金曜日を含む日付のリストを作成します。

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;

Calendar:
    load
           *,
        weekday(date) as weekday
    where year(date)=2022;
load
```

```
    makeweekdate(2022,recno()-2,4) as date
AutoGenerate 60;
```

結果

結果テーブル

日付	weekday
01/07/2022	Fri
01/14/2022	Fri
01/21/2022	Fri
01/28/2022	Fri
02/04/2022	Fri
02/11/2022	Fri
02/18/2022	Fri
02/25/2022	Fri
03/04/2022	Fri
03/11/2022	Fri
03/18/2022	Fri
03/25/2022	Fri
04/01/2022	Fri
04/08/2022	Fri
04/15/2022	Fri
04/22/2022	Fri
04/29/2022	Fri
05/06/2022	Fri
05/13/2022	Fri
05/20/2022	Fri
05/27/2022	Fri
06/03/2022	Fri
06/10/2022	Fri
06/17/2022	Fri
さらに 27 行以上	

makeweekdate() 関数は、2022 年のすべての金曜日を見つけます。週パラメータ-2 を使うことにより、日付を見過ごすことはなくなります。最後に、先行 load が追加の weekday 項目を作成し、各 date の値が金曜日であることを明確に示します。

minute

この関数は、**expression** の小数部が標準的な数値の解釈に従って時間と判断される場合に、分を表す整数を返します。

構文:

```
minute(expression)
```

戻り値データ型: 整数

使用に適しているケース

minute() 関数は、集計を分単位で比較する場合に便利です。例えば、分ごとのアクティビティ数分布を確認したい場合は、関数を使用できます。

これらの軸は、関数を使用してマスター カレンダー テーブルに項目を作成することにより、ロード スクリプトで作成できます。あるいは、計算軸としてチャートで直接使用することもできます。

関数の例

例	結果
minute ('09:14:36')	14 を返します。
minute ('0.5555')	19 を返します (0.5555 = 13:19:55 のため)

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データ ロード スクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロード スクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データ ロード エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 変数 (スクリプト)

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされるタイムスタンプによるトランザクションを含むデータセット。
- 既定の TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT) が使用されます。
- トランザクションがいつ発生するかを計算する、項目 minute の作成。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        minute(timestamp) as minute
        ;
Load
*
Inline
[
id,timestamp,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- timestamp
- minute

結果テーブル

日付と時刻	分
2022-01-01 22:10:22	10
2022-01-02 08:35:54	35
2022-01-03 05:40:49	40
2022-01-03 14:21:53	21
2022-01-04 18:49:38	49
2022-01-04 22:58:34	58
2022-01-05 19:04:57	4

日付と時刻	分
2022-01-05 19:34:46	34
2022-01-06 08:49:09	49
2022-01-06 11:29:38	29

minute 項目の値は、minute() 関数を使用し、先行するLOAD ステートメントの数式として timestamp を渡すことによって作成されます。

例 2 – チャート オブジェクト (チャート)

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 既定の TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT) が使用されます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。「minute」値は、チャート オブジェクトのメジャーを介して計算されます。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
*
Inline
[
id,timestamp,amount
9497,'2022-01-05 19:04:57',47.25,
9498,'2022-01-03 14:21:53',51.75,
9499,'2022-01-03 05:40:49',73.53,
9500,'2022-01-04 18:49:38',15.35,
9501,'2022-01-01 22:10:22',31.43,
9502,'2022-01-05 19:34:46',13.24,
9503,'2022-01-04 22:58:34',74.34,
9504,'2022-01-06 11:29:38',50.00,
9505,'2022-01-02 08:35:54',36.34,
9506,'2022-01-06 08:49:09',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: timestamp。

次のメジャーを作成します:

```
=minute(timestamp)
```

結果テーブル

日付と時刻	分
2022-01-01 22:10:22	10
2022-01-02 08:35:54	35
2022-01-03 05:40:49	40
2022-01-03 14:21:53	21
2022-01-04 18:49:38	49
2022-01-04 22:58:34	58
2022-01-05 19:04:57	4
2022-01-05 19:34:46	34
2022-01-06 08:49:09	49
2022-01-06 11:29:38	29

minute の値は、minute() 関数を使用し、チャート オブジェクトのメジャーの数式として timestamp を渡すことによって作成されます。

例 3 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- チケット バリアーのエントリーを表すために生成される、タイムスタンプのデータセット。
- Ticket_Barrier_Tracker というテーブルにロードされる、各 timestamp と対応する id を持つ情報。
- 既定の TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT) が使用されます。

ユーザーは、バリアー エントリー数を分ごとに示すチャート オブジェクトを求めています。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

tmpTimeStampCreator:
       load
                *
    where year(date)=2022;
load
```

```
       date(recno()+makedate(2021,12,31)) as date
AutoGenerate 1;

join load
       maketime(floor(rand()*24),floor(rand()*59),floor(rand()*59)) as time
autogenerate 10000;

Ticket_Barrier_Tracker:
load
       recno() as id,
       timestamp(date + time) as timestamp
resident tmpTimeStampCreator;

drop table tmpTimeStampCreator;
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の式を使用して計算軸を作成します。
   `=minute(timestamp)`
3. 次の集計メジャーを追加して、エントリーの合計数を計算します。
   `=count(id)`
4. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

minute(timestamp)	=count(id)
0	174
1	171
2	175
3	165
4	188
5	176
6	158
7	187
8	178
9	178
10	197
11	161
12	166

minute(timestamp)	=count(id)
13	184
14	159
15	161
16	152
17	160
18	176
19	164
20	170
21	170
22	142
23	145
24	155
さらに 35 行以上	

month

この関数は、環境変数 **MonthNames** および 1 から 12 までの整数で定義されている月名を持つデュアル値を返します。月は標準的な数値の解釈に従って、数式の日付の解釈により計算されます。

この関数は、特定の日付の月の名前を MonthName システム変数の形式で返します。これは通常、マスタカレンダーの軸として日付項目を作成するために使用します。

構文:

```
month(expression)
```

戻り値データ型: 整数

関数の例

例	結果
month(2012-10-12)	Oct を返します
month(35648)	Aug を返します (35648 = 1997-08-06 のため)

例 1 – DateFormat データセット (スクリプト)

ロード スクリプトと結果

概要

データ ロード エディター を開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています:

- Master_Calendar という名前の日付のデータセット。DateFormat システム変数は、DD/MM/YYYY に設定されています。
- month() 関数を使用して、month_name という名前の追加項目を作成する、先行する LOAD。
- date() 関数を使用して完全な日付を表示する、long_date という名前の追加項目。

ロード スクリプト

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

Master_Calendar:
Load
        date,
        date(date,'dd-MMMM-YYYY') as long_date,
        month(date) as month_name
Inline
[
date
03/01/2022
03/02/2022
03/03/2022
03/04/2022
03/05/2022
03/06/2022
03/07/2022
03/08/2022
03/09/2022
03/10/2022
03/11/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- long_date
- month_name

結果テーブル

日付	long_date	month_name
03/01/2022	03-January- 2022	Jan
03/02/2022	03-February- 2022	Feb
03/03/2022	03-March- 2022	Mar
03/04/2022	03-April- 2022	Apr
03/05/2022	03-May- 2022	May
03/06/2022	03-June- 2022	Jun
03/07/2022	03-July- 2022	Jul
03/08/2022	03-August- 2022	Aug
03/09/2022	03-September- 2022	Sep
03/10/2022	03-October- 2022	Oct
03/11/2022	03-November- 2022	Nov

該当月の名前は、スクリプトの month() 関数により正常に評価されています。

例 2 – ANSI 日付 (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- Master_Calendar という名前の日付のデータセット。DateFormat システム変数 DD/MM/YYYY が使用されます。ただし、データセットに含まれる日付は、ANSI 標準日付形式です。
- month() 関数を使用して month_name という名前の追加項目を作成する先行するロード。
- date() 関数を使用して完全な日付を表示する、long_date という名前の追加項目。

ロードスクリプト

```
SET DateFormat='DD/MM/YYYY';
Master_Calendar:
Load
       date,
       date(date,'dd-MMMM-YYYY') as long_date,
       month(date) as month_name

Inline
[
date
2022-01-11
```

```
2022-02-12
2022-03-13
2022-04-14
2022-05-15
2022-06-16
2022-07-17
2022-08-18
2022-09-19
2022-10-20
2022-11-21
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- long_date
- month_name

結果テーブル

日付	long_date	month_name
03/11/2022	11-March- 2022	11
03/12/2022	12-March- 2022	12
03/13/2022	13-March- 2022	13
03/14/2022	14-March- 2022	14
03/15/2022	15-March- 2022	15
03/16/2022	16-March- 2022	16
03/17/2022	17-March- 2022	17
03/18/2022	18-March- 2022	18
03/19/2022	19-March- 2022	19
03/20/2022	20-March- 2022	20
03/21/2022	21-March- 2022	21

該当月の名前は、スクリプトの month() 関数により正常に評価されています。

例 3 – 形式設定のない日付 (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- Master_Calendar という名前の日付のデータセット。 DateFormat システム変数 DD/MM/YYYY が使用されます。
- month() 関数を使用して month_name という名前の追加項目を作成する先行するロード。
- unformatted_date という名前の、形式設定がない元の日付。
- date() 関数を使用して完全な日付を表示する、long_date という名前の追加項目。

ロード スクリプト

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

Master_Calendar:
Load
      unformatted_date,
      date(unformatted_date,'dd-MMMM-YYYY') as long_date,
      month(unformatted_date) as month_name

Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- unformatted_date
- long_date
- month_name

結果テーブル

unformatted_date	long_date	month_name
44868	03-January- 2022	Jan
44898	03-February- 2022	Feb
44928	03-March- 2022	Mar
44958	03-April- 2022	Apr

unformatted_date	long_date	month_name
44988	03-May- 2022	May
45018	03-June- 2022	Jun
45048	03-July- 2022	Jul
45078	03-August- 2022	Aug
45008	03-September- 2022	Sep
45038	03-October- 2022	Oct
45068	03-November- 2022	Nov

該当月の名前は、スクリプトの month() 関数により正常に評価されています。

例 4 – 失効月の計算

ロードスクリプトとチャートの数式

概要

データロードエディター を開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています:

- 3 月に注文があった Subscriptions という名前のデータセット。テーブルには 3 項目が含まれています。
  - ID
  - order_date
  - amount

ロードスクリプト

```
Subscriptions:
Load
        id,
        order_date,
        amount
Inline
[
id,order_date,amount
1,03/01/2022,231.24
2,03/02/2022,567.28
3,03/03/2022,364.28
4,03/04/2022,575.76
5,03/05/2022,638.68
6,03/06/2022,785.38
7,03/07/2022,967.46
8,03/08/2022,287.67
9,03/09/2022,764.45
```

```
10,03/10/2022,875.43
11,03/11/2022,957.35
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: order_date。

注文の有効期限が切れる月を計算するには、メジャー =month(order_date+180) を作成します。

結果テーブル

order_date	=month(order_date+180)
03/01/2022	Jul
03/02/2022	Aug
03/03/2022	Aug
03/04/2022	Sep
03/05/2022	Oct
03/06/2022	Nov
03/07/2022	Dec
03/08/2022	Jan
03/09/2022	Mar
03/10/2022	Apr
03/11/2022	May

month() 関数は、3 月 11 日にあった注文は 7 月に失効すると正しく確定します。

monthend

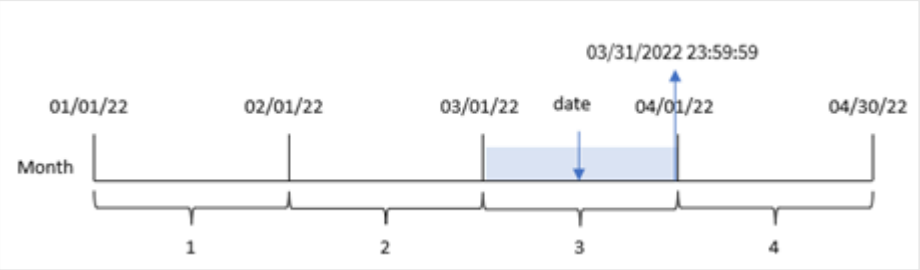
この関数は、date を含む月の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている DateFormat です。

構文:

```
MonthEnd(date[, period_no])
```

つまり、monthend() 関数は、日付がどの月に該当するかを判断します。次に、その月の最後のミリ秒のタイムスタンプを日付形式で返します。

monthend 関数の図。



使用に適しているケース

monthend() 関数は、ユーザーがまだ発生していない月の端数を計算に使用する場合に、数式の一部として使用されます。たとえば、その月にまだ発生していない利息の合計を計算したい場合などに使います。

戻り値データ型： dual

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数であり、0 が指定された場合または省略された場合、**date** を含む月を示します。**period_no** の値が負の場合は過去の月を、正の場合は将来の月を示します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
monthend('02/19/2012')	02/29/2012 23:59:59 を返します。
monthend('02/19/2001', -1)	01/31/2001 23:59:59 を返します。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数 (MM/DD/YYYY) 形式の日付項目。
- 次を含む、先行する LOAD ステートメント:
    - [end_of_month] 項目として設定されている monthend() 関数。
    - [end_of_month_timestamp] 項目として設定されている timestamp 関数。

ロード スクリプト

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

Transactions:
    Load
    *,
    monthend(date) as end_of_month,
    timestamp(monthend(date)) as end_of_month_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- end_of_month
- end_of_month_timestamp

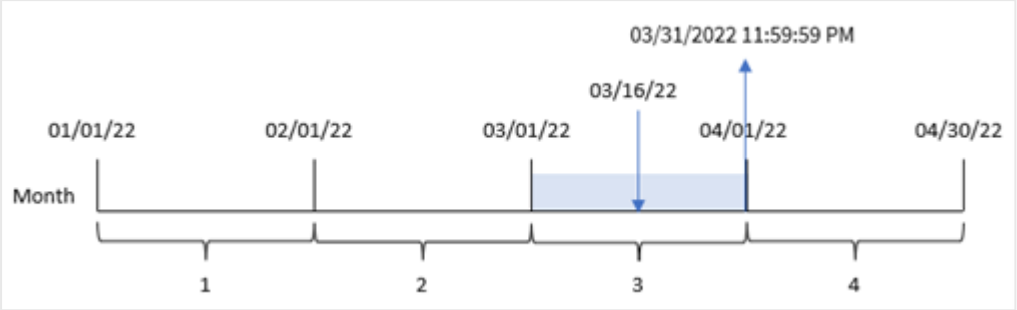
結果テーブル

ID	日付	end_of_month	end_of_month_timestamp
8188	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8189	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8199	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8200	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8201	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

[end_of_month] 項目は、monthend() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

monthend() 関数は、日付値がどの月に該当するかを識別し、その月の最後のミリ秒のタイムスタンプを返します。

選択した月が 3 月である monthend 関数の図。



トランザクション 8192 は 3 月 16 日に発生しました。monthend() 関数は、その月の最後のミリ秒、つまり3 月 31 日午後 11:59:59 を返します。

例 2 – period_no

ロード スクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

この例では、タスクはトランザクションが発生する前の月の終わりのタイムスタンプを返す、項目 [previous_month_end] の作成です。

ロード スクリプト

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

Transactions:
    Load
    *,
    monthend(date,-1) as previous_month_end,
    timestamp(monthend(date,-1)) as previous_month_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
```

```
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- previous_month_end
- previous_month_end_timestamp

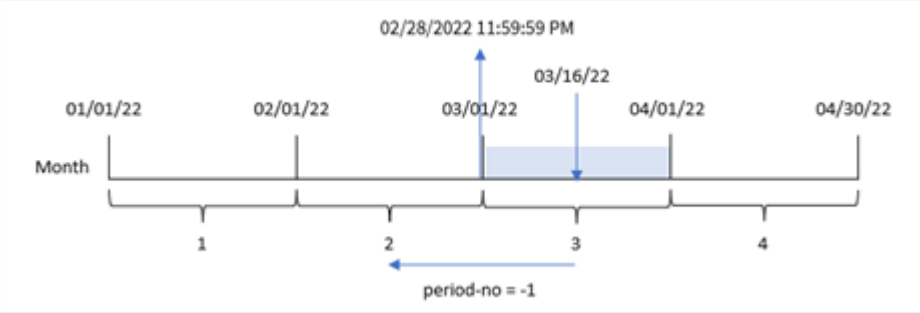
結果テーブル

ID	日付	previous_month_end	previous_month_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	01/31/2022	1/31/2022 11:59:59 PM
8191	2/28/2022	01/31/2022	1/31/2022 11:59:59 PM
8192	3/16/2022	02/28/2022	2/28/2022 11:59:59 PM
8193	4/1/2022	03/31/2022	3/31/2022 11:59:59 PM
8194	5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
8195	5/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8196	6/15/2022	05/31/2022	5/31/2022 11:59:59 PM
8197	6/26/2022	05/31/2022	5/31/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	07/31/2022	7/31/2022 11:59:59 PM
8203	8/8/2022	07/31/2022	7/31/2022 11:59:59 PM
8204	8/19/2022	07/31/2022	7/31/2022 11:59:59 PM

ID	日付	previous_month_end	previous_month_end_timestamp
8205	9/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8206	10/14/2022	09/30/2022	9/30/2022 11:59:59 PM
8207	10/29/2022	09/30/2022	9/30/2022 11:59:59 PM

-1 の period_no がオフセット引数として使用されたため、monthend() 関数は最初にトランザクションが発生する月を識別します。次に、1 か月前にずらして、その月の最後のミリ秒を識別します。

period_no 変数を持つ monthend 関数の図。



トランザクション 8192 は 3 月 16 日に発生しました。monthend() 関数は、トランザクションが発生した前の月が 2 月であったことを特定します。次に、その月の最後のミリ秒、2 月 28 日午後 11:59:59 を返します。

例 3 – チャートの例

ロードスクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

この例では、データセットは変更されず、アプリにロードされます。タスクは、トランザクションが発生した月の終わりのタイムスタンプをアプリのチャートのメジャーとして返す計算を作成することです。

ロードスクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
```

```
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- `date`
- `id`

トランザクションが発生する月の終わりを計算するには、次のメジャーを作成します。

- `=monthend(date)`
- `=timestamp(monthend(date))`

結果テーブル

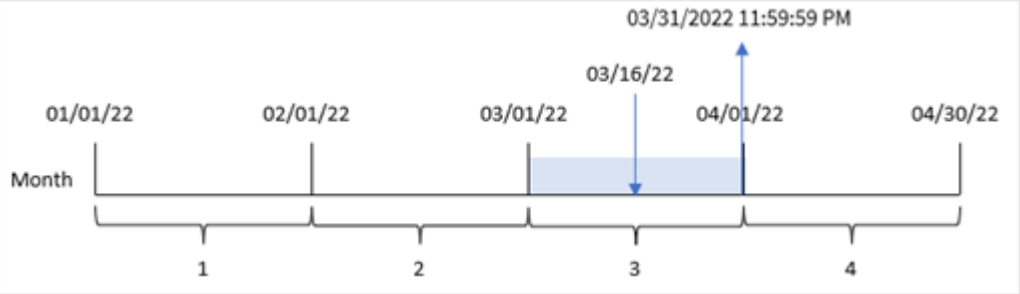
ID	日付	=monthend(date)	=timestamp(monthend(date))
8188	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8189	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM
8190	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8191	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8192	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8193	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8194	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8195	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8196	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8197	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8198	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8201	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM

ID	日付	=monthend(date)	=timestamp(monthend(date))
8202	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8203	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8204	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8205	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8206	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8207	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM

「end_of_month」メジャーは、monthend() 関数を使用し、関数の引数として日付項目を渡すことにより、チャートで作成されます。

monthend() 関数は、日付値がどの月に該当するかを識別し、その月の最後のミリ秒のタイムスタンプを返します。

period_no 変数を持つmonthend 関数の図。



トランザクション 8192 は 3 月 16 日に発生しました。monthend() 関数は、その月の最後のミリ秒、つまり3 月 31 日午後 11:59:59 を返します。

例 4 – シナリオ

ロードスクリプトと結果

概要

この例では、「Employee_Expenses」という名前のテーブルにデータセットがロードされます。テーブルには次の項目が含まれています。

- 従業員 ID
- 従業員名
- 各従業員の平均日次経費請求。

エンドユーザーは、従業員 ID と従業員名別に、その月の残りの期間にまだ発生する推定経費請求を表示するチャートを求めています。

ロード スクリプト

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- employee_id
- employee_name

累積利息を計算するには、このメジャーを作成します。

```
=floor(monthend(today(1),0)-today(1))*avg_daily_claim
```

このメジャーは動的であり、データをロードする日付によって異なるテーブル結果が生じます。

メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

employee_id	employee_name	=floor(monthend(today(1),0)-today(1))*avg_daily_claim
182	Mark	$30.00
183	Deryck	$25.00
184	Dexter	$25.00
185	Sydney	$54.00
186	Agatha	$36.00

monthend() 関数は、今日の日付を唯一の引数として使用することにより、現在の月の終了日を返します。式は、月の終了日から今日の日付を引くことによって、今月の残りの日数を返します。

次に、この値に各従業員による1日あたりの平均経費請求額を乗算して、月の残り期間に各従業員が行うと予想される請求の推定額を計算します。

monthname

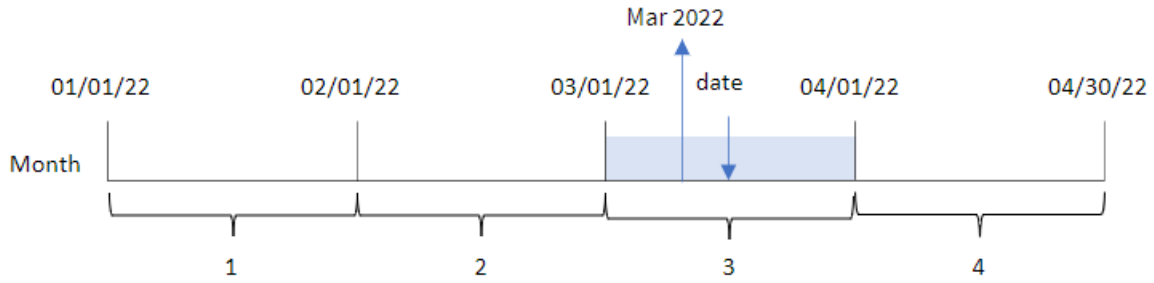
この関数は、月の初日の最初のミリ秒のタイムスタンプに対応する基底の数値を持つ、月 (**MonthNames** スクリプト変数に従った書式) および年の表示値を返します。

構文:

```
MonthName(date[, period_no])
```

戻り値データ型: dual

monthname 関数の図



引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数であり、0 が指定された場合または省略された場合、**date** を含む月を示します。**period_no** の値が負の場合は過去の月を、正の場合は将来の月を示します。

関数の例

例	結果
`monthname('10/19/2013')`	Oct 2013 を返します
`monthname('10/19/2013', -1)`	Sep 2013 を返します

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの `SET DateFormat` ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロード

エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 基本的な例

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生した月を返す項目 [transaction_month] の作成。

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        monthname(date) as transaction_month
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
```

```
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

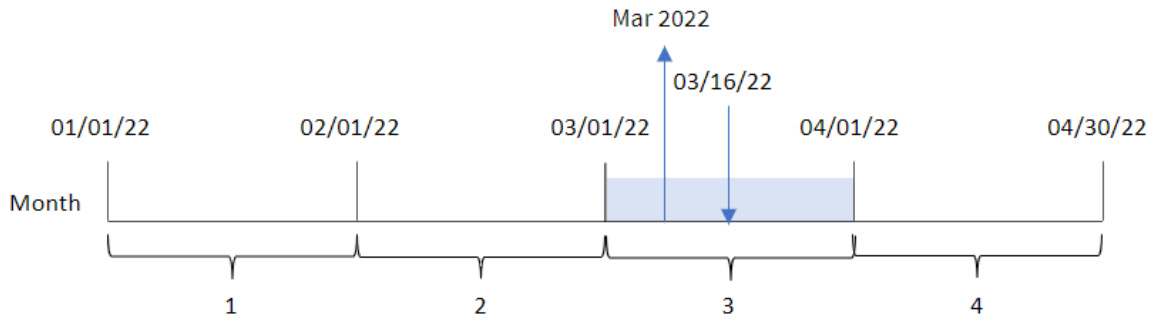
- date
- transaction_month

結果テーブル

日付	transaction_month
1/7/2022	2022 年 1 月
1/19/2022	2022 年 1 月
2/5/2022	2022 年 2 月
2/28/2022	2022 年 2 月
3/16/2022	2022 年 3 月
4/1/2022	2022 年 4 月
5/7/2022	2022 年 5 月
5/16/2022	2022 年 5 月
6/15/2022	2022 年 6 月
6/26/2022	2022 年 6 月
7/9/2022	2022 年 7 月
7/22/2022	2022 年 7 月
7/23/2022	2022 年 7 月
7/27/2022	2022 年 7 月
8/2/2022	2022 年 8 月
8/8/2022	2022 年 8 月
8/19/2022	2022 年 8 月
9/26/2022	2022 年 9 月
10/14/2022	2022 年 10 月
10/29/2022	2022 年 10 月

transaction_month 項目は、monthname() 関数を使用し、関数の引数として date 項目を渡すことにより、先行する LOAD ステートメントで作成されます。

monthname 関数の図、基本的な例



monthname() 関数は、トランザクション 8192 が 2022 年 3 月に発生したことを特定し、MonthNames システム変数を使用してこの値を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じ inline データセットとシナリオ。
- トランザクションが発生する前の月の終わりのタイムスタンプを返す、項目 [transaction_previous_month] の作成。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        monthname(date,-1) as transaction_previous_month
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
```

```
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- `date`
- `transaction_previous_month`

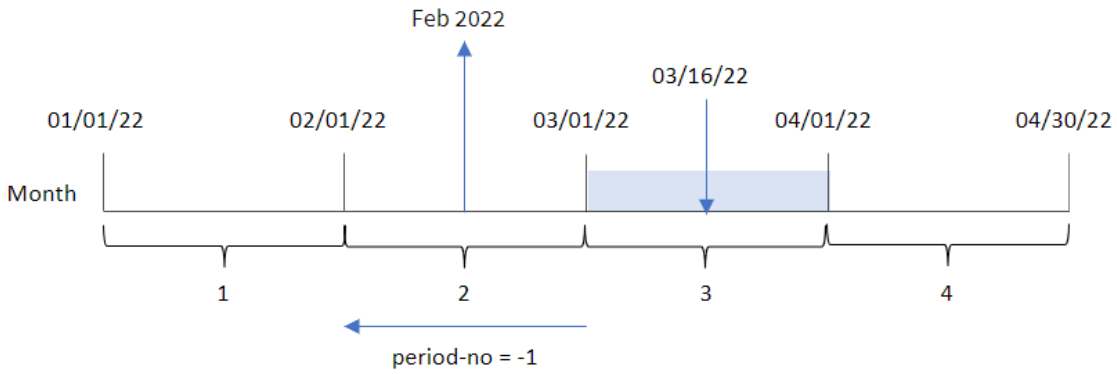
結果テーブル

日付	transaction_previous_month
1/7/2022	2021 年 12 月
1/19/2022	2021 年 12 月
2/5/2022	2022 年 1 月
2/28/2022	2022 年 1 月
3/16/2022	2022 年 2 月
4/1/2022	2022 年 3 月
5/7/2022	2022 年 4 月
5/16/2022	2022 年 4 月
6/15/2022	2022 年 5 月
6/26/2022	2022 年 5 月
7/9/2022	2022 年 6 月
7/22/2022	2022 年 6 月
7/23/2022	2022 年 6 月
7/27/2022	2022 年 6 月
8/2/2022	2022 年 7 月
8/8/2022	2022 年 7 月
8/19/2022	2022 年 7 月

日付	transaction_previous_month
9/26/2022	2022 年 8 月
10/14/2022	2022 年 9 月
10/29/2022	2022 年 9 月

この例では、-1 の period_no が monthname() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した月を識別します。次に、1 か月前に移動し、月名と年を返します。

monthname 関数の図、period_no の例



トランザクション 8192 は 3 月 16 日に発生しました。monthname() 関数は、トランザクションが発生する前の月が 2 月であることを識別し、2022 年と共に MonthNames システム変数形式でその月を返します。

例 3 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じ inline データセットとシナリオが含まれます。ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した月の終わりのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトの メジャーとして作成されます。

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
Load
*
Inline
[
```

```
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。

次のメジャーを作成します:

```
=monthname(date)
```

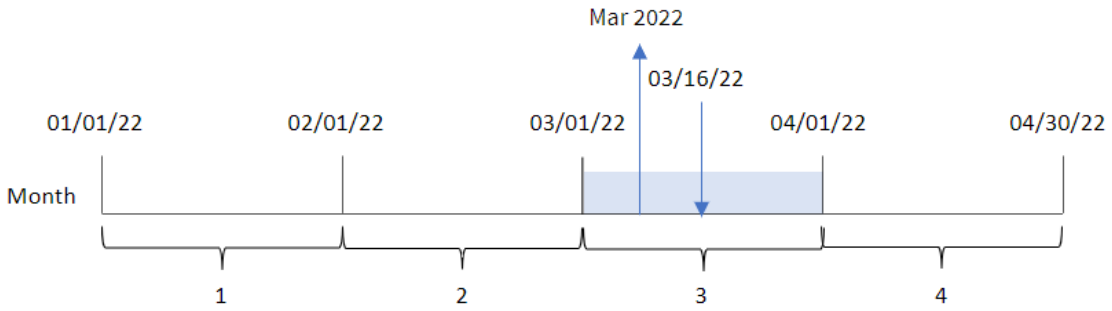
結果テーブル

日付	=monthname(date)
1/7/2022	2022 年 1 月
1/19/2022	2022 年 1 月
2/5/2022	2022 年 2 月
2/28/2022	2022 年 2 月
3/16/2022	2022 年 3 月
4/1/2022	2022 年 4 月
5/7/2022	2022 年 5 月
5/16/2022	2022 年 5 月
6/15/2022	2022 年 6 月
6/26/2022	2022 年 6 月
7/9/2022	2022 年 7 月

日付	=monthname(date)
7/22/2022	2022 年 7 月
7/23/2022	2022 年 7 月
7/27/2022	2022 年 7 月
8/2/2022	2022 年 8 月
8/8/2022	2022 年 8 月
8/19/2022	2022 年 8 月
9/26/2022	2022 年 9 月
10/14/2022	2022 年 10 月
10/29/2022	2022 年 10 月

[month_name] メジャーは、monthname() 関数を使用し、関数の引数として [date] 項目を渡すことにより、チャートオブジェクトで作成されます。

monthname 関数の図、チャートオブジェクトの例



monthname() 関数は、トランザクション 8192 が 2022 年 3 月に発生したことを特定し、MonthNames システム変数を使用してこの値を返します。

monthsend

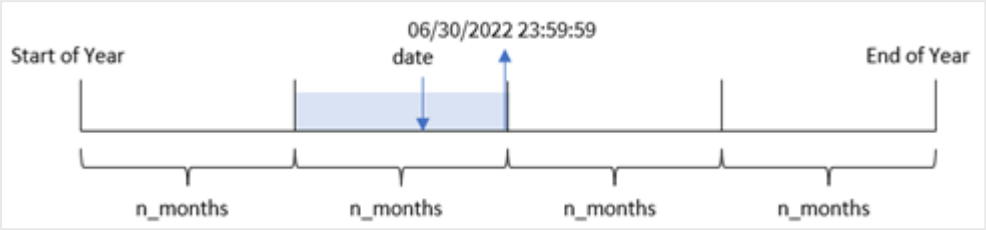
この関数は、ベース日付を含む月、2 か月、四半期、4 か月、半年のいずれかの期間の最後のミリ秒のタイムスタンプに相当する値を返します。その前後の期間の終わりのタイムスタンプを取得することもできます。デフォルトの出力形式は、スクリプトに設定されている DateFormat です。

構文:

```
MonthsEnd(n_months, date[, period_no [, first_month_of_year]])
```

戻り値データ型： dual

monthsend 関数の図。



引数

引数	説明
n_months	期間を定義する月数。整数、または計算結果が整数になる数式で次のうちのいずれかでなければならない。1 (inmonth() 関数と同機能)、2 (2 か月)、3 (inquarter() 関数と同機能)、4 (4 か月)、6 (半年)。
date	評価する日付またはタイムスタンプ。
period_no	期間は、**period_no**、整数、計算結果が整数になる数式を使用して補正できます。値 0 は **base_date** を含む期間を示します。**period_no** の値が負の場合は過去の期間を、正の場合は将来の期間を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

monthsend() 関数は、指定された n_months 引数に基づいて年をセグメントに分割します。次に、提供された各日付がどのセグメントに該当するかを評価し、そのセグメントの最後のミリ秒を日付形式で返します。関数は、前後のセグメントから終了タイムスタンプを返すことも、年の最初の月を最適することもできます。

次の年のセグメントは、n_month 引数として関数で使用できます。

n_month 引数

期間	月数
月	1
隔月	2
四半期	3
4 か月	4
半年	6

使用に適しているケース

monthsend() 関数は、ユーザーがこれまで経過した月の端数を計算に使用する場合に、数式の一部として使用されます。ユーザーには、変数を使用して、選択した項目の期間を選ぶ機会があります。例えば、monthsend() が入力変数を提供すると、ユーザーが月、四半期、半年にまだ未発生の合計利息を計算することができます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
monthsend(4, '07/19/2013')	08/31/2013 を返します。
monthsend(4, '10/19/2013', -1)	08/31/2013 を返します。
monthsend(4, '10/19/2013', 0, 2)	01/31/2014 を返します。 年の開始は、2 月からです。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数 (MM/DD/YYYY) 形式で提供される日付項目。
- 次を含む先行する LOAD ステートメント:
    - 項目 [bi_monthly_end] として設定されている monthsend 関数。これにより、トランザクションが隔月セグメントにグループ化されます。
    - 各トランザクションに対してセグメントの開始タイムスタンプを返す timestamp 関数。

ロード スクリプト

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

Transactions:
    Load
    *,
    monthsend(2,date) as bi_monthly_end,
    timestamp(monthsend(2,date)) as bi_monthly_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- bi_monthly_end
- bi_monthly_end_timestamp

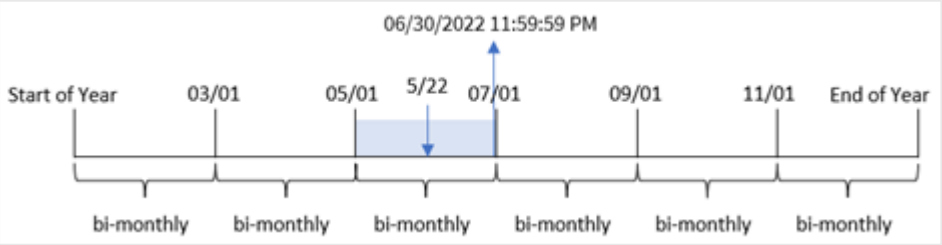
結果テーブル

ID	日付	bi_monthly_end	bi_monthly_end_timestamp
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM

ID	日付	bi_monthly_end	bi_monthly_end_timestamp
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	10/31/2022	10/31/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

[bi_monthly_end] 項目は、monthsend() 関数を使用して、前の Load ステートメントで作成されます。提供される最初の引数は 2 で、年を 2 か月のセグメントに分割します。2 番目の引数は、評価される項目を識別します。

2 か月セグメントの monthsend 関数の図。



トランザクション 8195 は 5 月 22 日に発生します。 関数は、最初に年を 2 か月のセグメントに分割します。monthsend() トランザクション 8195 は、5 月 1 日 ~ 6 月 30 日のセグメントに分類されます。結果として、関数はこのセグメントの最後のミリ秒 06/30/2022 11:59:59 PM を返します。

例 2 – period_no

ロード スクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

この例では、タスクはトランザクションが発生する前の 2 か月セグメントの最初のタイムスタンプを返す項目 [prev_bi_monthly_end] の作成です。

ロード スクリプト

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

Transactions:
    Load
    *,
    monthsend(2,date,-1) as prev_bi_monthly_end,
    timestamp(monthsend(2,date,-1)) as prev_bi_monthly_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

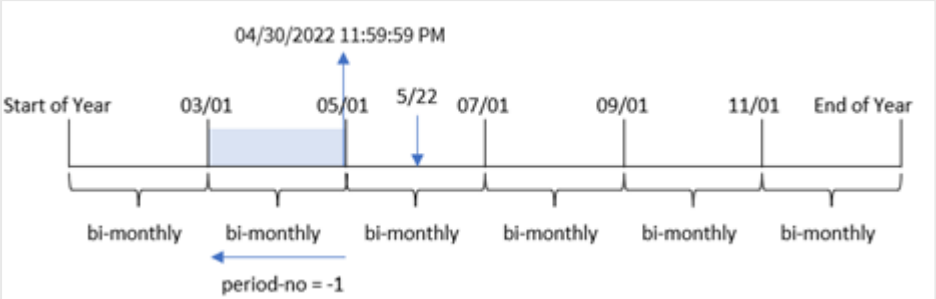
- id
- date
- prev_bi_monthly_end
- prev_bi_monthly_end_timestamp

結果テーブル

ID	日付	prev_bi_monthly_end	prev_bi_monthly_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	12/31/2021	12/31/2021 11:59:59 PM
8191	2/28/2022	12/31/2021	12/31/2021 11:59:59 PM
8192	3/16/2022	02/28/2022	2/28/2022 11:59:59 PM
8193	4/1/2022	02/28/2022	2/28/2022 11:59:59 PM
8194	5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
8195	5/22/2022	04/30/2022	4/30/2022 11:59:59 PM
8196	6/15/2022	04/30/2022	4/30/2022 11:59:59 PM
8197	6/26/2022	04/30/2022	4/30/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	06/30/2022	6/30/2022 11:59:59 PM
8203	8/8/2022	06/30/2022	6/30/2022 11:59:59 PM
8204	8/19/2022	06/30/2022	6/30/2022 11:59:59 PM
8205	9/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8206	10/14/2022	08/31/2022	8/31/2022 11:59:59 PM
8207	10/29/2022	08/31/2022	8/31/2022 11:59:59 PM

-1 を monthsend() 関数の period_no 引数として使用することにより、最初に 1 年を 2 か月のセグメントに分割した後、関数はトランザクションが発生したときの以前の 2 か月セグメントの最後のミリ秒を返します。

前の 2 か月のセグメントを返す monthsend 関数の図。



トランザクション 8195 は、5 月 ~ 6 月のセグメントに発生します。その結果、前の 2 か月セグメントは 3 月 1~ 4 月 30 日だったので、関数はこのセグメントの最後のミリ秒 04/30/2022 11:59:59 PM を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

この例では、組織ポリシーでは 4 月が会計期間の開始月に定められています。

トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセグメントの最後のミリ秒タイムスタンプを返す項目 [bi_monthly_end] を作成します。

ロードスクリプト

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

Transactions:
    Load
    *,
    monthsend(2,date,0,4) as bi_monthly_end,
    timestamp(monthsend(2,date,0,4)) as bi_monthly_end_timestamp
    ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- bi_monthly_end
- bi_monthly_end_timestamp

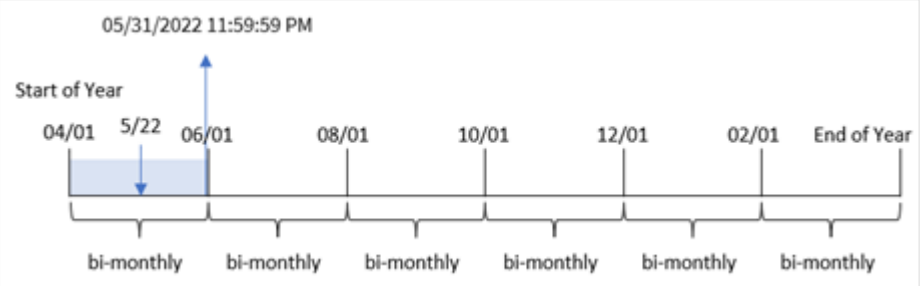
結果テーブル

ID	日付	bi_monthly_end	bi_monthly_end_timestamp
8188	1/7/2022	01/31/2022	1/31/2022 11:59:59 PM
8189	1/19/2022	01/31/2022	1/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	05/31/2022	5/31/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/22/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	07/31/2022	7/31/2022 11:59:59 PM
8197	6/26/2022	07/31/2022	7/31/2022 11:59:59 PM
8198	7/9/2022	07/31/2022	7/31/2022 11:59:59 PM
8199	7/22/2022	07/31/2022	7/31/2022 11:59:59 PM
8200	7/23/2022	07/31/2022	7/31/2022 11:59:59 PM
8201	7/27/2022	07/31/2022	7/31/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM

ID	日付	bi_monthly_end	bi_monthly_end_timestamp
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	11/30/2022	11/30/2022 11:59:59 PM
8207	10/29/2022	11/30/2022	11/30/2022 11:59:59 PM

monthsend() 関数の first_month_of_year 引数に 4 を使用することにより、関数は 4 月 1 日に年度を開始します。その後その年度を 2 か月単位のセグメントに分割します。4 月 ~ 5 月、6 月 ~ 7 月、8 月 ~ 9 月、10 月 ~ 11 月、12 月 ~ 1 月、2 月 ~ 3 月。

4 月が年の最初の月に設定された monthsend 関数の図。



トランザクション 8195 は 5 月 22 日に発生し、4 月 1 日 ~ 5 月 31 日のセグメントに分類されます。結果として、関数はこのセグメントの最後のミリ秒 05/31/2022 11:59:59 PM を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。ただし、この例ではデータセットは変更されず、アプリにロードされます。

この例で、タスクはトランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセグメントの最後のミリ秒タイムスタンプを、アプリのチャート オブジェクトでメジャーとして返す計算を作成することです。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
```

```
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

```
date
```

トランザクションが発生した 2 か月セグメントの最後のミリ秒タイムスタンプをフェッチするには、次のメジャーを作成します。

- `=monthsEnd(2,date)`
- `=timestamp(monthsend(2,date))`

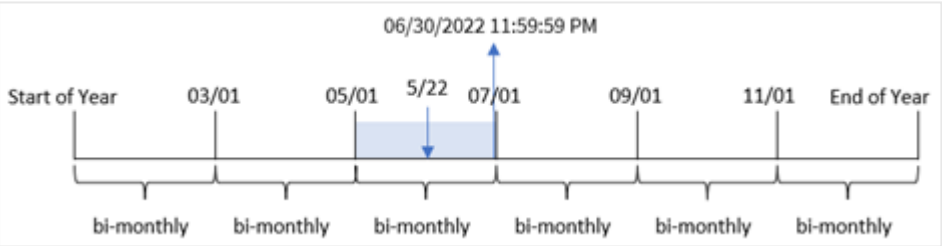
結果テーブル

ID	日付	=monthsend(2,date)	=timestamp(monthsend(2,date))
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	04/30/2022	4/30/2022 11:59:59 PM
8193	4/1/2022	04/30/2022	4/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM

ID	日付	=monthsend(2,date)	=timestamp(monthsend(2,date))
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	10/31/2022	10/31/2022 11:59:59 PM
8206	10/14/2022	10/31/2022	10/31/2022 11:59:59 PM
8207	10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

[bi_monthly_end] 項目は、monthsend() 関数を使用することにより、チャートオブジェクトに作成されます。提供される最初の引数は 2 で、これは年を 2 か月のセグメントに分割します。2 番目の引数は、評価される項目を識別します。

2 か月セグメントの monthsend 関数の図。



トランザクション 8195 は 5 月 22 日に発生します。 関数は、最初に年を 2 か月のセグメントに分割します。monthsend() トランザクション 8195 は、5 月 1 日 ~ 6 月 30 日のセグメントに分類されます。結果として、関数はこのセグメントの最初のミリ秒 06/30/2022 11:59:59 PM を返します。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

この例では、「Employee_Expenses」という名前のテーブルにデータセットがロードされます。テーブルには次の項目が含まれています。

- 従業員 ID
- 従業員名

- 各従業員の平均日次経費請求。

エンドユーザーは、従業員 ID と従業員名別に、自分で選択した期間の残りの期間にまだ発生する推定経費請求を表示するチャートを求めています。会計年度は 1 月に始まります。

ロード スクリプト

```
SET vPeriod = 1;

Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

結果

データをロードして新しいシートを開きます。

ロード スクリプトの開始時には、変数入力コントロールに関連付けられる変数 (vPeriod) が作成されます。

以下を実行します。

1. アセット パネルで、[**カスタム オブジェクト**] をクリックします。
2. [**Qlik ダッシュボード バンドル**] を選択し、**変数入力** オブジェクトを作成します。
3. チャート オブジェクトのタイトルを入力します。
4. [**変数**] で、[名前] に [**vPeriod**] を選択し、オブジェクトを [**ドロップ ダウン**] として表示するように設定します。
5. [**値**] で、[**ダイナミック**] 値をクリックします。以下を入力します。
   `='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'.`

新しいテーブルと、これらの項目を軸として作成します。

- employee_id
- employee_name

累積利息を計算するには、このメジャーを作成します。

```
=floor(monthsend($(vPeriod),today(1))-today(1))*avg_daily_claim
```

このメジャーは動的であり、データをロードする日付によって異なるテーブル結果が生じます。

メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

employee_id	employee_name	=floor(monthsend($(vPeriod),today(1))-today(1))*avg_daily_claim
182	Mark	$1410.00
183	Deryck	$1175.00
184	Dexter	$1175.00
185	Sydney	$2538.00
186	Agatha	$1692.00

monthsend() 関数は、最初の引数としてユーザー入力を、2 番目の因数として今日の日付を使用します。これは、ユーザーが選択した期間の最終日付を返します。次に、式は、この終了日から今日の日付を引くことによって、選択した期間の残りの日数を返します。

次に、この値に各従業員による1日あたりの平均経費請求額を乗算して、この期間の残り日数に各従業員が行うと予想される請求の推定額を計算します。

monthsname

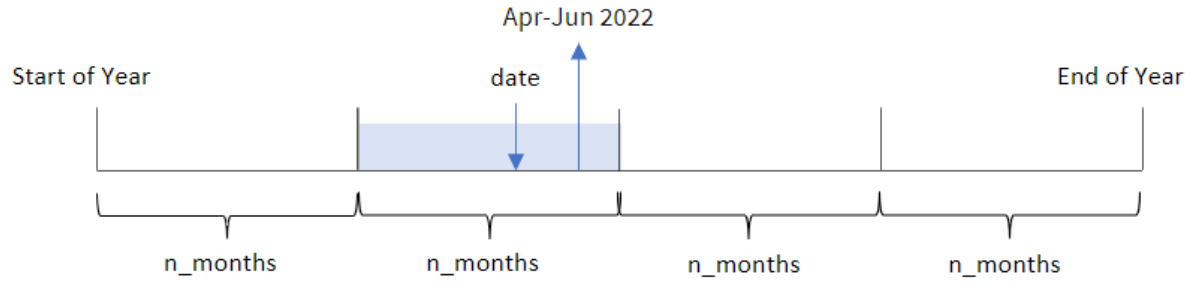
この関数は、期間の月の範囲 (**MonthNames** スクリプト変数に従った書式で表示) および年を表す表示値を返します。基底値は、ベース日付を含む月、2 か月、四半期、4 か月、半年のいずれかの期間の最初のミリ秒のタイムスタンプに相当する値です。

構文:

```
MonthsName(n_months, date[, period_no[, first_month_of_year]])
```

戻り値データ型: dual

monthsname 関数の図



monthsname() 関数は、指定された n_months 引数に基づいて年をセグメントに分割します。次に、提供された各 date が属するセグメントを評価し、そのセグメントの開始月名と終了月名、および年を返します。この関数には、前後のセグメントからこれらの境界を返したり、年の最初の月を再定義したりする機能もあります。

次の年のセグメントは、n_month 引数として関数で使用できます。

考えられるn_month引数

期間	月数
月	1
隔月	2
四半期	3
4か月	4
半年	6

引数

引数	説明
n_months	期間を定義する月数。整数、または計算結果が整数になる数式で次のうちのいずれかでなければならない。1 (inmonth() 関数と同機能)、2 (2か月)、3 (inquarter() 関数と同機能)、4 (4か月)、6 (半年)。
date	評価する日付またはタイムスタンプ。
period_no	期間は、**period_no**、整数、計算結果が整数になる数式を使用して補正できます。値 0 は **base_date** を含む期間を示します。**period_no** の値が負の場合は過去の期間を、正の場合は将来の期間を示します。
first_month_of_year	事業年度が1月以外の月に始まる場合は、**first_month_of_year** で 2 から12 の間の値を指定します。

使用に適しているケース

monthsname() 関数は、ユーザーが選択した期間で集計を比較する機能をユーザーに提供することを希望する場合に便利です。たとえば、入力変数を提供して、ユーザーが月、四半期、または半年ごとに製品の総売上を確認できるようにすることができます。

これらの軸は、マスター カレンダー テーブルの項目として関数を追加することによってロード スクリプトで作成するか、計算軸としてチャートに軸を直接作成することによって作成できます。

関数の例

例	結果
`monthsname(4, '10/19/2013')`	「Sep-Dec 2013」を返します。この例と他の例では、**SET Monthnames** ステートメントが Jan;Feb;Marのように設定されています。
`monthsname(4, '10/19/2013', -1)`	「May-Aug 2013」を返します
`monthsname(4, '10/19/2013', 0, 2)`	年は月 2で始まるように指定されているため、「Oct-Jan 2014」を返します。そのため、この 4 か月の期間は次の年の最初の月に終了します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセグメントの境界名を返す項目 [bi_monthly_range] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        monthsname(2,date) as bi_monthly_range
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
```

```
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- bi_monthly_range

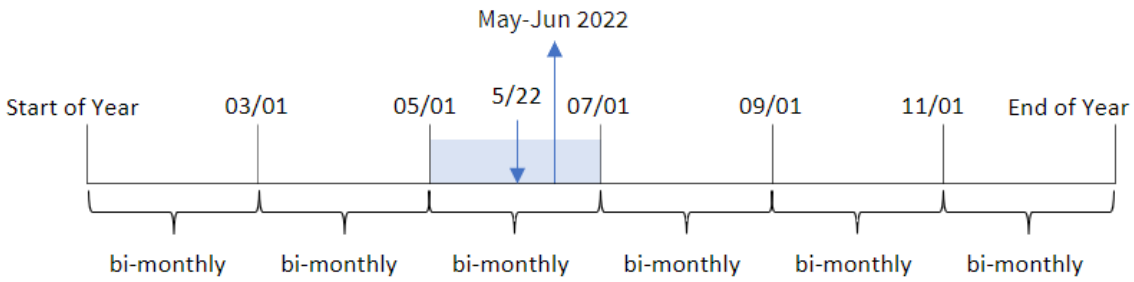
結果テーブル

日付	bi_monthly_range
2/19/2022	2022 年 1 月 ~ 2 月
3/7/2022	2022 年 3 月 ~ 4 月
3/30/2022	2022 年 3 月 ~ 4 月
4/5/2022	2022 年 3 月 ~ 4 月
4/16/2022	2022 年 3 月 ~ 4 月
5/1/2022	2022 年 5 月 ~ 6 月
5/7/2022	2022 年 5 月 ~ 6 月
5/22/2022	2022 年 5 月 ~ 6 月
6/15/2022	2022 年 5 月 ~ 6 月
6/26/2022	2022 年 5 月 ~ 6 月
7/9/2022	2022 年 7 月 ~ 8 月
7/22/2022	2022 年 7 月 ~ 8 月
7/23/2022	2022 年 7 月 ~ 8 月
7/27/2022	2022 年 7 月 ~ 8 月
8/2/2022	2022 年 7 月 ~ 8 月
8/8/2022	2022 年 7 月 ~ 8 月
8/19/2022	2022 年 7 月 ~ 8 月

日付	bi_monthly_range
9/26/2022	2022 年 9 月 ~ 10 月
10/14/2022	2022 年 9 月 ~ 10 月
10/29/2022	2022 年 9 月 ~ 10 月

bi_monthly_range 項目は、monthsname() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は 2 で、年を 2 か月のセグメントに分割します。2 番目の引数は、評価される項目を識別します。

monthsname 関数の図、基本的な例



トランザクション 8195 は 5 月 22 日に発生します。 関数は、最初に年を 2 か月のセグメントに分割します。monthsname() トランザクション 8195 は、5 月 1 日 ~ 6 月 30 日のセグメントに分類されます。したがって、この関数はこれらの月を MonthNames システム変数形式で返し、2022 年 5 月 ~ 6 月の年も返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じ inline データセットとシナリオ。
- トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとに前のセグメントの境界名を返す項目 [prev_bi_monthly_range] の作成。

必要に応じて、リストなどで他のテキストをここに追加します。

ロードスクリプト

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

Transactions:
    Load
        *,
```

```
        MonthsName(2,date,-1) as prev_bi_monthly_range
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- prev_bi_monthly_range

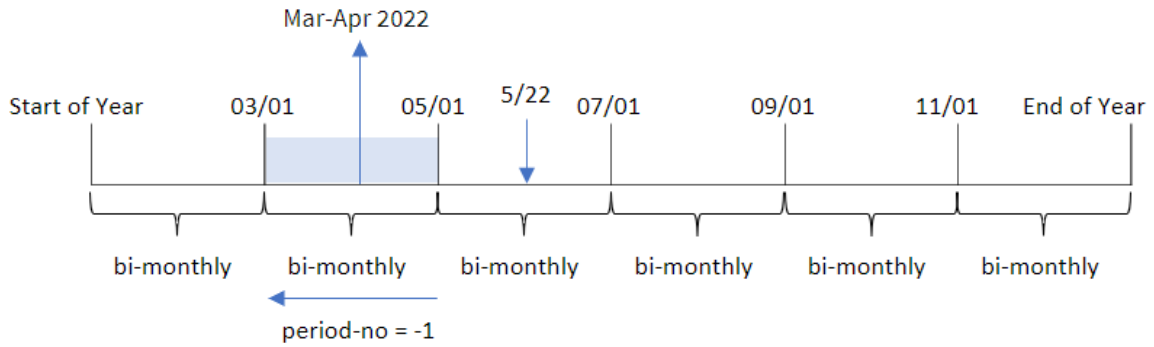
結果テーブル

日付	prev_bi_monthly_range
2/19/2022	2021 年 11 月 ~ 12 月
3/7/2022	2022 年 1 月 ~ 2 月
3/30/2022	2022 年 1 月 ~ 2 月
4/5/2022	2022 年 1 月 ~ 2 月
4/16/2022	2022 年 1 月 ~ 2 月
5/1/2022	2022 年 3 月 ~ 4 月
5/7/2022	2022 年 3 月 ~ 4 月
5/22/2022	2022 年 3 月 ~ 4 月

日付	prev_bi_monthly_range
6/15/2022	2022 年 3 月 ~ 4 月
6/26/2022	2022 年 3 月 ~ 4 月
7/9/2022	2022 年 5 月 ~ 6 月
7/22/2022	2022 年 5 月 ~ 6 月
7/23/2022	2022 年 5 月 ~ 6 月
7/27/2022	2022 年 5 月 ~ 6 月
8/2/2022	2022 年 5 月 ~ 6 月
8/8/2022	2022 年 5 月 ~ 6 月
8/19/2022	2022 年 5 月 ~ 6 月
9/26/2022	2022 年 7 月 ~ 8 月
10/14/2022	2022 年 7 月 ~ 8 月
10/29/2022	2022 年 7 月 ~ 8 月

この例では、monthsname() 関数の period_no 引数として -1 が使用されています。最初に 1 年を 2 か月のセグメントに分割した後、この関数はトランザクションが発生したときの以前のセグメント境界を返します。

monthsname 関数の図、period_no の例



トランザクション 8195 は、5 月 ~ 6 月のセグメントに発生します。したがって、前の 2 か月のセグメントは 3 月 1 日から 4 月 30 日の間であったため、この関数は 2022 年 3 月から 4 月を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じ inline データセットとシナリオ。
- トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセグメントの境界を返す別の項目 [bi_monthly_range] の作成。

ただし、この例では、4 月を会計年度の最初の月として設定する必要もあります。

ロードスクリプト

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

Transactions:
    Load
        *,
        MonthsName(2,date,0,4) as bi_monthly_range
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- bi_monthly_range

結果テーブル

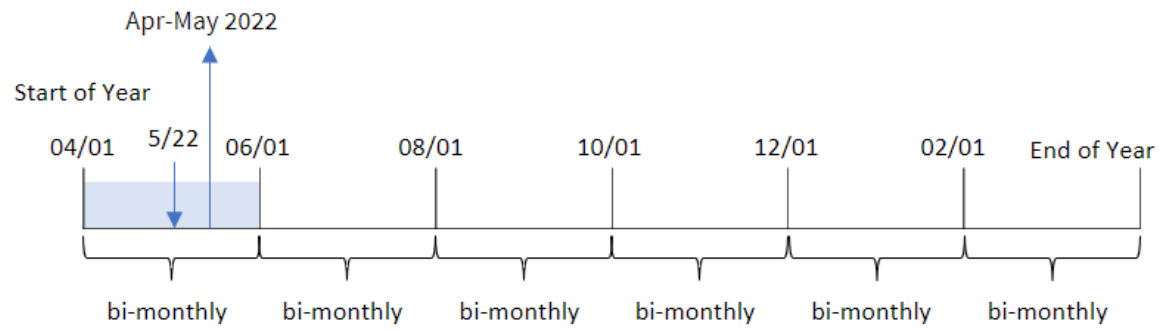
日付	bi_monthly_range
2/19/2022	2021 年 2 月 ~ 3 月
3/7/2022	2021 年 2 月 ~ 3 月
3/30/2022	2021 年 2 月 ~ 3 月
4/5/2022	2022 年 4 月 ~ 5 月
4/16/2022	2022 年 4 月 ~ 5 月
5/1/2022	2022 年 4 月 ~ 5 月
5/7/2022	2022 年 4 月 ~ 5 月
5/22/2022	2022 年 4 月 ~ 5 月
6/15/2022	2022 年 6 月 ~ 7 月
6/26/2022	2022 年 6 月 ~ 7 月
7/9/2022	2022 年 6 月 ~ 7 月
7/22/2022	2022 年 6 月 ~ 7 月
7/23/2022	2022 年 6 月 ~ 7 月
7/27/2022	2022 年 6 月 ~ 7 月
8/2/2022	2022 年 8 月 ~ 9 月
8/8/2022	2022 年 8 月 ~ 9 月
8/19/2022	2022 年 8 月 ~ 9 月
9/26/2022	2022 年 8 月 ~ 9 月
10/14/2022	2022 年 10 月 ~ 11 月
10/29/2022	2022 年 10 月 ~ 11 月

`monthsname()` 関数の `first_month_of_year` 引数に 4 を使用することにより、関数は 4 月 1 日に年度を開始します。その後その年度を 2 か月単位のセグメントに分割します。4 月 ~ 5 月、6 月 ~ 7 月、8 月 ~ 9 月、10 月 ~ 11 月、12 月 ~ 1 月、2 月 ~ 3 月

結果の段落テキスト。

トランザクション 8195 は 5 月 22 日に発生し、4 月 1 日 ~ 5 月 31 日のセグメントに分類されます。したがって、この関数は 2022 年 4 月 ~ 5 月を返します。

monthsname 関数の図、first_month_of_year の例



例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じ inline データセットとシナリオが含まれます。ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセグメントの境界を返す計算は、アプリケーションのチャート オブジェクトでメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
```

```
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

次のメジャーを作成します:

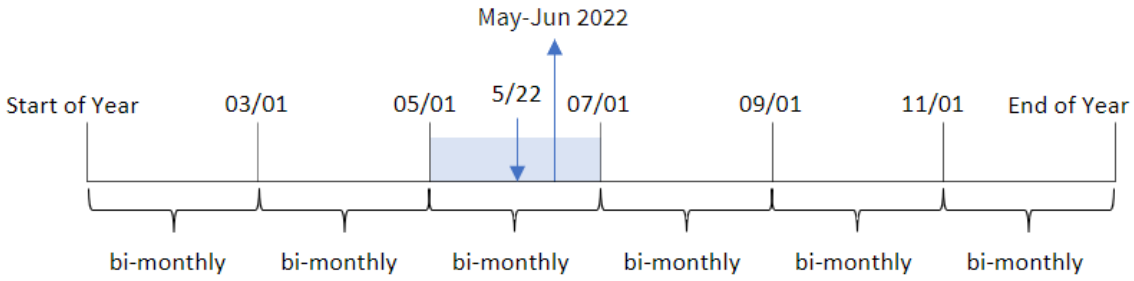
```
=monthsname(2,date)
```

結果テーブル

日付	=monthsname(2,date)
2/19/2022	2022 年 1 月 ~ 2 月
3/7/2022	2022 年 3 月 ~ 4 月
3/30/2022	2022 年 3 月 ~ 4 月
4/5/2022	2022 年 3 月 ~ 4 月
4/16/2022	2022 年 3 月 ~ 4 月
5/1/2022	2022 年 5 月 ~ 6 月
5/7/2022	2022 年 5 月 ~ 6 月
5/22/2022	2022 年 5 月 ~ 6 月
6/15/2022	2022 年 5 月 ~ 6 月
6/26/2022	2022 年 5 月 ~ 6 月
7/9/2022	2022 年 7 月 ~ 8 月
7/22/2022	2022 年 7 月 ~ 8 月
7/23/2022	2022 年 7 月 ~ 8 月
7/27/2022	2022 年 7 月 ~ 8 月
8/2/2022	2022 年 7 月 ~ 8 月
8/8/2022	2022 年 7 月 ~ 8 月
8/19/2022	2022 年 7 月 ~ 8 月
9/26/2022	2022 年 9 月 ~ 10 月
10/14/2022	2022 年 9 月 ~ 10 月
10/29/2022	2022 年 9 月 ~ 10 月

[bi_monthly_range] 項目は、monthsname() 関数を使用することにより、チャート オブジェクトに作成されます。提供される最初の引数は 2 で、年を 2 か月のセグメントに分割します。2 番目の引数は、評価される項目を識別します。

monthsname 関数の図、チャート オブジェクトの例



トランザクション 8195 は 5 月 22 日に発生します。 関数は、最初に年を 2 か月のセグメントに分割します。monthsname() トランザクション 8195 は、5 月 1 日 ~ 6 月 30 日のセグメントに分類されます。したがって、この関数はこれらの月を MonthNames システム変数形式で返し、2022 年 5 月 ~ 6 月の年も返します。

例 5 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。

エンド ユーザーは、自分で選択した期間ごとの総売上高を表示するチャート オブジェクトを望んでいます。これは、変数入力コントロールによって動的に変更される計算軸として monthsname() 関数を使用して、この軸がデータ モデルで使用できない場合でも実現できます。

ロード スクリプト

```
SET vPeriod = 1;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/7/2022',17.17
8189,'1/19/2022',37.23
```

```
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

データをロードしてシートを開きます。

ロードスクリプトの開始時には、変数入力コントロールに関連付けられる変数 (vPeriod) が作成されています。次に、変数をシートのカスタム オブジェクトとして構成します。

次の手順を実行します。

1. アセット パネルで、[**カスタム オブジェクト**] をクリックします。
2. [**Qlik ダッシュボード バンドル**] を選択し、**変数入力** オブジェクトを作成します。
3. チャート オブジェクトのタイトルを入力します。
4. [**変数**] で、[名前] に [**vPeriod**] を選択し、オブジェクトを [**ドロップ ダウン**] として表示するように設定します。
5. [**値**] で、動的な値を使用するようにオブジェクトを構成します。以下を入力します。
   ='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'

次に、結果テーブルを作成します。

次の手順を実行します。

1. 新しいテーブルを作成し、次の計算軸を追加します。
   =monthsname($(vPeriod),date)
2. このメジャーを追加して、総売上を計算します。
   =sum(amount)
3. メジャーの [**数値書式**] を [**通貨**] に設定します。[✔ **編集の完了**] をクリックします。変数オブジェクトの時間セグメントを調整することで、テーブルに表示されているデータを変更できるようになりました。

[tertial] オプションを選択した場合の結果テーブルは次のようになります。

結果テーブル

monthsname($(vPeriod),date)	=sum(amount)
2022 年 1 月 ~ 4 月	253.89
2021 年 5 月 ~ 8 月	713.58
2022 年 9 月 ~ 12 月	248.12

monthsstart

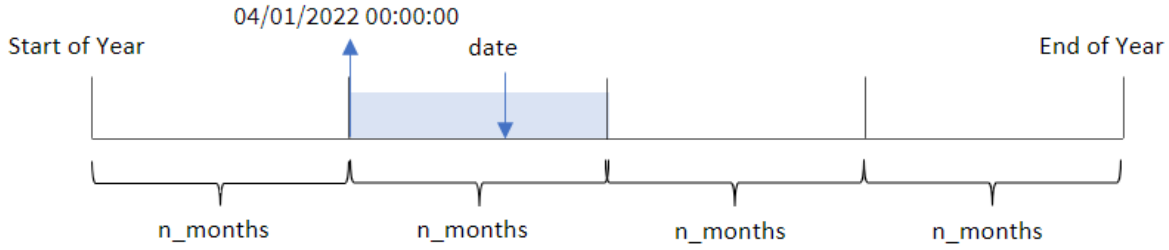
この関数は、ベース日付を含む月、2 か月、四半期、4 か月、半年のいずれかの期間の最初のミリ秒のタイムスタンプに相当する値を返します。その前後の期間のタイムスタンプを取得することもできます。既定の出力形式は、スクリプトに設定されている **DateFormat** です。

構文:

```
MonthsStart(n_months, date[, period_no [, first_month_of_year]])
```

戻り値データ型: dual

monthsstart() 関数の図



monthsstart() 関数は、指定された n_months 引数に基づいて年をセグメントに分割します。次に、提供された各日付がどのセグメントに該当するかを評価し、そのセグメントの最初のミリ秒を日付形式で返します。この関数には、前後のセグメントから開始タイムスタンプを返したり、年の最初の月を再定義したりする機能もあります。

次の年のセグメントは、n_month 引数として関数で使用できます。

考えられる n_month 引数

期間	月数
月	1
隔月	2
四半期	3
4 か月	4
半年	6

引数

引数	説明
n_months	期間を定義する月数。整数、または計算結果が整数になる数式で次のうちのいずれかでなければならない。1 (inmonth() 関数と同機能)、2 (2 か月)、3 (inquarter() 関数と同機能)、4 (4 か月)、6 (半年)。
date	評価する日付またはタイムスタンプ。
period_no	期間は、**period_no**、整数、計算結果が整数になる数式を使用して補正できます。値 0 は **base_date** を含む期間を示します。**period_no** の値が負の場合は過去の期間を、正の場合は将来の期間を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

使用に適しているケース

monthsstart() 関数は、ユーザーがまだ発生していない期間の端数を計算に使用する場合に、数式の一部として一般的に使用されます。これは、例えば、入力変数を提供して、ユーザーが月、四半期、半年でこれまでに累積した合計利息を計算するために使用できます。

関数の例

例	結果
`monthsstart(4, '10/19/2013')`	09/01/2013 を返します。
`monthsstart(4, '10/19/2013, -1)`	05/01/2013 を返します。
`monthsstart(4, '10/19/2013', 0, 2 )`	年の開始が 2 月になるため、10/01/2013 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセグメントの開始タイムスタンプを返す項目 [bi_monthly_start] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        monthsstart(2,date) as bi_monthly_start,
        timestamp(monthsstart(2,date)) as bi_monthly_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

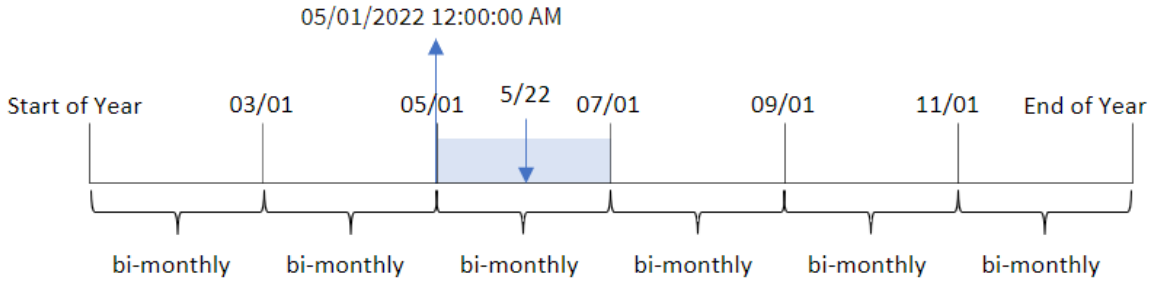
- date
- bi_monthly_start
- bi_monthly_start_timestamp

結果テーブル

日付	bi_monthly_start	bi_monthly_start_timestamp
2/19/2022	01/01/2022	1/1/2022 12:00:00 AM
3/7/2022	03/01/2022	3/1/2022 12:00:00 AM
3/30/2022	03/01/2022	3/1/2022 12:00:00 AM
4/5/2022	03/01/2022	3/1/2022 12:00:00 AM
4/16/2022	03/01/2022	3/1/2022 12:00:00 AM
5/1/2022	05/01/2022	5/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/22/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

bi_monthly_start 項目は、monthsstart() 関数を使用して、先行するLOAD ステートメントで作成されます。提供される最初の引数は 2 で、年を 2 か月のセグメントに分割します。2 番目の引数は、評価される項目を識別します。

monthsstart() 関数の図、追加の引数がない例



トランザクション 8195 は 5 月 22 日に発生します。 関数は、最初に年を 2 か月のセグメントに分割します。 monthsstart() トランザクション 8195 は、5 月 1 日 ~ 6 月 30 日のセグメントに分類されます。そのため、関数はこのセグメントの最初のミリ秒 2022 年 5 月 1 日 12:00:00 AM を返します。

例 2 – period_no

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- タスクはトランザクションが発生する前の 2 か月セグメントの最初のタイムスタンプを返す項目 [prev_bi_monthly_start] の作成。

ロード スクリプト

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

Transactions:
    Load
        *,
        monthsstart(2,date,-1) as prev_bi_monthly_start,
           timestamp(monthsstart(2,date,-1)) as prev_bi_monthly_start_timestamp
           ;
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
```

```
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- prev_bi_monthly_start
- prev_bi_monthly_start_timestamp

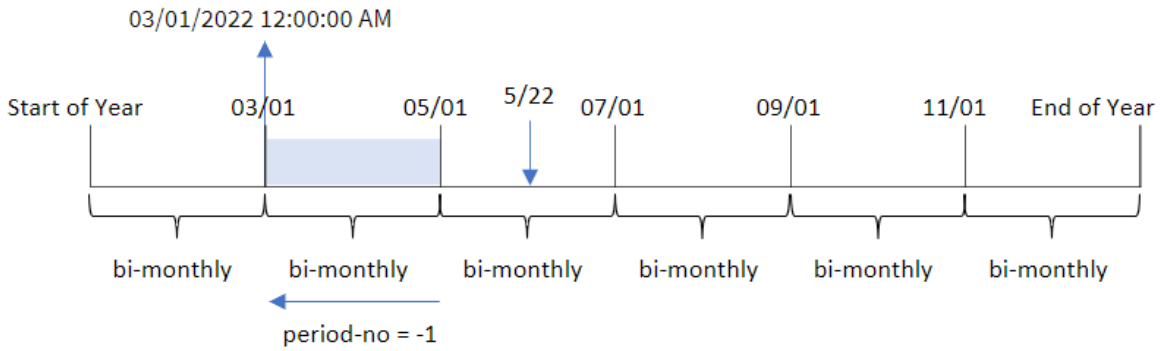
結果テーブル

日付	prev_bi_monthly_start	prev_bi_monthly_start_timestamp
2/19/2022	11/01/2021	11/1/2021 12:00:00 AM
3/7/2022	01/01/2022	1/1/2022 12:00:00 AM
3/30/2022	01/01/2022	1/1/2022 12:00:00 AM
4/5/2022	01/01/2022	1/1/2022 12:00:00 AM
4/16/2022	01/01/2022	1/1/2022 12:00:00 AM
5/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	03/01/2022	3/1/2022 12:00:00 AM
5/22/2022	03/01/2022	3/1/2022 12:00:00 AM
6/15/2022	03/01/2022	3/1/2022 12:00:00 AM
6/26/2022	03/01/2022	3/1/2022 12:00:00 AM
7/9/2022	05/01/2022	5/1/2022 12:00:00 AM
7/22/2022	05/01/2022	5/1/2022 12:00:00 AM
7/23/2022	05/01/2022	5/1/2022 12:00:00 AM
7/27/2022	05/01/2022	5/1/2022 12:00:00 AM
8/2/2022	05/01/2022	5/1/2022 12:00:00 AM

日付	prev_bi_monthly_start	prev_bi_monthly_start_timestamp
8/8/2022	05/01/2022	5/1/2022 12:00:00 AM
8/19/2022	05/01/2022	5/1/2022 12:00:00 AM
9/26/2022	07/01/2022	7/1/2022 12:00:00 AM
10/14/2022	07/01/2022	7/1/2022 12:00:00 AM
10/29/2022	07/01/2022	7/1/2022 12:00:00 AM

-1 を period_no 関数の monthsstart() 引数として使用することにより、最初に 1 年を 2 か月のセグメントに分割した後、関数はトランザクションが発生したときの以前の 2 か月セグメントの最初のミリ秒を返します。

monthsstart() 関数の図、period_no の例



トランザクション 8195 は、5 月 ~ 6 月のセグメントに発生します。そのため、前の 2 か月セグメントは 3 月 1~ 4 月 30 日だったため、関数はこのセグメントの最初のミリ秒 2022 年 3 月 1 日 12:00:00 AM を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセットの開始タイムスタンプを返す項目 [bi_monthly_start] の作成。

ただし、この例では、4 月を会計年度の最初の月として設定する必要もあります。

ロードスクリプト

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

```
Transactions:
    Load
        *,
        monthsstart(2,date,0,4) as bi_monthly_start,
        timestamp(monthsstart(2,date,0,4)) as bi_monthly_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- bi_monthly_start
- bi_monthly_start_timestamp

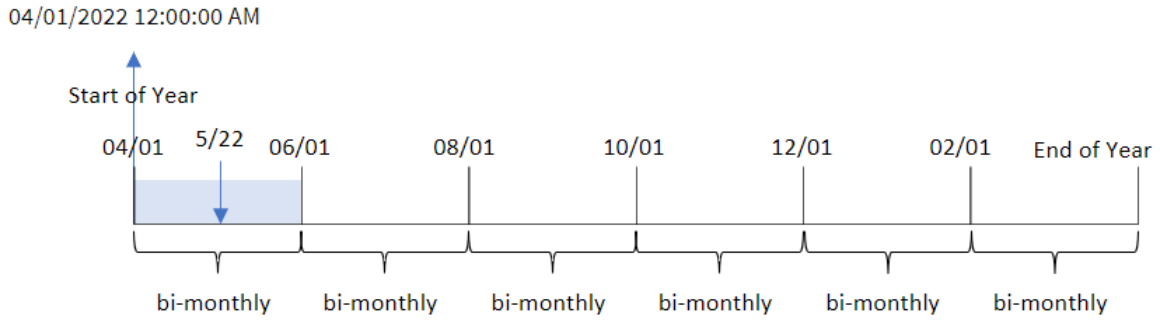
結果テーブル

日付	bi_monthly_start	bi_monthly_start_timestamp
2/19/2022	02/01/2022	2/1/2022 12:00:00 AM
3/7/2022	02/01/2022	2/1/2022 12:00:00 AM
3/30/2022	02/01/2022	2/1/2022 12:00:00 AM
4/5/2022	04/01/2022	4/1/2022 12:00:00 AM
4/16/2022	04/01/2022	4/1/2022 12:00:00 AM

日付	bi_monthly_start	bi_monthly_start_timestamp
5/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/22/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
9/26/2022	08/01/2022	8/1/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

`monthsstart()` 関数の `first_month_of_year` 引数に 4 を使用することにより、関数は 4 月 1 日に年度を開始します。その後その年度を 2 か月単位のセグメントに分割します。4 月 ~ 5 月、6 月 ~ 7 月、8 月 ~ 9 月、10 月 ~ 11 月、12 月 ~ 1 月、2 月 ~ 3 月

`monthsstart()` 関数の図、first_month_of_year の例



トランザクション 8195 は 5 月 22 日に発生し、4 月 1 日 ~ 5 月 31 日のセグメントに分類されます。そのため、関数はこのセグメントの最初のミリ秒 2022 年 4 月 1 日 12:00:00 AM を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションを 2 か月のセグメントにグループ化し、トランザクションごとにそのセットの開始タイムスタンプを返す計算は、アプリケーションのチャート オブジェクトでメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,2/19/2022,37.23
8189,3/7/2022,17.17
8190,3/30/2022,88.27
8191,4/5/2022,57.42
8192,4/16/2022,53.80
8193,5/1/2022,82.06
8194,5/7/2022,40.39
8195,5/22/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `date`。

次のメジャーを作成します:

```
=monthsstart(2,date)
```

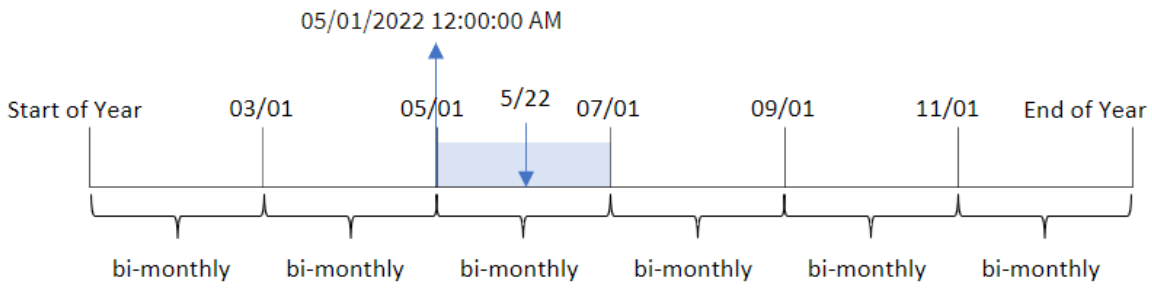
```
=timestamp(monthsstart(2,date))
```

これらの計算は、各トランザクションが発生する2か月の開始タイムスタンプを取得します。

結果テーブル

日付	=monthsstart(2,date)	=timestamp(monthsstart(2,date))
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
5/1/2022	05/01/2022	5/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/22/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
3/7/2022	03/01/2022	3/1/2022 12:00:00 AM
3/30/2022	03/01/2022	3/1/2022 12:00:00 AM
4/5/2022	03/01/2022	3/1/2022 12:00:00 AM
4/16/2022	03/01/2022	3/1/2022 12:00:00 AM
2/19/2022	01/01/2022	1/1/2021 12:00:00 AM

monthsstart() 関数の図、チャートオブジェクトの例



トランザクション 8195 は 5 月 22 日に発生しました。 関数は、最初に年を 2 か月のセグメントに分割します。 monthsstart() トランザクション 8195 は、5 月 1 日 ~ 6 月 30 日のセグメントに分類されます。そのため、関数はこのセグメントの最初のミリ秒 05/01/2022 12:00:00 AM を返します。

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Loans というテーブルにロードされる、一連のローン残高を含むデータセット。
- ローン ID、月の開始の残高、各ローンにかかる単利の年率で構成されるデータ。

エンドユーザーは、選択した期間の各ローンで発生した現在の利息をローン ID 別に表示するチャートオブジェクトを求めています。会計年度は 1 月に始まります。

ロードスクリプト

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

Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

結果

データをロードしてシートを開きます。

ロードスクリプトの開始時には、変数入力コントロールに関連付けられる変数 (vPeriod) が作成されています。次に、変数をシートのカスタムオブジェクトとして構成します。

次の手順を実行します。

1. アセットパネルで、[**カスタム オブジェクト**] をクリックします。
2. [**Qlik ダッシュボード バンドル**] を選択し、**変数入力** オブジェクトを作成します。
3. チャートオブジェクトのタイトルを入力します。

4. [**変数**] で、[名前] に [**vPeriod**] を選択し、オブジェクトを [**ドロップ ダウン**] として表示するように設定します。
5. [**値**] で、動的な値を使用するようにオブジェクトを構成します。以下を入力します。
   ='1~month|2~bi-month|3~quarter|4~tertial|6~half-year'

次に、結果テーブルを作成します。

次の手順を実行します。

1. 新しいテーブルを作成します。次の項目を軸として追加します。
   - employee_id
   - employee_name
2. メジャーを作成して、累積利息を計算します。
   =start_balance*(rate*(today(1)-monthsstart($(vPeriod),today(1)))/365)
3. メジャーの [**数値書式**] を [**通貨**] に設定します。[✔ **編集の完了**] をクリックします。変数オブジェクトの時間セグメントを調整することで、テーブルに表示されているデータを変更できるようになりました。

month 期間オプションを選択した場合の結果テーブルは次のようになります。

結果テーブル

loan_id	start_ balance	=start_balance*(rate*(today(1)-monthsstart($(vPeriod),today (1)))/365)
8188	$10000.00	$7.95
8189	$15000.00	$67.93
8190	$17500.00	$33.37
8191	$21000.00	$56.73
8192	$90000.00	$600.66

monthsstart() 関数は、最初の引数としてユーザーの入力を、2 番目の引数として今日の日付を使用し、ユーザーが選択した期間の開始日付を返します。その結果を現在の日付から減算することにより、数式はこの期間で経過した日数を返します。

次に、この値に利率を乗算して 365 で除算すると、この期間に発生する実効利率が返されます。次に、結果にローンの開始残高を掛けると、この期間でこれまでに発生した利息を返されます。

monthstart

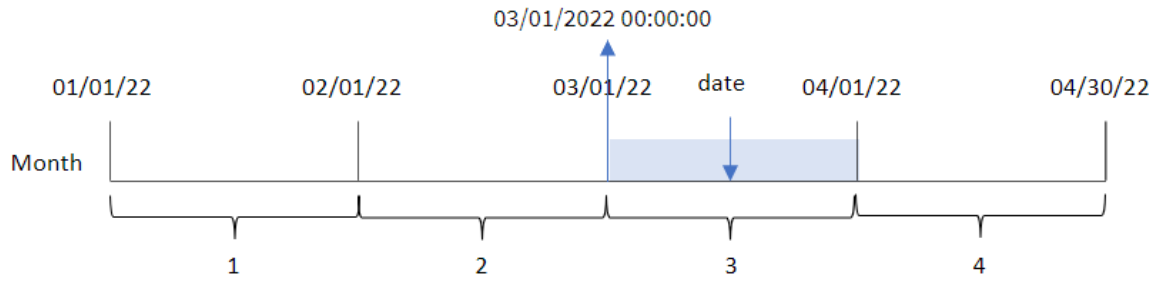
この関数は、**date** を含む月の初日の最初のミリ秒のタイムスタンプに対応する値を返します。デフォルトの出力形式は、スクリプトに設定されている **DateFormat** です。

構文:

```
MonthStart(date[, period_no])
```

戻り値データ型： dual

monthstart() 関数の図



monthstart() 関数は、日付がどの月に該当するかを判断します。次に、その月の最初のミリ秒のタイムスタンプを日付形式で返します。

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数であり、0 が指定された場合または省略された場合、**date** を含む月を示します。**period_no** の値が負の場合は過去の月を、正の場合は将来の月を示します。

使用に適しているケース

monthstart() 関数は、ユーザーがこれまで経過した月の端数を計算に使用する場合に、数式の一部として一般的に使用されます。たとえば、月の特定の日付までに累積した利息を計算するのに使用できます。

関数の例

例	結果
`monthstart('10/19/2001')`	10/01/2001 を返します。
`monthstart('10/19/2001', -1)`	09/01/2001 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの `SET DateFormat` ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する月の始めのタイムスタンプを返す、項目 [start_of_month] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        monthstart(date) as start_of_month,
        timestamp(monthstart(date)) as start_of_month_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- start_of_month
- start_of_month_timestamp

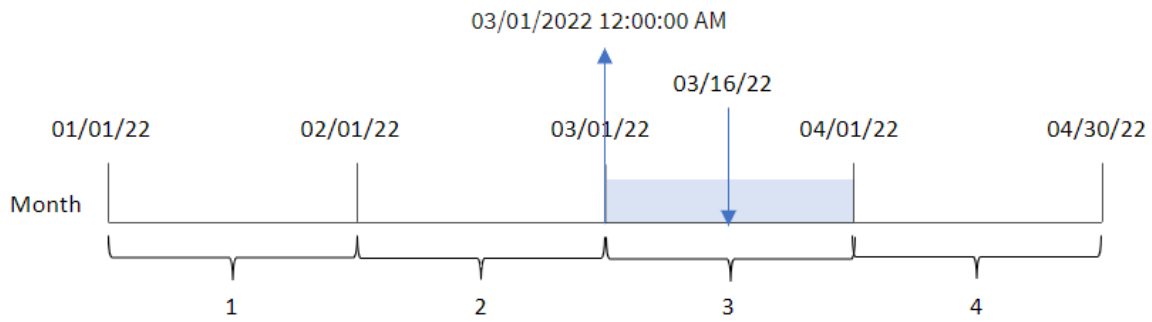
結果テーブル

日付	start_of_month	start_of_month_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/01/2022	2/1/2022 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/01/2022	5/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	07/01/2022	6/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

start_of_month 項目は、monthstart() 関数を使用し、関数の引数として日付項目を渡すことにより、先行するLOAD ステートメントで作成されます。

monthstart() 関数は、日付値がどの月に該当するかを識別し、その月の最初のミリ秒のタイムスタンプを返します。

monthstart() 関数の図、追加の引数がない例



トランザクション 8192 は 3 月 16 日に発生しました。monthstart() 関数は、その月の最初のミリ秒、つまり3月 1 日午前 12:00:00 を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の月の始めのタイムスタンプを返す、項目 [previous_month_start] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        monthstart(date,-1) as previous_month_start,
        timestamp(monthstart(date,-1)) as previous_month_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
```

```
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_month_start
- previous_month_start_timestamp

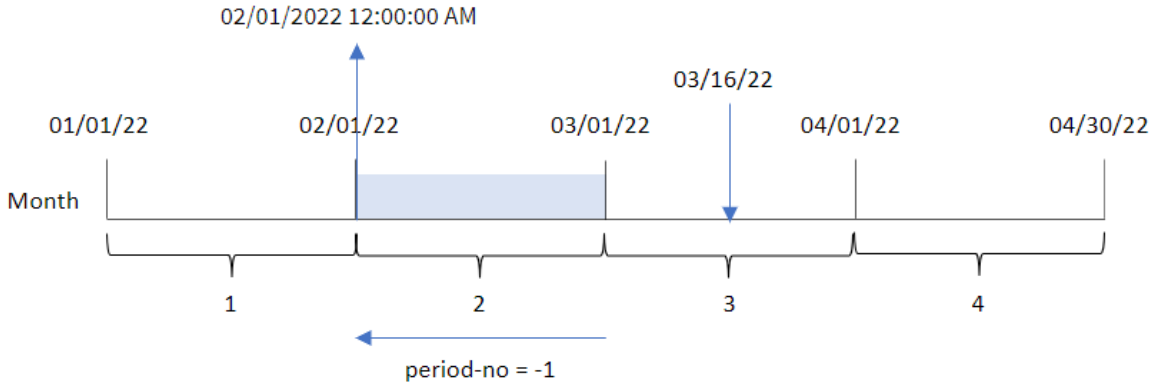
結果テーブル

日付	previous_month_start	previous_month_start_timestamp
1/7/2022	12/01/2021	12/1/2021 12:00:00 AM
1/19/2022	12/01/2021	12/1/2021 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	02/01/2022	2/1/2022 12:00:00 AM
4/1/2022	03/01/2022	3/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/16/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	05/01/2022	5/1/2022 12:00:00 AM
6/26/2022	05/01/2022	5/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM

日付	previous_month_start	previous_month_start_timestamp
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	08/01/2022	8/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

この例では、-1 の period_no が monthstart() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した月を識別します。次に、1 か月前にずらして、その月の最初のミリ秒を識別します。

monthstart() 関数の図、period_no の例



トランザクション 8192 は 3 月 16 日に発生しました。monthstart() 関数は、トランザクションが発生した前の月が 2 月であったことを特定します。次に、その月の最初のミリ秒、2 月 1 日 12:00:00 AM を返します。

例 3 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した月の始めのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

トランザクションが発生する月の始めを計算するには、次のメジャーを作成します。

- =monthstart(date)
- =timestamp(monthstart(date))

結果テーブル

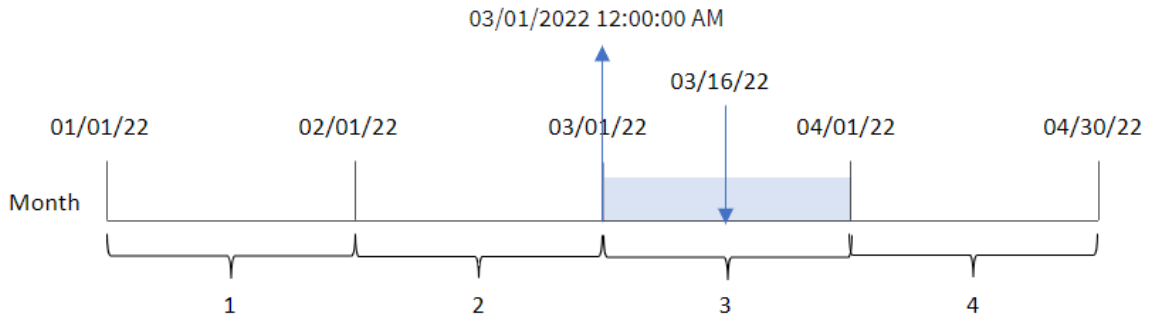
日付	=monthstart(date)	=timestamp(monthstart(date))
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
8/2/2022	08/01/2022	8/1/2022 12:00:00 AM
8/8/2022	08/01/2022	8/1/2022 12:00:00 AM
8/19/2022	08/01/2022	8/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM

日付	=monthstart(date)	=timestamp(monthstart(date))
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/01/2022	5/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/01/2022	2/1/2022 12:00:00 AM
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM

「start_of_month」メジャーは、monthstart() 関数を使用し、関数の引数として日付項目を渡すことにより、チャート オブジェクトで作成されます。

monthstart() 関数は、日付値がどの月に該当するかを識別し、その月の最初のミリ秒のタイムスタンプを返します。

monthstart() 関数の図、チャート オブジェクトの例



トランザクション 8192 は 3 月 16 日に発生しました。monthstart() 関数は、トランザクションが 3 月に発生したことを特定し、その月の最初のミリ秒、つまり3 月 1 日午前 12:00:00 を返します。

例 4 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Loans というテーブルにロードされる、一連のローン残高を含むデータセット。
- ローンID、月の開始の残高、各ローンにかかる単利の年率で構成されるデータ。

エンドユーザーは、月初来の各ローンで発生した現在の利息をローンID別に表示するチャートオブジェクトを求めています。

ロードスクリプト

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

Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:
   - loan_id
   - start_balance
2. 次に、メジャーを作成して、累積利息を計算します。
   `=start_balance*(rate*(today(1)-monthstart(today(1)))/365)`
3. メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8188	$10000.00	$16.44

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8189	$15000.00	$58.56
8190	$17500.00	$28.77
8191	$21000.00	$48.90
8192	$90000.00	$517.81

monthstart() 関数は、今日の日付を唯一の引数として使用することにより、現在の月の開始日を返します。その結果を現在の日付から減算することにより、数式は今月経過した日数を返します。

次に、この値に利率を乗算して 365 で除算すると、この期間に発生する実効利率が返されます。次に結果にローンの開始残高を掛け、今月これまでに発生した利息を返します。

networkdays

networkdays 関数は、オプションで指定された **holiday** を考慮した上で、**start_date** と **end_date** の間の当日を含む作業日数 (月 ~ 金曜日) を返します。

構文:

```
networkdays (start_date, end_date [, holiday])
```

戻り値データ型: 整数

networkdays 関数が返した日付範囲を表示するカレンダーの図

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10 start_date	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26 end_date	27
28	29	30	31			

networkdays 関数には次の制限事項があります:

- 勤務日を変更する方法はありません。つまり、月 ~ 金曜日以外の日付が関与する地域または状況のために関数を変更する方法はないということです。
- holiday パラメータは文字列定数である必要があります。数式は使用できません。

引数

引数	説明
start_date	評価する開始日。
end_date	評価する終了日。
holiday	作業日から除外する休日期間。休日は文字列定数の日付として示されます。コンマで区切り、複数の休日を設定できます。 **'12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014'**

使用に適しているケース

networkdays() 関数は、計算で 2 つの日付の間に週の労働日数を使用する場合に、数式の一部としてよく使われます例えば、PAYE (源泉課税) 契約の従業員が得る合計賃金を計算するような場合です。

関数の例

例	結果
networkdays ('12/19/2013', '01/07/2014')	14 を返します。この例では、休日を考慮に入れていません。
networkdays ('12/19/2013', '01/07/2014', '12/25/2013', '12/26/2013')	12 を返します。12/25/2013 から 12/26/2013 までの休日を考慮に入れています。
networkdays ('12/19/2013', '01/07/2014', '12/25/2013', '12/26/2013', '12/31/2013', '01/01/2014')	10 を返します。この例では、2 日間の休日期間を考慮に入れています。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- プロジェクトID、開始日付、終了日付を含むデータセット。この情報は、「Projects」というテーブルにロードされます。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- 各プロジェクトに関与する勤務日数を計算する追加項目 [net_work_days] の作成。

ロードスクリプト

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

Projects:
    Load
        *,
        networkdays(start_date,end_date) as net_work_days
        ;
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- end_date
- net_work_days

結果テーブル

ID	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	13

休日が予定されていないため (これは networkdays() 関数の第 3 引数に入っているはずである)、関数は start_date を end_date、そしてすべての週末から差し引いて、2 つの日付の間の勤務日数を計算します。

プロジェクト5 の勤務日が強調表示されたカレンダーの図 (休日なし)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

上記のカレンダーは、id が 5 のプロジェクトを視覚的に概説しています。プロジェクト5 は 2022 年 8 月 10 日 (水) に開始され、2022 年 8 月 26 日に終了します。土日はすべて無視されるため、これら2 つの日付を含むその間には 13 日あります。

例 2 – 単一の休日

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 前の例と同じデータセットとシナリオ。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- 各プロジェクトに関与する勤務日数を計算する追加項目 [net_work_days] の作成。

この例では、2022 年 8 月 19 日に休日が 1 日予定されています。

ロードスクリプト

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

Projects:
    Load
        *,
        networkdays(start_date,end_date,'08/19/2022') as net_work_days
        ;
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- end_date
- net_work_days

結果テーブル

ID	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	12

単一のスケジュールされた休日は、networkdays() 関数に第 3 引数として入力されます。

プロジェクト5 の勤務日が強調表示されたカレンダーの図 (休日 1 日)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

上記のカレンダーはプロジェクト5 を視覚的に概説しており、休日を含むこの調整を示しています。この休日は、プロジェクト5 の期間、2022 年 8 月 19 日 (金) に発生します。その結果、プロジェクト5 で合計 net_work_days の値が 13 日から12 日に 1 日減ります。

例 3 – 複数の休日

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- 各プロジェクトに関与する勤務日数を計算する追加項目 [net_work_days] の作成。

ただし、この例では、2022 年 8 月 18 日 ~ 8 月 21 日に 4 日間の休日が予定されています。

ロード スクリプト

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

Projects:
    Load
        *,
        networkdays(start_date,end_date,'08/18/2022','08/19/2022','08/20/2022','08/21/2022')
as net_work_days
        ;
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- end_date
- net_work_days

結果テーブル

ID	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	11

予定されている4日間は、networkdays() 関数の第 3 引数からカンマ区切りのリストとして入力されます。

プロジェクト5 の勤務日が強調表示されたカレンダーの図 (複数の休日)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18 Holiday	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

上記のカレンダーはプロジェクト5 を視覚的に概説しており、これらの休日を含むこの調整を示しています。予定された休日のこの期間は、プロジェクト5 の期間中、木曜日と金曜日の 2 日間に発生します。その結果、プロジェクト5 で合計 net_work_days の値が 13 日から11 日に減ります。

例 4 – 単一の休日

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。

2022 年 8 月 19 日に休日が1 日予定されています。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。net_work_days 項目は、チャートオブジェクトのメジャーとして計算されます。

ロード スクリプト

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

Projects:
Load
id,
start_date,
end_date
Inline
[
id,start_date,end_date
1,01/01/2022,01/18/2022
2,02/10/2022,02/17/2022
3,05/17/2022,07/05/2022
4,06/01/2022,06/12/2022
5,08/10/2022,08/26/2022
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- start_date
- end_date

次のメジャーを作成します:

```
= networkdays(start_date,end_date,'08/19/2022')
```

結果テーブル

ID	start_date	end_date	net_work_days
1	01/01/2022	01/18/2022	12
2	02/10/2022	02/17/2022	6
3	05/17/2022	07/05/2022	36
4	06/01/2022	06/12/2022	8
5	08/10/2022	08/26/2022	12

単一のスケジュールされた休日は、networkdays() 関数に第 3 引数として入力されます。

単一の休日で正味勤務日が表示されているカレンダーの図 (チャートオブジェクト)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19 Holiday	20
21	22	23	24	25	26	27
28	29	30	31			

上記のカレンダーはプロジェクト5を視覚的に概説しており、休日を含むこの調整を示しています。この休日は、プロジェクト5の期間、2022年8月19日 (金) に発生します。その結果、プロジェクト5で合計 net_work_days の値が13日から12日に1日減ります。

now

この関数は、現在の時刻のタイムスタンプを返します。この関数は、**TimeStamp** システム変数形式の値を返します。既定の **timer_mode** 値は1です。

構文:

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

戻り値データ型: dual

now() 関数は、ロードスクリプトまたはチャートオブジェクトのいずれかで使用できます。

引数

引数	説明
timer_mode	以下の値を取ることができます。 0 (最後にデータロードが終了した時刻) 1 (関数を呼び出した時刻) 2 (アプリを開いた時刻) *データロードスクリプトでこの関数を使用する場合、**timer_mode=0** を指定すると最後にデータロードが終了した時刻を取得でき、**timer_mode=1** を指定すると現在のデータロードで関数を呼び出した時刻を取得できます。*

now() 関数はパフォーマンスに大きな影響を与えるため、テーブルの数式でその関数を使用するとスクロールの問題が発生する可能性があります。その関数を使用しなくても問題がない場合は、代わりに today() 関数を使用することを推奨します。レイアウトで now() の使用が必要な場合は、定期的な再計算が必要ないため、可能な限り既定の設定以外の now(0) または now(2) を使用することを推奨します

使用に適しているケース

now() 関数は、数式内のコンポーネントとしてよく使用されます。例えば、製品のライフサイクルで残った時間を計算するのに使用できます。now() 関数は、数式に1日の端数を使用する場合に today() 関数の代わりに使用されます。

次のテーブルは、timer_mode 引数に異なる値を与えた場合に、now() 関数が返す結果についての説明を提供しています。

関数の例

timer_mode value	ロードスクリプトで使用された場合の結果	チャートオブジェクトで使用された場合の結果
0	TimeStamp システム変数形式で、最新のデータリロードの前に成功した前回のデータリロードのタイムスタンプを返します。	TimeStamp システム変数形式で、最新のデータリロードのタイムスタンプを返します。
1	TimeStamp システム変数形式で、最新のデータリロードのタイムスタンプを返します。	TimeStamp システム変数形式で、関数呼び出しのタイムスタンプを返します。
2	TimeStamp システム変数形式で、アプリケーションでユーザーのセッションが開始されたときのタイムスタンプを返します。これは、ユーザーがスクリプトをリロードしない限り構成されます。	TimeStamp システム変数形式で、アプリケーションでユーザーのセッションが開始されたときのタイムスタンプを返します。これは、新しいセッションが開始されたり、アプリケーションのデータがリロードされたりすると、更新されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – ロードスクリプトを使用したオブジェクトの生成

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

この例では、now() 関数を使用して 3 つの変数を作成しています。各変数は、timer_mode オプションの 1 つを使って効果を示します。

変数が目的を示すためには、スクリプトをリロードしてからしばらくして、2 回目のスクリプトのリロードを行います。これにより、now(0) と now(1) 変数で異なる値が表示されるため、目的が正しく示されます。

ロードスクリプト

```
LET vPreviousDataLoad = now(0);
LET vCurrentDataLoad = now(1);
LET vApplicationOpened = now(2);
```

結果

データが 2 回目にロードされたら、次の手順を使用して 3 つのテキストボックスを作成します。

最初に、以前にロードされたデータのテキストボックスを作成します。

次の手順を実行します。

1. [**テキストと画像**] チャートオブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーをオブジェクトに追加します。
   `=vPreviousDataLoad`
3. [**スタイル**] で **Show titles** を選択し、オブジェクトに「前回のリロード時刻」というタイトルを追加します。

次に、現在ロードしているデータのテキストボックスを作成します。

次の手順を実行します。

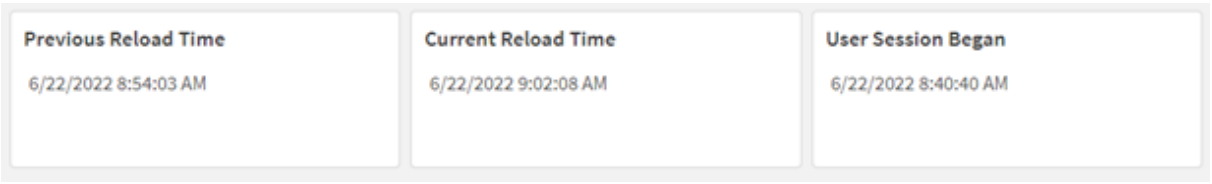
1. [**テキストと画像**] チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーをオブジェクトに追加します。
   `=vCurrentDataLoad`
3. [**スタイル**] で **Show titles** を選択し、オブジェクトに「現在のリロード時刻」というタイトルを追加します。

アプリケーションでユーザーのセッションがいつ開始されたかを示す最終的なテキストボックスを作成します。

次の手順を実行します。

1. [**テキストと画像**] チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーをオブジェクトに追加します。
   `=vApplicationOpened`
3. [**スタイル**] で **Show titles** を選択し、オブジェクトに「ユーザー セッション開始」というタイトルを追加します。

now() ロード スクリプト変数



上記の図は、作成された変数それぞれの値の例を示しています。例えば、次のような値が考えられます。

- 前回のリロード時刻: 6/22/2022 8:54:03 AM
- 現在のリロード時刻: 6/22/2022 9:02:08 AM
- ユーザー セッション開始: 6/22/2022 8:40:40 AM

例 2 – ロード スクリプトを使用しないオブジェクトの生成

ロード スクリプトとチャートの数式

概要

この例では、アプリケーションに変数もデータもロードせずに、`now()` 関数を使用して 3 つのチャート オブジェクトを作成します。各チャート オブジェクトは、`timer_mode` オプションの 1 つを使って効果を示します。

この例にロード スクリプトはありません。

次の手順を実行します。

1. データ ロード エディタを開きます。
2. 既存のロード スクリプトを変更せずに、[**データのロード**] をクリックします。
3. 少し待ってから、2 回目のスクリプトのロードを行います。

結果

データが 2 回目にロードされたら、3 つのテキストボックスを作成します。

まず、最新のデータリロードのテキストボックスを作成します。

次の手順を実行します。

1. [**テキストと画像**] チャートオブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーを追加します。
   `=now(0)`
3. [**スタイル**] で [**タイトルを表示**] を選択し、オブジェクトに「最新のデータリロード」というタイトルを追加します。

次に、現行時刻を示すテキストボックスを作成します。

次の手順を実行します。

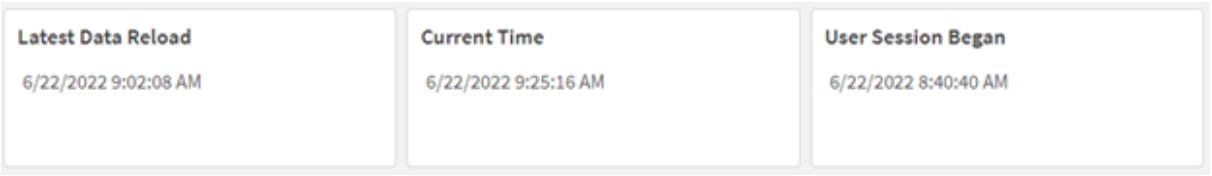
1. [**テキストと画像**] チャートオブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーを追加します。
   `=now(1)`
3. [**スタイル**] で [**タイトルを表示**] を選択し、オブジェクトに「現在の時刻」というタイトルを追加します。

アプリケーションでユーザーのセッションがいつ開始されたかを示す最終的なテキストボックスを作成します。

次の手順を実行します。

1. [**テキストと画像**] チャートオブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーを追加します。
   `=now(2)`
3. [**スタイル**] で [**タイトルを表示**] を選択し、オブジェクトに「ユーザーセッションを開始」というタイトルを追加します。

now() チャートオブジェクトの例



上記の図は、作成されたオブジェクトそれぞれの値の例を示しています。例えば、次のような値が考えられます。

- 最新のデータリロード: 6/22/2022 9:02:08 AM
- 現在の時刻: 6/22/2022 9:25:16 AM
- ユーザーセッション開始: 6/22/2022 8:40:40 AM

「最新のデータリロード」チャートオブジェクトは `timer_mode` 値 0 を使用します。これにより、データのリロードが前回成功したときのタイムスタンプが返されます。

「現在の時刻」チャート オブジェクトは timer_mode 値 1 を使用します。これにより、システム時計に従って現在の時刻が返されます。シートまたはオブジェクトが更新された場合、この値は更新されます。

「ユーザー セッションを開始」チャート オブジェクトは timer_mode 値 2 を使用します。これにより、アプリケーションが開かれ、ユーザーのセッションが開始されたときのタイムスタンプが返されます。

例 3 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 暗号通貨マイニング操作のインベントリで構成されたデータセットで、Inventory というテーブルにロードされます。
- 次の項目を持つデータ: id、purchase_date、および wph (時間当たりのワット数)。

ユーザーは、各マイニング リグが消費電力で今月今までに費やした総費用を、id 別に表示するテーブルを求めています。

この値は、チャート オブジェクトが更新されるたびに更新されます。現在の電気の費用は $0.0678/kWH です。

ロード スクリプト

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

Inventory:
Load
*
Inline
[
id,purchase_date,wph
8188,1/7/2022,1123
8189,1/19/2022,1432
8190,2/28/2022,1227
8191,2/5/2022,1322
8192,3/16/2022,1273
8193,4/1/2022,1123
8194,5/7/2022,1342
8195,5/16/2022,2342
8196,6/15/2022,1231
8197,6/26/2022,1231
8198,7/9/2022,1123
8199,7/22/2022,1212
8200,7/23/2022,1223
8201,7/27/2022,1232
8202,8/2/2022,1232
8203,8/8/2022,1211
8204,8/19/2022,1243
8205,9/26/2022,1322
```

```
8206,10/14/2022,1133
8207,10/29/2022,1231
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:id。

次のメジャーを作成します:

```
=(now(1)-monthstart(now(1)))*24*wph/1000*0.0678
```

チャートオブジェクトが 6/22/2022 10:39:05 AM に更新された場合、次の結果を返します。

結果テーブル

ID	=(now(1)-monthstart(now(1)))*24*wph/1000*0.0678
8188	$39.18
8189	$49.97
8190	$42.81
8191	$46.13
8192	$44.42
8193	$39.18
8194	$46.83
8195	$81.72
8196	$42.95
8197	$42.95
8198	$39.18
8199	$42.29
8200	$42.67
8201	$42.99
8202	$42.99
8203	$42.25
8204	$43.37
8205	$46.13
8206	$39.53

ユーザーは、オブジェクトが更新されるたびにオブジェクト結果を更新したいと思っています。そのため、数式で now() 関数のインスタンスに対して timer_mode 引数が提供されます。月の始めのタイムスタンプは、now() 関数を monthstart() 関数のタイムスタンプ引数として使用することで特定されるのですが、now() 関数によって特定される現在の時刻から差し引かれます。これにより、今月今までに経過した合計時間 (日) がわかります。

この値は 24 (1 日の時間数) で、次に wph 項目の値によって乗算されます。

ワット数/時からキロワット数/時に変換するには、提供された kWH 率で最後に乗算する前に、結果を 1000 で除算します。

quarterend

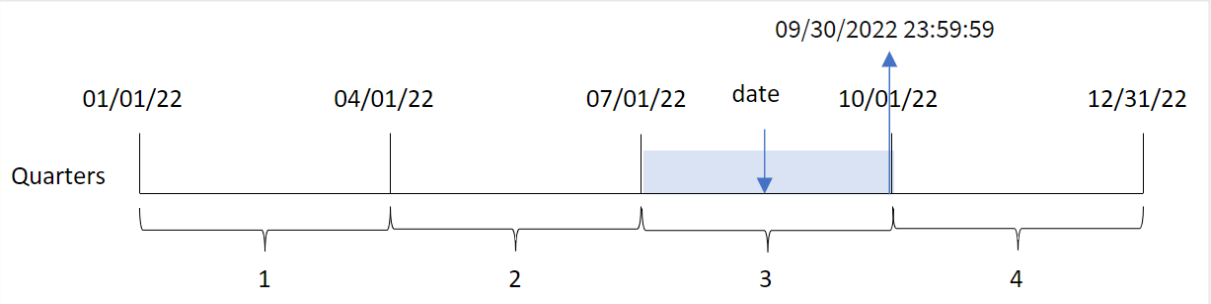
この関数は、**date** を含む四半期の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている **DateFormat** です。

構文:

```
QuarterEnd(date[, period_no[, first_month_of_year]])
```

戻り値データ型: dual

quarterend() 関数の図



quarterend() 関数は、日付がどの四半期に該当するかを判断します。次に、その四半期の最後の月の最後のミリ秒のタイムスタンプを日付形式で返します。年の最初の月は、既定では 1 月です。ただし、quarterend() 関数で first_month_of_year 引数を使用して、どの月を最初に設定するかを変更することができます。

quarterend() 関数は FirstMonthOfYear システム変数を考慮しません。first_month_of_year 引数を使用して変更しない限り、年は 1 月 1 日から始まります。

使用に適しているケース

quarterend() 関数は、ユーザーがまだ発生していない四半期の端数を計算に使用する場合に、数式の一部としてよく使用されます。たとえば、その四半期にまだ発生していない利息の合計を計算したい場合などに使います。

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数で、値 0 は **date** を含む四半期を示します。**period_no** の値が負の場合は過去の四半期を、正の場合は将来の四半期を示します。
first_month_of_year	事業年度が1月以外の月に始まる場合は、**first_month_of_year** で 2 から12 の間の値を指定します。

次の値を使用して、first_month_of_year 引数に年の最初の月を設定できます。

first_month_of_year values

月	値
February	2
3 月	3
April	4
May	5
June	6
7 月	7
8 月	8
September	9
10 月	10
November	11
12 月	12

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
quarterend('10/29/2005')	12/31/2005 23:59:59 を返します。
quarterend('10/29/2005', -1)	09/30/2005 23:59:59 を返します。
quarterend('10/29/2005', 0, 3)	11/30/2005 23:59:59 を返します。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Transactions」というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- 次を含む、先行する LOAD:
  - [end_of_quarter] 項目として設定され、トランザクションが発生する四半期の終わりのタイムスタンプを返す quarterend() 関数。
  - [end_of_quarter_timestamp] 項目として設定され、選択した四半期の終わりのタイムスタンプを返す timestamp() 関数。

ロードスクリプト

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

Transactions:
    Load
       *,
       quarterend(date) as end_of_quarter,
       timestamp(quarterend(date)) as end_of_quarter_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- end_of_quarter
- end_of_quarter_timestamp

結果テーブル

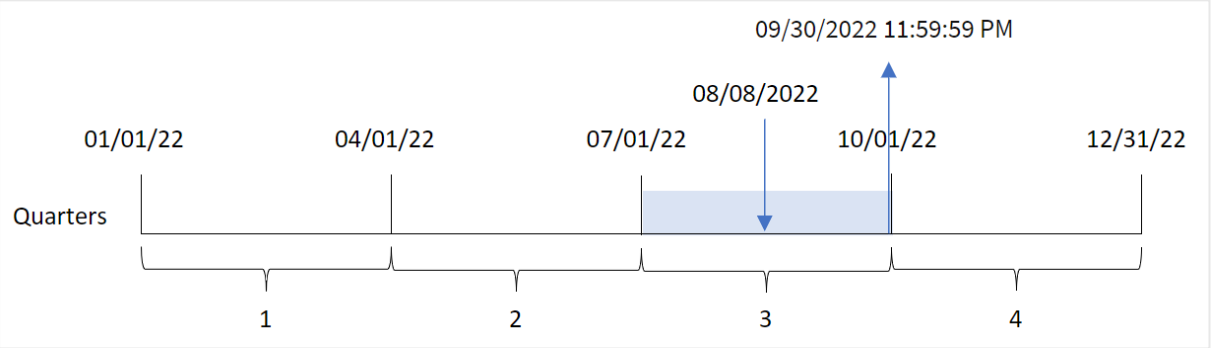
ID	日付	end_of_quarter	end_of_quarter_timestamp
8188	1/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8189	1/19/2022	03/31/2022	3/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	06/30/2022	6/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/16/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	09/30/2022	9/30/2022 11:59:59 PM
8199	7/22/2022	09/30/2022	9/30/2022 11:59:59 PM
8200	7/23/2022	09/30/2022	9/30/2022 11:59:59 PM
8201	7/27/2022	09/30/2022	9/30/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM

ID	日付	end_of_quarter	end_of_quarter_timestamp
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	10/29/2022	12/31/2022	12/31/2022 11:59:59 PM

[end_of_quarter] 項目は、quarterend() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

quarterend() 関数は、最初に日付値がどの四半期に該当するかを識別し、次にその四半期の最後のミリ秒のタイムスタンプを返します。

トランザクション 8203 の四半期末が特定された quarterend() 関数の図



トランザクション 8203 は 8 月 8 日に発生しました。quarterend() 関数は、トランザクションが第 3 四半期に発生したことを特定し、その四半期の最後のミリ秒である 9 月 30 日 11:59:59 PM を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- 次を含む、先行する LOAD:
  - [previous_quarter_end] 項目として設定され、トランザクションが発生する前の四半期の終わりのタイムスタンプを返す quarterend() 関数。
  - [previous_end_of_quarter_timestamp] 項目として設定され、トランザクションが発生する前の四半期の終わりの正確なタイムスタンプを返す timestamp() 関数。

ロード スクリプト

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

Transactions:
    Load
       *,
       quarterend(date, -1) as previous_quarter_end,
       timestamp(quarterend(date, -1)) as previous_quarter_end_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- previous_quarter_end
- previous_quarter_end_timestamp

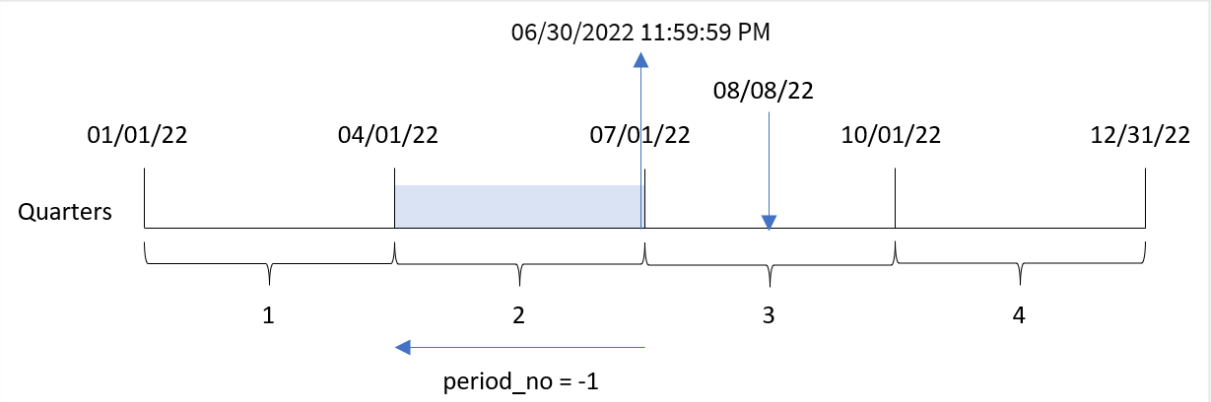
結果テーブル

ID	日付	previous_quarter_end	previous_quarter_end_timestamp
8188	1/7/2022	12/31/2021	12/31/2021 11:59:59 PM

ID	日付	previous_quarter_end	previous_quarter_end_timestamp
8189	1/19/2022	12/31/2021	12/31/2021 11:59:59 PM
8190	2/5/2022	12/31/2021	12/31/2021 11:59:59 PM
8191	2/28/2022	12/31/2021	12/31/2021 11:59:59 PM
8192	3/16/2022	12/31/2021	12/31/2021 11:59:59 PM
8193	4/1/2022	03/31/2022	3/31/2022 11:59:59 PM
8194	5/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8195	5/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8196	6/15/2022	03/31/2022	3/31/2022 11:59:59 PM
8197	6/26/2022	03/31/2022	3/31/2022 11:59:59 PM
8198	7/9/2022	06/30/2022	6/30/2022 11:59:59 PM
8199	7/22/2022	06/30/2022	6/30/2022 11:59:59 PM
8200	7/23/2022	06/30/2022	6/30/2022 11:59:59 PM
8201	7/27/2022	06/30/2022	6/30/2022 11:59:59 PM
8202	8/2/2022	06/30/2022	6/30/2022 11:59:59 PM
8203	8/8/2022	06/30/2022	6/30/2022 11:59:59 PM
8204	8/19/2022	06/30/2022	6/30/2022 11:59:59 PM
8205	9/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8206	10/14/2022	09/30/2022	9/30/2022 11:59:59 PM
8207	10/29/2022	09/30/2022	9/30/2022 11:59:59 PM

-1 の period_no が quarterend() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生する四半期を識別します。次に、1 四半期前にずらして、その四半期の最後のミリ秒を識別します。

period_no が -1 の quarterend() 関数の図



トランザクション 8203 は 8 月 8 日に発生しました。quarterend() 関数は、トランザクション発生前の四半期は 4 月 1 日 ~ 6 月 30 日に発生したことを特定しています。次に関数は、その四半期の最後のミリ秒である 6 月 30 日 11:59:59 PM を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- 次を含む、先行する LOAD:
    - [end_of_quarter] 項目として設定され、トランザクションが発生する四半期の終わりのタイムスタンプを返す quarterend() 関数。
    - [end_of_quarter_timestamp] 項目として設定され、選択した四半期の終わりのタイムスタンプを返す timestamp() 関数。

ただし、この例では、会社ポリシーで会計年度が 3 月 1 日に発生することが定められています。

ロードスクリプト

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

Transactions:
    Load
       *,
       quarterend(date, 0, 3) as end_of_quarter,
       timestamp(quarterend(date, 0, 3)) as end_of_quarter_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
```

```
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

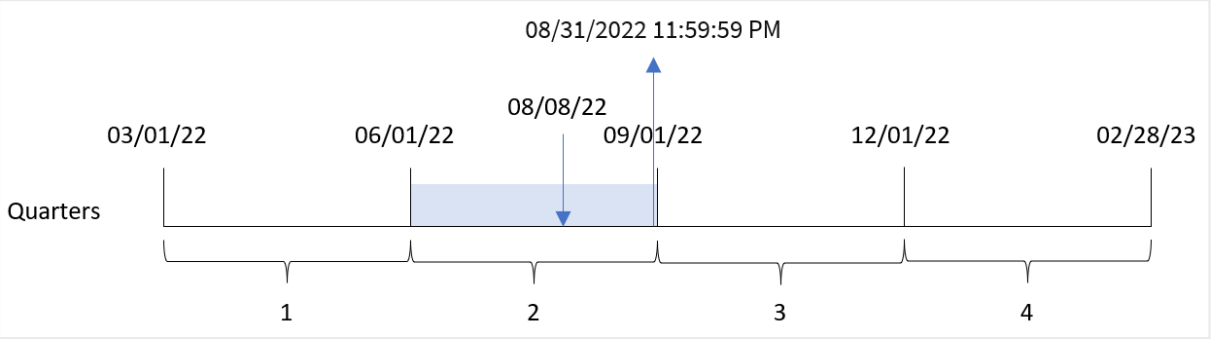
結果

結果テーブル

ID	日付	end_of_quarter	end_of_quarter_timestamp
8188	1/7/2022	02/28/2022	2/28/2022 11:59:59 PM
8189	1/19/2022	02/28/2022	2/28/2022 11:59:59 PM
8190	2/5/2022	02/28/2022	2/28/2022 11:59:59 PM
8191	2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
8192	3/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8193	4/1/2022	05/31/2022	5/31/2022 11:59:59 PM
8194	5/7/2022	05/31/2022	5/31/2022 11:59:59 PM
8195	5/16/2022	05/31/2022	5/31/2022 11:59:59 PM
8196	6/15/2022	08/31/2022	8/31/2022 11:59:59 PM
8197	6/26/2022	08/31/2022	8/31/2022 11:59:59 PM
8198	7/9/2022	08/31/2022	8/31/2022 11:59:59 PM
8199	7/22/2022	08/31/2022	8/31/2022 11:59:59 PM
8200	7/23/2022	08/31/2022	8/31/2022 11:59:59 PM
8201	7/27/2022	08/31/2022	8/31/2022 11:59:59 PM
8202	8/2/2022	08/31/2022	8/31/2022 11:59:59 PM
8203	8/8/2022	08/31/2022	8/31/2022 11:59:59 PM
8204	8/19/2022	08/31/2022	8/31/2022 11:59:59 PM
8205	9/26/2022	11/30/2022	11/30/2022 11:59:59 PM
8206	10/14/2022	11/30/2022	11/30/2022 11:59:59 PM
8207	10/29/2022	11/30/2022	11/30/2022 11:59:59 PM

first_month_of_year 引数である 3 が quarterend() 関数で使用されるため、年度の始めが 1 月 1 日から 3 月 1 日に移動します。

3 月が年の最初の月に設定された quarterend() 関数の図



トランザクション 8203 は 8 月 8 日に発生しました。年度の始まりは 3 月 1 日なので、年度の四半期は 3 ~ 5 月、6 ~ 8 月、9 ~ 11 月、12 ~ 2 月の間に発生します。

quarterend() 関数は、トランザクションが 6 月始めと 8 月終わりの間の四半期に発生したことを特定し、その四半期の最後のミリ秒である 8 月 31 日 11:59:59 PM を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例では、データセットは変更されず、アプリケーションにロードされます。トランザクションが発生した四半期の終わりのタイムスタンプを返す計算は、アプリのチャートのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
```

```
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date

トランザクションが発生する四半期の終わりを計算するには、次のメジャーを作成します。

- =quarterend(date)
- =timestamp(quarterend(date))

結果テーブル

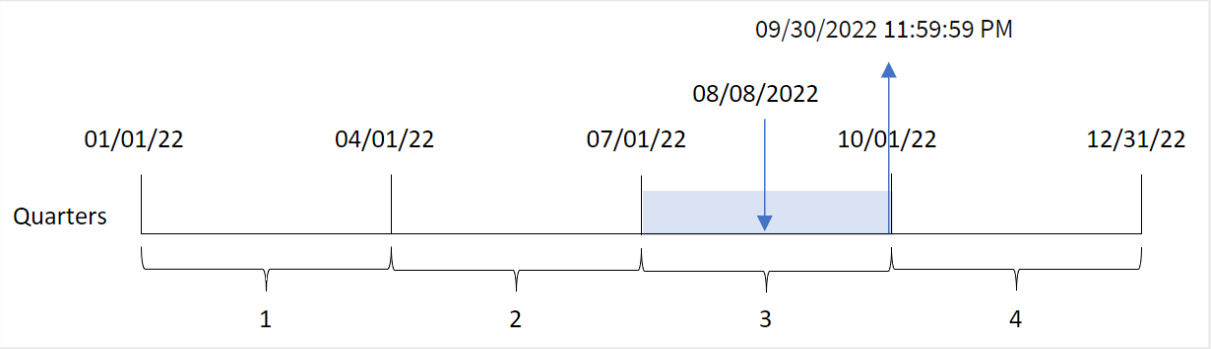
ID	日付	=quarterend(date)	=timestamp(quarterend(date))
8188	1/7/2022	03/31/2022	3/31/2022 11:59:59 PM
8189	1/19/2022	03/31/2022	3/31/2022 11:59:59 PM
8190	2/5/2022	03/31/2022	3/31/2022 11:59:59 PM
8191	2/28/2022	03/31/2022	3/31/2022 11:59:59 PM
8192	3/16/2022	03/31/2022	3/31/2022 11:59:59 PM
8193	4/1/2022	06/30/2022	6/30/2022 11:59:59 PM
8194	5/7/2022	06/30/2022	6/30/2022 11:59:59 PM
8195	5/16/2022	06/30/2022	6/30/2022 11:59:59 PM
8196	6/15/2022	06/30/2022	6/30/2022 11:59:59 PM
8197	6/26/2022	06/30/2022	6/30/2022 11:59:59 PM
8198	7/9/2022	09/30/2022	9/30/2022 11:59:59 PM
8199	7/22/2022	09/30/2022	9/30/2022 11:59:59 PM
8200	7/23/2022	09/30/2022	9/30/2022 11:59:59 PM
8201	7/27/2022	09/30/2022	9/30/2022 11:59:59 PM
8202	8/2/2022	09/30/2022	9/30/2022 11:59:59 PM
8203	8/8/2022	09/30/2022	9/30/2022 11:59:59 PM
8204	8/19/2022	09/30/2022	9/30/2022 11:59:59 PM

ID	日付	=quarterend(date)	=timestamp(quarterend(date))
8205	9/26/2022	09/30/2022	9/30/2022 11:59:59 PM
8206	10/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	10/29/2022	12/31/2022	12/31/2022 11:59:59 PM

[end_of_quarter] 項目は、quarterend() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

quarterend() 関数は、最初に日付値がどの四半期に該当するかを識別し、次にその四半期の最後のミリ秒のタイムスタンプを返します。

トランザクション 8203 の四半期末が特定された quarterend() 関数の図



トランザクション 8203 は 8 月 8 日に発生しました。quarterend() 関数は、トランザクションが第 3 四半期に発生したことを特定し、その四半期の最後のミリ秒である 9 月 30 日 11:59:59 PM を返します。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Employee_Expenses」というテーブルにロードされるデータセット。テーブルには次の項目が含まれています。
  - 従業員 ID
  - 従業員名
  - 各従業員の平均日次経費請求。

エンドユーザーは、従業員 ID と従業員名別に、その四半期の残りの期間にまだ発生する推定経費請求を表示するグラフオブジェクトを求めています。会計年度は 1 月に始まります。

ロード スクリプト

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- employee_id
- employee_name

累積利息を計算するには、次のメジャーを作成します。

- =(quarterend(today(1))-today(1))*avg_daily_claim

メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

employee_id	employee_name	=(quarterend(today(1))-today(1))*avg_daily_claim
182	Mark	$480.00
183	Deryck	$400.00
184	Dexter	$400.00
185	Sydney	$864.00
186	Agatha	$576.00

quarterend() 関数は、今日の日付を唯一の引数として使用す、現在の月の終了日を返します。次に、年の終了日から今日の日付を引き、数式が今月の残りの日数を返します。

次に、この値に各従業員による1日あたりの平均経費請求額を乗算して、四半期の残り期間に各従業員が行うと予想される請求の推定額を計算します。

quartername

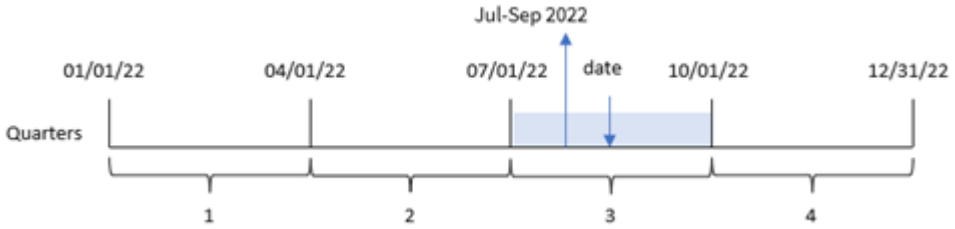
この関数は、四半期の初日の最初のミリ秒のタイムスタンプに対応する値を基底として、四半期の月数 (**MonthNames** スクリプト変数に従った書式) および年の表示値を返します。

構文:

```
QuarterName(date[, period_no[, first_month_of_year]])
```

戻り値データ型: dual

quartername() 関数の図



quartername() 関数は、日付がどの四半期に該当するかを判断します。次に、この四半期と年の開始と終了月を示す値を返します。この結果の基礎となる数値は、四半期の最初のミリ秒です。

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数で、値 0 は **date** を含む四半期を示します。**period_no** の値が負の場合は過去の四半期を、正の場合は将来の四半期を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

使用に適しているケース

quartername() 関数は、集計を四半期単位で比較する場合に便利です。たとえば、製品の総売上高を四半期ごとに表示する場合などが考えられます。

この関数は、マスター カレンダー テーブルに項目を作成することにより、ロード スクリプトで作成できます。あるいは、計算軸としてチャートで直接使用することもできます。

これらの例は、日付書式 DD/MM/YYYY を使用しています。日付書式は、データ ロード スクリプト上部の SET DateFormat ステートメントで指定されています。必要に応じて、書式を変更してください。

関数の例

例	結果
quartername('10/29/2013')	Oct-Dec 2013 を返します。
quartername('10/29/2013', -1)	Jul-Sep 2013 を返します。
quartername('10/29/2013', 0, 3)	Sep-Nov 2013 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数がない日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生した四半期を返す項目 [transaction_quarter] の作成。

必要に応じて、リストなどで他のテキストをここに追加します。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        quartername(date) as transaction_quarter
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
```

```
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- transaction_quarter

結果テーブル

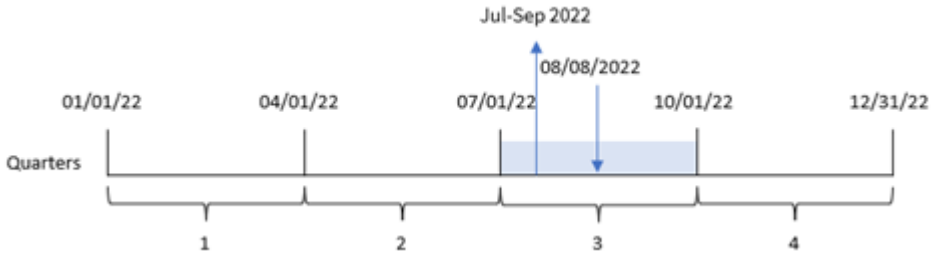
日付	transaction_quarter
1/7/2022	2022 年 1 月 ~ 3 月
1/19/2022	2022 年 1 月 ~ 3 月
2/5/2022	2022 年 1 月 ~ 3 月
2/28/2022	2022 年 1 月 ~ 3 月
3/16/2022	2022 年 1 月 ~ 3 月
4/1/2022	2022 年 4 月 ~ 6 月
5/7/2022	2022 年 4 月 ~ 6 月
5/16/2022	2022 年 4 月 ~ 6 月
6/15/2022	2022 年 4 月 ~ 6 月
6/26/2022	2022 年 4 月 ~ 6 月
7/9/2022	2022 年 7 月 ~ 9 月
7/22/2022	2022 年 7 月 ~ 9 月
7/23/2022	2022 年 7 月 ~ 9 月
7/27/2022	2022 年 7 月 ~ 9 月
8/2/2022	2022 年 7 月 ~ 9 月
8/8/2022	2022 年 7 月 ~ 9 月

日付	transaction_quarter
8/19/2022	2022 年 7 月 ~ 9 月
9/26/2022	2022 年 7 月 ~ 9 月
10/14/2022	2022 年 10 月 ~ 12 月
10/29/2022	2022 年 10 月 ~ 12 月

transaction_quarter 項目は、quartername() 関数を使用し、関数の引数として日付項目を渡すことにより、先行する LOAD ステートメントで作成されます。

quartername() 関数は始め、日付値が入っている四半期を特定します。次に、この四半期と年の開始と終了月を示す値を返します。

quartername() 関数の図、追加の引数がない例



トランザクション 8203 は 2020 年 8 月 8 日に発生しました。quartername() 関数は、トランザクションが第 3 四半期に発生したことを特定し、そのため 2022 年の 7 ~ 9 月を返します。月は、MonthNames システム変数と同じ形式で表示されます。

例 2 – period_no 引数を持つ日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の四半期を返す項目 [previous_quarter] の作成。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
```

```
        quartername(date,-1) as previous_quarter
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_quarter

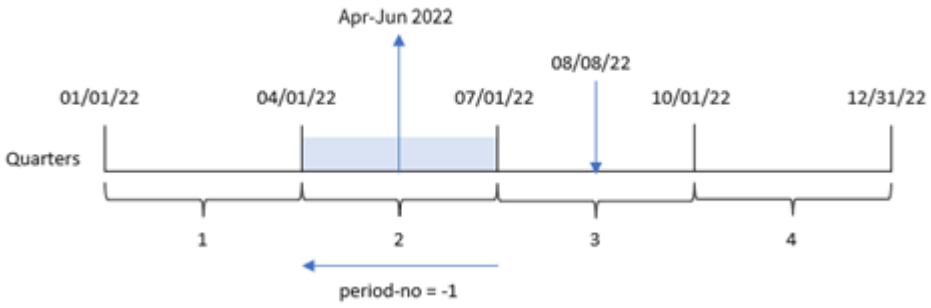
結果テーブル

日付	previous_quarter
1/7/2022	Oct-Dec 2021
1/19/2022	Oct-Dec 2021
2/5/2022	Oct-Dec 2021
2/28/2022	Oct-Dec 2021
3/16/2022	Oct-Dec 2021
4/1/2022	2022 年 1 月 ~ 3 月
5/7/2022	2022 年 1 月 ~ 3 月
5/16/2022	2022 年 1 月 ~ 3 月

日付	previous_quarter
6/15/2022	2022 年 1 月 ~ 3 月
6/26/2022	2022 年 1 月 ~ 3 月
7/9/2022	2022 年 4 月 ~ 6 月
7/22/2022	2022 年 4 月 ~ 6 月
7/23/2022	2022 年 4 月 ~ 6 月
7/27/2022	2022 年 4 月 ~ 6 月
8/2/2022	2022 年 4 月 ~ 6 月
8/8/2022	2022 年 4 月 ~ 6 月
8/19/2022	2022 年 4 月 ~ 6 月
9/26/2022	2022 年 4 月 ~ 6 月
10/14/2022	2022 年 7 月 ~ 9 月
10/29/2022	2022 年 7 月 ~ 9 月

この例では、-1 の period_no が quarternname() 関数でオフセット引数として使用されたため、関数はトランザクションが第 3 四半期に発生したことを識別します。次に、1 つ前の四半期に戻って、この四半期と年の開始と終了月を示す値を返します。

quarternname() 関数の図、period_no の例



トランザクション 8203 は 8 月 8 日に発生しました。quarternname() 関数は、トランザクション発生前の四半期は 4 月 1 日 ~ 6 月 30 日に発生したことを特定しています。そのため、Apr-Jun 2022 を返します。

例 3 – first_week_day 引数を持つ日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では会計年度の始めを3月1日に設定する必要があります。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';

Transactions:
    Load
        *,
        quartername(date,0,3) as transaction_quarter
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- transaction_quarter

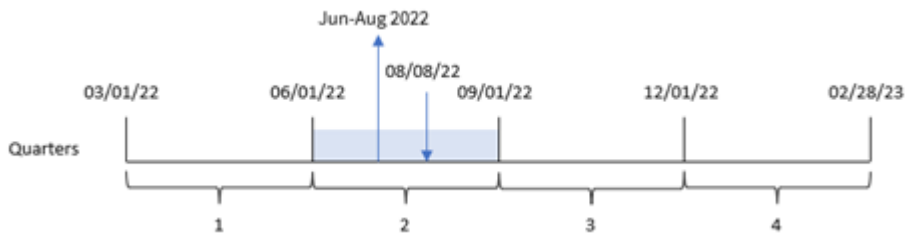
結果テーブル

日付	transaction_quarter
1/7/2022	Dec-Feb 2021

日付	transaction_quarter
1/19/2022	Dec-Feb 2021
2/5/2022	Dec-Feb 2021
2/28/2022	Dec-Feb 2021
3/16/2022	Mar-May 2022
4/1/2022	Mar-May 2022
5/7/2022	Mar-May 2022
5/16/2022	Mar-May 2022
6/15/2022	Jun-Aug 2022
6/26/2022	Jun-Aug 2022
7/9/2022	Jun-Aug 2022
7/22/2022	Jun-Aug 2022
7/23/2022	Jun-Aug 2022
7/27/2022	Jun-Aug 2022
8/2/2022	Jun-Aug 2022
8/8/2022	Jun-Aug 2022
8/19/2022	Jun-Aug 2022
9/26/2022	Sep-Nov 2022
10/14/2022	Sep-Nov 2022
10/29/2022	Sep-Nov 2022

このインスタンスでは、first_month_of_year 引数 3 が quartername() 関数で使用されているため、年度の始めが 1 月 1 日から 3 月 1 日に移動します。そのため、その年の四半期は 3 ~ 5 月、6 ~ 8 月、9 ~ 11 月、12 ~ 2 月に分けられます。

quartername() 関数、first_week_day 例の図



トランザクション 8203 は 8 月 8 日に発生しました。quartername() 関数は、トランザクションが 6 月の始め ~ 8 月の終わりの第 2 四半期に発生したことを特定しています。そのため、Jun-Aug 2022 を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した四半期の終わりのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。

次のメジャーを作成します:

```
=quartername(date)
```

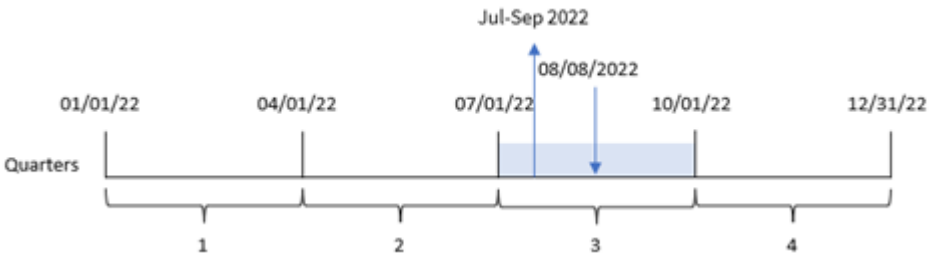
結果テーブル

日付	=quartername(date)
1/7/2022	2022 年 1 月 ~ 3 月
1/19/2022	2022 年 1 月 ~ 3 月
2/5/2022	2022 年 1 月 ~ 3 月
2/28/2022	2022 年 1 月 ~ 3 月
3/16/2022	2022 年 1 月 ~ 3 月
4/1/2022	2022 年 4 月 ~ 6 月
5/7/2022	2022 年 4 月 ~ 6 月
5/16/2022	2022 年 4 月 ~ 6 月
6/15/2022	2022 年 4 月 ~ 6 月
6/26/2022	2022 年 4 月 ~ 6 月
7/9/2022	2022 年 7 月 ~ 9 月
7/22/2022	2022 年 7 月 ~ 9 月
7/23/2022	2022 年 7 月 ~ 9 月
7/27/2022	2022 年 7 月 ~ 9 月
8/2/2022	2022 年 7 月 ~ 9 月
8/8/2022	2022 年 7 月 ~ 9 月
8/19/2022	2022 年 7 月 ~ 9 月
9/26/2022	2022 年 7 月 ~ 9 月
10/14/2022	2022 年 10 月 ~ 12 月
10/29/2022	2022 年 10 月 ~ 12 月

[transaction_quarter] メジャーは、quartername() 関数を使用し、関数の引数として [date] 項目を渡すことにより、チャートオブジェクトで作成されます。

quartername() 関数は始め、日付値が入っている四半期を特定します。次に、この四半期と年の開始と終了月を示す値を返します。

quartername() 関数の図、チャート オブジェクトの例



トランザクション 8203 は 2020 年 8 月 8 日に発生しました。quartername() 関数は、トランザクションが第 3 四半期に発生したことを特定し、そのため 2022 年の 7 ~ 9 月を返します。月は、MonthNames システム変数と同じ形式で表示されます。

例 5 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。

エンド ユーザーは、トランザクションの四半期ごとの総売上高を示すチャート オブジェクトを求めています。これは、チャートの計算軸として quartername() 関数を使用して、この軸がデータ モデルで使用できない場合でも実現できます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'1/7/2022',17.17
8189,'1/19/2022',37.23
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'5/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
```

```
8197,'6/26/2022',45.89
8198,'7/9/2022',36.23
8199,'7/22/2022',25.66
8200,'7/23/2022',82.77
8201,'7/27/2022',69.98
8202,'8/2/2022',76.11
8203,'8/8/2022',25.12
8204,'8/19/2022',46.23
8205,'9/26/2022',84.21
8206,'10/14/2022',96.24
8207,'10/29/2022',67.67
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の式を使用して計算軸を作成します。
   `=quartername(date)`
3. 次に、以下の集計メジャーを使って総売上を計算します。
   `=sum(amount)`
4. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

=quartername(date)	=sum(amount)
2022 年 7 月 ~ 9 月	$446.31
2022 年 4 月 ~ 6 月	$351.48
2022 年 1 月 ~ 3 月	$253.89
2022 年 10 月 ~ 12 月	$163.91

quarterstart

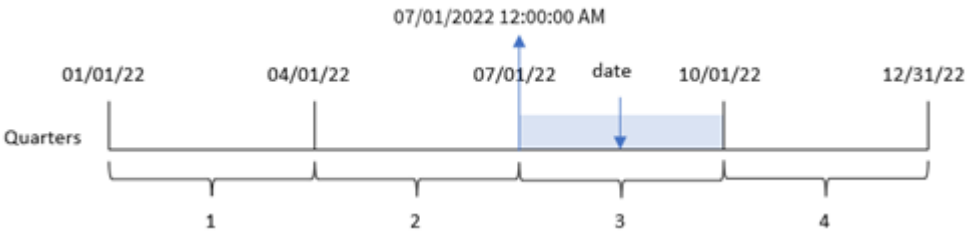
この関数は、**date** を含む四半期の最初のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている **DateFormat** です。

構文:

```
QuarterStart(date[, period_no[, first_month_of_year]])
```

戻り値データ型： dual

quarterstart() 関数の図



quarterstart() 関数は、date がどの四半期に該当するかを判断します。次に、その四半期の最初の月の最初のミリ秒のタイムスタンプを日付形式で返します。

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数で、値 0 は **date** を含む四半期を示します。**period_no** の値が負の場合は過去の四半期を、正の場合は将来の四半期を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。

使用に適しているケース

quarterstart() 関数は、ユーザーがこれまで経過した四半期の端数を計算に使用する場合に、数式の一部として一般的に使用されます。例えば、ユーザーが四半期の特定の日付までに累積した利息を計算したい場合などに使用できます。

関数の例

例	結果
`quarterstart('10/29/2005')`	10/01/2005 を返します。
`quarterstart('10/29/2005', -1 )`	07/01/2005 を返します。
`quarterstart('10/29/2005', 0, 3)`	09/01/2005 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの `SET DateFormat` ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する四半期の始めのタイムスタンプを返す、項目 [start_of_quarter] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        quarterstart(date) as start_of_quarter,
        timestamp(quarterstart(date)) as start_of_quarter_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
```

```
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

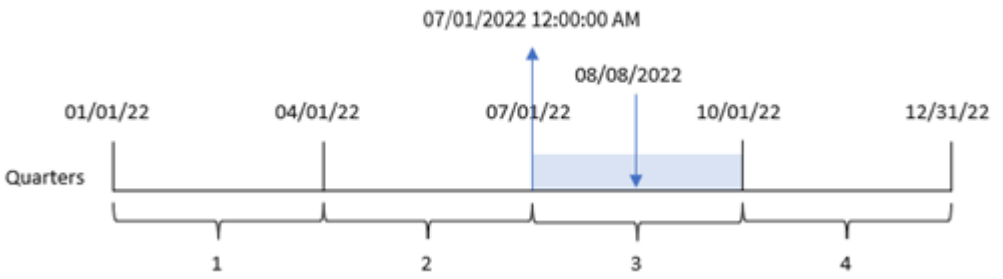
- date
- start_of_quarter
- start_of_quarter_timestamp

結果テーブル

日付	start_of_quarter	start_of_quarter_timestamp
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	01/01/2022	1/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2021 12:00:00 AM
5/7/2022	04/01/2022	4/1/2021 12:00:00 AM
5/16/2022	04/01/2022	4/1/2021 12:00:00 AM
6/15/2022	04/01/2022	4/1/2021 12:00:00 AM
6/26/2022	04/01/2022	4/1/2021 12:00:00 AM
7/9/2022	07/01/2022	7/1/2021 12:00:00 AM
7/22/2022	07/01/2022	7/1/2021 12:00:00 AM
7/23/2022	07/01/2022	7/1/2021 12:00:00 AM
7/27/2022	07/01/2022	7/1/2021 12:00:00 AM
8/2/2022	07/01/2022	7/1/2021 12:00:00 AM
8/8/2022	07/01/2022	7/1/2021 12:00:00 AM
8/19/2022	07/01/2022	7/1/2021 12:00:00 AM
9/26/2022	07/01/2022	7/1/2021 12:00:00 AM
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM

start_of_quarter 項目は、quarterstart() 関数を使用し、関数の引数として日付項目を渡すことにより、先行するLOAD ステートメントで作成されます。uarterstart() 関数はまず、日付値が入っている四半期を特定します。次に、その四半期の最初のミリ秒のタイムスタンプを返します。

quarterstart() 関数の図、追加の引数がない例



トランザクション 8203 は 8 月 8 日に発生しました。quarterstart() 関数は、トランザクションが第 3 四半期に発生したことを特定し、その四半期の最初のミリ秒である 7 月 1 日 12:00:00 AM を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の四半期の始めのタイムスタンプを返す、項目 [previous_quarter_start] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        quarterstart(date,-1) as previous_quarter_start,
        timestamp(quarterstart(date,-1)) as previous_quarter_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
```

```
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_quarter_start
- previous_quarter_start_timestamp

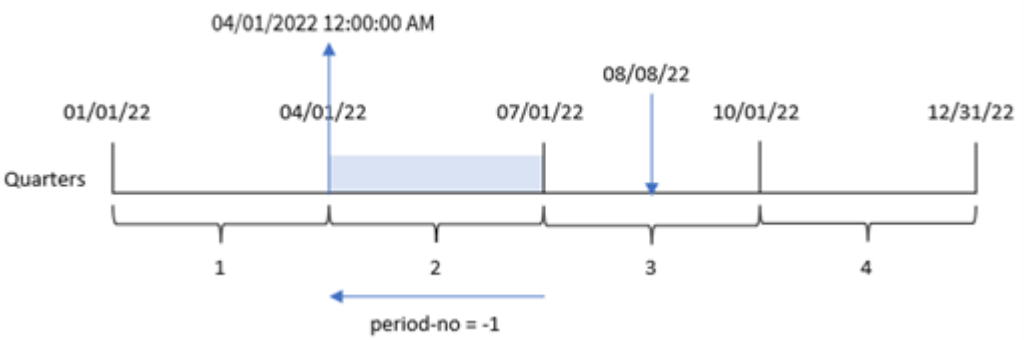
結果テーブル

日付	previous_quarter_start	previous_quarter_start_timestamp
1/7/2022	10/01/2021	10/1/2021 12:00:00 AM
1/19/2022	10/01/2021	10/1/2021 12:00:00 AM
2/5/2022	10/01/2021	10/1/2021 12:00:00 AM
2/28/2022	10/01/2021	10/1/2021 12:00:00 AM
3/16/2022	10/01/2021	10/1/2021 12:00:00 AM
4/1/2022	01/01/2022	1/1/2022 12:00:00 AM
5/7/2022	01/01/2022	1/1/2022 12:00:00 AM
5/16/2022	01/01/2022	1/1/2022 12:00:00 AM
6/15/2022	01/01/2022	1/1/2022 12:00:00 AM
6/26/2022	01/01/2022	1/1/2022 12:00:00 AM
7/9/2022	04/01/2022	4/1/2021 12:00:00 AM
7/22/2022	04/01/2022	4/1/2021 12:00:00 AM
7/23/2022	04/01/2022	4/1/2021 12:00:00 AM
7/27/2022	04/01/2022	4/1/2021 12:00:00 AM

日付	previous_quarter_start	previous_quarter_start_timestamp
8/2/2022	04/01/2022	4/1/2021 12:00:00 AM
8/8/2022	04/01/2022	4/1/2021 12:00:00 AM
8/19/2022	04/01/2022	4/1/2021 12:00:00 AM
9/26/2022	04/01/2022	4/1/2021 12:00:00 AM
10/14/2022	07/01/2022	7/1/2022 12:00:00 AM
10/29/2022	07/01/2022	7/1/2022 12:00:00 AM

この例では、-1 の period_no が quarterstart() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した四半期を識別します。次に、1 四半期前にずらして、その四半期の最初のミリ秒を識別します。

quarterstart() 関数の図、period_no の例



トランザクション 8203 は 8 月 8 日に発生しました。quarterstart() 関数は、トランザクション発生前の四半期は 4 月 1 日 ~ 6 月 30 日に発生したことを特定しています。次に、その四半期の最初のミリ秒、4 月 1 日 12:00:00 AM を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では会計年度の始めを 3 月 1 日に設定する必要があります。

ロードスクリプト

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

Transactions:
    Load
```

```
        *,
        quarterstart(date,0,3) as start_of_quarter,
        timestamp(quarterstart(date,0,3)) as start_of_quarter_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- start_of_quarter
- start_of_quarter_timestamp

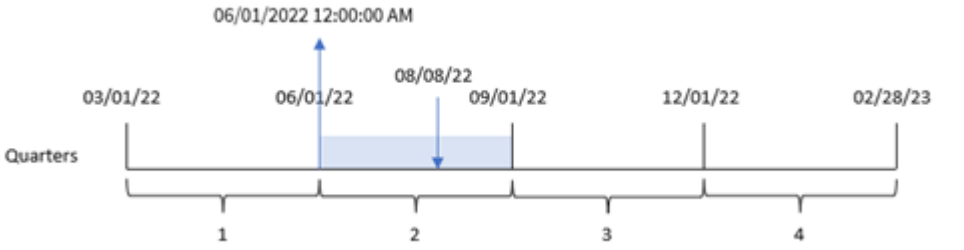
結果テーブル

日付	start_of_quarter	start_of_quarter_timestamp
1/7/2022	12/01/2021	12/1/2021 12:00:00 AM
1/19/2022	12/01/2021	12/1/2021 12:00:00 AM
2/5/2022	12/01/2021	12/1/2021 12:00:00 AM
2/28/2022	12/01/2021	12/1/2021 12:00:00 AM
3/16/2022	03/01/2022	3/1/2022 12:00:00 AM
4/1/2022	03/01/2022	3/1/2022 12:00:00 AM

日付	start_of_quarter	start_of_quarter_timestamp
5/7/2022	03/01/2022	3/1/2022 12:00:00 AM
5/16/2022	03/01/2022	3/1/2022 12:00:00 AM
6/15/2022	06/01/2022	6/1/2022 12:00:00 AM
6/26/2022	06/01/2022	6/1/2022 12:00:00 AM
7/9/2022	06/01/2022	6/1/2022 12:00:00 AM
7/22/2022	06/01/2022	6/1/2022 12:00:00 AM
7/23/2022	06/01/2022	6/1/2022 12:00:00 AM
7/27/2022	06/01/2022	6/1/2022 12:00:00 AM
8/2/2022	06/01/2022	6/1/2022 12:00:00 AM
8/8/2022	06/01/2022	6/1/2022 12:00:00 AM
8/19/2022	06/01/2022	6/1/2022 12:00:00 AM
9/26/2022	09/01/2022	9/1/2022 12:00:00 AM
10/14/2022	09/01/2022	9/1/2022 12:00:00 AM
10/29/2022	09/01/2022	9/1/2022 12:00:00 AM

このインスタンスでは、 `first_month_of_year` 引数である 3 が `quarterstart()` 関数で使用されるため、年度の始めが 1 月 1 日から 3 月 1 日に移動します。

`quarterstart()` 関数の図、first_month_of_year の例



トランザクション 8203 は 8 月 8 日に発生しました。年度の始まりは 3 月 1 日なので、年度の四半期は 3 ~ 5 月、6 ~ 8 月、9 ~ 11 月、12 ~ 2 月の間に発生します。`quarterstart()` 関数は、トランザクションが 6 月始めと 8 月終わりの間の四半期に発生したことを特定し、その四半期の最初のミリ秒である 6 月 1 日 12:00:00 AM を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した四半期の終わりのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。

次のメジャーを追加します。

- `=quarterstart(date)`
- `=timestamp(quarterstart(date))`

結果テーブル

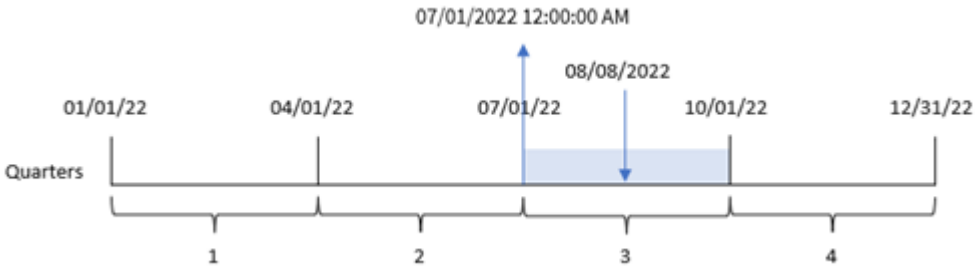
日付	=quarterstart(date)	=timestamp(quarterstart(date))
10/14/2022	10/01/2022	10/1/2022 12:00:00 AM
10/29/2022	10/01/2022	10/1/2022 12:00:00 AM
7/9/2022	07/01/2022	7/1/2022 12:00:00 AM

日付	=quarterstart(date)	=timestamp(quarterstart(date))
7/22/2022	07/01/2022	7/1/2022 12:00:00 AM
7/23/2022	07/01/2022	7/1/2022 12:00:00 AM
7/27/2022	07/01/2022	7/1/2022 12:00:00 AM
8/2/2022	07/01/2022	7/1/2022 12:00:00 AM
8/8/2022	07/01/2022	7/1/2022 12:00:00 AM
8/19/2022	07/01/2022	7/1/2022 12:00:00 AM
9/26/2022	07/01/2022	7/1/2022 12:00:00 AM
4/1/2022	04/01/2022	4/1/2022 12:00:00 AM
5/7/2022	04/01/2022	4/1/2022 12:00:00 AM
5/16/2022	04/01/2022	4/1/2022 12:00:00 AM
6/15/2022	04/01/2022	4/1/2022 12:00:00 AM
6/26/2022	04/01/2022	4/1/2022 12:00:00 AM
1/7/2022	01/01/2022	1/1/2022 12:00:00 AM
1/19/2022	01/01/2022	1/1/2022 12:00:00 AM
2/5/2022	01/01/2022	1/1/2022 12:00:00 AM
2/28/2022	01/01/2022	1/1/2022 12:00:00 AM
3/16/2022	01/01/2022	1/1/2022 12:00:00 AM

[start_of_quarter] メジャーは、quarterstart() 関数を使用し、関数の引数として [date] 項目を渡すことにより、チャートオブジェクトで作成されます。

quarterstart() 関数は、日付値がどの四半期に該当するかを識別し、その四半期の最初のミリ秒のタイムスタンプを返します。

quarterstart() 関数の図、チャートオブジェクトの例



トランザクション 8203 は 8 月 8 日に発生しました。quarterstart() 関数は、トランザクションが第 3 四半期に発生したことを特定し、その四半期の最初のミリ秒を返します。この返された値は、7 月 1 日 12:00:00 AM です。

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Loans というテーブルにロードされる、一連のローン残高を含むデータセット。
- ローンID、四半期の初めの残高、各ローンにかかる単利の年率で構成されるデータ。

エンドユーザーは、年初来の各ローンで発生した現在の利息をローンID別に表示するチャートオブジェクトを求めています。

ロードスクリプト

```
Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:
   - loan_id
   - start_balance
2. 次に、このメジャーを作成して、累積利息を計算します。
   `=start_balance*(rate*(today(1)-quarterstart(today(1)))/365)`
3. メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

loan_id	start_balance	=start_balance*(rate*(today(1)-quarterstart(today(1)))/365)
8188	$10000.00	$15.07
8189	$15000.00	$128.84

loan_id	start_balance	=start_balance*(rate*(today(1)-quarterstart(today(1)))/365)
8190	$17500.00	$63.29
8191	$21000.00	$107.59
8192	$90000.00	$1139.18

quarterstart() 関数は、今日の日付を唯一の引数として使用することにより、現在の年の開始日を返します。その結果を現在の日付から減算することにより、数式は今年四半期で今まで経過した日数を返します。

次に、この値に利率を乗算して 365 で除算すると、この期間に発生する実効利率が返されます。次に、結果にローンの開始残高を掛けると、今四半期これまでに発生した利息を返されます。

second

この関数は、**expression** の小数部が標準的な数値の解釈に従って時間と判断される場合に、秒を表す整数を返します。

構文:

```
second (expression)
```

戻り値データ型: 整数

使用に適しているケース

second() 関数は、集計を秒単位で比較する場合に便利です。例えば、秒ごとのアクティビティ数分布を確認したい場合は、関数を使用できます。

これらの軸は、関数を使用してマスター カレンダー テーブルに項目を作成することにより、ロード スクリプトで作成することも、計算軸としてチャートで直接使用することもできます。

関数の例

例	結果
second('09:14:36')	36 を返します
second('0.5555')	55 を返します (0.5555 = 13:19:55 のため)

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データ ロード スクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロード スクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データ ロード エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 変数

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされるタイムスタンプによるトランザクションを含むデータセット。
- 既定の TimeStamp システム変数 (M/D/YYYY h:mm:ss[.fff] TT) が使用されます。
- 購入がいつ発生するかを計算する、項目 second の作成。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
    Load
        *,
        second(date) as second
        ;
Load
*
Inline
[
id,date,amount
9497,'01/05/2022 7:04:57 PM',47.25
9498,'01/03/2022 2:21:53 PM',51.75
9499,'01/03/2022 5:40:49 AM',73.53
9500,'01/04/2022 6:49:38 PM',15.35
9501,'01/01/2022 10:10:22 PM',31.43
9502,'01/05/2022 7:34:46 PM',13.24
9503,'01/06/2022 10:58:34 PM',74.34
9504,'01/06/2022 11:29:38 AM',50.00
9505,'01/02/2022 8:35:54 AM',36.34
9506,'01/06/2022 8:49:09 AM',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- second

結果テーブル

日付	秒
01/01/2022 10:10:22 PM	22
01/02/2022 8:35:54 AM	54
01/03/2022 5:40:49 AM	49
01/03/2022 2:21:53 PM	53
01/04/2022 6:49:38 PM	38
01/05/2022 7:04:57 PM	57
01/05/2022 7:34:46 PM	46
01/06/2022 8:49:09 AM	9
01/06/2022 11:29:38 AM	38
01/06/2022 10:58:34 PM	34

second 項目の値は、second() 関数を使用し、先行するLOAD ステートメントの数式として日付を渡すことによって作成されます。

例 2 – チャート オブジェクト

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。「second」値は、チャート オブジェクトのメジャーを介して計算されます。

ロード スクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

Transactions:
Load
*
Inline
[
id,date,amount
9497,'01/05/2022 7:04:57 PM',47.25
9498,'01/03/2022 2:21:53 PM',51.75
9499,'01/03/2022 5:40:49 AM',73.53
9500,'01/04/2022 6:49:38 PM',15.35
9501,'01/01/2022 10:10:22 PM',31.43
9502,'01/05/2022 7:34:46 PM',13.24
9503,'01/06/2022 10:58:34 PM',74.34
```

```
9504,'01/06/2022 11:29:38 AM',50.00
9505,'01/02/2022 8:35:54 AM',36.34
9506,'01/06/2022 8:49:09 AM',74.23
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

次のメジャーを作成します:

```
=second(date)
```

結果テーブル

日付	=second(date)
01/01/2022 10:10:22 PM	22
01/02/2022 8:35:54 AM	54
01/03/2022 5:40:49 AM	49
01/03/2022 2:21:53 PM	53
01/04/2022 6:49:38 PM	38
01/05/2022 7:04:57 PM	57
01/05/2022 7:34:46 PM	46
01/06/2022 8:49:09 AM	9
01/06/2022 11:29:38 AM	38
01/06/2022 10:58:34 PM	34

second の値は、second() 関数を使用し、チャートオブジェクトのメジャーの数式とて日付を渡すことによって作成されます。

例 3 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 特定のフェスティバルのチケット販売ウェブサイトへのトラフィックを示すために生成された、タイムスタンプのデータセット。これらのタイムスタンプと対応する id は、Web_Traffic というテーブルにロードされます。
- TimeStamp システム変数 M/D/YYYY h:mm:ss[.fff] TT が使用されます。

このシナリオでは、チケットが10000枚あり、2021年5月20日9:00 AMに発売されました。1分後には売り切れました。

ユーザーは、ウェブサイトへの訪問数を秒単位で表示するチャートオブジェクトを求めています。

ロードスクリプト

```
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';

tmpTimeStampCreator:
load
    makedate(2022,05,20) as date
AutoGenerate 1;

join load
    maketime(9+floor(rand()*2),0,floor(rand()*59)) as time
autogenerate 10000;

Web_Traffic:
load
    recno() as id,
    timestamp(date + time) as timestamp
resident tmpTimeStampCreator;

drop table tmpTimeStampCreator;
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次に、次の数式を使用して計算軸を作成します。
   `=second(timestamp)`
3. 集計メジャーを作成して、エントリーの合計数を計算します。
   `=count(id)`

結果テーブルは下記のようになりますが、集計メジャーの値は異なります。

結果テーブル

second(timestamp)	=count(id)
0	150
1	184
2	163
3	178
4	179
5	158

second(timestamp)	=count(id)
6	177
7	169
8	149
9	186
10	169
11	179
12	186
13	182
14	180
15	153
16	191
17	203
18	158
19	159
20	163
+ 39 行	

setdateyear

この関数は入力として **timestamp** と **year** を取得し、入力で指定された **year** で **timestamp** を更新します。

構文:

```
setdateyear (timestamp, year)
```

戻り値データ型: dual

引数:

引数

引数	説明
timestamp	標準的な Qlik Sense タイムスタンプ (通常、日付のみ)
year	4 桁の年。

例と結果:
これらの例は、日付書式 **DD/MM/YYYY** を使用しています。日付書式は、データロードスクリプト上部の **SET DateFormat** ステートメントで指定されています。必要に応じて、書式を変更してください。

スクリプトの例

例	結果
`setdateyear ('29/10/2005', 2013)`	'29/10/2013' を返します
`setdateyear ('29/10/2005 04:26:14', 2013)`	'29/10/2013 04:26:14' を返します ビジュアライゼーションのタイムスタンプの時間の部分を表示するには、数字の形式を日付に設定し、時間の値を表示する形式について値を選択する必要があります。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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

];
```

結果テーブルには、元の日付と、年が 2013 に設定された列が含まれています。

結果テーブル

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

この関数は入力として **timestamp** と **month** 、**year** を取得し、入力で指定された **year** と **month** で **timestamp** を更新します。.

構文:

```
SetDateYearMonth (timestamp, year, month)
```

戻り値データ型: dual

引数:

引数

引数	説明
timestamp	標準的な Qlik Sense タイムスタンプ (通常、日付のみ)
year	4 桁の年。
month	1 桁または 2 桁の月。

例と結果:
これらの例は、日付書式 **DD/MM/YYYY** を使用しています。日付書式は、データロードスクリプト上部の **SET DateFormat** ステートメントで指定されています。必要に応じて、書式を変更してください。

スクリプトの例

例	結果
`setdateyearmonth ('29/10/2005', 2013, 3)`	'29/03/2013' を返します
`setdateyearmonth ('29/10/2005 04:26:14', 2013, 3)`	'29/03/2013 04:26:14' を返します ビジュアライゼーションのタイムスタンプの時間の部分を表示するには、数字の形式を日付に設定し、時間の値を表示する形式について値を選択する必要があります。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
SetYearMonth:

Load *,

SetDateYearMonth(testdates, 2013,3) as NewYearMonth

Inline [

testdates

1/11/2012

10/12/2012

2/1/2013

19/5/2013

15/9/2013

11/12/2013

14/5/2014

13/6/2014

7/7/2014

4/8/2014

];
```

結果テーブルには、元の日付と、年が 2013 に設定された列が含まれています。

結果テーブル

testdates	NewYearMonth
1/11/2012	1/3/2013
10/12/2012	10/3/2013
2/1/2012	2/3/2013
19/5/2013	19/3/2013
15/9/2013	15/3/2013
11/12/2013	11/3/2013
14/5/2014	14/3/2013
13/6/2014	13/3/2013
7/7/2014	7/3/2013
4/8/2014	4/3/2013

timezone

この関数を使うと、Qlik エンジンが実行されているコンピュータで定義された通りのタイム ゾーンが返されます。

構文:

```
TimeZone( )
```

戻り値データ型: dual

timezone( )
アプリのメジャーで異なるタイムゾーンを確認する場合、軸で localtime() 関数を使うことができます。

today

この関数は、現在の日付を返します。この関数は、DateFormat システム変数形式の値を返します。

構文:

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

戻り値データ型: dual

today() 関数は、ロード スクリプトまたはチャート オブジェクトのいずれかで使用できます。

既定の timer_mode 値は 1 です。

引数

引数	説明
timer_mode	以下の値を取ることができます。 0 (最後にデータロードが終了した日付) 1 (関数を呼び出した日付) 2 (アプリを開いた日付) *ロードスクリプトでこの関数を使用する場合、**timer_mode=0**を指定すると最後にデータロードが終了した日付を取得でき、**timer_mode=1**を指定すると現在のデータロードの日付を取得できます。*

関数の例

timer_mode value	ロードスクリプトで使用された場合の結果	チャートオブジェクトで使用された場合の結果
0	DateFormat システム変数形式で、最新のデータリロードの前に成功した前回のデータリロードの日付を返します。	DateFormat システム変数形式で、最新のデータリロードの日付を返します。
1	DateFormat システム変数形式で、最新のデータリロードの日付を返します。	DateFormat システム変数形式で、関数呼び出しの日付を返します。
2	DateFormat システム変数形式で、アプリケーションでユーザーのセッションが開始されたときの日付を返します。これは、ユーザーがスクリプトをリロードしない限り構成されます。	DateFormat システム変数形式で、アプリケーションでユーザーのセッションが開始されたときの日付を返します。これは、新しいセッションが開始されたり、アプリケーションのデータがリロードされたりすると、更新されます。

使用に適しているケース

today() 関数は、数式内のコンポーネントとしてよく使用されます。たとえば、月の現在の日付までに累積した利息を計算するのに使用できます。

次のテーブルは、timer_mode 引数に異なる値を与えた場合に、today() 関数が返す結果についての説明を提供しています。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – ロードスクリプトを使用したオブジェクトの生成

ロードスクリプトと結果

概要

次の例では、today() 関数を使用して 3 つの変数を作成しています。各変数は、timer_mode オプションの 1 つを使って効果を示します。

変数が目的を示すためには、スクリプトをリロードしてから 24 時間して、2 回目のスクリプトのリロードを行います。これにより、today(0) と today(1) 変数で異なる値が表示されるため、目的が正しく示されます。

ロードスクリプト

```
LET vPreviousDataLoad = today(0);
LET vCurrentDataLoad = today(1);
LET vApplicationOpened = today(2);
```

結果

データが 2 回目にロードされたら、次の手順を使用して 3 つのテキストボックスを作成します。

最初に、以前にロードされたデータのテキストボックスを作成します。

次の手順を実行します。

1. [**テキストと画像**] チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーをオブジェクトに追加します。
   `=vPreviousDataLoad`
3. [**スタイル**] で **Show titles** を選択し、オブジェクトに「前回のリロード時刻」というタイトルを追加します。

次に、現在ロードしているデータのテキストボックスを作成します。

次の手順を実行します。

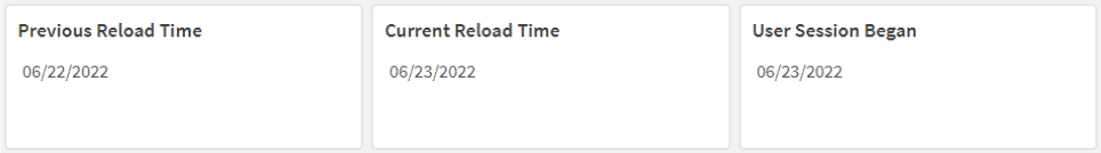
1. [**テキストと画像**] チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーをオブジェクトに追加します。
   `=vCurrentDataLoad`
3. [**スタイル**] で **Show titles** を選択し、オブジェクトに「現在のリロード時刻」というタイトルを追加します。

アプリケーションでユーザーのセッションがいつ開始されたかを示す最終的なテキストボックスを作成します。

次の手順を実行します。

1. **[テキストと画像]** チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーをオブジェクトに追加します。
   `=vApplicationOpened`
3. **[スタイル]** で **Show titles** を選択し、オブジェクトに「ユーザー セッション開始」というタイトルを追加します。

ロード スクリプトで today() 関数を使って作成された変数の図



上記の図は、作成された変数それぞれの値の例を示しています。例えば、次のような値が考えられます。

- 前回のリロード時刻: 06/22/2022
- 現在のリロード時刻: 06/23/2022
- ユーザー セッション開始: 06/23/2022

例 2 – ロード スクリプトを使用しないオブジェクトの生成

ロード スクリプトとチャートの数式

概要

次の例では、today() 関数を使用して 3 つのチャート オブジェクトを作成しています。各チャート オブジェクトは、timer_mode オプションの 1 つを使って効果を示します。

この例にロード スクリプトはありません。

結果

データが 2 回目にロードされたら、3 つのテキストボックスを作成します。

まず、最新のデータ リロードのテキストボックスを作成します。

次の手順を実行します。

1. **[テキストと画像]** チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーを追加します。
   `=today(0)`
3. **[スタイル]** で **[タイトルを表示]** を選択し、オブジェクトに「最新のデータ リロード」というタイトルを追加します。

次に、現行時刻を示すテキストボックスを作成します。

次の手順を実行します。

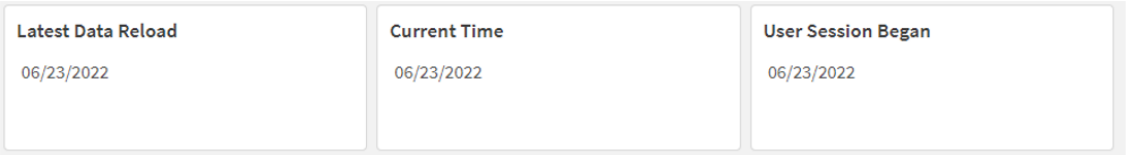
1. **[テキストと画像]** チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーを追加します。
   `=today(1)`
3. **[スタイル]** で **[タイトルを表示]** を選択し、オブジェクトに「現在の時刻」というタイトルを追加します。

アプリケーションでユーザーのセッションがいつ開始されたかを示す最終的なテキストボックスを作成します。

次の手順を実行します。

1. **[テキストと画像]** チャート オブジェクトを使用して、テキストボックスを作成します。
2. 次のメジャーを追加します。
   `=today(2)`
3. **[スタイル]** で **[タイトルを表示]** を選択し、オブジェクトに「ユーザー セッションを開始」というタイトルを追加します。

ロード スクリプトなしで `today()` 関数を使って作成されたオブジェクトの図



上記の図は、作成されたオブジェクトそれぞれの値の例を示しています。例えば、次のような値が考えられます。

- 最新データ リロード: 06/23/2022
- 現在の時刻: 06/23/2022
- ユーザー セッション開始: 06/23/2022

「最新のデータ リロード」チャート オブジェクトは `timer_mode` 値 0 を使用します。これにより、データのリロードが前回成功したときのタイムスタンプが返されます。

「現在の時刻」チャート オブジェクトは `timer_mode` 値 1 を使用します。これにより、システム時計に従って現在の時刻が返されます。シートまたはオブジェクトが更新された場合、この値は更新されます。

「ユーザー セッションを開始」チャート オブジェクトは `timer_mode` 値 2 を使用します。これにより、アプリケーションが開かれ、ユーザーのセッションが開始されたときのタイムスタンプが返されます。

例 3 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Loans というテーブルにロードされる、一連のローン残高を含むデータセット。
- ロードID、月の開始の残高、各ローンにかかる単利の年率の項目を持つテーブル データ。

エンド ユーザーは、年初来の各ローンで発生した現在の利息をローンID別に表示するチャート オブジェクトを求めています。アプリケーションがリロードされるのは週 1 回のみですが、ユーザーはオブジェクトまたはアプリケーションが更新されるたびに結果を更新したいと思っています。

ロード スクリプト

```
Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の項目を軸として追加します。
   - loan_id
   - start_balance
3. 次に、メジャーを作成して、累積利息を計算します。
   `=start_balance*(rate*(today(1)-monthstart(today(1)))/365)`
4. メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8188	$10000.00	$16.44

loan_id	start_balance	=start_balance*(rate*(today(1)-monthstart(today(1)))/365)
8189	$15000.00	$58.56
8190	$17500.00	$28.77
8191	$21000.00	$48.90
8192	$90000.00	$517.81

monthstart() 関数は、today() 関数を使って今日の日付を唯一の引数として使用することにより、現在の月の開始日を返します。もう一度 today() 関数を使ってその結果を現在の日付から減算することにより、数式は今月経過した日数を返します。

次に、この値に利率を乗算して 365 で除算すると、この期間に発生する実効利率が返されます。次に結果にローンの開始残高を掛け、今月これまでに発生した利息を返します。

値 1 が数式内の today() 関数の timer_mode 引数として使用されているため、(アプリケーションを開いて、ページを更新して、シート間を移動することにより) チャートオブジェクトが更新されるたびに、返される日付は現在の日付に対するものであるため、それに従って結果が更新されます。

UTC

現在の Coordinated Universal Time を返します。

構文:

```
UTC( )
```

戻り値データ型: デュアル

```
utc( )
```

week

この関数は、入力された日付に対応する週番号を表す整数を返します。

構文:

```
week(timestamp [, first_week_day [, broken_weeks [, reference_day]]])
```

戻り値データ型: integer

引数

引数	説明
timestamp	評価する日付またはタイムスタンプ。

引数	説明
first_week_day	週の開始日を指定します。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。 **first_week_day** には、0 が月曜日、1 が火曜日、2 が水曜日、3 が木曜日、4 が金曜日、5 が土曜日、6 が日曜日の値を使用できます。 システム変数の詳細については、*FirstWeekDay (page 228)* を参照してください。
broken_weeks	**broken_weeks** が指定されていない場合は、変数 **BrokenWeeks** の値を使用して、週が分離しているかどうかを定義します。
reference_day	**reference_day** が指定されていない場合は、変数 **ReferenceDay** の値を使用して、第 1 週を定義する参照日として設定する 1 月の日を定義します。デフォルトでは、Qlik Sense 関数は 4 を参照日として使用します。これは、第 1 週に必ず 1 月 4 日が含まれる、または第 1 週に少なくとも 1 月の 4 日間が常に含まれることを意味します。

week() 関数は、日付が属する週を判別し、その週の週番号を返します。

Qlik Sense では、アプリ作成時に地域設定がフェッチされ、対応する設定は環境変数としてスクリプトに保管されます。これらは、週番号を決定するために使用されます。

これはつまり、大部分のヨーロッパのアプリ開発者は、ISO 8601 定義に対応する次の環境変数を取得するということです。

```
Set FirstWeekDay =0;    // Monday as first week day
Set BrokenWeeks  =0;    // Use unbroken weeks
Set ReferenceDay =4;    // Jan 4th is always in week 1
```

北米のアプリ開発者は、頻繁に次の環境変数を取得します。

```
Set FirstWeekDay =6;    // Sunday as first week day
Set BrokenWeeks  =1;    // Use broken weeks
Set ReferenceDay =1;    // Jan 1st is always in week 1
```

週の最初の曜日は、FirstWeekDay システム変数によって決定されます。week() 関数で first_week_day 引数を使用して、週の最初の曜日を変更することもできます。

アプリケーションが分割された週を使用する場合、発生日数に関係なく週数のカウントは 1 月 1 日に始まり、FirstWeekDay システム変数の前日に終了します。

アプリケーションが連続した週を使用している場合、第 1 週は前年または 1 月の最初の数日間に開始できます。これは、FirstWeekDay および ReferenceDay 環境変数の使用方法によって異なります。

使用に適しているケース

The week() 関数は、集計を週単位で比較する場合に便利です。例えば、製品の総売上高を週ごとに表示する場合などに使用できます。week() 関数は、ユーザーがアプリケーションの BrokenWeeks、FirstWeekDay、または ReferenceDay システム変数を必ずしも使用せずに計算を行いたい場合に、weekname() よりも優先して選択されます。

たとえば、製品の総売上高を週ごとに表示する場合などが考えられます。

アプリケーションが連続週を使用している場合、第 1 週には前年の12 月の日付が含まれるか、今年の1 月の日付が除外されます。アプリケーションが分割された週を使用している場合、第 1 週 に含まれる日数が 7 日間を下回ることがあります。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

下記の例では

```
Set DateFormat= 'MM/DD/YYYY';
Set FirstWeekDay=0;
Set BrokenWeeks=0;
Set ReferenceDay=4;
```

関数の例

例	結果
week('12/28/2021')	52 を返すとみなされます。
week(44614)	これは、02/22/2022 のシリアル番号であるため、8 を返します。number for .
week('01/03/2021')	53 を返すと推測されます。
week('01/03/2021',6)	1 を返します。

例 1 - 既定システムの変数

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2021 年の最後の週と2022 年の最初の週のトランザクションを含むデータセットは、Transactions というテーブルにロードされます。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する年と週番号を返す、項目 [week_number] の作成。
- 各トランザクション日付の平日値を示している、week_day という項目の作成

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
    Load
        *,
        WeekDay(date) as week_day,
        Week(date) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week_day
- week_number

結果テーブル

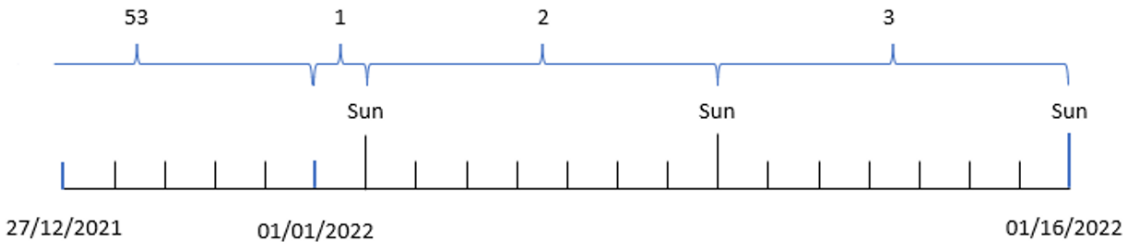
ID	日付	week_day	week_number
8183	12/27/2021	月	53
8184	12/28/2021	火	53
8185	12/29/2021	水	53
8186	12/30/2021	Thu	53
8187	12/31/2021	Fri	53
8188	01/01/2022	Sat	1
8189	01/02/2022	日	2
8190	01/03/2022	月	2
8191	01/04/2022	火	2
8192	01/05/2022	水	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	日	3
8197	01/10/2022	月	3
8198	01/11/2022	火	3
8199	01/12/2022	水	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

week_number 項目は、week() 関数を使用し、関数の引数として date 項目を渡すことにより、先行する LOAD ステートメントで作成されます。

関数に渡される他のパラメータはないため、week() 関数に影響する次の既定変数が有効となります。

- BrokenWeeks: 週のカウントは 1 月 1 日に始まります
- FirstWeekDay: 週の初日は日曜日です

既定のシステム変数を使用した week() 関数の図



アプリケーションは既定の BrokenWeeks システム変数を使用しているため、第 1 週は 1 月 1 日の土曜日に始まります。

既定の FirstWeekDay システム変数のため、週は日曜日に始まります。1 月 1 日以降の最初の日曜日は 1 月 2 日に発生し、この日から第 2 週が始まります。

例 2 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- トランザクションが発生する年と週番号を返す、項目 [week_number] の作成。
- 各トランザクション日付の平日日を示している、week_day という項目の作成。

子の例では、勤務週の初めを火曜日に設定します。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
    Load
        *,
        WeekDay(date) as week_day,
        Week(date,1) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
```

```
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week_day
- week_number

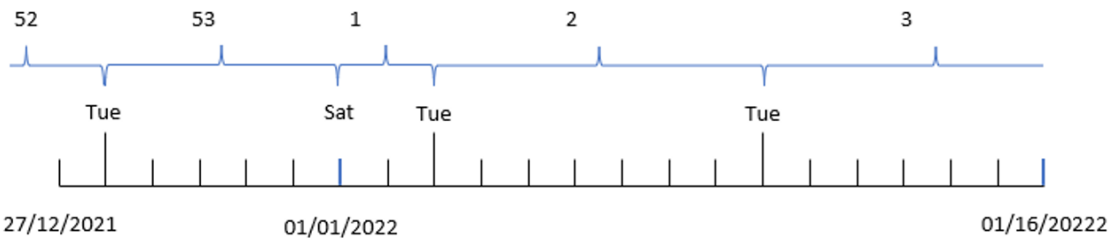
結果テーブル

ID	日付	week_day	week_number
8183	12/27/2021	月	52
8184	12/28/2021	火	53
8185	12/29/2021	水	53
8186	12/30/2021	Thu	53
8187	12/31/2021	Fri	53
8188	01/01/2022	Sat	1
8189	01/02/2022	日	1
8190	01/03/2022	月	1
8191	01/04/2022	火	2
8192	01/05/2022	水	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2

ID	日付	week_day	week_number
8195	01/08/2022	Sat	2
8196	01/09/2022	日	2
8197	01/10/2022	月	2
8198	01/11/2022	火	3
8199	01/12/2022	水	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

アプリケーションはまだ分離された週を使用しています。ただし、first_week_day 引数は week() 関数で 1 に設定されています。これにより、週の最初の日が火曜日に設定されます。

week() 関数、first_week_day 例の図



アプリケーションは既定の BrokenWeeks システム変数を使用しているため、第 1 週は 1 月 1 日の土曜日に始まります。

week() の first_week_day 引数により、週の最初の日が火曜日に設定されます。そのため、第 53 週は 2021 年 12 月 28 日に始まります。

しかし、関数ではそのまま分離週を使用しているため、1 月 1 日以降の最初の火曜日が 1 月 3 日になり、第 1 週は 2 日しかありません。

例 3 – unbroken_weeks

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。

この例では、未分離の週を使用します。

ロード スクリプト

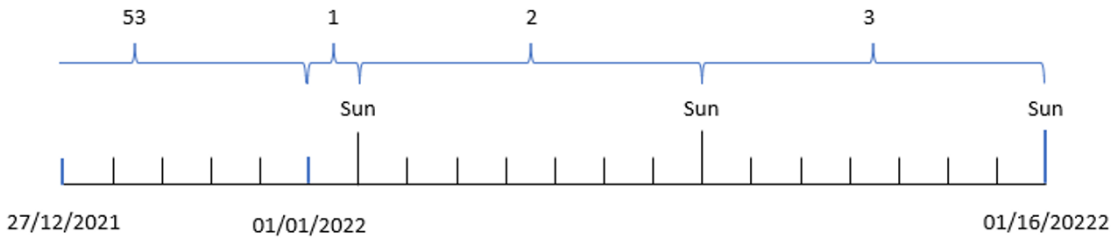
```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transactions:
    Load
        *,
        WeekDay(date) as week_day,
        Week(date,6,0) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week_day
- week_number

week() 関数の図、チャート オブジェクトの例



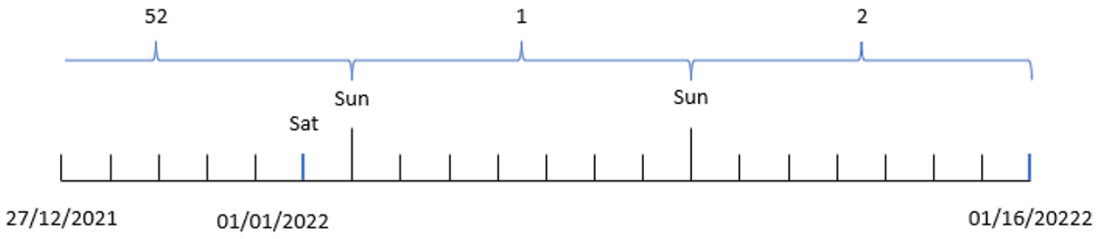
結果テーブル

ID	日付	week_day	week_number
8183	12/27/2021	月	52
8184	12/28/2021	火	52
8185	12/29/2021	水	52
8186	12/30/2021	Thu	52
8187	12/31/2021	Fri	52
8188	01/01/2022	Sat	52
8189	01/02/2022	日	1
8190	01/03/2022	月	1
8191	01/04/2022	火	1
8192	01/05/2022	水	1
8193	01/06/2022	Thu	1
8194	01/07/2022	Fri	1
8195	01/08/2022	Sat	1
8196	01/09/2022	日	2
8197	01/10/2022	月	2
8198	01/11/2022	火	2
8199	01/12/2022	水	2
8200	01/13/2022	Thu	2
8201	01/14/2022	Fri	2

first_week_date パラメータが 1 に設定されており、週の初日が火曜日となります。broken_weeks パラメータが 0 に設定されており、関数が分離されていない週を使うよう強制します。最後に、3 番目のパラメータが reference_day を 2 に設定します。

first_week_date パラメータが 6 に設定されており、週の初日が日曜日となります。broken_weeks パラメータが 0 に設定されており、関数が分離されていない週を使うよう強制します。

week() 関数の図、例では未分離の週を使用



未分離の週を使用することにより、第 1 週が 1 月 1 日に始まるとは限らず、最低 4 日間が必須となります。そのため、データセットでは、第 52 週は 2022 年 1 月 1 日の土曜日に終わります。そうすると第 1 週は FirstWeekDay システム変数、つまり1 月 2 日の日曜日に始まります。この週は次の土曜日である1 月 8 日に終わります。

例 4 – reference_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 例 3 と同じデータセットとシナリオ。
- トランザクションが発生する年と週番号を返す、項目 [week_number] の作成。
- 各トランザクション日付の平日日を示している、week_day という項目の作成。

さらに、次の条件も満たす必要があります。

- 勤務週は火曜日に始まります。
- 会社は未分離の週を使用します。
- reference_day 値は 2 です。つまり、第 1 週の最小日数は 2 となります。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;

Transcactions:
    Load
        *,
        WeekDay(date) as week_day,
```

```
        Week(date,1,0,2) as week_number
        ;
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week_day
- week_number

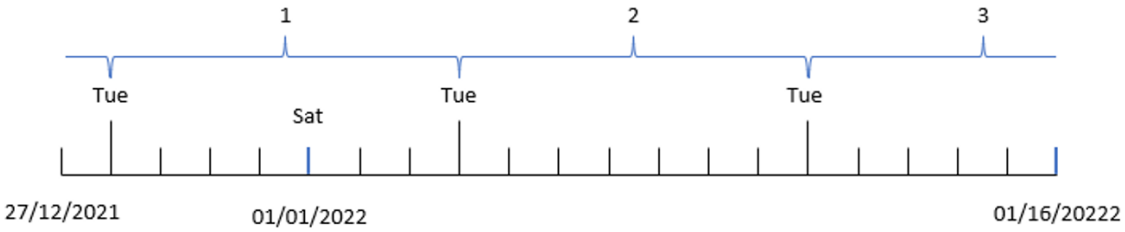
結果テーブル

ID	日付	week_day	week_number
8183	12/27/2021	月	52
8184	12/28/2021	火	1
8185	12/29/2021	水	1
8186	12/30/2021	Thu	1
8187	12/31/2021	Fri	1
8188	01/01/2022	Sat	1
8189	01/02/2022	日	1

ID	日付	week_day	week_number
8190	01/03/2022	月	1
8191	01/04/2022	火	2
8192	01/05/2022	水	2
8193	01/06/2022	Thu	2
8194	01/07/2022	Fri	2
8195	01/08/2022	Sat	2
8196	01/09/2022	日	2
8197	01/10/2022	月	2
8198	01/11/2022	火	3
8199	01/12/2022	水	3
8200	01/13/2022	Thu	3
8201	01/14/2022	Fri	3

first_week_date パラメータが 1 に設定されており、週の初日が火曜日となります。broken_weeks パラメータが 0 に設定されており、関数が分離されていない週を使うよう強制します。最後に、3 番目のパラメータが reference_day を 2 に設定します。

week() 関数、reference_day 例の図



関数が未分離の週を使用し、パラメータとして reference_day 値 2 を使用しているため、第 1 週に含める必要があるのは 1 月の 2 日間のみです。最初の平日が火曜日であるため、第 1 週は 2021 年 12 月 28 日に始まり、2022 年 1 月 3 日に終わります。

例 5 – Chart object example

ロードスクリプトとチャートの数式

概要

データロードエディタを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。週番号を返す計算は、チャートオブジェクトでメジャーとして作成されます。

ロードスクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2022,58.27
8184,12/28/2022,67.42
8185,12/29/2022,23.80
8186,12/30/2022,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の項目を軸として追加します。
   - id
   - date
3. 次に、下記のメジャーを作成します。
   `=week (date)`
4. 各トランザクション日付の平日値を表示するメジャー，week_day を作成します。
   `=weekday(date)`

結果テーブル

ID	日付	=week(date)	=weekday(date)
8183	12/27/2021	53	月
8184	12/28/2021	53	火

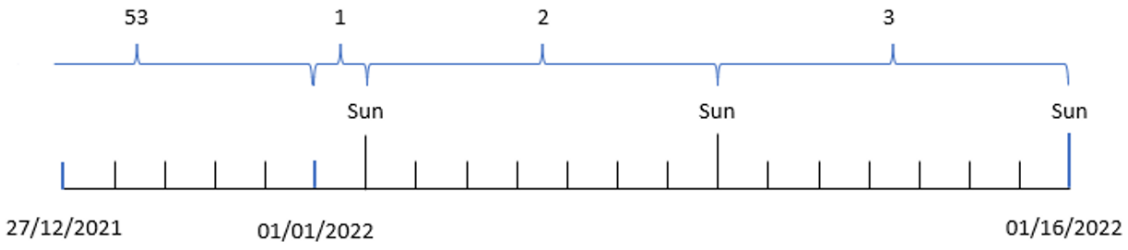
ID	日付	=week(date)	=weekday(date)
8185	12/29/2021	53	水
8186	12/30/2021	53	Thu
8187	12/31/2021	53	Fri
8188	01/01/2022	1	Sat
8189	01/02/2022	2	日
8190	01/03/2022	2	月
8191	01/04/2022	2	火
8192	01/05/2022	2	水
8193	01/06/2022	2	Thu
8194	01/07/2022	2	Fri
8195	01/08/2022	2	Sat
8196	01/09/2022	3	日
8197	01/10/2022	3	月
8198	01/11/2022	3	火
8199	01/12/2022	3	水
8200	01/13/2022	3	Thu
8201	01/14/2022	3	Fri

[week_number] 項目は、week() 関数を使用し、関数の引数として [date] 項目を渡すことにより、前の load ステートメントで作成されます。

関数に渡される他のパラメータはないため、week() 関数に影響する次の既定変数が有効となります。

- BrokenWeeks: 週のカウントは 1 月 1 日に始まります
- FirstWeekDay: 週の初日は日曜日です

week() 関数の図、チャート オブジェクトの例



アプリケーションは既定の BrokenWeeks システム変数を使用しているため、第 1 週は 1 月 1 日の土曜日に始まります。

既定の FirstWeekDay システム変数のため、週は日曜日に始まります。1 月 1 日以降の最初の日曜日は 1 月 2 日に発生し、この日から第 2 週が始まります。

例 6 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2019 年の最後の週と2020 年の最初の週のトランザクションを含むデータセットは、Transactions というテーブルにロードされます。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。

アプリケーションは、ダッシュボード全体で未分離の週を主に使用します。ただし、エンドユーザーは、未分離の週を使用して週ごとの総売上高を示すチャートオブジェクトを求めています。週が火曜日に始まるため、参照日は 1 月 2 日である必要があります。これは、チャートの計算軸として week() 関数を使用して、この軸がデータモデルで使用できない場合でも実現できます。

ロードスクリプト

```
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2019,58.27
8184,12/28/2019,67.42
8185,12/29/2019,23.80
8186,12/30/2019,82.06
8187,12/31/2019,40.56
8188,01/01/2020,37.23
8189,01/02/2020,17.17
8190,01/03/2020,88.27
8191,01/04/2020,57.42
8192,01/05/2020,53.80
8193,01/06/2020,82.06
8194,01/07/2020,40.56
8195,01/08/2020,53.67
8196,01/09/2020,26.63
8197,01/10/2020,72.48
```

```
8198,01/11/2020,18.37
8199,01/12/2020,45.26
8200,01/13/2020,58.23
8201,01/14/2020,18.52
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成します。
2. 次の計算軸を作成します。
   `=week(date)`
3. 次に、以下の集計メジャーを作成します。
   `=sum(amount)`
4. メジャーの [**数値書式**] を [**通貨**] に設定します。
5. [**ソート**] メニューを選択し、計算軸に対して、カスタム ソートを削除します。
6. [**数値でソート**] と [**アルファベット順でソート**] オプションの選択を解除します。

結果テーブル

week(date)	sum(amount)
52	$125.69
53	$146.42
1	$200.09
2	$347.57
3	$122.01

weekday

この関数は、以下を持つデュアル値を返します。

- 環境変数 **DayNames** で定義される日の名前。
- 曜日に相当する0 から6 までの整数。

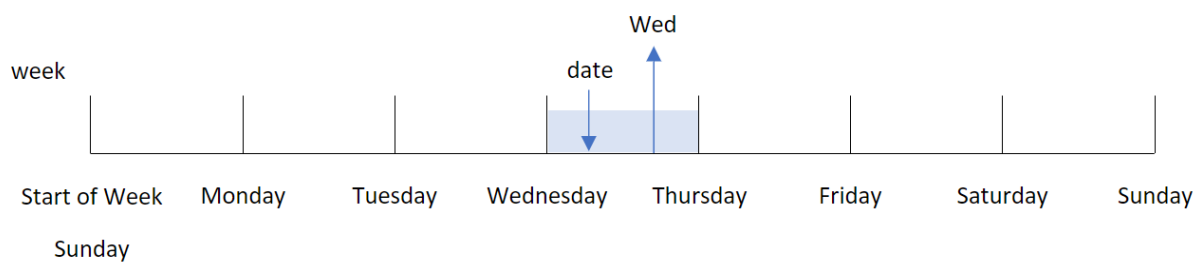
構文:

```
weekday(date [,first_week_day=0])
```

戻り値データ型: dual

weekday() 関数は、日付がどの曜日に当たるかを決定します。次に、その日を示す文字列値を返します。

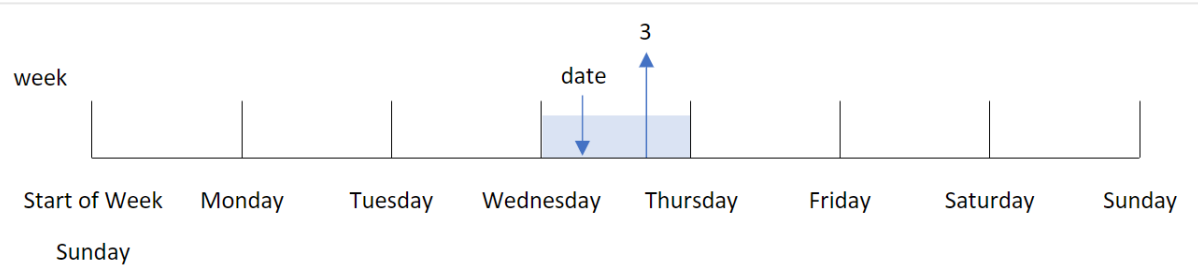
日付が当たる曜日の名前を返す weekday() 関数の図



結果は、週の開始日に基づいて、曜日 (0-6) に対応する数値を返します。例えば、週の初日が日曜日に設定されている場合、水曜日は 3 という数値を返します。この開始日は、FirstWeekDay システム変数、または first_week_day 関数パラメータによって決定されます。

この数値は、算術式の一部として使用できます。たとえば、1 を乗算すると、値そのものが返されます。

日の名前でな く数値が表示されている weekday() 関数の図



使用に適しているケース

weekday() 関数は、集計を曜日別に比較する場合に便利です。例えば、曜日別に製品の平均売上を比較するような場合です。

これらの軸は、関数を使用して**マスター カレンダー テーブル**に項目を作成することにより、ロード スクリプトで作成することも、計算メジャーとしてチャートで直接作成することもできます。

関連トピック

トピック	相互作用
FirstWeekDay (page 228)	各週の開始日を定義します。

引数

引数	説明
date	評価する日付またはタイムスタンプ。
first_week_day	週の開始日を指定します。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。 *FirstWeekDay (page 228)*

次の値を使用して、first_week_day 引数で週が始まる日を設定できます。

first_week_day values

毎日	値
月曜日	0
火曜日	1
水曜日	2
木曜日	3
金曜日	4
土曜日	5
日曜日	6

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

特に述べられていない限り、以下の例において FirstWeekDay は 0 に設定されます。

関数の例

例	結果
`weekday('10/12/1971')`	'Tue' と1 を返します。
`weekday('10/12/1971' , 6)`	'Tue' と2 を返します。 この例では、日曜日 (6) が週の初日です。

`SET FirstWeekDay=6;`
`...`
 	
`weekday('10/12/1971')`
	'Tue' と2 を返します。

例 1 - Weekday 文字列

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- (日曜日) に設定された FirstWeekDay システム変数。
- 既定の曜日名を使うよう設定された DayNames 変数。
- weekday() 関数を含む先行する LOAD で、[week_day] 項目として設定されており、トランザクションが発生した曜日を返します。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
    Load
       *,
       WeekDay(date) as week_day
       ;
Load
*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week_day

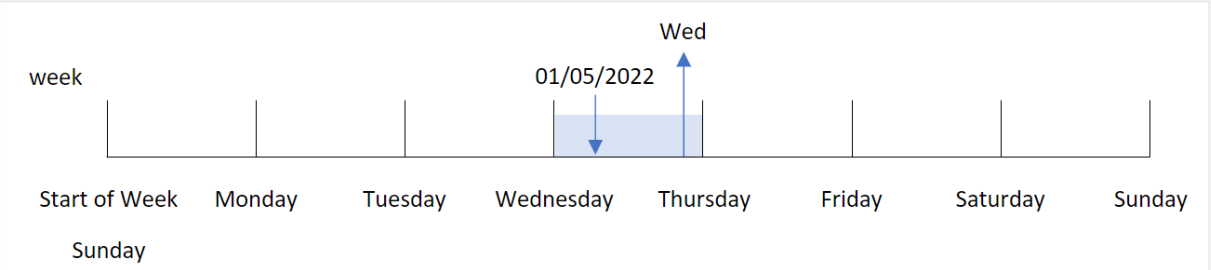
結果テーブル

ID	日付	week_day
8188	01/01/2022	Sat
8189	01/02/2022	日
8190	01/03/2022	月
8191	01/04/2022	火
8192	01/05/2022	水
8193	01/06/2022	Thu
8194	01/07/2022	Fri

[week_day] 項目は、weekday() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

weekday() 関数は曜日文字列値を返します。つまり、DayNames システム変数によって設定された曜日の名前を返します。

トランザクション 8192 の曜日として水曜日を返す weekday() 関数。



トランザクション 8192 は 1 月 5 日に発生しました。FirstWeekDay システム変数は週の初日を日曜日に設定します。weekday() 関数トランザクションは水曜日に発生し、この値を DayNames システム変数の略式で [week_day] 項目に返します。

[week_day] 項目の値は列で右揃えされていますが、これは項目 (水曜日、3) にデュアル値とテキスト結果があるためです。項目値を同等の数値に変換するため、項目を num() 関数の内部にラップすることができます。例えば、トランザクション 8192、水曜日の値は数値 3 に変換されます。

例 2 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- (日曜日) に設定された FirstWeekDay システム変数。
- 既定の曜日名を使うよう設定された DayNames 変数。
- weekday() 関数を含む先行する LOAD で、[week_day] 項目として設定されており、トランザクションが発生した曜日を返します。

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
    Load
       *,
       WeekDay(date,1) as week_day
       ;
Load
*
Inline
[
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

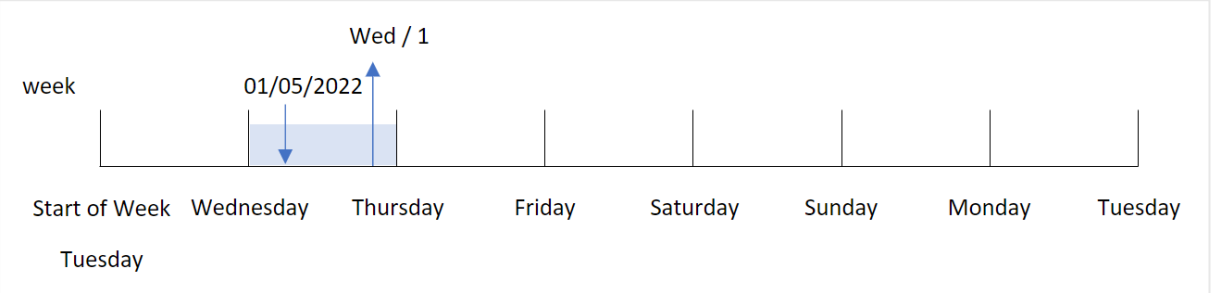
- id
- date
- week_day

結果テーブル

ID	日付	week_day
8188	01/01/2022	Sat
8189	01/02/2022	日
8190	01/03/2022	月
8191	01/04/2022	火
8192	01/05/2022	水

ID	日付	week_day
8193	01/06/2022	Thu
8194	01/07/2022	Fri

水曜日にデュアル値の 1 があることを示す weekday() 関数の図



first_week_day 引数が weekday() 関数で 1 に設定されているため、週の初日は火曜日です。そのため、火曜日に発生するすべてのトランザクションにはデュアル値 0 があります。

トランザクション 8192 は 1 月 5 日に発生します。weekday() 関数はこれが水曜日であることを特定するため、式はデュアル値 1 を返します。

例 3 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- 6 (日曜日) に設定された FirstWeekDay システム変数。
- 既定の曜日名を使うよう設定された DayNames 変数。

ただし、この例では、データセットは変更されず、アプリケーションにロードされます。曜日の値を特定する計算は、アプリのチャートのメジャーとして作成されます。

ロード スクリプト

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
Load
*
Inline
[
```

```
id,date,amount
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.39
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date

曜日の値を計算するには、次のメジャーを作成します。

- =weekday(date)

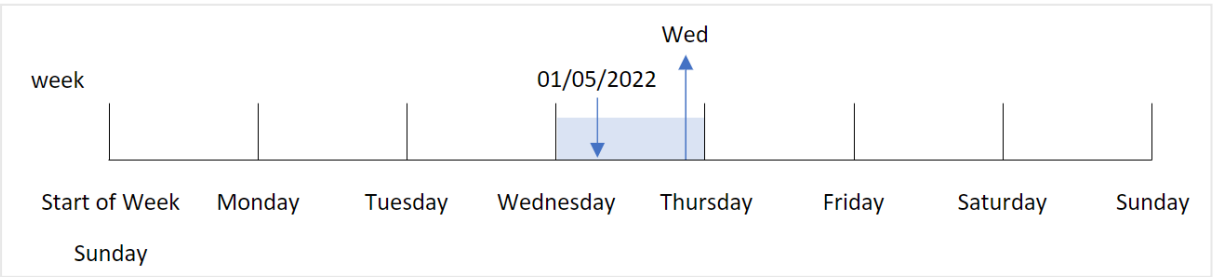
結果テーブル

ID	日付	=weekday(date)
8188	01/01/2022	Sat
8189	01/02/2022	日
8190	01/03/2022	月
8191	01/04/2022	火
8192	01/05/2022	水
8193	01/06/2022	Thu
8194	01/07/2022	Fri

[=weekday(date)] 項目は、weekday() 関数を使用し、関数の引数として日付項目を渡すことにより、チャートで作成されます。

weekday() 関数は曜日文字列値を返します。つまり、DayNames システム変数によって設定された曜日の名前を返します。

トランザクション 8192 の曜日として水曜日を返す weekday() 関数。



トランザクション 8192 は 1 月 5 日に発生しました。FirstWeekDay システム変数は週の初日を日曜日に設定します。weekday() 関数トランザクションは水曜日に発生し、この値を DayNames システム変数の略式で [=weekday(date)] 項目に返します。

例 4 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- 6 (日曜日) に設定された FirstWeekDay システム変数。
- 既定の曜日名を使うよう設定された DayNames 変数。

エンドユーザーは、トランザクションの曜日ごとの平均売上高を示すチャートを求めています。

ロードスクリプト

```
SET DateFormat='MM/DD/YYYY';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=6;

Transactions:
LOAD
  RecNo() AS id,
  MakeDate(2022, 1, Ceil(Rand() * 31)) as date,
  Rand() * 1000 AS amount

Autogenerate(1000);
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- =weekday(date)
- =avg(amount)

メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

=weekday(date)	Avg(amount)
日	$536.96
月	$500.80
火	$515.63

=weekday(date)	Avg(amount)
水	$509.21
Thu	$482.70
Fri	$441.33
Sat	$505.22

weekend

この関数は、**date** を含む暦週の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている**DateFormat** です。

構文:

```
WeekEnd(timestamp [, period_no [, first_week_day ]])
```

戻り値データ型: dual

weekend() 関数は、日付がどの週に該当するかを判断します。次に、その週の最後のミリ秒のタイムスタンプを日付形式で返します。週の最初の曜日は、FirstWeekDay 環境変数によって決定されます。ただし、weekend() 関数ではこれよりfirst_week_day 引数が優先されます。

引数

引数	説明
timestamp	評価する日付またはタイムスタンプ。
period_no	**shift** は整数で、値 0 は **date** を含む週を示します。shift の値が負の場合は過去の週を、正の場合は将来の週を示します。
first_week_day	週の開始日を指定します。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。 **first_week_day** には、0 が月曜日、1 が火曜日、2 が水曜日、3 が木曜日、4 が金曜日、5 が土曜日、6 が日曜日の値が考えられます。 システム変数の詳細については、*FirstWeekDay (page 228)* を参照してください

使用に適しているケース

weekend() 関数は、ユーザーが指定した日付に週の残りの日数を使う計算を使用する場合に、数式の一部としてよく使われます例えば、その週にまだ発生していない利息の合計を計算したい場合などに使用できます。

次の例は下記の内容を推測します。

```
SET FirstWeekDay=0;
```

例	結果
weekend('01/10/2013')	01/12/2013 23:59:59 を返します。
weekend('01/10/2013', -1)	01/05/2013 23:59:59. を返します。
weekend('01/10/2013', 0, 1)	01/14/2013 23:59:59 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

週数と週番号の ISO 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

```
Set DateFormat   ='YYYY-MM-DD';
Set FirstWeekDay =0;    // Monday as first week day
Set BrokenWeeks  =0;    //(use unbroken weeks)
Set ReferenceDay =4;    // Jan 4th is always in week 1
```

US 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

```
Set DateFormat   ='M/D/YYYY';
Set FirstWeekDay =6;    // Sunday as first week day
Set BrokenWeeks  =1;    //(use broken weeks)
Set ReferenceDay =1;    // Jan 1st is always in week 1
```

上記の例では、weekend() 関数から次のような結果になります。

Weekend 関数の例

Date	ISO week end	US week end
2020 年 12 月 26 日 (土)	2020-12-27	12/26/2020
2020 年 12 月 27 日 (日)	2020-12-27	1/2/2021
2020 年 12 月 28 日 (月)	2021-01-03	1/2/2021
2020 年 12 月 29 日 (火)	2021-01-03	1/2/2021
2020 年 12 月 30 日 (水)	2021-01-03	1/2/2021
2020 年 12 月 31 日 (木)	2021-01-03	1/2/2021
2021 年 1 月 1 日 (金)	2021-01-03	1/2/2021

Date	ISO week end	US week end
2021 年 1 月 2 日 (土)	2021-01-03	1/2/2021
2021 年 1 月 3 日 (日)	2021-01-03	1/9/2021
2021 年 1 月 4 日 (月)	2021-01-10	1/9/2021
2021 年 1 月 5 日 (火)	2021-01-10	1/9/2021

ISO 列は日曜日、US 列は土曜日が週末となります。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する週の終わりのタイムスタンプを返す、項目 [end_of_week] の作成。

ロードスクリプト

```
SET FirstWeekDay=6;

Transactions:
    Load
        *,
        weekend(date) as end_of_week,
        timestamp(weekend(date)) as end_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
```

```
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- end_of_week
- end_of_week_timestamp

結果テーブル

日付	end_of_week	end_of_week_timestamp
1/7/2022	01/08/2022	1/8/2022 11:59:59 PM
1/19/2022	01/22/2022	1/22/2022 11:59:59 PM
2/5/2022	02/05/2022	2/5/2022 11:59:59 PM
2/28/2022	03/05/2022	3/5/2022 11:59:59 PM
3/16/2022	03/19/2022	3/19/2022 11:59:59 PM
4/1/2022	04/02/2022	4/2/2022 11:59:59 PM
5/7/2022	05/07/2022	5/7/2022 11:59:59 PM
5/16/2022	05/21/2022	5/21/2022 11:59:59 PM
6/15/2022	06/18/2022	6/18/2022 11:59:59 PM
6/26/2022	07/02/2022	7/2/2022 11:59:59 PM
7/9/2022	07/09/2022	7/9/2022 11:59:59 PM
7/22/2022	07/23/2022	7/23/2022 11:59:59 PM
7/23/2022	07/23/2022	7/23/2022 11:59:59 PM
7/27/2022	07/30/2022	7/30/2022 11:59:59 PM
8/2/2022	08/06/2022	8/6/2022 11:59:59 PM
8/8/2022	08/13/2022	8/13/2022 11:59:59 PM
8/19/2022	08/20/2022	8/20/2022 11:59:59 PM

日付	end_of_week	end_of_week_timestamp
9/26/2022	10/01/2022	10/1/2022 11:59:59 PM
10/14/2022	10/15/2022	10/15/2022 11:59:59 PM
10/29/2022	10/29/2022	10/29/2022 11:59:59 PM

end_of_week 項目は、weekend() 関数を使用し、関数の引数として日付項目を渡すことにより、先行する LOAD ステートメントで作成されます。

weekend() 関数は、日付値がどの週に該当するかを識別し、その週の最後のミリ秒のタイムスタンプを返します。

weekend() 関数の図、基本的な例

トランザクション 8191 は 2 月 5 日に発生しました。FirstWeekDay システム変数は週の初日を日曜日に設定します。weekend() 関数は、2 月 5 日後の最初の土曜日、つまり週の終わりが 2 月 5 日であったことを特定します。そのため、そのトランザクションの end_of_week 値がその日の最後のミリ秒 2 月 5 日 11:59:59 PM を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の週の始めのタイムスタンプを返す、項目 [previous_week_end] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        weekend(date,-1) as previous_week_end,
        timestamp(weekend(date,-1)) as previous_week_end_timestamp
        ;
Load
*
```

```
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- previous_week_end
- previous_week_end_timestamp

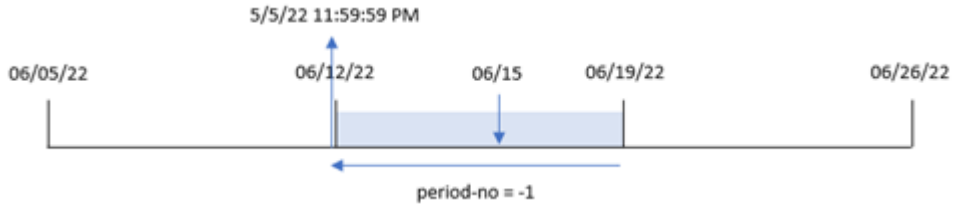
結果テーブル

日付	end_of_week	end_of_week_timestamp
1/7/2022	01/01/2022	1/1/2022 11:59:59 PM
1/19/2022	01/15/2022	1/15/2022 11:59:59 PM
2/5/2022	01/29/2022	1/29/2022 11:59:59 PM
2/28/2022	02/26/2022	2/26/2022 11:59:59 PM
3/16/2022	03/12/2022	3/12/2022 11:59:59 PM
4/1/2022	03/26/2022	3/26/2022 11:59:59 PM
5/7/2022	04/30/2022	4/30/2022 11:59:59 PM
5/16/2022	05/14/2022	5/14/2022 11:59:59 PM
6/15/2022	06/11/2022	6/11/2022 11:59:59 PM

日付	end_of_week	end_of_week_timestamp
6/26/2022	06/25/2022	6/25/2022 11:59:59 PM
7/9/2022	07/02/2022	7/2/2022 11:59:59 PM
7/22/2022	07/16/2022	7/16/2022 11:59:59 PM
7/23/2022	07/16/2022	7/16/2022 11:59:59 PM
7/27/2022	07/23/2022	7/23/2022 11:59:59 PM
8/2/2022	07/30/2022	7/30/2022 11:59:59 PM
8/8/2022	08/06/2022	8/6/2022 11:59:59 PM
8/19/2022	08/13/2022	8/13/2022 11:59:59 PM
9/26/2022	09/24/2022	9/24/2022 11:59:59 PM
10/14/2022	10/08/2022	10/8/2022 11:59:59 PM
10/29/2022	10/22/2022	10/22/2022 11:59:59 PM

この例では、-1 の period_no が weekend() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生する週を識別します。次に、1 週間前を調べて、その週の最後のミリ秒を識別します。

weekend() 関数の図、period_no の例



トランザクション 8196 は 6 月 15 日に発生しました。weekend() 関数は、週が 6 月 12 日に開始することを特定します。そのため、前の週は 6 月 11 日 11:59:59 PM に開始されます。これは、[previous_week_end] 項目に対して返される値です。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では、勤務週の初日として火曜日を設定する必要があります。

ロード スクリプト

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

Transactions:
    Load
        *,
        weekend(date,0,1) as end_of_week,
        timestamp(weekend(date,0,1)) as end_of_week_timestamp,
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- end_of_week
- end_of_week_timestamp

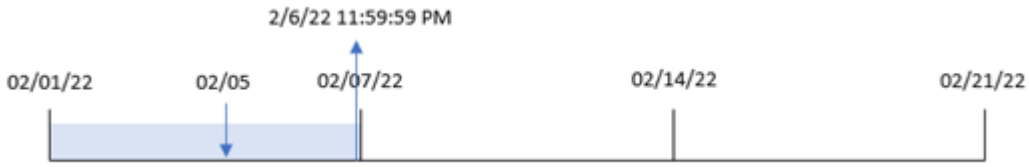
結果テーブル

日付	end_of_week	end_of_week_timestamp
1/7/2022	01/10/2022	1/10/2022 11:59:59 PM
1/19/2022	01/24/2022	1/24/2022 11:59:59 PM

日付	end_of_week	end_of_week_timestamp
2/5/2022	02/07/2022	2/7/2022 11:59:59 PM
2/28/2022	02/28/2022	2/28/2022 11:59:59 PM
3/16/2022	03/21/2022	3/21/2022 11:59:59 PM
4/1/2022	04/04/2022	4/4/2022 11:59:59 PM
5/7/2022	05/09/2022	5/9/2022 11:59:59 PM
5/16/2022	05/16/2022	5/16/2022 11:59:59 PM
6/15/2022	06/20/2022	6/20/2022 11:59:59 PM
6/26/2022	06/27/2022	6/27/2022 11:59:59 PM
7/9/2022	07/11/2022	7/11/2022 11:59:59 PM
7/22/2022	07/25/2022	7/25/2022 11:59:59 PM
7/23/2022	07/25/2022	7/25/2022 11:59:59 PM
7/27/2022	08/01/2022	8/1/2022 11:59:59 PM
8/2/2022	08/08/2022	8/8/2022 11:59:59 PM
8/8/2022	08/08/2022	8/8/2022 11:59:59 PM
8/19/2022	08/22/2022	8/22/2022 11:59:59 PM
9/26/2022	09/26/2022	9/26/2022 11:59:59 PM
10/14/2022	10/17/2022	10/17/2022 11:59:59 PM
10/29/2022	10/31/2022	10/31/2022 11:59:59 PM

この場合、`first_week_date` 引数 1 が `weekend()` 関数で使用されているため、週の初日として火曜日を設定します。

weekend() 関数、first_week_day 例の図



トランザクション 8191 は、2 月 5 日に発生しました。`weekend()` 関数は、この日付後の最初の月曜日、つまり週の最後および返された値は 2 月 6 日 11:59:59 PM であることを特定します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した四半期の終わりのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:`date`。

トランザクションが発生する週の初めを計算するには、次のメジャーを計算します。

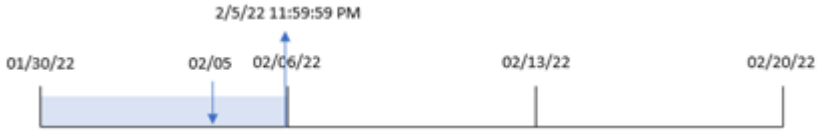
- `=weekend(date)`
- `=timestamp(weekend(date))`

結果テーブル

日付	=weekend(date)	=timestamp(weekend(date))
1/7/2022	01/08/2022	1/8/2022 11:59:59 PM
1/19/2022	01/22/2022	1/22/2022 11:59:59 PM
2/5/2022	02/05/2022	2/5/2022 11:59:59 PM
2/28/2022	03/05/2022	3/5/2022 11:59:59 PM
3/16/2022	03/19/2022	3/19/2022 11:59:59 PM
4/1/2022	04/02/2022	4/2/2022 11:59:59 PM
5/7/2022	05/07/2022	5/7/2022 11:59:59 PM
5/16/2022	05/21/2022	5/21/2022 11:59:59 PM
6/15/2022	06/18/2022	6/18/2022 11:59:59 PM
6/26/2022	07/02/2022	7/2/2022 11:59:59 PM
7/9/2022	07/09/2022	7/9/2022 11:59:59 PM
7/22/2022	07/23/2022	7/23/2022 11:59:59 PM
7/23/2022	07/23/2022	7/23/2022 11:59:59 PM
7/27/2022	07/30/2022	7/30/2022 11:59:59 PM
8/2/2022	08/06/2022	8/6/2022 11:59:59 PM
8/8/2022	08/13/2022	8/13/2022 11:59:59 PM
8/19/2022	08/20/2022	8/20/2022 11:59:59 PM
9/26/2022	10/01/2022	10/1/2022 11:59:59 PM
10/14/2022	10/15/2022	10/15/2022 11:59:59 PM
10/29/2022	10/29/2022	10/29/2022 11:59:59 PM

「end_of_week」メジャーは、weekend() 関数を使用し、関数の引数として日付項目を渡すことにより、チャートオブジェクトで作成されます。weekend() 関数は、日付値がどの週に該当するかを識別し、その週の最後のミリ秒のタイムスタンプを返します。

weekend() 関数の図、チャートオブジェクトの例



トランザクション 8191 は 2 月 5 日に発生しました。FirstWeekDay システム変数は週の初日を日曜日に設定します。weekend() 関数は、2 月 5 日後の最初の土曜日、つまり週の終わりが 2 月 5 日であったことを特定します。そのため、そのトランザクションの end_of_week 値がその日の最後のミリ秒 2 月 5 日 11:59:59 PM を返します。

例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Employee_Expenses」というテーブルにロードされるデータセット。
- 従業員 ID、従業員名および各従業員の平均日次経費請求で構成されたデータ。

エンドユーザーは、従業員 ID と従業員名別に、その週の残りの期間にまだ発生する推定経費請求を表示するグラフオブジェクトを求めています。

ロードスクリプト

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:
   - employee_id
   - employee_name
2. 次に、メジャーを作成して、累積利息を計算します。
   `=(weekend(today(1))-today(1))*avg_daily_claim`
3. メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

employee_id	employee_name	=(weekend(today(1))-today(1))*avg_daily_claim
182	Mark	$90.00
183	Deryck	$75.00

employee_id	employee_name	=(weekend(today(1))-today(1))*avg_daily_claim
184	Dexter	$75.00
185	Sydney	$162.00
186	Agatha	$108.00

weekend() 関数は、今日の日付を唯一の引数として使用することにより、現在の週の終了日を返します。次に、週の終了日から今日の日付を引くことによって、数式は今週の残りの日数を返します。

次に、この値に各従業員による1日あたりの平均経費請求額を乗算して、週の残り期間に各従業員が行うと予想される請求の推定額を計算します。

weekname

この関数は、**date** を含む週の初日の最初のミリ秒のタイムスタンプに対応する数値を基底として、年と週番号を表示する値を返します。

構文:

```
WeekName(date[, period_no [, first_week_day [, broken_weeks [, reference_day]]]])
```

weekname() 関数は、日付が属する週を判別し、その週の週番号と年を返します。週の最初の曜日は、FirstWeekDay システム変数によって決定されます。ただし、weekname() 関数で first_week_day 引数を使用して、週の最初の曜日を変更することもできます。

Qlik Sense では、アプリ作成時に地域設定がフェッチされ、対応する設定は環境変数としてスクリプトに保管されます。

米国のアプリ開発者は、分離した週に対応して、スクリプトで Set BrokenWeeks=1; をよく取得します。ヨーロッパのアプリ開発者は、未分離の週に対応して、スクリプトで Set BrokenWeeks=0; をよく取得します。

アプリケーションが分割された週を使用する場合、発生日数に関係なく週数のカウントは 1 月 1 日に始まり、FirstWeekDay システム変数の前日に終了します。

ただし、アプリケーションが連続した週を使用している場合、第 1 週は前年または 1 月の最初の数日間に開始できます。これは、ReferenceDay および FirstWeekDay システム変数の使用方法によって異なります。

Weekname 関数の例

Date	ISO week name	US week name
2020 年 12 月 26 日 (土)	2020/52	2020/52
2020 年 12 月 27 日 (日)	2020/52	2020/53
2020 年 12 月 28 日 (月)	2020/53	2020/53
2020 年 12 月 29 日 (火)	2020/53	2020/53
2020 年 12 月 30 日 (水)	2020/53	2020/53
2020 年 12 月 31 日 (木)	2020/53	2020/53

Date	ISO week name	US week name
2021 年 1 月 1 日 (金)	2020/53	2021/01
2021 年 1 月 2 日 (土)	2020/53	2021/01
2021 年 1 月 3 日 (日)	2020/53	2021/02
2021 年 1 月 4 日 (月)	2021/01	2021/02
2021 年 1 月 5 日 (火)	2021/01	2021/02

使用に適しているケース

weekname() 関数は、集計を週単位で比較する場合に便利です。

たとえば、製品の総売上高を週ごとに表示する場合などが考えられます。アプリケーションで BrokenWeeks 環境変数との一貫性を維持するには、weekname() の代わりに lunarweekname() を使用します。アプリケーションが連続週を使用している場合、第 1 週には前年の 12 月の日付が含まれるか、今年の 1 月の日付が除外されます。アプリケーションが分割された週を使用している場合、第 1 週 に含まれる日数が 7 日間を下回ることがあります。

戻り値データ型 : dual

引数

引数	説明
timestamp	評価する日付またはタイムスタンプ。
period_no	**shift** は整数で、値 0 は **date** を含む週を示します。shift の値が負の場合は過去の週を、正の場合は将来の週を示します。
first_week_day	週の開始日を指定します。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。 **first_week_day** には、0 が月曜日、1 が火曜日、2 が水曜日、3 が木曜日、4 が金曜日、5 が土曜日、6 が日曜日の値を使用できます。 システム変数の詳細については、*FirstWeekDay (page 228)* を参照してください。
broken_weeks	**broken_weeks** が指定されていない場合は、変数 **BrokenWeeks** の値を使用して、週が分離しているかどうかを定義します。
reference_day	**reference_day** が指定されていない場合は、変数 **ReferenceDay** の値を使用して、第 1 週を定義する参照日として設定する 1 月の日を定義します。デフォルトでは、Qlik Sense 関数は 4 を参照日として使用します。これは、第 1 週に必ず 1 月 4 日が含まれる、または第 1 週に少なくとも 1 月の 4 日間が常に含まれることを意味します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更

できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

下記の例では

```
Set FirstWeekDay=0;
Set BrokenWeeks=0;
Set ReferenceDay=4;
```

関数の例

例	結果
weekname('01/12/2013')	2013/02 を返します。
weekname('01/12/2013', -1)	2013/01 を返します。
weekname('01/12/2013', 0, 1)	2013/02 を返します。

例 1 – 追加の引数がない日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2021 年の最後の週と2022 年の最初の週のトランザクションを含むデータセットは、「Transactions」というテーブルにロードされます。
- MM/DD/YYYY 形式に設定された DateFormat システム変数。
- 1 に設定された BrokenWeeks システム変数。
- 6 に設定された FirstWeekDay システム変数。
- 次を含む、先行する LOAD:
  - 項目「week_number」として設定された weekday() 関数は、トランザクションが発生した年と週番号を返します。
  - 「week_day」という項目として設定される weekname() 関数は、各トランザクション日付の平日の値を表示します。

ロードスクリプト

```
SET BrokenWeeks=1;
SET DateFormat='MM/DD/YYYY';
SET FirstWeekDay=6;
```

```
Transactions:
   Load
     *,
     WeekDay(date) as week_day,
     Weekname(date) as week_number
     ;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week_day
- week_number

結果テーブル

ID	日付	week_day	week_number
8183	12/27/2021	月	2021/53
8184	12/28/2021	火	2021/53
8185	12/29/2021	水	2021/53
8186	12/30/2021	木	2021/53

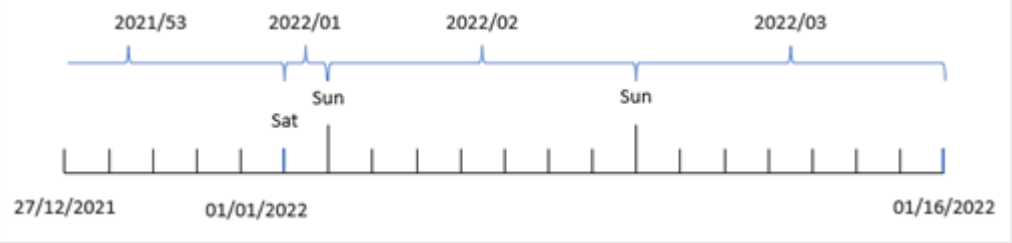
ID	日付	week_day	week_number
8187	12/31/2021	金	2021/53
8188	01/01/2022	土	2022/01
8189	01/02/2022	日	2022/02
8190	01/03/2022	月	2022/02
8191	01/04/2022	火	2022/02
8192	01/05/2022	水	2022/02
8193	01/06/2022	木	2022/02
8194	01/07/2022	金	2022/02
8195	01/08/2022	土	2022/02
8196	01/09/2022	日	2022/03
8197	01/10/2022	月	2022/03
8198	01/11/2022	火	2022/03
8199	01/12/2022	水	2022/03
8200	01/13/2022	木	2022/03
8201	01/14/2022	金	2022/03

「week_number」項目は、weekname() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

weekname() 関数は、最初に日付値が属する週を識別し、週番号のカウントとトランザクションが行われた年を返します。

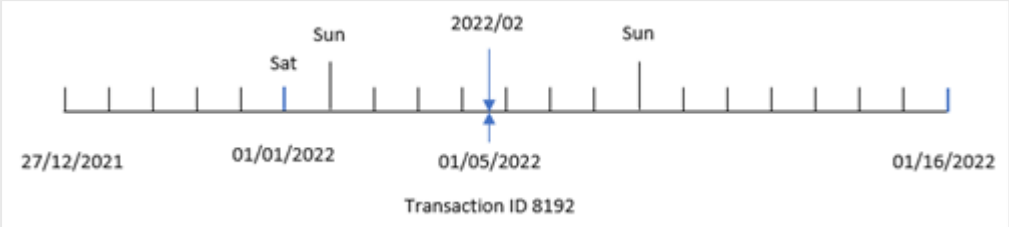
FirstWeekDay システム変数は、日曜日を最初の曜日に設定します。BrokenWeeks システム変数は、アプリケーションが分割週を使用するように設定します。つまり、第 1 週 は 1 月 1 日に始まります。

既定の変数を使用した weekname() 関数の図。



第 1 週 は土曜日である 1 月 1 日に始まるため、この日に発生したトランザクションは値 2022/01 (年と週番号) を返します。

トランザクション 8192 の週番号を識別する weekname() 関数の図。



アプリケーションが分割された週を使用しており、最初の平日が日曜日であるため、1 月 2 日 ~ 1 月 8 日に発生したトランザクションは値 2022/02 を返します (2022 年の第 2 週)。「week_number」項目の値 2022/02 を返します。

例 2 – period_no

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例のタスクは、トランザクションが発生する前の年と週番号を返す項目「previous_week_number」を作成することです。

データ ロード エディター を開き、次のロードスクリプトを新しいタブに追加します。

ロードスクリプト

```
SET BrokenWeeks=1;
SET FirstWeekDay=6;

Transactions:
   Load
     *,
     weekname(date,-1) as previous_week_number
     ;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
```

```
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

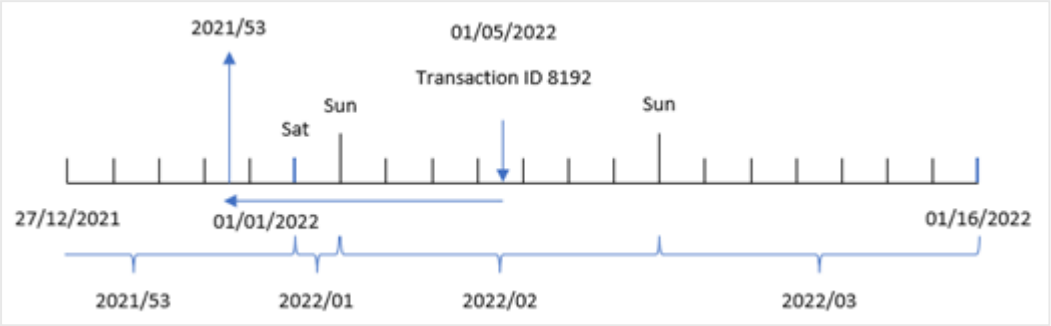
- id
- date
- week_day
- week_number

結果テーブル

ID	日付	week_day	week_number
8183	12/27/2021	月	2021/52
8184	12/28/2021	火	2021/52
8185	12/29/2021	水	2021/52
8186	12/30/2021	木	2021/52
8187	12/31/2021	金	2021/52
8188	01/01/2022	土	2021/52
8189	01/02/2022	日	2021/53
8190	01/03/2022	月	2021/53
8191	01/04/2022	火	2021/53
8192	01/05/2022	水	2021/53
8193	01/06/2022	木	2021/53
8194	01/07/2022	金	2021/53
8195	01/08/2022	土	2022/01
8196	01/09/2022	日	2022/02
8197	01/10/2022	月	2022/02
8198	01/11/2022	火	2022/02
8199	01/12/2022	水	2022/02
8200	01/13/2022	木	2022/02
8201	01/14/2022	金	2022/02

-1 の period_no が weekname() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した週を識別します。次に、1 週間前を調べて、その週の最初のミリ秒を識別します。

period_no オフセットが -1 の weekname() 関数の図。



トランザクション 8192 は 2022 年 1 月 5 日に発生しました。weekname() 関数は、2021 年 12 月 30 日の 1 週間前を検索し、その日付の週番号と年 – 2021/53 を返します。

例 3 – first_week_day

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただしこの例では、会社の方針により、週の勤務日は火曜日に始まります。

データロードエディターを開き、次のロードスクリプトを新しいタブに追加します。

ロードスクリプト

```
SET BrokenWeeks=1;
SET DateFormat='MM/DD/YYYY';

Transactions:
   Load
     *,
     weekday(date) as week_day,
     weekname(date,0,1) as week_number
     ;
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
```

```
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

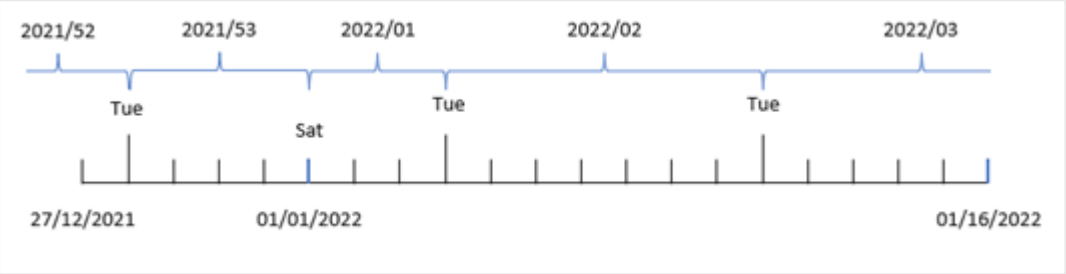
- id
- date
- week_day
- week_number

結果テーブル

ID	日付	week_day	week_number
8183	12/27/2021	月	2021/52
8184	12/28/2021	火	2021/53
8185	12/29/2021	水	2021/53
8186	12/30/2021	木	2021/53
8187	12/31/2021	金	2021/53
8188	01/01/2022	土	2022/01
8189	01/02/2022	日	2022/01
8190	01/03/2022	月	2022/01
8191	01/04/2022	火	2022/02
8192	01/05/2022	水	2022/02
8193	01/06/2022	木	2022/02
8194	01/07/2022	金	2022/02
8195	01/08/2022	土	2022/02
8196	01/09/2022	日	2022/02
8197	01/10/2022	月	2022/02

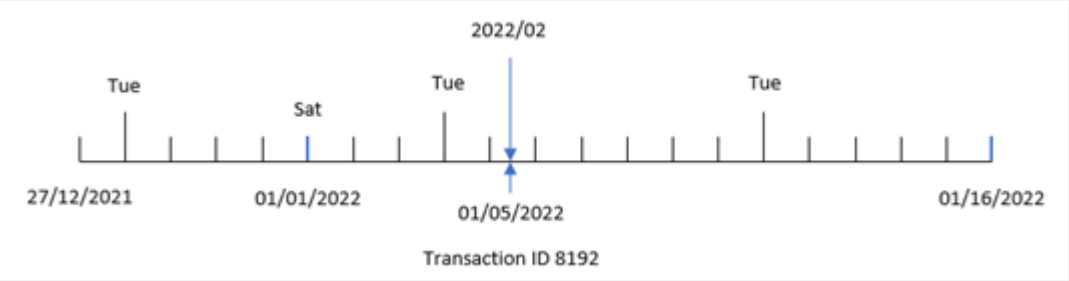
ID	日付	week_day	week_number
8198	01/11/2022	火	2022/03
8199	01/12/2022	水	2022/03
8200	01/13/2022	木	2022/03
8201	01/14/2022	金	2022/03

火曜日を最初の曜日とするweekname()関数の図。



1 の first_week_date 引数が weekname() 関数で使用されているため、火曜日が週の最初の曜日として使用されます。したがってこの関数では、2021 年の第 53 週が 12 月 28 日火曜日に始まると判断します。また、分割された週を使用するアプリケーションにより、第 1 週は 2022 年 1 月 1 日に始まり、2022 年 1 月 3 日の月曜日の最後のミリ秒で終わります。

火曜日が週の最初の日であるトランザクション8192の週番号を示す図。



トランザクション 8192 は 2022 年 1 月 5 日に発生しました。したがって、火曜日の first_week_day パラメータを使用すると、weekname() 関数は 'week_number' 項目の値 2022/02 を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例では、データセットは変更されず、アプリケーションにロードされます。トランザクションが発生した週の年番号を返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
SET BrokenWeeks=1;
Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2021,58.27
8184,12/28/2021,67.42
8185,12/29/2021,23.80
8186,12/30/2021,82.06
8187,12/31/2021,40.56
8188,01/01/2022,37.23
8189,01/02/2022,17.17
8190,01/03/2022,88.27
8191,01/04/2022,57.42
8192,01/05/2022,53.80
8193,01/06/2022,82.06
8194,01/07/2022,40.56
8195,01/08/2022,53.67
8196,01/09/2022,26.63
8197,01/10/2022,72.48
8198,01/11/2022,18.37
8199,01/12/2022,45.26
8200,01/13/2022,58.23
8201,01/14/2022,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- =week_day (date)

トランザクションが発生する週の初めを計算するには、次のメジャーを作成します。

```
=weekname(date)
```

結果テーブル

ID	日付	=weekday(date)	=weekname(date)
8183	12/27/2021	月	2021/53
8184	12/28/2021	火	2021/53
8185	12/29/2021	水	2021/53
8186	12/30/2021	木	2021/53

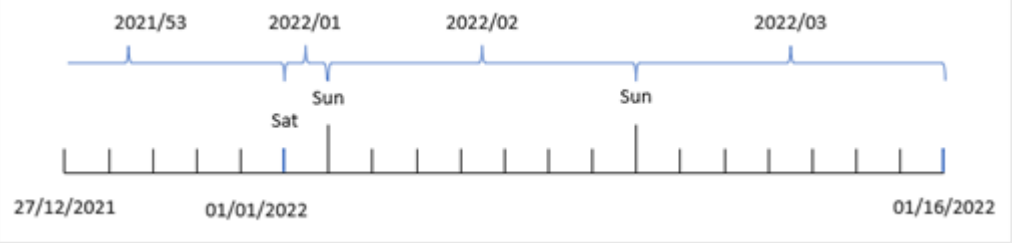
ID	日付	=weekday(date)	=weekname(date)
8187	12/31/2021	金	2021/53
8188	01/01/2022	土	2022/01
8189	01/02/2022	日	2022/02
8190	01/03/2022	月	2022/02
8191	01/04/2022	火	2022/02
8192	01/05/2022	水	2022/02
8193	01/06/2022	木	2022/02
8194	01/07/2022	金	2022/02
8195	01/08/2022	土	2022/02
8196	01/09/2022	日	2022/03
8197	01/10/2022	月	2022/03
8198	01/11/2022	火	2022/03
8199	01/12/2022	水	2022/03
8200	01/13/2022	木	2022/03
8201	01/14/2022	金	2022/03

「week_number」項目は、weekname() 関数を使用し、関数の引数として日付項目を渡すことにより、チャートオブジェクトのメジャーとして作成されます。

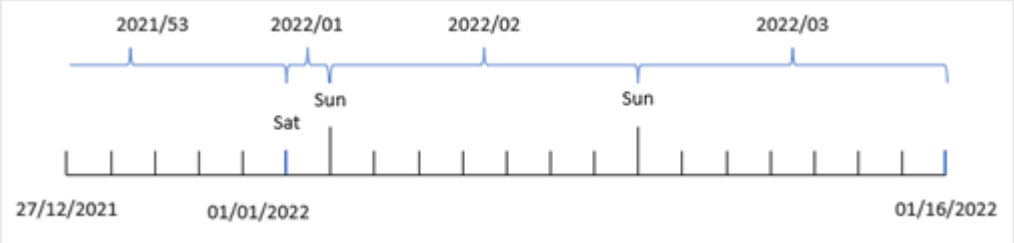
weekname() 関数は、最初に日付値が属する週を識別し、週番号のカウントとトランザクションが行われた年を返します。

FirstWeekDay システム変数は、日曜日を最初の曜日に設定します。BrokenWeeks システム変数は、アプリケーションが分割週を使用するように設定します。つまり、第 1 週 は 1 月 1 日に始まります。

日曜日を週の最初の曜日とする週番号を示す図。



トランザクション 8192 が 2 週目に発生したことを示す図。



アプリケーションが分割された週を使用しており、最初の平日が日曜日であるため、1 月 2 日 ~ 1 月 8 日に発生するトランザクションは値 2022/02、つまり2022 年の第 2 週 を返します。トランザクション 8192 は 1 月 5 日に発生し、「week_number」項目に対して値 2022/02 を返すことに注意してください。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2019 年の最後の週と2020 年の最初の週のトランザクションを含むデータセットは、Transactions というテーブルにロードされます。
- 0 に設定された BrokenWeeks システム変数。
- 2 に設定された ReferenceDay システム変数。
- MM/DD/YYYY 形式に設定された DateFormat システム変数。

ロードスクリプト

```
SET BrokenWeeks=0;
SET ReferenceDay=2;
SET DateFormat='MM/DD/YYYY';

Transactions:
Load
*
Inline
[
id,date,amount
8183,12/27/2019,58.27
8184,12/28/2019,67.42
8185,12/29/2019,23.80
8186,12/30/2019,82.06
8187,12/31/2019,40.56
8188,01/01/2020,37.23
8189,01/02/2020,17.17
8190,01/03/2020,88.27
8191,01/04/2020,57.42
8192,01/05/2020,53.80
```

```
8193,01/06/2020,82.06
8194,01/07/2020,40.56
8195,01/08/2020,53.67
8196,01/09/2020,26.63
8197,01/10/2020,72.48
8198,01/11/2020,18.37
8199,01/12/2020,45.26
8200,01/13/2020,58.23
8201,01/14/2020,18.52
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成します。

次の式を使用して計算軸を作成します。

```
=weekname(date)
```

総売上を計算するには、次の集計メジャーを作成します。

```
=sum(amount)
```

メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

weekname(date)	=sum(amount)
2019/52	$125.69
2020/01	$346.51
2020/02	$347.57
2020/03	$122.01

このシナリオで weekname() 関数を使用した結果を示すには、次の項目を軸として追加します。

```
date
```

日付項目を含む結果テーブル

weekname(date)	日付	=sum(amount)
2019/52	12/27/2019	$58.27
2019/52	12/28/2019	$67.42
2020/01	12/29/2019	$23.80
2020/01	12/30/2019	$82.06
2020/01	12/31/2019	$40.56
2020/01	01/01/2020	$37.23
2020/01	01/02/2020	$17.17

weekname(date)	日付	=sum(amount)
2020/01	01/03/2020	$88.27
2020/01	01/04/2020	$57.42
2020/02	01/05/2020	$53.80
2020/02	01/06/2020	$82.06
2020/02	01/07/2020	$40.56
2020/02	01/08/2020	$53.67
2020/02	01/09/2020	$26.63
2020/02	01/10/2020	$72.48
2020/02	01/11/2020	$18.37
2020/03	01/12/2020	$45.26
2020/03	01/13/2020	$58.23
2020/03	01/14/2020	$18.52

アプリケーションは連続した週を使用し、ReferenceDay システム変数のために第 1 週目は 1 月に最低 2 日間必要であるため、2020 年の第 1 週目には 2019 年 12 月 29 日からのトランザクションが含まれます。

weekstart

この関数は、**date** を含む暦週の初日の最初のミリ秒のタイムスタンプに対応する値を返します。デフォルトの出力形式は、スクリプトに設定されている **DateFormat** です。

構文:

```
WeekStart(timestamp [, period_no [, first_week_day ]])
```

戻り値データ型: dual

weekstart() 関数は、日付がどの週に該当するかを判断します。次に、その年の最初のミリ秒のタイムスタンプを日付形式で返します。週の最初の曜日は、FirstWeekDay 環境変数によって決定されます。ただし、weekstart() 関数ではこれよりfirst_week_day 引数が優先されます。

引数

引数	説明
timestamp	評価する日付またはタイムスタンプ。
period_no	**shift** は整数で、値 0 は **date** を含む週を示します。shift の値が負の場合は過去の週を、正の場合は将来の週を示します。

引数	説明
first_week_day	週の開始日を指定します。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。 **first_week_day** には、0 が月曜日、1 が火曜日、2 が水曜日、3 が木曜日、4 が金曜日、5 が土曜日、6 が日曜日の値を使用できます。 システム変数の詳細については、*FirstWeekDay (page 228)* を参照してください。

使用に適しているケース

weekstart() 関数は、ユーザーがこれまで経過した週の端数を計算に使用する場合に、数式の一部として一般的に使用されます。例えば、その週でこれまでに従業員が稼いだ合計賃金を計算したい場合に使用できます。

次の例は下記の内容を推測します。

```
SET FirstWeekDay=0;
```

関数の例

例	結果
weekstart('01/12/2013')	01/07/2013 を返します。
weekstart('01/12/2013', -1)	11/31/2012 を返します。
weekstart('01/12/2013', 0, 1)	01/08/2013 を返します。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

週数と週番号の ISO 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

```
Set DateFormat   ='YYYY-MM-DD';
Set FirstWeekDay =0;    // Monday as first week day
Set BrokenWeeks  =0;    //(use unbroken weeks)
Set ReferenceDay =4;    // Jan 4th is always in week 1
```

US 設定を希望する場合、スクリプトに必ず次を組み込むようにしてください。

```
Set DateFormat   ='M/D/YYYY';
Set FirstWeekDay =6;    // Sunday as first week day
Set BrokenWeeks  =1;    //(use broken weeks)
Set ReferenceDay =1;    // Jan 1st is always in week 1
```

上記の例では、weekstart() 関数から次のような結果になります。

Weekstart 関数の例

Date	ISO week start	US week start
2020 年 12 月 26 日 (土)	2020-12-21	12/20/2020
2020 年 12 月 27 日 (日)	2020-12-21	12/27/2020
2020 年 12 月 28 日 (月)	2020-12-28	12/27/2020
2020 年 12 月 29 日 (火)	2020-12-28	12/27/2020
2020 年 12 月 30 日 (水)	2020-12-28	12/27/2020
2020 年 12 月 31 日 (木)	2020-12-28	12/27/2020
2021 年 1 月 1 日 (金)	2020-12-28	12/27/2020
2021 年 1 月 2 日 (土)	2020-12-28	12/27/2020
2021 年 1 月 3 日 (日)	2020-12-28	1/3/2021
2021 年 1 月 4 日 (月)	2021-01-04	1/3/2021
2021 年 1 月 5 日 (火)	2021-01-04	1/3/2021

ISO 列は月曜日、US 列は日曜日が週初めとなります。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- トランザクションが発生する週の始めのタイムスタンプを返す、項目 [start_of_week] の作成。

ロードスクリプト

```
SET FirstWeekDay=6;

Transactions:
    Load
        *,
```

```
        weekstart(date) as start_of_week,
        timestamp(weekstart(date)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- start_of_week
- start_of_week_timestamp

結果テーブル

日付	start_of_week	start_of_week_timestamp
1/7/2022	01/02/2022	1/2/2022 12:00:00 AM
1/19/2022	01/16/2022	1/16/2022 12:00:00 AM
2/5/2022	01/30/2022	1/30/2022 12:00:00 AM
2/28/2022	02/27/2022	2/27/2022 12:00:00 AM
3/16/2022	03/13/2022	3/13/2022 12:00:00 AM
4/1/2022	03/27/2022	3/27/2022 12:00:00 AM

日付	start_of_week	start_of_week_timestamp
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/15/2022	5/15/2022 12:00:00 AM
6/15/2022	06/12/2022	6/12/2022 12:00:00 AM
6/26/2022	06/26/2022	6/26/2022 12:00:00 AM
7/9/2022	07/03/2022	7/3/2022 12:00:00 AM
7/22/2022	07/17/2022	7/17/2022 12:00:00 AM
7/23/2022	07/17/2022	7/17/2022 12:00:00 AM
7/27/2022	07/24/2022	7/24/2022 12:00:00 AM
8/2/2022	07/31/2022	7/31/2022 12:00:00 AM
8/8/2022	08/07/2022	8/7/2022 12:00:00 AM
8/19/2022	08/14/2022	8/14/2022 12:00:00 AM
9/26/2022	09/25/2022	9/25/2022 12:00:00 AM
10/14/2022	10/09/2022	10/9/2022 12:00:00 AM
10/29/2022	10/23/2022	10/23/2022 12:00:00 AM

start_of_week 項目は、weekstart() 関数を使用し、関数の引数として日付項目を渡すことにより、先行する LOAD ステートメントで作成されます。

weekstart() 関数は、まず日付値がどの週に該当するかを識別し、その週の最初のミリ秒のタイムスタンプを返します。

weekstart() 関数の図、追加の引数がない例

トランザクション 8191 は 2 月 5 日に発生しました。FirstWeekDay システム変数は週の初日を日曜日に設定します。weekstart() 関数は、2 月 5 日前の最初の日曜日、つまり週の始めが 1 月 30 日であったことを特定します。そのため、そのトランザクションの start_of_week 値がその日の最初のミリ秒 1 月 30 日 12:00:00 AM を返します。

例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- トランザクションが発生する前の四半期の始めのタイムスタンプを返す、項目 [previous_week_start] の作成。

ロードスクリプト

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

Transactions:
    Load
        *,
        weekstart(date,-1) as previous_week_start,
        timestamp(weekstart(date,-1)) as previous_week_start_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

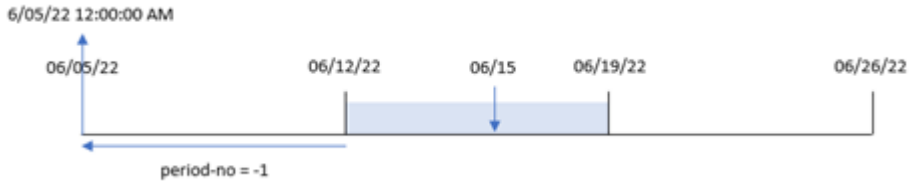
- date
- previous_week_start
- previous_week_start_timestamp

結果テーブル

日付	previous_week_start	previous_week_start_timestamp
1/7/2022	12/26/2021	12/26/2021 12:00:00 AM
1/19/2022	01/09/2022	1/9/2022 12:00:00 AM
2/5/2022	01/23/2022	1/23/2022 12:00:00 AM
2/28/2022	02/20/2022	2/20/2022 12:00:00 AM
3/16/2022	03/06/2022	3/6/2022 12:00:00 AM
4/1/2022	03/20/2022	3/20/2022 12:00:00 AM
5/7/2022	04/24/2022	4/24/2022 12:00:00 AM
5/16/2022	05/08/2022	5/8/2022 12:00:00 AM
6/15/2022	06/05/2022	6/5/2022 12:00:00 AM
6/26/2022	06/19/2022	6/19/2022 12:00:00 AM
7/9/2022	06/26/2022	6/26/2022 12:00:00 AM
7/22/2022	07/10/2022	7/10/2022 12:00:00 AM
7/23/2022	07/10/2022	7/10/2022 12:00:00 AM
7/27/2022	07/17/2022	7/17/2022 12:00:00 AM
8/2/2022	07/24/2022	7/24/2022 12:00:00 AM
8/8/2022	07/31/2022	7/31/2022 12:00:00 AM
8/19/2022	08/07/2022	8/7/2022 12:00:00 AM
9/26/2022	09/18/2022	9/18/2022 12:00:00 AM
10/14/2022	10/02/2022	10/2/2022 12:00:00 AM
10/29/2022	10/16/2022	10/16/2022 12:00:00 AM

この例では、-1 の period_no が weekstart() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した週を識別します。次に、1 週間前を調べて、その週の最初のミリ秒を識別します。

weekstart() 関数の図、period_no の例



トランザクション 8196 は 6 月 15 日に発生しました。weekstart() 関数は、週が 6 月 12 日に開始することを特定します。そのため、前の週は 6 月 5 日 12:00:00 AM に開始されます。これは、[previous_week_start] 項目に対して返される値です。

例 3 – first_week_day

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには、最初の例と同じデータセットとシナリオが含まれます。ただし、この例では、勤務週の初日として火曜日を設定する必要があります。

ロードスクリプト

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

Transactions:
    Load
        *,
        weekstart(date,0,1) as start_of_week,
        timestamp(weekstart(date,0,1)) as start_of_week_timestamp
        ;
Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- start_of_week
- start_of_week_timestamp

結果テーブル

日付	start_of_week	start_of_week_timestamp
1/7/2022	01/04/2022	1/4/2022 12:00:00 AM
1/19/2022	01/18/2022	1/18/2022 12:00:00 AM
2/5/2022	02/01/2022	2/1/2022 12:00:00 AM
2/28/2022	02/22/2022	2/22/2022 12:00:00 AM
3/16/2022	03/15/2022	3/15/2022 12:00:00 AM
4/1/2022	03/29/2022	3/29/2022 12:00:00 AM
5/7/2022	05/03/2022	5/3/2022 12:00:00 AM
5/16/2022	05/10/2022	5/10/2022 12:00:00 AM
6/15/2022	06/14/2022	6/14/2022 12:00:00 AM
6/26/2022	06/21/2022	6/21/2022 12:00:00 AM
7/9/2022	07/05/2022	7/5/2022 12:00:00 AM
7/22/2022	07/19/2022	7/19/2022 12:00:00 AM
7/23/2022	07/19/2022	7/19/2022 12:00:00 AM
7/27/2022	07/26/2022	7/26/2022 12:00:00 AM
8/2/2022	08/02/2022	8/2/2022 12:00:00 AM
8/8/2022	08/02/2022	8/2/2022 12:00:00 AM
8/19/2022	08/16/2022	8/16/2022 12:00:00 AM
9/26/2022	09/20/2022	9/20/2022 12:00:00 AM
10/14/2022	10/11/2022	10/11/2022 12:00:00 AM
10/29/2022	10/25/2022	10/25/2022 12:00:00 AM

この場合、first_week_date 引数 1 が weekstart() 関数で使用されているため、週の初日として火曜日を設定します。

weekstart() 関数、first_week_day 例の図

トランザクション 8191 は、2 月 5 日に発生しました。weekstart() 関数は、この日付前の最初の火曜日、つまり週の最初および返された値は 2 月 1 日 12:00:00 AM であることを特定します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディタを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。トランザクションが発生した月の始めのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:

Load
*
Inline
[
id,date,amount
8188,1/7/2022,17.17
8189,1/19/2022,37.23
8190,2/28/2022,88.27
8191,2/5/2022,57.42
8192,3/16/2022,53.80
8193,4/1/2022,82.06
8194,5/7/2022,40.39
8195,5/16/2022,87.21
8196,6/15/2022,95.93
8197,6/26/2022,45.89
8198,7/9/2022,36.23
8199,7/22/2022,25.66
8200,7/23/2022,82.77
8201,7/27/2022,69.98
8202,8/2/2022,76.11
8203,8/8/2022,25.12
8204,8/19/2022,46.23
8205,9/26/2022,84.21
8206,10/14/2022,96.24
8207,10/29/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:date。

トランザクションが発生する週の初めを計算するには、次のメジャーを計算します。

- =weekstart(date)
- =timestamp(weekstart(date))

結果テーブル

日付	start_of_week	start_of_week_timestamp
1/7/2022	01/02/2022	1/2/2022 12:00:00 AM
1/19/2022	01/16/2022	1/16/2022 12:00:00 AM
2/5/2022	01/30/2022	1/30/2022 12:00:00 AM
2/28/2022	02/27/2022	2/27/2022 12:00:00 AM
3/16/2022	03/13/2022	3/13/2022 12:00:00 AM
4/1/2022	03/27/2022	3/27/2022 12:00:00 AM
5/7/2022	05/01/2022	5/1/2022 12:00:00 AM
5/16/2022	05/15/2022	5/15/2022 12:00:00 AM
6/15/2022	06/12/2022	6/12/2022 12:00:00 AM
6/26/2022	06/26/2022	6/26/2022 12:00:00 AM
7/9/2022	07/03/2022	7/3/2022 12:00:00 AM
7/22/2022	07/17/2022	7/17/2022 12:00:00 AM
7/23/2022	07/17/2022	7/17/2022 12:00:00 AM
7/27/2022	07/24/2022	7/24/2022 12:00:00 AM
8/2/2022	07/31/2022	7/31/2022 12:00:00 AM
8/8/2022	08/07/2022	8/7/2022 12:00:00 AM
8/19/2022	08/14/2022	8/14/2022 12:00:00 AM
9/26/2022	09/25/2022	9/25/2022 12:00:00 AM
10/14/2022	10/09/2022	10/9/2022 12:00:00 AM
10/29/2022	10/23/2022	10/23/2022 12:00:00 AM

[start_of_week] メジャーは、weekstart() 関数を使用し、関数の引数として [date] 項目を渡すことにより、チャートオブジェクトで作成されます。

weekstart() 関数は、まず日付値がどの週に該当するかを識別し、その週の最初のミリ秒のタイムスタンプを返します。

weekstart() 関数の図、チャート オブジェクトの例

トランザクション 8191 は 2 月 5 日に発生しました。FirstWeekDay システム変数は週の初日を日曜日に設定します。weekstart() 関数は、2 月 5 日の前の最初の日曜日、つまり週の始めが 1 月 30 日であったことを特定します。そのため、トランザクションの start_of_week 値は、その日の最初のミリ秒である 1 月 30 日 12:00:00 AM を返します。

例 5 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 「Payroll」というテーブルにロードされるデータセット。
- 従業員 ID、従業員名および各従業員の平均日次賃金で構成されたデータ。

従業員は月曜日に勤務を開始し、週に 6 日間働きます。FirstWeekDay システム変数は変更できません。

エンドユーザーは、従業員 ID と従業員名別にその週のその日までに稼いだ賃金を表示するチャート オブジェクトを求めています。

ロード スクリプト

```
Payroll:
Load
*
Inline
[
employee_id,employee_name,day_rate
182,Mark, $150
183,Deryck, $125
184,Dexter, $125
185,Sydney,$270
186,Agatha,$128
];
```

結果

次の手順を実行します。

1. データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:
   - employee_id
   - employee_name
2. 次に、その週のその日までに稼いだ賃金を計算するメジャーを作成します。
   `=if(today(1)-weekstart(today(1),0,0)<7,(today(1)-weekstart(today(1),0,0))*day_rate,day_rate*6)`
3. メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

employee_id	employee_name	=if(today(1)-weekstart(today(1),0,0)<7,(today(1)-weekstart(today(1),0,0))*day_rate,day_rate*6)
182	Mark	$600.00
183	Deryck	$500.00
184	Dexter	$500.00
185	Sydney	$1080.00
186	Agatha	$512.00

weekstart() 関数は、今日の日付を第一引数、0 を第 3 引数として使うことにより、週の初日として月曜日を設定し、現在の週の開始日付を返します。その結果を現在の日付から減算することにより、数式は今週経過した日数を返します。

次に条件で、この週の経過日数が 6 日を超えているかどうかが評価されます。その場合、従業員の day_rate は 6 日間で乗算されます。そうでない場合、day_rate は、今週発生した日数で乗算されます。

weekyear

この関数は、環境変数に基づいた週番号が含まれる年を返します。週番号の範囲は、1 からおよそ 52 となります。

構文:

```
weekyear(timestamp [, first_week_day [, broken_weeks [, reference_day]]])
```

戻り値データ型: integer

引数

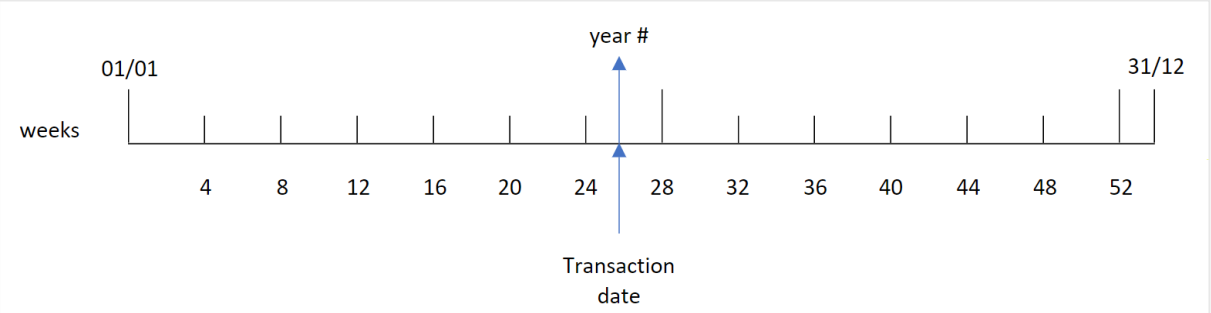
引数	説明
timestamp	評価する日付またはタイムスタンプ。

引数	説明
first_week_day	週の開始日を指定します。省略されている場合は、変数 **FirstWeekDay** の値が使用されます。 **first_week_day** には、0 が月曜日、1 が火曜日、2 が水曜日、3 が木曜日、4 が金曜日、5 が土曜日、6 が日曜日の値を使用できます。 システム変数の詳細については、*FirstWeekDay (page 228)* を参照してください。
broken_weeks	**broken_weeks** が指定されていない場合は、変数 **BrokenWeeks** の値を使用して、週が分離しているかどうかを定義します。
reference_day	**reference_day** が指定されていない場合は、変数 **ReferenceDay** の値を使用して、第 1 週を定義する参照日として設定する 1 月の日を定義します。デフォルトでは、Qlik Sense 関数は 4 を参照日として使用します。これは、第 1 週に必ず 1 月 4 日が含まれる、または第 1 週に少なくとも 1 月の 4 日間が常に含まれることを意味します。

weekyear() 関数は、日付が年のどの週に該当するかを判断します。次に、その週番号に対応する年を返します。

BrokenWeeks が 0 (false) に設定されると、weekyear() が year() と同じ結果を返します。

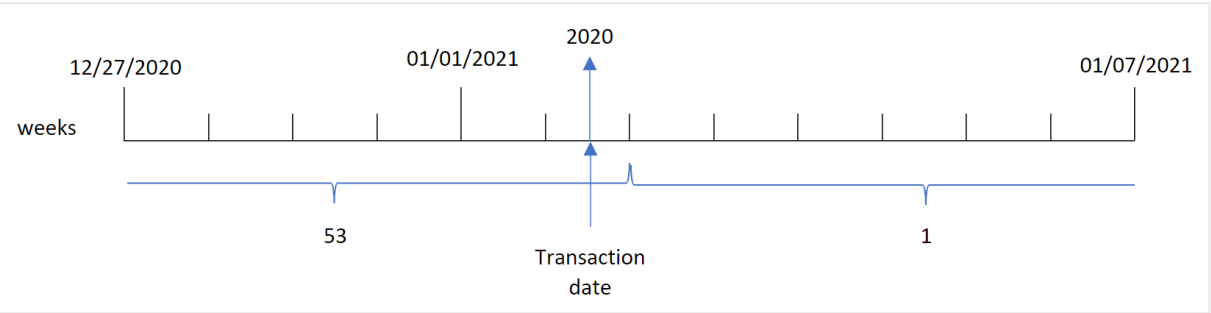
weekyear() 関数範囲の図



ただし、BrokenWeeks システム変数が未分離の週を使用するように設定されている場合、ReferenceDay システム変数で指定した値に基づいて、第 1 週は 1 月の特定の日数のみを含むことになります。

例えば、ReferenceDay 値 4 が使用される場合、第 1 週は 1 月に少なくとも 4 日を含む必要があります。第 1 週が前年 12 月の日付を含むことも、1 年の最後の週番号が翌年 1 月の日付を含むことも考えられます。このような状況では、weekyear() 関数は year() 関数に異なる値を返します。

未分離の週を使用した場合の、weekyear() 関数の範囲の図



使用に適しているケース

weekyear() 関数は、集計を年単位で比較する場合に便利です。たとえば、製品の総売上高を年ごとに表示する場合などが考えられます。ユーザーがアプリで BrokenWeeks システム変数との一貫性を維持したい場合は、year() ではなくweekyear() 関数が選択されます。

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
weekyear('12/30/1996',0,0,4)	1997 を返します (1997 年の第 1 週は 12/30/1996 に開始するため)
weekyear('01/02/1997',0,0,4)	1997 を返します
weekyear('12/28/1997',0,0,4)	1997 を返します
weekyear('12/30/1997',0,0,4)	1998 を返します (1998 年の第 1 週は 12/29/1997 に開始するため)
weekyear('01/02/1999',0,0,4)	1998 を返します (1998 年の第 53 週は 01/03/1999 に終了するため)

関連トピック

トピック	相互作用
week (page 1036)	ISO 8601 に従って、週番号を表す整数を返します

トピック	相互作用
year (page 1109)	expression が標準的な数値の解釈に従って日付と判断される場合に、年を表す整数を返します。

例 1 - 分離された週

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2021 年の最後の週と2021 年の最初の週のトランザクションを含むデータセットで、「Transactions」というテーブルにロードされます。
- 1 に設定された BrokenWeeks 変数。
- 次を含む、先行する LOAD:
    - 項目 [week_year] として設定された weekyear() 関数で、トランザクションが発生した年を返します。
    - 項目 [week] として設定された week() 関数で、各トランザクション日付の週番号を返します。

ロードスクリプト

```
SET BrokenWeeks=1;

Transactions:
       Load
       *,
       week(date) as week,
       weekyear(date) as week_year
       ;
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week
- week_year

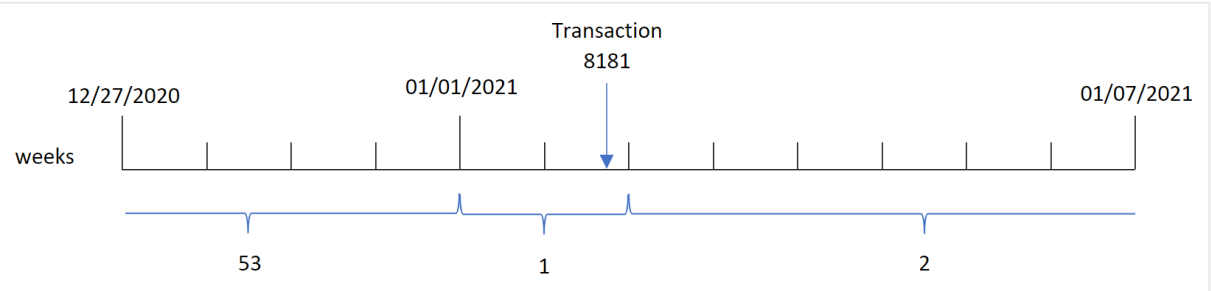
結果テーブル

ID	日付	週	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	1	2021
8181	01/02/2021	1	2021
8182	01/03/2021	2	2021
8183	01/04/2021	2	2021
8184	01/05/2021	2	2021
8185	01/06/2021	2	2021
8186	01/07/2021	2	2021

[week_year] 項目は、weekyear() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

BrokenWeeks システム変数は 1 に設定されます。つまり、アプリは分離された週を使うということです。第 1 週は 1 月 1 日に始まります。

未分離の週を使用した場合の、weekyear() 関数の範囲の図



トランザクション 8181 は、第 1 週である 1 月 2 日に発生します。そのため、[week_year] 項目に値 2021 を返します。

例 2 - 未分離の週

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2021年の最後の週と2021年の最初の週のトランザクションを含むデータセットで、Transactions というテーブルにロードされます。
- 0に設定されたBrokenWeeks変数。
- 次を含む、先行するLOAD:
    - 項目 [week_year] として設定されたweekyear() 関数で、トランザクションが発生した年を返します。
    - 項目 [week] として設定されたweek() 関数で、各トランザクション日付の週番号を返します。

ただし、この例では、会社ポリシーで未分離の週を使用するよう定められています。

ロードスクリプト

```
SET BrokenWeeks=0;

Transactions:
       Load
       *,
       week(date) as week,
       weekyear(date) as week_year
       ;
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- week
- week_year

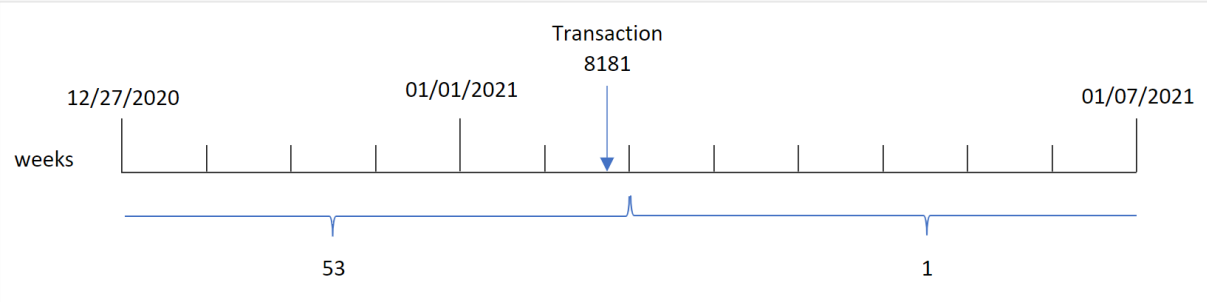
結果テーブル

ID	日付	週	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	53	2020
8181	01/02/2021	53	2020
8182	01/03/2021	1	2021
8183	01/04/2021	1	2021
8184	01/05/2021	1	2021
8185	01/06/2021	1	2021
8186	01/07/2021	1	2021

BrokenWeeks システム変数は 0 に設定されます。つまり、アプリケーションは未分離の週を使うということです。そのため、第 1 週は 1 月 1 日に始まる必要はありません。

2020 年の第 53 週は 2021 年の 1 月 2 日の終わりまで続き、2020 年の第 1 週は 2021 年 1 月 3 日の日曜日に始まります。

分離された週を使用した場合の、weekyear() 関数の範囲の図



トランザクション 8181 は、第 1 週である 1 月 2 日に発生します。そのため、[week_year] 項目に値 2021 を返します。

例 3 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例では、データセットは変更されず、アプリケーションにロードされます。トランザクションが発生した年の週番号を返す計算は、アプリのチャートのメジャーとして作成されます。

ロード スクリプト

```
SET BrokenWeeks=1;

Transactions:
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date

トランザクションが発生する週を計算するには、次のメジャーを作成します。

- =week(date)

週番号に基づいてトランザクションが発生する年を計算するには、次のメジャーを作成します。

- =weekyear(date)

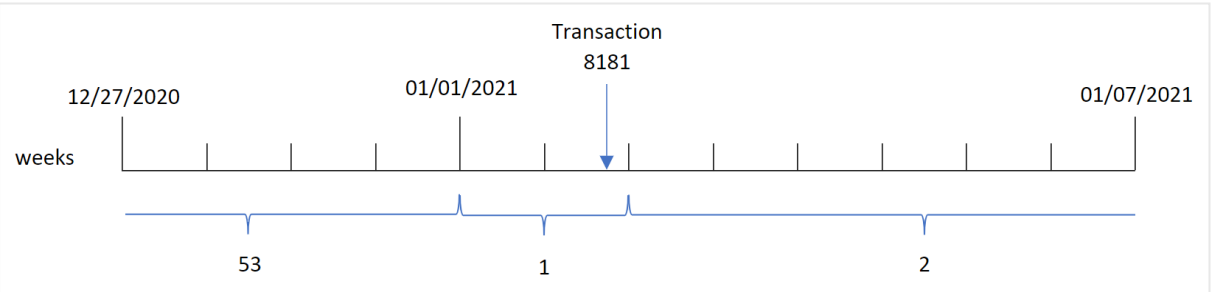
結果テーブル

ID	日付	週	week_year
8176	12/28/2020	53	2020
8177	12/29/2020	53	2020
8178	12/30/2020	53	2020
8179	12/31/2020	53	2020
8180	01/01/2021	1	2021
8181	01/02/2021	1	2021
8182	01/03/2021	2	2021
8183	01/04/2021	2	2021
8184	01/05/2021	2	2021
8185	01/06/2021	2	2021
8186	01/07/2021	2	2021

[week_year] 項目は、weekyear() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

BrokenWeeks システム変数は 1 に設定されます。つまり、アプリは分離された週を使うということです。第 1 週は 1 月 1 日に始まります。

未分離の週を使用した場合の、weekyear() 関数の範囲の図



トランザクション 8181 は、第 1 週である 1 月 2 日に発生します。そのため、[week_year] 項目に値 2021 を返します。

例 4 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2021 年の最後の週と2021 年の最初の週のトランザクションを含むデータセットで、Transactions というテーブルにロードされます。
- 0 に設定された BrokenWeeks 変数。これは、アプリが未分離の週を使うということです。
- 2 に設定された ReferenceDay 変数。これは、年が 1 月 2 日に始まり、1 月のうち少なくとも 2 日を含むということです。
- 1 に設定された FirstWeekDay 変数。これは、週の最初の日が火曜日であるということです。

会社ポリシーでは、分離された週を使用することになっています。エンドユーザーは、年ごとの総売上高を示すチャートを求めています。アプリは未分離の週を使用し、第 1 週に 1 月の少なくとも 2 日を含めます。

ロード スクリプト

```
SET BrokenWeeks=0;
SET ReferenceDay=2;
SET FirstWeekDay=1;

Transactions:
Load
*
Inline
[
id,date,amount
8176,12/28/2020,19.42
8177,12/29/2020,23.80
8178,12/30/2020,82.06
8179,12/31/2020,40.56
8180,01/01/2021,37.23
8181,01/02/2021,17.17
8182,01/03/2021,88.27
8183,01/04/2021,57.42
8184,01/05/2021,67.42
8185,01/06/2021,23.80
8186,01/07/2021,82.06
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成します。

週番号に基づいてトランザクションが発生する年を計算するには、次のメジャーを作成します。

- =weekyear(date)

総売上を計算するには、次のメジャーを作成します。

- sum(amount)

メジャーの [**数値書式**] を [**通貨**] に設定します。

結果テーブル

weekyear(date)	=sum(amount)
2020	19.42
2021	373.37

year

この関数は、**expression** が標準的な数値の解釈に従って日付と判断される場合に、年を表す整数を返します。

構文:

```
year(expression)
```

戻り値データ型: 整数

year() 関数はスクリプトとチャート関数の両方として使用できます。関数は、特定の日付の年を返します。これは通常、マスターカレンダーの軸として年項目を作成するために使用します。

使用に適しているケース

year() 関数は、集計を年単位で比較する場合に便利です。例えば、関数が製品の総売上高を年ごとに表示する場合などに使用できます。

これらの軸は、関数を使用してマスター カレンダー テーブルに項目を作成することにより、ロード スクリプトで作成できます。あるいは、計算軸としてチャートで直接使用することもできます。

関数の例

例	結果
year('2012-10-12')	2012 を返します
year('35648')	1997 を返します (35648 = 1997-08-06 のため)

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データ ロード スクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロード スクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データ ロード エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – DateFormat データセット (スクリプト)

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Master Calendar というテーブルにロードされる日付のデータセット。
- 既定の DateFormat システム変数 MM/DD/YYYY が使用されます。
- year() 関数を使用して、追加項目 year を作成するために使用される先行する LOAD。

ロードスクリプト

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

Master_Calendar:

    Load
        date,
        year(date) as year
        ;
Load
date
Inline
[
date
12/28/2020
12/29/2020
12/30/2020
12/31/2020
01/01/2021
01/02/2021
01/03/2021
01/04/2021
01/05/2021
01/06/2021
01/07/2021
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- year

結果テーブル

日付	年
12/28/2020	2020
12/29/2020	2020
12/30/2020	2020
12/31/2020	2020
01/01/2021	2021
01/02/2021	2021
01/03/2021	2021
01/04/2021	2021
01/05/2021	2021
01/06/2021	2021
01/07/2021	2021

例 2 – ANSI 日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Master Calendar というテーブルにロードされる日付のデータセット。
- 既定の DateFormat システム変数 (MM/DD/YYYY) が使用されます。ただし、データセットに含まれる日付は、ANSI 標準日付形式です。
- 先行ロードで、これは year() 関数を使用して、追加の項目 [year] を作成します。

ロードスクリプト

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

Master_Calendar:
    Load
        date,
        year(date) as year
        ;
Load
date
Inline
[
date
2020-12-28
```

```
2020-12-29
2020-12-30
2020-12-31
2021-01-01
2021-01-02
2021-01-03
2021-01-04
2021-01-05
2021-01-06
2021-01-07
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- year

結果テーブル

日付	年
2020-12-28	2020
2020-12-29	2020
2020-12-30	2020
2020-12-31	2020
2021-01-01	2021
2021-01-02	2021
2021-01-03	2021
2021-01-04	2021
2021-01-05	2021
2021-01-06	2021
2021-01-07	2021

例 3 – 形式設定のない日付

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Master Calendar というテーブルにロードされる、数値形式の日付のデータセット。
- 既定の DateFormat システム変数 (MM/DD/YYYY) が使用されます。
- 先行ロードで、これは year() 関数を使用して、追加の項目 [year] を作成します。

元の形式設定のない unformatted_date という日付がロードされます。明確にするため、数値の日付を形式設定のある日付項目に変換するために、date() 関数を使って追加項目である [long_date] が使用されます。

ロードスクリプト

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

Master_Calendar:
    Load
        unformatted_date,
        date(unformatted_date) as long_date,
        year(unformatted_date) as year
        ;
Load
unformatted_date
Inline
[
unformatted_date
44868
44898
44928
44958
44988
45018
45048
45078
45008
45038
45068
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- unformatted_date
- long_date
- year

結果テーブル

unformatted_date	long_date	年
44868	11/03/2022	2022
44898	12/03/2022	2022
44928	01/02/2023	2023

unformatted_date	long_date	年
44958	02/01/2023	2023
44988	03/03/2023	2023
45008	03/23/2023	2023
45018	04/02/2023	2023
45038	04/22/2023	2023
45048	05/02/2023	2023
45068	05/22/2023	2023
45078	06/01/2023	2023

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

この例では、注文のデータセットが「Sales」というテーブルにロードされます。テーブルには 3 項目が含まれています。

- id
- sales_date
- amount

製品販売時の保証期間は、販売日から2年間です。タスクは、各保証の有効期限が切れる年を決定するメジャーをチャートに作成することです。

ロード スクリプト

```
Sales:
Load
id,
sales_date,
amount
Inline
[
id,sales_date,amount
1,12/28/2020,231.24,
2,12/29/2020,567.28,
3,12/30/2020,364.28,
4,12/31/2020,575.76,
5,01/01/2021,638.68,
6,01/02/2021,785.38,
7,01/03/2021,967.46,
8,01/04/2021,287.67
```

```
9,01/05/2021,764.45,
10,01/06/2021,875.43,
11,01/07/2021,957.35
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: sales_date。

次のメジャーを作成します:

```
=year(sales_date+365*2)
```

結果テーブル

sales_date	=year(sales_date+365*2)
12/28/2020	2022
12/29/2020	2022
12/30/2020	2022
12/31/2020	2022
01/01/2021	2023
01/02/2021	2023
01/03/2021	2023
01/04/2021	2023
01/05/2021	2023
01/06/2021	2023
01/07/2021	2023

このメジャーの結果は上記のテーブルに表示されています。日付に 2 年間を追加するには、365 に 2 を乗算してから、その結果を販売日付に加えます。そのため、2020 年に発生した売上は 2022 年に失効します。

yearend

この関数は、**date** を含む年の最終日の最後のミリ秒のタイムスタンプに相当する値を返します。デフォルトの出力形式は、スクリプトに設定されている **DateFormat** です。

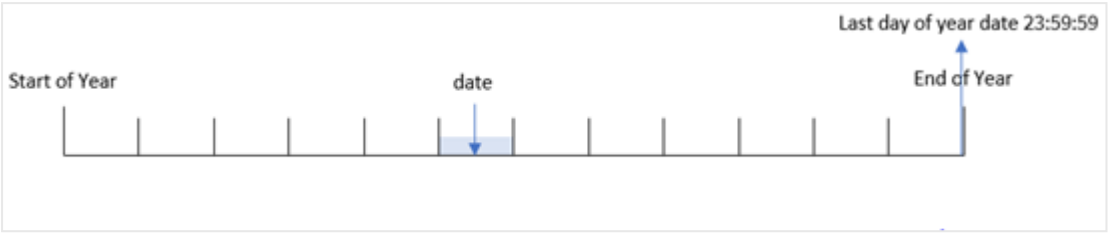
構文:

```
YearEnd( date[, period_no[, first_month_of_year = 1]])
```

つまり、yearend() 関数は、日付がどの年に該当するかを判断します。次に、その年の最後のミリ秒のタイムスタンプを日付形式で返します。年の最初の月は、既定では 1 月です。ただし、yearend() 関数で first_month_of_year 引数を使用して、どの月を最初に設定するかを変更することができます。

> *yearend() 関数は FirstMonthOfYear システム変数を考慮しません。first_month_of_year 引数を使用して変更しない限り、年は 1 月 1 日から始まります。*

yearend() 関数の図。



使用に適しているケース

yearend() 関数は、ユーザーがまだ発生していない年の端数を計算に使用する場合に、数式の一部として使用されます。たとえば、その年にまだ発生していない利息の合計を計算したい場合などに使います。

戻り値データ型： dual

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数で、値 0 は **date** を含む年を示します。**period_no** の値が負の場合は過去の年を、正の場合は将来の年を示します。
first_month_of_year	事業年度が1月以外の月に始まる場合は、**first_month_of_year** で 2 から12 の間の値を指定します。

次の値を使用して、first_month_of_year 引数に年の最初の月を設定できます。

first_month_of_year values

月	値
February	2
3 月	3
April	4
May	5
June	6
7 月	7
8 月	8
September	9
10 月	10
November	11
12 月	12

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
yearend('10/19/2001')	12/31/2001 23:59:59 を返します。
yearend('10/19/2001', -1)	12/31/2000 23:59:59 を返します。
yearend('10/19/2001', 0, 4)	03/31/2002 23:59:59 を返します。

例 1 – 追加の引数なし

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2020 年 ~ 2022 年の一連のトランザクションを含むデータセットは、「Transactions」というテーブルにロードされます。
- 日付項目は DateFormat システム変数 (MM/DD/YYYY) 形式で提供されています。
- 次を含む、先行する LOAD ステートメント:
    - [year_end] 項目として設定されている yearend() 関数。
    - [year_end_timestamp] 項目として設定されている Timestamp() 関数。

ロードスクリプト

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

Transactions:
    Load
       *,
       yearend(date) as year_end,
       timestamp(yearend(date)) as year_end_timestamp
```

```
      ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- year_end
- year_end_timestamp

結果テーブル

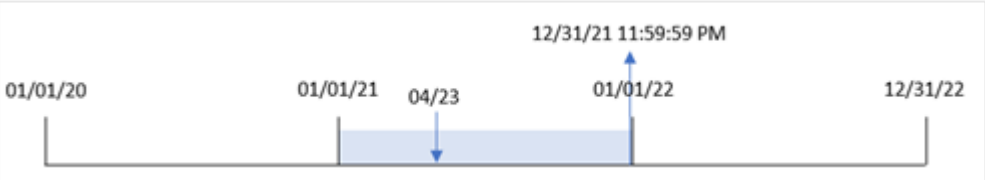
ID	日付	year_end	year_end_timestamp
8188	01/13/2020	12/31/2020	12/31/2020 11:59:59 PM
8189	02/26/2020	12/31/2020	12/31/2020 11:59:59 PM
8190	03/27/2020	12/31/2020	12/31/2020 11:59:59 PM
8191	04/16/2020	12/31/2020	12/31/2020 11:59:59 PM
8192	05/21/2020	12/31/2020	12/31/2020 11:59:59 PM
8193	08/14/2020	12/31/2020	12/31/2020 11:59:59 PM
8194	10/07/2020	12/31/2020	12/31/2020 11:59:59 PM

ID	日付	year_end	year_end_timestamp
8195	12/05/2020	12/31/2020	12/31/2020 11:59:59 PM
8196	01/22/2021	12/31/2021	12/31/2021 11:59:59 PM
8197	02/03/2021	12/31/2021	12/31/2021 11:59:59 PM
8198	03/17/2021	12/31/2021	12/31/2021 11:59:59 PM
8199	04/23/2021	12/31/2021	12/31/2021 11:59:59 PM
8200	05/04/2021	12/31/2021	12/31/2021 11:59:59 PM
8201	06/30/2021	12/31/2021	12/31/2021 11:59:59 PM
8202	07/26/2021	12/31/2021	12/31/2021 11:59:59 PM
8203	12/27/2021	12/31/2021	12/31/2021 11:59:59 PM
8204	06/06/2022	12/31/2022	12/31/2022 11:59:59 PM
8205	07/18/2022	12/31/2022	12/31/2022 11:59:59 PM
8206	11/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	12/12/2022	12/31/2022	12/31/2022 11:59:59 PM

「year_end」項目は、 yearend() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

yearend() 関数は、最初に日付値がどの年に該当するかを識別し、その年の最後のミリ秒のタイムスタンプを返します。

トランザクション 8199 が選択された yearend() 関数の図。



トランザクション 8199 は 2021 年 4 月 23 日に発生しました。yearend() 関数は、その年の最後のミリ秒、つまり12 月 31 日午後 11:59:59 を返します。

例 2 – period_no

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例のタスクは、トランザクションが発生する前の年の最終日付タイムスタンプを返す項目「previous_year_end」を作成することです。

ロード スクリプト

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

Transactions:
    Load
       *,
       yearend(date,-1) as previous_year_end,
       timestamp(yearend(date,-1)) as previous_year_end_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- previous_year_end
- previous_ year_end_timestamp

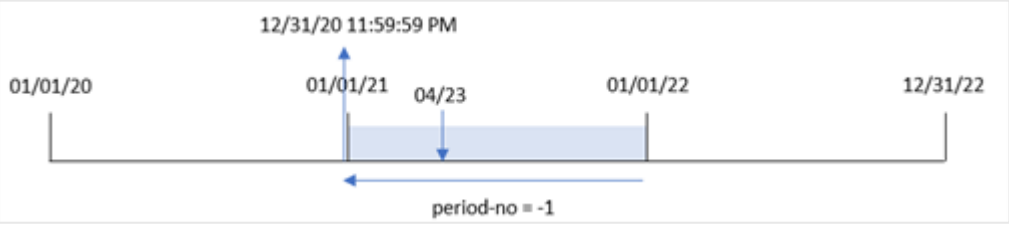
結果テーブル

ID	日付	previous_year_end	previous_year_end_timestamp
8188	01/13/2020	12/31/2019	12/31/2019 11:59:59 PM

ID	日付	previous_year_end	previous_year_end_timestamp
8189	02/26/2020	12/31/2019	12/31/2019 11:59:59 PM
8190	03/27/2020	12/31/2019	12/31/2019 11:59:59 PM
8191	04/16/2020	12/31/2019	12/31/2019 11:59:59 PM
8192	05/21/2020	12/31/2019	12/31/2019 11:59:59 PM
8193	08/14/2020	12/31/2019	12/31/2019 11:59:59 PM
8194	10/07/2020	12/31/2019	12/31/2019 11:59:59 PM
8195	12/05/2020	12/31/2019	12/31/2019 11:59:59 PM
8196	01/22/2021	12/31/2020	12/31/2020 11:59:59 PM
8197	02/03/2021	12/31/2020	12/31/2020 11:59:59 PM
8198	03/17/2021	12/31/2020	12/31/2020 11:59:59 PM
8199	04/23/2021	12/31/2020	12/31/2020 11:59:59 PM
8200	05/04/2021	12/31/2020	12/31/2020 11:59:59 PM
8201	06/30/2021	12/31/2020	12/31/2020 11:59:59 PM
8202	07/26/2021	12/31/2020	12/31/2020 11:59:59 PM
8203	12/27/2021	12/31/2020	12/31/2020 11:59:59 PM
8204	06/06/2022	12/31/2021	12/31/2021 11:59:59 PM
8205	07/18/2022	12/31/2021	12/31/2021 11:59:59 PM
8206	11/14/2022	12/31/2021	12/31/2021 11:59:59 PM
8207	12/12/2022	12/31/2021	12/31/2021 11:59:59 PM

-1 の period_no が yearend() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生した年を識別します。次に、1 年前を調べて、その年の最後のミリ秒を識別します。

period_no が -1 の yearend() 関数の図。



トランザクション 8199 は 2021 年 4 月 23 日に発生します。yearend() 関数は、「previous_year_end」項目に対して、前の年の最後のミリ秒、つまり12 月 31 日午後 11:59:59 を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただしこの例では、会社の方針により年度が 4 月 1 日に始まります。

ロードスクリプト

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

Transactions:
    Load
       *,
       yearend(date,0,4) as year_end,
       timestamp(yearend(date,0,4)) as year_end_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

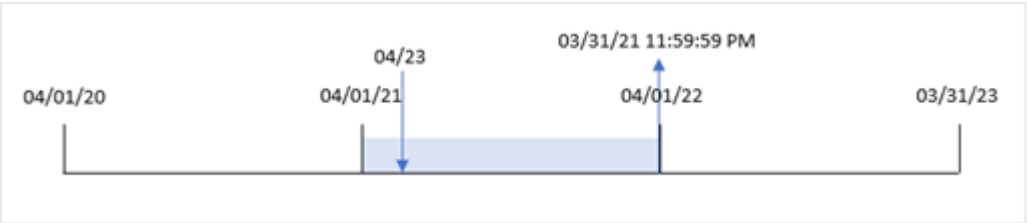
- id
- date
- year_end
- year_end_timestamp

結果テーブル

ID	日付	year_end	year_end_timestamp
8188	01/13/2020	03/31/2020	3/31/2020 11:59:59 PM
8189	02/26/2020	03/31/2020	3/31/2020 11:59:59 PM
8190	03/27/2020	03/31/2020	3/31/2020 11:59:59 PM
8191	04/16/2020	03/31/2021	3/31/2021 11:59:59 PM
8192	05/21/2020	03/31/2021	3/31/2021 11:59:59 PM
8193	08/14/2020	03/31/2021	3/31/2021 11:59:59 PM
8194	10/07/2020	03/31/2021	3/31/2021 11:59:59 PM
8195	12/05/2020	03/31/2021	3/31/2021 11:59:59 PM
8196	01/22/2021	03/31/2021	3/31/2021 11:59:59 PM
8197	02/03/2021	03/31/2021	3/31/2021 11:59:59 PM
8198	03/17/2021	03/31/2021	3/31/2021 11:59:59 PM
8199	04/23/2021	03/31/2022	3/31/2022 11:59:59 PM
8200	05/04/2021	03/31/2022	3/31/2022 11:59:59 PM
8201	06/30/2021	03/31/2022	3/31/2022 11:59:59 PM
8202	07/26/2021	03/31/2022	3/31/2022 11:59:59 PM
8203	12/27/2021	03/31/2022	3/31/2022 11:59:59 PM
8204	06/06/2022	03/31/2023	3/31/2023 11:59:59 PM
8205	07/18/2022	03/31/2023	3/31/2023 11:59:59 PM
8206	11/14/2022	03/31/2023	3/31/2023 11:59:59 PM
8207	12/12/2022	03/31/2023	3/31/2023 11:59:59 PM

yearend() 関数で 4 の first_month_of_year 引数が使用されているため、年の最初の日は 4 月 1 日、年の最後の日が 3 月 31 日に設定されます。

4 月を年の最初の月とする yearend() 関数の図。



トランザクション 8199 は 2021 年 4 月 23 日に発生します。yearend() 関数は年度初めを 4 月 1 日に設定するため、トランザクションの「year_end」値として 2022 年 3 月 31 日を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例では、データセットは変更されず、アプリケーションにロードされます。トランザクションが発生した年の終わりタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date

トランザクションが発生した年度を計算するには、次のメジャーを作成します。

- =yearend(date)
- =timestamp(yearend(date))

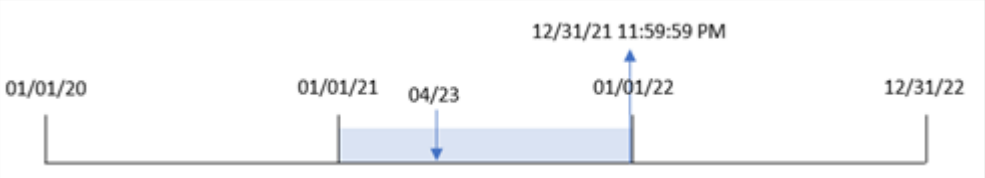
結果テーブル

ID	日付	=yearend(date)	=timestamp(yearend(date))
8188	01/13/2020	12/31/2020	12/31/2020 11:59:59 PM
8189	02/26/2020	12/31/2020	12/31/2020 11:59:59 PM
8190	03/27/2020	12/31/2020	12/31/2020 11:59:59 PM
8191	04/16/2020	12/31/2020	12/31/2020 11:59:59 PM
8192	05/21/2020	12/31/2020	12/31/2020 11:59:59 PM
8193	08/14/2020	12/31/2020	12/31/2020 11:59:59 PM
8194	10/07/2020	12/31/2020	12/31/2020 11:59:59 PM
8195	12/05/2020	12/31/2020	12/31/2020 11:59:59 PM
8196	01/22/2021	12/31/2021	12/31/2021 11:59:59 PM
8197	02/03/2021	12/31/2021	12/31/2021 11:59:59 PM
8198	03/17/2021	12/31/2021	12/31/2021 11:59:59 PM
8199	04/23/2021	12/31/2021	12/31/2021 11:59:59 PM
8200	05/04/2021	12/31/2021	12/31/2021 11:59:59 PM
8201	06/30/2021	12/31/2021	12/31/2021 11:59:59 PM
8202	07/26/2021	12/31/2021	12/31/2021 11:59:59 PM
8203	12/27/2021	12/31/2021	12/31/2021 11:59:59 PM
8204	06/06/2022	12/31/2022	12/31/2022 11:59:59 PM
8205	07/18/2022	12/31/2022	12/31/2022 11:59:59 PM
8206	11/14/2022	12/31/2022	12/31/2022 11:59:59 PM
8207	12/12/2022	12/31/2022	12/31/2022 11:59:59 PM

「end_of_year」メジャーは、yearend() 関数を使用し、関数の引数として日付項目を渡すことにより、チャートオブジェクトで作成されます。

yearend() 関数は、最初に日付値がどの年に該当するかを識別し、その年の最後のミリ秒のタイムスタンプを返します。

トランザクション 8199 が 4 月に発生したことを示す yearend() 関数の図。



トランザクション 8199 は 2021 年 4 月 23 日に発生します。yearend() 関数は、その年の最後のミリ秒、つまり 12 月 31 日午後 11:59:59 を返します。

例 5 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 「Employee_Expenses」というテーブルにロードされるデータセット。テーブルには次の項目が含まれています。
    - 従業員 ID
    - 従業員名
    - 各従業員の平均日次経費請求

エンド ユーザーは、従業員 ID と従業員名別に、その年の残りの期間にまだ発生する推定経費請求を表示するグラフ オブジェクトを求めています。会計年度は 1 月に始まります。

ロード スクリプト

```
Employee_Expenses:
Load
*
Inline
[
employee_id,employee_name,avg_daily_claim
182,Mark, $15
183,Deryck, $12.5
184,Dexter, $12.5
185,Sydney,$27
186,Agatha,$18
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- employee_id
- employee_name

予想される経費請求を計算するには、次のメジャーを作成します。

```
=(yearend(today(1))-today(1))*avg_daily_claim
```

メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

employee_id	employee_name	=(yearend(today(1))-today(1))*avg_daily_claim
182	Mark	$3240.00
183	Deryck	$2700.00
184	Dexter	$2700.00
185	Sydney	$5832.00
186	Agatha	$3888.00

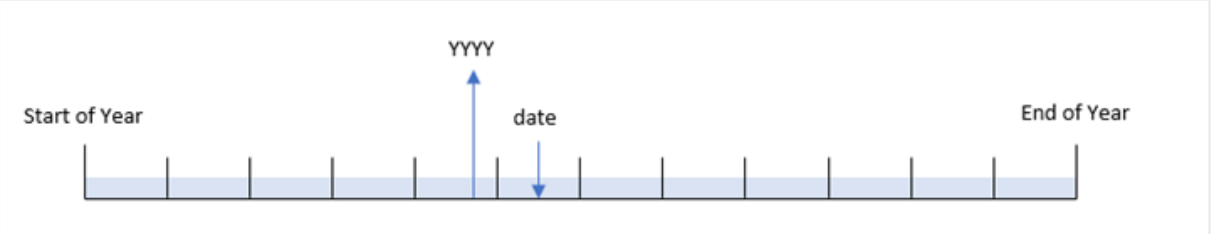
yearend() 関数は、今日の日付を唯一の引数として使用することにより、現在の年の終了日を返します。次に、その年の終了日から今日の日付を引くことによって、数式は今年の残りの日数を返します。

次に、この値に各従業員による1日あたりの平均経費請求額を乗算して、年度の残り期間に各従業員が行うと予想される請求の推定額を計算します。

yearname

この関数は、**date** を含む年の初日の最初のミリ秒のタイムスタンプに対応する数値を基底として、4桁の年の表示値を返します。

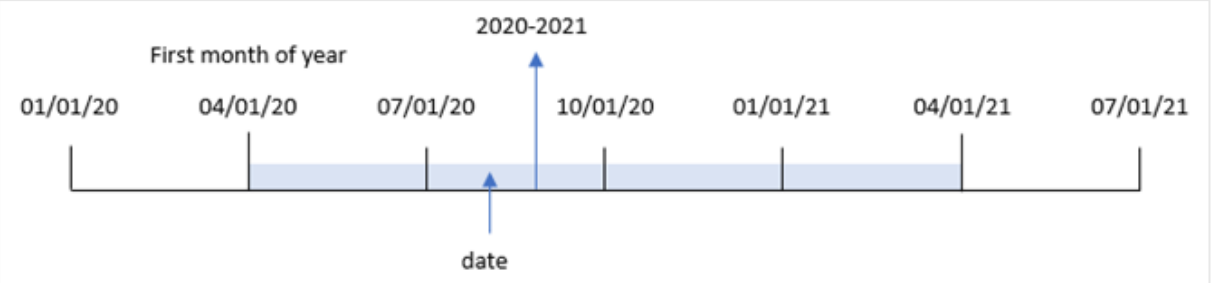
yearname() 関数の時間範囲の図。



yearname() 関数は、評価する日付をオフセットし、年の最初の月を設定できるという点で、year() 関数とは異なります。

年の最初の月が1月でない場合、この関数は、その日付を含む12か月期間を表す2つの4桁の年を返します。たとえば、年度の始まりが4月で、評価される日付が06/30/2020の場合、返される結果は2020-2021になります。

4 月が年の最初の月に設定された yearname() 関数の図。



構文:

```
YearName(date[, period_no[, first_month_of_year]] )
```

戻り値データ型: dual

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数で、値 0 は **date** を含む年を示します。**period_no** の値が負の場合は過去の年を、正の場合は将来の年を示します。
first_month_of_year	事業年度が 1 月以外の月に始まる場合は、**first_month_of_year** で 2 から 12 の間の値を指定します。表示値は、2 年を表す文字列になります。

次の値を使用して、first_month_of_year 引数に年の最初の月を設定できます。

first_month_of_year values

月	値
February	2
3 月	3
April	4
May	5
June	6
7 月	7
8 月	8
September	9
10 月	10
November	11
12 月	12

使用に適しているケース

yearname() 関数は、集計を年ごとに比較するのに役立ちます。たとえば、製品の総売上高を年ごとに表示する場合などが考えられます。

これらの軸は、関数を使用してマスター カレンダー テーブルに項目を作成することにより、ロード スクリプトで作成できます。また、計算軸としてチャートで作成することもできます

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データ ロード スクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロード スクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データ ロード エディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
yearname('10/19/2001')	「2001」を返します。
yearname('10/19/2001',-1)	「2000」を返します。
yearname('10/19/2001',0,4)	「2001-2002」を返します。

関連トピック

トピック	説明
year (page 1109)	この関数は、expression が標準的な数値の解釈に従って日付と判断される場合に、年を表す整数を返します。

例 1 – 追加の引数なし

ロード スクリプトと結果

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 2020 年 ~ 2022 年の一連のトランザクションを含むデータセットは、「Transactions」というテーブルにロードされます。
- 「MM/DD/YYYY」に設定された DateFormat システム変数。
- year_name 項目として設定されている yearname() 関数を使用する先行する LOAD。

ロード スクリプト

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

Transactions:
    Load
      *,
      yearname(date) as year_name
      ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- year_name

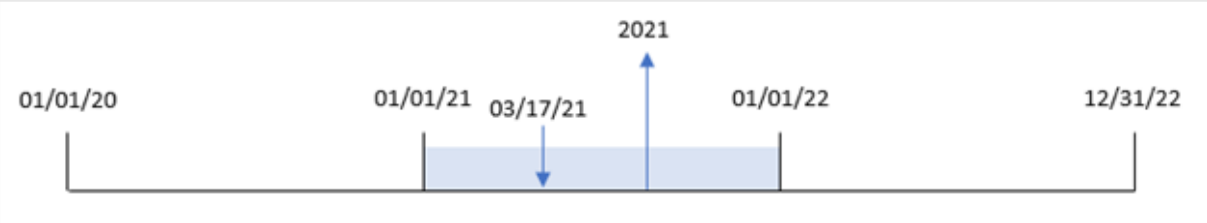
結果テーブル

日付	year_name
01/13/2020	2020
02/26/2020	2020
03/27/2020	2020
04/16/2020	2020
05/21/2020	2020
08/14/2020	2020
10/07/2020	2020
12/05/2020	2020
01/22/2021	2021
02/03/2021	2021
03/17/2021	2021
04/23/2021	2021
05/04/2021	2021
06/30/2021	2021
07/26/2021	2021
12/27/2021	2021
06/06/2022	2022
07/18/2022	2022
11/14/2022	2022
12/12/2022	2022

「year_name」項目は、yearname() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

yearname() 関数は、日付値がどの年に該当するかを識別し、これを 4 桁の年の値として返します。

2021 を年の値として示す yearname() 関数の図。



例 2 – period_no

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2020 年 ~ 2022 年の一連のトランザクションを含むデータセットは、「Transactions」というテーブルにロードされます。
- 「MM/DD/YYYY」に設定された DateFormat システム変数。
- year_name 項目として設定されている yearname() 関数を使用する先行ロード。

ロードスクリプト

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

Transactions:
    Load
      *,
      yearname(date,-1) as prior_year_name
      ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

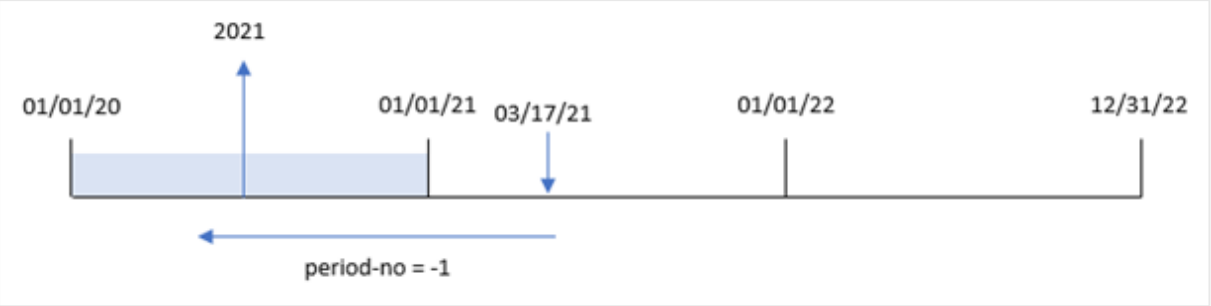
- date
- prior_year_name

結果テーブル

日付	prior_year_name
01/13/2020	2019
02/26/2020	2019
03/27/2020	2019
04/16/2020	2019
05/21/2020	2019
08/14/2020	2019
10/07/2020	2019
12/05/2020	2019
01/22/2021	2020
02/03/2021	2020
03/17/2021	2020
04/23/2021	2020
05/04/2021	2020
06/30/2021	2020
07/26/2021	2020
12/27/2021	2020
06/06/2022	2021
07/18/2022	2021
11/14/2022	2021
12/12/2022	2021

-1 の period_no が yearname() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生する年を識別します。次に、関数は 1 年前にシフトし、結果の年を返します。

period_no が -1 に設定された yearname() 関数の図。



例 3 – first_month_of_year

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセット。
- 「MM/DD/YYYY」に設定された DateFormat システム変数。
- year_name 項目として設定されている yearname() 関数を使用する先行ロード。

ロードスクリプト

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

Transactions:
    Load
      *,
      yearname(date,0,4) as year_name
      ;
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
```

```
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- year_name

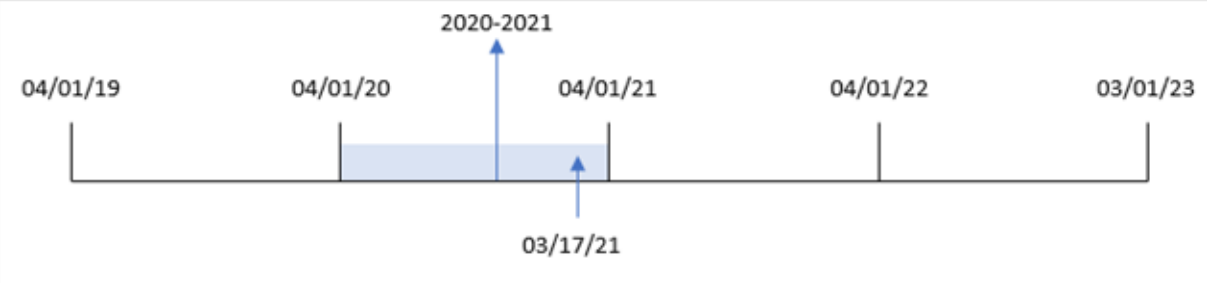
結果テーブル

日付	year_name
01/13/2020	2019-2020
02/26/2020	2019-2020
03/27/2020	2019-2020
04/16/2020	2020-2021
05/21/2020	2020-2021
08/14/2020	2020-2021
10/07/2020	2020-2021
12/05/2020	2020-2021
01/22/2021	2020-2021
02/03/2021	2020-2021
03/17/2021	2020-2021
04/23/2021	2021-2022
05/04/2021	2021-2022
06/30/2021	2021-2022
07/26/2021	2021-2022
12/27/2021	2021-2022
06/06/2022	2022-2023
07/18/2022	2022-2023
11/14/2022	2022-2023
12/12/2022	2022-2023

4 の first_month_of_year 引数が yearname() 関数で使用されているため、年度の始めが 1 月 1 日から 4 月 1 日に移動します。したがって、各 12 か月は 2 つの暦年にまたがり、yearname() 関数は評価される日付の年に対して 2 つの 4 桁の数字を返します。

トランザクション 8198 は 2021 年 3 月 17 日に発生します。yearname() 関数は、年度初めを 4 月 1 日に、終わりを 3 月 30 日に設定します。したがって、トランザクション 8198 は、2020 年 4 月 1 日 ~ 2021 年 3 月 30 日の期間に発生したことになります。その結果、yearname() 関数は値 2020-2021 を返します。

3 月が年の最初の月に設定された yearname() 関数の図。



例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- 最初の例と同じデータセット。
- 「MM/DD/YYYY」に設定された DateFormat システム変数。

ただし、トランザクションが発生した年を返す項目は、チャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
```

```
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します:

`date`

「`year_name`」項目を計算するには、次のメジャーを作成します。

```
=yearname(date)
```

結果テーブル

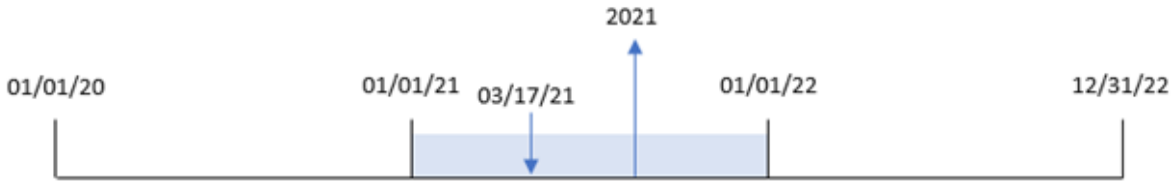
日付	=yearname(date)
01/13/2020	2020
02/26/2020	2020
03/27/2020	2020
04/16/2020	2020
05/21/2020	2020
08/14/2020	2020
10/07/2020	2020
12/05/2020	2020
01/22/2021	2021
02/03/2021	2021
03/17/2021	2021
04/23/2021	2021
05/04/2021	2021
06/30/2021	2021
07/26/2021	2021
12/27/2021	2021

日付	=yearname(date)
06/06/2022	2022
07/18/2022	2022
11/14/2022	2022
12/12/2022	2022

「year_name」メジャーは、yearname() 関数を使用し、関数の引数として日付項目を渡すことにより、チャートオブジェクトで作成されます。

関数は、日付値がどの年に該当するかを識別し、これを 4 桁の年の値として返します。yearname()

2021 が年の値に設定された yearname() 関数の図。



例 5 – シナリオ

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセット。
- 「MM/DD/YYYY」に設定された DateFormat システム変数。

エンドユーザーは、トランザクションの四半期ごとの総売上高を示すチャートを求めています。yearname() 軸がデータモデルで使用できない場合は、yearname() 関数を計算軸として使用してこのチャートを作成します。

ロードスクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,'01/13/2020',37.23
8189,'02/26/2020',17.17
8190,'03/27/2020',88.27
```

```
8191,'04/16/2020',57.42
8192,'05/21/2020',53.80
8193,'08/14/2020',82.06
8194,'10/07/2020',40.39
8195,'12/05/2020',87.21
8196,'01/22/2021',95.93
8197,'02/03/2021',45.89
8198,'03/17/2021',36.23
8199,'04/23/2021',25.66
8200,'05/04/2021',82.77
8201,'06/30/2021',69.98
8202,'07/26/2021',76.11
8203,'12/27/2021',25.12
8204,'06/06/2022',46.23
8205,'07/18/2022',84.21
8206,'11/14/2022',96.24
8207,'12/12/2022',67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成します。

集計を年別に比較するには、次の計算軸を作成します:

```
=yearname(date)
```

このメジャーを作成します:

```
=sum(amount)
```

メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

yearname(date)	=sum(amount)
2020	$463.55
2021	$457.69
2022	$294.35

yearstart

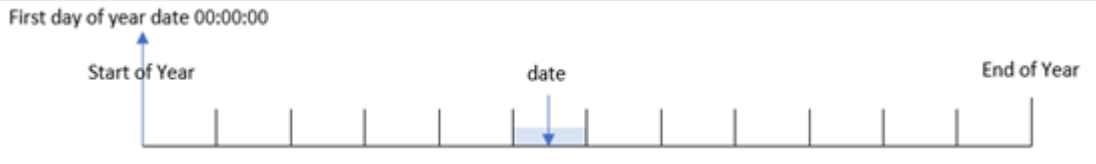
この関数は、**date**を含む年の最初の日の開始に対応するタイムスタンプを返します。デフォルトの出力形式は、スクリプトに設定されている**DateFormat** です。

構文:

```
YearStart(date[, period_no[, first_month_of_year]])
```

つまり、yearstart() 関数は、日付がどの年に該当するかを判断します。次に、その年の最初のミリ秒のタイムスタンプを日付形式で返します。年の最初の月は既定で1月に設定されていますが、yearstart() 関数で first_month_of_year 引数を使用して、どの月を最初に設定するかを変更することができます。

関数がカバーできる時間の範囲を示す yearstart() 関数の図。



使用に適しているケース

yearstart() 関数は、すでに経過した年の端数を計算に使用する場合に、数式の一部として使用されます。たとえば、年初来に累積した利息を計算する場合などです。

戻り値データ型： dual

引数

引数	説明
date	評価する日付またはタイムスタンプ。
period_no	**period_no** は整数で、値 0 は **date** を含む年を示します。**period_no** の値が負の場合は過去の年を、正の場合は将来の年を示します。
first_month_of_year	事業年度が1月以外の月に始まる場合は、**first_month_of_year** で 2 から12 の間の値を指定します。

次の月は first_month_of_year argumentで使用できます。:

first_month_of_year values

月	値
2 月	2
3 月	3
April	4
May	5
June	6
7 月	7
8 月	8
September	9
10 月	10
11 月	11
12 月	12

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザーインターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

関数の例

例	結果
yearstart('10/19/2001')	Returns 01/01/2001 00:00:00.
yearstart('10/19/2001',-1)	Returns 01/01/2000 00:00:00.
yearstart('10/19/2001',0,4)	Returns 04/01/2001 00:00:00.

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 2020 年 ~ 2022 年の一連のトランザクションを含むデータセットは、「Transactions」というテーブルにロードされます。
- 日付項目は DateFormat システム変数 (MM/DD/YYYY) 形式で提供されています。
- 次を含む、先行する LOAD ステートメント:
  - [year_start] 項目として設定されている yearstart() 関数。
  - [year_start_timestamp] 項目として設定されている Timestamp() 関数

ロードスクリプト

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

Transactions:
    Load
       *,
       yearstart(date) as year_start,
       timestamp(yearstart(date)) as year_start_timestamp
       ;
```

```
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- year_start
- year_start_timestamp

結果テーブル

ID	日付	year_start	year_start_timestamp
8188	01/13/2020	01/01/2020	1/1/2020 12:00:00 AM
8189	02/26/2020	01/01/2020	1/1/2020 12:00:00 AM
8190	03/27/2020	01/01/2020	1/1/2020 12:00:00 AM
8191	04/16/2020	01/01/2020	1/1/2020 12:00:00 AM
8192	05/21/2020	01/01/2020	1/1/2020 12:00:00 AM
8193	08/14/2020	01/01/2020	1/1/2020 12:00:00 AM
8194	10/07/2020	01/01/2020	1/1/2020 12:00:00 AM

ID	日付	year_start	year_start_timestamp
8195	12/05/2020	01/01/2020	1/1/2020 12:00:00 AM
8196	01/22/2021	01/01/2021	1/1/2021 12:00:00 AM
8197	02/03/2021	01/01/2021	1/1/2021 12:00:00 AM
8198	03/17/2021	01/01/2021	1/1/2021 12:00:00 AM
8199	04/23/2021	01/01/2021	1/1/2021 12:00:00 AM
8200	05/04/2021	01/01/2021	1/1/2021 12:00:00 AM
8201	06/30/2021	01/01/2021	1/1/2021 12:00:00 AM
8202	07/26/2021	01/01/2021	1/1/2021 12:00:00 AM
8203	12/27/2021	01/01/2021	1/1/2021 12:00:00 AM
8204	06/06/2022	01/01/2022	1/1/2022 12:00:00 AM
8205	07/18/2022	01/01/2022	1/1/2022 12:00:00 AM
8206	11/14/2022	01/01/2022	1/1/2022 12:00:00 AM
8207	12/12/2022	01/01/2022	1/1/2022 12:00:00 AM

「year_start」項目は、yearstart() 関数を使用し、関数の引数として日付項目を渡すことにより、前の load ステートメントで作成されます。

yearstart() 関数は、最初に日付値がどの年に該当するかを識別し、その年の最初のミリ秒のタイムスタンプを返します。

yearstart() 関数とトランザクション 8199 の図。

トランザクション 8199 は 2021 年 4 月 23 日に発生しました。yearstart() 関数は、その年の最後のミリ秒、つまり1 月 1 日午前 12:00:00 を返します。

例 2 – period_no

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例のタスクは、トランザクションが発生する前の年の開始日付タイムスタンプを返す項目「previous_year_start」を作成することです。

ロード スクリプト

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

Transactions:
    Load
       *,
       yearstart(date,-1) as previous_year_start,
       timestamp(yearstart(date,-1)) as previous_year_start_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date
- previous_year_start
- previous_ year_start_timestamp

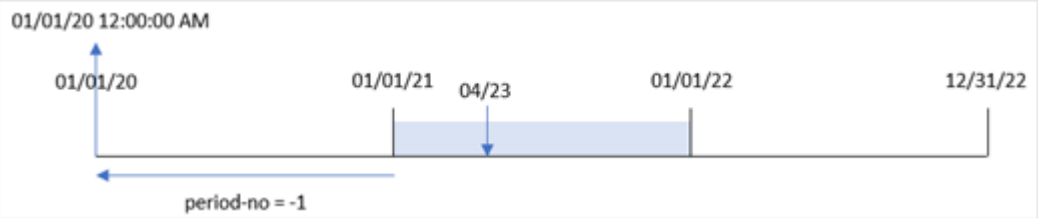
結果テーブル

ID	日付	previous_year_start	previous_year_start_timestamp
8188	01/13/2020	01/01/2019	1/1/2019 12:00:00 AM

ID	日付	previous_year_start	previous_year_start_timestamp
8189	02/26/2020	01/01/2019	1/1/2019 12:00:00 AM
8190	03/27/2020	01/01/2019	1/1/2019 12:00:00 AM
8191	04/16/2020	01/01/2019	1/1/2019 12:00:00 AM
8192	05/21/2020	01/01/2019	1/1/2019 12:00:00 AM
8193	08/14/2020	01/01/2019	1/1/2019 12:00:00 AM
8194	10/07/2020	01/01/2019	1/1/2019 12:00:00 AM
8195	12/05/2020	01/01/2019	1/1/2019 12:00:00 AM
8196	01/22/2021	01/01/2020	1/1/2020 12:00:00 AM
8197	02/03/2021	01/01/2020	1/1/2020 12:00:00 AM
8198	03/17/2021	01/01/2020	1/1/2020 12:00:00 AM
8199	04/23/2021	01/01/2020	1/1/2020 12:00:00 AM
8200	05/04/2021	01/01/2020	1/1/2020 12:00:00 AM
8201	06/30/2021	01/01/2020	1/1/2020 12:00:00 AM
8202	07/26/2021	01/01/2020	1/1/2020 12:00:00 AM
8203	12/27/2021	01/01/2020	1/1/2020 12:00:00 AM
8204	06/06/2022	01/01/2021	1/1/2021 12:00:00 AM
8205	07/18/2022	01/01/2021	1/1/2021 12:00:00 AM
8206	11/14/2022	01/01/2021	1/1/2021 12:00:00 AM
8207	12/12/2022	01/01/2021	1/1/2021 12:00:00 AM

この例では、-1 の period_no が yearstart() 関数でオフセット引数として使用されたため、関数は最初にトランザクションが発生する年を識別します。次に、1 年前を調べて、その年の最初のミリ秒を識別します。

period_no が -1 の yearstart() 関数の図。



トランザクション 8199 は 2021 年 4 月 23 日に発生しました。yearstart() 関数は、「previous_year_start」項目に対して、前の年の最初のミリ秒、つまり1 月 1 日午前 12:00:00 を返します。

例 3 – first_month_of_year

ロードスクリプトと結果

概要

最初の例と同じデータセットとシナリオが使用されます。

ただしこの例では、会社の方針により年度が 4 月 1 日に始まります。

ロードスクリプト

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

Transactions:
    Load
       *,
       yearstart(date,0,4) as year_start,
       timestamp(yearstart(date,0,4)) as year_start_timestamp
       ;
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

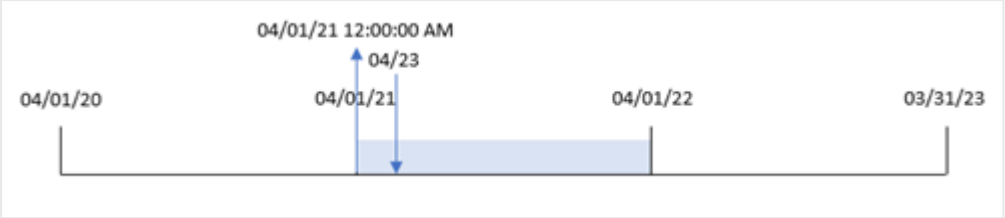
- id
- date
- year_start
- year_start_timestamp

結果テーブル

ID	日付	year_start	year_start_timestamp
8188	01/13/2020	04/01/2019	4/1/2019 12:00:00 AM
8189	02/26/2020	04/01/2019	4/1/2019 12:00:00 AM
8190	03/27/2020	04/01/2019	4/1/2019 12:00:00 AM
8191	04/16/2020	04/01/2020	4/1/2020 12:00:00 AM
8192	05/21/2020	04/01/2020	4/1/2020 12:00:00 AM
8193	08/14/2020	04/01/2020	4/1/2020 12:00:00 AM
8194	10/07/2020	04/01/2020	4/1/2020 12:00:00 AM
8195	12/05/2020	04/01/2020	4/1/2020 12:00:00 AM
8196	01/22/2021	04/01/2020	4/1/2020 12:00:00 AM
8197	02/03/2021	04/01/2020	4/1/2020 12:00:00 AM
8198	03/17/2021	04/01/2020	4/1/2020 12:00:00 AM
8199	04/23/2021	04/01/2021	4/1/2021 12:00:00 AM
8200	05/04/2021	04/01/2021	4/1/2021 12:00:00 AM
8201	06/30/2021	04/01/2021	4/1/2021 12:00:00 AM
8202	07/26/2021	04/01/2021	4/1/2021 12:00:00 AM
8203	12/27/2021	04/01/2021	4/1/2021 12:00:00 AM
8204	06/06/2022	04/01/2022	4/1/2022 12:00:00 AM
8205	07/18/2022	04/01/2022	4/1/2022 12:00:00 AM
8206	11/14/2022	04/01/2022	4/1/2022 12:00:00 AM
8207	12/12/2022	04/01/2022	4/1/2022 12:00:00 AM

この例では、yearstart() 関数で 4 の first_month_of_year 引数が使用されているため、年の最初の日は 4 月 1 日、年の最後の日が 3 月 31 日に設定されます。

4 月が最初の月に設定された yearstart() 関数の図。



トランザクション 8199 は 2021 年 4 月 23 日に発生しました。yearstart() 関数は年度初めを 4 月 1 日に設定するため、トランザクションの「year_start」値として 4 月 1 日を返します。

例 4 – チャート オブジェクトの例

ロード スクリプトとチャートの数式

概要

最初の例と同じデータセットとシナリオが使用されます。

ただし、この例では、データセットは変更されず、アプリケーションにロードされます。トランザクションが発生した年の初めのタイムスタンプを返す計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,01/13/2020,37.23
8189,02/26/2020,17.17
8190,03/27/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,06/06/2022,46.23
8205,07/18/2022,84.21
8206,11/14/2022,96.24
8207,12/12/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- id
- date

トランザクションが発生した年度を計算するには、次のメジャーを作成します。

- =yearstart(date)
- =timestamp(yearstart(date))

結果テーブル

ID	日付	=yearstart(date)	=timestamp(yearstart(date))
8188	06/06/2022	01/01/2022	1/1/2022 12:00:00 AM
8189	07/18/2022	01/01/2022	1/1/2022 12:00:00 AM
8190	11/14/2022	01/01/2022	1/1/2022 12:00:00 AM
8191	12/12/2022	01/01/2022	1/1/2022 12:00:00 AM
8192	01/22/2021	01/01/2021	1/1/2021 12:00:00 AM
8193	02/03/2021	01/01/2021	1/1/2021 12:00:00 AM
8194	03/17/2021	01/01/2021	1/1/2021 12:00:00 AM
8195	04/23/2021	01/01/2021	1/1/2021 12:00:00 AM
8196	05/04/2021	01/01/2021	1/1/2021 12:00:00 AM
8197	06/30/2021	01/01/2021	1/1/2021 12:00:00 AM
8198	07/26/2021	01/01/2021	1/1/2021 12:00:00 AM
8199	12/27/2021	01/01/2021	1/1/2021 12:00:00 AM
8200	01/13/2020	01/01/2020	1/1/2020 12:00:00 AM
8201	02/26/2020	01/01/2020	1/1/2020 12:00:00 AM
8202	03/27/2020	01/01/2020	1/1/2020 12:00:00 AM
8203	04/16/2020	01/01/2020	1/1/2020 12:00:00 AM
8204	05/21/2020	01/01/2020	1/1/2020 12:00:00 AM
8205	08/14/2020	01/01/2020	1/1/2020 12:00:00 AM
8206	10/07/2020	01/01/2020	1/1/2020 12:00:00 AM
8207	12/05/2020	01/01/2020	1/1/2020 12:00:00 AM

「start_of_year」メジャーは、yearstart() 関数を使用し、関数の引数として日付項目を渡すことにより、チャートオブジェクトで作成されます。

yearstart() 関数は、最初に日付値がどの年に該当するかを識別し、その年の最初のミリ秒のタイムスタンプを返します。

yearstart() 関数とトランザクション 8199 の図。

トランザクション 8199 は 2021 年 4 月 23 日に発生しました。yearstart() 関数は、その年の最後のミリ秒、つまり1月 1 日午前 12:00:00 を返します。

例 5 – シナリオ

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 「Loans」というテーブルにロードされるデータセット。テーブルには次の項目が含まれています。
  - ローン ID。
  - 年度初めの残高。
  - 年間の各ローンに課される単利率。

エンドユーザーは、年初来の各ローンで発生した現在の利息をローン ID 別に表示するチャートオブジェクトを求めています。

ロードスクリプト

```
Loans:
Load
*
Inline
[
loan_id,start_balance,rate
8188,$10000.00,0.024
8189,$15000.00,0.057
8190,$17500.00,0.024
8191,$21000.00,0.034
8192,$90000.00,0.084
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- loan_id
- start_balance

累積利息を計算するには、次のメジャーを作成します。

```
=start_balance*(rate*(today(1)-yearstart(today(1)))/365)
```

メジャーの **[数値書式]** を **[通貨]** に設定します。

結果テーブル

loan_id	start_balance	=start_balance*(rate*(today(1)-yearstart(today(1)))/365)
8188	$10000.00	$39.73
8189	$15000.00	$339.66
8190	$17500.00	$166.85
8191	$21000.00	$283.64
8192	$90000.00	$3003.29

yearstart() 関数は、今日の日付を唯一の引数として使用することにより、現在の年の開始日を返します。その結果を現在の日付から減算することにより、数式は今年経過した日数を返します。

次に、この値に利率を乗算して 365 で除算すると、その期間の実効利率が返されます。次に、その期間の実効利率にローンの開始残高を掛けると、今年これまでに発生した利息を返されます。

yeartodate

この関数は、入力したタイムスタンプがスクリプトが最後にロードされた日付の年に該当するかどうかを算出し、該当する場合は True を返し、該当しない場合は False を返します。

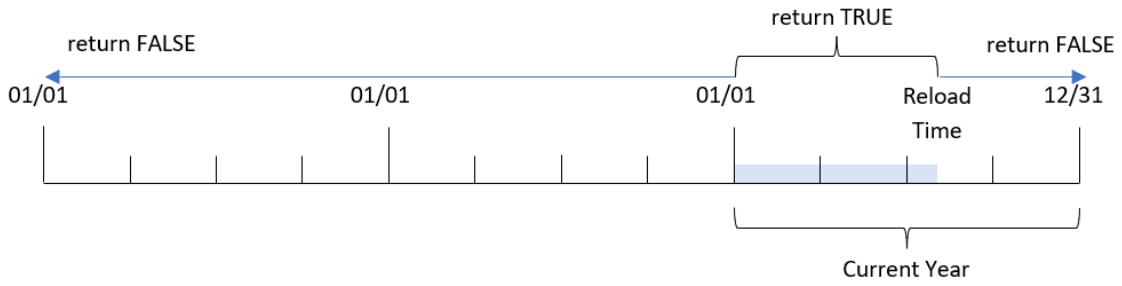
構文:

```
YearToDate(timestamp[ , yearoffset [ , firstmonth [ , todaydate] ] ])
```

戻り値データ型: ブール値

> *Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。*

yeartodate() 関数の図の例



オプションのパラメータがどれも使用されていない場合、年初から当日までとは、さかのぼって直近の1月1日からスクリプトを最後に実行した日付までを含む1暦年以内のいずれかの日付を意味します。

つまり、yeartodate() 関数が、追加パラメータなしでトリガーされると、タイムスタンプを評価するために使用され、日付がリロードが発生した日付までのカレンダー年に発生したかどうかに基づいてブール値結果を返します。

ただし、firstmonth 引数を使用して年の開始日を上書きすることも、yearoffset 引数を使って前後の年と比較することもできます。

最後に、過去のデータセットの場合、yeartodate() 関数は todaydate を設定するパラメータを提供し、これによりタイムスタンプが todaydate 引数で提供された日付までのカレンダー年と比較されます。

引数

引数	説明
timestamp	評価するタイムスタンプ (例: '10/12/2012')。
yearoffset	**yearoffset** を指定することで、**yeartodate** は、別の年の同じ期間について True を返します。負の値の **yearoffset** は過去の年を示し、正の値の場合は将来の年を示します。前年1年間を取得するときは、yearoffset = -1 を指定します。省略された場合は、0 として処理されます。
firstmonth	**firstmonth** を1~12で指定することにより(省略した場合は1)、年度の始めを任意の月の1日に動かすことができます。例えば、会計年度を5月1日から開始する場合には、**firstmonth** = 5 と指定します。値1は、1月1日に始まる会計年度を示し、値12は12月1日に始まる会計年度を示します。
todaydate	**todaydate** を指定することにより(省略した場合は最後にスクリプトを実行したタイムスタンプ)、期間の上限として使用する日付を移動できます。

使用に適しているケース

yeartodate() 関数はブール値の結果を返します。通常、このタイプの関数は IF 式の条件として使用されます。これにより、評価された日付が、アプリケーションの前回のリロード日付を含む年度に発生したかどうかに応じて、集計または計算を返します。

例えば、YearToDate() 関数を使用して、現在の週のこれまでに製造されたすべての機器を識別することができます。

次の例では、最後のリロード時を11/18/2011としています。

関数の例

例	結果
`yeartodate( '11/18/2010')`	の戻り値: False
`yeartodate( '02/01/2011')`	の戻り値: True
`yeartodate( '11/18/2011')`	の戻り値: True
`yeartodate( '11/19/2011')`	の戻り値: False
`yeartodate( '11/19/2011', 0, 1, '12/31/2011')`	の戻り値: True

例	結果
yeartodate('11/18/2010', -1)	の戻り値: True
yeartodate('11/18/2011', -1)	の戻り値: False
yeartodate('04/30/2011', 0, 5)	の戻り値: False
yeartodate('05/01/2011', 0, 5)	の戻り値: True

地域の設定

特に指定のない限り、このトピックの例では次の日付書式を使用しています: MM/DD/YYYY。日付書式は、データロードスクリプトの SET DateFormat ステートメントで指定されています。既定の日付書式は、地域の設定やその他の要因により、システムによって異なる場合があります。以下の例の書式は、要件に合わせて変更できます。または、これらの例に一致するようにロードスクリプトの書式を変更できます。

App の既定の地域設定は、Qlik Sense がインストールされているコンピューターまたはサーバーの地域システム設定に基づいています。アクセスしている Qlik Sense サーバーがスウェーデンに設定されてい場合、データロードエディターは、日付、時間、および通貨にスウェーデンの地域設定を使用します。これらの地域の書式設定は、Qlik Sense ユーザー インターフェースに表示される言語とは関係ありません。Qlik Sense は使用しているブラウザと同じ言語で表示されます。

例 1 – 基本的な例

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。
- 前回のリロード日付までのそのカレンダー年に発生したトランザクションを決定する項目 year_to_date の作成。

書き込み時には、日付は 2022 年 4 月 26 日です。

ロードスクリプト

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

Transactions:
    Load
        *,
        yeartodate(date) as year_to_date
        ;
```

```
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

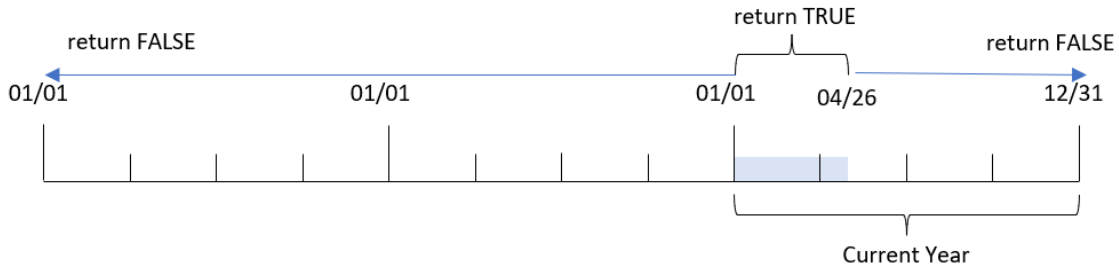
- date
- year_to_date

結果テーブル

日付	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0

日付	year_to_date
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

yeartodate() 関数の図、基本的な例



year_to_date 項目は、yeartodate() 関数を使用し、関数の引数として date 項目を渡すことにより、先行するLOAD ステートメントで作成されます。

それ以外のパラメータは関数に渡されないため、yeartodate() 関数は最初にリロード日付、そして現在のカレンダー年 (1 月 1 日開始) の境界を特定し、ブール値結果 TRUE を返します。

したがって、1 月 1 日 ~ 4 月 26 日に発生したトランザクションの場合、リロード日付は TRUEのブール値の結果を返します。2022 年始めの前に発生したトランザクションは、ブール値結果 FALSE を返します。

例 2 – yearoffset

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- カレンダー年初来前 2 年間全体に発生したトランザクションを決定する項目 [two_years_prior] の作成。

ロード スクリプト

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

Transactions:
    Load
        *,
        yeartodate(date,-2) as two_years_prior
        ;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- two_years_prior

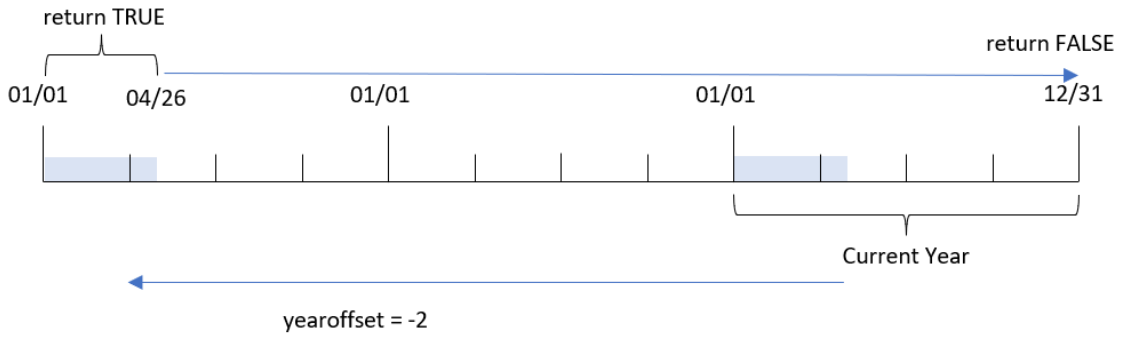
結果テーブル

日付	two_years_prior
01/10/2020	-1

日付	two_years_prior
02/28/2020	-1
04/09/2020	-1
04/16/2020	-1
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	0
02/26/2022	0
03/07/2022	0
03/11/2022	0

yeartodate() 関数で -2 を yearoffset 引数として使用することにより、関数は、比較対象のカレンダー年の境界を 2 年間ずらします。最初、年セグメントは 2022 年 1 月 1 日 ~ 4 月 26 日に相当します。yearoffset 引数は次にこのセグメントを 2 年前にオフセットします。そうすると、日付の境界線は 2020 年 1 月 1 日 ~ 4 月 26 日に入ります。

yeartodate() 関数の図、yearoffset の例



したがって、1 月 1 日 ~ 4 月 26 日に発生したトランザクションは、TRUEのブール値の結果を返します。このセグメントの前後に発生されるトランザクションは FALSE を返します。

例 3 – firstmonth

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 前回のリロード日付までのそのカレンダー年に発生したトランザクションを決定する項目 [year_to_date] の作成。

この例では、会計年度の開始を 7 月 1 日に設定します。

ロードスクリプト

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

Transactions:
    Load
        *,
        yeartodate(date,0,7) as year_to_date
        ;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
```

```
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- year_to_date

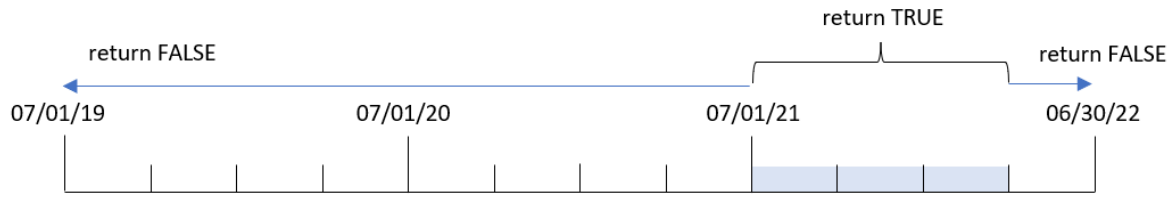
結果テーブル

日付	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0

日付	year_to_date
07/26/2021	-1
12/27/2021	-1
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

この例では、yeartodate() 関数で firstmonth 引数 7 が使用されているため、年の最初の日は 7 月 1 日、年の最後の日が 6 月 30 日に設定されます。

yeartodate() 関数、firstmonth 例の図



したがって、2021 年 7 月 1 日 ~ 2022 年 4 月 26 日に発生したトランザクションの場合、リロード日付は TRUE のブール値の結果を返します。2021 年 7 月 1 日の前に発生したトランザクションは、ブール値結果 FALSE を返します。

例 4 – todaydate

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- 最初の例と同じデータセットとシナリオ。
- 前回のリロード日付までのそのカレンダー年に発生したトランザクションを決定する項目 [year_to_date] の作成。

ただし、この例では、2022 年 3 月 1 日までを含むカレンダー年に発生したすべてのトランザクションを特定する必要があります。

ロードスクリプト

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

```
Transactions:
    Load
        *,
        yeartodate(date, 0, 1, '03/01/2022') as year_to_date
;
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- date
- year_to_date

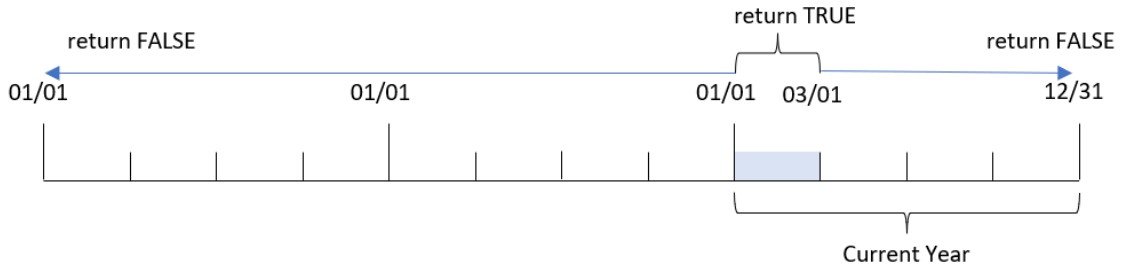
結果テーブル

日付	year_to_date
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0

日付	year_to_date
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	0
03/11/2022	0

この場合、todaydate 引数 03/01/2022 が yeartodate() 関数で使用されるため、比較対象カレンダー年の終了境界線が 2022 年 3 月 1 日に設定されます。firstmonth パラメータ (1 ~ 2) を提供することが重要です。そうしないと、関数が null 結果を返すことになります。

todaydate 引数を使用している yeartodate() 関数の例の図



したがって、2022 年 1 月 1 日 ~ 2022 年 3 月 1 日に発生したトランザクションの場合、todaydate パラメータはブール値の結果 TRUE を返します。2022 年 1 月 1 日の前、または 2022 年 3 月 1 日の後に発生したトランザクションは、ブール値結果 FALSE を返します。

例 5 – Chart object example

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには、最初の例と同じデータセットとシナリオが含まれます。

ただし、この例では、変更されていないデータセットがアプリケーションにロードされます。前回のリロードまでを含むカレンダー年に発生したトランザクションを決定する計算は、アプリケーションのチャート オブジェクトのメジャーとして作成されます。

ロード スクリプト

```
Transactions:
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、この項目を軸として追加します: `date`。

次のメジャーを追加します。

```
=yeartodate(date)
```

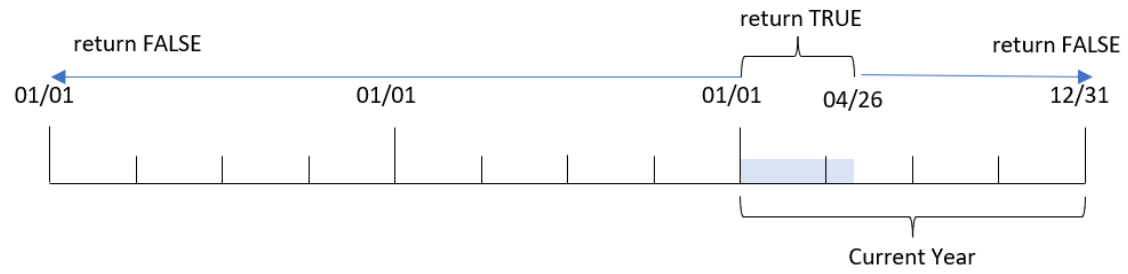
結果テーブル

日付	=yeartodate(date)
01/10/2020	0
02/28/2020	0
04/09/2020	0
04/16/2020	0
05/21/2020	0
08/14/2020	0
10/07/2020	0
12/05/2020	0
01/22/2021	0
02/03/2021	0
03/17/2021	0
04/23/2021	0
05/04/2021	0
06/30/2021	0
07/26/2021	0
12/27/2021	0
02/02/2022	-1
02/26/2022	-1
03/07/2022	-1
03/11/2022	-1

[year_to_date] メジャーは、yeartodate() 関数を使用し、関数の引数として [date] 項目を渡すことにより、チャートオブジェクトで作成されます。

それ以外のパラメータは関数に渡されないため、yeartodate() 関数は最初にリロード日付、そして現在のカレンダー年 (1月 1日開始) の境界を特定し、ブール値結果 TRUE を返します。

チャート オブジェクトを使用する yeartodate() 関数の例の図



1 月 1 日 ~ 4 月 26 日に発生したトランザクションの場合、リロード日付は TRUEのブール値の結果を返します。2022 年始めの前に発生したトランザクションは、ブール値結果 FALSE を返します。

例 6 – シナリオ

ロード スクリプトとチャートの数式

概要

データ ロード エディターを開き、以下のロード スクリプトを新しいタブに追加します。

ロード スクリプトには次が含まれています。

- Transactions というテーブルにロードされる、2020 年 ~ 2022 年の一連のトランザクションを含むデータセット。
- DateFormat システム変数形式 (MM/DD/YYYY) で提供されている日付項目。

エンド ユーザーは、2021 年の相当期間の総売上高を、前回のリロード時の現在の年初来として提示する KPI オブジェクトを求めています。

書き込み時には、日付は 2022 年 6 月 16 日です。

ロード スクリプト

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

Transactions:
Load
*
Inline
[
id,date,amount
8188,01/10/2020,37.23
8189,02/28/2020,17.17
8190,04/09/2020,88.27
8191,04/16/2020,57.42
8192,05/21/2020,53.80
8193,08/14/2020,82.06
8194,10/07/2020,40.39
8195,12/05/2020,87.21
```

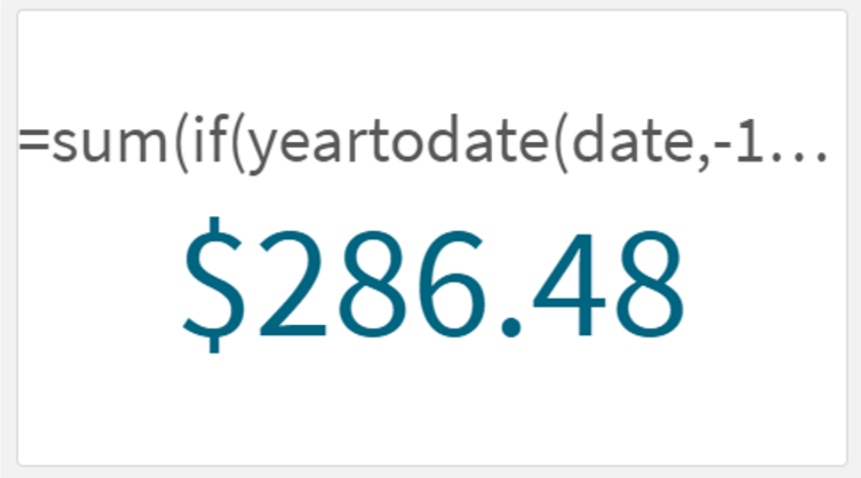
```
8196,01/22/2021,95.93
8197,02/03/2021,45.89
8198,03/17/2021,36.23
8199,04/23/2021,25.66
8200,05/04/2021,82.77
8201,06/30/2021,69.98
8202,07/26/2021,76.11
8203,12/27/2021,25.12
8204,02/02/2022,46.23
8205,02/26/2022,84.21
8206,03/07/2022,96.24
8207,03/11/2022,67.67
];
```

結果

次の手順を実行します。

1. KPI オブジェクトを作成します。
2. 総売上を計算するには、次の集計メジャーを作成します。
   `=sum(if(yeartodate(date,-1),amount,0))`
3. メジャーの **[数値書式]** を **[通貨]** に設定します。

2021 の KPI yeartodate() チャート



yeartodate() 関数は、各トランザクション ID の日付を評価するときにブール値を返します。リロードは 2022 年 6 月 16 日に発生したため、yeartodate 関数は年期間を 01/01/2022 と06/16/2022 に区分します。ただし、period_no 値 -1 が関数で使用されたため、これらも境界線は前の年に移動します。そのため、01/01/2021 ~ 06/16/2021 に発生したトランザクションについては、yeartodate() 関数がブール値 TRUE と金額の合計を返します。

8.8 指数関数と対数関数

このセクションでは、指数および対数の計算に関連する関数について説明します。すべての関数は、データロードスクリプトおよびチャートの数式の両方で使用できます。

次の関数では、パラメータは数式であり、**x** と**y** は実際の数値と解釈されます。

exp
自然対数の底 **e** を底として使用する自然指数関数 e^x。結果は正の数値です。

```
exp(x )
```

例と結果:

exp(3) は、20.085 を返します。

log
x の自然対数。関数は、**x**> 0 の場合にのみ定義されます。結果は数値で返されます。

```
log(x )
```

例と結果:

log(3) は、1.0986 を返します

log10
x の常用対数 (10 を底とする対数) です。関数は、**x**> 0 の場合にのみ定義されます。結果は数値で返されます。

```
log10(x )
```

例と結果:

log10(3) は、0.4771 を返します

pow
x の **y** 乗を返します。結果は数値で返されます。

```
pow(x,y )
```

例と結果:

pow(3, 3) は、27 を返します

sqr
x の 2 乗 (**x** の 2 のべき乗)。結果は数値で返されます。

```
sqr (x )
```

例と結果:

sqr(3) は、9 を返します

sqrt
x の平方根です。関数は、**x** >= 0 の場合にのみ定義されます。結果は正の数値です。

```
sqrt(x )
```

例と結果:

sqrt(3) は、1.732 を返します

8.9 項目関数

これらの関数は、チャート式でのみ使用できます。

整数または文字列を返す項目関数は、項目選択におけるさまざまな側面を識別します。

カウント関数

GetAlternativeCount
GetAlternativeCount() は、特定された項目に含まれる代替値 (薄いグレー) の数を返します。

```
GetAlternativeCount - チャート関数 (field_name)
```

GetExcludedCount
GetExcludedCount() は、特定した項目に含まれる除外値の数を返します。除外値には、代替値 (薄いグレー)、除外 (濃いグレー)、選択された除外値 (チェックマーク付きの濃いグレー) などの項目があります。

```
GetExcludedCount - チャート関数 (page 1172)(field_name)
```

GetNotSelectedCount
このチャート関数は、**fieldname** という名前の項目内に含まれる未選択の値の数を返します。この関数が機能するには、この項目を And モードにする必要があります。

```
GetNotSelectedCount - チャート関数(fieldname [, includeexcluded=false])
```

GetPossibleCount
GetPossibleCount()は、特定した項目に含まれる絞込値の数を返します。識別された項目に選択が含まれている場合は、選択された項目 (緑) がカウントされます。その他の場合は関連値 (白) がカウントされます。

```
GetPossibleCount - チャート関数(field_name)
```

GetSelectedCount
GetSelectedCount() は、項目内で選択された (緑) 値の数を返します。

```
GetSelectedCount - チャート関数 (field_name [, include_excluded])
```

項目および選択関数

GetCurrentSelections

GetCurrentSelections()は、アプリ内の現在の選択条件のリストを返します。代わりに、検索ボックスで検索文字列を使用して選択が行われた場合、**GetCurrentSelections()**は検索文字列を返します。

```
GetCurrentSelections - チャート関数([record_sep [,tag_sep [,value_sep [,max_values]]]])
```

GetFieldSelections

GetFieldSelections() は、項目内の現在の選択の **string** を返します。

```
GetFieldSelections - チャート関数 ( field_name [, value_sep [, max_values]])
```

GetObjectDimension

GetObjectDimension() は、軸の名前を返します。**Index** は、返される軸を示す任意の整数です。

```
GetObjectDimension - チャート関数 ([index])
```

GetObjectField

GetObjectField() は、軸の名前を返します。**Index** は、返される軸を示す任意の整数です。

```
GetObjectField - チャート関数 ([index])
```

GetObjectMeasure

GetObjectMeasure() はメジャーの名前を返します。**Index** は、返される軸を示す任意の整数です。

```
GetObjectMeasure - チャート関数 ([index])
```

GetAlternativeCount - チャート関数

GetAlternativeCount() は、特定された項目に含まれる代替値 (薄いグレー) の数を返します。

構文:

```
GetAlternativeCount (field_name)
```

戻り値データ型: 整数

引数:

引数

引数	説明
field_name	メジャー対象となるデータ範囲が含まれている項目です。

例と結果:

次の例では、フィルター パネルにロードされた **First name** 項目を使用しています。

例と結果

例	結果
John が **First name** に選択されている場合 `GetAlternativeCount ([First name])`	4 (**First name** に 4 種類の除外値 (グレー) があるため)
John と**Peter** が選択されている場合 `GetAlternativeCount ([First name])`	3 (**First name** に 3 種類の除外値 (グレー) があるため)
First name に値が選択されていない場合 `GetAlternativeCount ([First name])`	0 (何も選択されていません)

例で使用されているデータ:

```
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 - チャート関数

GetCurrentSelections()は、アプリ内の現在の選択条件のリストを返します。代わりに、検索ボックスで検索文字列を使用して選択が行われた場合、**GetCurrentSelections()**は検索文字列を返します。

オプションを使用している場合、record_sep を指定する必要があります。新しい行を指定するには、**record_sep** を **chr(13)&chr(10)** に設定します。

特定の値を除くすべての値を選択する場合、除外する値が 2 つの場合は「NOT x,y」、1 つの場合は「NOT y」形式を使用します。すべての値を選択し、その値のカウントが max_values よりも大きい場合、ALL のテキストが返されます。

構文:

```
GetCurrentSelections ([record_sep [, tag_sep [, value_sep [, max_values [, state_name]]]]])
```

戻り値データ型: string

引数:

引数

引数	説明
record_sep	項目レコードの間に置かれる区切り記号です。デフォルトでは、新しい行を意味する <CR><LF> が使用されます。

引数	説明
tag_sep	項目名のタグと項目値の間に置かれる区切り記号です。デフォルトは「: 」です。
value_sep	項目値の間に配置される区切り記号。デフォルトは「,」です。
max_values	個々にリストされる項目値の最大数です。この数が大きくなると、「x of yvalue」(x/y 個) という表記が使用されます。デフォルトは 6 です。
state_name	特定のビジュアライゼーションのために選択された並列ステートの名前。**state_name** 引数が使用されると、特定のステート名に関連付けられた選択のみが対象になります。

例と結果:

以下の例では、異なるフィルター パネルにロードされた 2 つの項目 (**First name** と**Initials**) を使用しています。

例と結果

例	結果
John が **First name** に選択されている場合 `GetCurrentSelections ()`	'First name: John'
Johnと**Peter** が **First name** に選択されている場合 `GetCurrentSelections ()`	'First name: John, Peter'
John と**Peter** が **First name** に、そして **JA** が**Initials** に選択されている場合 `GetCurrentSelections ()`	'First name: John, Peter Initials: JA'
John が **First name** に、そして **JA** が**Initials** に選択されている場合 `GetCurrentSelections ( chr(13)&chr(10) , ' = ' )`	'First name = John Initials = JA'
Sue 以外のすべての名前が **First name** に選択されており、**Initials** が選択されていない場合 `GetCurrentSelections (chr(13)&chr(10), '=', ',' ,3)`	'First name=NOT Sue'

例で使用されているデータ:

```
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 - チャート関数

GetExcludedCount() は、特定した項目に含まれる除外値の数を返します。除外値には、代替値 (薄いグレー)、除外 (濃いグレー)、選択された除外値 (チェックマーク付きの濃いグレー) などの項目があります。

構文:

```
GetExcludedCount (field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
field_name	メジャー対象となるデータ範囲が含まれている項目です。

例と結果:

以下の例では、異なるフィルター パネルにロードされた 3 つの項目 (**First name**、**Last name**、**Initials**) を使用しています。

例と結果

例	結果
First name で値が選択されていない場合。	`GetExcludedCount (Initials) = 0` 選択がありません。
First name で **John** が選択されている場合。	`GetExcludedCount (Initials) = 5` 濃いグレー色の **Initials** には、除外値が 5 つあります。6 番目のセル (JA) は、**First name** での選択 John に関連付けられているため、白色で表示されます。
John と**Peter** が選択されている場合。	`GetExcludedCount (Initials) = 3` **Initials** において、John は 1 つの値に関連付けられ、Peter は 2 つの値と関連付けられます。
First name で **John** および **Peter** が選択されている場合、**Last name** では **Franc** が選択されます。	`GetExcludedCount ([First name]) = 4` 濃いグレー色の [名] には除外値が 4 つあります。 **GetExcludedCount()** は、代替および選択された除外項目を含め、除外値が入っている項目を評価します。
First name で **John** と**Peter** が選択されている場合、**Last name** では **Anderson** と**Franc** が選択されます。	`GetExcludedCount (Initials) = 4` 濃いグレー色の **Initials** には、除外値が 4 つあります。他の 2 つのセル (JA と PF) は **First name** での選択 (John と Peter) に関連付けられているため、白で表示されます。
First name で **John** と**Peter** が選択されている場合、**Last name** では **Anderson** と**Franc** が選択されます。	`GetExcludedCount ([Last name]) = 4` **Initials** には、除外値が 4 つあります。Devonshire は薄いグレー色で表示されるのに対して、Brown、Carr、Elliot は濃いグレー色で表示されます。

例で使用されているデータ:

```
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 - チャート関数

GetFieldSelections() は、項目内の現在の選択の **string** を返します。

すべての値の中で一部を除外して選択する場合、除外する値が 2 つの場合は「NOT x,y」、1 つの場合は「NOT y」形式を使用します。すべての値を選択し、その値のカウントが max_values よりも大きい場合、ALL のテキストが返されます。

構文:

GetFieldSelections (field_name [, value_sep [, max_values [, state_name]]]**)**

戻り値データ型: 文字列

返される文字列の書式

書式	説明
'a, b, c'	選択された値の数が max_values 以下の場合、返される文字列は選択値のリストになります。 値は value_sep を区切り記号として区切ります。
'NOT a, b, c'	非選択値の数が max_values 以下の場合、返される文字列は、非選択値のリストに NOT を前置したものです。 値は value_sep を区切り記号として区切ります。
'x of y'	x = 選択値の数 y = 値の合計数 max_values < x < (y - max_values) のときに返されます。
'ALL'	すべての値が選択されている場合に返されます。
'-'	値が選択されていない場合に返されます。
<search string>	[検索] を選択した場合、検索文字列が返されます。

引数:

引数

引数	説明
field_name	メジャー対象となるデータ範囲が含まれている項目です。
value_sep	項目値の間に配置される区切り記号。デフォルトは「,」です。
max_values	個々にリストされる項目値の最大数です。この数が大きくなると、「x of yvalue」(x/y 個) という表記が使用されます。デフォルトは 6 です。
state_name	特定のビジュアライゼーションのために選択された並列ステートの名前。**state_name** 引数が使用されると、特定のステート名に関連付けられた選択のみが対象になります。

例と結果:

次の例では、フィルター パネルにロードされた **First name** 項目を使用しています。

例と結果

例	結果
John が **First name** に選択されている場合 `GetFieldSelections ([First name])`	「John」
John と**Peter** が選択されている場合 `GetFieldSelections ([First name])`	「John,Peter」
John と**Peter** が選択されている場合 `GetFieldSelections ([First name],'; ')`	「John; Peter」
Johnと**Sue**、**Mark** が **First name** に選択されている場合 `GetFieldSelections ([First name],';',2)`	NOT Jane;Peter (max_values 引数の値に 2 が指定されているため) それ以外の場合は、John; Sue; Mark. が返されます。

例で使用されているデータ:

```
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 - チャート関数

このチャート関数は、**fieldname** という名前の項目内に含まれる未選択の値の数を返します。この関数が機能するには、この項目を And モードにする必要があります。

構文:

```
GetNotSelectedCount(fieldname [, includeexcluded=false])
```

引数:

引数

引数	説明
fieldname	評価される項目の名前。
includeexcluded	**includeexcluded** が True に指定されている場合、カウントには別の項目の選択によって除外された選択値が含まれます。

```
GetNotSelectedCount( Country )
```

```
GetNotSelectedCount( Country, true )
```

GetObjectDimension - チャート関数

GetObjectDimension() は、軸の名前を返します。**Index** は、返される軸を示す任意の整数です。

この機能は、以下の場所にあるチャートでは使用できません: タイトル、サブタイトル、フッター、基準線式、最小/最大式。

Object ID を使用して、他のオブジェクトの軸または メジャーの名前を参照することはできません。

構文:

```
GetObjectDimension ([index])
```

```
GetObjectDimension(1)
```

例: チャートの数式

Qlik Sense のテーブルは、チャートの数式における GetObjectDimension 関数の例を示しています

transaction_date	customer_id	transaction_quantity	=GetObjectDimension ()	=GetObjectDimension (0)	=GetObjectDimension (1)
2018/08/30	049681	13	transaction_date	transaction_date	customer_id
2018/08/30	203521	6	transaction_date	transaction_date	customer_id
2018/08/30	203521	21	transaction_date	transaction_date	customer_id

メジャーの名前を返す場合は、代わりに **GetObjectMeasure** 関数を使用します。

GetObjectField - チャート関数

GetObjectField() は、軸の名前を返します。**Index** は、返される軸を示す任意の整数です。

この機能は、以下の場所にあるチャートでは使用できません: タイトル、サブタイトル、フッター、基準線式、最小/最大式。

Object ID を使用して、他のオブジェクトの軸またはメジャーの名前を参照することはできません。

構文:

```
GetObjectField ([index])
```

```
GetObjectField(1)
```

例: チャートの数式

Qlik Sense のテーブルは、チャートの数式における GetObjectField 関数の例を示しています。

transaction_date	customer_id	transaction_quantity	=GetObjectField ()	=GetObjectField (0)	=GetObjectField (1)
2018/08/30	049681	13	transaction_date	transaction_date	customer_id
2018/08/30	203521	6	transaction_date	transaction_date	customer_id
2018/08/30	203521	21	transaction_date	transaction_date	customer_id

メジャーの名前を返す場合は、代わりに **GetObjectMeasure** 関数を使用します。

GetObjectMeasure - チャート関数

GetObjectMeasure() は メジャーの名前を返します。**Index** は、返される軸を示す任意の整数です。

この機能は、以下の場所にあるチャートでは使用できません: タイトル、サブタイトル、フッター、基準線式、最小/最大式。

Object ID を使用して、他のオブジェクトの軸または メジャーの名前を参照することはできません。

構文:

```
GetObjectMeasure ([index])
```

```
GetObjectMeasure(1)
```

例: チャートの数式

Qlik Sense のテーブルは、チャートの数式における GetObjectMeasure 関数の例を示しています

customer_id	sum (transaction_ quantity)	Avg (transaction_ quantity)	=GetObjectMeasure ()	=GetObjectMeasure(0)	=GetObjectMeasure(1)
49681	13	13	sum (transaction_ quantity)	sum (transaction_ quantity)	Avg(transaction_ quantity)
203521	27	13.5	sum (transaction_ quantity)	sum (transaction_ quantity)	Avg(transaction_ quantity)

軸の名前を返す場合は、代わりに **GetObjectField** 関数を使用します。

GetPossibleCount - チャート関数

GetPossibleCount()は、特定した項目に含まれる絞込値の数を返します。識別された項目に選択が含まれている場合は、選択された項目 (緑) がカウントされます。その他の場合は関連値 (白) がカウントされます。

選択が含まれている項目の場合、**GetPossibleCount()** を使用すると選択した項目 (緑色) の数が返されます。

戻り値データ型: 整数

構文:

```
GetPossibleCount (field_name)
```

引数:

引数

引数	説明
field_name	メジャー対象となるデータ範囲が含まれている項目です。

例と結果:

以下の例では、異なるフィルター パネルにロードされた 2 つの項目 (**First name** と**Initials**) を使用しています。

例と結果

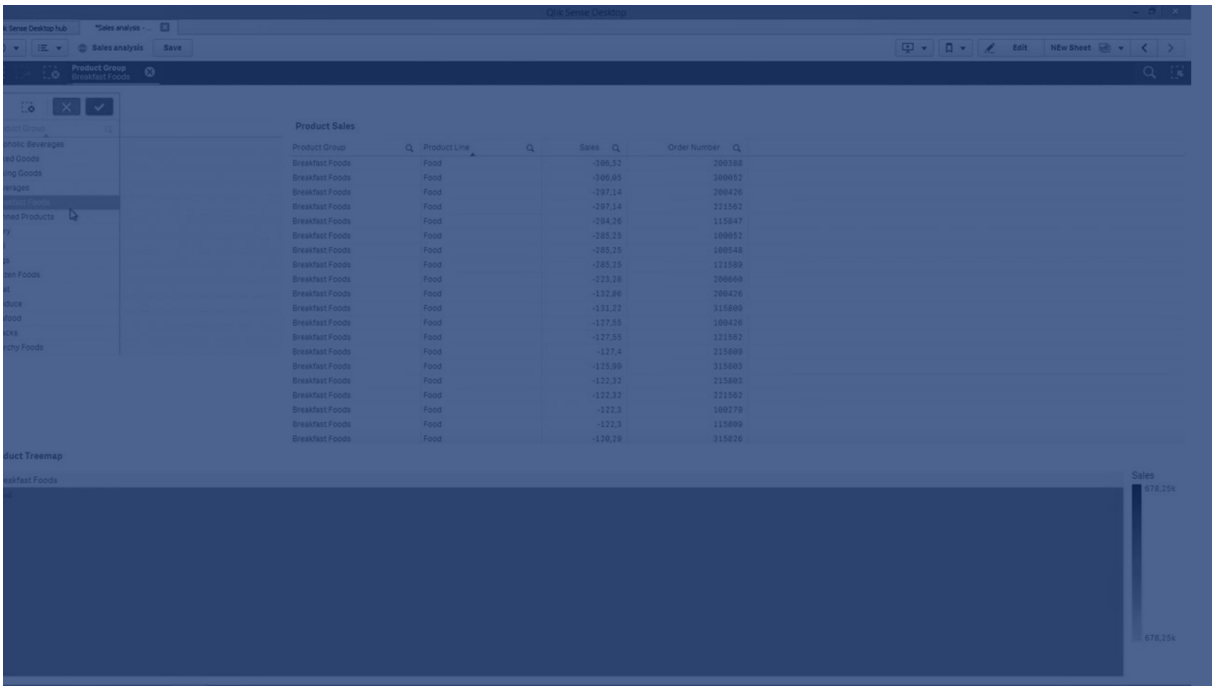
例	結果
John が **First name** に選択されている場合 `GetPossibleCount ([Initials])`	1 (Initials には、**John** という**First name** に関連付けられている値が 1 つ存在するため)
John が **First name** に選択されている場合 `GetPossibleCount ([First name])`	1 (**John** は **First name** に 1 つしか存在しないため)
Peterが **First name** に選択されている場合 `GetPossibleCount ([Initials])`	2 (Peter は **Initials** の 2 つの値に関連付けられているため)
First name に値が選択されていない場合 `GetPossibleCount ([First name])`	5 (何も選択されていないが、**First name** に固有の値が 5 つ存在するため)
First name に値が選択されていない場合 `GetPossibleCount ([Initials])`	6 (何も選択されていないが、**Initials** に固有の値が 6 つ存在するため)

例で使用されているデータ:

```
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 - チャート関数

GetSelectedCount() は、項目内で選択された (緑) 値の数を返します。



構文:

```
GetSelectedCount (field_name [, include_excluded [, state_name]])
```

戻り値データ型: 整数

引数:

引数

引数	説明
field_name	メジャー対象となるデータ範囲が含まれている項目です。
include_excluded	**True()**に設定されている場合、他の項目の選択によって現在除外されている選択値がカウントに含まれます。False の場合や省略されている場合、これらの値は含まれません。
state_name	特定のビジュアライゼーションのために選択された並列ステートの名前。**state_name** 引数が使用されると、特定のステート名に関連付けられた選択のみが対象になります。

例と結果:

以下の例では、異なるフィルター パネルにロードされた 3 つの項目 (**First name**、**Initials**、**Has cellphone**) を使用しています。

例と結果

例	結果
John が **First name** に選択されている場合 `GetSelectedCount ([First name])`	1 (**First name** に選択されている値が 1 つあるため)
John が **First name** に選択されている場合 `GetSelectedCount ([Initials])`	0 (**Initials** に選択されている値がないため)
First name が選択されておらず、**Initials** ですべての値が選択されており、**Has cellphone** が **Yes** になっている場合 `GetSelectedCount ([Initials], True ())`	6 (**Initials** MC および PD は **Has cellphone** が **No** に選択されているが、引数 include_excluded が True() に設定されているため、結果はいずれにせよ 6 となるため)

例で使用されているデータ:

```
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 '|');
```

8.10 ファイル関数

ファイル関数 (スクリプト式でのみ有効) は、現在読み込まれているテーブル ファイルに関する情報を返します。これらの関数は、テーブル ファイル以外のデータ ソースの場合は NULL を返します (**ConnectString( )** は例外)。

ファイル関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Attribute

このスクリプト関数は、異なるメディア ファイルのメタ タグの値をテキストとして返します。次の形式がサポートされています。MP3、WMA、WMV、PNG、および JPG です。**filename** ファイルが存在しない場合、ファイル形式がサポートされていない場合、または **attributename** というメタ タグが含まれていない場合は、NULL を返します。

```
Attribute (filename, attributename)
```

ConnectString

ConnectString() 関数は、ODBC 接続または OLE DB 接続のアクティブなデータ接続の名前を返します。connections.**connect** ステートメントが実行されていない場合、または **disconnect** ステートメントの実行後は、空の文字列を返します。

```
ConnectString ()
```

FileBaseName

FileBaseName 関数は、現在読み取り中のテーブル ファイルのファイル名を、パスや拡張子を省略した文字列で返します。

```
FileBaseName ()
```

FileDir

FileDir 関数は、現在読み取り中のテーブル ファイルのディレクトリパスを文字列で返します。

```
FileDir ()
```

FileExtension

FileExtension 関数は、現在読み取り中のテーブル ファイルの拡張子を文字列で返します。

```
FileExtension ()
```

FileName

FileName 関数は、現在読み取り中のテーブル ファイルのファイル名を、パスを省略し、拡張子を付けて文字列で返します。

```
FileName ()
```

FilePath

FilePath 関数は、現在読み取り中のテーブル ファイルのフル パスを文字列で返します。

```
FilePath ()
```

FileSize

FileSize 関数は、filename ファイル のサイズをバイト数で表した整数を返します。filename が指定されていない場合は、現在読み取り中のテーブル ファイルのサイズを返します。

```
FileSize ()
```

FileTime

FileTime 関数は、指定されたファイルの最後に更新された日付と時刻を UTC フォーマットで返します。ファイルが指定されていない場合、関数は現在読み込まれているテーブル ファイルの最後に更新された日付と時刻を返します。

```
FileTime ([ filename ])
```

GetFolderPath

GetFolderPath 関数は、Microsoft Windows *SHGetFolderPath* 関数の値を返します。この関数は、Microsoft Windows フォルダの名前を入力として返し、フォルダのフルパスを返します。

```
GetFolderPath ()
```

QvdCreateTime
このスクリプト関数は、QVD ファイルに含まれた XML ヘッダーの日付と時刻を返します (ない場合は NULL を返します)。タイムスタンプでは、時刻は UTC で提供されます。

```
QvdCreateTime (filename)
```

QvdFieldName
このスクリプト関数は、QVDファイルの項目番号**fieldno**の名前を返します。項目が存在しない場合は、NULL を返します。

```
QvdFieldName (filename , fieldno)
```

QvdNoOfFields
このスクリプト関数は、QVD ファイル内の項目数を返します。

```
QvdNoOfFields (filename)
```

QvdNoOfRecords
このスクリプト関数は、QVD ファイル内に含まれるレコードの数を返します。

```
QvdNoOfRecords (filename)
```

QvdTableName
このスクリプト関数は、QVD ファイルに保存されているテーブルの名前を返します。

```
QvdTableName (filename)
```

Attribute

このスクリプト関数は、異なるメディア ファイルのメタ タグの値をテキストとして返します。次の形式がサポートされています。MP3、WMA、WMV、PNG、および JPG です。**filename** ファイルが存在しない場合、ファイル形式がサポートされていない場合、または **attributename** というメタ タグが含まれていない場合は、NULL を返します。

構文:

```
Attribute(filename, attributename)
```

さまざまなメタ タグを読み込むことができます。このトピックの例では、サポートされている各ファイル タイプ別に読み込むことができるタグを示しています。

> *[**Windows File Explorer**] で保存されたメタ情報ではなく、関連性に従ってファイルで保存されたメタ タグのみが読み込み可能です。例えば、ID2v3 は MP3 ファイルのタグで、EXIF は JPG ファイルのタグです。*

引数:

引数

引数	説明
filename	メディア ファイルのファイル名で、場合によってはフォルダデータ接続のようなパスを含みます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data***
attributename	メタタグの名前。

この例では、メディア ファイルへのパスを取得するために **GetFolderPath** 関数を使用しています。**GetFolderPath** はレガシー モードでのみサポートされるため、この関数を標準モードまたは Qlik Sense SaaS で使用する場合は、**GetFolderPath** への参照を lib:// データ接続パスに替える必要があります。

ファイル システム アクセス制御 (page 1481)

Example 1: MP3 ファイル

このスクリプトでは、存在する可能性のあるすべての MP3 メタタグを、*MyMusic* フォルダで読み込みます。

```
// Script to read MP3 meta tags
for each vExt in 'mp3'
for each vFoundFile in filelist( GetFolderPath('MyMusic') & '\*.'& vExt )
FileList:
LOAD FileLongName,
    subfield(FileLongName,'\',-1) as FileShortName,
    num(FileSize(FileLongName),'# ### ### ###',',',' ') as FileSize,
    FileTime(FileLongName) as FileTime,
    // ID3v1.0 and ID3v1.1 tags
    Attribute(FileLongName, 'Title') as Title,
    Attribute(FileLongName, 'Artist') as Artist,
    Attribute(FileLongName, 'Album') as Album,
    Attribute(FileLongName, 'Year') as Year,
    Attribute(FileLongName, 'Comment') as Comment,
    Attribute(FileLongName, 'Track') as Track,
    Attribute(FileLongName, 'Genre') as Genre,

    // ID3v2.3 tags
    Attribute(FileLongName, 'AENC') as AENC, // Audio encryption
    Attribute(FileLongName, 'APIC') as APIC, // Attached picture
```

```
Attribute(FileLongName, 'COMM') as COMM, // Comments
Attribute(FileLongName, 'COMR') as COMR, // Commercial frame
Attribute(FileLongName, 'ENCR') as ENCR, // Encryption method registration
Attribute(FileLongName, 'EQUA') as EQUA, // Equalization
Attribute(FileLongName, 'ETCO') as ETCO, // Event timing codes
Attribute(FileLongName, 'GEOB') as GEOB, // General encapsulated object
Attribute(FileLongName, 'GRID') as GRID, // Group identification registration
Attribute(FileLongName, 'IPLS') as IPLS, // Involved people list
Attribute(FileLongName, 'LINK') as LINK, // Linked information
Attribute(FileLongName, 'MCDI') as MCDI, // Music CD identifier
Attribute(FileLongName, 'MLLT') as MLLT, // MPEG location lookup table
Attribute(FileLongName, 'OWNE') as OWNE, // Ownership frame
Attribute(FileLongName, 'PRIV') as PRIV, // Private frame
Attribute(FileLongName, 'PCNT') as PCNT, // Play counter
Attribute(FileLongName, 'POPM') as POPM, // Popularimeter

Attribute(FileLongName, 'POSS') as POSS, // Position synchronisation frame
Attribute(FileLongName, 'RBUF') as RBUF, // Recommended buffer size
Attribute(FileLongName, 'RVAD') as RVAD, // Relative volume adjustment
Attribute(FileLongName, 'RVRB') as RVRB, // Reverb
Attribute(FileLongName, 'SYLT') as SYLT, // Synchronized lyric/text
Attribute(FileLongName, 'SYTC') as SYTC, // Synchronized tempo codes
Attribute(FileLongName, 'TALB') as TALB, // Album/Movie/Show title
Attribute(FileLongName, 'TBPM') as TBPM, // BPM (beats per minute)
Attribute(FileLongName, 'TCOM') as TCOM, // Composer
Attribute(FileLongName, 'TCON') as TCON, // Content type
Attribute(FileLongName, 'TCOP') as TCOP, // Copyright message
Attribute(FileLongName, 'TDAT') as TDAT, // Date
Attribute(FileLongName, 'TDLY') as TDLY, // Playlist delay

Attribute(FileLongName, 'TENC') as TENC, // Encoded by
Attribute(FileLongName, 'TEXT') as TEXT, // Lyricist/Text writer
Attribute(FileLongName, 'TFLT') as TFLT, // File type
Attribute(FileLongName, 'TIME') as TIME, // Time
Attribute(FileLongName, 'TIT1') as TIT1, // Content group description
Attribute(FileLongName, 'TIT2') as TIT2, // Title/songname/content description
Attribute(FileLongName, 'TIT3') as TIT3, // Subtitle/Description refinement
Attribute(FileLongName, 'TKEY') as TKEY, // Initial key
Attribute(FileLongName, 'TLAN') as TLAN, // Language(s)
Attribute(FileLongName, 'TLEN') as TLEN, // Length
Attribute(FileLongName, 'TMED') as TMED, // Media type

Attribute(FileLongName, 'TOAL') as TOAL, // Original album/movie/show title
Attribute(FileLongName, 'TOFN') as TOFN, // Original filename
Attribute(FileLongName, 'TOLY') as TOLY, // Original lyricist(s)/text writer(s)
Attribute(FileLongName, 'TOPE') as TOPE, // Original artist(s)/performer(s)
Attribute(FileLongName, 'TORY') as TORY, // Original release year
Attribute(FileLongName, 'TOWN') as TOWN, // File owner/licensee
Attribute(FileLongName, 'TPE1') as TPE1, // Lead performer(s)/Soloist(s)
Attribute(FileLongName, 'TPE2') as TPE2, // Band/orchestra/accompaniment

Attribute(FileLongName, 'TPE3') as TPE3, // Conductor/performer refinement
Attribute(FileLongName, 'TPE4') as TPE4, // Interpreted, remixed, or otherwise modified by
Attribute(FileLongName, 'TPOS') as TPOS, // Part of a set
Attribute(FileLongName, 'TPUB') as TPUB, // Publisher
```

```
    Attribute(FileLongName, 'TRCK') as TRCK, // Track number/Position in set
    Attribute(FileLongName, 'TRDA') as TRDA, // Recording dates
    Attribute(FileLongName, 'TRSN') as TRSN, // Internet radio station name
    Attribute(FileLongName, 'TRSO') as TRSO, // Internet radio station owner

    Attribute(FileLongName, 'TSIZ') as TSIZ, // Size
    Attribute(FileLongName, 'TSRC') as TSRC, // ISRC (international standard recording code)
    Attribute(FileLongName, 'TSSE') as TSSE, // Software/Hardware and settings used for
encoding
    Attribute(FileLongName, 'TYER') as TYER, // Year
    Attribute(FileLongName, 'TXXX') as TXXX, // User defined text information frame
    Attribute(FileLongName, 'UFID') as UFID, // Unique file identifier
    Attribute(FileLongName, 'USER') as USER, // Terms of use
    Attribute(FileLongName, 'USLT') as USLT, // Unsychronized lyric/text transcription
    Attribute(FileLongName, 'WCOM') as WCOM, // Commercial information
    Attribute(FileLongName, 'WCOP') as WCOP, // Copyright/Legal information

    Attribute(FileLongName, 'WOAF') as WOAF, // Official audio file webpage
    Attribute(FileLongName, 'WOAR') as WOAR, // Official artist/performer webpage
    Attribute(FileLongName, 'WOAS') as WOAS, // Official audio source webpage
    Attribute(FileLongName, 'WORS') as WORS, // Official internet radio station homepage
    Attribute(FileLongName, 'WPAY') as WPAY, // Payment
    Attribute(FileLongName, 'WPUB') as WPUB, // Publishers official webpage
    Attribute(FileLongName, 'WXXX') as WXXX; // User defined URL link frame
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels);
Next vFoundFile
Next vExt
```

Example 2: JPEG

このスクリプトでは、存在する可能性のあるすべての EXIF メタタグを、*MyPictures* フォルダの JPG ファイルから読み込みます。

```
// Script to read Jpeg Exif meta tags
for each vExt in 'jpg', 'jpeg', 'jpe', 'jfif', 'jif', 'jfi'
for each vFoundFile in filelist( GetFolderPath('MyPictures') & '\*.'& vExt )

FileList:
LOAD FileLongName,
    subfield(FileLongName,'\',-1) as FileShortName,
    num(FileSize(FileLongName),'# ### ### ###',',',' ') as FileSize,
    FileTime(FileLongName) as FileTime,
    // ************   Exif Main (IFD0) Attributes   ************
    Attribute(FileLongName, 'ImageWidth') as ImageWidth,
    Attribute(FileLongName, 'ImageLength') as ImageLength,
    Attribute(FileLongName, 'BitsPerSample') as BitsPerSample,
    Attribute(FileLongName, 'Compression') as Compression,

    //  examples: 1=uncompressed, 2=CCITT, 3=CCITT 3, 4=CCITT 4,

    //5=LZW, 6=JPEG (old style), 7=JPEG, 8=Deflate, 32773=PackBits RLE,
    Attribute(FileLongName, 'PhotometricInterpretation') as PhotometricInterpretation,

    //  examples: 0=WhiteIsZero, 1=BlackIsZero, 2=RGB, 3=Palette, 5=CMYK, 6=YCbCr,
    Attribute(FileLongName, 'ImageDescription') as ImageDescription,
```

```
    Attribute(FileLongName, 'Make') as Make,
    Attribute(FileLongName, 'Model') as Model,
    Attribute(FileLongName, 'StripOffsets') as StripOffsets,
    Attribute(FileLongName, 'Orientation') as Orientation,

    //  examples: 1=TopLeft, 2=TopRight, 3=BottomRight, 4=BottomLeft,

    // 5=LeftTop, 6=RightTop, 7=RightBottom, 8=LeftBottom,
    Attribute(FileLongName, 'SamplesPerPixel') as SamplesPerPixel,
    Attribute(FileLongName, 'RowsPerStrip') as RowsPerStrip,
    Attribute(FileLongName, 'StripByteCounts') as StripByteCounts,
    Attribute(FileLongName, 'XResolution') as XResolution,
    Attribute(FileLongName, 'YResolution') as YResolution,
    Attribute(FileLongName, 'PlanarConfiguration') as PlanarConfiguration,

    //  examples: 1=chunky format, 2=planar format,
    Attribute(FileLongName, 'ResolutionUnit') as ResolutionUnit,

    //  examples: 1=none, 2=inches, 3=centimeters,
    Attribute(FileLongName, 'TransferFunction') as TransferFunction,
    Attribute(FileLongName, 'Software') as Software,
    Attribute(FileLongName, 'DateTime') as DateTime,
    Attribute(FileLongName, 'Artist') as Artist,
    Attribute(FileLongName, 'HostComputer') as HostComputer,
    Attribute(FileLongName, 'WhitePoint') as WhitePoint,
    Attribute(FileLongName, 'PrimaryChromaticities') as PrimaryChromaticities,
    Attribute(FileLongName, 'YCbCrCoefficients') as YCbCrCoefficients,
    Attribute(FileLongName, 'YCbCrSubSampling') as YCbCrSubSampling,
    Attribute(FileLongName, 'YCbCrPositioning') as YCbCrPositioning,

    //  examples: 1=centered, 2=co-sited,
    Attribute(FileLongName, 'ReferenceBlackWhite') as ReferenceBlackWhite,
    Attribute(FileLongName, 'Rating') as Rating,
    Attribute(FileLongName, 'RatingPercent') as RatingPercent,
    Attribute(FileLongName, 'ThumbnailFormat') as ThumbnailFormat,

    //  examples: 0=Raw Rgb, 1=Jpeg,
    Attribute(FileLongName, 'Copyright') as Copyright,
    Attribute(FileLongName, 'ExposureTime') as ExposureTime,
    Attribute(FileLongName, 'FNumber') as FNumber,
    Attribute(FileLongName, 'ExposureProgram') as ExposureProgram,

    //  examples: 0=Not defined, 1=Manual, 2=Normal program, 3=Aperture priority, 4=Shutter
priority,

    // 5=Creative program, 6=Action program, 7=Portrait mode, 8=Landscape mode, 9=Bulb,
    Attribute(FileLongName, 'ISOSpeedRatings') as ISOSpeedRatings,
    Attribute(FileLongName, 'TimeZoneOffset') as TimeZoneOffset,
    Attribute(FileLongName, 'SensitivityType') as SensitivityType,

    //  examples: 0=Unknown, 1=Standard output sensitivity (SOS), 2=Recommended exposure index
(REI),

    // 3=ISO speed, 4=Standard output sensitivity (SOS) and Recommended exposure index (REI),
```

```
    //5=Standard output sensitivity (SOS) and ISO Speed, 6=Recommended exposure index (REI)
and ISO Speed,

    // 7=Standard output sensitivity (SOS) and Recommended exposure index (REI) and ISO speed,
    Attribute(FileLongName, 'ExifVersion') as ExifVersion,
    Attribute(FileLongName, 'DateTimeOriginal') as DateTimeOriginal,
    Attribute(FileLongName, 'DateTimeDigitized') as DateTimeDigitized,
    Attribute(FileLongName, 'ComponentsConfiguration') as ComponentsConfiguration,

    //  examples: 1=Y, 2=Cb, 3=Cr, 4=R, 5=G, 6=B,
    Attribute(FileLongName, 'CompressedBitsPerPixel') as CompressedBitsPerPixel,
    Attribute(FileLongName, 'ShutterSpeedValue') as ShutterSpeedValue,
    Attribute(FileLongName, 'ApertureValue') as ApertureValue,
    Attribute(FileLongName, 'BrightnessValue') as BrightnessValue, //  examples: -1=Unknown,
    Attribute(FileLongName, 'ExposureBiasValue') as ExposureBiasValue,
    Attribute(FileLongName, 'MaxApertureValue') as MaxApertureValue,
    Attribute(FileLongName, 'SubjectDistance') as SubjectDistance,

    //  examples: 0=Unknown, -1=Infinity,
    Attribute(FileLongName, 'MeteringMode') as MeteringMode,

    //  examples: 0=Unknown, 1=Average, 2=CenterWeightedAverage, 3=Spot,

    // 4=MultiSpot, 5=Pattern, 6=Partial, 255=Other,
    Attribute(FileLongName, 'LightSource') as LightSource,

    //  examples: 0=Unknown, 1=Daylight, 2=Fluorescent, 3=Tungsten, 4=Flash, 9=Fine weather,

    // 10=Cloudy weather, 11=Shade, 12=Daylight fluorescent,

    // 13=Day white fluorescent, 14=Cool white fluorescent,

    // 15=White fluorescent, 17=Standard light A, 18=Standard light B, 19=Standard light C,

    // 20=D55, 21=D65, 22=D75, 23=D50, 24=ISO studio tungsten, 255=other light source,
    Attribute(FileLongName, 'Flash') as Flash,
    Attribute(FileLongName, 'FocalLength') as FocalLength,
    Attribute(FileLongName, 'SubjectArea') as SubjectArea,
    Attribute(FileLongName, 'MakerNote') as MakerNote,
    Attribute(FileLongName, 'UserComment') as UserComment,
    Attribute(FileLongName, 'SubSecTime') as SubSecTime,

    Attribute(FileLongName, 'SubsecTimeOriginal') as SubsecTimeOriginal,
    Attribute(FileLongName, 'SubsecTimeDigitized') as SubsecTimeDigitized,
    Attribute(FileLongName, 'XPTitle') as XPTitle,
    Attribute(FileLongName, 'XPComment') as XPComment,

    Attribute(FileLongName, 'XPAuthor') as XPAuthor,
    Attribute(FileLongName, 'XPKeywords') as XPKeywords,
    Attribute(FileLongName, 'XPSubject') as XPSubject,
    Attribute(FileLongName, 'FlashpixVersion') as FlashpixVersion,
    Attribute(FileLongName, 'ColorSpace') as ColorSpace, //  examples: 1=sRGB,
65535=Uncalibrated,
    Attribute(FileLongName, 'PixelXDimension') as PixelXDimension,
```

```
Attribute(FileLongName, 'PixelYDimension') as PixelYDimension,
Attribute(FileLongName, 'RelatedSoundFile') as RelatedSoundFile,

Attribute(FileLongName, 'FocalPlaneXResolution') as FocalPlaneXResolution,
Attribute(FileLongName, 'FocalPlaneYResolution') as FocalPlaneYResolution,
Attribute(FileLongName, 'FocalPlaneResolutionUnit') as FocalPlaneResolutionUnit,

//  examples: 1=None, 2=Inch, 3=Centimeter,
Attribute(FileLongName, 'ExposureIndex') as ExposureIndex,
Attribute(FileLongName, 'SensingMethod') as SensingMethod,

//  examples: 1=Not defined, 2=One-chip color area sensor, 3=Two-chip color area sensor,

// 4=Three-chip color area sensor, 5=Color sequential area sensor,

// 7=Trilinear sensor, 8=Color sequential linear sensor,
Attribute(FileLongName, 'FileSource') as FileSource,

//  examples: 0=Other, 1=Scanner of transparent type,

// 2=Scanner of reflex type, 3=Digital still camera,
Attribute(FileLongName, 'SceneType') as SceneType,

//  examples: 1=A directly photographed image,
Attribute(FileLongName, 'CFAPattern') as CFAPattern,
Attribute(FileLongName, 'CustomRendered') as CustomRendered,

//  examples: 0=Normal process, 1=Custom process,
Attribute(FileLongName, 'ExposureMode') as ExposureMode,

//  examples: 0=Auto exposure, 1=Manual exposure, 2=Auto bracket,
Attribute(FileLongName, 'WhiteBalance') as WhiteBalance,

//  examples: 0=Auto white balance, 1=Manual white balance,
Attribute(FileLongName, 'DigitalZoomRatio') as DigitalZoomRatio,
Attribute(FileLongName, 'FocalLengthIn35mmFilm') as FocalLengthIn35mmFilm,
Attribute(FileLongName, 'SceneCaptureType') as SceneCaptureType,

//  examples: 0=Standard, 1=Landscape, 2=Portrait, 3=Night scene,
Attribute(FileLongName, 'GainControl') as GainControl,

//  examples: 0=None, 1=Low gain up, 2=High gain up, 3=Low gain down, 4=High gain down,
Attribute(FileLongName, 'Contrast') as Contrast,

//  examples: 0=Normal, 1=Soft, 2=Hard,
Attribute(FileLongName, 'Saturation') as Saturation,

//  examples: 0=Normal, 1=Low saturation, 2=High saturation,
Attribute(FileLongName, 'Sharpness') as Sharpness,

//  examples: 0=Normal, 1=Soft, 2=Hard,
Attribute(FileLongName, 'SubjectDistanceRange') as SubjectDistanceRange,

//  examples: 0=Unknown, 1=Macro, 2=Close view, 3=Distant view,
Attribute(FileLongName, 'ImageUniqueID') as ImageUniqueID,
```

```
    Attribute(FileLongName, 'BodySerialNumber') as BodySerialNumber,
    Attribute(FileLongName, 'CMNT_GAMMA') as CMNT_GAMMA,
    Attribute(FileLongName, 'PrintImageMatching') as PrintImageMatching,
    Attribute(FileLongName, 'OffsetSchema') as OffsetSchema,

    // ************   Interoperability Attributes   ************
    Attribute(FileLongName, 'InteroperabilityIndex') as InteroperabilityIndex,
    Attribute(FileLongName, 'InteroperabilityVersion') as InteroperabilityVersion,
    Attribute(FileLongName, 'InteroperabilityRelatedImageFileFormat') as
InteroperabilityRelatedImageFileFormat,
    Attribute(FileLongName, 'InteroperabilityRelatedImageWidth') as
InteroperabilityRelatedImageWidth,
    Attribute(FileLongName, 'InteroperabilityRelatedImageLength') as
InteroperabilityRelatedImageLength,
    Attribute(FileLongName, 'InteroperabilityColorSpace') as InteroperabilityColorSpace,

    //  examples: 1=sRGB, 65535=Uncalibrated,
    Attribute(FileLongName, 'InteroperabilityPrintImageMatching') as
InteroperabilityPrintImageMatching,
    // ************   GPS Attributes   ************
    Attribute(FileLongName, 'GPSVersionID') as GPSVersionID,
    Attribute(FileLongName, 'GPSLatitudeRef') as GPSLatitudeRef,
    Attribute(FileLongName, 'GPSLatitude') as GPSLatitude,
    Attribute(FileLongName, 'GPSLongitudeRef') as GPSLongitudeRef,
    Attribute(FileLongName, 'GPSLongitude') as GPSLongitude,
    Attribute(FileLongName, 'GPSAltitudeRef') as GPSAltitudeRef,

    //  examples: 0=Above sea level, 1=Below sea level,
    Attribute(FileLongName, 'GPSAltitude') as GPSAltitude,
    Attribute(FileLongName, 'GPSTimeStamp') as GPSTimeStamp,
    Attribute(FileLongName, 'GPSSatellites') as GPSSatellites,
    Attribute(FileLongName, 'GPSStatus') as GPSStatus,
    Attribute(FileLongName, 'GPSMeasureMode') as GPSMeasureMode,
    Attribute(FileLongName, 'GPSDOP') as GPSDOP,
    Attribute(FileLongName, 'GPSSpeedRef') as GPSSpeedRef,

    Attribute(FileLongName, 'GPSSpeed') as GPSSpeed,
    Attribute(FileLongName, 'GPSTrackRef') as GPSTrackRef,
    Attribute(FileLongName, 'GPSTrack') as GPSTrack,
    Attribute(FileLongName, 'GPSImgDirectionRef') as GPSImgDirectionRef,
    Attribute(FileLongName, 'GPSImgDirection') as GPSImgDirection,
    Attribute(FileLongName, 'GPSMapDatum') as GPSMapDatum,
    Attribute(FileLongName, 'GPSDestLatitudeRef') as GPSDestLatitudeRef,

    Attribute(FileLongName, 'GPSDestLatitude') as GPSDestLatitude,
    Attribute(FileLongName, 'GPSDestLongitudeRef') as GPSDestLongitudeRef,
    Attribute(FileLongName, 'GPSDestLongitude') as GPSDestLongitude,
    Attribute(FileLongName, 'GPSDestBearingRef') as GPSDestBearingRef,
    Attribute(FileLongName, 'GPSDestBearing') as GPSDestBearing,
    Attribute(FileLongName, 'GPSDestDistanceRef') as GPSDestDistanceRef,

    Attribute(FileLongName, 'GPSDestDistance') as GPSDestDistance,
    Attribute(FileLongName, 'GPSProcessingMethod') as GPSProcessingMethod,
    Attribute(FileLongName, 'GPSAreaInformation') as GPSAreaInformation,
```

```
    Attribute(FileLongName, 'GPSDateStamp') as GPSDateStamp,
    Attribute(FileLongName, 'GPSDifferential') as GPSDifferential;

    //  examples: 0=No correction, 1=Differential correction,
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels);
Next vFoundFile
Next vExt
```

Example 3: Windows メディア ファイル

このスクリプトでは、存在する可能性のあるすべての WMA/WMV ASF メタ タグを、*MyMusic* フォルダで読み込みます。

```
/ Script to read WMA/WMV ASF meta tags
for each vExt in 'asf', 'wma', 'wmv'
for each vFoundFile in filelist( GetFolderPath('MyMusic') & '\*.'& vExt )

FileList:
LOAD FileLongName,
    subfield(FileLongName,'\',-1) as FileShortName,
    num(FileSize(FileLongName),'# ### ### ###',',',' ') as FileSize,
    FileTime(FileLongName) as FileTime,
    Attribute(FileLongName, 'Title') as Title,
    Attribute(FileLongName, 'Author') as Author,
    Attribute(FileLongName, 'Copyright') as Copyright,
    Attribute(FileLongName, 'Description') as Description,

    Attribute(FileLongName, 'Rating') as Rating,
    Attribute(FileLongName, 'PlayDuration') as PlayDuration,
    Attribute(FileLongName, 'MaximumBitrate') as MaximumBitrate,
    Attribute(FileLongName, 'WMFSDKVersion') as WMFSDKVersion,
    Attribute(FileLongName, 'WMFSDKNeeded') as WMFSDKNeeded,
    Attribute(FileLongName, 'IsVBR') as IsVBR,
    Attribute(FileLongName, 'ASFLeakyBucketPairs') as ASFLeakyBucketPairs,

    Attribute(FileLongName, 'PeakValue') as PeakValue,
    Attribute(FileLongName, 'AverageLevel') as AverageLevel;
LOAD @1:n as FileLongName Inline "$(vFoundFile)" (fix, no labels);
Next vFoundFile
Next vExt
```

Example 4: PNG

このスクリプトでは、存在する可能性のあるすべての PNG メタ タグを、*MyPictures* フォルダで読み込みます。

```
// 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() 関数は、ODBC 接続または OLE DB 接続のアクティブなデータ接続の名前を返します。connections.**connect** ステートメントが実行されていない場合、または **disconnect** ステートメントの実行後は、空の文字列を返します。

構文:

```
ConnectString()
```

例と結果:

スクリプトの例

例	結果

`LIB CONNECT TO 'Tutorial ODBC';`
 	
`ConnectString:`
 	
`Load ConnectString() as ConnectString AutoGenerate 1;`
	項目 ConnectString の「Tutorial ODBC」を返します。
	
	この例は、Tutorial ODBC という名前の利用可能なデータ接続があることを前提としています。

FileBaseName

FileBaseName 関数は、現在読み取り中のテーブル ファイルのファイル名を、パスや拡張子を省略した文字列で返します。

構文:

```
FileBaseName()
```

例と結果:

スクリプトの例

例	結果

`LOAD *, filebasename( ) as X from`
 	
`C:\UserFiles\abc.txt`
	読み込まれた各レコードの項目 X に「abc」を返します。

FileDir

FileDir 関数は、現在読み取り中のテーブル ファイルのディレクトリパスを文字列で返します。

構文:

```
FileDir()
```

この関数は、標準モードのフォルダデータ接続のみに対応しています。

例と結果:

スクリプトの例

例	結果
Load *, filedir( ) as X from C:\UserFiles\abc.txt	読み込まれた各レコードの項目 X に 'C:\UserFiles' を返します。

FileExtension

FileExtension 関数は、現在読み取り中のテーブル ファイルの拡張子を文字列で返します。

構文:

```
FileExtension()
```

例と結果:

スクリプトの例

例	結果
LOAD *, FileExtension( ) as X from C:\UserFiles\abc.txt	読み込まれた各レコードの項目 X に「txt」を返します。

FileName

FileName 関数は、現在読み取り中のテーブル ファイルのファイル名を、パスを省略し、拡張子を付けて文字列で返します。

構文:

```
FileName()
```

例と結果:

スクリプトの例

例	結果
LOAD *, FileName( ) as X from C:\UserFiles\abc.txt	読み込まれた各レコードの項目 X に ''abc.txt'' を返します。

FilePath

FilePath 関数は、現在読み取り中のテーブル ファイルのフル パスを文字列で返します。

構文:

FilePath()

この関数は、標準モードのフォルダ データ接続のみに対応しています。

例と結果:

スクリプトの例

例	結果
Load *, FilePath( ) as X from C:\UserFiles\abc.txt	読み込まれた各レコードの項目 X に ''C:\UserFiles\abc.txt'' を返します。

FileSize

FileSize 関数は、filename ファイル のサイズをバイト数で表した整数を返します。filename が指定されていない場合は、現在読み取り中のテーブル ファイルのサイズを返します。

構文:

FileSize([filename]**)**

引数:

引数

引数	説明
filename	フォルダーまたは Web ファイル データ接続としてのファイルの名前 (必要に応じてパスも含む)。ファイル名を指定しない場合は、現在読み取られているテーブル ファイルが使用されます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***

例と結果:

スクリプトの例

例	結果
`LOAD *, FileSize( ) as X from abc.txt;`	読み込まれた各レコードの項目 X に、指定されたファイル (abc.txt) のサイズを整数で返します。
`FileSize( 'lib://DataFiles/xyz.xls' )`	xyz.xls のファイル サイズを返します。

FileTime

FileTime 関数は、指定されたファイルの最後に更新された日付と時刻を UTC フォーマットで返します。ファイルが指定されていない場合、関数は現在読み込まれているテーブル ファイルの最後に更新された日付と時刻を返します。

構文:

```
FileTime([ filename ])
```

引数:

引数

引数	説明
filename	ファイルの名前で、場合によってはフォルダまたはウェブ ファイル データ接続のようなパスも含まれます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***

例と結果:

スクリプトの例

例	結果
`LOAD *, FileTime( ) as X from abc.txt;`	読み込まれた各レコードの項目 X に、ファイル (abc.txt) の最終更新のタイムスタンプを返します。
`FileTime( 'xyz.xls' )`	ファイル xyz.xls の最終更新のタイムスタンプを返します。

GetFolderPath

GetFolderPath 関数は、Microsoft Windows *SHGetFolderPath* 関数の値を返します。この関数は、Microsoft Windows フォルダの名前を入力として返し、フォルダのフルパスを返します。

この関数は標準モードに対応していません。。

構文:

```
GetFolderPath(foldername)
```

引数:

引数

引数	説明
foldername	Microsoft Windows フォルダの名前。 フォルダ名には、空白を含めないでください。Windows Explorer でのフォルダ名の空白は、フォルダ名から削除する必要があります。 例: *MyMusic* *MyDocuments*

例と結果:

この例では、次の Microsoft Windows フォルダのパスを取得することが目標です。*MyMusic* および *MyPictures*、*Windows*。例のスクリプトをアプリに追加し、リロードします。

```
LOAD
 GetFolderPath('MyMusic') as MyMusic,
 GetFolderPath('MyPictures') as MyPictures,
 GetFolderPath('Windows') as Windows
AutoGenerate 1;
```

アプリをリロードすると、項目 *MyMusic* および *MyPictures*、*Windows* がデータモデルに追加されます。各項目には、入力で定義されたフォルダへのパスが含まれます。例:

- *C:\Users\smu\Music* for the folder *MyMusic*
- *C:\Users\smu\Pictures* for the folder *MyPictures*
- *C:\Windows* for the folder *Windows*

QvdCreateTime

このスクリプト関数は、QVD ファイルに含まれた XML ヘッダーの日付と時刻を返します (ない場合は NULL を返します)。タイムスタンプでは、時刻は UTC で提供されます。

構文:

```
QvdCreateTime(filename)
```

引数:

引数

引数	説明
filename	QVD ファイルの名前で、必要な場合はフォルダや Web ファイル データ接続といったパスも含まれます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***

```
QvdCreateTime('MyFile.qvd')

QvdCreateTime('C:\MyDir\MyFile.qvd')

QvdCreateTime('lib://DataFiles/MyFile.qvd')
```

QvdFieldName

このスクリプト関数は、QVDファイルの項目番号**fieldno**の名前を返します。項目が存在しない場合は、NULLを返します。

構文:

```
QvdFieldName(filename , fieldno)
```

引数:

引数

引数	説明
filename	QVD ファイルの名前で、必要な場合はフォルダや Web ファイル データ接続といったパスも含まれます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***
fieldno	QVDファイルに含まれる表内の項目の番号です。

```
QvdFieldName ('MyFile.qvd', 5)

QvdFieldName ('C:\MyDir\MyFile.qvd', 5)

QvdFieldName ('lib://DataFiles/MyFile.qvd', 5)
```
3 つの例はすべて、QVDファイルに含まれる表の 5 番目の項目の名前を返します。

QvdNoOfFields

このスクリプト関数は、QVD ファイル内の項目数を返します。

構文:

```
QvdNoOfFields(filename)
```

引数:

引数

引数	説明
filename	QVD ファイルの名前で、必要な場合はフォルダや Web ファイル データ接続といったパスも含まれます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***

```
QvdNoOfFields ('MyFile.qvd')

QvdNoOfFields ('C:\MyDir\MyFile.qvd')

QvdNoOfFields ('lib://DataFiles/MyFile.qvd')
```

QvdNoOfRecords

このスクリプト関数は、QVD ファイル内に含まれるレコードの数を返します。

構文:

```
QvdNoOfRecords(filename)
```

引数:

引数

引数	説明
filename	QVD ファイルの名前で、必要な場合はフォルダや Web ファイル データ接続といったパスも含まれます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***

```
QvdNoOfRecords ('MyFile.qvd')

QvdNoOfRecords ('C:\MyDir\MyFile.qvd')

QvdNoOfRecords ('lib://DataFiles/MyFile.qvd')
```

QvdTableName

このスクリプト関数は、QVD ファイルに保存されているテーブルの名前を返します。

構文:

```
QvdTableName(filename)
```

引数:

引数

引数	説明
filename	QVD ファイルの名前で、必要な場合はフォルダや Web ファイル データ接続といったパスも含まれます。 ***'lib://Table Files/'*** レガシー スクリプト モードは、次のパス形式にも対応しています。 • 絶対パス ***c:\data*** • Qlik Sense アプリ作業ディレクトリへの相対パス。 ***data*** • インターネットまたはイントラネット上の位置を示す URL アドレス (HTTP あるいは FTP)。 ***http://www.qlik.com***

```
QvdTableName ('MyFile.qvd')

QvdTableName ('C:\MyDir\MyFile.qvd')

QvdTableName ('lib://data\MyFile.qvd')
```

8.11 財務関数

財務関数は、データロード スクリプトおよびチャート式で使用可能で、支払いと金利を計算します。
すべての引数で、支払う現金は負の数で表します。受領する現金は正の数で表します。
財務関数に使用される引数をリストアップします (**range** で始まるもの以外)。

*すべての財務関数において重要なのは、**rate**と**nper** に同じ単位を指定することです。年利 6 % の 5 年ローンの月賦には、**rate** に 0.005 (6%/12)、**nper** に 60 (5*12) を使用します。同じローンの年賦の場合は、**rate** に 6%、**nper** に 5 を使用します。*

財務関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

FV
この関数は、定期的な一定の支払と年間の単利に基づき、投資の将来価値を返します。

```
FV (rate, nper, pmt [ ,pv [ , type ] ])
```

nPer
この関数は、定期、定額支払、固定金利での投資の期間数を返します。

```
nPer (rate, pmt, pv [ ,fv [ , type ] ])
```

Pmt
この関数は、定期、定額支払、固定金利でのローンの支払額を返します。年金の期間内で変更することはできません。支払額は、-20 のように負の数で指定されます。

```
Pmt (rate, nper, pv [ ,fv [ , type ] ] )
```

PV
この関数は、投資の現在価値を返します。

```
PV (rate, nper, pmt [ ,fv [ , type ] ])
```

Rate
この関数は、年金の期間あたりの利率を返します。結果は、**Fix** 小数点 2 桁と% のデフォルトの数値書式で返されます。

```
Rate (nper, pmt , pv [ ,fv [ , type ] ])
```

BlackAndSchole

Black and Scholes モデルは、金融派生商品の数学的モデルです。この方程式は、オプションの理論値を計算します。Qlik Sense の **BlackAndSchole** 関数は、Black and Scholes オリジナル方程式 (ヨーロッパ スタイル オプション) に基づいて値を返します。

```
BlackAndSchole(strike , time_left , underlying_price , vol , risk_free_rate , type)
```

戻り値データ型: 数値

引数:

引数

引数	説明
strike	将来の株の購入価格です。
time_left	残存期間です。
underlying_price	株の時価です。

引数	説明
vol	期間あたりの予想変動率 (株の価格) の10進形式のパーセンテージ単位の表現です。
risk_free_rate	期間あたりのリスクフリー利回りの10進形式のパーセンテージ単位の表現です。
call_or_put	オプションのタイプ: コール オプションの場合は、'c'、'call' または任意のゼロでない数値、 プット オプションの場合は、'p'、'put'、または '0' です。

制限事項:

strike、time_leftおよびunderlying_priceの値は、ゼロより大きい値にする必要があります。

volおよびrisk_free_rateの値は、ゼロより小さいかまたはゼロより大きい値にする必要があります。

例と結果:

スクリプトの例

例	結果
`BlackAndSchole(130, 4, 68.5, 0.4, 0.04, 'call')` これは、時価 68.5 の株を4年以内に1株あたり130で購入するオプションの理論価格を計算します。方程式では、予想変動率を年 0.4 (40%)、リスクフリー利回りを0.04 (4%) と仮定しています。	11.245 を返します

FV

この関数は、定期的な一定の支払と年間の単利に基づき、投資の将来価値を返します。

構文:

```
FV(rate, nper, pmt [ ,pv [ , type ] ])
```

戻り値データ型: 数値既定では、結果は通貨としてフォーマットされます。.

引数:

引数

引数	説明
rate	期間あたりの利率。
nper	年金の支払期間の総数。
pmt	各期間の支払額。年金の期間内で変更することはできません。支払額は、-20 のように負の数で指定されます。

引数	説明
pv	現在価値、つまり将来の一連の支払額が現在持つ価値の合計額です。**pv** が省略されている場合、0 (ゼロ) と見なされます。
type	支払い期限が期末の場合は 0 で、支払い期限が期首の場合は 1 です。**type** は省略されると、0 と見なされます。

例と結果:

スクリプトの例

例	結果
月々 $20 の 36 回払いで、新しい家電製品を買うとします。年率 6% です。請求書は毎月末に届きます。最終支払が行われた時点での投資総額はいくらでしょうか。 `FV(0.005,36,-20)`	$786.72 を返します

nPer

この関数は、定期、定額支払、固定金利での投資の期間数を返します。

構文:

```
nPer(rate, pmt, pv [ ,fv [ , type ] ])
```

戻り値データ型: 数値

引数:

引数

引数	説明
rate	期間あたりの利率。
nper	年金の支払期間の総数。
pmt	各期間の支払額。年金の期間内で変更することはできません。支払額は、-20 のように負の数で指定されます。
pv	現在価値、つまり将来の一連の支払額が現在持つ価値の合計額です。**pv** が省略されている場合、0 (ゼロ) と見なされます。
fv	将来価値、または最終支払が行われた後の目標とする現金残高です。**fv** は省略されると、0 と見なされます。
type	支払い期限が期末の場合は 0 で、支払い期限が期首の場合は 1 です。**type** は省略されると、0 と見なされます。

例と結果:

スクリプトの例

例	結果
月々 $20 で、家電製品を買うとします。年率 6% です。請求書は毎月末に届きます。最終支払が行われた後の受領額が $800 になるためには何期必要でしょうか。 `nPer(0.005,-20,0,800)`	36.56 を返します

Pmt

この関数は、定期、定額支払、固定金利でのローンの支払額を返します。年金の期間内で変更することはできません。支払額は、-20 のように負の数で指定されます。

```
Pmt(rate, nper, pv [ ,fv [ , type ] ] )
```

戻り値データ型: 数値既定では、結果は通貨としてフォーマットされます。.

ローン期間の支払総額を算出するには、返された **pmt** の値に **nper** を掛けます。

引数:

引数

引数	説明
rate	期間あたりの利率。
nper	年金の支払期間の総数。
pv	現在価値、つまり将来の一連の支払額が現在持つ価値の合計額です。**pv** が省略されている場合、0 (ゼロ) と見なされます。
fv	将来価値、または最終支払が行われた後の目標とする現金残高です。**fv** は省略されると、0 と見なされます。
type	支払い期限が期末の場合は 0 で、支払い期限が期首の場合は 1 です。**type** は省略されると、0 と見なされます。

例と結果:

スクリプトの例

例	結果
次の方程式は、$20,000 のローンを年率 10 %、8 か月で完済する場合の月々の支払額を返します。 `Pmt(0.1/12,8,20000)`	-$2,594.66 を返します
同じローンで、支払期日が期首である場合の支払は次のようになります。 `Pmt(0.1/12,8,20000,0,1)`	-$2,573.21 を返します

PV

この関数は、投資の現在価値を返します。

```
PV(rate, nper, pmt [ ,fv [ , type ] ])
```

戻り値データ型: 数値既定では、結果は通貨としてフォーマットされます。.

現在価値とは、将来的な一連の支払いの現在の価値総額です。例えば、借金をしている場合、その融資額が貸手にとっての現在価値です。

引数:

引数

引数	説明
rate	期間あたりの利率。
nper	年金の支払期間の総数。
pmt	各期間の支払額。年金の期間内で変更することはできません。支払額は、-20 のように負の数で指定されます。
fv	将来価値、または最終支払が行われた後の目標とする現金残高です。**fv** は省略されると、0 と見なされます。
type	支払い期限が期末の場合は 0 で、支払い期限が期首の場合は 1 です。**type** は省略されると、0 と見なされます。

例と結果:

スクリプトの例

例	結果
月末ごとに $100 が 7% の利率で 5 年間にわたって支払われるとすると、現在の負債の額はいくらでしょうか。 `PV(0.07/12,12*5,-100,0,0)`	$5,050.20 を返します

Rate

この関数は、年金の期間あたりの利率を返します。結果は、**Fix** 小数点 2 桁と % のデフォルトの数値書式で返されます。

構文:

```
Rate(nper, pmt , pv [ ,fv [ , type ] ])
```

戻り値データ型: 数値

rate は、反復によって計算され、ゼロまたは複数の解を持つ場合があります。**rate** の連続的な結果が収束しない場合は、NULL 値が返されます。

引数:

引数

引数	説明
nper	年金の支払期間の総数。
pmt	各期間の支払額。年金の期間内で変更することはできません。支払額は、-20 のように負の数で指定されます。
pv	現在価値、つまり将来の一連の支払額が現在持つ価値の合計額です。**pv** が省略されている場合、0 (ゼロ) と見なされます。
fv	将来価値、または最終支払が行われた後の目標とする現金残高です。**fv** は省略されると、0 と見なされます。
type	支払い期限が期末の場合は 0 で、支払い期限が期首の場合は 1 です。**type** は省略されると、0 と見なされます。

例と結果:

スクリプトの例

例	結果
期間が 5 年の $10,000 の年金ローンで、月々の支払額が $300 の場合の利率はいくらでしょうか。 `Rate(60,-300,10000)`	2.00% を返します

8.12 書式設定関数

書式設定関数は、入力数値項目または数式に表示形式を適用します。データ型に応じて、小数点の記号、3 桁区切りの記号などの文字を指定できます。

この関数は、すべて文字列と数値の両方を持つデュアル値を返しますが、数値から文字列への変換を実行するものとみなすことができます。**Dual()** は特別なケースですが、その他の書式設定関数は入力式の数値を取得し、数値を表す文字列を生成します。

それに対して、変換関数は上記と正反対のことを行います。つまり、文字列式を取得し、それを数値として評価し、結果として返される数値の書式を指定します。

この関数は、データロードスクリプトおよびチャート式の両方で使用できます。

すべての数値表現で小数点に小数点の記号を使用しています。

書式設定関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

ApplyCodepage

ApplyCodepage() は、数式に記載された項目やテキストに異なるコードページ文字を適用します。
codepage 引数は数値形式でなければなりません。

```
ApplyCodepage (text, codepage)
```

Date

Date() は、データロードスクリプトのシステム変数またはオペレーティング システムの書式設定、または提供されている場合は書式文字列を使用して、数式を日付として書式設定します。

```
Date (number[, format])
```

Dual

Dual() は、数値と文字列を組み合わせて1つのレコードにし、そのレコードの数値表現をソートや計算に使用できるようする一方で、文字列値を表示に使用できるようにします。

```
Dual (text, number)
```

Interval

Interval() は、データロードスクリプトのシステム変数またはオペレーティング システムの書式、または提供されている場合は書式文字列を使用して、数値を時間間隔として書式設定します。

```
Interval (number[, format])
```

Money

Money() は、データロードスクリプトのシステム変数またはオペレーティング システムの書式設定 (書式文字列が提供されている場合を除く)、およびオプションの小数点記号と千の桁区切りを使用して数式を数字で金額値として書式設定します。

```
Money (number[, format[, dec_sep [, thou_sep]]])
```

Num

Num() は数値をフォーマットします。つまり、2番目のパラメーターで指定された形式を使用して、入力の数値をテキストを表示するように変換します。2番目のパラメーターを省略すると、データロードスクリプトで設定された10進数と1000進数の区切り文字が使用されます。カスタムの小数点および桁区切り記号は、オプションのパラメータです。

```
Num (number[, format[, dec_sep [, thou_sep]]])
```

Time

Time() は、書式文字列が提供されている場合を除き、データロードスクリプトのシステム変数またはオペレーティング システムの時刻書式設定を使用して、数式を時刻値として書式設定します。

```
Time (number[, format])
```

Timestamp

TimeStamp() は、書式文字列が提供されている場合を除き、データロードスクリプトのシステム変数またはオペレーティング システムのタイムスタンプ書式設定を使用して、数式を日付と時刻の値として書式設定します。

```
Timestamp (number[, format])
```

参照先:

- *変換関数 (page 1241)*

ApplyCodepage

ApplyCodepage() は、数式に記載された項目やテキストに異なるコードページ文字を適用します。**codepage** 引数は数値形式でなければなりません。

ApplyCodepage はチャート数式で使用できますが、一般的にはデータロードエディターでスクリプト関数として使用されます。例えば、制御できない別の文字セットで保存されている可能性があるファイルをロードするときには、必要な文字セットを表すコードページを適用できます。

構文:

```
ApplyCodepage(text, codepage)
```

戻り値データ型: string

引数:

引数

引数	説明
text	別のコードページを適用する項目またはテキスト(引数**codepage** で指定)。
codepage	**text** で指定される項目または数式に適用されるコードページを表す数値。

例と結果:

スクリプトの例

例	結果
LOAD ApplyCodepage(ROWX,1253) as GreekProduct, ApplyCodepage (ROWY, 1255) as HebrewProduct, ApplyCodepage (ROWZ, 65001) as EnglishProduct; SQL SELECT ROWX, ROWY, ROWZ From Products;	ソースをSQL からロードするときには、UTF-8 形式とは異なる文字セット(キリル、ヘブライなど) が混在している場合があります。このようなソースは 1 行ずつロードして、行ごとに異なるコードページを適用する必要があります。 **codepage** の値 1253 は Windows のギリシャ文字セット、値 1255 はヘブライ、および値 65001 は標準的なラテンUTF-8 文字を表しています。

参照先: 文字セット *(page 166)*

Date

Date() は、データロードスクリプトのシステム変数またはオペレーティング システムの書式設定、または提供されている場合は書式文字列を使用して、数式を日付として書式設定します。

構文:

```
Date(number[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
number	書式設定する数値。
format	結果文字列の形式を説明する文字列。書式文字列が提供されていない場合は、データロードスクリプトまたはオペレーティング システムのシステム変数に設定されている日付形式が使用されます。

例と結果:
この例では、次のデフォルト設定を前提としています。

- 日付の設定 1: YY-MM-DD
- 日付の設定 2: M/D/YY

```
Date( A )
```
A=35648 の場合

結果テーブル

結果	設定 1	設定 2
文字列:	97-08-06	8/6/97
数値:	35648	35648

```
Date( A, 'YY.MM.DD' )
```
A=35648 の場合

結果テーブル

結果	設定 1	設定 2
文字列:	97.08.06	97.08.06
数値:	35648	35648

```
Date( A, 'DD.MM.YYYY' )
```
A=35648.375 の場合

結果テーブル

結果	設定 1	設定 2
文字列:	06.08.1997	06.08.1997
数値:	35648.375	35648.375

```
Date( A, 'YY.MM.DD' )
```
A=8/6/97 の場合

結果テーブル

結果	設定 1	設定 2
文字列:	NULL (なし)	97.08.06
数値:	NULL	35648

Dual

Dual() は、数値と文字列を組み合わせて1つのレコードにし、そのレコードの数値表現をソートや計算に使用できるようする一方で、文字列値を表示に使用できるようにします。

構文:

```
Dual(text, number)
```

戻り値データ型: デュアル

デュアルの戻り値はすべて右揃えです。

引数:

引数

引数	説明
text	数値引数と組み合わせて使用される文字列値。
number	文字列引数の文字列と組み合わせて使用される数値。

Qlik Sense では、すべての項目値がデュアル値になる可能性があります。つまり、項目値には、数値とテキスト値の両方を含むことができることを意味します。この一例となるのが日付で、数値の 40908 とテキスト表記の '2011-12-31' の両方が可能です。

1 つの項目に読み込まれている複数のデータアイテムで、文字列表現が異なっていても同じ有効な数値が表現があるなら、それらはすべて最初に出現した文字列表現を共有します。

***dual** 関数は、一般的にスクリプトの早い段階、関係する項目に他のデータが読み取られる前に使用され、フィルターパネルなどで表示される最初の文字列表現を作成します。*

例と結果:

スクリプトの例

例	説明

スクリプトに次の例を追加して実行します。
`Load dual ( NameDay,NumDay ) as DayOfWeek inline`
`[ NameDay,NumDay`
`Monday,0`
`Tuesday,1`
`Wednesday,2`
`Thursday,3`
`Friday,4`
`Saturday,5`
`Sunday,6 ];`
	項目 DayOfWeek は、たとえば、ビジュアライゼーションで軸として使用できます。曜日を持つテーブルでは、アルファベット順ではなく、正しいシーケンス番号に自動的にソートされます。

例	説明
`Load Dual('Q' & Ceil (Month(Now())/3), Ceil (Month(Now())/3)) as Quarter AutoGenerate 1;`	この例では、現在の四半期を取得します。**Now()** 関数が年の最初の 3 ヶ月で実行される場合、Q1 と表示され、年の 2 番目の 3 ヶ月で実行される場合、Q2 と表示され、以下同様に表示されます。ただし、ソートで使用する場合、Quarter は数値 1 ~ 4 として動作します。
`Dual('Q' & Ceil(Month (Date)/3), Ceil(Month (Date)/3)) as Quarter`	上の例に示されているように、項目 Quarter はテキスト値 'Q1' ~ 'Q4' を用いて作成され、数値 1 ~ 4 が割り当てられます。これをスクリプトで使用するには、Date の値をロードする必要があります。
`Dual(WeekYear(Date) & '-W' & Week(Date), WeekStart(Date)) as YearWeek`	この例では、項目 YearWeek ('2012-W22' 形式のテキスト値 を持つ) を作成し、同時にその週の最初の日の日付値に対応する数値を割り当てます (たとえば、41057。これをスクリプトで使用するには、Date の値をロードする必要があります。

Interval

Interval() は、データロードスクリプトのシステム変数またはオペレーティング システムの書式、または提供されている場合は書式文字列を使用して、数値を時間間隔として書式設定します。

時間間隔は、時間や日数、あるいは日、時、分、秒、それ以下の時間区分の組み合わせとして書式設定できます。

構文:

```
Interval(number[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
number	書式設定する数値。
format	結果として返される間隔文字列の書式を設定する方法を記述する文字列。省略されている場合は、オペレーティング システムで設定された短い日付書式、時間書式、および小数点記号が使用されます。

例と結果:
この例では、次のデフォルト設定を前提としています。

- 日付の書式設定 1: YY-MM-DD
- 日付の書式設定 2: hh:mm:ss
- 小数点記号:

結果テーブル

例	文字列	数値
Interval(A) ここで A=0.375	09:00:00	0.375
Interval(A) ここで A=1.375	33:00:00	1.375
Interval(A, 'D hh:mm') ここで A=1.375	1 09:00	1.375
Interval(A-B, 'D hh:mm') ここで A=97-08-06 09:00:00、B=96-08-06 00:00:00	365 09:00	365.375

Money

Money() は、データロードスクリプトのシステム変数またはオペレーティング システムの書式設定 (書式文字列が提供されている場合を除く)、およびオプションの小数点記号と千の桁区切りを使用して数式を数字で金額値として書式設定します。

構文:

```
Money(number[, format[, dec_sep[, thou_sep]]])
```

戻り値データ型: dual

引数:

引数

引数	説明
number	書式設定する数値。
format	結果として返される通貨文字列の書式を設定する方法を記述する文字列。
dec_sep	小数点記号を指定する文字列。
thou_sep	3 桁区切りの記号を指定する文字列。

引数 2 ~ 4 が省略されている場合は、オペレーティング システムで設定されている通貨書式が使用されます。

例と結果:
この例では、次のデフォルト設定を前提としています。

- MoneyFormat setting 1:kr ##0,00, MoneyThousandSep' '
- MoneyFormat setting 2:$ #,##0.00, MoneyThousandSep','

Money(A)
ここで A=35648

結果テーブル

結果	設定 1	設定 2
文字列:	kr 35 648,00	$ 35,648.00
数値:	35648.00	35648.00

```
Money( A, '#,##0 ¥', '.' , ',' )
```
ここで A=3564800

結果テーブル

結果	設定 1	設定 2
文字列:	3,564,800 ¥	3,564,800 ¥
数値:	3564800	3564800

Num

Num() は数値をフォーマットします。つまり、2 番目のパラメーターで指定された形式を使用して、入力の数値をテキストを表示するように変換します。2 番目のパラメーターを省略すると、データロードスクリプトで設定された 10 進数と 1000 進数の区切り文字が使用されます。カスタムの小数点および桁区切り記号は、オプションのパラメータです。

構文:

```
Num(number[, format[, dec_sep [, thou_sep]]])
```

戻り値データ型: dual

Num 関数は、文字列と数値の両方が指定されたデュアル値を返します。この関数は、入力式の数値を取得し、数値を表す文字列を生成します。

引数:

引数

引数	説明
number	書式設定する数値。
format	結果として返される文字列の書式を設定する方法を指定する文字列。省略した場合、データロードスクリプトで設定されている 10 進数と 1000 進数の区切り文字が使用されます。
dec_sep	小数点記号を指定する文字列。省略した場合、データロードスクリプトで設定されている変数 DecimalSep の値が使用されます。
thou_sep	3 桁区切りの記号を指定する文字列。省略されている場合は、データロードスクリプトで設定された変数 ThousandSep 値が使用されます。

例: チャートの数式

次の表は、項目 A が 35648.312 に等しい場合の結果を示しています。

結果

ライン番号の隣にある	結果
Num(A)	35648.312 (スクリプト内の環境変数に依存)
Num(A, '0.0', '.')	35648.3
Num(A, '0,00', ',')	35648,31
Num(A, '#,##0.0', '.', ',')	35,648.3
Num(A, '# ##0', ',', ' ')	35 648

例: ロードスクリプト

ロードスクリプト

Num は、ロードスクリプトで千の桁区切りと小数点の記号がすでに設定されている場合であっても、そのロードスクリプトで使用して数字の書式を設定できます。以下のロードスクリプトには、特定の千の桁区切りと小数点の記号が含まれていますが、*Num* を使用して異なる方法でデータの書式を設定します。

データ ロード エディターで、新しいセクションを作成し、サンプル スクリプトを追加して実行します。その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

```
SET ThousandSep=',';
SET DecimalSep='.';
Transactions:
Load
*,
Num(transaction_amount) as [No formatting],
Num(transaction_amount,'0') as [0],
Num(transaction_amount,'#,##0') as [#,##0],
Num(transaction_amount,'# ###,00') as [# ###,00],
Num(transaction_amount,'# ###,00',',',' ') as [# ###,00 , ',' , ' '],
Num(transaction_amount,'#,###.00','.',',') as [#,###.00 , '.' , ','],
Num(transaction_amount,'$#,###.00') as [$#,###.00],
;
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 20180830, 12423.56, 23, 0,2038593, L, Red
3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180916, 15.75, 1, 0.22, 5646471, S, blue
3753, 20180922, 1251, 7, 0, 3036491, l, Black
3754, 20180922, 21484.21, 1356, 75, 049681, xs, Red
3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, Blue
3757, 20180923, 3177.4, 21, .14, 203521, XL, Black
];
```

ロードスクリプトでの *Num* 関数のさまざまな使用法の結果を示した Qlik Sense テーブル。テーブルの 4 列目には、目的など、誤った方法の書式設定が含まれています。

No formatting	**0**	**#,##0**	**# ###,00**	**# ###,00 , ' ' , ' '**	**#,###.00 , ' ' , ' '**	**$#,###.00**
-59.18	-59	-59	-59###,00	-59,18	-59.18	$-59,18
15.75	16	16	16###,00	15,75	15.75	$15,75
1251	1251	1,251	1251###,00	1 251,00	1,251.00	$1,251.00
3177.4	3177	3,177	3177###,00	3 177,40	3,177.40	$3,177.40
5356.31	5356	5,356	5356###,00	5 356,31	5,356.31	$5,356.31
12423.56	12424	12,424	12424###,00	12 423,56	12,423.56	$12,423.56
21484.21	21484	21,484	21484###,00	21 484,21	21,484.21	$21,484.21

例: ロードスクリプト

ロードスクリプト

Num は、ロードスクリプトで使用して、数字をパーセントとして書式設定できます。

データ ロード エディターで、新しいセクションを作成し、サンプル スクリプトを追加して実行します。7その後、結果列に含まれている項目をアプリのシートに追加して結果を表示します。

```
SET ThousandSep=',';
SET DecimalSep='.';
Transactions:
Load
*,
Num(discount,'#,##0%') as [Discount #,##0%]
;
Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, discount,
customer_id, size, color_code
3750, 20180830, 12423.56, 23, 0,2038593, L, Red
3751, 20180907, 5356.31, 6, 0.1, 203521, m, orange
3752, 20180916, 15.75, 1, 0.22, 5646471, S, blue
3753, 20180922, 1251, 7, 0, 3036491, l, Black
3754, 20180922, 21484.21, 1356, 75, 049681, xs, Red
3756, 20180922, -59.18, 2, 0.333333333333333, 2038593, M, Blue
3757, 20180923, 3177.4, 21, .14, 203521, XL, Black
];
```

パーセントの書式設定のためにロードスクリプトで使用されている *Num* 関数の結果を示した Qlik Sense テーブル。

Discount	Discount #,##0%
0.333333333333333	33%
0.22	22%
0	0%
.14	14%
0.1	10%
0	0%
75	7,500%

Time

Time() は、書式文字列が提供されている場合を除き、データロードスクリプトのシステム変数またはオペレーティング システムの時刻書式設定を使用して、数式を時刻値として書式設定します。

構文:

```
Time(number[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
number	書式設定する数値。
format	結果として返される時刻文字列の書式を設定する方法を記述する文字列。省略されている場合は、オペレーティング システムで設定された短い日付書式、時間書式、および小数点記号が使用されます。

例と結果:
この例では、次のデフォルト設定を前提としています。

- 時刻の書式設定 1: hh:mm:ss
- 時刻の書式設定 2: hh.mm.ss

```
Time( A )
```
ここで A=0.375

結果テーブル

結果	設定 1	設定 2
文字列:	09:00:00	09.00.00
数値:	0.375	0.375

```
Time( A )
```
ここで A=35648.375

結果テーブル

結果	設定 1	設定 2
文字列:	09:00:00	09.00.00
数値:	35648.375	35648.375

```
Time( A, 'hh-mm' )
```
ここで A=0.99999

結果テーブル

結果	設定 1	設定 2
文字列:	23-59	23-59
数値:	0.99999	0.99999

Timestamp

TimeStamp() は、書式文字列が提供されている場合を除き、データロードスクリプトのシステム変数またはオペレーティング システムのタイムスタンプ書式設定を使用して、数式を日付と時刻の値として書式設定します。

構文:

```
Timestamp(number[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
number	書式設定する数値。

引数	説明
format	結果として返される日付と時刻文字列の書式を設定する方法を記述する文字列。省略されている場合は、オペレーティング システムで設定された短い日付書式、時間書式、および小数点記号が使用されます。

例と結果:
この例では、次のデフォルト設定を前提としています。

- TimeStampFormat の設定 1: YY-MM-DD hh:mm:ss
- TimeStampFormat の設定 2: M/D/YY hh:mm:ss

```
Timestamp( A )
```
ここで A=35648.375

結果テーブル

結果	設定 1	設定 2
文字列:	97-08-06 09:00:00	8/6/97 09:00:00
数値:	35648.375	35648.375

```
Timestamp( A,'YYYY-MM-DD hh.mm')
```
ここで A=35648

結果テーブル

結果	設定 1	設定 2
文字列:	1997-08-06 00.00	1997-08-06 00.00
数値:	35648	35648

8.13 一般的な数値関数

これらの一般的な数値関数では、引数は数式であり、**x** は実際の数値と解釈されます。すべての関数は、データロードスクリプトおよびチャート式の両方で使用できます。

一般的な数値関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

bitcount
BitCount() は、10進数を2進数表記した場合に1に設定されるビット数を返します。つまり、この関数は **integer_number** に設定されたビット数を返します。**integer_number** は符号付きの32ビットの整数と解釈されます。

```
BitCount(integer_number)
```

div
Div() は、1番目の引数を2番目の引数で割り算して得られる整数部分を返します。パラメータは両方とも実数として解釈されるため、整数である必要はありません。

```
Div (integer_number1, integer_number2)
```

fabs
Fabs() は、**x** の絶対値を返します。結果は正の数値です。

```
Fabs (x)
```

fact
Fact() は、正の整数 **x** の階乗を返します。

```
Fact (x)
```

frac
Frac() は、**x** の小数部を返します。

```
Frac (x)
```

sign
Sign() は、**x** が正の数か0、あるいは負の数かによって1、0、-1を返します。

```
Sign (x)
```

組み合わせ関数と順列関数

combin
Combin() は、**p** アイテムのセットから選択できる **q** 要素の組み合わせの数を返します。次の方程式で表されます:`Combin(p,q) = p! / q!(p-q)!` アイテムの選択順序に意味はありません。

```
Combin (p, q)
```

permut
Permut() は、**p** アイテムのセットから選択できる **q** 順列の数を返します。次の方程式で表されます:`Permut (p,q) = (p)! / (p - q)!` アイテムの選択順序には有意性があります。

```
Permut (p, q)
```

モジュロ関数

fmod

fmod()は、1 番目の引数 (被除数) を 2 番目の引数 (除数) で割り算して得られる整数部分を返す、一般化モジュロ関数です。結果は実数です。引数は両方とも実数として解釈されるため、整数である必要はありません。

```
Fmod (a, b)
```

mod

Mod() は、整数除算による負でない余りを返す、数学的モジュロ関数です。1 番目の引数は被除数で、2 番目の引数は除数で、両方の引数とも整数でなければなりません。

```
Mod (integer_number1, integer_number2)
```

パリティ関数

even

Even() はTrue (-1) を返します。これは、**integer_number** が偶数の整数またはゼロの場合です。これが False (0) を返すのは **integer_number** が奇数の整数である場合で、NULL を返すのは **integer_number** が整数ではない場合です。

```
Even (integer_number)
```

odd

Odd() はTrue (-1) を返します。これは、**integer_number** が奇数の整数またはゼロの場合です。これが False (0) を返すのは **integer_number** が偶数の整数である場合で、NULL を返すのは **integer_number** が整数ではない場合です。

```
Odd (integer_number)
```

丸め関数

ceil

Ceil()は、**step** (**offset** によりシフト) に最も近い倍数に数値を切り上げます。

```
Ceil (x[, step[, offset]])
```

floor

Floor()は、**step** (**offset** によりシフト) に最も近い倍数に数値を切下げます。

```
Floor (x[, step[, offset]])
```

round

Round() は、**offset** によりシフトされた **step** の最も近い倍数で切り上げた/切り下げた結果を返します。

```
Round ( x [ , step [ , offset ]])
```

BitCount

BitCount() は、10進数を2進数表記した場合に1に設定されるビット数を返します。つまり、この関数は **integer_number** に設定されたビット数を返します。**integer_number** は符号付きの32ビットの整数と解釈されます。

構文:

```
BitCount(integer_number)
```

戻り値データ型: integer

例と結果:

例と結果

例	結果
`BitCount ( 3 )`	3はバイナリでは11個のため、2が返されます
`BitCount ( -1 )`	-1はバイナリでは64個の1のため、64が返されます

Ceil

Ceil()は、**step** (**offset** によりシフト) に最も近い倍数に数値を切り上げます。

これは、数値の切り下げに使用する**floor** 関数とは対照的です。

構文:

```
Ceil(x[, step[, offset]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
x	数値を入力します。
step	間隔は増分です。デフォルト値は1です。
offset	ステップ間隔の基準を定義します。デフォルト値は0です。

例と結果:

例と結果

例	結果
`Ceil(2.4 )`	3 を返します この例では、ステップの大きさは 1、ステップ間隔の基準は 0 となっています。 間隔は ...0 < x <=1, 1 < x <= 2, **2< x <=3**, 3< x <=4... です
`Ceil(4.2 )`	5 を返します
`Ceil(3.88 ,0.1)`	3.9 を返します この例では、間隔の大きさは 0.1、ステップ間隔の基準は 0 となっています。 間隔は ... 3.7 < x <= 3.8, **3.8 < x <= 3.9**, 3.9 < x <= 4.0... です
`Ceil(3.88 ,5)`	5 を返します
`Ceil(1.1 ,1)`	2 を返します
`Ceil(1.1 ,1,0.5)`	1.5 を返します この例では、ステップの大きさは 1、オフセットは 0.5 となっています。つまり、ステップ間隔の基準は 0 ではなく 0.5 です。 間隔は ...**0.5 < x <=1.5**, 1.5 < x <= 2.5, 2.5< x <=3.5, 3.5< x <=4.5... です
`Ceil(1.1 ,1,-0.01)`	1.99 を返します 間隔は ...-0.01< x <= 0.99, **0.99< x <= 1.99**, 1.99 < x <=2.99... です

Combin

Combin() は、**p** アイテムのセットから選択できる **q** 要素の組み合わせの数を返します。次の方程式で表されます:`Combin(p,q) = p! / q!(p-q)!` アイテムの選択順序に意味はありません。

構文:

```
Combin(p, q)
```

戻り値データ型: integer

制限事項:

整数以外の項目は切り捨てられます。

例と結果:

例と結果

例	結果
合計 35 個のロトナンバーから7 個の数を選ぶ場合、組み合わせは何通りあるか。 `Combin( 35,7 )`	6,724,520 を返します

Div

Div() は、1 番目の引数を 2 番目の引数で割り算して得られる整数部分を返します。パラメータは両方とも実数として解釈されるため、整数である必要はありません。

構文:

```
Div(integer_number1, integer_number2)
```

戻り値データ型: integer

例と結果:

例と結果

例	結果
`Div( 7,2 )`	3 を返します
`Div( 7.1,2.3 )`	3 を返します
`Div( 9,3 )`	3 を返します
`Div( -4,3 )`	-1 を返します
`Div( 4,-3 )`	-1 を返します
`Div( -4,-3 )`	1 を返します

Even

Even() はTrue (-1) を返します。これは、**integer_number** が偶数の整数またはゼロの場合です。これが False (0) を返すのは **integer_number** が奇数の整数である場合で、NULL を返すのは **integer_number** が整数ではない場合です。

構文:

```
Even(integer_number)
```

戻り値データ型: ブール値

例と結果:

例と結果

例	結果
`Even( 3 )`	False の場合は 0 を返します
`Even( 2 * 10 )`	True の場合は -1 を返します
`Even( 3.14 )`	NULL を返します

Fabs

Fabs() は、**x** の絶対値を返します。結果は正の数値です。

構文:

```
fabs(x)
```

戻り値データ型: 数値

例と結果:

例と結果

例	結果
`fabs( 2.4 )`	2.4 を返します
`fabs( -3.8 )`	3.8 を返します

Fact

Fact() は、正の整数 **x** の階乗を返します。

構文:

```
Fact(x)
```

戻り値データ型: integer

制限事項:

数値 **x** が整数以外の場合は切り捨てられます。正の数でない場合は、NULL を返します。

例と結果:

例と結果

例	結果
`Fact( 1 )`	1 を返します
`Fact( 5 )`	120 を返します (1 * 2 * 3 * 4 * 5 = 120)
`Fact( -5 )`	NULL を返します

Floor

Floor()は、**step** (**offset** によりシフト) に最も近い倍数に数値を切下げます。

これは、数値の切り上げに使用する**ceil** 関数とは対照的です。

構文:

```
Floor(x[, step[, offset]])
```

戻り値データ型: 数値

引数:

引数

引数	説明
x	数値を入力します。
step	間隔は増分です。デフォルト値は 1 です。
offset	ステップ間隔の基準を定義します。デフォルト値は 0 です。

例と結果:

例と結果

例	結果
`Floor(2.4)`	2 を返します In this example, the size of the step is 1 and the base of the step interval is 0. The intervals are ...0 <= x <1, 1 <= x < 2, **2<= x <3**, 3<= x <4....
`Floor(4.2)`	4 を返します

例	結果
Floor(3.88 ,0.1)	3.8 を返します この例では、間隔の大きさは 0.1、ステップ間隔の基準は 0 となっています。 間隔は ... 3.7 <= x < 3.8, **3.8 <= x < 3.9**, 3.9 <= x < 4.0... です
Floor(3.88 ,5)	0 を返します
Floor(1.1 ,1)	1 を返します
Floor(1.1 ,1,0.5)	0.5 を返します この例では、ステップの大きさは 1、オフセットは 0.5 となっています。つまり、ステップ間隔の基準は 0 ではなく 0.5 です。 間隔は ...**0.5 <= x <1.5**, 1.5 <= x < 2.5, 2.5<= x <3.5,... です

Fmod

fmod()は、1 番目の引数 (被除数) を 2 番目の引数 (除数) で割り算して得られる整数部分を返す、一般化モジュロ関数です。結果は実数です。引数は両方とも実数として解釈されるため、整数である必要はありません。

構文:

```
fmod(a, b)
```

戻り値データ型: 数値

引数:

引数

引数	説明
a	被除数
b	除数

例と結果:

例と結果

例	結果
fmod(7,2)	1 を返します
fmod(7.5,2)	1.5 を返します
fmod(9,3)	0 を返します
fmod(-4,3)	-1 を返します
fmod(4,-3)	1 を返します
fmod(-4,-3)	-1 を返します

Frac

Frac() は、**x** の小数部を返します。

小数部分は、Frac(x) + Floor(x) = x と定義されます。つまり、正の数値の小数部分は、数値 (x) と小数部分の前にある整数との差となります。

例: 11.43 の小数部分 = 11.43 - 11 = 0.43

-1.4 などの負の数の場合、Floor(-1.4) = -2 となり、次の結果が得られます。

-1.4 の小数部分 = 1.4 - (-2) = -1.4 + 2 = 0.6

構文:

```
Frac(x)
```

戻り値データ型: 数値

引数:

引数

引数	説明
x	返す小数部の値

例と結果:

例と結果

例	結果
Frac(11.43)	0.43 を返します
Frac(-1.4)	0.6 を返します
日付と時刻の数値表現から時間成分を抽出し、日付を省略します。 Time(Frac(44518.663888889))	[3:56:00 PM] を返します

Mod

Mod() は、整数除算による負でない余りを返す、数学的モジュロ関数です。1 番目の引数は被除数で、2 番目の引数は除数で、両方の引数とも整数でなければなりません。

構文:

```
Mod(integer_number1, integer_number2)
```

戻り値データ型: integer

制限事項:

integer_number2 は、0 よりも大きい値でなければなりません。

例と結果:

例と結果

例	結果
`Mod( 7,2 )`	1 を返します
`Mod( 7.5,2 )`	NULL を返します
`Mod( 9,3 )`	0 を返します
`Mod( -4,3 )`	2 を返します
`Mod( 4,-3 )`	NULL を返します
`Mod( -4,-3 )`	NULL を返します

Odd

Odd() はTrue (-1) を返します。これは、**integer_number** が奇数の整数またはゼロの場合です。これが False (0) を返すのは **integer_number** が偶数の整数である場合で、NULL を返すのは **integer_number** が整数ではない場合です。

構文:

```
Odd(integer_number)
```

戻り値データ型: ブール値

例と結果:

例と結果

例	結果
`Odd( 3 )`	True の場合は -1 を返します
`Odd( 2 * 10 )`	False の場合は 0 を返します
`Odd( 3.14 )`	NULL を返します

Permut

Permut() は、**p** アイテムのセットから選択できる **q** 順列の数を返します。次の方程式で表されます:`Permut (p,q) = (p)! / (p - q)!` アイテムの選択順序には有意性があります。

構文:

```
Permut(p, q)
```

戻り値データ型: integer

制限事項:

引数が整数以外の場合、小数点以下は切り捨てられます。

例と結果:

例と結果

例	結果
8 人の選手が参加する100 m 決勝で、金、銀、銅のメダルの分配方法は何通りあるか。 `Permut( 8,3 )`	336 を返します

Round

Round() は、**offset** によりシフトされた **step** の最も近い倍数で切り上げた/切り下げた結果を返します。

値が区間の中心に位置する場合は、切り上げられます。

構文:

```
Round(x[, step[, offset]])
```

戻り値データ型: 数値

> *浮動小数点数を切り上げたり、切り下げるとエラーが生じる場合があります。こうしたエラーは、浮動小数点数が 2 進の有限数で表されることにより生じます。よって、すでに切り上げや切り下げが行われている数を使用して計算されます。作業に支障が生じる場合は、切り下げや切り上げを行う前に数値を乗算し、整数に変換します。*

引数:

引数

引数	説明
x	数値を入力します。
step	間隔は増分です。デフォルト値は 1 です。
offset	ステップ間隔の基準を定義します。デフォルト値は 0 です。

例と結果:

例と結果

例	結果
`Round(3.8 )`	4 を返します この例では、ステップの大きさは 1、ステップ間隔の基準は 0 となっています。 間隔は ...0 <= x <1, 1 <= x < 2, 2<= x <3, **3<= x <4**... です
`Round(3.8,4 )`	4 を返します
`Round(2.5 )`	3 を返します。 この例では、ステップの大きさは 1、ステップ間隔の基準は 0 となっています。 間隔は次の通りとなります: ...0 <= x <1, 1 <= x <2, **2<= x <3**...
`Round(2,4 )`	4 を返します。2 はデフォルトのステップ間隔 4 のちょうど中間のため、切り上げとなります。 この例では、ステップの大きさは 4、ステップ間隔の基準は 0 となっています。 間隔は次の通りとなります: ...**0 <= x <4**, 4 <= x <8, 8<= x <12...
`Round(2,6 )`	0 を返します。2 はステップ間隔 6 の中間よりも小さいため、切り捨てとなります。 この例では、ステップの大きさは 6、ステップ間隔の基準は 0 となっています。 間隔は次の通りとなります: ...**0 <= x <6**, 6 <= x <12, 12<= x <18...
`Round(3.88 ,0.1)`	3.9 を返します この例では、ステップの大きさは 0.1、ステップ間隔の基準は 0 となっています。 間隔は ... 3.7 <= x <3.8, **3.8 <= x <3.9**, 3.9 <= x < 4.0... です
`Round (3.88875,1/1000)`	3.889 を返します この例では、ステップのサイズは 0.001 です。これにより、数値が切り上げられ、小数点以下 3 桁に制限されます。
`Round(3.88 ,5)`	5 を返します
`Round(1.1 ,1,0.5)`	1.5 を返します この例では、ステップの大きさは 1、ステップ間隔の基準は 0.5 となっています。 間隔は ...**0.5 <= x <1.5**, 1.5 <= x <2.5, 2.5<= x <3.5... です

Sign

Sign() は、**x** が正の数か 0、あるいは負の数かによって 1、0、-1 を返します。

構文:

```
Sign(x)
```

戻り値データ型: 数値

制限事項:

数値が見つからない場合は、NULL が返されます。

例と結果:

例と結果

例	結果
`Sign( 66 )`	1 を返します
`Sign( 0 )`	0 を返します
`Sign( - 234 )`	-1 を返します

8.14 地理空間関数

地理空間関数は、マップ ビジュアライゼーションでの地理空間的データの処理に使用します。Qlik Sense は地理空間的データについて GeoJSON 仕様に準拠し、以下をサポートします。

- Point
- Linestring
- Polygon
- Multipolygon

GeoJSON 仕様の詳細については以下を参照してください。
GeoJSON.org

地理空間関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

地理空間関数には、集計と非集計の 2 つのカテゴリがあります。

集計関数は、入力としてジオメトリセット (ポイントまたはエリア) を取得し、1 つのジオメトリを返します。たとえば、複数のエリアを結合し、集計の 1 つの境界をマップに描画することができます。

非集計関数は、1 つのジオメトリを取得し、1 つのジオメトリを返します。たとえば、関数 GeoGetPolygonCenter() で、1 つのエリアの境界ジオメトリを入力として設定すると、そのエリアの中心のポイント ジオメトリ(経度と緯度) が返されます。

次の関数は集計関数です。

GeoAggrGeometry
GeoAggrGeometry() は、多くのサブリージョンを1つのリージョンにまとめるなど、多数のエリアを1つの大きなエリアに集計します。

```
GeoAggrGeometry (field_name)
```

GeoBoundingBox
GeoBoundingBox() は、ジオメトリをエリアに集計し、すべての座標を含む最小境界ボックスを計算します。

```
GeoBoundingBox (field_name)
```

GeoCountVertex
GeoCountVertex() は、ポリゴンジオメトリに含まれる頂点の数を特定します。

```
GeoCountVertex(field_name)
```

GeoInvProjectGeometry
GeoInvProjectGeometry() は、ジオメトリをエリアに集計し、投影の逆を適用します。

```
GeoInvProjectGeometry(type, field_name)
```

GeoProjectGeometry
GeoProjectGeometry() は、ジオメトリをエリアに集計し、投影を適用します。

```
GeoProjectGeometry(type, field_name)
```

GeoReduceGeometry
GeoReduceGeometry() は、ジオメトリの頂点数を削減し、多数のエリアを1つのエリアに集計します (各エリアの境界線は維持されます)。

```
GeoReduceGeometry (geometry)
```

次の関数は非集計関数です。

GeoGetBoundingBox
GeoGetBoundingBox() は、スクリプトとチャートの数式でジオメトリのすべての座標を含む最小境界ボックスを計算します。

```
GeoGetBoundingBox (geometry)
```

GeoGetPolygonCenter
GeoGetPolygonCenter() は、スクリプトとチャートの数式でジオメトリの中心点を計算して返します。

```
GeoGetPolygonCenter (geometry)
```

GeoMakePoint
GeoMakePoint() は、スクリプトとチャートの数式で緯度と経度を使ってポイントを作成してタグ付けします。

```
GeoMakePoint (lat_field_name, lon_field_name)
```

GeoProject
GeoProject() は、スクリプトとチャートの数式でジオメトリに投影を適用します。

```
GeoProject (type, field_name)
```

GeoAggrGeometry

GeoAggrGeometry() は、多くのサブリージョンを1つのリージョンにまとめるなど、多数のエリアを1つの大きなエリアに集計します。

構文:

```
GeoAggrGeometry(field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント(またはポイントセット)の場合があります。

一般的には、**GeoAggrGeometry()** は、地理空間の境界データを組み合わせるために使用できます。たとえば、郊外の郵便番号地域のデータと各地域の売り上げデータがあるとします。営業担当者の担当地域が、複数の郵便番号地域である場合は、個別の地域ごとではなく、担当地域ごとの総売り上げを表示し、結果を色分けしたマップで示したほうが便利です。

GeoAggrGeometry() は個別の郊外ジオメトリの集計を計算し、データモデルに結合した担当地域ジオメトリを生成します。営業担当地域が調整された場合は、データは新しく結合された境界をリロードし、売り上げがマップに反映されます。

GeoAggrGeometry() は集計関数のため、スクリプトで使用する場合は **Group by** 節を含む **LOAD** ステートメントが必要です。

> ***GeoAggrGeometry()** を使用し作成されたマップの境界線は、結合された地域の境界線です。集計前の地域の個別の境界線を表示する場合は、**GeoReduceGeometry()** を使用します。*

例:
この例は地域情報を含む KML ファイルをロードした後、集計された地域情報を含むテーブルをロードします。

```
[MapSource]: LOAD [world.Name], [world.Point], [world.Area] FROM [lib://Downloads/world.kml]
(kml, Table is [World.shp/Features]); Map: LOAD world.Name, GeoAggrGeometry(world.Area) as
[AggrArea] resident MapSource Group By world.Name;

Drop Table MapSource;
```

GeoBoundingBox

GeoBoundingBox() は、ジオメトリをエリアに集計し、すべての座標を含む最小境界ボックスを計算します。

GeoBoundingBox は 4 つの値 (左、右、上、下) のリストとして表されます。

構文:

```
GeoBoundingBox(field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。

GeoBoundingBox()は、ジオメトリのセットを集計し、ジオメトリを集計したすべての座標を含む最も小さい長方形の 4 つの座標を返します。

マップの結果を表示するには、4 つの座標の結果文字列をポリゴン形式に変換し、地理ポリゴン形式で変換した項目にタグ付けし、その項目をマップ オブジェクトにドラッグ アンド ドロップします。マップ ビジュアライゼーションに長方形のボックスが表示されます。

GeoCountVertex

GeoCountVertex() は、ポリゴン ジオメトリに含まれる頂点の数を特定します。

構文:

```
GeoCountVertex(field_name)
```

戻り値データ型: integer

引数:

引数

引数	説明
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。

GeoGetBoundingBox

GeoGetBoundingBox() は、スクリプトとチャートの数式でジオメトリのすべての座標を含む最小境界ボックスを計算します。

関数 GeoBoundingBox() で作成される地理空間境界ボックスは 4 つの値 (左、右、上、下) のリストとして表されます。

構文:

```
GeoGetBoundingBox(field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。

*ロード中にエラーが発生するため、データロードエディタで、この関数および集計されていない地理空間関数と**Group by**条件を使用しないでください。*

GeoGetPolygonCenter

GeoGetPolygonCenter() は、スクリプトとチャートの数式でジオメトリの中心点を計算して返します。

マップを色で塗りつぶすのではなく、点を描くことが要件の場合もあります。既存の地理空間データが地域ジオメトリの形式でのみ使用できる場合 (たとえば、境界) は、**GeoGetPolygonCenter()** を使用して、地域の中心の経度と緯度のペアを取得します。

構文:

```
GeoGetPolygonCenter(field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。

*ロード中にエラーが発生するため、データロードエディタで、この関数および集計されていない地理空間関数と**Group by**条件を使用しないでください。*

GeoInvProjectGeometry

GeoInvProjectGeometry() は、ジオメトリをエリアに集計し、投影の逆を適用します。

構文:

```
GeoInvProjectGeometry(type, field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
type	マップのジオメトリの変換に使用する投影のタイプ。次の 2 つの値のうちの 1 つを使用できます。1:1 投影となる 'ユニット' (デフォルト) または標準的なメルカトル図法を使用する 'メルカトル'。
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。

例:

スクリプトの例

例	結果
Load ステートメントで: GeoInvProjectGeometry ('mercator',AreaPolygon) as InvProjectGeometry	**AreaPolygon** としてロードされるジオメトリは、メルカトル図法の逆変換を使用して変換され、ビジュアライゼーションで使用する **InvProjectGeometry** として保存されます。

GeoMakePoint

GeoMakePoint() は、スクリプトとチャートの数式で緯度と経度を使ってポイントを作成してタグ付けします。GeoMakePoint は、経度、緯度の順でポイントを返します。

構文:

```
GeoMakePoint(lat_field_name, lon_field_name)
```

戻り値データ型: string、形式 [経度, 緯度]

引数:

引数

引数	説明
lat_field_name	ポイントの緯度を表す項目を参照する項目または式。
lon_field_name	ポイントの経度を表す項目を参照する項目または式。

*ロード中にエラーが発生するため、データロードエディタで、この関数および集計されていない地理空間関数と**Group by**条件を使用しないでください。*

GeoProject

GeoProject() は、スクリプトとチャートの数式でジオメトリに投影を適用します。

構文:

```
GeoProject(type, field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
type	マップのジオメトリの変換に使用する投影のタイプ。次の 2 つの値のうちの 1 つを使用できます。1:1 投影となる 'ユニット' (デフォルト) または Web メルカトル図法を使用する 'メルカトル'。
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。

*ロード中にエラーが発生するため、データロードエディタで、この関数および集計されていない地理空間関数と**Group by**条件を使用しないでください。*

例:

スクリプトの例

例	結果
Load ステートメントで: `GeoProject('mercator',Area) as GetProject`	メルカトル図法は、**Area** としてロードされるジオメトリに適用され、結果は、**GetProject** として保存されます.

GeoProjectGeometry

GeoProjectGeometry() は、ジオメトリをエリアに集計し、投影を適用します。

構文:

```
GeoProjectGeometry(type, field_name)
```

戻り値データ型: string

引数:

引数

引数	説明
type	マップのジオメトリの変換に使用する投影のタイプ。次の 2 つの値のうちの 1 つを使用できます。1:1 投影となる 'ユニット' (デフォルト) または Web メルカトル図法を使用する 'メルカトル'。
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。

例:

例	結果
Load ステートメントで: `GeoProjectGeometry ('mercator',AreaPolygon) as ProjectGeometry`	**AreaPolygon** としてロードされるジオメトリは、メルカトル図法を使用して変換され、ビジュアライゼーションで使用する **ProjectGeometry** として保存されます。

GeoReduceGeometry

GeoReduceGeometry() は、ジオメトリの頂点数を削減し、多数のエリアを 1 つのエリアに集計します (各エリアの境界線は維持されます)。

構文:

```
GeoReduceGeometry(field_name[, value])
```

戻り値データ型: string

引数:

引数

引数	説明
field_name	表現するジオメトリを含む項目を参照する項目または数式。これは、経度または緯度、エリアを示すいずれかのポイント (またはポイントセット) の場合があります。
value	ジオメトリに適用する削減量。範囲は 0 ~ 1 で、0 は削減なし、1 は頂点の最大削減量を意味します。 *複雑なデータセットで value を 0.9 以上に設定すると、視覚的表現が不正確になるレベルまで頂点の数を削減することができます。*

GeoReduceGeometry() は **GeoAggrGeometry()** と同様の関数を実行し、多数のエリアを1つのエリアに集計します。**GeoReduceGeometry()** を使用する場合、個別の境界線と集計前のデータの差がマップに表示されます。

GeoReduceGeometry() は集計関数のため、スクリプトで使用する場合は **Group by** 節を含む **LOAD** ステートメントが必要です。

例:
この例は、エリア情報を含む KML ファイルをロードした後、削減されたエリア情報および集計されたエリア情報を含むテーブルをロードします。

```
[MapSource]:
LOAD [world.Name],
       [world.Point],
       [world.Area]
 FROM [lib://Downloads/world.kml]
(kml, Table is [World.shp/Features]);

Map:
LOAD world.Name,
     GeoReduceGeometry(world.Area,0.5) as [ReducedArea]
resident MapSource Group By world.Name;

Drop Table MapSource;
```

8.15 変換関数

変換関数は、入力テキスト項目または数式のコンテンツを評価し、結果として返される数値に、指定されたデータ形式を適用します。これらの関数を使用して、小数点記号、3桁区切りの記号、および日付書式などの属性を含むデータ型に基づいて数値の書式を指定できます。

この変換関数は、すべて文字列と数値の両方を持つデュアル値を返しますが、文字列から数値への変換を実行するものとみなすことができます。この関数は、入力式のテキスト値を取得し、文字列を表す数値を生成します。

それに対して、書式設定関数は上記の正反対のことを行います。つまり数式を取得し、それを文字列として評価し、結果として返されるテキスト値の表示形式を指定します。

変換関数が使用されない場合、Qlik Sense は、スクリプトの変数とオペレーティング システムが定義する既定の数値、日付、時間の書式を使用して、データを数値、日付、時刻、タイムスタンプ、文字列の組み合わせとして解釈します。

すべての変換関数は、データロードスクリプトおよびチャート式の両方で使用できます。

すべての数値表現で小数点に小数点の記号を使用しています。

変換関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Date#

Date# は、2 番目の引数で指定されている書式で日付として、数式を評価します。形式コードが省略されている場合は、オペレーティング システムが使用している既定の日付形式が使用されます。

```
Date# (page 1243)(text[, format])
```

Interval#

Interval#() は、既定では、テキスト表現をオペレーティング システムの書式設定の時間間隔として評価します。ただし、2 番目の引数で書式が指定されている場合は、その書式の時間間隔として評価します。

```
Interval# (page 1244)(text[, format])
```

Money#

Money#()は、形式の文字列がある場合を除き、ロードスクリプトまたはオペレーティング システムで設定された形式で、テキスト文字列を金額値に変換します。カスタムの小数点および桁区切り記号は、オプションのパラメータです。

```
Money# (page 1244)(text[, format[, dec_sep[, thou_sep ] ] ])
```

Num#

Num#() は、テキスト文字列を数値として解釈します。つまり、2 番目のパラメータで指定された形式を使用して、入力文字列を数値に変換します。2 番目のパラメーターを省略すると、データロードスクリプトで設定された10 進数と1000 進数の区切り文字が使用されます。カスタムの小数点および桁区切り記号は、オプションのパラメータです。

```
Num# (page 1246)(text[ , format[, dec_sep[ , thou_sep]]])
```

Text

Text() は、数値として解釈できる場合でも、数式をテキストとして処理します。

```
Text(expr)
```

Time#

Time#() は、書式文字列が提供されている場合を除き、データロードスクリプトまたはオペレーティング システムの時刻書式設定で、数式を時刻値として評価します。.

```
Time# (page 1247)(text[, format])
```

Timestamp#

Timestamp#() は、書式文字列が提供されている場合を除き、データロードスクリプトまたはオペレーティング システムのタイムスタンプ書式設定で、数式を日付と時刻の値として評価します。

```
Timestamp# (page 1248)(text[, format])
```

参照先:

- *書式設定関数 (page 1207)*

Date#

Date# は、2 番目の引数で指定されている書式で日付として、数式を評価します。

構文:

```
Date#(text[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
text	評価対象のテキスト文字列。
format	評価されるテキスト文字列の形式を説明する文字列。省略した場合、データロードスクリプトまたはオペレーティング システムのシステム変数に設定された日付形式が使用されます。

例と結果:
次の例では、日付形式 **M/D/YYYY** を使用しています。日付形式は、データロードスクリプトの先頭にある **SET DateFormat** ステートメントで指定されます。

この例のスクリプトをアプリに追加し、実行します。

```
Load *,

Num(Date#(StringDate)) as Date;

LOAD * INLINE [

StringDate

8/7/97

8/6/1997

]
```

軸として **StringDate** および **Date** を使用してテーブルを作成すると、結果は次のようになります。

結果

StringDate	Date
8/7/97	35649
8/6/1997	35648

Interval#

Interval#() は、既定では、テキスト表現をオペレーティング システムの書式設定の時間間隔として評価します。ただし、2 番目の引数で書式が指定されている場合は、その書式の時間間隔として評価します。

構文:

```
Interval#(text[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
text	評価対象のテキスト文字列。
format	文字列を数字間隔に変換する際にしようが予想される入力形式を説明する文字列。 省略されている場合は、オペレーティング システムで設定された短い日付書式、時間書式、および小数点記号が使用されます。

interval#関数は、テキストの時間間隔を数字に変換します。

例と結果:
この例では、次のオペレーティング システムの設定を前提としています。

- 短い日付書式: YY-MM-DD
- 時間書式: M/D/YY
- 小数点記号:

結果

例	結果
`Interval#( A, 'D hh:mm' )` ここで A='1 09:00'	1.375

Money#

Money#()は、形式の文字列がある場合を除き、ロード スクリプトまたはオペレーティング システムで設定された形式で、テキスト文字列を金額値に変換します。カスタムの小数点および桁区切り記号は、オプションのパラメータです。

構文:

```
Money#(text[, format[, dec_sep [, thou_sep ] ] ])
```

戻り値データ型： dual

引数：

引数

引数	説明
text	評価対象のテキスト文字列。
format	文字列を数字間隔に変換する際にしようが予想される入力形式を説明する文字列。 省略されている場合、オペレーティング システムで設定された金額形式が使用されます。
dec_sep	小数点記号を指定する文字列。省略されている場合は、データロード スクリプトで設定された MoneyDecimalSep 値が使用されます。
thou_sep	3 桁区切りの記号を指定する文字列。省略されている場合は、データロード スクリプトで設定された MoneyThousandSep 値が使用されます。

money# 関数は、原則として **num#** 関数と同じように機能しますが、小数点および桁区切り記号のデフォルト値をスクリプトの通貨書式の変数または通貨のシステム設定から決定します。

例と結果:
この例では、次の 2 つのオペレーティング システムの設定を前提としています。

- 通貨書式のデフォルト設定 1: kr # ##0,00
- 通貨書式のデフォルト設定 2: $ #,##0.00

```
Money#(A , '# ##0,00 kr' )
```
ここで A=35 648,37 kr

結果

結果	設定 1	設定 2
文字列	35 648.37 kr	35 648.37 kr
数値	35648.37	3564837

```
Money#( A, ' $#', '.', ',' )
```
ここで A= $35,648.37

結果

結果	設定 1	設定 2
文字列	$35,648.37	$35,648.37
数値	35648.37	35648.37

Num#

Num#() は、テキスト文字列を数値として解釈します。つまり、2 番目のパラメータで指定された形式を使用して、入力文字列を数値に変換します。2 番目のパラメーターを省略すると、データロードスクリプトで設定された10 進数と1000 進数の区切り文字が使用されます。カスタムの小数点および桁区切り記号は、オプションのパラメータです。

構文:

```
Num#(text[, format[, dec_sep [, thou_sep ] ] ])
```

戻り値データ型: dual

Num#() 関数は、文字列と数値の両方が指定されたデュアル値を返します。この関数は、入力式のテキスト表現を取得し、数値を生成します。数値の形式は変更されません。出力は入力と同じ方法で形式化されます。

引数:

引数

引数	説明
text	評価対象のテキスト文字列。
format	最初のパラメータで使用される数値形式を指定する文字列。省略した場合、データロードスクリプトで設定されている10 進数と1000 進数の区切り文字が使用されます。
dec_sep	小数点記号を指定する文字列。省略した場合、データロードスクリプトで設定された変数 DecimalSep 値が使用されます。
thou_sep	3 桁区切りの記号を指定する文字列。省略されている場合は、データロードスクリプトで設定された変数 ThousandSep 値が使用されます。

例と結果:
次のテーブルは、A のさまざまな値に対する *Num#(A, '#', '.', ',')* の結果を示しています。

結果

ライン番号の隣にある	文字列表現	数値 (ここでは小数で表示されます)
35,648.31	35,648.31	35648.31
35 648.312	35 648.312	35648.312
35.648,3123	35.648,3123	-
35 648,31234	35 648,31234	-

Text

Text() は、数値として解釈できる場合でも、数式をテキストとして処理します。

構文:
```
Text (expr)
```

戻り値データ型: dual

```
Text( A )
```
ここで A=1234

結果

文字列	数値
1234	-

```
Text( pi( ) )
```

結果

文字列	数値
3.1415926535898	-

Time#

Time#() は、書式文字列が提供されている場合を除き、データロードスクリプトまたはオペレーティング システムの時刻書式設定で、数式を時刻値として評価します。.

構文:
```
time#(text[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
text	評価対象のテキスト文字列。
format	評価されるテキスト文字列の形式を説明する文字列。省略されている場合は、オペレーティング システムで設定された短い日付書式、時間書式、および小数点記号が使用されます。

- 時間書式のデフォルト設定 1: hh:mm:ss
- 時間書式のデフォルト設定 2: hh.mm.ss

```
time#( A )
```
A=09:00:00

結果

結果	設定 1	設定 2
文字列:	09:00:00	09:00:00
数値:	0.375	-

- 時間書式のデフォルト設定 1: hh:mm:ss
- 時間書式のデフォルト設定 2: hh.mm.ss

```
time#( A, 'hh.mm' )
```
A=09.00

結果

結果	設定 1	設定 2
文字列:	09.00	09.00
数値:	0.375	0.375

Timestamp#

Timestamp#() は、書式文字列が提供されている場合を除き、データロードスクリプトまたはオペレーティング システムのタイムスタンプ書式設定で、数式を日付と時刻の値として評価します。

構文:

```
timestamp#(text[, format])
```

戻り値データ型: dual

引数:

引数

引数	説明
text	評価対象のテキスト文字列。
format	評価されるテキスト文字列の形式を説明する文字列。省略されている場合は、オペレーティング システムで設定された短い日付書式、時間書式、および小数点記号が使用されます。ISO 8601 は、タイムスタンプでサポートされています。

次の例では、日付形式 **M/D/YYYY** を使用しています。日付形式は、データロードスクリプトの先頭にある **SET DateFormat** ステートメントで指定されます

この例のスクリプトをアプリに追加し、実行します。

```
Load *,
Timestamp(Timestamp#(String)) as TS;
LOAD * INLINE [
String
2015-09-15T12:13:14
1952-10-16T13:14:00+0200
1109-03-01T14:15
];
```

String および **TS** を軸として使用してテーブルを作成する場合、結果は次のようになります。

結果

文字列	TS
2015-09-15T12:13:14	9/15/2015 12:13:14 PM
1952-10-16T13:14:00+0200	10/16/1952 11:14:00 AM
1109-03-01T14:15	3/1/1109 2:15:00 PM

8.16 レコード間関数

レコード間関数は、次のスクリプトと数式に使用します。

- データロードスクリプト (現在のレコードの評価に以前にロードされたデータのレコード値が必要な場合)
- チャート式 (ビジュアライゼーションのデータセットからもう1 つ値が必要な場合)

チャートの式いずれかにレコード間のチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでレコード間のチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。この制限は、同等のスクリプト関数 (ある場合) には当てはまりません。

信頼性のある自己参照型数式の定義は、行数が 100 未満のテーブルでのみ作成可能ですが、これは Qlik エンジンが実行されているハードウェアによって変わる可能性があります。

行関数

これらの関数は、チャート式でのみ使用できます。

Above

Above() テーブルの列セグメント内の現在の行の上にある行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは真上の行です。テーブル以外のチャートでは、**Above()** は、チャートのストレートテーブルに相当するセグメントの現在の行よりも上にある行を評価します。

Above - チャート関数 ([**TOTAL** [<fld{,fld}>]] expr [, offset [,count]]**)**

Below

Below() テーブルの列セグメント内の現在の行の下にある行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは真下の行です。テーブル以外のチャートでは、**Below()** は、チャートのストレートテーブルに相当するセグメントの現在の行よりも下にある行を評価します。

```
Below - チャート関数([TOTAL[<fld{,fld}>]] expression [ , offset [,count ]])
```

Bottom

Bottom() テーブルの列セグメント内の最後 (最下部) の行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは最下部の行です。テーブル以外のチャートでは、評価はチャートのストレートテーブルに相当する現在の列の最後の行を評価します。

```
Bottom - チャート関数([TOTAL[<fld{,fld}>]] expr [ , offset [,count ]])
```

Top

Top() テーブルの列セグメント内の最初 (最上部) の行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは最上部の行です。テーブル以外のチャートでは、**Top()** 評価はチャートのストレートテーブルに相当する現在の列の最初の行を評価します。

```
Top - チャート関数([TOTAL [<fld{,fld}>]] expr [ , offset [,count ]])
```

NoOfRows

NoOfRows() は、テーブルの現在の列セグメント内の行の数を返します。ビットマップチャートの場合、**NoOfRows()** はチャートのストレートテーブルに相当するセグメントに含まれる行の数を返します。

```
NoOfRows - チャート関数([TOTAL])
```

列関数

これらの関数は、チャート式でのみ使用できます。

Column

Column() は、軸に関係なく、ストレートテーブルで **ColumnNo** に対応する列の値を返します。例えば、**Column(2)** は 2 番目のメジャー列の値を返します。

```
Column - チャート関数(ColumnNo)
```

Dimensionality

Dimensionality() 現在の行の軸の数を返します。ピボットテーブルの場合、この関数は、集計以外の内容 (部分合計または折りたたまれた集計を含まない) を含む軸列の合計数を返します。

```
Dimensionality - チャート関数 ( )
```

Secondarydimensionality

SecondaryDimensionality() は、集計以外の内容 (部分合計または折りたたまれた集計を含まない) を含む軸のピボットテーブル行の数を返します。この関数は、水平ピボットテーブル軸の **dimensionality()** 関数に相当します。

```
SecondaryDimensionality- チャート関数 ( )
```

項目関数

FieldIndex

FieldIndex() は、**field_name** 項目内の **value** 項目値の位置を返します (ロード順)。

```
FieldIndex (field_name , value)
```

FieldValue

FieldValue() は、**field_name** 項目の **elem_no** の位置にある値を返します (ロード順)。

```
FieldValue (field_name , elem_no)
```

FieldValueCount

FieldValueCount() は **integer** 関数で、項目に含まれる固有値の数を返します。

```
FieldValueCount (field_name)
```

ピボットテーブル関数

これらの関数は、チャート式でのみ使用できます。

After

After() は、ピボットテーブルの行セグメント内の現在列の後の列に、ピボットテーブルの軸値で評価された expression の値を返します。

```
After - チャート関数([TOTAL] expression [ , offset [,n]])
```

Before

Before() は、ピボットテーブルの行セグメント内の現在列の前の列に、ピボットテーブルの軸値で評価された expression の値を返します。

```
Before - チャート関数([TOTAL] expression [ , offset [,n]])
```

First

First() は、ピボットテーブルの現在の行セグメントの最初の列に、ピボットテーブルの軸値で評価された expression の値を返します。ピボットテーブル以外のすべてのチャートタイプの場合、この関数は NULL を返します。

```
First - チャート関数([TOTAL] expression [ , offset [,n]])
```

Last

Last() は、ピボットテーブルの現在の行セグメントの最後の列に、ピボットテーブルの軸値で評価された expression の値を返します。ピボットテーブル以外のすべてのチャートタイプの場合、この関数は NULL を返します。

```
Last - チャート関数([TOTAL] expression [ , offset [,n]])
```

ColumnNo

ColumnNo() は、ピボットテーブルの現在の行セグメント内の現在列の番号を返します。最初の列の番号は 1 です。

```
ColumnNo - チャート関数([TOTAL])
```

NoOfColumns
NoOfColumns() は、ピボット テーブルの現在の行 セグメント内の列の数を返します。

```
NoOfColumns - チャート関数([TOTAL])
```

データ ロード スクリプトのレコード間関数

Exists

Exists() は、特定の項目値がデータ ロード スクリプトの項目にすでにロードされているかどうかを決定します。この関数は TRUE または FALSE を返すため、**LOAD** ステートメントまたは **IF** ステートメントの **where** 句で使用できます。

```
Exists (field_name [, expr])
```

LookUp

Lookup() は、すでにロードされているテーブルを参照し、項目 **match_field_name** における値 **match_field_value** の最初の出現に対応する **field_name** の値を返します。テーブルは、現在のテーブルまたは前にロードして別のテーブルにすることができます。

```
LookUp (field_name, match_field_name, match_field_value [, table_name])
```

Peek

Peek() は、すでにロードされている行に対してテーブルの項目値を返します。行番号は、テーブルと同様に指定できます。行番号が指定されていない場合は、最後にロードされたレコードが使用されます。

```
Peek (field_name[, row_no[, table_name ] ])
```

Previous

Previous()は、**where**節のために破棄されなかった以前の入力レコードのデータを使用して、**expr**数式の値を算出します。内部テーブルの最初のレコードの場合は、NULL を返します。

```
Previous (page 1288)(expr)
```

参照先:

- *範囲関数 (page 1307)*

Above - チャート関数

Above() テーブルの列セグメント内の現在の行の上にある行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは真上の行です。テーブル以外のチャートでは、**Above()** は、チャートのストレート テーブルに相当するセグメントの現在の行よりも上にある行を評価します。

構文:

```
Above([TOTAL] expr [ , offset [,count]])
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
offset	offsetn を 0 より大きい値にすると、数式の評価が現在の行から n 行上に移動します。 offset に 0 を指定すると、現在の行で数式が評価されます。 offset が負の値である場合、Above 関数は、相応する正の offset 値が付いた Below 関数と同様に機能します。
count	3 番目の引数 **count** を 1 より大きい値にすると、この関数は **count** 値の範囲を返します。つまり、オリジナルのテーブルのセルから上方向に **count** 行の各値を返します。 この形式では、特別な範囲関数の引数として関数を使用できます。 *範囲関数 (page 1307)*
TOTAL	テーブルが 1 軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

列セグメントの最初の行では、上に行がないため、NULL 値が返されます。

列セグメントは、現在のソート順で軸に同じ値を持つ連続したセルのサブセットとして定義されます。レコード間チャート関数は、チャートのストレートテーブルに相当する右端の軸を除外して列セグメントで実行されます。チャートに軸が 1 つしかない場合、または TOTAL 修飾子が指定されていると、数式はテーブル全体を評価します。

テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除くすべての軸列の現在行と同じ値を持つ行だけが含まれます。

制限事項:

- 再帰呼び出しは NULL を返します。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

例と結果:

Example 1:

例 1 のテーブルのビジュアライゼーション

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

この例で示されているテーブルのスクリーンショットでは、軸 **Customer** とメジャー `Sum(Sales)` および `Above(Sum(Sales))` からテーブルのビジュアライゼーションが作成されています。

行 **Customer** の上には行がないため、列 Above(Sum(Sales)) は、**Astrida** を含む Customer に対して NULL を返します。たとえば、行 **Betacab** の結果は **Astrida** の Sum(Sales) の値を示し、**Canutility** の結果は **Betacab** などの **Sum(Sales)** の値を示します。

Sum(Sales)+Above(Sum(Sales)) というラベルの列では、**Betacab** 行の結果には、行 **Betacab** と**Astrida** 値の加算結果 (539+587) が **Sum(Sales)** に表示されます。**Canutility** 行の結果には、行 **Canutility** と **Betacab** 値の加算結果 (683+539) が **Sum(Sales)** に表示されます。

数式 `Sum(Sales)+Above(Sum(Sales), 3)` を使用して作成された Above offset 3 というラベルのメジャーは、引数 **offset** が 3 に設定されており、現在の行より3 行上の値を取得します。また、現在の **Customer** の **Sum(Sales)** 値が 3 行上の **Customer** の値に追加されます。**Customer** の最初の 3 行で返される値は Null です。

テーブルには、`Sum(Sales)+Above(Sum(Sales))` から作られたものと、**Higher?** というラベルの `IF(Sum(Sales)>Above(Sum(Sales)), 'Higher')` から作成されたより複雑なメジャーも表示されます。

この関数は、棒グラフなどテーブル以外のチャートでも使用できます。

その他のチャートの場合、どの行が関数に関連しているかわかりやすくするために、チャートをストレートテーブルに相当するセグメントに変換します。

Example 2:

この例で示されているテーブルのスクリーンショットでは、より多くの軸がビジュアライゼーションに追加されています。(**Month** および **Product**) が保存されます。複数の軸が含まれているチャートでは、**Above**、**Below**、**Top**、**Bottom** 関数を含む数式の結果は、Qlik Sense における列軸のソート順序によって変わります。Qlik Sense は、最後にソートされた軸の結果である列セグメントに基づいて関数を評価します。列のソート順は、[**ソート**] のプロパティパネルで制御され、必ずしも列がテーブルに表示される順序ではありません。

例 2 のテーブルのビジュアライゼーションを示した以下のスクリーンショットでは、最後にソートされた軸が **Month** のため、**Above** 関数は月に基づいて評価を行います。列セグメントにある各月 (**Jan**から**Aug**) の各 **Product** の値について一連の結果が出されます。この後に次の列セグメントのシリーズ、つまり次の **Product** の各 **Month** の値が続きます。各 **Product** のそれぞれの **Customer** 値に列セグメントが生成されます。

例 2 のテーブルのビジュアライゼーション

Customer	Product	Month	Sum([Sales])	Above(Sum(Sales))
			2566	-
Astrida	AA	Jan	46	-
Astrida	AA	Feb	60	46
Astrida	AA	Mar	70	60
Astrida	AA	Apr	13	70
Astrida	AA	May	78	13
Astrida	AA	Jun	20	78
Astrida	AA	Jul	45	20
Astrida	AA	Aug	65	45

Example 3:

例 3 のテーブルのビジュアライゼーションを示したスクリーンショットでは、最後にソートされた軸が **Product** になっています。これは、プロパティパネルにあるソートタブで軸 Product を位置 3 に移動することで達成できます。**Above** 関数は各 **Product** について評価されます。製品は **AA** と**BB** の 2 つしかないため、Null 以外の結果は各シリーズにつき 1 つのみとなります。月が **Jan** の行 **BB** では、**Above(Sum(Sales))** の値は 46 です。行 **AA** では、値は Null です。**AA** の上には **Product** の値がないため、任意の月の行 AA の値は常に Null になります。2 番目のシリーズは、**Customer** 値、**Astrida** について、月 **Feb** の **AA** と**BB** で評価されます。**Astrida** ですべての月を評価したら、2 番目の **Customer**、Betacab などでもこのステップを繰り返します。

例 3 のテーブルのビジュアライゼーション

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

例 4:

Example 4:	**結果**
Above 関数は、範囲関数への入力として使用できます。例: `RangeAvg (Above(Sum (Sales),1,3))`。	Above() 関数の引数では、offset は 1 に設定され、count は 3 に設定されています。この関数は、列セグメントの現在行のすぐ上の 3 行 (行がある場合) で数式 Sum (Sales) の結果を算出します。この 3 つの値は、RangeAvg() 関数への入力として使用され、指定された数値の範囲で平均値を算出します。 軸として Customer を有するテーブルによって、RangeAvg() の数式について次の結果が得られます。 Astrida - Betacab 587 Canutility 563 Divadip: 603

例で使用されているデータ:

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

参照先:

Below - チャート関数 (page 1257)

Bottom - チャート関数 (page 1260)
Top - チャート関数 (page 1289)
RangeAvg (page 1310)

Below - チャート関数

Below() テーブルの列セグメント内の現在の行の下にある行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは真下の行です。テーブル以外のチャートでは、**Below()** は、チャートのストレートテーブルに相当するセグメントの現在の行よりも下にある行を評価します。

構文:

```
Below([TOTAL] expr [ , offset [,count ]])
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
offset	**offset**n を 1 より大きい値にすると、数式の評価が現在の行から n 行下に移動します。 offset に 0 を指定すると、現在の行で数式が評価されます。 offset が負の値である場合、**Below** 関数は、相応する正の offset 値が付いた **Above** 関数と同様に機能します。
count	3 番目のパラメータ **count** を 1 より大きい値にした場合、この関数は **count** 値の範囲を返します。つまり、オリジナルのテーブルのセルから下方向に **count** 行の各値を返します。この形式では、特別な範囲関数の引数として関数を使用できます。 *範囲関数 (page 1307)*
TOTAL	テーブルが 1 軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

列セグメントの最後の行では下に行が存在しないため、NULL 値が返されます。

列セグメントは、現在のソート順で軸に同じ値を持つ連続したセルのサブセットとして定義されます。レコード間チャート関数は、チャートのストレートテーブルに相当する右端の軸を除外して列セグメントで実行されます。チャートに軸が 1 つしかない場合、または TOTAL 修飾子が指定されていると、数式はテーブル全体を評価します。

テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除くすべての軸列の現在行と同じ値を持つ行だけが含まれます。

制限事項:

- 再帰呼び出しは NULL を返します。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

例と結果:

Example 1:

例 1 のテーブルのビジュアライゼーション

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	-	-	-	-

例 1 のスクリーンショットに示されているテーブルでは、軸 **Customer** とメジャー `Sum(Sales)` および `Below(Sum(Sales))` からテーブルのビジュアライゼーションが作成されています。

Divadip を含む **Customer** 行の下には行がないため、列 **Below(Sum(Sales))** はこの行に対して NULL を返します。たとえば、行 **Canutility** の結果は **Divadip** の Sum(Sales) の値を示し、**Betacab** の結果は **Canutility** などの **Sum(Sales)** の値を示します。

テーブルには、より複雑なメジャーも含まれます (列 `Sum(Sales)+Below(Sum(Sales))`, **Below +Offset 3** および **Higher?** というラベルのコラムを参照) これらの数式は、次の説明のとおり機能します。

Sum(Sales)+Below(Sum(Sales)) というラベルの列では、**Astrida** 行の結果には、行 **Betacab** と **Astrida** 値の加算結果 (539+587) が **Sum(Sales)** に表示されます。**Betacab** 行の結果には、行 **Canutility** と **Betacab** 値の加算結果 (539+683) が **Sum(Sales)** に表示されます。

数式 `Sum(Sales)+Below(Sum(Sales), 3)` を使用して作成された **Below +Offset 3** というラベルのメジャーは、引数 **offset** が 3 に設定されており、現在の行より 3 つ下の行の値を取得します。現在の **Customer** の **Sum (Sales)** 値を 3 行下の **Customer** の値に加えます。**Customer** の最後の 3 行の値は Null です。

Higher? というラベルのメジャーは、`IF(Sum(Sales)>Below(Sum(Sales)), 'Higher')` という数式から作成されています。これは、メジャー **Sum(Sales)** の現在の行の値をその下の行と比較します。現在の行の値が大きい場合、「Higher」が出力されます。

この関数は、棒グラフなどテーブル以外のチャートでも使用できます。

その他のチャートの場合、どの行が関数に関連しているかわかりやすくするために、チャートをストレートテーブルに相当するセグメントに変換します。

複数の軸が含まれているチャートでは、**Above**、**Below**、**Top**、**Bottom** 関数を含む数式の結果は、Qlik Sense における列軸のソート順序によって変わります。Qlik Sense は、最後にソートされた軸の結果である列セグメントに基づいて関数を評価します。列のソート順は、[ソート] のプロパティパネルで制御され、必ずしも列がテーブルに表示される順序ではありません。詳細については、**Above** 関数の例 2 を参照してください。

例 2

Example 2:	**結果**
Below 関数は、範囲関数への入力として使用できます。例: `RangeAvg (Below(Sum(Sales),1,3))`。	**Below()** 関数の引数では、offset は 1 に設定され、count は 3 に設定されています。この関数は、列セグメントの現在行のすぐ下の 3 行 (行がある場合) で数式 **Sum(Sales)** の結果を算出します。この 3 つの値は、RangeAvg() 関数への入力として使用され、指定された数値の範囲で平均値を算出します。 軸として **Customer** を有するテーブルによって、RangeAvg() の数式について次の結果が得られます。
	Astrida 659.67 Betacab 720 Canutility 757 Divadip: -

例で使用されているデータ:

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
```

```
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

参照先:

- *Above - チャート関数 (page 1252)*
- *Bottom - チャート関数 (page 1260)*
- *Top - チャート関数 (page 1289)*
- *RangeAvg (page 1310)*

Bottom - チャート関数

Bottom() テーブルの列セグメント内の最後 (最下部) の行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは最下部の行です。テーブル以外のチャートでは、評価はチャートのストレートテーブルに相当する現在の列の最後の行を評価します。

構文:

Bottom([**TOTAL**] expr [, offset [,count]]**)**

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
offset	**offset**n を 1 より大きい値にすると、数式の評価が最終行から n 行上に移動します。 offset が負の値である場合、**Bottom** 関数は、相応する正の offset 値が付いた **Top** 関数と同様に機能します。
count	3 番目のパラメータである **count** を 1 より大きい値に設定した場合、この関数は、1 つの値ではなく、**count** 値の範囲を返します。つまり、現在の列セグメントの最後の **count** 行の各値を返します。この形式では、特別な範囲関数の引数として関数を使用できます。 *範囲関数 (page 1307)*
TOTAL	テーブルが 1 軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

列セグメントは、現在のソート順で軸に同じ値を持つ連続したセルのサブセットとして定義されます。レコード間チャート関数は、チャートのストレートテーブルに相当する右端の軸を除外して列セグメントで実行されます。チャートに軸が 1 つしかない場合、または TOTAL 修飾子が指定されていると、数式はテーブル全体を評価します。

テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除くすべての軸列の現在行と同じ値を持つ行だけが含まれます。

制限事項:

- 再帰呼び出しは NULL を返します。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

例と結果:

例 1 のテーブルのビジュアライゼーション

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

この例で示されているテーブルのスクリーンショットでは、軸 **Customer** とメジャー `Sum(Sales)` および `Bottom(Sum(Sales))` からテーブルのビジュアライゼーションが作成されています。

列 **Bottom(Sum(Sales))**はすべての行で 757 を返します (最終行**Divadip**の値)。

テーブルには、`Sum(Sales)+Bottom(Sum(Sales))`から作成されたものと、数式`Sum(Sales)+Bottom(Sum(Sales), 3)`を使用して作成され、引数**offset**が3に設定されている**Bottom offset 3**という、より複雑なメジャーも表示されています。最後から3つ上にある行の値に現在の行の **Sum(Sales)** 値を加えます (現在の行 + **Betacab** の値)。

2

この例で示されているテーブルのスクリーンショットでは、より多くの軸がビジュアライゼーションに追加されています。および **Month**、**Product** の 2 つのメジャーがあります。複数の軸が含まれているチャートでは、**Above**、**Below**、**Top**、**Bottom** 関数を含む数式の結果は、Qlik Sense における列軸のソート順序によって変わります。Qlik Sense は、最後にソートされた軸の結果である列セグメントに基づいて関数を評価します。列のソート順は、[**ソート**] のプロパティパネルで制御され、必ずしも列がテーブルに表示される順序ではありません。

最初のテーブルで数式は **Month**、2 番目のテーブルでは **Product** に基づいて評価されます。メジャー **End value** には数式 `Bottom(Sum(Sales))` が含まれています。**Month** の最終行は Dec で、Dec の **Product** の値は両方とも 22 になっています (スクリーンショット参照)。(スペース上の理由から、一部削除されている行があります)。

例 2 の最初のテーブル。End value メジャーの Bottom の値は Month (Dec) に基づいています。

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

例 2 の 2 番目のテーブル。End value メジャーの Bottom の値は、Product (Astrida の BB) に基いています。

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

詳細については、**Above** 関数の例 2 を参照してください。

例 3

3	結果
Bottom 関数は、範囲関数への入力として使用できます。例: `RangeAvg (Bottom(Sum(Sales),1,3))`。	**Bottom()** 関数の引数では、offset は 1 に設定され、count は 3 に設定されています。この関数は、列セグメントの最終行の上の行から始まる 3 行 (offset=1 のため)、およびその上の 2 行 (行がある場合) で数式 **Sum(Sales)** の結果を算出します。この 3 つの値は、RangeAvg() 関数への入力として使用され、指定された数値の範囲で平均値を算出します。 軸として **Customer** を有するテーブルによって、RangeAvg() の数式について次の結果が得られます。
	Astrida 659.67 Betacab 659.67 Canutility 659.67 Divadip: 659.67

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

参照先:

- *Top - チャート関数 (page 1289)*

Column - チャート関数

Column() は、軸に関係なく、ストレートテーブルで **ColumnNo** に対応する列の値を返します。例えば、**Column(2)** は 2 番目のメジャー列の値を返します。

構文:

```
Column(ColumnNo)
```

戻り値データ型: dual

引数:

引数

引数	説明
ColumnNo	メジャーを含むテーブルの列番号です。 *Column() 関数は軸列を無視します。*

制限事項:

- 再帰呼び出しは NULL を返します。
- **ColumnNo** が参照する列にメジャーがない場合は、NULL 値が返されます。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

例と結果:

総売上高の割合

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

Customer	Product	UnitPrice	UnitSales	Order Value	Total Sales Value	% Sales
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

選択した顧客の売上高率

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

例と結果

例	結果
Order Value は、数式 `Sum (UnitPrice*UnitSales)` を伴うメジャーとしてテーブルに追加されます。 Total Sales Value は、次の数式を伴うメジャーとして追加されます。`Sum(TOTAL UnitPrice*UnitSales)` % Sales は、以下の数式を伴うメジャーとして追加されます。`100*Column(1)/Column(2)`	Column(1) の結果は、最初のメジャー列である Order Value から得られます。 Column(2) の結果は、2 番目のメジャー列である Total Sales Value から得られます。 例 *総売上高の割合 (page 1264)* の % Sales 列の結果を参照してください。
Customer A を選択します。	選択が Total Sales Value に変わるため、%Sales になります。*選択した顧客の売上高率 (page 1265)* の例を参照してください。

例で使用されているデータ:

```
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 - チャート関数

Dimensionality() 現在の行の軸の数を返します。ピボットテーブルの場合、この関数は、集計以外の内容 (部分合計または折りたたまれた集計を含まない) を含む軸列の合計数を返します。

構文:

```
Dimensionality ( )
```

戻り値データ型: 整数

制限事項:

この関数は、チャートでのみ使用できます。ピボットテーブル以外のすべてのチャートタイプの場合は、合計行を除くすべての行の軸の数を返します。合計行の場合は 0 を返します。

チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

例: Dimensionality を使用したチャートの数式

例: チャートの数式

Dimensionality() 関数は、集計されていないデータがある行で軸の数に応じて異なるセルの書式設定を適用したい場所でチャートの数式としてピボットテーブルと共に使用できます。この例では、Dimensionality() 関数を使用して、指定された条件に一致するテーブルセルに背景色を適用しています。

ロードスクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
ProductSales:

Load * inline [
Country,Product,Sales,Budget
Sweden,AA,100000,50000
Germany,AA,125000,175000
Canada,AA,105000,98000
Norway,AA,74850,68500
Ireland,AA,49000,48000
Sweden,BB,98000,99000
Germany,BB,115000,175000
Norway,BB,71850,68500
Ireland,BB,31000,48000
] (delimiter is ',');
```

チャートの数式

国 と**製品** を軸として使用した Qlik Sense シートでピボットテーブルのビジュアライゼーションを作成します。**Sum (Sales)**、**Sum(Budget)**、**Dimensionality()** をメジャーとして追加します。

プロパティ パネルで、以下の式を **Sum(Sales)** メジャーの **背景色数式** に入力します。

```
If(Dimensionality()=1 and Sum(Sales)<Sum(Budget),RGB(255,156,156),
If(Dimensionality()=2 and Sum(Sales)<Sum(Budget),RGB(178,29,29)
))
```

結果:

Country / Product

	Values		
	Sum(Sales)	Sum([Budget])	Dimensionality()
Canada	105000	98000	1
AA	105000	98000	2
Germany	240000	350000	1
Ireland	80000	96000	1
AA	49000	48000	2
BB	31000	48000	2
Norway	146700	137000	1
AA	74850	68500	2
BB	71850	68500	2
Sweden	198000	149000	1

説明

式には、`If(Dimensionality()=1 and Sum(Sales)<Sum(Budget),RGB(255,156,156), If(Dimensionality()=2 and Sum(Sales)<Sum(Budget),RGB(178,29,29)))` dimensionality 値と各製品の Sum(Sales) と Sum (Budget) をチェックする条件付きステートメントが含まれています。条件が満たされると、背景色が Sum (Sales) 値に適用されます。

Exists

Exists() は、特定の項目値がデータロードスクリプトの項目にすでにロードされているかどうかを決定します。この関数は TRUE または FALSE を返すため、**LOAD** ステートメントまたは **IF** ステートメントの **where** 句で使用できます。

> ***Not Exists()*** *を使用して、項目値がロード済みかどうかを判断することもできますが、* ***Not Exists ()*** *を where 節で使用する場合は、用心するようお勧めします。* ***Exists()*** *関数は、以前にロードされたテーブルと、現在のテーブルに以前にロードされた値の両方をテストします。そのため、最初の出現のみがロードされます。2 番目の出現に遭遇したときには、値はすでにロード済みです。詳細については、例を参照してください。*

構文:

```
Exists(field_name [, expr])
```

戻り値データ型: ブール値

引数:

引数

引数	説明
field_name	値を検索する項目の名前。明示的な項目名を引用符なしで使用できます。 この項目は、スクリプトによってロード済みでなければなりません。つまり、スクリプト内の節にロードされた項目を参照することはできません。
expr	存在するかどうかを確認したい値。明示的な値または数式を使用して、現在の Load ステートメント内の 1 つまたは複数の項目を参照できます。 *現在の Load ステートメントに含まれていない項目を参照することはできません。* この引数は省略可能です。省略した場合、この関数で、現在のレコード内に **field_name** の値がすでに存在しているかどうかが確認されます。

例と結果:

例 1

```
Exists (Employee)
```

現在のレコード内の項目 **Employee** の値が、その項目を含む読み取り済みのレコード内にすでに存在してれば、-1 (True) を返します。

ステートメント `Exists (Employee, Employee)` と `Exists (Employee)` は同じ働きをします。

例 2

```
Exists(Employee, 'Bill')
```

項目値 **'Bill'** が項目 **Employee** の現在のコンテンツに含まれていれば、-1 (True) を返します。

例 3

```
Employees:
LOAD * inline [
Employee|ID|Salary
Bill|001|20000
John|002|30000
Steve|003|35000
] (delimiter is '|');
```

```
Citizens:
Load * inline [
Employee|Address
Bill|New York
Mary|London
Steve|Chicago
Lucy|Madrid
Lucy|Paris
John|Miami
] (delimiter is '|') where Exists (Employee);

Drop Tables Employees;
```

この結果は、Employee とAddress の軸を使用してテーブル ビジュアライゼーションで使用できるテーブルとなります。

where 節 `where Exists (Employee)` は、テーブル Citizens からEmployees にも存在するName のみを新しいテーブルにロードすることを意味します。Drop ステートメントは、混同を避けるためにテーブル Employees を削除します。

結果

Employee	Address
Bill	New York
John	Miami
Steve	Chicago

例 4:

```
Employees:
Load * inline [
Employee|ID|Salary
Bill|001|20000
John|002|30000
Steve|003|35000
] (delimiter is '|');

Citizens:
Load * inline [
Employee|Address
Bill|New York
Mary|London
Steve|Chicago
Lucy|Madrid
Lucy|Paris
John|Miami
] (delimiter is '|') where not Exists (Employee);
```

```
Drop Tables Employees;
```

where 節に not が含まれています: `where not Exists (Employee)`

つまり、テーブル Citizens から Employees に存在しない名前のみが新しいテーブルにロードされます。

Citizens テーブルには Lucy の値が 2 つありますが、結果テーブルに含まれているのは 1 つだけであることに注目してください。値 Lucy を使用して最初の行をロードすると、その行は Employee 項目に含められます。このため、2 行目を確認すると、その値は既に存在しています。

結果

Employee	住所
Mary	London
Lucy	Madrid

例 5

この例は、すべての値をロードする方法を示しています。

```
Employees:
Load Employee As Name;
LOAD * inline [
Employee|ID|Salary
Bill|001|20000
John|002|30000
Steve|003|35000
] (delimiter is '|');

Citizens:
Load * inline [
Employee|Address
Bill|New York
Mary|London
Steve|Chicago
Lucy|Madrid
Lucy|Paris
John|Miami
] (delimiter is '|') where not Exists (Name, Employee);

Drop Tables Employees;
```

Lucy のすべての値を取得できるようにするために、次の 2 つが変更されました。

- Employees テーブルへの先行する LOAD が挿入され、Employee の名前が Name に変更されました。
  `Load Employee As Name;`
- Citizens の Where 条件は次のように変更されました。
  `not Exists (Name, Employee).`

これにより、Name とEmployee の項目が作成されます。Lucy の 2 行目を確認すると、引き続き Name には存在していません。

結果

Employee	住所
Mary	London
Lucy	Madrid
Lucy	Paris

FieldIndex

FieldIndex() は、**field_name** 項目内の **value** 項目値の位置を返します (ロード順)。

構文:

```
FieldIndex(field_name , value)
```

戻り値データ型: 整数

引数:

引数

引数	説明
field_name	インデックスが必要な項目名。たとえば、テーブルの列など。文字列値でなければなりません。これは、項目名は単一引用符で囲む必要があることを意味します。
value	項目 **field_name** の値。

制限事項:

- **value** が項目 **field_name** の項目値の中にない場合は、0 を返します。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。この制限は、同等のスクリプト関数には当てはまりません。

例と結果:

以下の例では、テーブル **Names** の項目 **First name** を使用しています。

例と結果

例	結果
アプリに例のデータを追加して実行します。	テーブル **Names** が、例のデータのようにロードされます。

例	結果
チャート関数:軸 First name を含むテーブルで、メジャーとして次の項目を追加します。	
`FieldIndex ('First name','John')`	1 ('John' が **First name** 項目のロード順序の最初に登場するため) ただし、フィルター パネルではアルファベット順にソートされるため、**John** は上から 2 番目に表示されます)
`FieldIndex ('First name','Peter')`	4 (**FieldIndex()** では、1 つの値しか返されないため。この場合は 4 がロード順序における最初の項目)
スクリプト関数: テーブル **Names** が、例のデータのようにロードされます。	
`John1: Load FieldIndex('First name','John') as MyJohnPos Resident Names;`	`MyJohnPos`=1 ('John' が**First name** 項目のロード順序の最初に登場するため) ただし、フィルター パネルではアルファベット順にソートされるため、**John** は上から 2 番目に表示されます)
`Peter1: Load FieldIndex('First name','Peter') as MyPeterPos Resident Names;`	`MyPeterPos`=4 (**FieldIndex()** では、1 つの値しか返されないため。この場合は 4 がロード順序における最初の項目)

例で使用されているデータ:

```
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() は、**field_name** 項目の **elem_no** の位置にある値を返します (ロード順)。

構文:

```
FieldValue(field_name , elem_no)
```

戻り値データ型： デュアル

引数：

引数

引数	説明
field_name	値が必要な項目名。たとえば、テーブルの列など。文字列値でなければなりません。これは、項目名は単一引用符で囲む必要があることを意味します。
elem_no	ロード順序において、返される値が含まれる項目の位置 (要素) 番号を表します。これは、テーブルの行に対応している場合がありますが、要素 (行) がロードされる順序によっては対応していない場合もあります。

制限事項:

- **elem_no** が項目値の数より大きい場合は、NULL が返されます。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。この制限は、同等のスクリプト関数には当てはまりません。

例

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下の例を作成します。

```
Names:

LOAD * inline [
First name|Last name|Initials|Has cellphone
John|Anderson|JA|Yes
Sue|Brown|SB|Yes
Mark|Carr|MC |No
Peter|Devonshire|PD|No
Jane|Elliot|JE|Yes
Peter|Franc|PF|Yes ] (delimiter is '|');

John1:

Load FieldValue('First name',1) as MyPos1
Resident Names;

Peter1:

Load FieldValue('First name',5) as MyPos2
Resident Names;
```

ビジュアライゼーションの作成

Qlik Sense シートにテーブル ビジュアライゼーションを作成します。項目 **First name**、**MyPos1**、**MyPos2** をテーブルに追加します。

結果

First name	MyPos1	MyPos2
Jane	John	Jane
John	John	Jane
Mark	John	Jane
Peter	John	Jane
Sue	John	Jane

説明

FieldValue('First name','1') は、John が **First name** 項目のロード順序で最初に表示されるため、すべての名の **MyPos1** の値として、John になります。ただし、フィルター パネルではアルファベット順にソートされるため、John は上から2 番目、Jane の後に表示されます)

FieldValue('First name','5') は、Jane が **First name** 項目のロード順で 5 番目に表示されるため、すべての名の **MyPos2** の値として Jane になります。

FieldValueCount

FieldValueCount() は **integer** 関数で、項目に含まれる固有値の数を返します。

部分的なリロードにより、データ由来の値が削除される可能性がありますが、返される数値には反映されません。返される数値は、最初のリロードまたはその後の部分的なリロードのいずれかでロードされたすべての個別値に対応します。

チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。この制限は、同等のスクリプト関数には当てはまりません。

構文:

```
FieldValueCount(field_name)
```

戻り値データ型: 整数

引数:

引数

引数	説明
field_name	値が必要な項目名。たとえば、テーブルの列など。文字列値でなければなりません。これは、項目名は単一引用符で囲む必要があることを意味します。

例と結果:

以下の例では、テーブル **Names** の項目 **First name** を使用しています。

例と結果

例	結果
アプリに例のデータを追加して実行します。	テーブル **Names** が、例のデータのようにロードされます。
チャート関数:軸 First name を含むテーブルで、メジャーとして次の項目を追加します。	
`FieldValueCount('First name')`	5 (**Peter** は 2 回登場するため)
`FieldValueCount('Initials')`	6 (**Initials** には固有の値しか含まれていないため)
スクリプト関数: テーブル **Names** が、例のデータのようにロードされます。	

`FieldCount1:`
`Load FieldValueCount('First name') as`
`MyFieldCount1`
`Resident Names;`
	`MyFieldCount1`=5 (Peter は 2 回登場するため)

`FieldCount2:`
`Load FieldValueCount('Initials') as`
`MyInitialsCount1`
`Resident Names;`
	`MyFieldCount1`=6 (Initials には固有の値しか含まれていないため)

例で使用されているデータ:

```
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() は、すでにロードされているテーブルを参照し、項目 **match_field_name** における値 **match_field_value** の最初の出現に対応する **field_name** の値を返します。テーブルは、現在のテーブルまたは前にロードして別のテーブルにすることができます。

構文:

```
lookup(field_name, match_field_name, match_field_value [, table_name])
```

戻り値データ型: デュアル

引数:

引数

引数	説明
field_name	戻り値が必要な項目名。入力値は文字列として指定する必要があります (たとえば、引用符で囲まれた文字列)。
match_field_name	**match_field_value** を検索する項目の名前。入力値は文字列として指定する必要があります (たとえば、引用符で囲まれた文字列)。
match_field_value	**match_field_name** 項目で検索する値。
table_name	値を検索するテーブルの名前。入力値は文字列として指定する必要があります (例えば、引用符で囲まれた文字列)。 **table_name** が省略されている場合は、現在のテーブルとして処理されます。

引数に引用符がない場合、現在のテーブルを参照します。他のテーブルを参照するには、引数を単一引用符で囲みます。

制限事項:

連結などの複雑な操作の結果テーブルでは、検索順は明確に定義されませんが、それ以外の場合、検索はロード順に実行されます。**field_name** と**match_field_name** は両方とも、**table_name** で指定された同一テーブルの項目であることが必要です。

一致するものがない場合は、NULL が返されます。

例

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下の例を作成します。

```
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;
```

ビジュアライゼーションの作成

Qlik Sense シートにテーブル ビジュアライゼーションを作成します。項目 **ProductID**、**InvoiceID**、**CustomerID**、**Units**、**CategoryID** をテーブルに追加します。

結果

結果のテーブル

ProductID	InvoiceID	CustomerID	ユニット:	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

説明

サンプル データでは、**Lookup()** 関数を以下の形式で使用します。

```
Lookup('Category', 'ProductID', ProductID, 'ProductList')
```

ProductList テーブルが最初にロードされます。

Lookup() 関数を使用して、**OrderData** テーブルが構築されます。この関数では、3 番目の引数として、**ProductID** が指定されています**ProductList** の **'ProductID'** (単一引用符で囲まれた 2 番目の引数) で、この項目の値が検索されます。

関数は、**'Category'** (**ProductList** テーブル) の値を **CategoryID.** としてロードして返します。

drop ステートメントは、不要になった **ProductList** テーブルをデータモデルから削除します。結果の **OrderData** テーブルが残ります。

> *Lookup() 関数には柔軟性があり、過去にロードしたテーブルにもアクセスできます。ただし、Applymap() 関数と比べると、処理に時間がかかります。*

参照先:

- *ApplyMap (page 1300)*

NoOfRows - チャート関数

NoOfRows() は、テーブルの現在の列セグメント内の行の数を返します。ビットマップチャートの場合、**NoOfRows()** はチャートのストレートテーブルに相当するセグメントに含まれる行の数を返します。

テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除くすべての軸列の現在行と同じ値を持つ行だけが含まれます。

> *チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。*

構文:

```
NoOfRows([TOTAL])
```

戻り値データ型: 整数

引数:

引数

引数	説明
TOTAL	テーブルが 1 軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

例: NoOfRows を使用したチャートの数式

例 - チャートの数式

ロード スクリプト

以下のデータをインライン データとしてデータロードエディタにロードして、以下のチャートの数式の例を作成します。

```
        Temp:
LOAD * inline [
Region|SubRegion|RowNo()|NoOfRows()
Africa|Eastern
Africa|Western
Americas|Central
Americas|Northern
Asia|Eastern
Europe|Eastern
Europe|Northern
Europe|Western
Oceania|Australia
] (delimiter is '|');
```

チャートの数式

Qlik Sense シートに **Region** と **SubRegion** を軸としたテーブルのビジュアライゼーションを作成します。`RowNo()`、 `NoOfRows()`、 `NoOfRows(Total)` をメジャーとして追加します。

結果

Region	SubRegion	RowNo()	NoOfRows()	NoOfRows (Total)
Africa	Eastern	1	2	9
Africa	Western	2	2	9
Americas	Central	1	2	9
Americas	Northern	2	2	9
Asia	Eastern	1	1	9
Europe	Eastern	1	3	9
Europe	Northern	2	3	9
Eurrope	Western	3	3	9
Oceania	Australia	1	1	9

説明

この例では、ソート順は最初の軸であるRegionによるものです。その結果、各列セグメントは、同じ値を持つ地域のグループで構成されます (例: アフリカ)。

RowNo() 列には、各列セグメントの行番号が表示されます。例えば、アフリカ地域には 2 つの行があります。行番号は、次の列セグメントAmericas でも再度 1 から始まります。

NoOfRows() 列は、各列セグメントの行数をカウントします。例えば、ヨーロッパには列セグメントに 3 つの行があります。

NoOfRows(Total) 列は NoOfRows() の TOTAL 引数のために軸を無視し、テーブルの行をカウントします。

テーブルが 2 番目の軸であるSubRegion でソートされた場合、列セグメントはその軸に基づいているため、SubRegion ごとに行番号が変更されます。

参照先:

- *RowNo - チャート関数 (page 581)*

Peek

Peek() は、すでにロードされている行に対してテーブルの項目値を返します。行番号は、テーブルと同様に指定できます。行番号が指定されていない場合は、最後にロードされたレコードが使用されます。

peek() 関数は、以前にロードされたテーブル内の関連する境界、つまり特定の項目の最初の値または最後の値を見つけるために最もよく使用されます。ほとんどの場合、この値は後で使用するために、例えば do-while ループの条件として変数に格納されます。

構文:

```
Peek(
field_name
[, row_no[, table_name ] ])
```

戻り値データ型: dual

引数:

引数

引数	説明
field_name	戻り値が必要な項目名。入力値は文字列として指定する必要があります (たとえば、引用符で囲まれた文字列)。

引数	説明
row_no	テーブルの行。必要な項目がある行を指定します。数式の場合を指定することもできますが、結果が整数になる必要があります。0 は最初のレコード、1 は 2 番目のレコードを示し、以下同様に表されます。負の数は、テーブルの最後から見た順序を表します。-1 は、読み取られた最後のレコードを示します。 **row_no** が指定されていない場合は、-1 として処理されます。
table_name	末尾にコロンが付いていない、テーブルのラベルです。**table_name** が指定されていない場合は、現在のテーブルとして処理されます。**LOAD** ステートメント以外で使用する、または他のテーブルを参照する場合は、**table_name** が含まれている必要があります。

制限事項:

この関数は、既にロードされているレコードからのみ値を返すことができます。これは、テーブルの最初のレコードで、row_no として -1 を使用する呼び出しが NULL を返すことを意味します。

例と結果:

例 1

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
EmployeeDates:
Load * Inline [
EmployeeCode|StartDate|EndDate
101|02/11/2010|23/06/2012
102|01/11/2011|30/11/2013
103|02/01/2012|
104|02/01/2012|31/03/2012
105|01/04/2012|31/01/2013
106|02/11/2013|
] (delimiter is '|');

First_last_Employee:
Load
EmployeeCode,
Peek('EmployeeCode',0,'EmployeeDates') As FirstCode,
Peek('EmployeeCode',-1,'EmployeeDates') As LastCode
Resident EmployeeDates;
```

結果のテーブル

従業員コード	StartDate	EndDate	FirstCode	LastCode
101	02/11/2010	23/06/2012	101	106
102	01/11/2011	30/11/2013	101	106
103	02/01/2012		101	106
104	02/01/2012	31/03/2012	101	106

従業員コード	StartDate	EndDate	FirstCode	LastCode
105	01/04/2012	31/01/2013	101	106
106	02/11/2013		101	106

`Peek('EmployeeCode',0, 'EmployeeDates')` が、EmployeeDates テーブルの EmployeeCode の最初の値を返すため、FirstCode = 101 になります。

`Peek('EmployeeCode',-1, 'EmployeeDates')` はテーブル EmployeeDates の EmployeeCode の最後の値を返すため LastCode = 106 です。

引数 **row_no** の値を置き換えた場合、以下のように、テーブルの他の行の値を返します。

`Peek('EmployeeCode',2, 'EmployeeDates')` はテーブル内の 3 番目の値 103 を FirstCode として返します。

ただし、これらの例で 3 番目の引数 **table_name** にテーブルを指定しない場合は、関数は現在のテーブル (この場合は、内部テーブル) を参照します。

例 2

テーブルのさらに下のデータにアクセスする場合は、2 つのステップでアクセスする必要があります。最初にテーブル全体を一時的なテーブルにロードし、次に **Peek()** を使用するときに再ソートします。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
T1:
LOAD * inline [
ID|Value
1|3
1|4
1|6
3|7
3|8
2|1
2|11
5|2
5|78
5|13
] (delimiter is '|');

T2:

LOAD *,
IF(ID=Peek('ID'), Peek('List')&','&Value,Value) AS List
RESIDENT T1
ORDER BY ID ASC;
DROP TABLE T1;
```

Create a table in a sheet in your app with **ID**, **List**, and **Value** as the dimensions.

結果のテーブル

ID	リスト	値
1	3,4	4
1	3,4,6	6
1	3	3
2	1,11	11
2	1	1
3	7,8	8
3	7	7
5	2,78	78
5	2,78,13	13
5	2	2

この **IF()** ステートメントは、一時的なテーブル T1 を利用して構築されています。
`Peek('ID')` は、現在のテーブル T2 の現在の行の 1 行前の ID 項目を参照します。
`Peek('List')` は、T2 の現在の行の 1 行前の List 項目を参照しており、評価対象の数式として構築されています。

ステートメントは次のように評価されます。
ID の現在の値が ID の 1 行前の値と同じ場合、Peek('List') の値を Value の現在の値と連結して書き込みます。それ以外の場合は、Value の現在の値のみ書き込みます。

Peek('List') に連結された結果がすでに含まれている場合、Peek('List') の新しい結果がその結果に連結されます。

> ***Order by** 節に注意してください。表のソート方法 (ID による昇順) を指定しています。この指定がない場合、**Peek()** は、内部テーブルのデータ順序を使用するので、予測不可能な結果につながります。*

例 3

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
Amounts:
Load
Date#(Month,'YYYY-MM') as Month,
Amount,
Peek(Amount) as AmountMonthBefore
Inline
[Month,Amount
2022-01,2
2022-02,3
2022-03,7
```

```
2022-04,9
2022-05,4
2022-06,1];
```

結果のテーブル

Amount	AmountMonthBefore	月
1	4	2022-06
2	-	2022-01
3	2	2022-02
4	9	2022-05
7	3	2022-03
9	7	2022-04

項目 AmountMonthBefore は前月の金額を保持します。

ここでは、row_no パラメータとtable_name パラメータが省略されているため、既定値が使用されます。この例では、次の 3 つの関数呼び出しは同等です。

- Peek(Amount)
- Peek(Amount,-1)
- Peek(Amount,-1,'Amounts')

row_no として -1 を使用することは、前の行の値が使用されることを意味します。この値を代入することにより、テーブル内の他の行の値を取得できます。

Peek(Amount,2) は、テーブルの 3 番目の値 7 を返します。

例 4:

正しい結果を得るには、データを正しくソートする必要がありますが、残念ながら、これが常に当てはまるとは限りません。さらに、Peek() 関数を使用して、まだロードされていないデータを参照することはできません。一時的なテーブルを使用し、データを複数回パスすることで、このような問題を回避できます。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
tmp1Amounts:
Load * Inline
[Month,Product,Amount
2022-01,B,3
2022-01,A,8
2022-02,B,4
2022-02,A,6
```

```
2022-03,B,1
2022-03,A,6
2022-04,A,5
2022-04,B,5
2022-05,B,6
2022-05,A,7
2022-06,A,4
2022-06,B,8];

tmp2Amounts:
Load *,
If(Product=Peek(Product),Peek(Amount)) as AmountMonthBefore
Resident tmp1Amounts
Order By Product, Month Asc;
Drop Table tmp1Amounts;

Amounts:
Load *,
If(Product=Peek(Product),Peek(Amount)) as AmountMonthAfter
Resident tmp2Amounts
Order By Product, Month Desc;
Drop Table tmp2Amounts;
```

説明

初期テーブルは月ごとに並べ替えられます。つまり、peek() 関数は、多くの場合、間違った製品の金額を返します。したがって、このテーブルは再ソートする必要があります。これは、新しいテーブル tmp2Amounts を作成するデータの 2 番目のパスを実行することによって行われます。Order By 節に注意してください。最初に製品ごとに、次に月ごとに昇順でレコードをソートします。

AmountMonthBefore は、前の行に同じ製品の前月のデータが含まれている場合にのみ計算する必要があるため、If() 関数が必要です。現在の行の製品を前の行の製品と比較することにより、この条件を検証できます。

2 番目のテーブルが作成されると、Drop Table ステートメントを使用して最初のテーブル tmp1Amounts がドロップされます。

最後に、データを介して 3 番目のパスが作成されますが、月は逆の順序でソートされます。このようにして、AmountMonthAfter も計算できます。

Order by 句は、テーブルの順序を指定します。これらがないと、Peek() 関数は内部テーブルの任意の順序を使用するため、予期しない結果が生じる可能性があります。

結果

結果のテーブル

月	製品	Amount	AmountMonthBefore	AmountMonthAfter
2022-01	ライン番号の隣にある	8	-	6
2022-02	B	3	-	4
2022-03	ライン番号の隣にある	6	8	6
2022-04	B	4	3	1
2022-05	ライン番号の隣にある	6	6	5
2022-06	B	1	4	5
2022-01	ライン番号の隣にある	5	6	7
2022-02	B	5	1	6
2022-03	ライン番号の隣にある	7	5	4
2022-04	B	6	5	8
2022-05	ライン番号の隣にある	4	7	-
2022-06	B	8	6	-

例 5

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
T1:
Load * inline [
Quarter, Value
2003q1, 10000
2003q1, 25000
2003q1, 30000
2003q2, 1250
2003q2, 55000
2003q2, 76200
```

```
2003q3, 9240
2003q3, 33150
2003q3, 89450
2004q4, 1000
2004q4, 3000
2004q4, 5000
2004q1, 1000
2004q1, 1250
2004q1, 3000
2004q2, 5000
2004q2, 9240
2004q2, 10000
2004q3, 25000
2004q3, 30000
2004q3, 33150
2004q4, 55000
2004q4, 76200
2004q4, 89450 ];

T2:
Load *, rangesum(SumVal,peek('AccSumVal')) as AccSumVal;
Load Quarter, sum(Value) as SumVal resident T1 group by Quarter;
```

結果

結果のテーブル

四半期	SumVal	AccSumVal
2003q1	65000	65000
2003q2	132450	197450
2003q3	131840	329290
2004q4	9000	338290
2004q1	5250	343540
2004q2	24240	367780
2004q3	88150	455930
2004q4	220650	676580

説明

Load ステートメント **Load *, rangesum(SumVal,peek('AccSumVal')) as AccSumVal** には、前の値が現在の値に追加される再帰呼び出しが含まれています。この演算子はスクリプト内の値の累積を計算するために使用されます。

参照先:

Previous

Previous()は、**where**節のために破棄されなかった以前の入力レコードのデータを使用して、**expr**数式の値を算出します。内部テーブルの最初のレコードの場合は、NULL を返します。

構文:

```
Previous(expr)
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。 数式に **previous()** 関数をネストすることで、さらに前のレコードにアクセスすることもできます。データは入力ソースから直接取得されるため、Qlik Sense にまだロードされていない項目を参照することができます。つまり、その連想データベースに保存されていなくても項目を参照できます。

制限事項:

内部テーブルの最初のレコードの場合は、NULL を返します。

次をロードスクリプトに入力します。

```
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 '|');
```

Previous() 関数を **Load** ステートメントで使用することで、Sales の現在の値を先行する値と比較でき、3 番目の項目 Increase で使用できます。

結果のテーブル

月	売上	増加
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 - チャート関数

Top() テーブルの列セグメント内の最初 (最上部) の行の数式を評価します。どの行が計算されるかは、**offset** 値により決定されますが、デフォルトは最上部の行です。テーブル以外のチャートでは、**Top()** 評価はチャートのストレートテーブルに相当する現在の列の最初の行を評価します。

構文:

```
Top([TOTAL] expr [ , offset [,count ]])
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
offset	**offset** の n を 1 より大きい値にすると、数式の評価が先頭行から n 行下に移動します。 offset が負の値である場合、**Top** 関数は、相応する正の offset 値が付いた **Bottom** 関数と同様に機能します。
count	3 番目のパラメータである **count** を 1 より大い値に設定すると、この関数は **count** 値の範囲、つまり現在の列セグメントの最後の **count** 行の各値を返します。この形式では、特別な範囲関数の引数として関数を使用できます。 *範囲関数 (page 1307)*
TOTAL	テーブルが 1 軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

列セグメントは、現在のソート順で軸に同じ値を持つ連続したセルのサブセットとして定義されます。レコード間チャート関数は、チャートのストレートテーブルに相当する右端の軸を除外して列セグメントで実行されます。チャートに軸が 1 つしかない場合、または TOTAL 修飾子が指定されていると、数式はテーブル全体を評価します。

テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除くすべての軸列の現在行と同じ値を持つ行だけが含まれます。

制限事項:

- 再帰呼び出しは NULL を返します。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

例と結果:

1

この例で示されているテーブルのスクリーンショットでは、軸 **Customer** とメジャー `Sum(Sales)` および `Top(Sum(Sales))` からテーブルのビジュアライゼーションが作成されています。

列 **Top(Sum(Sales))**はすべての行で 587 を返します (開始行 **Astrida** の値)。

テーブルには、Sum(Sales)+Top(Sum(Sales))から作成されたものと、数式Sum(Sales)+Top(Sum(Sales), 3)を使用して作成され、引数**offset**が3に設定されている**Top offset 3**という、より複雑なメジャーも表示されています。先頭から3つ下にある行の値に現在の行の **Sum(Sales)** 値を加えます (現在の行 + **Canutility** の値)。

例 1

Top and Bottom

Customer	Sum(Sales)	Top(Sum(Sales))	Sum(Sales)+Top(Sum(Sales))	Top offset 3
Totals	2566	587	3153	3249
Astrida	587	587	1174	1270
Betacab	539	587	1126	1222
Canutility	683	587	1270	1366
Divadip	757	587	1344	1440

2

この例で示されているテーブルのスクリーンショットでは、より多くの軸がビジュアライゼーションに追加されています。および **Month**、**Product** の 2 つのメジャーがあります。複数の軸が含まれているチャートでは、**Above**、**Below**、**Top**、**Bottom** 関数を含む数式の結果は、Qlik Sense における列軸のソート順序によって変わります。Qlik Sense は、最後にソートされた軸の結果である列セグメントに基づいて関数を評価します。列のソート順は、[ソート] のプロパティパネルで制御され、必ずしも列がテーブルに表示される順序ではありません。

例 2 の最初のテーブル。First value メジャーの Top の値は Month (Jan) に基づいています。

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

例 2 の 2 番目のテーブル。First value メジャーの Top の値は、Product (Astrida の AA) に基いています。

Customer	Product	Month	Sum(Sales)	First value
			2566	-
Astrida	AA	Jan	46	46
Astrida	BB	Jan	46	46
Astrida	AA	Feb	60	60
Astrida	BB	Feb	60	60
Astrida	AA	Mar	70	70
Astrida	BB	Mar	70	70
Astrida	AA	Apr	13	13
Astrida	BB	Apr	13	13

詳細については、**Above** 関数の例 2 を参照してください。

例 3

3	**結果**
Top 関数は、範囲関数への入力として使用できます。例: `RangeAvg (Top(Sum(Sales),1,3))`。	**Top()** 関数の引数では、offset は 1 に設定され、count は 3 に設定されています。この関数は、列セグメントの最終行の下の行から始まる 3 行 (offset=1 のため) およびその下の 2 行 (行がある場合) で数式 **Sum(Sales)** の結果を算出します。この 3 つの値は、RangeAvg() 関数への入力として使用され、指定された数値の範囲で平均値を算出します。 軸として **Customer** を有するテーブルによって、RangeAvg() の数式について次の結果が得られます。
	Astrida 603 Betacab 603 Canutility 603 Divadip: 603

```
Monthnames:
LOAD *, Dual(MonthText,MonthNumber) as Month INLINE [
MonthText, MonthNumber
Jan, 1
Feb, 2
Mar, 3
Apr, 4
May, 5
Jun, 6
Jul, 7
Aug, 8
```

```
Sep, 9
Oct, 10
Nov, 11
Dec, 12
];

Sales2013:
Crosstable (MonthText, Sales) LOAD * inline [
Customer|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec
Astrida|46|60|70|13|78|20|45|65|78|12|78|22
Betacab|65|56|22|79|12|56|45|24|32|78|55|15
Canutility|77|68|34|91|24|68|57|36|44|90|67|27
Divadip|57|36|44|90|67|27|57|68|47|90|80|94
] (delimiter is '|');
```

参照先:

- *Bottom - チャート関数 (page 1260)*
- *Above - チャート関数 (page 1252)*
- *Sum - チャート関数 (page 348)*
- *RangeAvg (page 1310)*
- *範囲関数 (page 1307)*

SecondaryDimensionality- チャート関数

SecondaryDimensionality() は、集計以外の内容 (部分合計または折りたたまれた集計を含まない) を含む軸のピボットテーブル行の数を返します。この関数は、水平ピボットテーブル軸の **dimensionality()** 関数に相当します。

構文:

```
SecondaryDimensionality( )
```

戻り値データ型: integer

制限事項:

- ピボットテーブルで使用される場合を除き、**SecondaryDimensionality** 関数は常に 0 を返します。
- チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

After - チャート関数

After() は、ピボットテーブルの行セグメント内の現在列の後の列に、ピボットテーブルの軸値で評価された expression の値を返します。

構文:

```
after([TOTAL] expr [, offset [, count ]])
```

チャートの式いずれかにこのチャート関数が使用されている場合、チャートのy値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

ピボットテーブル以外のすべてのチャートタイプの場合、この関数はNULLを返します。

引数:

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
offset	**offset** nを1より大きい値にすると、数式の評価が現在の行からn行右に移動します。 offsetに0を指定すると、現在の行で数式が評価されます。 offsetが負の値である場合、**After**関数は、相応する正のoffset値が付いた**Before**関数と同様に機能します。
count	3番目のパラメータ**count**を1より大きい値にした場合、この関数は**count**値の範囲を返します。つまり、オリジナルのテーブルのセルから右方向にcount行の各値を返します。
TOTAL	テーブルが1軸の場合、または**TOTAL**修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

行セグメントの最後の列では、これより後に列が存在しないためNULL値が返されます。

ピボットテーブルに複数の水平軸が存在する場合、現在の行セグメントには、項目ソート順の最後の水平軸を示す行を除くすべての軸行の現在列と同じ値を持つ列だけが含まれます。ピボットテーブルの水平軸の項目間ソート順は、上から下への軸の順序で定義されます。.

```
after( sum( Sales ))
after( sum( Sales ), 2 )
after( total sum( Sales ))
```

`rangeavg (after(sum(x),1,3))`は、在列のすぐ右の3つの列で評価された**sum(x)**関数の3つの結果の平均を返します。

Before - チャート関数

Before()は、ピボットテーブルの行セグメント内の現在列の前の列に、ピボットテーブルの軸値で評価されたexpressionの値を返します。

構文:

```
before([TOTAL] expr [, offset [, count]])
```

ピボット テーブル以外のすべてのチャート タイプの場合、この関数は NULL を返します。

チャートの式いずれかにこのチャート関数が使用されている場合、チャートの y 値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
offset	**offset** n を 1 より大きい値にすると、数式の評価が現在の行から n 行左に移動します。 offset に 0 を指定すると、現在の行で数式が評価されます。 offset が負の値である場合、**Before** 関数は、相応する正の offset 値が付いた **After** 関数と同様に機能します。
count	3 番目のパラメータ **count** を 1 より大きい値にした場合、この関数は **count** 値の範囲を返します。つまり、オリジナルのテーブルのセルから左方向に count 行の各値を返します。
TOTAL	テーブルが 1 軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

行セグメントの最初の列では、これより前に列が存在しないため NULL 値が返されます。

ピボット テーブルに複数の水平軸が存在する場合、現在の行セグメントには、項目ソート順の最後の水平軸を示す行を除くすべての軸行の現在列と同じ値を持つ列だけが含まれます。ピボット テーブルの水平軸の項目間ソート順は、上から下への軸の順序で定義されます。.

```
before( sum( Sales ))
before( sum( Sales ), 2 )
before( total sum( Sales ))
```

`rangeavg (before(sum(x),1,3))` は、在列のすぐ左の 3 つの列で評価された **sum(x)** 関数の 3 つの結果の平均を返します。

First - チャート関数

First() は、ピボット テーブルの現在の行セグメントの最初の列に、ピボット テーブルの軸値で評価された expression の値を返します。ピボット テーブル以外のすべてのチャート タイプの場合、この関数は NULL を返します。

チャートの式いずれかにこのチャート関数が使用されている場合、チャートのy値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

構文:

```
first([TOTAL] expr [, offset [, count]])
```

引数:

引数

引数	説明
expression	メジャーの対象となるデータが含まれている数式または項目。
offset	**offset** n を1より大きい値にすると、数式の評価が現在の行からn行右に移動します。 offset に 0 を指定すると、現在の行で数式が評価されます。 offset が負の値である場合、**First** 関数は、相応する正の offset 値が付いた **Last** 関数と同様に機能します。
count	3番目のパラメータ**count** を1より大きい値にした場合、この関数は **count** 値の範囲を返します。つまり、オリジナルのテーブルのセルから右方向に count 行の各値を返します。
TOTAL	テーブルが1軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

ピボットテーブルに複数の水平軸が存在する場合、現在の行セグメントには、項目ソート順の最後の水平軸を示す行を除くすべての軸行の現在列と同じ値を持つ列だけが含まれます。ピボットテーブルの水平軸の項目間ソート順は、上から下への軸の順序で定義されます。.

```
first( sum( Sales ))
first( sum( Sales ), 2 )
first( total sum( Sales )
rangeavg (first(sum(x),1,5))
```

現在の行セグメントの左端 5 つの列で評価された **sum(x)** 関数の結果の平均を返します。

Last - チャート関数

Last() は、ピボットテーブルの現在の行セグメントの最後の列に、ピボットテーブルの軸値で評価された expression の値を返します。ピボットテーブル以外のすべてのチャートタイプの場合、この関数は NULL を返します。

チャートの式いずれかにこのチャート関数が使用されている場合、チャートのy値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

構文:

```
last([TOTAL] expr [, offset [, count]])
```

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
offset	**offset** n を1より大きい値にすると、数式の評価が現在の行からn行左に移動します。 offset に 0 を指定すると、現在の行で数式が評価されます。 offset が負の値である場合、**First** 関数は、相応する正の offset 値が付いた**Last** 関数と同様に機能します。
count	3番目のパラメータ**count** を1より大きい値にした場合、この関数は **count** 値の範囲を返します。つまり、オリジナルのテーブルのセルから左方向に count 行の各値を返します。
TOTAL	テーブルが1軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

ピボットテーブルに複数の水平軸が存在する場合、現在の行セグメントには、項目ソート順の最後の水平軸を示す行を除くすべての軸行の現在列と同じ値を持つ列だけが含まれます。ピボットテーブルの水平軸の項目間ソート順は、上から下への軸の順序で定義されます。.

```
last( sum( Sales ))
last( sum( Sales ), 2 )
last( total sum( Sales )
```

`rangeavg (last(sum(x),1,5))` は、現在の行セグメントの右端 5 つの列で評価された**sum(x)** 関数の結果の平均を返します。

ColumnNo - チャート関数

ColumnNo() は、ピボットテーブルの現在の行セグメント内の現在列の番号を返します。最初の列の番号は1です。

構文:

```
ColumnNo([total])
```

引数:

引数

引数	説明
TOTAL	テーブルが1軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

ピボットテーブルに複数の水平軸が存在する場合、現在の行セグメントには、項目ソート順の最後の水平軸を示す行を除くすべての軸行の現在列と同じ値を持つ列だけが含まれます。ピボットテーブルの水平軸の項目間ソート順は、上から下への軸の順序で定義されます。.

チャートの式いずれかにこのチャート関数が使用されている場合、チャートのy値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

```
if( ColumnNo( )=1, 0, sum( Sales ) / before( sum( Sales )))
```

NoOfColumns - チャート関数

NoOfColumns() は、ピボットテーブルの現在の行セグメント内の列の数を返します。

チャートの式いずれかにこのチャート関数が使用されている場合、チャートのy値のソート、またはテーブルの式列ごとのソートは許可されません。よって、これらのソート機能は自動的に無効になります。ビジュアライゼーションまたはテーブルでこのチャート関数を使用すると、ビジュアライゼーションのソートは、レコード間の関数に対するソートされた入力の状態に戻ります。

構文:

```
NoOfColumns([total])
```

引数:

引数

引数	説明
TOTAL	テーブルが1軸の場合、または **TOTAL** 修飾子が引数として使用される場合は、現在の列セグメントは常に列全体と等しくなります。

ピボットテーブルに複数の水平軸が存在する場合、現在の行セグメントには、項目ソート順の最後の軸を示す行を除くすべての軸行の現在列と同じ値を持つ列だけが含まれます。ピボットテーブルの水平軸の項目間ソート順は、上から下への軸の順序で定義されます。.

```
if( ColumnNo( )=NoOfColumns( ), 0, after( sum( Sales )))
```

8.17 論理関数

このセクションでは、論理演算子を処理する関数について説明します。すべての関数は、データロードスクリプトおよびチャートの数式の両方で使用できます。

IsNum

数式を数値として解釈できる場合は -1 (True)、それ以外の場合は 0 (False) を返します。

```
IsNum( expr )
```

IsText

数式にテキスト表現がある場合は -1 (True)、それ以外の場合は 0 (False) を返します。

```
IsText( expr )
```

> ***IsNum*** *も* ***IsText*** *も、数式が NULL の場合、0 を返します。*

以下の例では、テキストと数値が混ざった値を持つインライン テーブルをロードし、それぞれの値が数値かテキスト値かチェックします。

```
Load *, IsNum(Value), IsText(Value)
Inline [
Value
23
Green
Blue
12
33Red];
```

この結果、テーブルは次のようになります。

Resulting table

Value	IsNum(Value)	IsText(Value)
23	-1	0
Green	0	-1
Blue	0	-1
12	-1	0
33Red	0	-1

8.18 マッピング関数

このセクションでは、マッピング テーブルを処理する関数について説明します。マッピング テーブルは、スクリプトの実行中に項目値または項目名を置き換える際に使用できます。

マッピング関数は、データロード スクリプトでのみ使用できます。

マッピング関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

ApplyMap
ApplyMap スクリプト関数は、以前ロードされたマッピング テーブルの数式のアウトプットのマッピングに使用されます。

```
ApplyMap ('mapname', expr [ , defaultexpr ] )
```

MapSubstring
MapSubstring スクリプト関数を使用すると、以前ロードされたマッピング テーブルに任意の数式の一部をマップできます。マッピングでは大文字と小文字が区別され、反復されません。サブストリングは左から右にマップされます。

```
MapSubstring ('mapname', expr)
```

ApplyMap

ApplyMap スクリプト関数は、以前ロードされたマッピング テーブルの数式のアウトプットのマッピングに使用されます。

構文:

```
ApplyMap('map_name', expression [ , default_mapping ] )
```

戻り値データ型: dual

引数:

引数

引数	説明
map_name	以前に **mapping load** または **mapping select** ステートメントで作成されたマッピング テーブルの名前です。この名前は、単一引用符で囲む必要があります。 *この関数をマクロ展開の変数で使用し、存在しないマッピング テーブルを参照する場合、関数呼び出しが失敗して項目は作成されません。*

引数	説明
expression	結果がマッピングされる数式です。
default_mapping	指定された場合、この値は、マッピング テーブルに expression に一致する値が存在しない場合、既定値として使用されます。指定されない場合は、expression の値がそのまま返されます。

ApplyMap の出力項目の名前は、ApplyMap の入力項目の名前と同じにしないようにする必要があります。同じにすると予期しない結果になる可能性があります。使用しない例: `ApplyMap ('Map', A) as A`。

この例では、Salesperson とその居住国の国コードのリストをロードします。国コードを国名に置き換えるために、国コードを国にマッピングしたテーブルを使用します。このマッピング テーブルでは、3 つの国のみが定義されており、他の国は 'Rest of the world' としてマッピングされています。

```
// 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';
```

出力されるテーブル (Salespersons) は次のようになります。

Resulting table

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

MapSubstring

MapSubstring スクリプト関数を使用すると、以前ロードされたマッピング テーブルに任意の数式の一部をマップできます。マッピングでは大文字と小文字が区別され、反復されません。サブストリングは左から右にマップされます。

構文:

```
MapSubstring('map_name', expression)
```

戻り値データ型: string

引数:

引数

引数	説明
map_name	**mapping load** または **mapping select** ステートメントを使って事前にロードされたマッピング テーブルの名前です。名前は単一引用符で囲む必要があります。  *この関数をマクロ展開の変数で使用し、存在しないマッピング テーブルを参照する場合、関数呼び出しが失敗して項目は作成されません。*
expression	結果がサブストリングによってマッピングされる数式です。

この例では、製品モデルのリストをロードします。各モデルの属性は、合成されたコードで記述されます。マッピング テーブルとMapSubstring を使用すると、属性コードを説明書きに展開できます。

```
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';
```

この結果、テーブルは次のようになります。

Resulting table

Model	**Description**
Twixie	Red Cotton Small
Boomer	Blue Polyester Large
Raven	Yellow Polyester Medium
Seedling	Red Cotton Large
SeedlingPlus	Red Cotton Large with hood
Younger	Blue Cotton with patch
MultiStripe	Red Yellow Blue Cotton Small/Medium/Large

8.19 数学関数

このセクションでは、数理定数とブール値の関数について説明します。これらの関数はパラメータを持ちませんが、括弧は必要です。

すべての関数は、データロードスクリプトおよびチャートの数式の両方で使用できます。

e
この関数は、自然対数の底 **e** (2.71828...) を返します。

```
e( )
```

false
この関数は、数式の中で論理偽として使用できるテキスト値 'False' と数値 0 のデュアル値を返します。

```
false( )
```

pi
この関数は π 値 (3.14159...) を返します。

```
pi( )
```

rand
この関数は、0 ~ 1 間の乱数を返します。この関数は、サンプル データの作成に使用できます。

```
rand( )
```

この例のスクリプトは、ランダムに選択された大文字の文字 (65 から 91 (65+26) までの範囲の文字) が含まれた 1000 件のレコードを持つテーブルを作成します。

```
Load
     Chr( Floor(rand() * 26) + 65) as UCaseChar,
     RecNo() as ID
     Autogenerate 1000;
```

true
この関数は、数式の中で論理真として使用できるテキスト値 'True' と数値 -1 のデュアル値を返します。

```
true( )
```

8.20 NULL 関数

このセクションでは、NULL 値を返したり検出する関数について説明します。

すべての関数は、データ ロード スクリプトおよびチャートの数式の両方で使用できます。

NULL 関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

EmptyIsNull
EmptyIsNull 関数は、空の文字列を NULL に変換します。したがって、パラメータが空の文字列の場合は NULL を返し、そうでない場合はパラメータを返します。

```
EmptyIsNull (expr )
```

IsNull
IsNull 関数は、数式の値が NULL かどうかを検定します。NULL の場合は -1 (True)、NULL でない場合は 0 (False) を返します。

```
IsNull (expr )
```

Null
Null 関数は、NULL 値を返します。

```
NULL( )
```

EmptyIsNull

EmptyIsNull 関数は、空の文字列を NULL に変換します。したがって、パラメータが空の文字列の場合は NULL を返し、そうでない場合はパラメータを返します。

構文:

```
EmptyIsNull(exp )
```

例と結果:

スクリプトの例

例	結果
`EmptyIsNull(AdditionalComments)`	この数式は、空の文字列ではなく [*AdditionalComments*] 項目の空の文字列値を null として返します。空でない文字列と数値が返されます。
`EmptyIsNull(PurgeChar (PhoneNumber, ' -()'))`	この数式は、[*PhoneNumber*] 項目からダッシュ、スペース、括弧を取り除きます。文字が残っていない場合、EmptyIsNull 関数は空の文字列を null として返します。空の電話番号は、電話番号がないことと同じです。

IsNull

IsNull 関数は、数式の値が NULL かどうかを検定します。NULL の場合は -1 (True)、NULL でない場合は 0 (False) を返します。

構文:

```
IsNull(expr )
```

*長さ 0 の文字列は NULL とみなされず、**IsNull** は False を返します。*

データ ロード スクリプト

この例では、4 つの行を持つインライン テーブルがロードされ、その最初の 3 行の Value 列は、空になっているか、- または 'NULL' になっています。真ん中の先行する **LOAD** で **Null** 関数を使って、これらの値を実際の NULL 値表現に変換します。

1 つ目の先行する**LOAD** では、値が NULL かどうかをチェックする項目を**IsNull** 関数を使って追加しています。

```
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];
```

この結果、テーブルは次のようになります。ValueNullConv 列では、NULL 値が - で表されています。

Resulting table

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T
2	-	-	T
3	Value	Value	F

NULL

Null 関数は、NULL 値を返します。

構文:

```
Null( )
```

データ ロード スクリプト

この例では、4 つの行を持つインライン テーブルがロードされ、その最初の 3 行の Value 列は、空になっているか、- または 'NULL' になっています。これらの値を実際の NULL 値表現に変換します。

真ん中の先行する**LOAD** は、**Null** 関数を使用して変換を実行します。

1 つ目の先行する**LOAD** は、値が NULL かどうかをチェックする項目を追加しています。この例では、この項目は、見る人にわかりやすくするためにのみ追加されています。

```
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];
```

この結果、テーブルは次のようになります。ValueNullConv 列では、NULL 値が - で表されています。

Resulting table

ID	Value	ValueNullConv	IsItNull
0		-	T
1	NULL	-	T
2	-	-	T
3	Value	Value	F

8.21 範囲関数

範囲関数は、値の配列を取得し、結果として1つの値を生成する関数です。すべての範囲関数は、データロードスクリプトおよびチャート式の両方で使用できます。

たとえば、ビジュアライゼーションでは、範囲関数でレコード間配列から1つの値を計算できます。データロードスクリプトでは、範囲関数で内部テーブルの値の配列から1つの値を計算できます。

*範囲関数は、サポート対象外となった旧式の一般的な数値関数である **numsum**、**numavg**、**numcount**、**nummin**、**nummax** の代替として導入された新たな関数です。*

基本的な範囲関数

RangeMax

RangeMax() は、数式または項目に含まれる最大値を返します。

```
RangeMax (first_expr[, Expression])
```

RangeMaxString

RangeMaxString() は、数式または項目における、テキストソート順の最後の値を返します。

```
RangeMaxString (first_expr[, Expression])
```

RangeMin

RangeMin() は、数式または項目に含まれる最小値を返します。

```
RangeMin (first_expr[, Expression])
```

RangeMinString

RangeMinString() は、数式または項目における、テキストソート順の最初の値を返します。

```
RangeMinString (first_expr[, Expression])
```

RangeMode
RangeMode() は、数値または項目において、最も頻繁に登場する値 (モード値) を返します。

```
RangeMode (first_expr[, Expression])
```

RangeOnly
RangeOnly()は、数式が1つの一意の値を評価する場合に値を返すデュアル関数です。それ以外の場合は **NULL** が返されます。

```
RangeOnly (first_expr[, Expression])
```

RangeSum
RangeSum() は値の範囲の合計を返します。数値以外の値はすべて0として扱われます。

```
RangeSum (first_expr[, Expression])
```

カウンタ範囲関数

RangeCount
RangeCount() は、指定した数式または項目に含まれる値の数を、テキストおよび数字の両方で返します。

```
RangeCount (first_expr[, Expression])
```

RangeMissingCount
RangeMissingCount() は、数式または項目に含まれる、数値以外の値 (NULL を含む) を返します。

```
RangeMissingCount (first_expr[, Expression])
```

RangeNullCount
RangeNullCount() は、数式または項目に含まれるNULL値の数を返します。

```
RangeNullCount (first_expr[, Expression])
```

RangeNumericCount
RangeNumericCount() は、数値または項目に含まれる、数値の数を返します。

```
RangeNumericCount (first_expr[, Expression])
```

RangeTextCount
RangeTextCount() は、数値または項目に含まれる、テキスト値の数を返します。

```
RangeTextCount (first_expr[, Expression])
```

統計的範囲関数

RangeAvg
RangeAvg()は、範囲の平均を返します。関数には、値の範囲または数式のいずれかを入力できます。

```
RangeAvg (first_expr[, Expression])
```

RangeCorrel
RangeCorrel() は、2 つのデータセットの相関係数を返します。相関係数はデータセット間の関係を表すメジャーです。

```
RangeCorrel (x_values , y_values[, Expression])
```

RangeFractile
RangeFractile() は、数値の範囲における n 番目の **fractile** (変位値) に相当する値を返します。

```
RangeFractile (fractile, first_expr[ ,Expression])
```

RangeKurtosis
RangeKurtosis() は、数値の範囲の尖度に相当する値を返します。

```
RangeKurtosis (first_expr[, Expression])
```

RangeSkew
RangeSkew() は、数値の範囲の歪度に相当する値を返します。

```
RangeSkew (first_expr[, Expression])
```

RangeStdev
RangeStdev() は、数値の範囲の標準偏差を返します。

```
RangeStdev (expr1[, Expression])
```

財務範囲関数

RangeIRR
RangeIRR() は、入力値で表される一連のキャッシュフローの内部収益率を返します。

```
RangeIRR (value[, value][, Expression])
```

RangeNPV
RangeNPV() は、割引率および一連の将来の定期的な支払 (負の値) および収入 (正の値) に基づき、投資の正味現在価値を返します。結果は、**money** のデフォルトの数値書式で返されます。

```
RangeNPV (discount_rate, value[, value][, Expression])
```

RangeXIRR
RangeXIRR() は、キャッシュフロー明細表に対する内部収益率 (年次) を返します。キャッシュフロー明細表は、定期的である必要はありません。一連の定期的キャッシュフローに対する内部利益率の計算には、**RangeIRR** 関数を使用します。

```
RangeXIRR (values, dates[, Expression])
```

RangeXNPV
RangeXNPV() は、**pmt** と **date** の数値ペアで表されるキャッシュフロー計算書の値 (不定期の場合もあります) の正味現在価値を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

```
RangeXNPV (discount_rate, values, dates[, Expression])
```

参照先:

- *レコード間関数 (page 1249)*

RangeAvg

RangeAvg()は、範囲の平均を返します。関数には、値の範囲または数式のいずれかを入力できます。

構文:

```
RangeAvg(first_expr[, Expression])
```

戻り値データ型: 数値

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

数値が見つからない場合は、NULL が返されます。

例と結果:

スクリプトの例

例	結果
`RangeAvg (1,2,4)`	2.33333333 を返します
`RangeAvg (1,'xyz')`	1 を返します
`RangeAvg (null( ), 'abc')`	NULL を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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
];
```

結果テーブルには、テーブルの各レコードに対するMyRangeAvgの戻り値が表示されます。

結果のテーブル

RangeID	MyRangeAvg
1	7
2	4
3	6
4	12.666
5	6.333
6	5

数式を用いた例:

```
RangeAvg (Above(MyField),0,3))
```

現在の行とその上の2行で計算された、**MyField**の3つの値域の結果のスライド平均が返されます。3番目の引数に3を指定すると、**Above()**関数は、上に十分な行のある場所に3つの値を返し、**RangeAvg()**関数への入力として取得されます。

例で使用されているデータ:

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeAvg (Above (MyField,0,3))	Comments
10	10	これが先頭の行のため、範囲は1つの値のみで構成されます。
2	6	この行の上に1つの行しかないため、範囲は10,2.
8	6.6666666667	RangeAvg(10,2,8) に相当。
18	9.333333333	-
5	10. 333333333	-
9	10.6666666667	-

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
```

```
5
9
] ;
```

参照先:

- *Avg - チャート関数 (page 400)*
- *Count - チャート関数 (page 352)*

RangeCorrel

RangeCorrel() は、2 つのデータセットの相関係数を返します。相関係数はデータセット間の関係を表すメジャーです。

構文:

```
RangeCorrel(x_value , y_value[, Expression])
```

戻り値データ型: 数値

データ系列は、(x,y) ペアとして入力してください。例えば、array1 (array1 = 2,6,9) とarray2 (array2 = 3,8,4) の 2 種類のデータ系列を評価する場合、`RangeCorrel (2,3,6,8,9,4)` と入力すると、0.269 が返されます。

引数:

引数

引数	説明
x-value, y-value	それぞれの値は、3 番目のオプション パラメータを持つレコード間関数によって返される単一の値または値域を表します。それぞれの値や値域は、**x-value** または **y-values** の値域に対応していなければなりません。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

この関数の計算には、少なくとも 2 組の座標が必要です。

テキスト値および NULL 値、欠損値は NULL を返します。

例と結果:

関数の例

例	結果
`RangeCorrel (2,3,6,8,9,4,8,5)`	0.2492 を返します。この関数は、スクリプトにロードしたり、数式エディタのビジュアライゼーションに追加したりすることができます。

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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 '|');
```

軸とメジャー RangeCorrel(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6))としてID1を含むテーブルで、**RangeCorrel()** 関数は、各 ID1 値について、6 つの x,y ペアの範囲にわたる **Correl** の値を見つけます。

結果のテーブル

ID1	MyRangeCorrel
01	-0.9517
02	-0.5209
03	-0.5209
04	-0.1599

```
XY:
LOAD recno() as RangeID, * Inline [
X|Y
2|3
6|8
9|4
8|5
](delimiter is '|');
```

RangeID が軸とメジャーのテーブル: RangeCorrel(Below(X,0,4,BelowY,0,4)) で、**RangeCorrel()** 関数は **Below()** 関数の結果を使用します。この関数は、3 つめの引数 (count) が 4 と設定されているため、ロードされたテーブル XY から 4 つの x-y 値を生成します。

結果のテーブル

RangeID	MyRangeCorrel2
01	0.2492
02	-0.9959
03	-1.0000
04	-

RangeID 01 の値は、手入力した RangeCorrel(2,3,6,8,9,4,8,5) と同じです。RangeID のその他の値として Below() 関数が生成する一連の値は、(6,8,9,4,8,5)、(9,4,8,5)、および (8,5) となり、最後の値は NULL を返します。

参照先:

- *Correl - チャート関数 (page 404)*

RangeCount

RangeCount() は、指定した数式または項目に含まれる値の数を、テキストおよび数字の両方で返します。

構文:

```
RangeCount(first_expr[, Expression])
```

戻り値データ型: 整数

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	データを含む数式または項目をカウントします。
Expression	データの範囲を含むオプションの数式または項目をカウントします。

制限事項:

NULL 値はカウントされません。

例と結果:

関数の例

例	結果
`RangeCount (1,2,4)`	3 を返します

例	結果
RangeCount (2,'xyz')	2 を返します
RangeCount (null( ))	0 を返します
RangeCount (2,'xyz', null())	2 を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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
];
```

結果テーブルには、テーブルの各レコードに対する MyRangeCount の戻り値が表示されます。

結果テーブル

RangeID	MyRangeCount
1	3
2	3
3	3
4	3
5	3
6	3

数式を用いた例:

```
RangeCount (Above(MyField,1,3))
```

MyField の 3 つの結果に含まれる値の数が返されます。**Above()** 関数の 1 番目の引数を 1 として指定し、2 番目の引数を 3 として指定すると、十分な行があれば現在行のすぐ上の 3 項目から値が返されます。これらの値は、**RangeCount()** 関数への入力として取得されます。

例で使用されているデータ:

サンプル データ

MyField	RangeCount(Above(MyField,1,3))
10	0

MyField	RangeCount(Above(MyField,1,3))
2	1
8	2
18	3
5	3
9	3

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

参照先:

- *Count - チャート関数 (page 352)*

RangeFractile

RangeFractile() は、数値の範囲におけるn番目の **fractile** (変位値) に相当する値を返します。

RangeFractile() は、分位数の計算時に最も近いランク間で線形補間を使用します。

構文:

```
RangeFractile(fractile, first_expr[, Expression])
```

戻り値データ型: 数値

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
fractile	計算対象となる分位数 (変位値) に相当する値 (0 ~ 1 の範囲内)。
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

例と結果:

関数の例

例	結果
`RangeFractile (0.24,1,2,4,6)`	1.72 を返します
`RangeFractile(0.5,1,2,3,4,6)`	3 を返します
`RangeFractile (0.5,1,2,5,6)`	3.5 を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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
];
```

結果テーブルには、テーブルの各レコードに対する MyRangeFrac の戻り値が表示されます。

結果のテーブル

RangeID	MyRangeFrac
1	6
2	3
3	8
4	11
5	5
6	4

数式を用いた例:

```
RangeFractile (0.5, Above(Sum(MyField),0,3))
```

この例では、レコード間関数 **Above()** にオプションで offset および count 引数が含まれています。これにより、範囲関数への入力として使用できる結果の範囲が生成されます。この場合、`Above(Sum(MyField),0,3)` は現在行およびその上の 2 行に `MyField` の値を返します。これらの値によって、**RangeFractile()** 関数への入力が

指定されます。そのため、次のテーブルの最終行では RangeFractile(0.5, 3,4,6) と等しくなります。つまり、一連の3、4、6の0.5番目の分位数を計算します。次の表の最初の2行では、範囲内の値の数は減少し、現在行の上に行はありません。同様の結果が、他のレコード間関数でも生成されます。

サンプル データ

MyField	RangeFractile(0.5, Above(Sum(MyField),0,3))
1	1
2	1.5
3	2
4	3
5	4
6	5

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
1
2
3
4
5
6
] ;
```

参照先:

- *Above - チャート関数 (page 1252)*
- *Fractile - チャート関数 (page 408)*

RangeIRR

RangeIRR() は、入力値で表される一連のキャッシュフローの内部収益率を返します。

内部収益率は、定期的に発生する支払い (負の値) と収入 (正の値) からなる投資の利率です。

この関数は、内部利益率 (IRR) を計算するためにニュートン法の簡素化されたバージョンを使用します。

構文:

```
RangeIRR(value[, value][, Expression])
```

戻り値データ型: 数値

引数

引数	説明
value	3 番目のオプション パラメータを持つレコード間関数によって返される単一値または値域です。この関数の計算には、少なくとも 1 つの正の値と1 つの負の値が必要です。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

テキスト値、NULL 値、欠損値は無視されます。

テーブルの例

例	結果
`RangeIRR(-70000,12000,15000,18000,21000,26000)`	0.0866 を返す

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。
`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 '|');`
	結果テーブルには、テーブルの各レコードに対する RangeIRR の戻り値が表示されます。
	RangeID: 1 → RangeIRR: 0.0639
	2 → 0.8708
	3 → -
	4 → 5.8419
	5 → 0.9318
	6 → -0.2566

参照先:

- *レコード間関数 (page 1249)*

RangeKurtosis

RangeKurtosis() は、数値の範囲の尖度に相当する値を返します。

構文:

```
RangeKurtosis(first_expr[, Expression])
```

戻り値データ型: 数値

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

数値が見つからない場合は、NULL が返されます。

例と結果:

関数の例

例	結果
RangeKurtosis (1,2,4,7)	-0.28571428571429 を返します

参照先:

- *Kurtosis - チャート関数 (page 416)*

RangeMax

RangeMax() は、数式または項目に含まれる最大値を返します。

構文:

```
RangeMax(first_expr[, Expression])
```

戻り値データ型: 数値

引数:

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

数値が見つからない場合は、NULL が返されます。

例と結果:

関数の例

例	結果
`RangeMax (1,2,4)`	4 を返します
`RangeMax (1,'xyz')`	1 を返します
`RangeMax (null( ), 'abc')`	NULL を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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
];
```

結果テーブルには、テーブルの各レコードに対する MyRangeMax の戻り値が表示されます。

結果のテーブル

RangeID	MyRangeMax
1	10
2	7

RangeID	MyRangeMax
3	8
4	18
5	9
6	9

数式を用いた例:

```
RangeMax (Above(MyField,0,3))
```

現在の行とその上の 2 行で計算された、**MyField** の 3 つの値域の結果の最大値が返されます。3 番目の引数に 3 を指定すると、**Above()** 関数は、上に十分な行のある場所に 3 つの値を返し、**RangeMax()** 関数への入力として取得されます。

例で使用されているデータ:

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeMax (Above(Sum(MyField),1,3))
10	10
2	10
8	10
18	18
5	18
9	18

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

RangeMaxString

RangeMaxString() は、数式または項目における、テキスト ソート順の最後の値を返します。

構文:

```
RangeMaxString(first_expr[, Expression])
```

戻り値データ型: string

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

例と結果:

関数の例

例	結果
`RangeMaxString (1,2,4)`	4 を返します
`RangeMaxString ('xyz','abc')`	'xyz' を返します
`RangeMaxString (5,'abc')`	'abc' を返します
`RangeMaxString (null( ))`	NULL を返します

数式を用いた例:

```
RangeMaxString (Above(MaxString(MyField),0,3))
```

現在の行とその上の 2 つの行で評価された **MaxString(MyField)** 関数の 3 つの結果のうち、(テキスト順で) 最後の値が返されます。

例で使用されているデータ:

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeMaxString(Above(MaxString(MyField),0,3))
10	10
abc	abc
8	abc
def	def

MyField	RangeMaxString(Above(MaxString(MyField),0,3))
xyz	xyz
9	xyz

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
'def'
'xyz'
9
] ;
```

参照先:

- *MaxString - チャート関数 (page 534)*

RangeMin

RangeMin() は、数式または項目に含まれる最小値を返します。

構文:

RangeMin(first_expr[, Expression]**)**

戻り値データ型: 数値

引数:

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

数値が見つからない場合は、NULL が返されます。

例と結果:

関数の例

例	結果
RangeMin (1,2,4)	1 を返します

例	結果
RangeMin (1,'xyz')	1 を返します
RangeMin (null( ), 'abc')	NULL を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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
];
```

結果テーブルには、テーブルの各レコードに対する MyRangeMin の戻り値が表示されます。

結果のテーブル

RangeID	MyRangeMin
1	5
2	2
3	2
4	9
5	5
6	2

数式を用いた例:

```
RangeMin (Above(MyField,0,3)
```

現在の行とその上の 2 行で計算された、**MyField** の 3 つの値域の結果の最小値が返されます。3 番目の引数に 3 を指定すると、**Above()** 関数は、上に十分な行のある場所に 3 つの値を返し、**RangeMin()** 関数への入力として取得されます。

例で使用されているデータ:

サンプル データ

MyField	RangeMin(Above(MyField,0,3))
10	10

MyField	RangeMin(Above(MyField,0,3))
2	2
8	2
18	2
5	5
9	5

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

参照先:

- *Min - チャート関数 (page 339)*

RangeMinString

RangeMinString() は、数式または項目における、テキストソート順の最初の値を返します。

構文:

```
RangeMinString(first_expr[, Expression])
```

戻り値データ型: string

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

例と結果:

関数の例

例	結果
RangeMinString (1,2,4)	1 を返します
RangeMinString ('xyz','abc')	'abc' を返します
RangeMinString (5,'abc')	5 を返します
RangeMinString (null( ))	NULL を返します

数式を用いた例:

```
RangeMinString (Above(MinString(MyField),0,3))
```

現在の行とその上の 2 つの行で評価された **MinString(MyField)** 関数の 3 つの結果のうち、(テキスト順で) 最初の値が返されます。

例で使用されているデータ:

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeMinString(Above(MinString(MyField),0,3))
10	10
abc	10
8	8
def	8
xyz	8
9	9

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
'def'
'xyz'
9
] ;
```

参照先:

- *MinString - チャート関数 (page 537)*

RangeMissingCount

RangeMissingCount() は、数式または項目に含まれる、数値以外の値 (NULL を含む) を返します。

構文:

```
RangeMissingCount(first_expr[, Expression])
```

戻り値データ型: 整数

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	データを含む数式または項目をカウントします。
Expression	データの範囲を含むオプションの数式または項目をカウントします。

例と結果:

関数の例

例	結果
`RangeMissingCount (1,2,4)`	0 を返します
`RangeMissingCount (5,'abc')`	1 を返します
`RangeMissingCount (null( ))`	1 を返します

数式を用いた例:

```
RangeMissingCount (Above(MinString(MyField),0,3))
```

現在の行とその上の 2 つの行で評価された **MinString(MyField)** 関数の 3 つの結果に含まれる、数値以外の値の数が返されます。

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeMissingCount (Above(MinString (MyField),0,3))	Explanation
10	2	2 が返されます (上に他の行がないため、3 つの値のうち、2 つが欠けていると認識されるため)
abc	2	2 が返されます (上に行が 1 つしかなく、現在の行 ('abc') が数値でないため)
8	1	1 が返されます (3 つの行のうち、1 つの行に数値以外の値 ('abc') が含まれているため)
def	2	2 が返されます (3 つの行のうち、2 つの行に数値以外の値 ('def' と'abc') が含まれているため)
xyz	2	2 が返されます (3 つの行のうち、2 つの行に数値以外の値 (' xyz' と'def') が含まれているため)
9	2	2 が返されます (3 つの行のうち、2 つの行に数値以外の値 (' xyz' と'def') が含まれているため)

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
'def'
'xyz'
9
] ;
```

参照先:

- *MissingCount - チャート関数 (page 356)*

RangeMode

RangeMode() は、数値または項目において、最も頻繁に登場する値 (モード値) を返します。

構文:

```
RangeMode(first_expr {, Expression})
```

戻り値データ型: 数値

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

同じ最大頻度を持つ値が複数ある場合は、NULL が返されます。

例と結果:

関数の例

例	結果
`RangeMode (1,2,9,2,4)`	2 を返します
`RangeMode ('a',4,'a',4)`	NULL を返します
`RangeMode (null( ))`	NULL を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
RangeTab3:
LOAD recno() as RangeID, RangeMode(Field1,Field2,Field3) as MyRangeMode INLINE [
Field1, Field2, Field3
10,5,6
2,3,7
8,2,8
18,11,9
5,5,9
9,4,2
];
```

結果テーブルには、テーブルの各レコードに対する **MyRangeMode** の戻り値が表示されます。

結果テーブル

RangeID	MyRangMode
1	-
2	-
3	8
4	-
5	5
6	-

数式を用いた例:

```
RangeMode (Above(MyField,0,3))
```

現在の行とその上の 2 つの行で評価された **MyField** の 3 つの結果のうち、最も頻繁に出現する値が返されます。3 番目の引数に 3 を指定すると、**Above()** 関数は、上に十分な行のある場所に 3 つの値を返し、**RangeMode()** 関数への入力として取得されます。

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeMode(Above(MyField,0,3))
10	10 が返されます (上に他の行がないため、この単一値が最も頻繁に登場するため)
2	-
8	-
18	-
5	-
9	-

参照先:

- *Mode - チャート関数 (page 342)*

RangeNPV

RangeNPV() は、割引率および一連の将来の定期的な支払 (負の値) および収入 (正の値) に基づき、投資の正味現在価値を返します。結果は、**money** のデフォルトの数値書式で返されます。

必ずしも定期的でないキャッシュフローについては、*RangeXNPV (page 1344)* を参照してください。

構文:

```
RangeNPV(discount_rate, value[,value][, Expression])
```

戻り値データ型: 数値

引数

引数	説明
discount_rate	期間あたりの利率。
value	各期末に発生する支払または収入。それぞれの値は、3 番目のオプション パラメータを持つレコード間関数によって返される、単一値または値域を表している可能性があります。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

テキスト値、NULL 値、欠損値は無視されます。

例	結果
`RangeNPV(0.1,-10000,3000,4200,6800)`	1188.44 を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。
`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 '|');`
	結果テーブルには、テーブルの各レコードに対する RangeNPV の戻り値が表示されます。
	RangeID RangeNPV
	1 $-49.13
	2 $777.78
	3 $98.77
	4 $25.51
	5 $250.83
	6 $20.40

参照先:

レコード間関数 (page 1249)

RangeNullCount

RangeNullCount() は、数式または項目に含まれるNULL値の数を返します。

構文:

```
RangeNullCount(first_expr [, Expression])
```

戻り値データ型: integer

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

例と結果:

関数の例

例	結果
`RangeNullCount (1,2,4)`	0 を返します
`RangeNullCount (5,'abc')`	0 を返します
`RangeNullCount (null( ), null( ))`	2 を返します

数式を用いた例:

```
RangeNullCount (Above(Sum(MyField),0,3))
```

現在の行とその上の 2 つの行で評価された **Sum(MyField)** 関数の結果に含まれるNULL値の数が返されます。

*以下の例では、**MyField** をコピーしてもNULL値は生じません。*

サンプル データ

MyField	RangeNullCount(Above(Sum(MyField),0,3))
10	2 が返されます (上に他の行がないため、3 つの値のうち、2 つが欠けている (=NULL) と認識されるため)

MyField	RangeNullCount(Above(Sum(MyField),0,3))
'abc'	1 が返されます (上に行が 1 つしかないため、3 つの値のうち、1 つが欠けている (=NULL) と認識されるため)
8	0 が返されます (3 つの行が NULL 値になっていないため)

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
] ;
```

参照先:

- *NullCount - チャート関数 (page 358)*

RangeNumericCount

RangeNumericCount() は、数値または項目に含まれる、数値の数を返します。

構文:

```
RangeNumericCount(first_expr[, Expression])
```

戻り値データ型: integer

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

例と結果:

関数の例

例	結果
RangeNumericCount (1,2,4)	3 を返します
RangeNumericCount (5,'abc')	1 を返します
RangeNumericCount (null( ))	0 を返します

数式を用いた例:

```
RangeNumericCount (Above(MaxString(MyField),0,3))
```

現在の行とその上の 2 つの行で評価された **MaxString(MyField)** 関数の 3 つの結果に存在する数値の数が返されます。

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeNumericCount(Above(MaxString(MyField),0,3))
10	1
abc	1
8	2
def	1
xyz	1
9	1

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
def
xyz
9
] ;
```

参照先:

- *NumericCount - チャート関数 (page 361)*

RangeOnly

RangeOnly()は、数式が 1 つの一意の値を評価する場合に値を返すデュアル関数です。それ以外の場合は **NULL** が返されます。

構文:

```
RangeOnly(first_expr[, Expression])
```

戻り値データ型: dual

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

例と結果:

例	結果
`RangeOnly (1,2,4)`	NULL を返します
`RangeOnly (5,'abc')`	NULL を返します
`RangeOnly (null( ), 'abc')`	'abc' を返します
`RangeOnly(10,10,10)`	10 を返します

参照先:

- *Only - チャート関数 (page 345)*

RangeSkew

RangeSkew() は、数値の範囲の歪度に相当する値を返します。

構文:

```
RangeSkew(first_expr[, Expression])
```

戻り値データ型: 数値

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

数値が見つからない場合は、NULL が返されます。

例と結果:

関数の例

例	結果
rangeskew (1,2,4)	0.93521952958283 を返します
rangeskew (above (SalesValue,0,3))	現在の行とその上の 2 行で計算された数式 above() 関数から返された 3 つの値域のスライド歪度を返します。

例で使用されているデータ:

サンプル データ

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
] ;
```

参照先:

- *Skew - チャート関数 (page 447)*

RangeStdev

RangeStdev() は、数値の範囲の標準偏差を返します。

構文:

```
RangeStdev(first_expr[, Expression])
```

戻り値データ型: 数値

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

数値が見つからない場合は、NULL が返されます。

例と結果:

関数の例

例	結果
`RangeStdev (1,2,4)`	1.5275252316519 を返します
`RangeStdev (null( ))`	NULL を返します
`RangeStdev (above (SalesValue),0,3))`	現在の行とその上の 2 つの行で計算された above() 関数から返された 3 つの値域のスライド標準偏差を返します。

例で使用されているデータ:

サンプル データ

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
] ;
```

参照先:

- *Stdev - チャート関数 (page 450)*

RangeSum

RangeSum() は値の範囲の合計を返します。数値以外の値はすべて 0 として扱われます。

構文:

```
RangeSum(first_expr[, Expression])
```

戻り値データ型: 数値

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

制限事項:

RangeSum 関数は数値以外の値をすべて 0 として扱います。

例と結果:

例

例	結果
`RangeSum (1,2,4)`	7 を返します
`RangeSum (5,'abc')`	5 を返します
`RangeSum (null( ))`	0 を返します

アプリにスクリプト例を追加して実行します。結果を表示するには、結果列に含まれている項目をアプリのシートに追加します。

```
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
];
```

結果テーブルには、テーブルの各レコードに対するMyRangeSumの戻り値が表示されます。

結果のテーブル

RangeID	MyRangeSum
1	21
2	12
3	18
4	38
5	19
6	15

数式を用いた例:

```
RangeSum (Above(MyField,0,3))
```

現在の行とその上の2つの行から**MyField)** 関数の3つの値の合計が返されます。3番目の引数に3を指定すると、**Above()** 関数は、上に十分な行のある場所に3つの値を返し、**RangeSum()** 関数への入力として取得されます。

例で使用されているデータ:

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

サンプル データ

MyField	RangeSum(Above(MyField,0,3))
10	10
2	12
8	20
18	28
5	31
9	32

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
2
8
18
5
9
] ;
```

参照先:

- *Sum - チャート関数 (page 348)*
- *Above - チャート関数 (page 1252)*

RangeTextCount

RangeTextCount() は、数値または項目に含まれる、テキスト値の数を返します。

構文:

```
RangeTextCount(first_expr[, Expression])
```

戻り値データ型: integer

引数:

この関数の引数には、値のリストをそれ自体の中で返すレコード間関数が含まれます。

引数

引数	説明
first_expr	メジャーの対象となるデータが含まれている数式または項目。
Expression	メジャーの対象となるデータ範囲が含まれている任意の数式または項目。

例と結果:

関数の例

例	結果
RangeTextCount (1,2,4)	0 を返します
RangeTextCount (5,'abc')	1 を返します
RangeTextCount (null( ))	0 を返します

数式を用いた例:

```
RangeTextCount (Above(MaxString(MyField),0,3))
```

現在の行とその上の 2 つの行で評価された **MaxString(MyField)** 関数の 3 つの結果に存在するテキスト値の数が返されます。

例で使用されているデータ:

*このような場合、集計エラーを防ぐため **MyField** によるソートは無効にしておきます。*

データの例

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

例で使用されているデータ:

```
RangeTab:
LOAD * INLINE [
MyField
10
'abc'
8
null()
'xyz'
9
] ;
```

参照先:

- *TextCount - チャート関数 (page 364)*

RangeXIRR

RangeXIRR() は、キャッシュフロー明細表に対する内部収益率 (年次) を返します。キャッシュフロー明細表は、定期的である必要はありません。一連の定期的キャッシュフローに対する内部利益率の計算には、**RangeIRR** 関数を使用します。

Qlik の XIRR 機能 (**XIRR()** および **RangeXIRR()** 関数) は、次の方程式を使用して `Rate` 値を解き、正しい XIRR 値を決定します。

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

この方程式は、ニュートン法の簡素化されたバージョンを使用して解かれます。

構文:

```
RangeXIRR(value, date{, value, date})
```

戻り値データ型: 数値

引数

引数	説明
value	支払予定日の日付に対応する単一または一連のキャッシュフローです。一連の値は、少なくとも 1 つの正の値と1 つの負の値を含む必要があります。
date	キャッシュフローの支払いに対応する支払日または支払予定日です。

この関数を使用する場合は、次の制限が適用されます。

- テキスト値、NULL 値、欠損値は無視されます。
- すべての支払いは、年 365 日の日割り計算で割り引かれます。
- この関数には、少なくとも 1 つの有効なマイナスの支払いと1 つの有効なプラスの支払い (対応する有効日付付き) が必要です。これらの支払いが入力されない場合、NULL 値が返されます。

次のトピックは、この関数を使用するのに役立つかもしれません。

- *RangeXNPV (page 1344)*: この関数を使用すると、キャッシュフロー明細表に対する正味現在価値を計算します。キャッシュフロー明細表は、定期的である必要はありません。
- *XIRR (page 379)*: **XIRR()** 関数は、キャッシュフロー明細表に対する集計済み内部収益率 (年次) を計算します (キャッシュフロー明細表は、定期的である必要はありません)。

Qlik Sense Client-Managed のバージョンが異なると、この関数で使用される基になるアルゴリズムが異なります。アルゴリズムの最近のアップデートについて詳しくは、サポート記事「XIRR 関数の修正とアップデート」を参照してください。

例と結果:

例と結果

例	結果
`RangeXIRR(-2500,'2008-01-01',2750,'2008-09-01')`	0.1532 を返します

参照先:

- *RangeIRR (page 1318)*
- *RangeXNPV (page 1344)*
- *XIRR (page 379)*
- XIRR 関数の修正およびアップデート

RangeXNPV

RangeXNPV() は、**pmt** と**date** の数値ペアで表されるキャッシュフロー計算書の値 (不定期の場合もあります) の正味現在価値を返します。すべての支払いは、年 365 日の日割り計算で割り引かれます。

構文:

```
RangeXNPV(discount_rate, value, date{, value, date})
```

戻り値データ型: 数値

引数

引数	説明
discount_rate	**discount_rate** は、支払いが割引されるべき年率です。
value	支払予定日の日付に対応する単一または一連のキャッシュフローです。それぞれの値は、3 番目のオプション パラメータを持つレコード間関数によって返される、単一値または値域を表している可能性があります。一連の値は、少なくとも 1 つの正の値と1 つの負の値を含む必要があります。
date	キャッシュフローの支払いに対応する支払日または支払予定日です。

この関数を使用する場合は、次の制限が適用されます。

- テキスト値、NULL 値、欠損値は無視されます。
- すべての支払いは、年 365 日の日割り計算で割り引かれます。

例 - スクリプト

ロードスクリプトと結果

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- RangeTab3 と呼ばれるテーブルに含まれる財務データ。
- **RangeXNPV()** 関数を使用すると、正味現行値が計算されます。

ロードスクリプト

```
RangeTab3:
LOAD *,
recno() as RangeID,
RangeXNPV(DiscountRate,Value1,Date1,Value2,Date2) as RangeXNPV;
LOAD * INLINE [
DiscountRate|Value1|Date1|Value2|Date2
0.1|-100|2021-01-01|100|2022-01-01|
0.1|-100|2021-01-01|110|2022-01-01|
0.1|-100|2021-01-01|125|2022-01-01|
] (delimiter is '|');
```

結果

データをロードしてシートを開きます。新しいテーブルを作成し、これらの項目を軸として追加します:

- RangeID
- RangeXNPV

結果テーブル

RangeID	RangeXNPV
1	-$9.09
2	-$0.00
3	$13.64

例 - チャートの数式

ロードスクリプトとチャートの数式

概要

データロードエディターを開き、以下のロードスクリプトを新しいタブに追加します。

ロードスクリプトには次が含まれています。

- RangeTab3 と呼ばれるテーブルに含まれる財務データ。
- **RangeXNPV()** 関数を使用すると、正味現行値が計算されます。

ロードスクリプト

```
RangeTab3:
LOAD *,
recno() as RangeID,
RangeXNPV(DiscountRate,Value1,Date1,Value2,Date2) as RangeXNPV;
LOAD * INLINE [
DiscountRate|Value1|Date1|Value2|Date2
0.1|-100|2021-01-01|100|2022-01-01|
0.1|-100|2021-01-01|110|2022-01-01|
0.1|-100|2021-01-01|125|2022-01-01|
] (delimiter is '|');
```

結果

次の手順を実行します。

データをロードしてシートを開きます。新しいテーブルを作成し、メジャーとして次の計算を追加します。

```
=RangeXNPV(0.1, -2500,'2008-01-01',2750,'2008-09-01')
```

結果テーブル

=XIRR(Payments, Date)
$80.25

参照先:

XNPV (page 385)

8.22 関係関数

これは、すでに集計されている数値を利用して、グラフ内の個々の次元の数値の特性を計算する関数群です。

関数は、関数の出力がデータ点自体の値だけでなく、その値と他のデータ点との関係にも依存するという意味で関係的です。例えば、ランクは他の軸の値がなければ計算できません。

これらの関数は、チャート式でのみ使用できます。ランクは、ロードスクリプトで使用できません。

軸は比較に必要な他のデータポイントを定義するため、チャートで必要となります。よって、関係関数は軸チャートにおいては有意ではありません (KPI チャートなど)。

ランキング関数

これらの関数が使用される場合、0 値を隠す機能は自動的に無効になります。NULL 値は無視されます。

Rank

Rank() は、数式におけるチャートの行を評価し、それぞれの行に対して、数式で評価される軸の値の相対位置を示します。この関数は数式の評価時に、結果を現在の列セグメントに含まれるその他の行の結果と比較して、セグメント内の現在の行の順位付けを返します。

```
Rank - チャート関数([TOTAL [<fld {, fld}>]] expr[, mode[, fmt]])
```

HRank

HRank() は expression を評価し、結果をピボットテーブルの現在の行セグメント内のその他の列の結果と比較します。この関数は、セグメント内の現在の例のランキングを返します。

```
HRank- チャート関数([TOTAL] expr[, mode[, fmt]])
```

クラスター関数

KMeans2D

[サイト ライセンス] プロパティグループには、Qlik Sense システムのライセンスに関連するプロパティが含まれています。全項目が必須で、空欄のままにはできません。

サイト ライセンスのプロパティ

プロパティ名	説明
[所有者名]	Qlik Sense 製品所有者のユーザー名。
[所有者の組織]	Qlik Sense 製品所有者が所属する組織の名称。
[シリアル番号]	Qlik Sense ソフトウェアに割り当てられているシリアル番号。
[コントロール番号]	Qlik Sense ソフトウェアに割り当てられているコントロール番号。
LEF アクセス	Qlik Sense ソフトウェアに割り当てられる ライセンス認証 ファイル (LEF)。

KMeans2D() は、K 平均法 クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターのクラスター ID を表示します。クラスタリング アルゴリズムで使用される列は，それぞれ，パラメーター coordinate_1 とcoordinate_2 によって決定されます。これらはともに集計です。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。データは、オプションで norm パラメーターによって正規化できます。

```
KMeans2D - チャート関数(num_clusters, coordinate_1, coordinate_2 [, norm])
```

KMeansND

KMeansND() は、K 平均法 クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターのクラスター ID を表示します。クラスタリング アルゴリズムで使用される列は，パラメーター coordinate_1、coordinate_2、などによって、最大 n 列まで決定されます。これらはすべて集計で

す。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。

```
KMeansND - チャート関数(num_clusters, num_iter, coordinate_1, coordinate_2
[,coordinate_3 [, ...]])
```

KMeansCentroid2D

KMeansCentroid2D() は、K 平均法クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターの目的の座標を表示します。クラスタリング アルゴリズムで使用される列は，それぞれ，パラメーターcoordinate_1 とcoordinate_2 によって決定されます。これらはともに集計です。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。データは、オプションで norm パラメーターによって正規化できます。

```
KMeansCentroid2D - チャート関数(num_clusters, coordinate_no, coordinate_1,
coordinate_2 [, norm])
```

KMeansCentroidND

KMeansCentroidND() は、K 平均法クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターの目的の座標を表示します。クラスタリング アルゴリズムで使用される列は，パラメーターcoordinate_1、coordinate_2、などによって、最大 n 列まで決定されます。これらはすべて集計です。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。

```
KMeansCentroidND - チャート関数(num_clusters, num_iter, coordinate_no,
coordinate_1, coordinate_2 [,coordinate_3 [, ...]])
```

時系列分解の関数

STL_Trend

STL_Trend は時系列の分解関数です。**STL_Seasonal** と**STL_Residual** と合わせて、この関数は、時系列を季節、トレンド、残差のコンポーネントに分解するために使用します。STL アルゴリズムのコンテキストでは、入力指標と他のパラメータが与えられた場合、繰り返される季節パターンと一般的なトレンドの両方を識別するために時系列分解を使用します。**STL_Trend** 関数は、時系列データの季節パターンやサイクルと関係なく、一般的トレンドを識別します。

```
STL_Trend - チャート関数(target_measure, period_int [,seasonal_smoother [,trend_
smoother]])
```

STL_Seasonal

STL_Seasonal は時系列の分解関数です。**STL_Trend** と**STL_Residual** と合わせて、この関数は、時系列を季節、トレンド、残差のコンポーネントに分解するために使用します。STL アルゴリズムのコンテキストでは、入力指標と他のパラメータが与えられた場合、繰り返される季節パターンと一般的なトレンドの両方を識別するために時系列分解を使用します。**STL_Seasonal** 関数は、データに表示される一般的トレンドと区別しながら、時系列内の季節パターンを特定します。

```
STL_Seasonal - チャート関数(target_measure, period_int [,seasonal_smoother
[,trend_smoother]])
```

STL_Residual

STL_Residual は時系列の分解関数です。**STL_Seasonal** と**STL_Trend** と合わせて、この関数は、時系列を季節、トレンド、残差のコンポーネントに分解するために使用します。STL アルゴリズムのコンテキストでは、入力指標と他のパラメータが与えられた場合、繰り返される季節パターンと一般的なトレンドの両方を識別するために時系列分解を使用します。この演算を実行すると、季節コンポーネントまたはトレンドコンポーネントのいずれにも当てはまらない入力メトリクスの変動の一部が、残差コンポーネントとして定義されます。**STL_Residual** チャート関数は、計算のこの部分を捕捉します。

```
STL_Residual - チャート関数(target_measure, period_int [,seasonal_smoother [,trend_smoother]])
```

Rank - チャート関数

Rank() は、数式におけるチャートの行を評価し、それぞれの行に対して、数式で評価される軸の値の相対位置を示します。この関数は数式の評価時に、結果を現在の列セグメントに含まれるその他の行の結果と比較して、セグメント内の現在の行の順位付けを返します。

列セグメント

テーブル以外のチャートでは、現在の列セグメントはチャートのストレートテーブルに相当するセグメントに従い定義されます。

構文:

```
Rank([TOTAL] expr[, mode[, fmt]])
```

戻り値データ型: dual

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
mode	関数の計算結果の数値表現を指定します。
fmt	関数の計算結果のテキスト表現を指定します。
TOTAL	チャートが 1 軸の場合、または数式の前に **TOTAL** 修飾子が付加されている場合は、関数は列全体に沿って評価されます。テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除く、すべての軸列の現在行と同じ値を持つ行だけが含まれます。

ランキングは、dual 値として返されます。行ごとに固有のランキングがある場合、1 から現在の列セグメント内の行数を示す整数になります。

複数の行がランキングを共有する場合は、テキストおよび数値表現を**mode** および **fmt** のパラメータで制御できます。

mode
2 番目の引数 **mode** は、次の値を取ることができます。

mode 例

値	説明
0 (デフォルト)	共有グループ内のすべての順位がランキング全体の中間値以下に入る場合、いずれの行も共有グループ内の最低順位を取得します。 共有グループ内のすべての順位が順位付け全体の中間値以上に入る場合は、いずれの行も共有グループ内の最高順位を取得します。 共有グループ内の順位がランキング全体の中間値をまたぐ場合は、いずれの行も列セグメント全体の最高順位と最低順位の平均に相当する値を取得します。
1	すべての行における最低順位。
2	すべての行における平均順位。
3	すべての行における最高順位。
4	最初の行における最低順位、その後は行ごとに 1 ずつ増加。

fmt
3 番目の引数 **fmt** は、次のいずれかの値になります。

fmt例

値	説明
0 (デフォルト)	すべての行における低い値 **-** 高い値 (例: 3 - 4)。
1	すべての行における低い値。
2	最初の行における低い値、その後のグループ内の行は空白。

mode 4 と**fmt** 2 の行の順序は、チャート軸のソート順で決定されます。

例と結果:

軸 Product とSales から 2 つのビジュアライゼーションを作成し、Product とUnitSales から別のビジュアライゼーションを作成します。次のテーブルに示すように、メジャーを追加します。

ランクの例

例	結果
例 1. 軸 `Customer` および `Sales` とメジャー `Rank(Sales)` を持つテーブルを作成する	結果は軸のソート順により異なります。テーブルが Customer でソートされると、テーブルには Astrida、次にBetacab, などについて、Sales のすべての値が表示されます。Rank(Sales) の結果は、Sales の値 12 に対して 10、Sales の値 13 に対して 9 と順番に、Sales の値 78 に対して rank 値 1 が返されます。次の列セグメントは Betacab で始まり、セグメント内の Sales の最初の値は 12 です。この Rank(Sales) の rank 値は 11 として指定されています。 テーブルが Sales でソートされている場合、列セグメントは Sales の値と対応する Customer の値で構成されます。12 というSales の値が 2 つ (Astrida および Betacab) あるため、その列セグメントの Rank(Sales) の値は Customer の各値で 1-2 となります。これは、Sales の値が 12 の Customer が 2 つあるためです。値が 4 つあった場合は、すべての行で 1-4 になります。これは、引数 fmt のデフォルト値 (0) の結果がどのようになるかを示しています。
例 2. 軸 Customer を Product に置換し、メジャー `Rank(Sales,1,2)` を追加する	この場合、引数 **mode** および **fmt** はそれぞれ 1 と2 に設定されているため、各列セグメントの最初の行には 1 が返され、その他の行は空白となります。

例 1 の結果、Customer でソートされたテーブル:

結果テーブル

Customer	Sales	Rank(Sales)
Astrida	12	10
Astrida	13	9
Astrida	20	8
Astrida	22	7
Astrida	45	6
Astrida	46	5
Astrida	60	4
Astrida	65	3
Astrida	70	2
Astrida	78	1
Betcab	12	11

例 1 の結果、Sales でソートされたテーブル:

結果テーブル

Customer	Sales	Rank(Sales)
Astrida	12	1-2
Betacab	12	1-2
Astrida	13	1
Betacab	15	1
Astrida	20	1
Astrida	22	1-2
Betacab	22	1-2
Betacab	24	1-2
Canutility	24	1-2

例で使用されているデータ:

```
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 '|');
```

参照先:

- *Sum - チャート関数 (page 348)*

HRank- チャート関数

HRank() は expression を評価し、結果をピボットテーブルの現在の行セグメント内のその他の列の結果と比較します。この関数は、セグメント内の現在の例のランキングを返します。

構文:

```
HRank([ TOTAL ] expr [ , mode [, fmt ] ])
```

戻り値データ型: dual

> *この関数は、ピボットテーブルでのみ有効です。他の種類のチャートでは、NULL を返します。*

引数:

引数

引数	説明
expr	メジャーの対象となるデータが含まれている数式または項目。
mode	関数の計算結果の数値表現を指定します。
fmt	関数の計算結果のテキスト表現を指定します。
TOTAL	チャートが 1 軸の場合、または数式の前に **TOTAL** 修飾子が付加されている場合は、関数は列全体に沿って評価されます。テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在の列セグメントには、項目間ソート順の最後の軸を表示する列を除く、すべての軸列の現在行と同じ値を持つ行だけが含まれます。

ピボットテーブルが 1 軸の場合、または数式の前に **total** 修飾子が配置されている場合は、現在の行セグメントは常に行全体になります。ピボットテーブルに複数の水平軸が存在する場合、現在の行セグメントには、項目ソート順の最後の水平軸を示す行を除くすべての軸行の現在列と同じ値を持つ列だけが含まれます。

順位付けは、デュアル値として返されます。列ごとに一意の順位付けがある場合は、1 と現在の行セグメント内の列数の間の整数になります。

複数の列がランキングを共有する場合は、テキストおよび数値表現を **mode** および **format** の引数で制御できます。

2 番目の引数 **mode** は、関数結果の数値表現を指定:

mode 例

値	説明
0 (デフォルト)	共有グループ内のすべての順位が順位付け全体の中間値の下位側に入る場合は、すべての列は、共有グループ内の最も低い順位を取得します。 共有グループ内のすべての順位が順位付け全体の中間値の上位側に入る場合は、すべての列は、共有グループ内の最も高い順位を取得します。 共有グループ内の順位がランキング全体の中間値をまたぐ場合は、いずれの行も列セグメント全体の最高順位と最低順位の平均に相当する値を取得します。
1	グループ内のすべての列における最も低い順位。
2	グループ内のすべての列における平均順位。
3	グループ内のすべての列における最も高い順位。
4	最初の列における最も低い順位、その後はグループ内の列ごとに1ずつ増加。

3番目の引数 **format** は、関数結果のテキスト表現を指定:

format 例

値	説明
0 (デフォルト)	グループ内のすべての列における小さい値 **&' - '&** 大きい値 (例: 3 - 4)。
1	グループ内のすべての列における小さい順位。
2	最初の列における小さい値、その後のグループ内の列は空白。

mode 4 と**format** 2 の列の順序は、チャート軸のソート順で決定されます。

```
HRank( sum( Sales ))

HRank( sum( Sales ), 2 )

HRank( sum( Sales ), 0, 1 )
```

k-means を使用した最適化: 実世界の例

次の例は、データセットに KMeans クラスタリング関数とCentroid 関数を適用した実際の使用例を示しています。KMeans 関数は、データポイントを類似性のあるクラスターに分離します。KMeans アルゴリズムを設定可能な数のイタレーションで適用すると、クラスターはよりコンパクトになり、差別化されます。

Kmeans は、さまざまな用途で多くの分野で使用されています。クラスタリングのユースケースの例としては、顧客セグメンテーション、不正検出、アカウントの減少予測、顧客インセンティブのターゲティング、サイバー犯罪者の特定、配送ルートの最適化などがあります。パターンを推測してサービスの提供を最適化しようとする企業では、KMeansのクラスタリングアルゴリズムの利用が増加しています。

Qlik Sense KMeans 関数とCentroid 関数

Qlik Sense には、データポイントを類似性に基づいてクラスタに分類する2 つの KMeans 関数があります。「*KMeans2D - チャート関数 (page 1363)*」および「*KMeansND - チャート関数 (page 1378)*」を参照してください。**KMeans2D** 関数は 2 つの軸を受けて、**散布図**チャートを使用して結果を視覚化するのに適しています。**KMeansND** 関数は、2 つ以上の軸を受けます。標準的なチャートでは 2D の結果を概念化することが容易なため、次のデモでは、2 次元を使用した**散布図**に KMeans を適用します。KMeans のクラスタリングは式、またはこの例で説明されているように軸による配色で可視化することができます。

Qlik Sense のセントロイド関数は、クラスター内のすべてのデータポイントの算術平均位置を求め、そのクラスターの中心点 (セントロイド) を特定します。centroid関数は、各チャートの行 (またはレコード) に対して、このデータポイントが割り当てられているクラスターの座標を表示します。「*KMeansCentroid2D - チャート関数 (page 1393)*」および「*KMeansCentroidND - チャート関数 (page 1394)*」を参照してください。

使用例と例の概要

次の例では、現実世界のシナリオをシミュレートしています。米国ニューヨーク州の繊維会社は、配送コストを最小限に抑えて経費を削減する必要があります。その一つとして、流通業者に近い場所に倉庫を移す方法があります。その会社はニューヨーク州全域で 118 の流通業者を雇用しています。次のデモでは、オペレーション マネージャが KMeans 関数を使用して販売店をクラスタ化された 5 つの地域にセグメント化し、次に Centroid 関数を使用してこれらのクラスタの中心となる 5 つの最適な倉庫の場所を特定する方法をシミュレートします。この目的は、5 つの中央倉庫の場所を特定するために使用できるマッピング座標を検出することです。

データセット

データセットは、実際の緯度と経度の座標でランダムに生成されたニューヨーク州の名前と住所を基にしています。データセットには、次の 10 の列が含まれます: id (ID)、first_name (名)、last_name (姓)、telephone (電話)、address (住所)、city (市区町村)、state (都道府県)、zip (郵便番号)、latitude (緯度)、longitude (経度). データセットは、ローカルにダウンロードしてから、Qlik Sense にアップロード可能なファイルとして、またはデータロードエディタでインラインとして、以下で利用できます。作成されるアプリは、*Distributors KMeans and Centroid* という名前になり、アプリの最初のシートは *Distribution cluster analysis* という名前になります。

以下のリンクを選択して、サンプル データ ファイルをダウンロードします。DistributorData.csv

Distributor データセット: Qlik Sense のデータ ロード エディター用 インライン ロード (page 1361)

タイトル: 販売店データ

レコードの合計数: 118

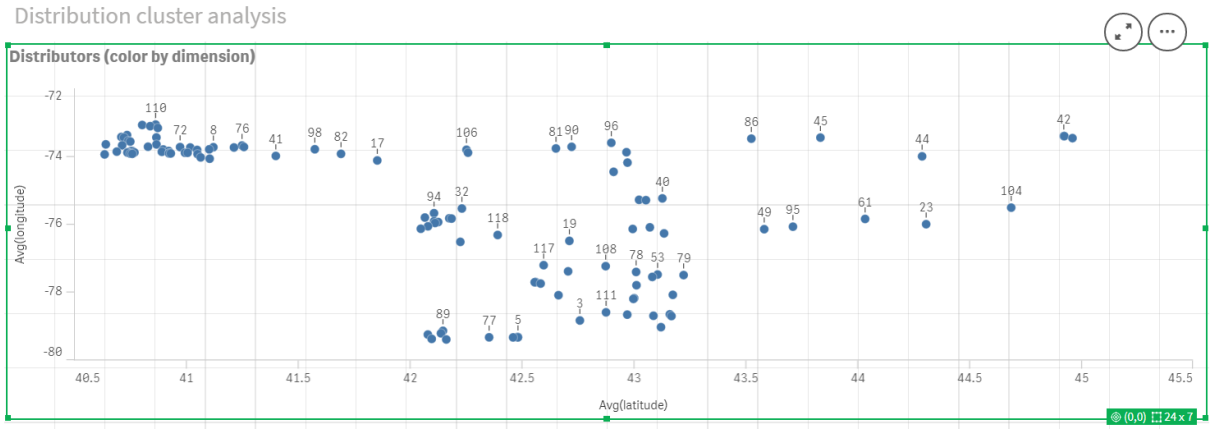
KMeans2D 関数の適用

この例では、*DistributorData* データセットを使用して、**KMeans2D** 関数を適用してチャートを軸ごとに色分けした**散布図**チャートの設定を実演しています。

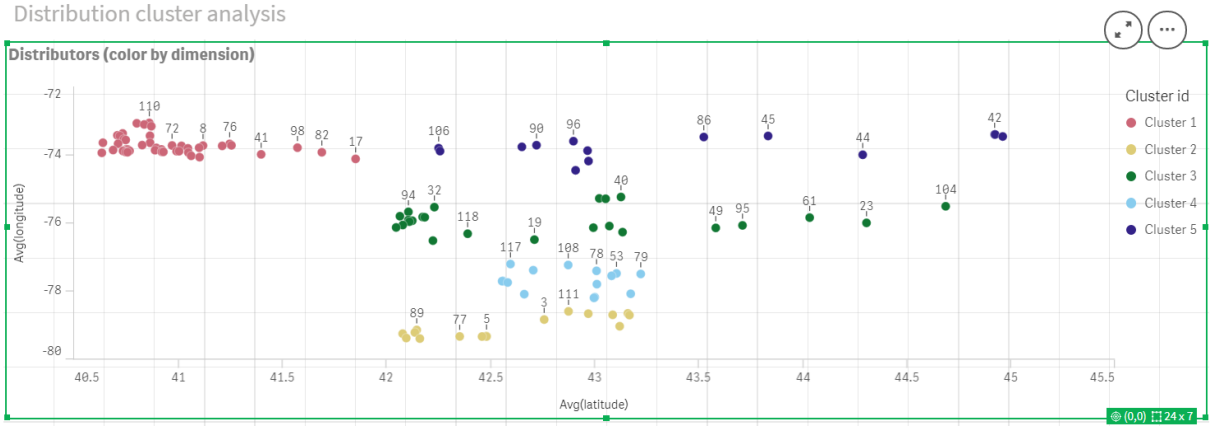
なお、Qlik Sense KMeans 関数では、深度差法 (DeD) と呼ばれる方法でオートクラスター化をサポートしています。ユーザーがクラスターの数に 0 を設定すると、そのデータセットに最適なクラスターの数が決定されます。ただし、この例では、**num_clusters** の引数に変数が作成されます(構文については *KMeans2D - チャート関数 (page 1363)* を参照します) 。従って、希望するクラスタ数 (k=5) を変数で指定します。

1. **散布図** チャートをシートにドラッグして、*販売店 (軸ごと)*と言う名前を付けます。
2. クラスターの数を指定する**変数**が作成されます。**変数**名は *vDistClusters* です。変数 **Definition** には、*5* を入力します。
3. チャートの **データ** 構成:
   a. **軸** で、**バブル** に *ID* 項目が選択されます。**ラベル** の *クラスターID* が入力されます。
   b. **メジャー**では、*Avg([latitude])* は **X 軸**の式です。
   c. **メジャー**では、*Avg([longitude])* は **Y 軸**の式です。
4. **外観**の設定:
   a. **[色と凡例]** で、**[色]** に**カスタム**が選択されます。
   b. チャートの配色に **軸ごと**が選択されます。
   c. 次の式が入力されます: *=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')*
   d. **永続色** のチェックボックスが選択されます。

軸ごとに KMeans の配色を適用する前の散布図



軸ごとに KMeans の配色を適用した後の散布図



テーブルの追加: *販売店*

関連するデータにすぐにアクセスするための表が手元にあると便利です。**分布図** チャートでは *ID* が表示されますが、対応する販売店名称の表が、参照用に追加されています。

1. *販売店*という名前の**表**が、次の**列** (軸) が追加された状態でシートにドラッグされます: *id (ID)*、*first_name (名)*、*last_name (姓)*。

テーブル: 販売店名

Distributors

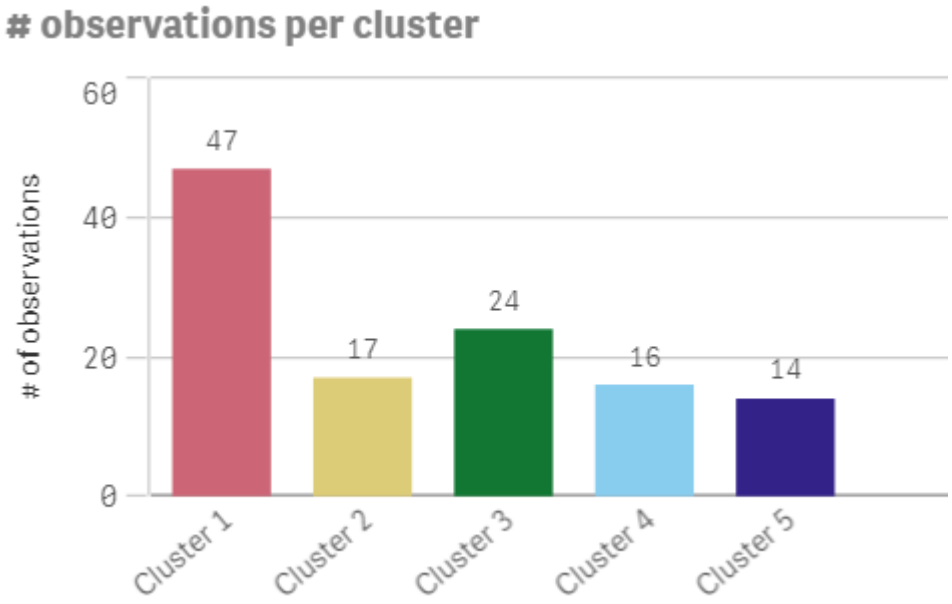
id	first_name	last_name
1	Kaiya	Snow
2	Dean	Roy
3	Eden	Paul
4	Bryanna	Higgins
5	Elisabeth	Lee
6	Skylar	Robinson
7	Cody	Bailey
8	Dario	Sims
9	Deacon	Hood

棒グラフの追加: *# クラスターごとの観察*

倉庫の物流シナリオでは、いくつの販売店が各倉庫でサービスを提供するのかを把握するのに役立ちます。これによって、各クラスタに割り当てられている販売店の数を測定する**棒グラフ**が作成されます。

1. **棒グラフ**がシートにドラッグされます。グラフに付けられた名前: *# クラスターごとの観察*
2. **棒グラフ用のデータ**構成
   a. *クラスター*というラベルが付けられた**軸**が追加されます(ラベルは式を適用した後に追加できます) 。次の式が入力されます: *=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')*
   b. *#of observations*というラベルが付けられた**メジャー**が追加されます。次の式が入力されます: *=count(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id))*
3. **外観**の設定:
   a. [**色と凡例**] で、[**色**] に**カスタム**が選択されます。
   b. チャートの配色に**軸ごと**を選択します。
   c. 次の式が入力されます: *=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1, 'Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5')*
   d. **永続色** のチェックボックスが選択されます。
   e. [**凡例の表示**]がオフになります。
   f. **プレゼンテーション**で、**値ラベル**が**自動**に切り替わります。
   g. **X軸**: **クラスター、ラベルのみ**が選択されます。

棒グラフ:# クラスターごとの観察



Centroid2D 関数の適用

2つ目のテーブルが **Centroid2D** 関数用に追加され、倉庫の候補地の座標が特定されます。次の表は、特定された 5 つの販売店のグループの中央の場所 (図心値) を示しています。

1. **表**がシートにドラッグされ、*クラスター重心*と言う名前が付けられて、次の列が追加されます:
   a. *クラスター*というラベルが付けられた**軸**が追加されます。次の式を入力します: *=pick(aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)+1,'Warehouse 1','Warehouse 2','Warehouse 3','Warehouse 4','Warehouse 5')*
   b. *緯度 (D1)*と言うラベルが付けられた**メジャー**が追加されます。次の式を入力します: *=only(aggr(KMeansCentroid2D(vDistClusters,0,only(latitude),only(longitude)),id))*
   なお、パラメータ**coordinate_no**は、1つ目の軸(0)に対応します。このケースでは、X 軸に対して*緯度*軸がプロットされます。**CentroidND** 関数を使用して、最大 6 つの軸がある場合、これらのパラメータの項目エントリに次の6つが使用できます:0、1、2、3、4、または5。
   c. *経度 (D2)*と言うラベルが付けられた**メジャー**が追加されます。次の式を入力します: *=only(aggr(KMeansCentroid2D(vDistClusters,1,only(latitude),only(longitude)),id))*
   この数式内のパラメータ**coordinate_no** は、2つ目の軸(1) に対応します。Y 軸に対して*経度*軸がプロットされます。

テーブル: クラスター重心の計算

Cluster centroids

Clusters	latitude (D1)	longitude (D2)
Totals	-	-
Warehouse 1	40.945422240426	-73.719966482979
Warehouse 2	42.590538729412	-79.067889217647
Warehouse 3	42.805089516667	-75.901621883333
Warehouse 4	42.8581692625	-77.6800485875
Warehouse 5	43.436770771429	-73.734622635714

重心のマッピング

次のステップでは、重心のマッピングを行います。ビジュアライゼーションを別のシートに配置するかどうかは、アプリ開発者次第です。

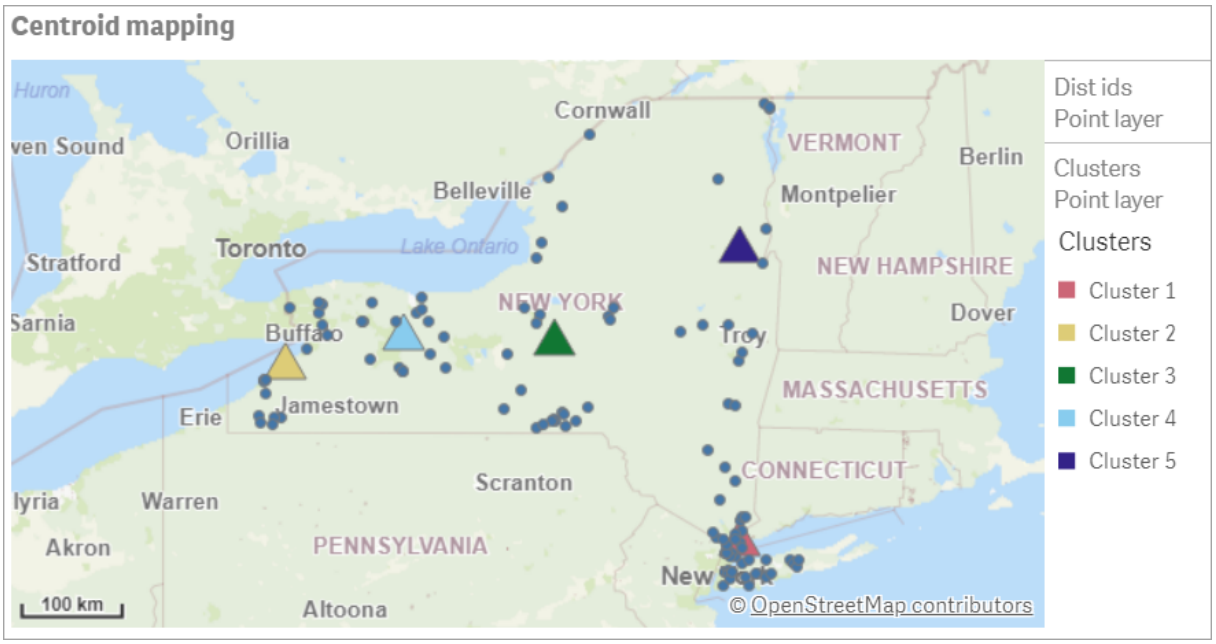
1. *重心のマッピング*と言う名前の**マッピング**をシートにドラッグします。
2. **レイヤー**セクション。[**レイヤーの追加**] を選択し、次に [**ポイントレイヤー**] を選択します。
    a. **項目**の *ID* を選択し、*Dist ID*ラベルを追加します。
    b. [**ロケーション**] セクションでは、[**緯度および経度欄**] のチェックボックスを選択します。
    c. **緯度**には、*緯度*フィールドを選択します。
    d. **経度**には、*経度*フィールドを選択します。
    e. [**サイズと形状**] セクション内で、[**形状**] で [**バブル**] を選択して、ライダーで好みに合わせてス [**サイズ**] を縮小します。
    f. [**カラー**] セクションで、[**単色**] を選択して [**色**] に青を、[**輪郭**] 色にグレーを選択します(これらの選択も好みの問題です) 。
3. [**レイヤー**] セクションで、[**レイヤーの追加**] を選択してから [**ポイントレイヤー**] を選択すると、2つ目の**ポイントレイヤー**が追加されます。
    a. 次の式が入力されます: *=aggr(KMeans2D(vDistClusters,only(latitude),only(longitude)),id)*
    b. **ラベル** *クラスター*を追加します。
    c. [**ロケーション**] セクションでは、[**緯度および経度欄**] のチェックボックスを選択します。
    d. ここでは、X 軸に沿ってプロットされる **緯度** に対して、次の式が追加されます: *=aggr(KMeansCentroid2D(vDistClusters,0,only(latitude),only(longitude)),id)*
    e. ここでは、Y 軸に沿ってプロットされる **経度** に対して、次の式が追加されます: *=aggr(KMeansCentroid2D(vDistClusters,1,only(latitude),only(longitude)),id)*
    f. [**サイズと形状**] セクション内で、[**形状**] で [**トライアングル**] を選択して、ライダーで好みに合わせてス [**サイズ**] を縮小します。
    g. [**色と凡例**] で、[**色**] に**カスタム**を選択します。

h. チャートの配色に**軸ごと**を選択します。次の式が入力されます: *=pick(aggr(KMeans2D (vDistClusters,only(latitude),only(longitude)),id)+1,'Cluster 1','Cluster 2','Cluster 3','Cluster 4','Cluster 5')*

i. 軸にクラスターとラベル付けされます。

4. [**マッピング設定**]で、[**プロジェクション**] に [**適応型**] を選択します。[**測定単位**] に、[**メートル法**] を選択します。

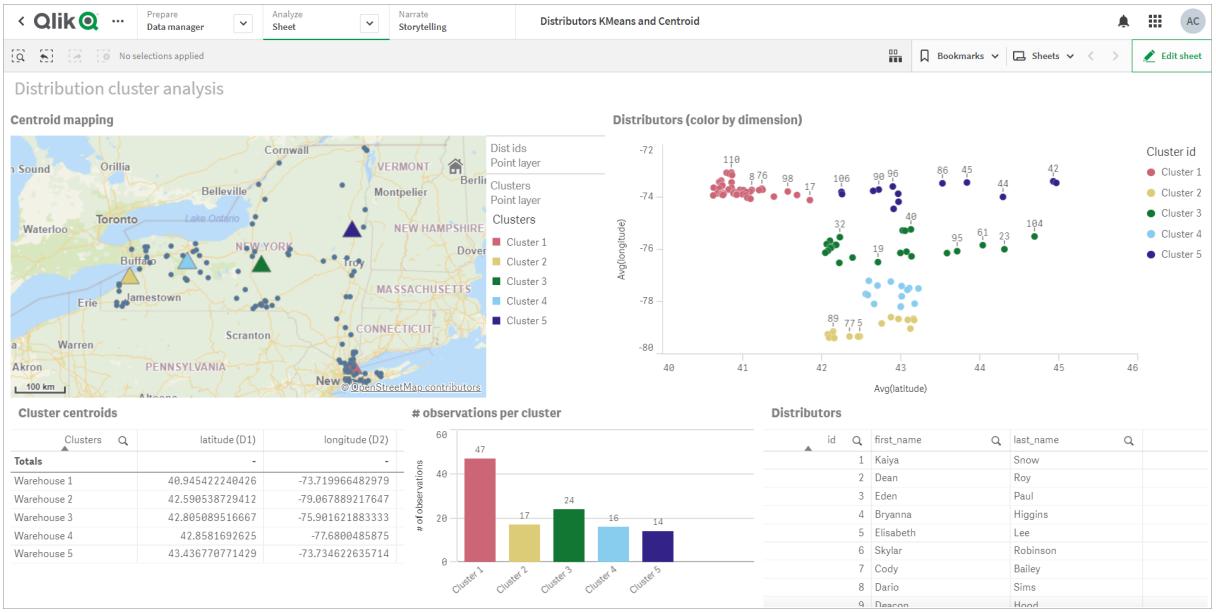
マッピング: クラスタによってマッピングされた重心



結論

この実世界用のシナリオでは、KMeans関数を使用して、販売業者は類似性に基づいて類似グループまたはクラスターにセグメント化されます。このケースでは、一方に近接しています。これらのクラスタに重心関数を適用して、5 つのマッピング座標を識別しました。これらの座標は、倉庫を建設または配置するための初期に中心となる場所を提示します。重心関数が**マッピング** チャートに適用されるため、アプリ利用者は、周囲のクラスターのデータポイントに対する重心の位置を視覚化できます。結果として、ニューヨーク州内の販売店への配送コストを最小限に抑えることができる倉庫の候補地を示す座標が得られました。

アプリ: KMeansと重心解析の例



Distributor データセット: Qlik Sense のデータ ロード エディター用 インライン ロード

```
DistributorData:
Load * Inline [
id,first_name,last_name,telephone,address,city,state,zip,latitude,longitude
1,Kaiya,Snow,(716) 201-1212,6231 Tonawanda Creek Rd #APT 308,Lockport,NY,14094,43.08926,-
78.69313
2,Dean,Roy,(716) 201-1588,6884 E High St,Lockport,NY,14094,43.16245,-78.65036
3,Eden,Paul,(716) 202-4596,4647 Southwestern Blvd #APT 350,Hamburg,NY,14075,42.76003,-78.83194
4,Bryanna,Higgins,(716) 203-7041,418 Park Ave,Dunkirk,NY,14048,42.48279,-79.33088
5,Elisabeth,Lee,(716) 203-7043,36 E Courtney St,Dunkirk,NY,14048,42.48299,-79.31928
6,Skylar,Robinson,(716) 203-7166,26 Greco Ln,Dunkirk,NY,14048,42.4612095,-79.3317925
7,Cody,Bailey,(716) 203-7201,114 Lincoln Ave,Dunkirk,NY,14048,42.4801269,-79.322232
8,Dario,Sims,(408) 927-1606,N Castle Dr,Armonk,NY,10504,41.11979,-73.714864
9,Deacon,Hood,(410) 244-6221,4856 44th St,Woodside,NY,11377,40.748372,-73.905445
10,Zackery,Levy,(410) 363-8874,61 Executive Blvd,Farmingdale,NY,11735,40.7197457,-73.430239
11,Rey,Hawkins,(412) 344-8687,4585 Shimerville Rd,Clarence,NY,14031,42.972075,-78.6592452
12,Phillip,Howard,(413) 269-4049,464 Main St #101,Port Washington,NY,11050,40.8273756,-
73.7009971
13,Shirley,Tyler,(434) 985-8943,114 Glann Rd,Apalachin,NY,13732,42.0482515,-76.1229725
14,Aniyah,Jarvis,(440) 244-1808,87 N Middletown Rd,Pearl River,NY,10965,41.0629,-74.0159
15,Alayna,Woodard,(478) 335-3704,70 W Red Oak Ln,West Harrison,NY,10604,41.0162722,-73.7234926
16,Jermaine,Lambert,(508) 561-9836,24 Kellogg Rd,New Hartford,NY,13413,43.0555739,-75.2793197
17,Harper,Gibbs,(239) 466-0238,Po Box 33,Cottekill,NY,12419,41.853392,-74.106082
18,Osvaldo,Graham,(252) 246-0816,6878 Sand Hill Rd,East Syracuse,NY,13057,43.073215,-76.081448
19,Roberto,Wade,(270) 469-1211,3936 Holley Rd,Moravia,NY,13118,42.713044,-76.481227
20,Kate,Mcguire,(270) 788-3080,6451 State 64 Rte #3,Naples,NY,14512,42.707366,-77.380489
21,Dale,Andersen,(281) 480-5690,205 W Service Rd,Champlain,NY,12919,44.9645392,-73.4470831
22,Lorelai,Burch,(302) 644-2133,1 Brewster St,Glen Cove,NY,11542,40.865177,-73.633019
23,Amiyah,Flowers,(303) 223-0055,46600 Us Interstate 81 Rte,Alexandria
Bay,NY,13607,44.309626,-75.988365
```

```
24,Mckinley,Clements,(303) 918-3230,200 Summit Lake Dr,Valhalla,NY,10595,41.101145,-73.778298
25,Marc,Gibson,(607) 203-1233,25 Robinson St,Binghamton,NY,13901,42.107416,-75.901614
26,Kali,Norman,(607) 203-1400,1 Ely Park Blvd #APT 15,Binghamton,NY,13905,42.125866,-75.925026
27,Laci,Cain,(607) 203-1437,16 Zimmer Road,Kirkwood,NY,13795,42.066516,-75.792627
28,Mohammad,Perez,(607) 203-1652,71 Endicott Ave #APT 12,Johnson City,NY,13790,42.111894,-
75.952187
29,Izabelle,Pham,(607) 204-0392,434 State 369 Rte,Port Crane,NY,13833,42.185838,-75.823074
30,Kiley,Mays,(607) 204-0870,244 Ballyhack Rd #14,Port Crane,NY,13833,42.175612,-75.814917
31,Peter,Trevino,(607) 205-1374,125 Melbourne St.,Vestal,NY,13850,42.080254,-76.051124
32,Ani,Francis,(607) 208-4067,48 Caswell St,Afton,NY,13730,42.232065,-75.525674
33,Jared,Sheppard,(716) 386-3002,4709 430th Rte,Bemus Point,NY,14712,42.162175,-79.39176
34,Dulce,Atkinson,(914) 576-2266,501 Pelham Rd,New Rochelle,NY,10805,40.895449,-73.782602
35,Jayla,Beasley,(716) 526-1054,5010 474th Rte,Ashville,NY,14710,42.096859,-79.375561
36,Dane,Donovan,(718) 545-3732,5014 31st Ave,Woodside,NY,11377,40.756967,-73.909506
37,Brendon,Clay,(585) 322-7780,133 Cummings Ave,Gainesville,NY,14066,42.664309,-78.085651
38,Asia,Nunez,(718) 426-1472,2407 Gilmore ,East Elmhurst,NY,11369,40.766662,-73.869185
39,Dawson,Odonnell,(718) 342-2179,5019 H Ave,Brooklyn,NY,11234,40.633245,-73.927591
40,Kyle,Collins,(315) 733-7078,502 Rockhaven Rd,Utica,NY,13502,43.129184,-75.226726
41,Eliza,Hardin,(315) 331-8072,502 Sladen Place,West Point,NY,10996,41.3993,-73.973003
42,Kasen,Klein,(518) 298-4581,2407 Lake Shore Rd,Chazy,NY,12921,44.925561,-73.387373
43,Reuben,Bradford,(518) 298-4581,33 Lake Flats Dr,Champlain,NY,12919,44.928092,-73.387884
44,Henry,Grimes,(518) 523-3990,2407 Main St,Lake Placid,NY,12946,44.291487,-73.98474
45,Kyan,Livingston,(518) 585-7364,241 Alexandria Ave,Ticonderoga,NY,12883,43.836553,-73.43155
46,Kaitlyn,Short,(516) 678-3189,241 Chance Dr,Oceanside,NY,11572,40.638534,-73.63079
47,Damaris,Jacobs,(914) 664-5331,241 Claremont Ave,Mount Vernon,NY,10552,40.919852,-73.827848
48,Alivia,Schroeder,(315) 469-4473,241 Lafayette Rd,Syracuse,NY,13205,42.996446,-76.12957
49,Bridget,Strong,(315) 298-4355,241 Maltby Rd,Pulaski,NY,13142,43.584966,-76.136317
50,Francis,Lee,(585) 201-7021,166 Ross St,Batavia,NY,14020,43.0031502,-78.17487
51,Makaila,Phelps,(585) 201-7422,58 S Main St,Batavia,NY,14020,42.99941,-78.1939285
52,Jazlynn,Stephens,(585) 203-1087,1 Sinclair Dr,Pittsford,NY,14534,43.084157,-77.545452
53,Ryann,Randolph,(585) 203-1519,331 Eaglehead Rd,East Rochester,NY,14445,43.10785,-77.475552
54,Rosa,Baker,(585) 204-4011,42 Ossian St,Dansville,NY,14437,42.560761,-77.70088
55,Marcel,Barry,(585) 204-4013,42 Jefferson St,Dansville,NY,14437,42.557735,-77.702983
56,Dennis,Schmitt,(585) 204-4061,750 Dansville Mount Morris Rd,Dansville,NY,14437,42.584458,-
77.741648
57,Cassandra,Kim,(585) 204-4138,3 Perine Ave APT1,Dansville,NY,14437,42.562865,-77.69661
58,Kolton,Jacobson,(585) 206-5047,4925 Upper Holly Rd,Holley,NY,14470,43.175957,-78.074465
59,Nathanael,Donovan,(718) 393-3501,9604 57th Ave,Corona,NY,11373,40.736077,-73.864858
60,Robert,Frazier,(718) 271-3067,300 56th Ave,Corona,NY,11373,40.735304,-73.873997
61,Jessie,Mora,(315) 405-8991,9607 Forsyth Loop,Watertown,NY,13603,44.036466,-75.833437
62,Martha,Rollins,(347) 242-2642,22 Main St,Corona,NY,11373,40.757727,-73.829331
63,Emely,Townsend,(718) 699-0751,60 Sanford Ave,Corona,NY,11373,40.755466,-73.831029
64,Kylie,Cooley,(347) 561-7149,9608 95th Ave,Ozone Park,NY,11416,40.687564,-73.845715
65,Wendy,Cameron,(585) 571-4185,9608 Union St,Scottsville,NY,14546,43.013327,-77.7907839
66,Kayley,Peterson,(718) 654-5027,961 E 230th St,Bronx,NY,10466,40.889275,-73.850555
67,Camden,Ochoa,(718) 760-8699,59 Vark St,Yonkers,NY,10701,40.929322,-73.89957
68,Priscilla,Castillo,(910) 326-7233,9359 Elm St,Chadwicks,NY,13319,43.024902,-75.26886
69,Dana,Schultz,(913) 322-4580,99 Washington Ave,Hastings on Hudson,NY,10706,40.99265,-
73.879748
70,Blaze,Medina,(914) 207-0015,60 Elliott Ave,Yonkers,NY,10705,40.921498,-73.896682
71,Finnegan,Tucker,(914) 207-0015,90 Hillside Drive,Yonkers,NY,10705,40.922514,-73.892911
72,Pranav,Palmer,(914) 214-8376,5 Bruce Ave,Harrison,NY,10528,40.970916,-73.711493
73,Kolten,Wong,(914) 218-8268,70 Barker St,Mount Kisco,NY,10549,41.211993,-73.723202
74,Jasiah,Vazquez,(914) 231-5199,30 Broadway,Dobbs Ferry,NY,10522,41.004629,-73.879825
75,Lamar,Pierce,(914) 232-0380,68 Ridge Rd,Katonah,NY,10536,41.256662,-73.707964
76,Carla,Coffey,(914) 232-0469,197 Beaver Dam Rd,Katonah,NY,10536,41.247934,-73.664363
```

```
77,Brooklynn,Harmon,(716) 595-3227,8084 Glasgow Rd,Cassadega,NY,14718,42.353861,-79.329558
78,Raquel,Hodges,(585) 398-8125,809 County Road ,Victor,NY,14564,43.011745,-77.398806
79,Jerimiah,Gardner,(585) 787-9127,809 Houston Rd,Webster,NY,14580,43.224204,-77.491353
80,Clarence,Hammond,(720) 746-1619,809 Pierpont Ave,Piermont,NY,10968,41.0491181,-73.918622
81,Rhys,Gill,(518) 427-7887,81 Columbia St,Albany,NY,12210,42.652824,-73.752096
82,Edith,Parrish,(845) 452-7621,81 Glenwood Ave,Poughkeepsie,NY,12603,41.691058,-73.910829
83,Kobe,Mcintosh,(845) 371-1101,81 Heitman Dr,Spring Valley,NY,10977,41.103227,-74.054396
84,Ayden,Waters,(516) 796-2722,81 Kingfisher Rd,Levittown,NY,11756,40.738939,-73.52826
85,Francis,Rogers,(631) 427-7728,81 Knollwood Ave,Huntington,NY,11743,40.864905,-73.426107
86,Jaden,Landry,(716) 496-4038,12839 39th Rte,Chaffee,NY,14030,43.527396,-73.462786
87,Giancarlo,Campos,(518) 885-5717,1284 Saratoga Rd,Ballston Spa,NY,12020,42.968594,-73.862847
88,Eduardo,Contreras,(716) 285-8987,1285 Saunders Sett Rd,Niagara Falls,NY,14305,43.122963,-
79.029274
89,Gabriela,Davidson,(716) 267-3195,1286 Mee Rd,Falconer,NY,14733,42.147339,-79.137976
90,Evangeline,Case,(518) 272-9435,1287 2nd Ave,Watervliet,NY,12189,42.723132,-73.703818
91,Tyrone,Ellison,(518) 843-4691,1287 Midline Rd,Amsterdam,NY,12010,42.9730876,-74.1700608
92,Bryce,Bass,(518) 943-9549,1288 Leeds Athens Rd,Athens,NY,12015,42.259381,-73.876897
93,Londyn,Butler,(518) 922-7095,129 Argersinger Rd,Fultonville,NY,12072,42.910969,-74.441917
94,Graham,Becker,(607) 655-1318,129 Baker Rd,Windsor,NY,13865,42.107271,-75.66408
95,Rolando,Fitzgerald,(315) 465-4166,17164 County 90 Rte,Mannsville,NY,13661,43.713443,-
76.06232
96,Grant,Hoover,(518) 692-8363,1718 County 113 Rte,Schaghticote,NY,12154,42.900648,-73.585036
97,Mark,Goodwin,(631) 584-6761,172 Cambon Ave,Saint James,NY,11780,40.871152,-73.146032
98,Deacon,Cantu,(845) 221-7940,172 Carpenter Rd,Hopewell Junction,NY,12533,41.57388,-73.77609
99,Tristian,Walsh,(516) 997-4750,172 E Cabot Ln,Westbury,NY,11590,40.7480397,-73.54819
100,Abram,Alexander,(631) 588-3817,172 Lorenzo Cir,Ronkonkoma,NY,11779,40.837123,-73.09367
101,Lesly,Bush,(516) 489-3791,172 Nassau Blvd,Garden City,NY,11530,40.71147,-73.660753
102,Pamela,Espinoza,(716) 201-1520,172 Niagara St ,Lockport,NY,14094,43.169871,-78.70093
103,Bryanna,Newton,(914) 328-4332,172 Warren Ave,White Plains,NY,10603,41.047207,-73.79572
104,Marcelo,Schmitt,(315) 393-4432,319 Mansion Ave,Ogdensburg,NY,13669,44.690246,-75.49992
105,Layton,Valenzuela,(631) 676-2113,319 Singingwood Dr,Holbrook,NY,11741,40.801391,-73.058993
106,Roderick,Rocha,(518) 671-6037,319 Warren St,Hudson,NY,12534,42.252527,-73.790629
107,Camryn,Terrell,(315) 635-1680,3192 Olive Dr,Baldinsville,NY,13027,43.136843,-76.260303
108,Summer,Callahan,(585) 394-4195,3192 Smith Road,Canandaigua,NY,14424,42.875457,-77.228039
109,Pierre,Novak,(716) 665-2524,3194 Falconer Kimball Stand Rd,Falconer,NY,14733,42.138439,-
79.211091
110,Kennedi,Fry,(315) 543-2301,32 College Rd,Selden,NY,11784,40.861624,-73.04757
111,Wyatt,Pruitt,(716) 681-4042,277 Ransom Rd,Lancaster ,NY,14086,42.87702,-78.591302
112,Lilly,Jensen,(631) 841-0859,2772 Schliegel Blvd,Amityville,NY,11701,40.708021,-73.413015
113,Tristin,Hardin,(631) 920-0927,278 Fulton Street,West Babylon,NY,11704,40.733578,-73.357321
114,Tanya,Stafford,(716) 484-0771,278 Sampson St,Jamestown,NY,14701,42.0797,-79.247805
115,Paris,Cordova,(607) 589-4857,278 Washburn Rd,Spencer,NY,14883,42.225046,-76.510257
116,Alfonso,Morse,(718) 359-5582,200 Colden St,Flushing,NY,11355,40.750403,-73.822752
117,Maurice,Hooper,(315) 595-6694,4435 Italy Hill Rd,Branchport,NY,14418,42.597957,-77.199267
118,Iris,Wolf,(607) 539-7288,444 Harford Rd,Brooktondale,NY,14817,42.392164,-76.30756
];
```

KMeans2D - チャート関数

KMeans2D() は、K 平均法クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターのクラスター ID を表示します。クラスタリング アルゴリズムで使用される列は，それぞれ，パラメーター coordinate_1 とcoordinate_2 によって決定されます。これらはともに集計です。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。データは、オプションで norm パラメーターによって正規化できます。

KMeans2D は、データポイントごとに 1 つの値を返します。戻り値はデュアル値であり、各データポイントが割り当てられているクラスターに対応する整数値です。

構文:

```
KMeans2D(num_clusters, coordinate_1, coordinate_2 [, norm])
```

戻り値データ型: dual

引数:

引数

引数	説明
num_clusters	クラスターの数を指定する整数。
coordinate_1	チャートから作成できる散布図の最初の座標 (通常は x 軸) を計算する集約です。追加のパラメーターである coordinate_2 は、2 番目の座標を計算します。
norm	KMeans クラスタリングの前にデータセットに適用されるオプションの正規化方法。 考えられる値: 正規化なしの場合は 0 または「なし」 z-score の正規化の場合は 1 または「zscore」 min-max の正規化の場合は 2 または「minmax」 パラメーターが指定されていない場合、または指定されたパラメーターが正しくない場合、正規化は適用されません。 z-score は、機能平均と標準偏差に基づいてデータを正規化します。z-score は、各機能のスケールが同じであることを保証するものではありませんが、外れ値を処理する場合は、min-max よりも優れたアプローチです。 min-max の正規化は、それぞれの最小値と最大値を取得し、各データポイントを再計算することにより、機能が同じスケールを持つことを保証します。

例: チャートの数式
この例では、*アイリス*データセットを使用して散布図チャートを作成し、KMeans を使用して式でデータに色を付けます。

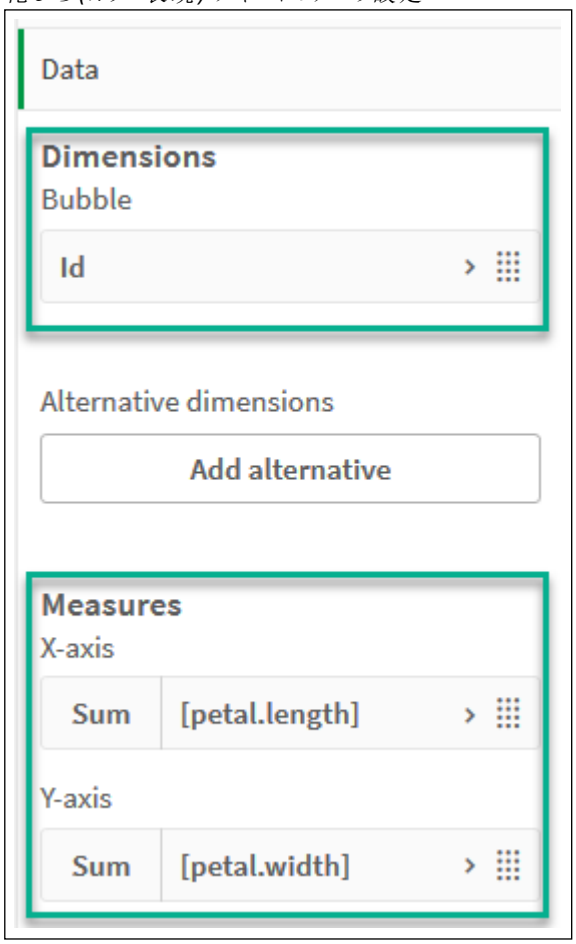
また、*num_clusters* 引数のための変数を作成し、変数値入力ボックスを使用してクラスターの数を変更します。

*アイリス*データセットは様々な形式で公開されています。Qlik Senseのデータロードエディターを使ってロードするインライン テーブルとしてデータを提供しました。この例では、データテーブルに *ID* 列を追加したことに注意してください。

Qlik Senseにデータをロードした後、以下を実施します。

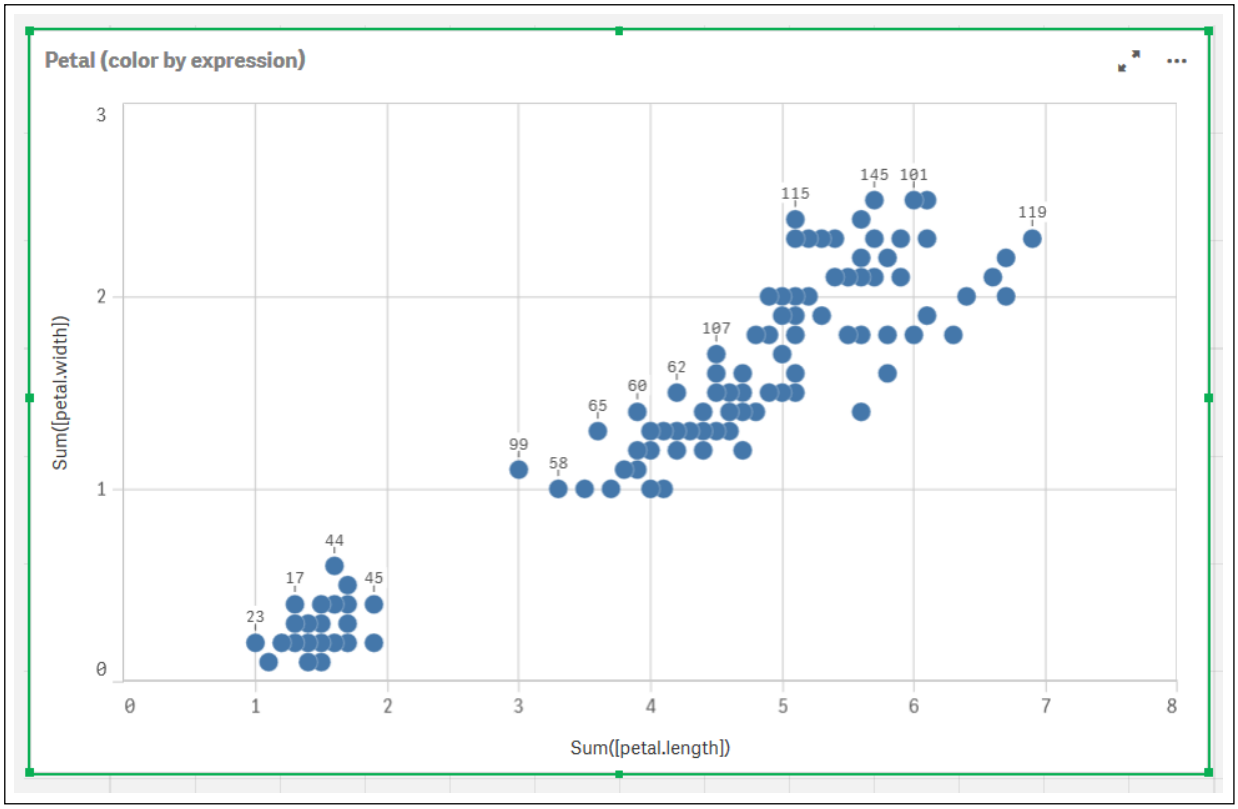
1. 新しいシートに**散布図**チャートをドラッグします。チャート*花びら(カラー表現)*に名前を付けます。
2. クラスターの数を指定する変数を作成します。変数の**名前**には、*KmeansPetalClusters* を入力します。変数の**定義**には、*=2* を入力します。
3. チャートの**データ**を構成:
   i. [**軸**] で、[**バブル**] の項目の [*ID*] を選択します。ラベルのクラスターIDを入力します。
   ii. [**メジャー**]で、**X 軸**の式として *Sum([petal.length])* を選択します。
   iii. [**メジャー**]で、**Y 軸**の式として *Sum([petal.width])* を選択します。

花びら(カラー表現) チャートのデータ設定



データポイントがチャートにプロットされます。

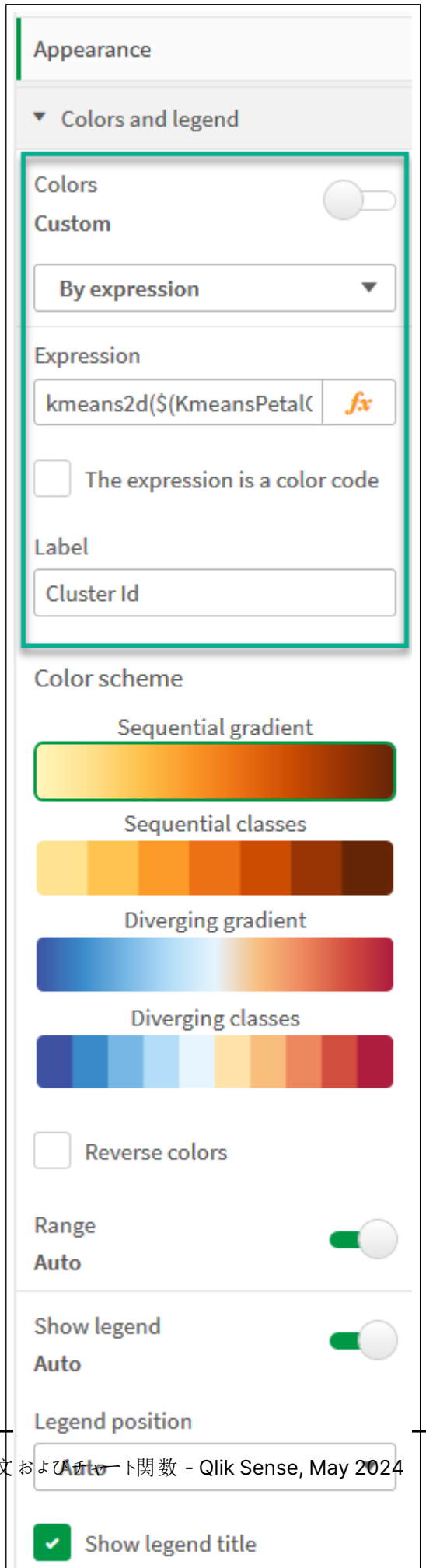
花びら(カラー表現)チャート上のデータポイント



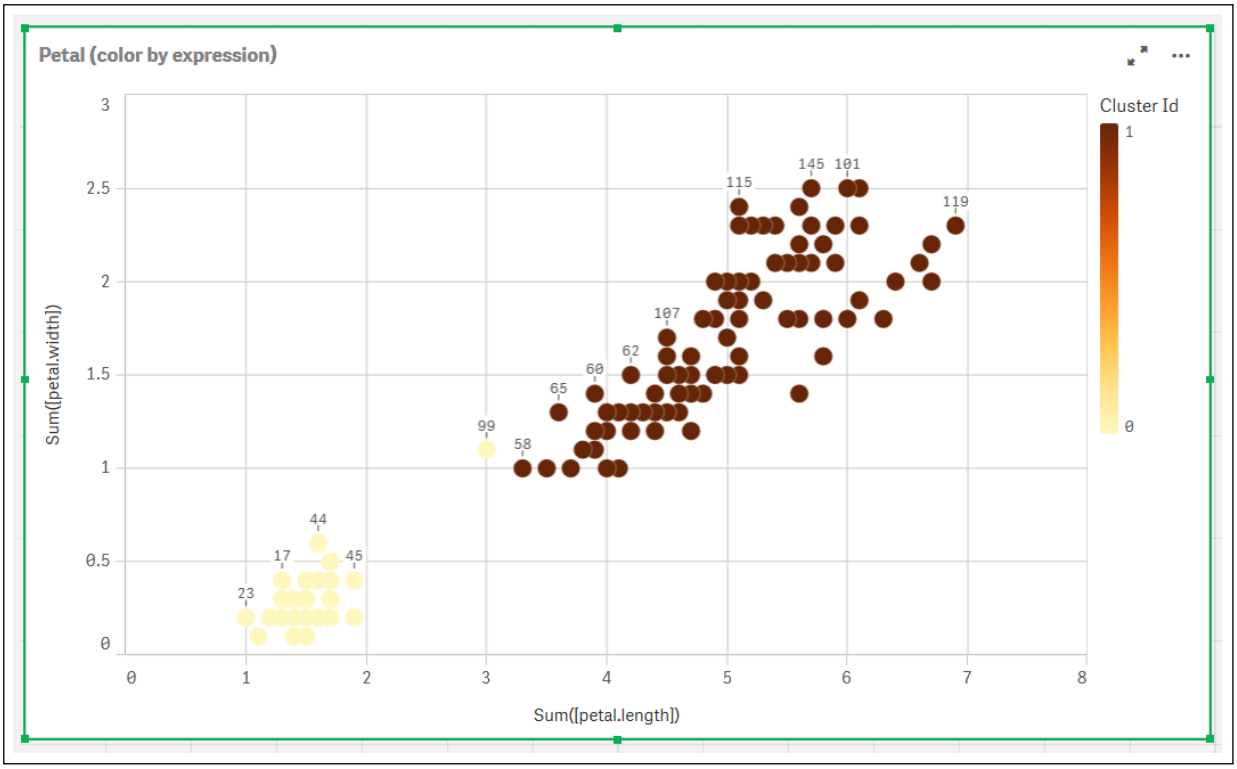
4. チャートの**外観**を構成:
   i. **[色と凡例]**で、**[色]**に**カスタム**を選択します。
   ii. チャートの色を**数式を使用**して選択します。
   iii. **数式**に次のように入力します: *kmeans2d($(KmeansPetalClusters), Sum([petal.length]), Sum([petal.width]))*
   *KmeansPetalClusters*は、*2*に設定した変数であることに注意してください。
   あるいは、次のように入力してください: *kmeans2d(2, Sum([petal.length]), Sum([petal.width]))*
   iv. **[数式は色コード]**チェックボックスの選択を解除します。

v. **ラベル**に次を入力します: *クラスター ID*

花びら(数式で色付け) チャートの外観設定


Appearance
Colors and legend
Colors
Custom
By expression
Expression
kmeans2d($(KmeansPetal(
The expression is a color code
Label
Cluster Id
Color scheme
Sequential gradient
Sequential classes
Diverging gradient
Diverging classes
Reverse colors
Range
Auto
Show legend
Auto
Legend position
Auto
Show legend title

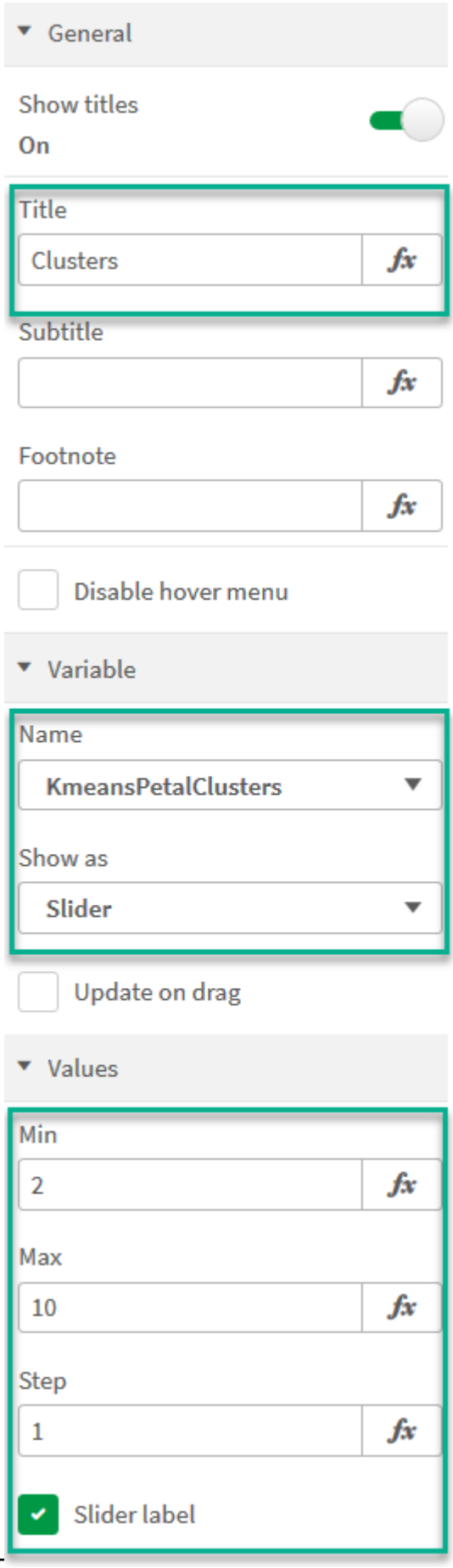
チャート上の 2 つのクラスターは、数式 KMeans によって色分けされています。
花びら(数式で色付け) チャート上の、数式で色付けされたクラスター



5. クラスター数の**変数値入力**ボックスを追加します。
   i. [**アセット**] パネルの [**カスタム オブジェクト**] で、[**Qlik ダッシュボード バンドル**] を選択します。ダッシュボード バンドルにアクセスできない場合でも、作成した変数を使用してクラスター数を変更するか、式の整数として直接変更できます。
   ii. **変数値入力**ボックスをシートへドラッグします。
   iii. [**外観**] で、[**一般**] をクリックします。
   iv. **タイトル**に次を入力します: *クラスター*
   v. [**変数**] をクリックします。
   vi. **名前**で次の変数値を選択します:*KmeansPetalClusters*。
   vii. [**表示の設定**] で [**スライダー**] を選択します。

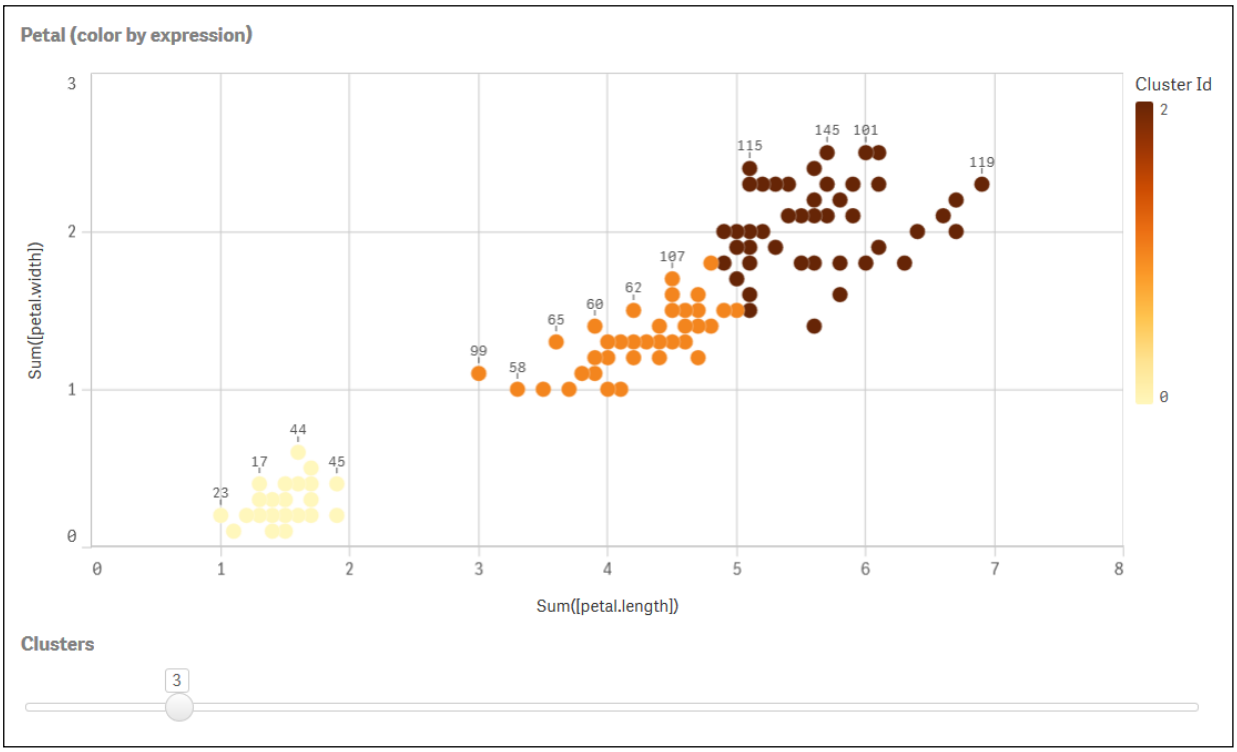
viii. **値**を選択し、必要に応じて設定を行います。

クラスターの外観変数入力ボックス


General
Show titles
On
Title
Clusters
Subtitle
Footnote
Disable hover menu
Variable
Name
KmeansPetalClusters
Show as
Slider
Update on drag
Values
Min
2
Max
10
Step
1
Slider label

編集の完了後に、[クラスター] 変数値入力ボックスのスライダを使用して、クラスターの数を変更できます。

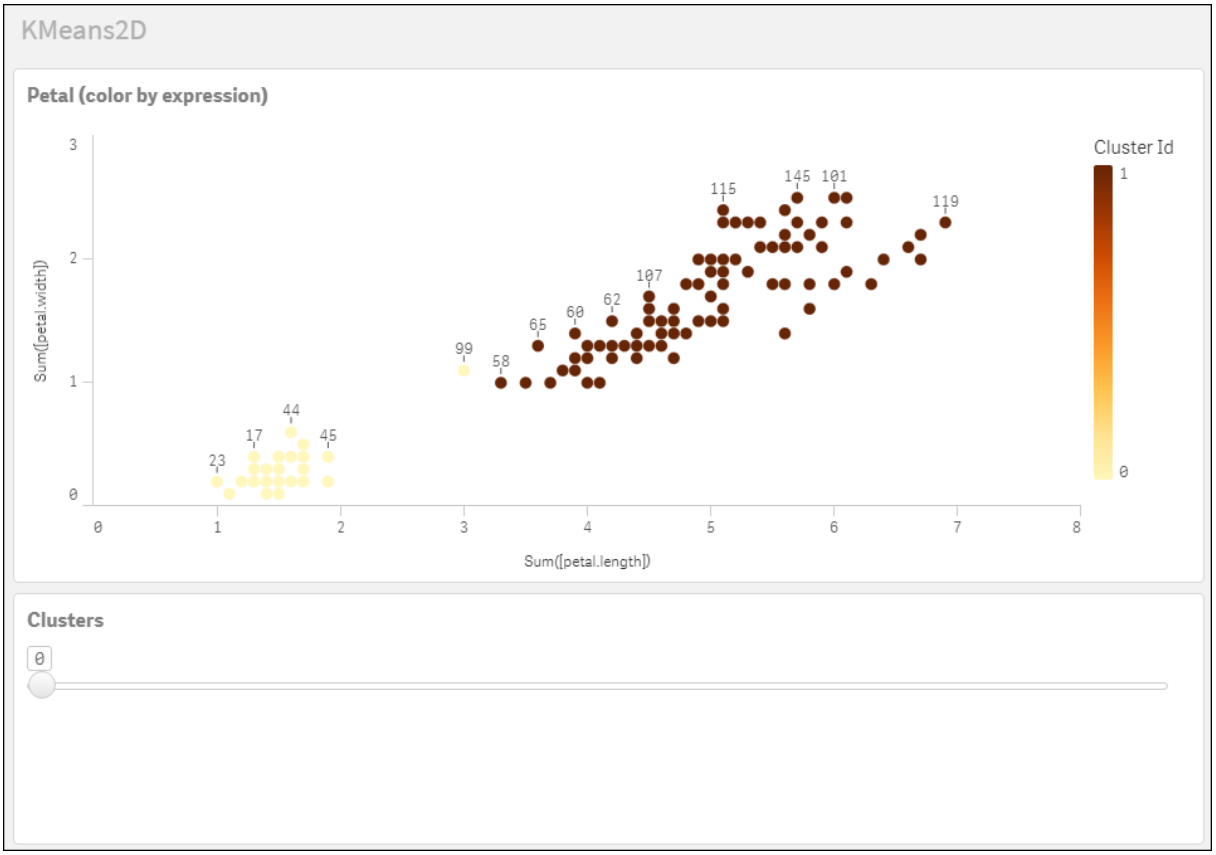
花びら (数式で色付け) チャート上の、数式で色付けされたクラスター



自動クラスタリング

KMeans 関数は、深度差異 (DeD) と呼ばれる方法を使用した自動クラスタリングをサポートします。ユーザーがクラスターの数に 0 を設定すると、そのデータセットに最適なクラスターの数が決定されます。クラスター数 (*k*) の整数は明示的に返されませんが、KMeans アルゴリズム内で計算されることに注意してください。例えば、*KmeansPetalClusters* の値の関数で 0 が指定されている場合、または変数入力ボックスを介して設定されている場合、クラスターの割り当ては、クラスターの最適な数に基づいてデータセットに対して自動的に計算されます。

(k) が 0 に設定されている場合、K 平均法深度差異メソッドはクラスターの最適数を決定します



Iris データセット:Qlik Sense のデータロードエディター用インラインロード

```
IrisData:
Load * Inline [
sepal.length, sepal.width, petal.length, petal.width, variety, id
5.1, 3.5, 1.4, 0.2, Setosa, 1
4.9, 3, 1.4, 0.2, Setosa, 2
4.7, 3.2, 1.3, 0.2, Setosa, 3
4.6, 3.1, 1.5, 0.2, Setosa, 4
5, 3.6, 1.4, 0.2, Setosa, 5
5.4, 3.9, 1.7, 0.4, Setosa, 6
4.6, 3.4, 1.4, 0.3, Setosa, 7
5, 3.4, 1.5, 0.2, Setosa, 8
4.4, 2.9, 1.4, 0.2, Setosa, 9
4.9, 3.1, 1.5, 0.1, Setosa, 10
5.4, 3.7, 1.5, 0.2, Setosa, 11
4.8, 3.4, 1.6, 0.2, Setosa, 12
4.8, 3, 1.4, 0.1, Setosa, 13
4.3, 3, 1.1, 0.1, Setosa, 14
5.8, 4, 1.2, 0.2, Setosa, 15
5.7, 4.4, 1.5, 0.4, Setosa, 16
5.4, 3.9, 1.3, 0.4, Setosa, 17
5.1, 3.5, 1.4, 0.3, Setosa, 18
5.7, 3.8, 1.7, 0.3, Setosa, 19
5.1, 3.8, 1.5, 0.3, Setosa, 20
5.4, 3.4, 1.7, 0.2, Setosa, 21
```

```
5.1, 3.7, 1.5, 0.4, Setosa, 22
4.6, 3.6, 1, 0.2, Setosa, 23
5.1, 3.3, 1.7, 0.5, Setosa, 24
4.8, 3.4, 1.9, 0.2, Setosa, 25
5, 3, 1.6, 0.2, Setosa, 26
5, 3.4, 1.6, 0.4, Setosa, 27
5.2, 3.5, 1.5, 0.2, Setosa, 28
5.2, 3.4, 1.4, 0.2, Setosa, 29
4.7, 3.2, 1.6, 0.2, Setosa, 30
4.8, 3.1, 1.6, 0.2, Setosa, 31
5.4, 3.4, 1.5, 0.4, Setosa, 32
5.2, 4.1, 1.5, 0.1, Setosa, 33
5.5, 4.2, 1.4, 0.2, Setosa, 34
4.9, 3.1, 1.5, 0.1, Setosa, 35
5, 3.2, 1.2, 0.2, Setosa, 36
5.5, 3.5, 1.3, 0.2, Setosa, 37
4.9, 3.1, 1.5, 0.1, Setosa, 38
4.4, 3, 1.3, 0.2, Setosa, 39
5.1, 3.4, 1.5, 0.2, Setosa, 40
5, 3.5, 1.3, 0.3, Setosa, 41
4.5, 2.3, 1.3, 0.3, Setosa, 42
4.4, 3.2, 1.3, 0.2, Setosa, 43
5, 3.5, 1.6, 0.6, Setosa, 44
5.1, 3.8, 1.9, 0.4, Setosa, 45
4.8, 3, 1.4, 0.3, Setosa, 46
5.1, 3.8, 1.6, 0.2, Setosa, 47
4.6, 3.2, 1.4, 0.2, Setosa, 48
5.3, 3.7, 1.5, 0.2, Setosa, 49
5, 3.3, 1.4, 0.2, Setosa, 50
7, 3.2, 4.7, 1.4, Versicolor, 51
6.4, 3.2, 4.5, 1.5, Versicolor, 52
6.9, 3.1, 4.9, 1.5, Versicolor, 53
5.5, 2.3, 4, 1.3, Versicolor, 54
6.5, 2.8, 4.6, 1.5, Versicolor, 55
5.7, 2.8, 4.5, 1.3, Versicolor, 56
6.3, 3.3, 4.7, 1.6, Versicolor, 57
4.9, 2.4, 3.3, 1, Versicolor, 58
6.6, 2.9, 4.6, 1.3, Versicolor, 59
5.2, 2.7, 3.9, 1.4, Versicolor, 60
5, 2, 3.5, 1, Versicolor, 61
5.9, 3, 4.2, 1.5, Versicolor, 62
6, 2.2, 4, 1, Versicolor, 63
6.1, 2.9, 4.7, 1.4, Versicolor, 64
5.6, 2.9, 3.6, 1.3, Versicolor, 65
6.7, 3.1, 4.4, 1.4, Versicolor, 66
5.6, 3, 4.5, 1.5, Versicolor, 67
5.8, 2.7, 4.1, 1, Versicolor, 68
6.2, 2.2, 4.5, 1.5, Versicolor, 69
5.6, 2.5, 3.9, 1.1, Versicolor, 70
5.9, 3.2, 4.8, 1.8, Versicolor, 71
6.1, 2.8, 4, 1.3, Versicolor, 72
6.3, 2.5, 4.9, 1.5, Versicolor, 73
6.1, 2.8, 4.7, 1.2, Versicolor, 74
6.4, 2.9, 4.3, 1.3, Versicolor, 75
6.6, 3, 4.4, 1.4, Versicolor, 76
```

```
6.8, 2.8, 4.8, 1.4, Versicolor, 77
6.7, 3, 5, 1.7, Versicolor, 78
6, 2.9, 4.5, 1.5, Versicolor, 79
5.7, 2.6, 3.5, 1, Versicolor, 80
5.5, 2.4, 3.8, 1.1, Versicolor, 81
5.5, 2.4, 3.7, 1, Versicolor, 82
5.8, 2.7, 3.9, 1.2, Versicolor, 83
6, 2.7, 5.1, 1.6, Versicolor, 84
5.4, 3, 4.5, 1.5, Versicolor, 85
6, 3.4, 4.5, 1.6, Versicolor, 86
6.7, 3.1, 4.7, 1.5, Versicolor, 87
6.3, 2.3, 4.4, 1.3, Versicolor, 88
5.6, 3, 4.1, 1.3, Versicolor, 89
5.5, 2.5, 4, 1.3, Versicolor, 90
5.5, 2.6, 4.4, 1.2, Versicolor, 91
6.1, 3, 4.6, 1.4, Versicolor, 92
5.8, 2.6, 4, 1.2, Versicolor, 93
5, 2.3, 3.3, 1, Versicolor, 94
5.6, 2.7, 4.2, 1.3, Versicolor, 95
5.7, 3, 4.2, 1.2, Versicolor, 96
5.7, 2.9, 4.2, 1.3, Versicolor, 97
6.2, 2.9, 4.3, 1.3, Versicolor, 98
5.1, 2.5, 3, 1.1, Versicolor, 99
5.7, 2.8, 4.1, 1.3, Versicolor, 100
6.3, 3.3, 6, 2.5, Virginica, 101
5.8, 2.7, 5.1, 1.9, Virginica, 102
7.1, 3, 5.9, 2.1, Virginica, 103
6.3, 2.9, 5.6, 1.8, Virginica, 104
6.5, 3, 5.8, 2.2, Virginica, 105
7.6, 3, 6.6, 2.1, Virginica, 106
4.9, 2.5, 4.5, 1.7, Virginica, 107
7.3, 2.9, 6.3, 1.8, Virginica, 108
6.7, 2.5, 5.8, 1.8, Virginica, 109
7.2, 3.6, 6.1, 2.5, Virginica, 110
6.5, 3.2, 5.1, 2, Virginica, 111
6.4, 2.7, 5.3, 1.9, Virginica, 112
6.8, 3, 5.5, 2.1, Virginica, 113
5.7, 2.5, 5, 2, Virginica, 114
5.8, 2.8, 5.1, 2.4, Virginica, 115
6.4, 3.2, 5.3, 2.3, Virginica, 116
6.5, 3, 5.5, 1.8, Virginica, 117
7.7, 3.8, 6.7, 2.2, Virginica, 118
7.7, 2.6, 6.9, 2.3, Virginica, 119
6, 2.2, 5, 1.5, Virginica, 120
6.9, 3.2, 5.7, 2.3, Virginica, 121
5.6, 2.8, 4.9, 2, Virginica, 122
7.7, 2.8, 6.7, 2, Virginica, 123
6.3, 2.7, 4.9, 1.8, Virginica, 124
6.7, 3.3, 5.7, 2.1, Virginica, 125
7.2, 3.2, 6, 1.8, Virginica, 126
6.2, 2.8, 4.8, 1.8, Virginica, 127
6.1, 3, 4.9, 1.8, Virginica, 128
6.4, 2.8, 5.6, 2.1, Virginica, 129
7.2, 3, 5.8, 1.6, Virginica, 130
7.4, 2.8, 6.1, 1.9, Virginica, 131
```

```
7.9, 3.8, 6.4, 2, Virginica, 132
6.4, 2.8, 5.6, 2.2, Virginica, 133
6.3, 2.8, 5.1, 1.5, Virginica, 134
6.1, 2.6, 5.6, 1.4, Virginica, 135
7.7, 3, 6.1, 2.3, Virginica, 136
6.3, 3.4, 5.6, 2.4, Virginica, 137
6.4, 3.1, 5.5, 1.8, Virginica, 138
6, 3, 4.8, 1.8, Virginica, 139
6.9, 3.1, 5.4, 2.1, Virginica, 140
6.7, 3.1, 5.6, 2.4, Virginica, 141
6.9, 3.1, 5.1, 2.3, Virginica, 142
5.8, 2.7, 5.1, 1.9, Virginica, 143
6.8, 3.2, 5.9, 2.3, Virginica, 144
6.7, 3.3, 5.7, 2.5, Virginica, 145
6.7, 3, 5.2, 2.3, Virginica, 146
6.3, 2.5, 5, 1.9, Virginica, 147
6.5, 3, 5.2, 2, Virginica, 148
6.2, 3.4, 5.4, 2.3, Virginica, 149
5.9, 3, 5.1, 1.8, Virginica, 150
];
```

KMeansND - チャート関数

KMeansND() は、K 平均法クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターのクラスター ID を表示します。クラスタリング アルゴリズムで使用される列は、パラメーター coordinate_1、coordinate_2、などによって、最大 n 列まで決定されます。これらはすべて集計です。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。

KMeansND は、データポイントごとに 1 つの値を返します。戻り値はデュアル値であり、各データポイントが割り当てられているクラスターに対応する整数値です。

構文:

```
KMeansND(num_clusters, num_iter, coordinate_1, coordinate_2 [,coordinate_3 [,...]])
```

戻り値データ型: dual

引数:

引数

引数	説明
num_clusters	クラスターの数を指定する整数。
num_iter	再初期化されたクラスター中心を使用したクラスター化の反復回数。
coordinate_1	(チャートから作成できる散布図の) 通常は x 軸の最初の座標を計算する集計。追加のパラメーターは、2 番目、3 番目、4 番目の座標などを計算します。

例: チャートの数式
この例では、*アイリス*データセットを使用して散布図チャートを作成し、KMeans を使用して式でデータに色を付けます。

また、*num_clusters* 引数のための変数を作成し、変数値入力ボックスを使用してクラスターの数を変更します。

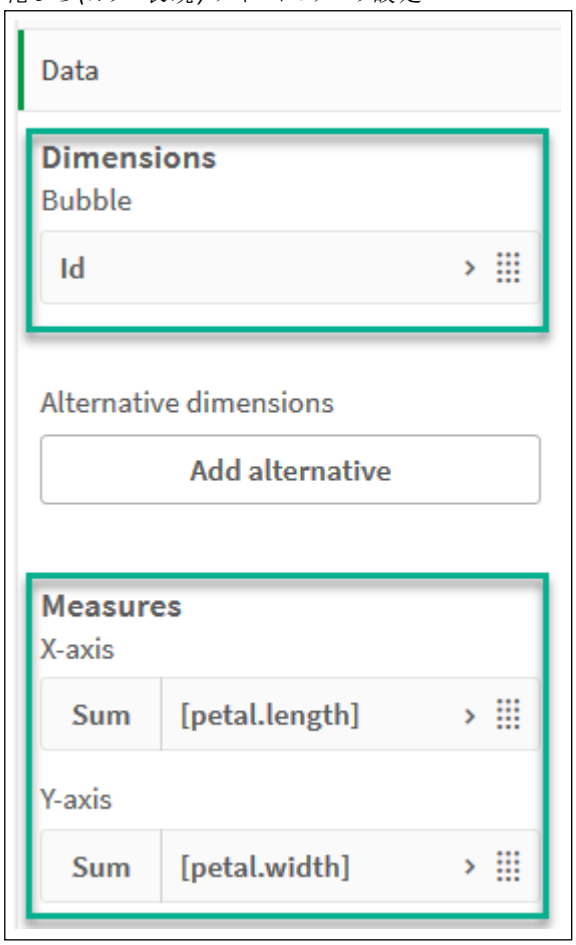
また、*num_iter* 引数のための変数を作成し、2 つ目の変数値入力ボックスを使用して反復の数を変更します。

*アイリス*データセットは様々な形式で公開されています。Qlik Senseのデータロードエディターを使ってロードするインライン テーブルとしてデータを提供しました。この例では、データテーブルに *ID* 列を追加したことに注意してください。

Qlik Senseにデータをロードした後、以下を実施します。

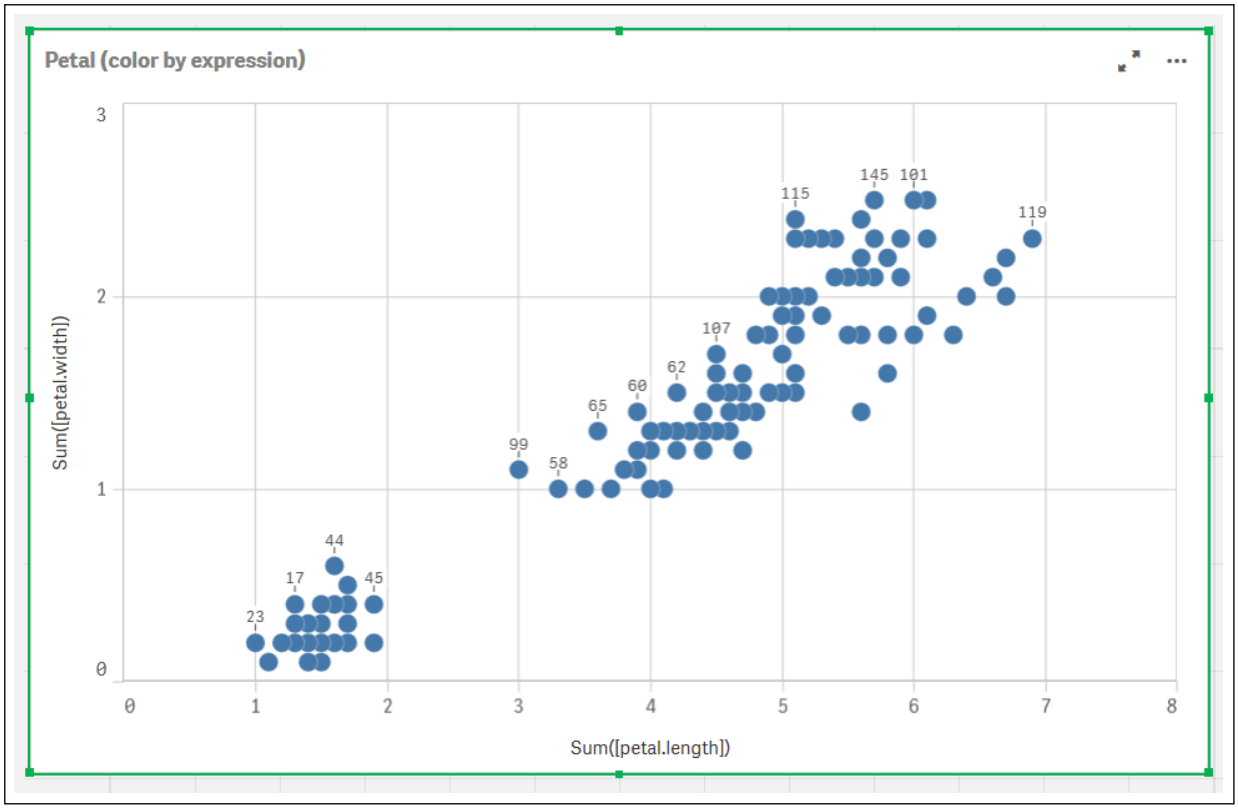
1. 新しいシートに**散布図**チャートをドラッグします。チャート*花びら (カラー表現)*に名前を付けます。
2. クラスターの数を指定する変数を作成します。変数の**名前**には、*KmeansPetalClusters* を入力します。変数の**定義**には、*=2* を入力します。
3. 反復の数を指定する変数を作成します。変数の**名前**には、*KmeansNumberIterations* を入力します。変数の**定義**には、*=1* を入力します。
4. チャートの**データ**を構成:
    i. [**軸**] で、[**バブル**] の項目の [*ID*] を選択します。ラベルのクラスター ID を入力します。
    ii. [**メジャー**]で、**X 軸**の式として *Sum([petal.length])* を選択します。
    iii. [**メジャー**]で、**Y 軸**の式として *Sum([petal.width])* を選択します。

花びら(カラー表現) チャートのデータ設定



データ ポイントがチャートにプロットされ ます。

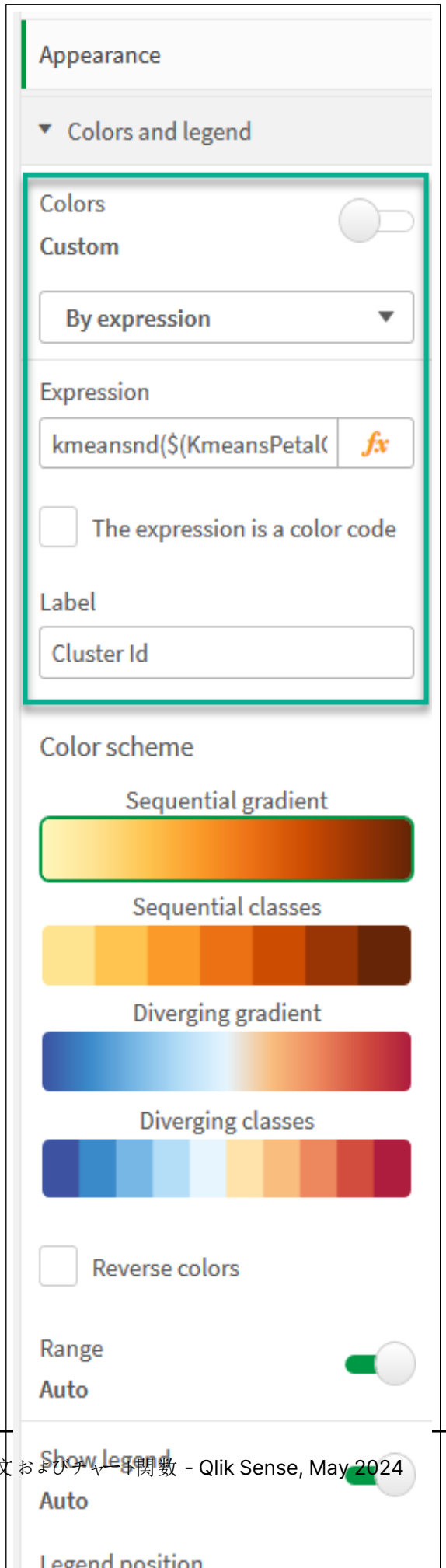
花びら(カラー表現)チャート上のデータポイント



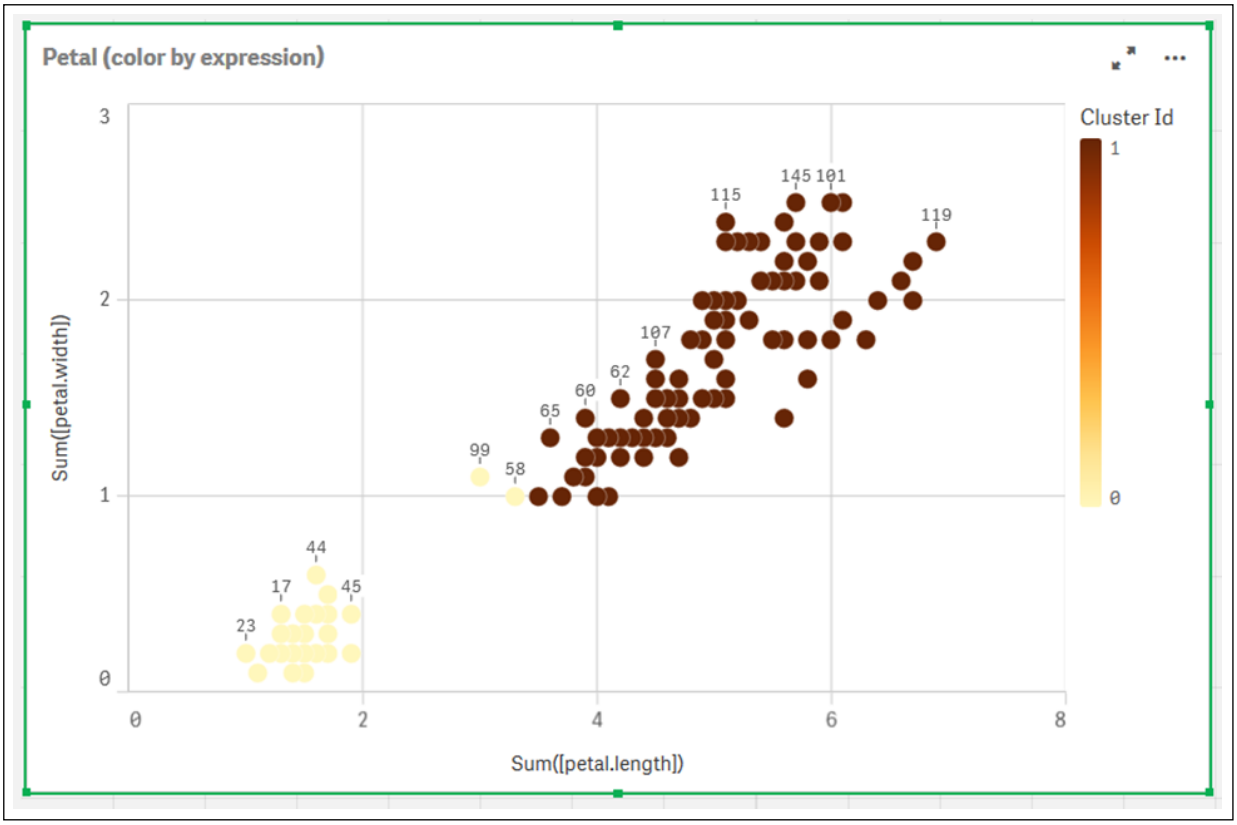
5. チャートの**外観**を構成:
   i. **[色と凡例]** で、**[色]** に**カスタム**を選択します。
   ii. チャートの色を**数式を使用**して選択します。
   iii. **数式**に次のように入力します: *kmeansnd ($(KmeansPetalClusters),$(KmeansNumberIterations), Sum([petal.length]), Sum ([petal.width]),Sum([sepal.length]), Sum([sepal.width]))*
   KmeansPetalClusters は、*2*に設定した変数であることに注意してください。
   KmeansNumberIterations は、*1* に設定した変数です。
   あるいは、次のように入力してください: *kmeansnd(2, 2, Sum([petal.length]), Sum ([petal.width]),Sum([sepal.length]), Sum([sepal.width]))*
   iv. **[数式は色コード]** チェックボックスの選択を解除します。

v. **ラベル**に次を入力します: *クラスター ID*

花びら(数式で色付け) チャートの外観設定


Appearance
Colors and legend
Colors
Custom
By expression
Expression
kmeansnd($(KmeansPetal(
The expression is a color code
Label
Cluster Id
Color scheme
Sequential gradient
Sequential classes
Diverging gradient
Diverging classes
Reverse colors
Range
Auto
Show legend
Auto
Legend position

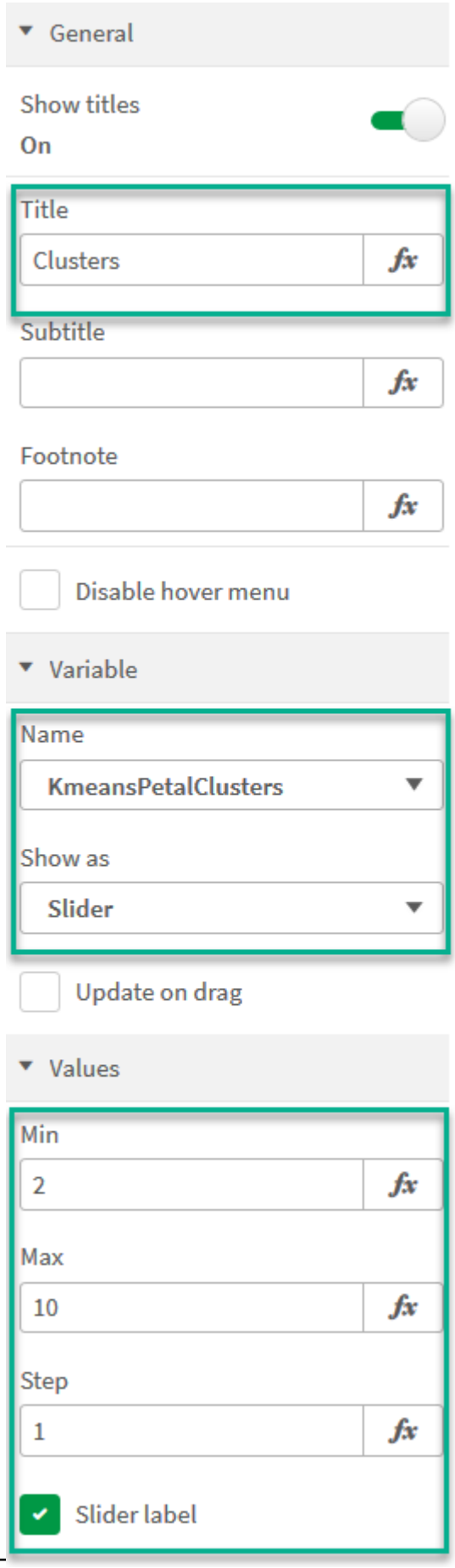
チャート上の 2 つのクラスターは、数式 KMeans によって色分けされています。
花びら(数式で色付け) チャート上の、数式で色付けされたクラスター



6. クラスター数の**変数値入力**ボックスを追加します。
   i. [**アセット**] パネルの [**カスタム オブジェクト**] で、[**Qlik ダッシュボード バンドル**] を選択します。ダッシュボード バンドルにアクセスできない場合でも、作成した変数を使用してクラスター数を変更するか、式の整数として直接変更できます。
   ii. **変数値入力**ボックスをシートへドラッグします。
   iii. [**外観**] で、[**一般**] をクリックします。
   iv. **タイトル**に次を入力します: *クラスター*
   v. [**変数**] をクリックします。
   vi. **名前**で次の変数値を選択します:*KmeansPetalClusters*。
   vii. [**表示の設定**] で [**スライダー**] を選択します。

viii. **値**を選択し、必要に応じて設定を行います。

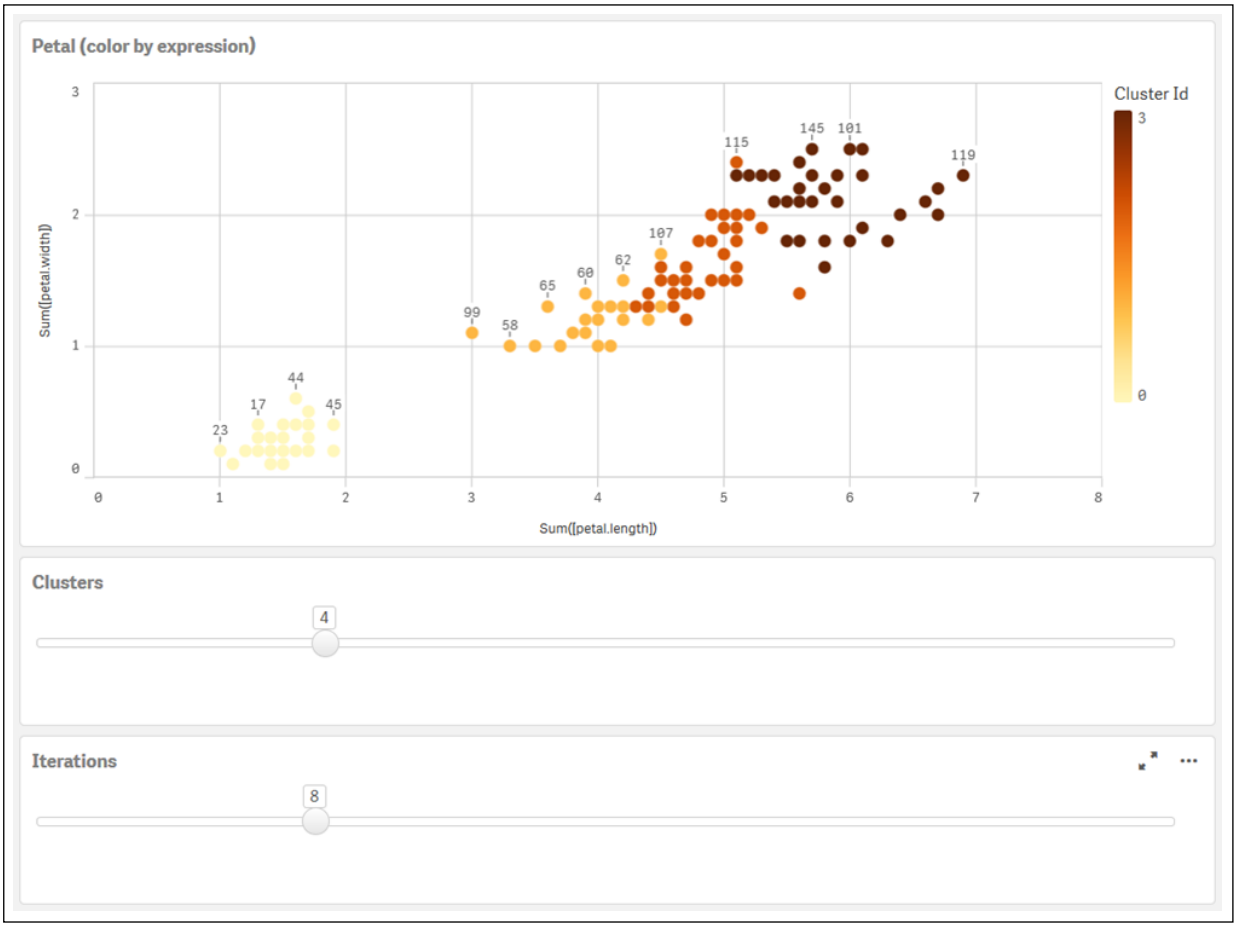
クラスターの外観変数入力ボックス


General
Show titles
On
Title
Clusters
Subtitle
Footnote
Disable hover menu
Variable
Name
KmeansPetalClusters
Show as
Slider
Update on drag
Values
Min
2
Max
10
Step
1
Slider label

7. 反復数の**変数値入力**ボックスを追加します。
   i. **変数値入力**ボックスをシートへドラッグします。
   ii. **[外観]** で、**[一般]** を選択します。
   iii. **タイトル**に次を入力します:*反復*
   iv. **[外観]** で、**[変数]** を選択します。
   v. **名前**で次の変数値を選択します:*KmeansNumberIterations*。
   vi. 必要に応じて追加の設定を行います、

   変数入力ボックスのスライダーを使用して、クラスターの数を変更できるようになりました。

   花びら(数式で色付け)チャート上の、数式で色付けされたクラスター

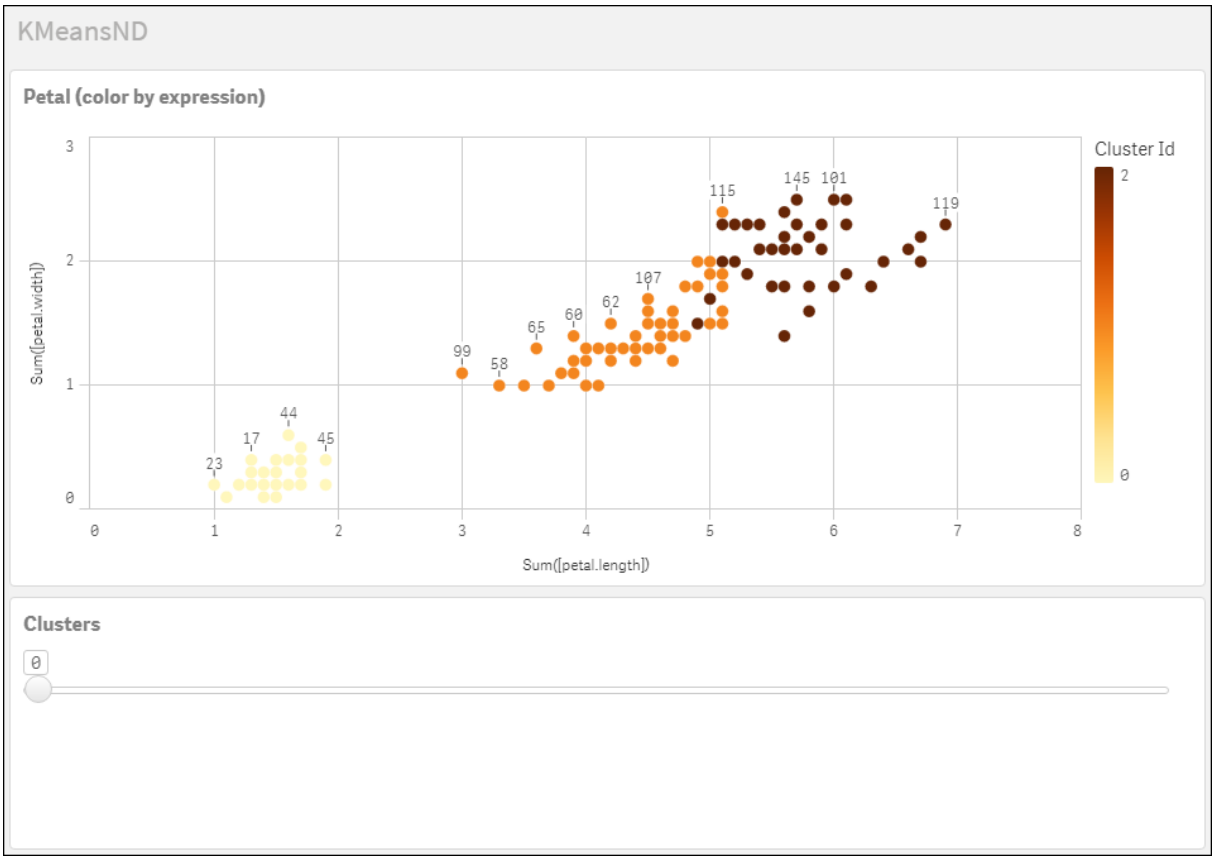


自動クラスタリング

KMeans 関数は、深度差異 (DeD) と呼ばれる方法を使用した自動クラスタリングをサポートします。ユーザーがクラスターの数に 0 を設定すると、そのデータセットに最適なクラスターの数が決定されます。クラスター数 (*k*) の整数は明示的に返されませんが、KMeans アルゴリズム内で計算されることに注意してください。例えば、*KmeansPetalClusters* の値の関数で 0 が指定されている場合、または変数入力ボックスを介して設定されて

いる場合、クラスターの割り当ては、クラスターの最適な数に基づいてデータセットに対して自動的に計算されます。アイリス データセットが与えられた場合、クラスターの数に 0 が選択されている場合、アルゴリズムはこのデータセットに最適なクラスターの数 (3) を決定します (自動クラスター)。

(k) が 0 に設定されている場合、K 平均法深度差異メソッドはクラスターの最適数を決定します。



Iris データ セット:Qlik Sense のデータ ロード エディター用 インライン ロード

```
IrisData:
Load * Inline [
sepal.length, sepal.width, petal.length, petal.width, variety, id
5.1, 3.5, 1.4, 0.2, Setosa, 1
4.9, 3, 1.4, 0.2, Setosa, 2
4.7, 3.2, 1.3, 0.2, Setosa, 3
4.6, 3.1, 1.5, 0.2, Setosa, 4
5, 3.6, 1.4, 0.2, Setosa, 5
5.4, 3.9, 1.7, 0.4, Setosa, 6
4.6, 3.4, 1.4, 0.3, Setosa, 7
5, 3.4, 1.5, 0.2, Setosa, 8
4.4, 2.9, 1.4, 0.2, Setosa, 9
4.9, 3.1, 1.5, 0.1, Setosa, 10
5.4, 3.7, 1.5, 0.2, Setosa, 11
4.8, 3.4, 1.6, 0.2, Setosa, 12
4.8, 3, 1.4, 0.1, Setosa, 13
4.3, 3, 1.1, 0.1, Setosa, 14
5.8, 4, 1.2, 0.2, Setosa, 15
5.7, 4.4, 1.5, 0.4, Setosa, 16
```

```
5.4, 3.9, 1.3, 0.4, Setosa, 17
5.1, 3.5, 1.4, 0.3, Setosa, 18
5.7, 3.8, 1.7, 0.3, Setosa, 19
5.1, 3.8, 1.5, 0.3, Setosa, 20
5.4, 3.4, 1.7, 0.2, Setosa, 21
5.1, 3.7, 1.5, 0.4, Setosa, 22
4.6, 3.6, 1, 0.2, Setosa, 23
5.1, 3.3, 1.7, 0.5, Setosa, 24
4.8, 3.4, 1.9, 0.2, Setosa, 25
5, 3, 1.6, 0.2, Setosa, 26
5, 3.4, 1.6, 0.4, Setosa, 27
5.2, 3.5, 1.5, 0.2, Setosa, 28
5.2, 3.4, 1.4, 0.2, Setosa, 29
4.7, 3.2, 1.6, 0.2, Setosa, 30
4.8, 3.1, 1.6, 0.2, Setosa, 31
5.4, 3.4, 1.5, 0.4, Setosa, 32
5.2, 4.1, 1.5, 0.1, Setosa, 33
5.5, 4.2, 1.4, 0.2, Setosa, 34
4.9, 3.1, 1.5, 0.1, Setosa, 35
5, 3.2, 1.2, 0.2, Setosa, 36
5.5, 3.5, 1.3, 0.2, Setosa, 37
4.9, 3.1, 1.5, 0.1, Setosa, 38
4.4, 3, 1.3, 0.2, Setosa, 39
5.1, 3.4, 1.5, 0.2, Setosa, 40
5, 3.5, 1.3, 0.3, Setosa, 41
4.5, 2.3, 1.3, 0.3, Setosa, 42
4.4, 3.2, 1.3, 0.2, Setosa, 43
5, 3.5, 1.6, 0.6, Setosa, 44
5.1, 3.8, 1.9, 0.4, Setosa, 45
4.8, 3, 1.4, 0.3, Setosa, 46
5.1, 3.8, 1.6, 0.2, Setosa, 47
4.6, 3.2, 1.4, 0.2, Setosa, 48
5.3, 3.7, 1.5, 0.2, Setosa, 49
5, 3.3, 1.4, 0.2, Setosa, 50
7, 3.2, 4.7, 1.4, Versicolor, 51
6.4, 3.2, 4.5, 1.5, Versicolor, 52
6.9, 3.1, 4.9, 1.5, Versicolor, 53
5.5, 2.3, 4, 1.3, Versicolor, 54
6.5, 2.8, 4.6, 1.5, Versicolor, 55
5.7, 2.8, 4.5, 1.3, Versicolor, 56
6.3, 3.3, 4.7, 1.6, Versicolor, 57
4.9, 2.4, 3.3, 1, Versicolor, 58
6.6, 2.9, 4.6, 1.3, Versicolor, 59
5.2, 2.7, 3.9, 1.4, Versicolor, 60
5, 2, 3.5, 1, Versicolor, 61
5.9, 3, 4.2, 1.5, Versicolor, 62
6, 2.2, 4, 1, Versicolor, 63
6.1, 2.9, 4.7, 1.4, Versicolor, 64
5.6, 2.9, 3.6, 1.3, Versicolor, 65
6.7, 3.1, 4.4, 1.4, Versicolor, 66
5.6, 3, 4.5, 1.5, Versicolor, 67
5.8, 2.7, 4.1, 1, Versicolor, 68
6.2, 2.2, 4.5, 1.5, Versicolor, 69
5.6, 2.5, 3.9, 1.1, Versicolor, 70
5.9, 3.2, 4.8, 1.8, Versicolor, 71
```

```
6.1, 2.8, 4, 1.3, Versicolor, 72
6.3, 2.5, 4.9, 1.5, Versicolor, 73
6.1, 2.8, 4.7, 1.2, Versicolor, 74
6.4, 2.9, 4.3, 1.3, Versicolor, 75
6.6, 3, 4.4, 1.4, Versicolor, 76
6.8, 2.8, 4.8, 1.4, Versicolor, 77
6.7, 3, 5, 1.7, Versicolor, 78
6, 2.9, 4.5, 1.5, Versicolor, 79
5.7, 2.6, 3.5, 1, Versicolor, 80
5.5, 2.4, 3.8, 1.1, Versicolor, 81
5.5, 2.4, 3.7, 1, Versicolor, 82
5.8, 2.7, 3.9, 1.2, Versicolor, 83
6, 2.7, 5.1, 1.6, Versicolor, 84
5.4, 3, 4.5, 1.5, Versicolor, 85
6, 3.4, 4.5, 1.6, Versicolor, 86
6.7, 3.1, 4.7, 1.5, Versicolor, 87
6.3, 2.3, 4.4, 1.3, Versicolor, 88
5.6, 3, 4.1, 1.3, Versicolor, 89
5.5, 2.5, 4, 1.3, Versicolor, 90
5.5, 2.6, 4.4, 1.2, Versicolor, 91
6.1, 3, 4.6, 1.4, Versicolor, 92
5.8, 2.6, 4, 1.2, Versicolor, 93
5, 2.3, 3.3, 1, Versicolor, 94
5.6, 2.7, 4.2, 1.3, Versicolor, 95
5.7, 3, 4.2, 1.2, Versicolor, 96
5.7, 2.9, 4.2, 1.3, Versicolor, 97
6.2, 2.9, 4.3, 1.3, Versicolor, 98
5.1, 2.5, 3, 1.1, Versicolor, 99
5.7, 2.8, 4.1, 1.3, Versicolor, 100
6.3, 3.3, 6, 2.5, Virginica, 101
5.8, 2.7, 5.1, 1.9, Virginica, 102
7.1, 3, 5.9, 2.1, Virginica, 103
6.3, 2.9, 5.6, 1.8, Virginica, 104
6.5, 3, 5.8, 2.2, Virginica, 105
7.6, 3, 6.6, 2.1, Virginica, 106
4.9, 2.5, 4.5, 1.7, Virginica, 107
7.3, 2.9, 6.3, 1.8, Virginica, 108
6.7, 2.5, 5.8, 1.8, Virginica, 109
7.2, 3.6, 6.1, 2.5, Virginica, 110
6.5, 3.2, 5.1, 2, Virginica, 111
6.4, 2.7, 5.3, 1.9, Virginica, 112
6.8, 3, 5.5, 2.1, Virginica, 113
5.7, 2.5, 5, 2, Virginica, 114
5.8, 2.8, 5.1, 2.4, Virginica, 115
6.4, 3.2, 5.3, 2.3, Virginica, 116
6.5, 3, 5.5, 1.8, Virginica, 117
7.7, 3.8, 6.7, 2.2, Virginica, 118
7.7, 2.6, 6.9, 2.3, Virginica, 119
6, 2.2, 5, 1.5, Virginica, 120
6.9, 3.2, 5.7, 2.3, Virginica, 121
5.6, 2.8, 4.9, 2, Virginica, 122
7.7, 2.8, 6.7, 2, Virginica, 123
6.3, 2.7, 4.9, 1.8, Virginica, 124
6.7, 3.3, 5.7, 2.1, Virginica, 125
7.2, 3.2, 6, 1.8, Virginica, 126
```

```
6.2, 2.8, 4.8, 1.8, Virginica, 127
6.1, 3, 4.9, 1.8, Virginica, 128
6.4, 2.8, 5.6, 2.1, Virginica, 129
7.2, 3, 5.8, 1.6, Virginica, 130
7.4, 2.8, 6.1, 1.9, Virginica, 131
7.9, 3.8, 6.4, 2, Virginica, 132
6.4, 2.8, 5.6, 2.2, Virginica, 133
6.3, 2.8, 5.1, 1.5, Virginica, 134
6.1, 2.6, 5.6, 1.4, Virginica, 135
7.7, 3, 6.1, 2.3, Virginica, 136
6.3, 3.4, 5.6, 2.4, Virginica, 137
6.4, 3.1, 5.5, 1.8, Virginica, 138
6, 3, 4.8, 1.8, Virginica, 139
6.9, 3.1, 5.4, 2.1, Virginica, 140
6.7, 3.1, 5.6, 2.4, Virginica, 141
6.9, 3.1, 5.1, 2.3, Virginica, 142
5.8, 2.7, 5.1, 1.9, Virginica, 143
6.8, 3.2, 5.9, 2.3, Virginica, 144
6.7, 3.3, 5.7, 2.5, Virginica, 145
6.7, 3, 5.2, 2.3, Virginica, 146
6.3, 2.5, 5, 1.9, Virginica, 147
6.5, 3, 5.2, 2, Virginica, 148
6.2, 3.4, 5.4, 2.3, Virginica, 149
5.9, 3, 5.1, 1.8, Virginica, 150
];
```

KMeansCentroid2D - チャート関数

KMeansCentroid2D() は、K 平均法クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターの目的の座標を表示します。クラスタリング アルゴリズムで使用される列は，それぞれ，パラメーターcoordinate_1 とcoordinate_2 によって決定されます。これらはともに集計です。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。データは、オプションで norm パラメーターによって正規化できます。

KMeansCentroid2D は、データポイントごとに 1 つの値を返します。戻り値はデュアル値であり、データポイントが割り当てられているクラスター中心に対応する位置の座標の 1 つです。

構文:

```
KMeansCentroid2D(num_clusters, coordinate_no, coordinate_1, coordinate_2 [, norm])
```

戻り値データ型: dual

引数:

引数

引数	説明
num_clusters	クラスターの数を指定する整数。
coordinate_no	図心の望ましい座標番号 (たとえば、x、y、または z 軸に対応)。

引数	説明
coordinate_1	チャートから作成できる散布図の最初の座標 (通常は x 軸) を計算する集約です。追加のパラメーターである coordinate_2 は、2 番目の座標を計算します。
norm	KMeans クラスタリングの前にデータセットに適用されるオプションの正規化方法。 考えられる値: 正規化なしの場合は 0 または「なし」 z-score の正規化の場合は 1 または「zscore」 min-max の正規化の場合は 2 または「minmax」 パラメーターが指定されていない場合、または指定されたパラメーターが正しくない場合、正規化は適用されません。 z-score は、機能平均と標準偏差に基づいてデータを正規化します。z-score は、各機能のスケールが同じであることを保証するものではありませんが、外れ値を処理する場合は、min-max よりも優れたアプローチです。 min-max の正規化は、それぞれの最小値と最大値を取得し、各データポイントを再計算することにより、機能が同じスケールを持つことを保証します。

自動クラスタリング

KMeans 関数は、深度差異 (DeD) と呼ばれる方法を使用した自動クラスタリングをサポートします。ユーザーがクラスターの数に 0 を設定すると、そのデータセットに最適なクラスターの数が決定されます。クラスター数 (*k*) の整数は明示的に返されませんが、KMeans アルゴリズム内で計算されることに注意してください。例えば、*KmeansPetalClusters* の値の関数で 0 が指定されている場合、または変数入力ボックスを介して設定されている場合、クラスターの割り当ては、クラスターの最適な数に基づいてデータセットに対して自動的に計算されます。

KMeansCentroidND - チャート関数

KMeansCentroidND() は、K 平均法クラスタリングを適用してチャートの行を評価し、チャートの各行に、このデータポイントが割り当てられているクラスターの目的の座標を表示します。クラスタリング アルゴリズムで使用される列は，パラメーター coordinate_1、coordinate_2、などによって、最大 n 列まで決定されます。これらはすべて集計です。作成されるクラスターの数は、num_clusters パラメーターによって決定されます。

KMeansCentroidND は、行ごとに 1 つの値を返します。戻り値はデュアル値であり、データポイントが割り当てられているクラスター中心に対応する位置の座標の 1 つです。

構文:

```
KMeansCentroidND((num_clusters, num_iter, coordinate_no, coordinate_1,
coordinate_2 [,coordinate_3 [, ...]])
```

戻り値データ型： dual

引数：

引数

引数	説明
num_clusters	クラスターの数を指定する整数。
num_iter	再初期化されたクラスター中心を使用したクラスター化の反復回数。
coordinate_no	図心の望ましい座標番号 (たとえば、x、y、または z 軸に対応)。
coordinate_1	(チャートから作成できる散布図の) 通常は x 軸の最初の座標を計算する集計。追加のパラメーターは、2 番目、3 番目、4 番目の座標などを計算します。

自動クラスタリング

KMeans 関数は、深度差異 (DeD) と呼ばれる方法を使用した自動クラスタリングをサポートします。ユーザーがクラスターの数に 0 を設定すると、そのデータセットに最適なクラスターの数が決定されます。クラスター数 (*k*) の整数は明示的に返されませんが、KMeans アルゴリズム内で計算されることに注意してください。例えば、*KmeansPetalClusters* の値の関数で 0 が指定されている場合、または変数入力ボックスを介して設定されている場合、クラスターの割り当ては、クラスターの最適な数に基づいてデータセットに対して自動的に計算されます。

STL_Trend - チャート関数

STL_Trend は時系列の分解関数です。**STL_Seasonal** と**STL_Residual** と合わせて、この関数は、時系列を季節、トレンド、残差のコンポーネントに分解するために使用します。STL アルゴリズムのコンテキストでは、入力指標と他のパラメータが与えられた場合、繰り返される季節パターンと一般的なトレンドの両方を識別するために時系列分解を使用します。**STL_Trend** 関数は、時系列データの季節パターンやサイクルと関係なく、一般的トレンドを識別します。

3 つの STL 関数は、単純合計を使った入力メトリクスに関連しています。

STL_Trend + **STL_Seasonal** + **STL_Residual** = 入力メトリクス

STL (Loss を使用した季節およびトレンドの分解) では、データ平滑化手法を採用し、入力パラメーターを介して、実行する計算の周期性をユーザーが調整できるようにします。この周期性により、入力メトリクス (メジャー) の時間軸が分析でセグメント化される方法を決定します。

少なくとも、**STL_Trend** は `period_int` のために入力メトリクス (`target_measure`) と整数値を取得し、浮動小数値を返します。入力メトリクスは、時間軸に応じた集計の形式になります。オプションで、`seasonal_smoother` と `trend_smoother` の値を含めて、平滑化アルゴリズムを調整することができます。

チャートの数式エディターに直接入力することで、この関数を使用できます。

構文：

```
STL_Trend(target_measure, period_int [,seasonal_smoother [,trend_smoother]])
```

戻り値データ型： デュアル

引数

引数	説明
target_measure	季節コンポーネントとトレンドコンポーネントに分解するメジャー。これは、時間軸に沿って異なるSum(Sales) または Sum(Passengers) などのメジャーである必要があります。 これは定数値にできません。
period_int	データセットの周期性。このパラメータは、信号の1周期 (季節サイクル) を構成する離散ステップの数を表す整数値です。 例えば、時系列が1年の四半期ごとに1つのセクションに分割されている場合、周期性を「年」と定義するために、**period_int** を4という値に設定する必要があります。
seasonal_smoother	季節性スムーザーの長さ。これは偶数の整数である必要があります。季節性スムーザーは、一定期間数の季節性変数において特定の段階のデータを使用します。時間軸の1つの離散ステップは、各期間から使用されます。季節性スムーザーは、スムージングに使用される期間数を示しています。 例えば、時間軸を月でセグメント化し、期間を年 (12) とすると、季節コンポーネントは、各年の特定の月がその年と隣接する年の同じ月のデータから算出されるように計算されます。**seasonal_smoother** の値は、スムージングに使用される年数です。
trend_smoother	トレンドスムーザーの長さ。これは偶数の整数である必要があります。トレンドスムーサーは、**period_int** パラメータと同じ時間スケールを使用し、その値はスムージングに使用される粒子の数です。 たとえば、時間系列が月でセグメント化している場合、トレンドスムーザーはスムージングに使用される月数となります。

[**STL_Trend**] チャート関数は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
STL_Seasonal - チャート関数 (page 1397)	これは、時系列の季節コンポーネントを計算するのに使用される関数です。

関数	相互作用
STL_Residual - チャート関数 (page 1399)	入力メトリクスを季節およびトレンドコンポーネントに分割する際、メジャーのバリエーションの一部は 2 つの主要コンポーネントに当てはまりません。**STL_Residual** 関数は、分解のこの部分を計算します。

この関数を使用する方法を示した詳しい例によるチュートリアルは、*チュートリアル - Qlik Sense の時系列の分解 (page 1401)*を参照してください。

STL_Seasonal - チャート関数

STL_Seasonal は時系列の分解関数です。**STL_Trend** と**STL_Residual** と合わせて、この関数は、時系列を季節、トレンド、残差のコンポーネントに分解するために使用します。STL アルゴリズムのコンテキストでは、入力指標と他のパラメータが与えられた場合、繰り返される季節パターンと一般的なトレンドの両方を識別するために時系列分解を使用します。**STL_Seasonal** 関数は、データに表示される一般的トレンドと区別しながら、時系列内の季節パターンを特定します。

3 つの STL 関数は、単純合計を使った入力メトリクスに関連しています。

STL_Trend + **STL_Seasonal** + **STL_Residual** = 入力メトリクス

STL (Loss を使用した季節およびトレンドの分解) では、データ平滑化手法を採用し、入力パラメーターを介して、実行する計算の周期性をユーザーが調整できるようにします。この周期性により、入力メトリクス (メジャー) の時間軸が分析でセグメント化される方法を決定します。

少なくとも、**STL_Seasonal** は `period_int` のために入力メトリクス (`target_measure`) と整数値を取得し、浮動小数値を返します。入力メトリクスは、時間軸に応じた集計の形式になります。オプションで、`seasonal_smoother` と `trend_smoother` の値を含めて、平滑化アルゴリズムを調整することができます。

チャートの数式エディターに直接入力することで、この関数を使用できます。

構文:

```
STL_Seasonal(target_measure, period_int [,seasonal_smoother [,trend_smoother]])
```

戻り値データ型： デュアル

引数

引数	説明
target_measure	季節コンポーネントとトレンドコンポーネントに分解するメジャー。これは、時間軸に沿って異なるSum(Sales) または Sum(Passengers) などのメジャーである必要があります。 これは定数値にできません。
period_int	データセットの周期性。このパラメータは、信号の1周期 (季節サイクル) を構成する離散ステップの数を表す整数値です。 例えば、時系列が1年の四半期ごとに1つのセクションに分割されている場合、周期性を「年」と定義するために、**period_int** を4という値に設定する必要があります。
seasonal_smoother	季節性スムーザーの長さ。これは偶数の整数である必要があります。季節性スムーザーは、一定期間数の季節性変数において特定の段階のデータを使用します。時間軸の1つの離散ステップは、各期間から使用されます。季節性スムーザーは、スムージングに使用される期間数を示しています。 例えば、時間軸を月でセグメント化し、期間を年 (12) とすると、季節コンポーネントは、各年の特定の月がその年と隣接する年の同じ月のデータから算出されるように計算されます。**seasonal_smoother** の値は、スムージングに使用される年数です。
trend_smoother	トレンドスムーザーの長さ。これは偶数の整数である必要があります。トレンドスムーサーは、**period_int** パラメータと同じ時間スケールを使用し、その値はスムージングに使用される粒子の数です。 たとえば、時間系列が月でセグメント化している場合、トレンドスムーザーはスムージングに使用される月数となります。

[**STL_Seasonal**] チャート関数は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
STL_Trend - チャート関数 (page 1395)	これは、時系列のトレンドコンポーネントを計算するのに使用される関数です。

関数	相互作用
STL_Residual - チャート関数 (page 1399)	入力メトリクスを季節およびトレンドコンポーネントに分割する際、メジャーのバリエーションの一部は 2 つの主要コンポーネントに当てはまりません。**STL_Residual** 関数は、分解のこの部分を計算します。

この関数を使用する方法を示した詳しい例によるチュートリアルは、*チュートリアル - Qlik Sense の時系列の分解 (page 1401)*を参照してください。

STL_Residual - チャート関数

STL_Residual は時系列の分解関数です。**STL_Seasonal** と**STL_Trend** と合わせて、この関数は、時系列を季節、トレンド、残差のコンポーネントに分解するために使用します。STL アルゴリズムのコンテキストでは、入力指標と他のパラメータが与えられた場合、繰り返される季節パターンと一般的なトレンドの両方を識別するために時系列分解を使用します。この演算を実行すると、季節コンポーネントまたはトレンドコンポーネントのいずれにも当てはまらない入力メトリクスの変動の一部が、残差コンポーネントとして定義されます。**STL_Residual** チャート関数は、計算のこの部分を捕捉します。

3 つの STL 関数は、単純合計を使った入力メトリクスに関連しています。

STL_Trend + **STL_Seasonal** + **STL_Residual** = 入力メトリクス

STL (Loss を使用した季節およびトレンドの分解) では、データ平滑化手法を採用し、入力パラメーターを介して、実行する計算の周期性をユーザーが調整できるようにします。この周期性により、入力メトリクス (メジャー) の時間軸が分析でセグメント化される方法を決定します。

時系列分解は主にデータ内の季節性と一般的変動を検索するため、残差内の情報は、3 つのコンポーネントのうちで重要性が最も低いものとみなされます。しかし、歪んだ残差コンポーネント、または周期的残差コンポーネントは、誤った周期性の設定など、計算上の問題を識別するために役立ちます。

少なくとも、**STL_Residual** は `period_int` のために入力メトリクス (`target_measure`) と整数値を取得し、浮動小数値を返します。入力メトリクスは、時間軸に応じた集計の形式になります。オプションで、`seasonal_smoother` と`trend_smoother` の値を含めて、平滑化アルゴリズムを調整することができます。

チャートの数式エディターに直接入力することで、この関数を使用できます。

構文:

```
STL_Residual(target_measure, period_int [,seasonal_smoother [,trend_
smoother]])
```

戻り値データ型: デュアル

引数

引数	説明
target_measure	季節コンポーネントとトレンドコンポーネントに分解するメジャー。これは、時間軸に沿って異なるSum(Sales) または Sum(Passengers) などのメジャーである必要があります。 これは定数値にできません。
period_int	データセットの周期性。このパラメータは、信号の1周期 (季節サイクル) を構成する離散ステップの数を表す整数値です。 例えば、時系列が1年の四半期ごとに1つのセクションに分割されている場合、周期性を「年」と定義するために、**period_int** を4という値に設定する必要があります。
seasonal_smoother	季節性スムーザーの長さ。これは偶数の整数である必要があります。季節性スムーザーは、一定期間数の季節性変数において特定の段階のデータを使用します。時間軸の1つの離散ステップは、各期間から使用されます。季節性スムーザーは、スムージングに使用される期間数を示しています。 例えば、時間軸を月でセグメント化し、期間を年 (12) とすると、季節コンポーネントは、各年の特定の月がその年と隣接する年の同じ月のデータから算出されるように計算されます。**seasonal_smoother** の値は、スムージングに使用される年数です。
trend_smoother	トレンドスムーザーの長さ。これは偶数の整数である必要があります。トレンドスムーサーは、**period_int** パラメータと同じ時間スケールを使用し、その値はスムージングに使用される粒子の数です。 たとえば、時間系列が月でセグメント化している場合、トレンドスムーザーはスムージングに使用される月数となります。

[STL_Residual] チャート関数は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
STL_Seasonal - チャート関数 (page 1397)	これは、時系列の季節コンポーネントを計算するのに使用される関数です。

関数	相互作用
STL_Trend - チャート関数 (page 1395)	これは、時系列のトレンドコンポーネントを計算するのに使用される関数です。

この関数を使用する方法を示した詳しい例によるチュートリアルは、*チュートリアル - Qlik Sense の時系列の分解 (page 1401)*を参照してください。

チュートリアル - Qlik Sense の時系列の分解

このチュートリアルでは、STL アルゴリズムを使って時系列を分解する 3 つのチャート関数の使用をお見せします。

このチュートリアルでは、1 ヶ月あたりの航空会社利用乗客数の時系列データを用いて、STL アルゴリズムの機能を説明します。**STL_Trend**、**STL_Seasonal**、および **STL_Residual** チャート関数を使用してビジュアライゼーションを作成します。Qlik Sense における時系列分解の詳細については、*時系列分解の関数 (page 1348)* を参照してください。

アプリを作成する

新しいアプリを作成して、データセットをインポートすることから始めます。

このデータセットをダウンロード:

チュートリアル - 時系列の分解

このファイルには、1 ヶ月当たりの空港会社の乗客数に関するデータが含まれます。

次の手順を実行します。

1. ハブから、[**アプリの新規作成**] をクリックします。
2. アプリを開き、*Tutorial - Time series decomposition.csv* ファイルをアプリにドロップします。

データを準備してロードする

Qlik Sense が YearMonth 項目を正しく解釈するためには、データマネージャーを使用して、項目を文字列値を持つ項目ではなく、日付項目として認識させる必要がある場合があります。通常、このステップは自動的に処理されますが、この場合は日付があまり使用されない *YYYY-MM* 形式で表示されています。

1. [データマネージャー] で、テーブルを選択し、 をクリックします。
2. [*YearMonth*] 項目を選択した状態で、 をクリックして、[**項目タイプ**] を [**日付**] に設定します。
3. [**入力形式**] に、*YYYY-MM* と入力します。
4. [**表示形式**] に、*YYYY-MM* と入力してから、[**OK**] をクリックします。
   項目には、カレンダーアイコンが表示されるようになります。
5. [**データのロード**] をクリックします。

これで、STL 関数を使ってデータを視覚的に表示し始めることができます。

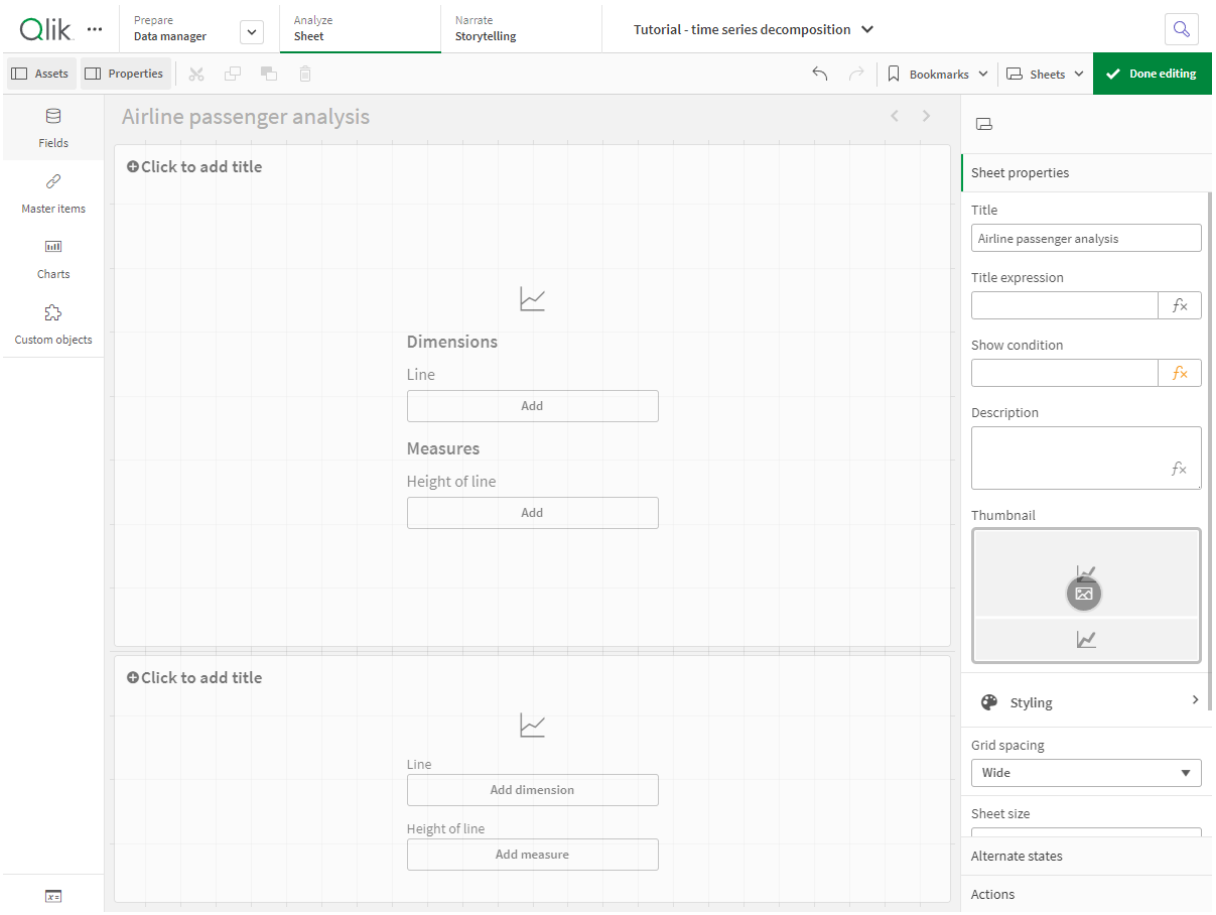
ビジュアライゼーションの作成

次に、**STL_Trend**、**STL_Seasonal**、**STL_Residual** チャート関数の機能を示す、2 つの折れ線グラフを作成します。

新しいシートを開いて、タイトルを付けます。

シートに 2 つの折れ線グラフを追加します。次の画像に合わせて、チャートのサイズと位置を変更します。

Qlik Sense ブランク アプリシートのグリッド輪郭



1 番目の折れ線グラフ: トレンドおよび季節コンポーネント

次の手順を実行します。

1. 1 番目の折れ線グラフに、*Seasonal and Trend* というタイトルを付けます。
2. *YearMonth* を軸として追加し、*Date* というラベルを付けます。
3. 以下のメジャーを追加して、*Passengers per month* というラベルを付けます。
   =Sum(Passengers)
4. [**データ**] で、*Passengers per month* メジャーを展開して、[**トレンド線を追加**] をクリックします。
5. [**タイプ**] を [**線形**] に設定します。

このトレンド線を、トレンドコンポーネントのスムージングされた出力と比較します。

6. 以下のメジャーを追加して、トレンドコンポーネントを分布させ、*Trend* というラベルを付けます。
   =STL_Trend(SUM(Passengers), 12)
7. 次に、以下のメジャーを追加して、季節コンポーネントを分布させ、*Seasonal* というラベルを付けます。
   =STL_Seasonal(SUM(Passengers), 12)
8. **[スタイル]** > **[プレゼンテーション]** で、**[スクロール バー]** を **[なし]** に設定します。
9. 既定の色を使うか、または好みに合わせて変更します。

2 番目の折れ線グラフ: 残差コンポーネント

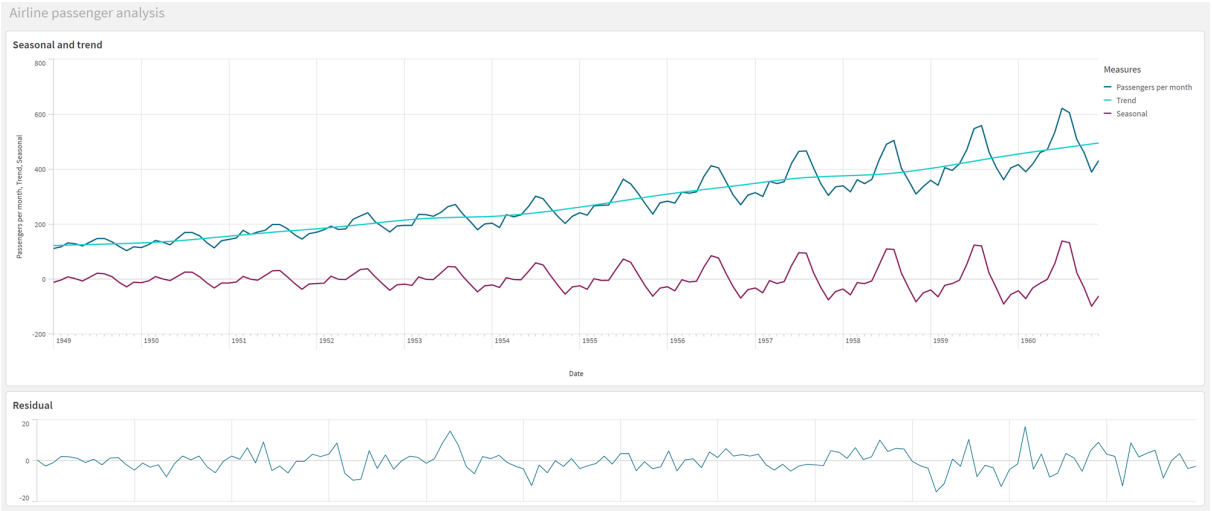
次に、2 番目の折れ線グラフを設定します。このビジュアライゼーションでは、時系列の残差コンポーネントを表示します。

次の手順を実行します。

1. シートに折れ線グラフをドラッグします。*Residual* というタイトルを付けます。
2. *Date* を軸として追加します。
3. 次のメジャーを追加して、*Residual* というラベルを付けます:
   =STL_Residual(SUM(Passengers), 12)
4. **[スタイル]** > **[プレゼンテーション]** で、**[スクロール バー]** を **[なし]** に設定します。

シートは次のようになると思います。

航空会社乗客分析のための Qlik Sense シート



データの解釈と説明

STL チャート関数を使うと、時系列データから多数のインサイトを獲得できます。

トレンドコンポーネント

トレンドコンポーネントの統計情報は非季節化されています。これにより、経時的な一般的、非反復的変動を確認しやすくなります。*Passengers per month* の直線的なトレンド線と比較すると、STL のトレンドコンポーネントはトレンドの変化をよく捉えています。読みやすい方法で、明確な偏差値を表示します。STL アルゴリズムのスムージング動作により、これがキャプチャされます。

STL のトレンドグラフに見られる航空会社旅客数の減少は、1950 年代に発生した不況の経済的影響の一部として説明できます。

季節性コンポーネント

トレンド除去された季節コンポーネントは、時系列全体で反復される変動を分離し、分析の部分から一般的なトレンド情報を除去しました。年 ·月集計で構成されたデータセットから開始します。このデータの場合、1 ヶ月単位でデータをセグメント化分割していることが暗黙の了解です。期間値を 12 と定義することにより、1 年 (12 ヶ月) サイクルの季節パターンをモデル化するように設定します。

データ上では、航空乗客数が夏場に急増し、冬場は減少するという季節的パターンが繰り返されています。これは、夏場は通常休暇を取ったり旅行したりする人が多いということに合致しています。また、時系列で見ると、これらの季節サイクルの振幅が急激に大きくなっていることがわかります。

残差コンポーネント

残差コンポーネントのチャートには、トレンドおよび季節コンポーネントでキャプチャされなかった情報がすべて表示されます。残差コンポーネントには統計ノイズが含まれますが、STL トレンドや季節関数の引数の設定が正しくないことを示す場合もあります。一般的に、信号の残差コンポーネントに周期的振動がある場合や、表示される情報が明らかにランダムでない場合、通常、季節コンポーネントやトレンドコンポーネントで現在キャプチャされていない情報が時系列に存在することを示します。この場合、各関数の引数を再確認して、期間を変更しなければならない場合があります。

スムーザー値

トレンドおよび季節性スムーザーの値を指定しなかったため、関数はこれらのパラメータの既定値を使用します。Qlik Sense で、STL アルゴリズムの既定スムーザー値は効果的な結果を生成します。その結果、ほとんどの場合、これらの引数は式から省略できます。

> *3 つの STL 関数のいずれかで、季節性またはトレンドのスムーザーを 0 と設定すると、アルゴリズムは 0 ではなく既定値を使用するようになります。*

トレンドスムーザー値は、チャートで指定された軸を使用します。[*YearMonth*] 項目ではデータが月別に表示されるため、トレンドスムーザー値は月数となります。季節性スムーザーは、定義された期間を反映します。この場合、1 期間を 12 ヶ月 (1 年) と定義したため、季節性スムーザーの値は年数になります。少々わかりにくいかもしれませんが、季節性を見出すためには、多くの季節を確認する必要があるということです。この数値が季節性スムーザーです。

その他の有用な情報

季節サイクルは経時的に振幅が大きくなることから、より高度な分析アプローチを使うと、対数関数を用いて乗法的な分解を行うことができます。実際には、季節コンポーネントをトレンドコンポーネントで除算することにより、相対的な振幅の簡単な尺度を Qlik Sense に作成することができます。これを実行すると、時間の経過とともに、各周期の夏場のピークが相対的に大きくなっていることがわかります。しかし、冬場の低ポイントの振幅は、時間が経っても大きくなりません。

8.23 統計的分布関数

統計分布関数は、所定の入力変数に対して、考えられるさまざまな結果が発生する確率を返します。これらの関数を使って、データポイントの潜在的値を計算できます。

下記の 3 つのグループの統計的分布関数は、すべて Cephes 関数ライブラリを使用して Qlik Sense に実装されています。使用されるアルゴリズム、精度などの詳細な参照文献は、Cephes library に掲載されています。Cephes 関数ライブラリの使用には、許可が必要です。

- 確率関数は、供給された値による分布の点での確率を計算します。
  - 頻度関数は、離散分布に対して使用されます。
  - 密度関数は、継続関数に対して使用されます。
- 分布関数は、供給された値による分布の点での累積確率を計算します。
- INV 関数は、分布の所定の累積確率の逆関数の値を計算します。

すべての関数は、データロードスクリプトおよびチャートの数式の両方で使用できます。

統計的分布関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

BetaDensity

`BetaDensity()` は、ベータ分布の確率を返します。

```
BetaDensity (value, alpha, beta)
```

BetaDist

`BetaDist()` は、ベータ分布の累計確率を返します。

```
BetaDist (value, alpha, beta)
```

BetaInv

`BetaINV()` は、ベータ分布の累計確率の逆関数を返します。

```
BetaInv (prob, alpha, beta)
```

BinomDist

`BinomDist()` は、2 項分布の累計確率を返します。

```
BinomDist (value, trials, trial_probability)
```

BinomFrequency

`BinomFrequency()` は、2 項確率分布を返します。

```
BinomFrequency (value, trials, trial_probability)
```

BinomInv

`BinomInv()` は、2 項分布の累計確率の逆関数を返します。

```
BinomInv (prob, trials, trial_probability)
```

ChiDensity

ChiDensity() は、カイ 2 分布の片側確率を返します。カイ2 密度関数は、カイ2 テストに関連付けられています。

```
ChiDensity (value, degrees_freedom)
```

ChiDist

ChiDist() は、分布の片側確率を返します。カイ2 分布は、カイ2 テストに関連付けられています。

```
ChiDist (value, degrees_freedom)
```

ChiInv

ChiInv() は、chi2 分布の片側確率の逆関数の値を返します。

```
ChiInv (prob, degrees_freedom)
```

FDensity

FDensity() は、F 分布の確率を返します。

```
FDensity (value, degrees_freedom1, degrees_freedom2)
```

FDist

FDist() は、F 分布の累計確率を返します。

```
FDist (value, degrees_freedom1, degrees_freedom2)
```

FInv

FInv() は、F 分布の累計確率の逆関数を返します。

```
FInv (prob, degrees_freedom1, degrees_freedom2)
```

GammaDensity

GammaDensity() は、ガンマ分布の確率を返します。

```
GammaDensity (value, k, θ)
```

GammaDist

GammaDist() は、ガンマ分布の累計確率を返します。

```
GammaDist (value, k, θ)
```

GammaInv

GammaInv() は、ガンマ分布の累計確率の逆関数を返します。

```
GammaInv (prob, k, θ)
```

NormDist

NormDist() は、指定された平均と標準偏差について、累積正規分布を返します。例えば、mean = 0、standard_dev = 1 の場合は、標準正規分布の値が返されます。

```
NormDist (value, mean, standard_dev)
```

NormInv

NormInv() は、指定された平均と標準偏差について、累積正規分布の逆関数分布を返します。

```
NormInv (prob, mean, standard_dev)
```

PoissonDist

PoissonDist() は、ガンマ分布の累計確率を返します。

```
PoissonDist (value, mean)
```

PoissonFrequency

PoissonFrequency() は、ポアソン確率分布を返します。

```
PoissonFrequency (value, mean)
```

PoissonInv

PoissonInv() は、Gamma distribution分布の累計確率の逆関数を返します。

```
PoissonInv (prob, mean)
```

TDensity

TDensity() は、スチューデントの t 密度関数の値を返します。ここでの数値は t の計算値であり、この値に対して確率が計算されます。

```
TDensity (value, degrees_freedom, tails)
```

TDist

TDist() は、スチューデント t 分布における確率を返します。ここでの数値は t の計算値であり、この値に対して確率が計算されます。

```
TDist (value, degrees_freedom, tails)
```

TInv

TInv() は、スチューデント t 分布の t 値を確率と自由度の関数として返します。

```
TInv (prob, degrees_freedom)
```

参照先:

- *統計集計関数 (page 393)*

BetaDensity

BetaDensity() は、ベータ分布の確率を返します。

構文:

```
BetaDensity(value, alpha, beta)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は 0 ~ 1 である必要があります。
alpha	最初の形状パラメータを定義する正の数値。確率変数の指数です
beta	2 番目の形状パラメータを定義する正の数値。分母の自由度を数値で示します。

BetaDist

BetaDist() は、ベータ分布の累計確率を返します。

構文:

```
BetaDist(value, alpha, beta)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は 0 ~ 1 である必要があります。
alpha	最初の形状パラメータを定義する正の数値。確率変数の指数です
beta	2 番目の形状パラメータを定義する正の数値。分布の形状を制御する指数です。

この関数は、次のように BetaInv 関数に関連します。
If prob = BetaDist(value, alpha, beta), then BetaInv(prob, alpha, beta) = value

BetaInv

BetaINV() は、ベータ分布の累計確率の逆関数を返します。

構文:

```
BetaInv(prob, alpha, beta)
```

戻り値データ型: 数値

引数

引数	説明
prob	ベータ確率分布に関連付けられた確率。0 ~ 1 の数値を指定します。
alpha	最初の形状パラメータを定義する正の数値。確率変数の指数です
beta	2 番目の形状パラメータを定義する正の数値。分布の形状を制御する指数です。

この関数は、次のように BetaDist 関数に関連します。
If prob = BetaDist(value, alpha, beta), then BetaInv(prob, alpha, beta) = value

BinomDist

BinomDist() は、2 項分布の累計確率を返します。

構文:

```
BinomDist(value, trials, trial_probability)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。この値は 0 より大きく、かつ試行回数未満の整数となります。
trials	試行回数を示す正の整数。
trial_probability	各試行の成功確率。必ず 0 ~ 1 の数値となります。

この関数は、次のように BinomInv 関数に関連します。
If prob = BinomDIST(value, trials, trial_probability), then BinomInv(prob, trials, trial_probability) = value

BinomFrequency

BinomFrequency() は、2 項確率分布を返します。

構文:

```
BinomFrequency(value, trials, trial_probability)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。この値は 0 より大きく、かつ試行回数未満の整数となります。
trials	試行回数を示す正の整数
trial_probability	各試行の成功確率。必ず 0 ~ 1 の数値となります。

BinomInv

BinomInv() は、2 項分布の累計確率の逆関数を返します。

構文:

```
BinomInv(prob, trials, trial_probability)
```

戻り値データ型: 数値

引数

引数	説明
prob	2 項確率分布に関連付けられた確率。0 ~ 1 の数値を指定します。
trials	試行回数を示す正の整数。
trial_probability	各試行の成功確率。必ず 0 ~ 1 の数値となります。

この関数は、次のように BinomDist 関数に関連します。
If prob = BinomDist(value, trials, trial_probability), then BinomInv(prob, trials, trial_probability) = value

ChiDensity

ChiDensity() は、カイ 2 分布の片側確率を返します。カイ2 密度関数は、カイ2 テストに関連付けられています。

構文:

```
ChiDensity(value, degrees_freedom)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
degrees_freedom	分子の自由度を数値で示す正の整数です。

ChiDist

ChiDist() は、分布の片側確率を返します。カイ2 分布は、カイ2 テストに関連付けられています。

構文:

```
CHIDIST(value, degrees_freedom)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
degrees_freedom	自由度を数値で示す正の整数です。

この関数は、次のように **ChiInv** 関数に関連します。
`If prob = CHIDIST(value,df), then CHIINV(prob, df) = value`

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

例と結果:

例	結果
CHIDIST(8, 15)	0.9238 を返します

ChiInv

`ChiInv()` は、chi^2 分布の片側確率の逆関数の値を返します。

構文:

```
CHIINV(prob, degrees_freedom)
```

戻り値データ型: 数値

引数:

引数

引数	説明
prob	chi^2 分布に関連付けられた確率。0 ~ 1 の数値を指定します。
degrees_freedom	自由度を数値で示す整数です。

この関数は、次のように **ChiDist** 関数に関連します。
`If prob = CHIDIST(value,df), then CHIINV(prob, df) = value`

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

例と結果:

例	結果
CHIINV(0.9237827, 15)	8.0000 を返します

FDensity

`FDensity()` は、F 分布の確率を返します。

構文:

```
FDensity(value, degrees_freedom1, degrees_freedom2)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
degrees_freedom1	分子の自由度を数値で示す正の整数です。
degrees_freedom2	分母の自由度を数値で示す正の整数です。

FDist

FDist() は、F 分布の累計確率を返します。

構文:

```
FDist(value, degrees_freedom1, degrees_freedom2)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
degrees_freedom1	分子の自由度を数値で示す正の整数です。
degrees_freedom2	分母の自由度を数値で示す正の整数です。

この関数は、次のように **FInv** 関数に関連します。

```
If prob = FDIST(value, df1, df2), then FINV(prob, df1, df2) = value
```

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

例と結果:

例	結果
FDIST(15, 8, 6)	0.0019 を返します

FInv

FInv() は、F 分布の累計確率の逆関数を返します。

構文:

```
FInv(prob, degrees_freedom1, degrees_freedom2)
```

戻り値データ型: 数値

引数:

引数

引数	説明
prob	F 分布に関連する確率で、0 ~ 1 の間の数値を指定します。
degrees_freedom	自由度を数値で示す整数です。

この関数は、次のように **FDist** 関数に関連します。
`If prob = FDIST(value, df1, df2), then FINV(prob, df1, df2) = value`

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

例と結果:

例	結果
FINV(0.0019369, 8, 6)	15.0000 を返します

GammaDensity

`GammaDensity()` は、ガンマ分布の確率を返します。

構文:

```
GammaDensity(value, k, θ)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
k	形状パラメータを定義する正の数値。
θ	スケール パラメータを定義する正の数値。

GammaDist

`GammaDist()` は、ガンマ分布の累計確率を返します。

構文:

```
GammaDist(value, k, θ)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
k	形状パラメータを定義する正の数値。
θ	スケール パラメータを定義する正の数値。

この関数は、次のように GammaINV 関数に関連します。
`If prob = GammaDist(value, k, θ), then GammaInv(prob, k, θ) = value`

GammaInv

`GammaInv()` は、ガンマ分布の累計確率の逆関数を返します。

構文:

```
GammaInv(prob, k, θ)
```

戻り値データ型: 数値

引数

引数	説明
prob	ガンマ確率分布に関連付けられた確率。0 ~ 1 の数値を指定します。
k	形状パラメータを定義する正の数値。
θ	スケール パラメータを定義する正の数値。

この関数は、次のように GammaDist 関数に関連します。
`If prob = GammaDist(value, k, θ), then GammaInv(prob, k, θ) = value`

NormDist

`NormDist()` は、指定された平均と標準偏差について、累積正規分布を返します。例えば、mean = 0、standard_dev = 1 の場合は、標準正規分布の値が返されます。

構文:

```
NORMDIST(value, [mean], [standard_dev], [cumulative])
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	分布を評価する値。
mean	分布の算術平均を示す、省略可能な値です。 この引数を指定しない場合、既定値は 0 です。
standard_dev	分布の標準偏差を示す、省略可能な正の値です。 この引数を指定しない場合、既定値は 1 です。
cumulative	任意で標準正規分布または累積分布を使用することも選択できます。 0 = 標準正規分布 1 = 累積分布 (既定)

この関数は、次のように **NormInv** 関数に関連します。
`If prob = NORMDIST(value, m, sd), then NORMINV(prob, m, sd) = value`

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

例と結果:

例	結果
NORMDIST(0.5, 0, 1)	0.6915 を返します

NormInv

`NormInv()` は、指定された平均と標準偏差について、累積正規分布の逆関数分布を返します。

構文:
```
NORMINV(prob, mean, standard_dev)
```

戻り値データ型: 数値

引数:

引数

引数	説明
prob	正規分布に関連付けられた確率。0 ~ 1 の数値を指定します。

引数	説明
mean	分布の算術平均を示す値です。
standard_dev	分布の標準偏差を示す正の値です。

この関数は、次のように **NormDist** 関数に関連します。
`If prob = NORMDIST(value, m, sd), then NORMINV(prob, m, sd) = value`

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

例と結果:

例	結果
NORMINV(0.6914625, 0, 1)	0.5000 を返します

PoissonDist

`PoissonDist()` は、ガンマ分布の累計確率を返します。

構文:

```
PoissonDist(value, mean)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
mean	平均結果を定義する正の数値。

この関数は、次のように `PoissonInv` 関数に関連します。
`If prob = PoissonDist(value, mean), then PoissonInv(prob, mean) = value`

PoissonFrequency

`PoissonFrequency()` は、ポアソン確率分布を返します。

構文:

```
PoissonFrequency(value, mean)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
mean	平均結果を定義する正の数値。

PoissonInv

PoissonInv() は、Gamma distribution分布の累計確率の逆関数を返します。

構文:

```
PoissonInv(prob, mean)
```

戻り値データ型: 数値

引数

引数	説明
prob	ポアソン確率分布に関連付けられた確率。0 ~ 1 の数値を指定します。
mean	平均結果を定義する正の数値。

この関数は、次のように PoissonDIST 関数に関連します。
If prob = PoissonDist(value, mean), then PoissonInv(prob, mean) = value

TDensity

TDensity() は、スチューデントの t 密度関数の値を返します。ここでの数値は t の計算値であり、この値に対して確率が計算されます。

構文:

```
TDensity(value, degrees_freedom)
```

戻り値データ型: 数値

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
degrees_freedom	自由度を数値で示す正の整数です。

TDist

TDist() は、スチューデント t 分布における確率を返します。ここでの数値は t の計算値であり、この値に対して確率が計算されます。

構文:
```
TDist(value, degrees_freedom, tails)
```

戻り値データ型: 数値

引数:

引数

引数	説明
value	分布を評価する値。値は、負の値でないことが条件です。
degrees_freedom	自由度を数値で示す正の整数です。
tails	1 (片側分布) または 2 (両側分布) のどちらかです。

この関数は、次のように **TInv** 関数に関連します。
```
If prob = TDIST(value, df ,2), then TINV(prob, df) = value
```

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

例と結果:

例	結果
TDIST(1, 30, 2)	0.3253 を返します

TInv

TInv() は、スチューデント t 分布の t 値を確率と自由度の関数として返します。

構文:
```
TINV(prob, degrees_freedom)
```

戻り値データ型: 数値

引数:

引数

引数	説明
prob	t 分布に関連付けられた両側確率です。0 ~ 1 の数値を指定します。
degrees_freedom	自由度を数値で示す整数です。

制限事項:

すべての引数は数値でなくてはなりません。数値でない場合は NULL が返されます。

この関数は、次のように **TDist** 関数に関連します。
`If prob = TDIST(value, df ,2), then TINV(prob, df) = value`。

例と結果:

例	結果
TINV(0.3253086, 30)	1.0000 を返します

8.24 文字列関数

このセクションでは、文字列の取り扱いと操作を行うための関数について説明します。

すべての関数は、データロードスクリプトおよびチャートの数式の両方で使用できます。例外は **Evaluate** で、これはデータロードスクリプトでしか使用できません。

文字列関数の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Capitalize

Capitalize()は、すべての単語の頭文字が大文字の文字列を返します。

```
Capitalize (text)
```

Chr

Chr() は、指定された整数に対応するUnicode文字を返します。

```
Chr (int)
```

Evaluate

Evaluate() は、指定されたテキスト文字列が有効なQlik Sense数式かどうかを評価し、有効な数式の場合は、数式の値を文字列として返します。指定された数式が有効な数式でない場合は、NULL が返されます。

```
Evaluate (expression_text)
```

FindOneOf

FindOneOf() は、文字列を検索して、指定された文字のセットのいずれかの文字が出現する位置を取得します。3番目の引数に1よりも大きい値が指定されていない限り、指定された文字のセットのいずれかの文字が最初に出現した位置が返されます。出現しない場合は、**0** が返されます。

```
FindOneOf (text, char_set[, count])
```

Hash128

Hash128() は、複合入力式の値の128ビットハッシュ値を返します。結果は、22文字の文字列になります。

```
Hash128 (expr{, expression})
```

Hash160

Hash160() は、複合入力式の値の160ビットハッシュ値を返します。結果は、27文字の文字列になります。

```
Hash160 (expr{, expression})
```

Hash256

Hash256() は、複合入力式の値の 256 ビット ハッシュ値を返します。結果は、43 文字の文字列になります。

```
Hash256 (expr{, expression})
```

Index

Index() は、文字列を検索して、指定されたサブストリングが n 回目に出現する開始位置を取得します。n の値は、オプションの 3 番目の引数で指定されます。省略されている場合は、1 になります。負の値が指定された場合は、文字列の末尾から検索を行います。文字列内での位置は、**1** から順に番号が付けられます。

```
Index  (text, substring[, count])
```

IsJson

IsJson() は、指定された文字列に有効な JSON (JavaScript Object Notation) データが含まれているかどうかをテストします。特定の JSON データ タイプを検証することもできます。

```
IsJson (json [, type])
```

JsonGet

JsonGet() は、JSON (JavaScript Object Notation) データ文字列のパスを返します。データは有効な JSON にする必要がありますが、余分なスペースや改行を含めることができます。

```
JsonGet (json, path)
```

JsonSet

JsonSet() は、JSON (JavaScript Object Notation) データを含んでいる文字列を変更します。パスにより指定される新しい場所を使用した JSON 値を設定もしくは挿入することができます。データは有効な JSON にする必要がありますが、余分なスペースや改行を含めることができます。

```
JsonSet(json, path, value)
```

KeepChar

KeepChar() は、最初の文字列である 'text' で構成される文字列から、2 番目の文字列である "keep_chars" に含まれない文字を除いて返します。

```
KeepChar (text, keep_chars)
```

Left

Left() は、入力文字列の最初の (一番左にある) 文字で構成される文字列を返します。ここで、文字数は 2 番目の引数により決定されます。

```
Left (text, count)
```

Len

Len() は、指定された文字列の長さを返します。

```
Len (text)
```

LevenshteinDist

LevenshteinDist() は、2 つの文字列間の Levenshtein の距離を返します。これは、1 つの文字列を別の文字列に変更するために必要な 1 文字の編集 (挿入、削除、または置換) の最小数として定義されます。この関数は、あいまい文字列の比較に役立ちます。

```
LevenshteinDist (text1, text2)
```

Lower

Lower() は、指定された文字列のすべての文字を小文字に変換します。

```
Lower (text)
```

LTrim

LTrim() は、指定された文字列を先頭のスペースを削除して返します。

```
LTrim (text)
```

Mid

Mid() は、2 番目の引数 'start' で定義された文字の位置で始まり、3 番目の引数 'count' で定義された文字数を返す入力文字列の一部を返します。'count' が省略されている場合、入力文字列の残りが返されます。入力文字列の最初の文字には、1 が付けられます。

```
Mid (text, start[, count])
```

Ord

Ord() は、指定された文字列の最初の文字の Unicode コードポイント番号を返します。

```
Ord (text)
```

PurgeChar

PurgeChar() は、2 番目の引数 ('remove_chars') に表示されるものを除き、入力文字列 ('text') に含まれる文字で構成される文字列を返します。

```
PurgeChar (text, remove_chars)
```

Repeat

Repeat() は、指定された文字列を、2 番目の引数で指定された回数分繰り返した文字列を返します。

```
Repeat (text[, repeat_count])
```

Replace

Replace() は、指定された文字列内に含まれる指定されたサブストリングすべてを別のサブストリングで置き換えた文字列を返します。この関数は非再帰関数で、左から右へ処理されます。

```
Replace (text, from_str, to_str)
```

Right

Right() は、指定された文字列の末尾 (右端) から、2 番目の引数で指定された文字数の文字列を返します。

```
Right (text, count)
```

RTrim

RTrim() は、指定された文字列を末尾のスペースを削除して返します。

```
RTrim (text)
```

SubField

SubField()は、元のレコード項目が区切り文字で区切られた複数の部分で構成されている文字列項目からサブストリング部分を抽出するために使用されます。

```
SubField (text, delimiter[, field_no ])
```

SubStringCount

SubStringCount()は、指定された文字列テキストに、指定されたサブストリングが出現する回数を返します。出現しない場合は、0 を返します。

```
SubStringCount (text, substring)
```

TextBetween

TextBetween() は、区切り文字として指定された文字間で行われる入力文字列でのテキストを返します。

```
TextBetween (text, delimiter1, delimiter2[, n])
```

Trim

Trim() は、指定された文字列を先頭と末尾のスペースを削除して返します。

```
Trim (text)
```

Upper

Upper() は、数式のすべてのテキスト文字について、入力文字列のすべての文字を大文字に変換します。数字と記号は無視されます。

```
Upper (text)
```

Capitalize

Capitalize()は、すべての単語の頭文字が大文字の文字列を返します。

構文:

```
Capitalize(text)
```

戻り値データ型: string

例: チャートの数式

例	結果
Capitalize ('star trek')	'Star Trek' を返します
Capitalize ('AA bb cC Dd')	'Aa Bb Cc Dd' を返します

例: ロードスクリプト

```
Load  String, Capitalize(String) Inline [String rHode iSland washingTon d.C. new york];
```

結果

文字列	**Capitalize(String)**
rHode iSland	Rhode Island
washingTon d.C.	Washington D.C.
new york	New York

Chr

Chr() は、指定された整数に対応するUnicode 文字を返します。

構文:

```
Chr(int)
```

戻り値データ型: string

例と結果:

例	**結果**
Chr(65)	文字列 'A' を返します
Chr(163)	文字列 '£' を返します
Chr(35)	文字列 '#' を返します

Evaluate

Evaluate() は、指定されたテキスト文字列が有効な Qlik Sense 数式かどうかを評価し、有効な数式の場合は、数式の値を文字列として返します。指定された数式が有効な数式でない場合は、NULL が返されます。

構文:

```
Evaluate(expression_text)
```

戻り値データ型: dual

この文字列関数は、チャートの数式で使用できません。

例と結果:

関数の例	**結果**
`Evaluate ( 5 * 8 )`	'40' を返します

ロード スクリプトの例

```
Load  Evaluate(String) as Evaluated, String Inline [String 4 5+3 0123456789012345678 Today()
];
```

結果

文字列	評価済み
4	4
5+3	8
0123456789012345678	0123456789012345678
Today()	2022-02-02

FindOneOf

FindOneOf() は、文字列を検索して、指定された文字のセットのいずれかの文字が出現する位置を取得します。3 番目の引数に 1 よりも大きい値が指定されていない限り、指定された文字のセットのいずれかの文字が最初に出現した位置が返されます。出現しない場合は、**0** が返されます。

構文:

```
FindOneOf(text, char_set[, count])
```

戻り値データ型: 整数

引数:

引数

引数	説明
text	元の文字列。
char_set	text で検索する文字セット
count	検索するいずれかの文字のうち、出現する文字を定義します。たとえば、値 2 は 2 番目の出現を検索します。

例: チャートの数式

例	結果
FindOneOf('my example text string', 'et%s')	「e」は例の文字列の 4 番目の文字であるため、「4」を返します。

例	結果
`FindOneOf( 'my example text string', 'et%s', 3)`	いずれかの文字のが検索されるため e、t、% または s、および "t" は 3 番目に出現し、例の文字列の位置 12 にあるため、「12」を返します。
`FindOneOf( 'my example text string', '¤%&')`	例の文字列に ¤、%、または & の文字が存在しないため、「0」を返します。

例: ロードスクリプト

```
Load * Inline [SearchFor, Occurrence et%s,1 et%s,3 ¤%&,1]
```

結果

SearchFor	Occurrence	FindOneOf('my example text string', SearchFor, Occurrence)
et%s	1	4
et%s	3	12
¤%&	1	0

Hash128

Hash128() は、複合入力式の値の 128 ビットハッシュ値を返します。結果は、22 文字の文字列になります。

構文:

```
Hash128(expr{, expression})
```

戻り値データ型: string

例: チャートの数式

例	結果
Hash128 ('abc', 'xyz', '123')	「MA&5]6+3=:>:>G%S<U*S2+」を返します。
Hash128 (Region, Year, Month) Note: Region, Year, and Month are table fields.	「G7*=6GKPJ(Z+)^KM?<$'A+」を返します。

例: ロードスクリプト

```
Hash_128: Load *,  Hash128(Region, Year, Month) as Hash128; Load * inline [ Region, Year, Month abc, xyz, 123 EU, 2022, 01 UK, 2022, 02 US, 2022, 02 ];
```

結果

地域	年	月	**Hash128**
abc	xyz	123	MA&5]6+3=:>;>G%S<U*S2+
EU	2022	01	B40^K&[T@!;VB'XR]<5=/$
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!
US	2022	02	C6@#]4#_G-(]J7EQY#KRW0

Hash160

Hash160() は、複合入力式の値の 160 ビット ハッシュ値を返します。結果は、27 文字の文字列になります。

構文:

```
Hash160(expr{, expression})
```

戻り値データ型: string

例: チャートの数式

例	結果
Hash160 ('abc', 'xyz', '123')	「MA&5]6+3=:>;>G%S<U*S2I:`=X*」を返します。
Hash160 (Region, Year, Month) Note: Region, Year, and Month are table fields.	「G7*=6GKPJ(Z+)^KM?<$'AI.)?U$」を返します。

例: ロード スクリプト

```
Hash_160: Load *,  Hash160(Region, Year, Month) as Hash160; Load * inline [ Region, Year, Month abc, xyz, 123 EU, 2022, 01 UK, 2022, 02 US, 2022, 02 ];
```

結果

地域	年	月	**Hash160**
abc	xyz	123	MA&5]6+3=:>;>G%S<U*S2I:`=X*
EU	2022	01	B40^K&[T@!;VB'XR]<5=//_F853
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!T"FWZ
US	2022	02	C6@#]4#_G-(]J7EQY#KRW`@KF+W

Hash256

Hash256() は、複合入力式の値の 256 ビットハッシュ値を返します。結果は、43 文字の文字列になります。

構文:

```
Hash256(expr{, expression})
```

戻り値データ型: string

例: チャートの数式

例	結果
Hash256 ('abc', 'xyz', '123')	「MA&5]6+3=:>;>G%S<U*S2I:`=X*A.IO*8N\%Y7Q;YEJ」を返します。
Hash256 (Region, Year, Month) Note: Region, Year, and Month are table fields.	「G7*=6GKPJ(Z+)^KM?<$'AI.)?U$#X2RB[:0ZP=+Z`F:」を返します。

例: ロードスクリプト

```
Hash_256: Load *,  Hash256(Region, Year, Month) as Hash256; Load * inline [ Region, Year, Month abc, xyz, 123 EU, 2022, 01 UK, 2022, 02 US, 2022, 02 ];
```

結果

地域	年	月	**Hash256**
abc	xyz	123	MA&5]6+3=:>;>G%S<U*S2I:`=X*A.IO*8N\%Y7Q;YEJ
EU	2022	01	B40^K&[T@!;VB'XR]<5=//_F853?BE6'G&,YH*T'MF)
UK	2022	02	O5T;+1?[B&"F&1//MA[MN!T"FWZT=4\#V`M%6_\0C>4
US	2022	02	C6@#]4#_G-(]J7EQY#KRW`@KF+W-0]`[Z8R+#'")=+0

Index

Index() は、文字列を検索して、指定されたサブストリングが n 回目に出現する開始位置を取得します。n の値は、オプションの 3 番目の引数で指定されます。省略されている場合は、1 になります。負の値が指定された場合は、文字列の末尾から検索を行います。文字列内での位置は、**1** から順に番号が付けられます。

構文:

```
Index(text, substring[, count])
```

戻り値データ型: 整数

引数:

引数

引数	説明
text	元の文字列。
substring	text で検索する文字の文字列。
count	検索する **substring** のうち出現するサブストリングを定義します。たとえば、値 2 は 2 番目の出現を検索します。

例と結果:

例	結果
Index('abcdefg', 'cd')	3 を返します
Index('abcdabcd', 'b', 2)	6 ('b' の 2 番目の出現) を返します
Index('abcdabcd', 'b',-2)	2 ('b' の終わりから始めて 2 番目の出現) を返します
`Left( Date, Index( Date,'-') -1 )` where **Date** = 1997-07-14	1997 を返します
`Mid( Date, Index( Date, '-', 2 ) -2, 2 )` where **Date** = 1997-07-14	07 を返します

スクリプト

```
T1: Load  *,  index(String, 'cd') as Index_CD,              // returns 3  in Index_CD index
(String, 'b') as Index_B,               // returns 2 in Index_B index(String, 'b', -1) as
Index_B2;           // returns 2 or 6 in Index_B2 Load * inline [ String abcdefg abcdabcd ];
```

IsJson

IsJson() は、指定された文字列に有効な JSON (JavaScript Object Notation) データが含まれているかどうかをテストします。特定の JSON データタイプを検証することもできます。

構文:

```
value IsJson(json [, type])
```

戻り値データ型: dual

引数

引数	説明
json	テストする文字列。余分なスペースまたは改行を含めることができます。
type	テストするJSON データ型を指定するオプションの引数。 • 'value' (既定) • 'object' • 'array' • '文字列' • 'number' • 'Boolean' • 'null'

例: 有効な JSON とタイプ

例	結果
`IsJson('null')`	-1 (true) を返します
`IsJson('"abc"', 'value')`	-1 (true) を返します
`IsJson('"abc"', 'string')`	-1 (true) を返します
`IsJson(123, 'number')`	-1 (true) を返します

例: 無効な JSON またはタイプ

例	結果	説明
`IsJson('text')`	0 (false) を返します	'text' は有効な JSON 値ではありません
`IsJson('"text"', 'number')`	0 (false) を返します	'"text"' は有効な JSON の数ではありません
`IsJson('"text"', 'text')`	0 (false) を返します	'text' は有効な JSON タイプではありません

JsonGet

JsonGet() は、JSON (JavaScript Object Notation) データ文字列のパスを返します。データは有効な JSON にする必要がありますが、余分なスペースや改行を含めることができます。

構文:

```
value JsonGet(json, path)
```

戻り値データ型： dual

引数

引数	説明
json	JSON データを含む文字列。
path	パスは RFC6901 に従って指定する必要があります。これにより、複雑なサブストリングやインデックス関数を使用せずに、JSON データ内のプロパティを検索できます。

例: 有効な JSON とパス

例	結果
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '')`	'{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}' を返します
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/a')`	'{"foo":"bar"}' を返します
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/a/foo')`	'"bar"' を返します
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b')`	'[123,"abc","ABC"]' を返します
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b/0')`	'123' を返します
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b/1')`	'"abc"' を返します
`JsonGet('{"a":{"foo":"bar"},"b":[123,"abc","ABC"]}', '/b/2')`	'"ABC"' を返します

例: 無効な JSON またはパス

例	結果	説明
`JsonGet('{"a":"b"}','/b')`	null を返します	パスが JSON データの有効な部分を指していません。
`JsonGet('{"a"}','/a')`	null を返します	JSON データは有効な JSON ではありません (メンバー「a」には値がありません)。

JsonSet

JsonSet() は、JSON (JavaScript Object Notation) データを含んでいる文字列を変更します。パスにより指定される新しい場所を使用した JSON 値を設定もしくは挿入することができます。データは有効な JSON にする必要がありますが、余分なスペースや改行を含めることができます。

構文:

```
value JsonSet(json, path, value)
```

戻り値データ型: dual

引数

引数	説明
json	JSON データを含む文字列。
path	パスは RFC6901 に従って指定する必要があります。これにより、複雑なサブストリングまたはインデックス関数や連結を使用せずに、JSON データ内にプロパティを構築できます。
value	JSON 形式の新しい文字列値。

例: 有効な JSON、パス、値

例	結果
`JsonSet('{}','/a','"b"')`	'{"a":"b"}' を返します
`JsonSet('[]','/0','"x"')`	'["x"]' を返します
`JsonSet('"abc"','','123')`	123 を返します

例: 無効な JSON、パス、または値

例	結果	説明
`JsonSet('"abc"','/x','123')`	null を返します	パスが JSON データの有効な部分を指していません。
`JsonSet('{"a":{"b":"c"}}','a/b','"x"')`	null を返します	パスが無効です。
`JsonSet('{"a":"b"}','/a','abc')`	null を返します	値は有効な JSON ではありません。文字列は引用符で囲む必要があります。

KeepChar

KeepChar() は、最初の文字列である 'text' で構成される文字列から、2 番目の文字列である "keep_chars" に含まれない文字を除いて返します。

構文:

```
KeepChar(text, keep_chars)
```

戻り値データ型: string

引数:

引数

引数	説明
text	元の文字列。
keep_chars	text に保持される文字を含む文字列。

例: チャートの数式

例	結果
`KeepChar ( 'a1b2c3','123' )`	'123' を返します。
`KeepChar ( 'a1b2c3','1234' )`	'123' を返します。
`KeepChar ( 'a1b22c3','1234' )`	'1223' を返します。
`KeepChar ( 'a1b2c3','312' )`	'123' を返します。

例: ロードスクリプト

```
T1:
Load
*,
keepchar(String1, String2) as KeepChar;
Load * inline [
String1, String2
'a1b2c3', '123'
];
```

結果

ロードスクリプトで *KeepChar* 関数を使用した結果の出力を示す Qlik Sense テーブル。

String1	String2	KeepChar
a1b2c3	123	123

参照先:

PurgeChar (page 1438)

Left

Left() は、入力文字列の最初の (一番左にある) 文字で構成される文字列を返します。ここで、文字数は 2 番目の引数により決定されます。

構文:

```
Left(text, count)
```

戻り値データ型: string

引数:

引数	説明
text	元の文字列。
count	**text** 文字列の左側から含まれている文字の数を定義します。

例: チャートの数式

例	結果
`Left('abcdef', 3)`	'abc' を返します

例: ロードスクリプト

```
T1: Load  *,  left(Text,Start) as Left;        Load * inline [ Text, Start 'abcdef', 3 '2021-07-14', 4 '2021-07-14', 2 ];
```

結果

ロードスクリプトで *Left* 関数を使用した結果の出力を示す Qlik Sense のテーブル。

テキスト	開始	Left
abcdef	3	abc
2021-07-14	4	2021
2021-07-14	2	20

より複雑な文字列の分析が可能な *Index (page 1427)* も参照してください。

Len

Len() は、指定された文字列の長さを返します。

構文:

```
Len(text)
```

戻り値データ型: 整数

例: チャートの数式

例	結果
`Len('Peter')`	'5' を返します

例: ロード スクリプト

```
T1: Load String, First&Second as NewString; Load *, mid(String,len(First)+1) as Second; Load *, upper(left(String,1)) as First; Load  * inline [ String this is a sample text string capitalize first letter only ];
```

結果

文字列	NewString
this is a sample text string	This is a sample text string
capitalize first letter only	Capitalize first letter only

LevenshteinDist

LevenshteinDist() は、2 つの文字列間の Levenshtein の距離を返します。これは、1 つの文字列を別の文字列に変更するために必要な 1 文字の編集 (挿入、削除、または置換) の最小数として定義されます。この関数は、あいまい文字列の比較に役立ちます。

構文:

```
LevenshteinDist(text1, text2)
```

戻り値データ型: 整数

例: チャートの数式

例	結果
LevenshteinDist('Kitten','Sitting')	「3」を返します

例: ロード スクリプト

ロード スクリプト

```
T1: Load *, recno() as ID; Load 'Silver' as String_1,* inline [ String_2 Sliver SSiver SSiveer ];  T1: Load *, recno()+3 as ID; Load 'Gold' as String_1,* inline [ String_2 Bold Bool Bond ];  T1: Load *, recno()+6 as ID; Load 'Ove' as String_1,* inline [ String_2 Ove Uve Üve ];  T1: Load *, recno()+9 as ID; Load 'ABC' as String_1,* inline [ String_2 DEFG abc ビビビ ];  set nullinterpret = '<NULL>'; T1: Load *, recno()+12 as ID; Load 'X' as String_1,* inline [ String_2 '' <NULL> 1 ];  R1: Load  ID,  String_1,  String_2,  LevenshteinDist(String_1, String_2) as LevenshteinDistance  resident T1;  Drop table T1;
```

結果

ID	String_1	String_2	LevenshteinDistance
1	Silver	Sliver	2

ID	String_1	String_2	LevenshteinDistance
2	Silver	SSiver	2
3	Silver	SSiveer	3
4	Gold	太字	1
5	Gold	Bool	3
6	Gold	Bond	2
7	Ove	Ove	0
8	Ove	Uve	1
9	Ove	Üve	1
10	ABC	DEFG	4
11	ABC	abc	3
12	ABC	ビビビ	3
13	X		1
14	X	-	1
15	X	1	1

Lower

Lower() は、指定された文字列のすべての文字を小文字に変換します。

構文:

```
Lower(text)
```

戻り値データ型: string

例: チャートの数式

例	結果
`Lower('abcD')`	'abcd' を返します

例: ロードスクリプト

```
Load  String, Lower(String) Inline [String rHode iSland washingTon d.C. new york];
```

結果

文字列	Lower(String)
rHode iSland	rhode island
washingTon d.C.	washington d.c.
new york	new york

LTrim

LTrim() は、指定された文字列を先頭のスペースを削除して返します。

構文:

```
LTrim(text)
```

戻り値データ型: string

例: チャートの数式

例	結果
LTrim(' abc')	'abc' を返します
LTrim('abc ')	'abc ' を返します

例: ロードスクリプト

```
Set verbatim=1;  T1:  Load *, len(LtrimString) as LtrimStringLength;       Load *, ltrim
(String) as LtrimString; Load *, len(String) as StringLength;                Load * Inline [
String '   abc    ' ' def  '];
```

> *ltrim 関数のデモンストレーションの前にスペースが自動的にトリミングされないようにするために、「Setverbatim = 1」ステートメントが例に含まれています。詳細については、Verbatim (page 210) を参照してください。*

結果

文字列	StringLength	LtrimStringLength
def	6	5
abc	10	7

参照先:

- *RTrim (page 1441)*

Mid

Mid() は、2 番目の引数 'start' で定義された文字の位置で始まり、3 番目の引数 'count' で定義された文字数を返す入力文字列の一部を返します。'count' が省略されている場合、入力文字列の残りが返されます。入力文字列の最初の文字には、1 が付けられます。

構文:

```
Mid(text, start[, count])
```

戻り値データ型: string

引数:

引数

引数	説明
text	元の文字列。
start	text に含まれる最初の文字の位置を定義する整数。
count	出力文字列の文字列長を定義します。省略されている場合は、**start** で定義された位置からのすべての文字が含まれます。

例: チャートの数式

例	結果
`Mid('abcdef',3 )`	'cdef' を返します
`Mid('abcdef',3, 2 )`	'cd' を返します

例: ロードスクリプト

```
T1: Load *,  mid(Text,Start) as Mid1,            mid(Text,Start,Count) as Mid2;     Load *
inline [ Text, Start, Count 'abcdef', 3, 2 'abcdef', 2, 3 '210714', 3, 2 '210714', 2, 3 ];
```

結果

ロードスクリプトで *Mid* 関数を使用した結果の出力を示す Qlik Sense のテーブル。

テキスト	開始	Mid1	Count	Mid2
abcdef	2	bcdef	3	bcd
abcdef	3	cdef	2	cd
210714	2	10714	3	107
210714	3	0714	2	07

参照先:

Index (page 1427)

Ord

Ord() は、指定された文字列の最初の文字の Unicode コードポイント番号を返します。

構文:

```
Ord(text)
```

戻り値データ型: integer

例と結果:

チャートの数式

例	結果
Ord('A')	整数 65 を返します。
Ord('Ab')	整数 65 を返します。

ロードスクリプト

```
//Guqin (Chinese: 古琴) – 7-stringed zithers T2: Load *,  ord(Chinese) as OrdUnicode,
        ord(Western) as OrdASCII;                       Load * inline [ Chinese, Western 古琴,
Guqin ];
```

結果:

中国語	欧文	OrdASCII	OrdUnicode
古琴	Guqin	71	21476

PurgeChar

PurgeChar() は、2 番目の引数 ('remove_chars') に表示されるものを除き、入力文字列 ('text') に含まれる文字で構成される文字列を返します。

構文:

```
PurgeChar(text, remove_chars)
```

戻り値データ型: string

引数:

引数

引数	説明
text	元の文字列。
remove_chars	text で削除される文字を含む文字列。

戻り値データ型: string

例: チャートの数式

例	結果
`PurgeChar ( 'a1b2c3','123' )`	'abc' を返します。
`PurgeChar ( 'a1b2c3','312' )`	'abc' を返します。

例: ロードスクリプト

```
T1:
Load
*,
purgechar(String1, String2) as PurgeChar;
Load * inline [
String1, String2
'a1b2c3', '123'
];
```

結果

ロードスクリプトで *PurgeChar* 関数を使用した結果の出力を示す Qlik Sense テーブル。

String1	String2	PurgeChar
a1b2c3	123	abc

参照先:

- *KeepChar (page 1431)*

Repeat

Repeat() は、指定された文字列を、2 番目の引数で指定された回数分繰り返した文字列を返します。

構文:

```
Repeat(text[, repeat_count])
```

戻り値データ型: string

引数:

引数

引数	説明
text	元の文字列。
repeat_count	**text** 文字列の文字が出力文字列で繰り返される回数を定義します。

例: チャートの数式

例	結果
`Repeat( ' * ', rating )` when **rating** = 4	'****' を返します

例: ロードスクリプト

```
T1: Load *, repeat(String,2) as Repeat; Load * inline [ String hello world! hOw aRe you? ];
```

結果

文字列	繰り返し
hello world!	hello world!hello world!
hOw aRe you?	hOw aRe you?hOw aRe you?

Replace

Replace() は、指定された文字列内に含まれる指定されたサブストリングすべてを別のサブストリングで置き換えた文字列を返します。この関数は非再帰関数で、左から右へ処理されます。

構文:

```
Replace(text, from_str, to_str)
```

戻り値データ型: string

引数:

引数

引数	説明
text	元の文字列。
from_str	入力文字列 **text** に1回以上現れる文字列。
to_str	**text**文字列に現れる**from_str** をすべて置き換える文字列。

例と結果:

例	結果
`Replace('abccde','cc','xyz')`	'abxyzde' を返します

参照先:

Right

Right() は、指定された文字列の末尾 (右端) から、2番目の引数で指定された文字数の文字列を返します。

構文:

```
Right(text, count)
```

戻り値データ型: string

引数:

引数

引数	説明
text	元の文字列。
count	文字列 **text** の右端から含まれる文字数を定義します。

例: チャートの数式

例	結果
`Right('abcdef', 3)`	'def' を返します

例: ロードスクリプト

```
T1:
Load
*,
right(Text,Start) as Right;
Load * inline [
Text, Start
'abcdef', 3
'2021-07-14', 4
'2021-07-14', 2
];
```

結果

ロードスクリプトで *Right* 関数を使用した結果の出力を示す Qlik Sense テーブル。

テキスト	開始	Right
abcdef	3	DEF
2021-07-14	4	7-14
2021-07-14	2	14

RTrim

RTrim() は、指定された文字列を末尾のスペースを削除して返します。

構文:

```
RTrim(text)
```

戻り値データ型: string

例: チャートの数式

例	結果
`RTrim( ' abc' )`	' abc' を返します
`RTrim( 'abc ' )`	'abc' を返します

例: ロードスクリプト

```
Set verbatim=1;    T1:  Load *, len(RtrimString) as RtrimStringLength;         Load *, rtrim
(String) as RtrimString; Load *, len(String) as StringLength;              Load * Inline [
String '   abc    ' ' def  '];
```

> *rtrim 関数のデモンストレーションの前にスペースが自動的にトリミングされないようにするために、「Setverbatim = 1」ステートメントが例に含まれています。詳細については、Verbatim (page 210) を参照してください。*

結果

文字列	StringLength	RtrimStringLength
def	6	4
abc	10	6

参照先:

- *LTrim (page 1436)*

SubField

SubField()は、元のレコード項目が区切り文字で区切られた複数の部分で構成されている文字列項目からサブストリング部分を抽出するために使用されます。

Subfield() 関数は、フルネームで構成されるレコードのリストからファーストネームと姓、パス名のコンポーネントパーツを抽出したり、コンマ区切りのテーブルからデータを抽出したりするために使用できます。

LOAD ステートメントで、オプションの field_no パラメータを省略して、**Subfield()** 関数を使用する場合、サブストリングごとに完全な 1 つのレコードが生成されます。**Subfield()**を使用してを複数の項目がロードされる場合、すべての組み合わせのデカルト積が作成されます。

構文:

```
SubField(text, delimiter[, field_no ])
```

戻り値データ型: string

引数:

引数

引数	説明
text	元の文字列。これは、ハードコードされたテキスト、変数、ドル記号展開、またはその他の数式となります。
delimiter	文字列をコンポーネントパーツに分割する入力 **text** 内の文字。
field_no	オプションの 3 番目の引数は、親文字列 **text** のサブストリングのどれが返されるかを指定する整数です。値 1 を使用すると最初のサブストリングが返され、値 2 を使用すると2 番目のサブストリングが返されます。値 3 以降も同様です。 • **field_no** が正の値の場合、サブストリングは左から右に抽出されます。 • **field_no** が負の値の場合、サブストリングは右から左に抽出されます。

SubField() は、Len()、Right()、Left()、Mid()、およびその他の文字列関数など、複雑な関数の組み合わせの代わりに使用することができます。

例: SubField を使用するスクリプトとチャートの数式

例 - スクリプトとチャートの数式

基本的な例

例	結果
`SubField(S, ';' ,2)`	**S** が 'abc;cde;efg' の場合、'cde' を返します。
`SubField(S, ';' ,1)`	**S** が空の文字列の場合、空の文字列を返します。
`SubField(S, ';' ,1)`	**S** が ';' の場合、空の文字列を返します。
vMyPath というパス名を持つ変数があるとすると、 `Set vMyPath=\Users\ext_jrb\Documents\Qlik\Sense\Apps;`。	テキストと画像チャートで、メジャーの `SubField(vMyPath, '\',-3)` などを追加できます。これは、変数 vMyPath の右端から3 番目のサブ文字列であるため、「Qlik」を返します。

スクリプト例 1

ロード スクリプト

データ ロード エディターで、以下のスクリプト式とデータをロードします。

```
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;
```

ビジュアライゼーションの作成

Qlik Sense シートに、[**Name**]、[**FirstName**]、[**SurName**] を軸としたテーブルのビジュアライゼーションを作成します。

結果

Name	FirstName	SurName
Dave Owen	Dave	Owen
Joe Tem	Joe	Tem

説明

[**SubField()**] 関数は、[**field_no**] 引数を 1 に設定することで、[**Name**] の最初のサブ文字列を抽出します。[**field_no**] の値は正なので、左から右の順にサブ文字列が抽出されます。2 番目の関数呼び出しでは、[**field_no**] 引数を -1 にすることで、右から左の順に 2 番目のサブ文字列を抽出します。

スクリプト例 2

ロード スクリプト

データ ロード エディターで、以下のスクリプト式とデータをロードします。

```
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 '|');
```

ビジュアライゼーションの作成

Qlik Sense シートに [**Instrument**]、[**Player**]、[**Project**] を軸としたテーブルのビジュアライゼーションを作成します。

結果

Instrument	Player	Project
Guitar	Mike	Music
Guitar	Mike	Video
Guitar	Mike	OST
Guitar	Neil	Music
Guitar	Neil	Video
Guitar	Neil	OST
Synth	Jen	Music
Synth	Jen	Video
Synth	Jen	OST
Synth	Jo	Music
Synth	Neil	Music
Synth	Neil	Video
Synth	Neil	OST

説明

この例では、**Subfield()** 関数の複数のインスタンスを、すべての組み合わせのデカルト積を作成する同じ **LOAD** ステートメント内からfield_no パラメータを除外して使用する方法を示しています。**DISTINCT** オプションを使用すると、レコードの複製が作成されるのを避けることができます。

SubStringCount

SubStringCount()は、指定された文字列テキストに、指定されたサブストリングが出現する回数を返します。出現しない場合は、0 を返します。

構文:

```
SubStringCount(text, sub_string)
```

戻り値データ型: 整数

引数:

引数	説明
text	元の文字列。
sub_string	**text**入力文字列に1回以上現れる文字列。

例: チャートの数式

例	結果
`SubStringCount ( 'abcdefgcdxyz', 'cd' )`	'2' を返します
`SubStringCount ( 'abcdefgcdxyz', 'dc' )`	'0' を返します

例: ロードスクリプト

```
T1: Load *,  substringcount(upper(Strings),'AB') as SubStringCount_AB; Load * inline [ Strings
ABC:DEF:GHI:AB:CD:EF:GH aB/cd/ef/gh/Abc/abandoned ];
```

結果

文字列	SubStringCount_AB
aB/cd/ef/gh/Abc/abandoned	3
ABC:DEF:GHI:AB:CD:EF:GH	2

TextBetween

TextBetween() は、区切り文字として指定された文字間で行われる入力文字列でのテキストを返します。

構文:

```
TextBetween(text, delimiter1, delimiter2[, n])
```

戻り値データ型: string

引数:

引数	説明
text	元の文字列。
delimiter1	**text** で検索する最初の区切り文字 (または文字列) を指定します。
delimiter2	**text** で検索する 2 番目の区切り文字 (または文字列) を指定します。
n	検索する区切り文字のペアのうち、出現する区切り文字を定義します。たとえば、2 という値は、区切り文字 1 が 2 回目に現れてから、区切り文字 2 が 2 回目に現れるまでの文字を返します。

例: チャートの数式

例	結果
`TextBetween('<abc>', '<', '>')`	'abc' を返します
`TextBetween('<abc><de>', '<', '>',2)`	'de' を返します

例	結果
TextBetween('abc', '<', '>') TextBetween('<a<b', '<', '>')	どちらの例もNULLを返します。 文字列に区切り文字が見つからない場合、NULLが返されます。
TextBetween('<>', '<', '>')	長さゼロの文字列を返します。
TextBetween('<abc>', '<', '>', 2)	n は区切り文字の出現回数よりも大きいため、NULL を返します。

例: ロードスクリプト

```
Load *, textbetween(Text,'<','>') as TextBetween, textbetween(Text,'<','>',2) as
SecondTextBetween; Load * inline [ Text <abc><de> <def><ghi><jkl> ];
```

結果

テキスト	TextBetween	SecondTextBetween
<abc><de>	abc	de
<def><ghi><jkl>	def	ghi

Trim

Trim() は、指定された文字列を先頭と末尾のスペースを削除して返します。

構文:

```
Trim(text)
```

戻り値データ型: string

例と結果:

チャートの数式

例	結果
Trim(' abc')	'abc' を返します
Trim('abc ')	'abc' を返します
Trim(' abc ')	'abc' を返します

ロードスクリプト

```
Set verbatim=1;                                          T1: Load *, len(TrimString) as TrimStringLength;
(String) as TrimString; Load *, len(String) as StringLength;              Load * inline [
String '   abc    ' ' def  '](delimiter is '\t');
```

trim 関数のデモンストレーションの前にスペースが自動的にトリミングされないようにするために、「Setverbatim = 1」ステートメントが例に含まれています。詳細については、Verbatim (page 210) を参照してください。

結果:

文字列	StringLength	TrimStringLength
def	6	3
abc	10	3

Upper

Upper() は、数式のすべてのテキスト文字について、入力文字列のすべての文字を大文字に変換します。数字と記号は無視されます。

構文:

```
Upper(text)
```

戻り値データ型: string

例: チャートの数式

例	結果
`Upper(' abcD')`	'ABCD' を返します

例: ロードスクリプト

```
Load  String,Upper(String) Inline [String rHode iSland washingTon d.C. new york];
```

結果

文字列	Upper(String)
rHode iSland	RHODE ISLAND
washingTon d.C.	WASHINGTON D.C.
new york	NEW YORK

8.25 システム関数

システム関数は、システム、デバイスおよび Qlik Sense アプリのプロパティにアクセスするための関数を提供します。

システム関数の概要

関数の中には、概要の後に詳細が示されているものもあります。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Author()

この関数は、現在のアプリの作成者プロパティを含む文字列を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

> *作成者プロパティは、Qlik Sense の現行バージョンでは設定できません。QlikView ドキュメントを移行しても、作成者プロパティは維持されます。*

ClientPlatform()

この関数は、クライアントブラウザのユーザーエージェント文字列を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.114 Safari/537.36
```

ComputerName

この関数は、オペレーティング システムが返すコンピュータ名を含む文字列を返します。 これは、データロードスクリプトおよびチャート式の両方で使用できます。

> *コンピュータの名前が 15 文字を超えている場合、文字列に含まれるのは最初の 15 文字だけです。*

```
ComputerName( )
```

DocumentName

この関数は、現在の Qlik Sense アプリのファイル名について、パスなしで拡張子を含む文字列を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
DocumentName( )
```

DocumentPath

この関数は、現在の Qlik Sense アプリへの完全なパスを含む文字列を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
DocumentPath( )
```

> *この関数は標準モードに対応していません。。*

DocumentTitle

この関数は、現在の Qlik Sense アプリのタイトルを含む文字列を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
DocumentTitle( )
```

EngineVersion

このスクリプト関数は、Qlik Sense エンジンのフル バージョンを文字列で返します。

```
EngineVersion ()
```

GetCollationLocale

このスクリプト関数は、使用されている照合ロケールのカルチャ名を返します。変数 CollationLocale がまだ設定されていない場合は、実際のユーザー マシンのロケールを返します。

```
GetCollationLocale( )
```

GetObjectField

GetObjectField() は、軸の名前を返します。**Index** は、返される軸を示す任意の整数です。

```
GetObjectField - チャート関数([index])
```

GetRegistryString

この関数は、Windows レジストリにある key の値を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
GetRegistryString(path, key)
```

この関数は標準モードに対応していません。。

GetSysAttr

この関数は、選択したアプリのテナントとスペース ドメインの属性を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
GetSysAttr (name)
```

この関数を Qlik Sense Client-Managed で使用すると、空のデータ値のみが返されます。

GroupDimensionIndex

この関数は、指定されたサイクリック軸のアクティブな項目のインデックスを返します。最初の項目のインデックス値は 1 です。サイクリック軸の名前を一重引用符で囲んで入力します。チャートの数式でのみ使用できます。

```
GroupDimensionIndex(dim_name Expression)
```

GroupDimensionLabel

この関数は、指定されたサイクリック軸内の現在のステップの項目ラベルを返します。サイクリック軸の名前を一重引用符で囲んで入力します。チャートの数式でのみ使用できます。

```
GroupDimensionLabel(dim_name Expression)
```

IsPartialReload

この関数は、現在のリロードが部分的である場合は -1 (True)、それ以外の場合は 0 (False) を返します。

```
IsPartialReload ()
```

InObject

InObject() チャート関数は、現行オブジェクトが関数引数で指定された ID を持つ別のオブジェクト内に含まれているかどうかを評価します。オブジェクトはシートまたはビジュアライゼーションのいずれかです。

```
InObject - チャート関数(id_str)
```

ObjectId

ObjectId() チャート関数は、式が評価されるオブジェクトの ID を返します。この関数は、オプションの引数を取り、どのタイプのオブジェクトを考慮するかを指定します。オブジェクトはシートまたはビジュアライゼーションのいずれかです。この関数はチャートの数式でのみ使用できます。

```
ObjectId - チャート関数([object_type_str])
```

OSUser

この関数は、現在接続しているユーザーの名前を含む文字列を返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
OSUser( )
```

> *Qlik Sense Desktop および Qlik Sense Client-Managed モバイルで、この関数は常に 'Personal\Me' を返します。*

ProductVersion

このスクリプト関数は、Qlik Sense のフルバージョンとビルド番号を文字列で返します。

この関数は廃止予定であり、**EngineVersion()** に置き換えられます。

```
ProductVersion ()
```

ReloadTime

この関数は、最後にデータロードを実行したときのタイムスタンプを返します。これは、データロードスクリプトおよびチャート式の両方で使用できます。

```
ReloadTime( )
```

StateName

StateName() は使用されているビジュアライゼーションの並列ステートの名前を返します。StateName は、例えば、ビジュアライゼーションのステートが変更されたことを反映する動的テキストやカラーを使ってビジュアライゼーションを作成するのに使用できます。この関数はチャート数式で使用できますが、数式が参照するステートを特定するためには使用できません。

```
StateName - チャート関数()
```

EngineVersion

このスクリプト関数は、Qlik Sense エンジンのフル バージョンを文字列で返します。

構文:

```
EngineVersion()
```

GetSysAttr

この関数は、選択したアプリのテナントとスペース ドメインの属性を返します。これは、データ ロード スクリプトおよびチャート式の両方で使用できます。

この関数を Qlik Sense Client-Managed で使用すると、空のデータ値が返されます。よって、後ほど Qlik Cloud に app をアップロードするために、この関数を使用してエラーを発生させることなくQlik Sense Client-Managed でロード スクリプトを開発できます。

Qlik Cloud の関数の完全なドキュメントにアクセスするには、GetSysAttr - スクリプトとチャート関数 を参照してください。

InObject - チャート関数

InObject() チャート関数は、現行オブジェクトが関数引数で指定された ID を持つ別のオブジェクト内に含まれているかどうかを評価します。オブジェクトはシートまたはビジュアライゼーションのいずれかです。

この関数は、最上位のシート オブジェクトから、他のビジュアライゼーションにネストされたビジュアライゼーションまで、シート内のオブジェクトの階層を表示するために使用できます。この関数は、**if** と**ObjectId** 関数と併用し、アプリでカスタム ナビゲーションを作成できます。

構文:

```
InObject(id_str)
```

戻り値データ型: ブール値

Qlik Sense では、真のブール値は -1 で表現され、偽の値は 0 で表現されます。

引数

引数	説明
id_str	評価されるオブジェクトの ID を表す文字列値。

このシート ID は、アプリURL から取得できます。ビジュアライゼーションのために、[**開発者**] オプションを使ってオブジェクト型のオブジェクト ID とテキスト文字列を特定します。

次の手順を実行します。

1. 分析モードで、次のテキストを URL を追加します。
   /options/developer

2. ビジュアライゼーションを右 クリックして、[開発者] をクリックします。
3. [プロパティ] で、ダイアログ ヘッダーからオブジェクトID、「qType」プロパティからオブジェクト タイプを取得します。

制限事項:

この関数は、マスター アイテムであるコンテナ内のオブジェクト (ボタンなど) で呼び出された場合、想定外の結果をもたらすことがあります。この制限事項は、多数のリストボックスのコンテナである、フィルター パネルマスター アイテムにも適用されます。その理由は、マスター アイテムがオブジェクト階層を使用する方法です。

InObject() は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
if (page 558)	**if** および **ObjectId** 関数を併用して、条件式を作成することができます。例えば、ビジュアライゼーションではこれらの関数を使った数式により、条件付きの色分けを行うことができます。
ObjectId - チャート関数 (page 1456)	**if** と同様、**ObjectId** も **InObject** と併用すると、条件式を作成できます。

例 1 – 基本的機能

チャートの数式および結果

次の基本例では、オブジェクトが別のオブジェクトの中に入っているかどうか判断する方法がわかります。この場合、シートの ID を引数として使用することで、[テキストと画像] オブジェクトがシート オブジェクトの中にあるかどうかを確認します。

次の手順を実行します。

1. シートを開いて、シートに [**テキストと画像**] チャートをドラッグします。
2. プロパティパネルで、[**メジャーを追加**] をクリックします。
3. $f(x)$ をクリックして、数式エディタを開きます。
4. 次の数式を展開先パスに貼り付けます。
   `=InObject()`
5. 数式を変更して、括弧間の文字列としてシートの ID を含めます。
   例えば、ID が 1234-5678 のシートについては、次を使用します。
   `=InObject('1234-5678')`
6. [**適用**] をクリックします。

値 -1 がチャートに表示され、数式の評価が true であることを示します。

例 2 – 条件付きで色分けされたオブジェクト

チャートの数式および結果

概要

次の例は、現在開いているシートを示すために色分けして表示するカスタム ナビゲーション ボタンの作成方法を示しています。

最初に、新しいアプリを作成して、データ ロード エディターを開きます。次のロード スクリプトを新しいタブに貼り付けます。データ自体はプレースホルダーであり、例の内容には使用されないことに注意してください。

ロード スクリプト

```
Transactions:

Load
*
Inline
[
id,date,amount
8188,'1/19/2022',37.23
8189,'1/7/2022',17.17
8190,'2/28/2022',88.27
8191,'2/5/2022',57.42
8192,'3/16/2022',53.80
8193,'4/1/2022',82.06
8194,'4/7/2022',40.39
8195,'5/16/2022',87.21
8196,'6/15/2022',95.93
8197,'7/26/2022',45.89
8198,'8/9/2022',36.23
8199,'9/22/2022',25.66
8200,'11/23/2022',82.77
8201,'12/27/2022',69.98
8202,'1/1/2023',76.11
```

```
8203,'2/8/2022',25.12
8204,'3/19/2022',46.23
8205,'6/26/2022',84.21
8206,'9/14/2022',96.24
8207,'11/29/2022',67.67
];
```

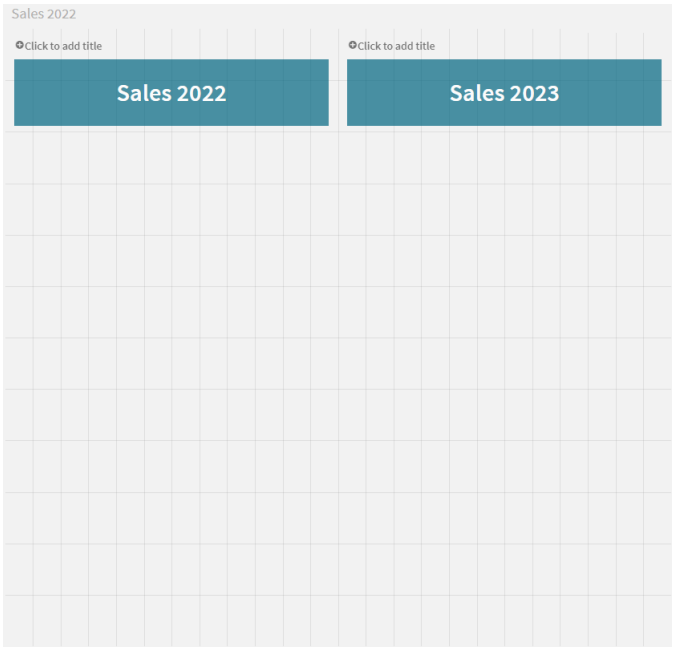
ビジュアライゼーションの作成

データをロードして新しいシートを 2 つ作成します。それぞれ *Sales 2022* と *Sales 2023* というタイトルを付けます。

次に、2 つのシート間の移動に使用される 2 つのボタン オブジェクトをビルドします。

次の手順を実行します。

1. シートに 2 つの **ボタン** オブジェクトを追加します。
2. [**スタイル**] > [**基本設定**] で、各ボタンの [**ラベル**] をそれぞれ *Sales 2022* と *Sales 2023* に設定します。
3. 次の画像のようにボタンを配置します。
   2 つのナビゲーション ボタンを使った Sales 2022 シートの配置



4. [*Sales 2022*] ボタンを選択してから、プロパティ パネルで [**アクションおよびナビゲーション**] を拡張します。
5. [**アクションを追加**] をクリックして、[**ナビゲーション**] で、[**シートの表示**] を選択します。
6. [**シート**] で、[*Sales 2022*] を選択します。
7. このボタン アクションを繰り返して、[**Sales 2023**] ボタンを [*Sales 2023*] シートにリンクさせます。
8. ボタンを右クリックして [**マスター アイテムに追加**] を選択することにより、ボタンをマスター アイテムに変換します。

これで、各ボタンをコピーし、同じサイズと配置で *Sales 2023* シートに貼り付けることができます。

条件付きの色の作成

次に、現在開いているシートにリンクされている場合は青、開いてないシートにリンクされたている場合はグレーとなるようにボタンを設定します。

次の手順を実行します。

1. [*Sales 2022*] シートを開き、URL からシート ID を取得します。[*Sales 2022*] シートを開いたままにします。
2. [**Sales 2022**] ボタンのマスター アイテムをクリックして、プロパティ パネルで [**編集**] を選択します。
3. [**スタイル**] > [**背景**] で、ボタンを [**数式別**] に色分けするように選択します。
4. [**数式**] に、次のテキストを貼り付けます。
   =if(InObject(''), Blue(), LightGray())
5. 上記数式の括弧の間に、[*Sales 2022*] シートのシート ID を貼り付けます。

これで、[*Sales 2022*] シートが開いていればボタンが青に、開いていなければグレーになるよう設定されました。

[*Sales 2023*] シートにも上記の手順を繰り返し、[**Sales 2023**] ボタンのマスター アイテムを [*Sales 2023*] シート ID にリンクします。

各シートには現在 2 つのボタンが表示され、青い色で現在開いているシートを示しています。

[Sales 2022] シートのボタンが青く [Sales 2022] が現在表示されていることを示した状態

IsPartialReload

この関数は、現在のリロードが部分的である場合は -1 (True)、それ以外の場合は 0 (False) を返します。

構文:

```
IsPartialReload()
```

ObjectId - チャート関数

ObjectId() チャート関数は、式が評価されるオブジェクトの ID を返します。この関数は、オプションの引数を取り、どのタイプのオブジェクトを考慮するかを指定します。オブジェクトはシートまたはビジュアライゼーションのいずれかです。この関数はチャートの数式でのみ使用できます。

構文:

```
ObjectId([object_type_str])
```

戻り値データ型: 文字列

関数の唯一の引数である**object_type_str** は任意であり、オブジェクトのタイプを示す文字列値を参照します。

引数

引数	説明
object_type_str	評価されるオブジェクトのタイプを表す文字列値。

関数式で指定された引数がない場合、**ObjectId()** が数式が使用されるオブジェクトの ID を返します。ビジュアライゼーションが表示されるシートオブジェクトの ID を返すには、*ObjectId('sheet')* を使用します。

ビジュアライゼーションが他のビジュアライゼーション オブジェクトの中にネストされている場合、関数引数で希望するオブジェクト タイプを指定すると、異なる結果を得ることができます。例えば、コンテナ内の [**テキストと画像**] チャートの場合、*'text-image'* を使用して [**テキストと画像**] オブジェクトを、*'container'* を使用してコンテナの ID を返します。

次の手順を実行します。

1. 分析モードで、次のテキストを URL を追加します。
   /options/developer
2. ビジュアライゼーションを右クリックして、[**開発者**] をクリックします。
3. [**プロパティ**] で、ダイアログ ヘッダーからオブジェクト ID、**「qType」**プロパティからオブジェクト タイプを取得します。

制限事項:

この関数は、マスター アイテムであるコンテナ内のオブジェクト (ボタンなど) で呼び出された場合、想定外の結果をもたらすことがあります。この制限事項は、多数のリストボックスのコンテナである、フィルター パネル マスター アイテムにも適用されます。その理由は、マスター アイテムがオブジェクト階層を使用する方法です。

その場合、チャートの数式 *ObjectId('sheet')* は、空の文字列を返しますが、*ObjectId('masterobject')* は所有するマスター アイテムの ID を表示します。

ObjectId() は、多くの場合、次の関数と組み合わせて使用されます。

関連する関数

関数	相互作用
if (page 558)	**if** および **ObjectId** 関数を併用して、条件式を作成することができます。例えば、ビジュアライゼーションではこれらの関数を使った数式により、条件付きの色分けを行うことができます。
InObject - チャート関数 (page 1452)	**if** と同様、**InObject** も **ObjectId** と併用すると、条件式を作成できます。

例 1 – チャート オブジェクト ID を返す

チャートの数式および結果
次の基本例は、ビジュアライゼーションの ID を返す方法を示しています。

次の手順を実行します。

1. シートを開いて、シートに [**テキストおよび画像**] チャートをドラッグします。
2. プロパティパネルで、[**メジャーを追加**] をクリックします。
3. $f(x)$ をクリックして、数式エディタを開きます。
4. 次の数式を展開先パスに貼り付けます。
   ```
   =ObjectId()
   ```
5. [**適用**] をクリックします。

[**テキストおよび画像**] オブジェクトの ID がビジュアライゼーションに表示されます。

次の数式でも同じ結果を得られます。

```
=ObjectId('text-image')
```

例 2 – シート ID を返す

チャートの数式および結果
次の基本例は、ビジュアライゼーションが表示されるシートの ID を返す方法を示しています。

次の手順を実行します。

1. シートを開いて、シートに [**テキストおよび画像**] チャートをドラッグします。
2. プロパティパネルで、[**メジャーを追加**] をクリックします。
3. f_x をクリックして、数式エディタを開きます。
4. 次の数式を展開先パスに貼り付けます。
   `=ObjectId('sheet')`
5. [**適用**] をクリックします。

シートの ID がビジュアライゼーションに表示されます。

例 3 – ネストされた数式

チャートの数式および結果
次の例では、**ObjectId()** 関数が他の数式内にどのようにネストされるかを示しています。

次の手順を実行します。

1. シートを開いて、シートに [**テキストおよび画像**] チャートをドラッグします。
2. プロパティパネルで、[**メジャーを追加**] をクリックします。
3. f_x をクリックして、数式エディタを開きます。
4. 次の数式を展開先パスに貼り付けます。
   =if(InObject(ObjectId('text-image')), 'In Text & image', 'Not in Text & image')
5. [**適用**] をクリックします。

チャートに「*テキストおよび画像内*」というテキストが表示され、数式で参照されるオブジェクトが [**テキストおよび画像**] チャートであることが示されます。

条件付き色分けの詳しい例については、*InObject - チャート関数 (page 1452)* の例を参照してください。

ProductVersion

このスクリプト関数は、Qlik Sense のフルバージョンとビルド番号を文字列で返します。この関数は廃止予定であり、**EngineVersion()** に置き換えられます。

構文:

```
ProductVersion()
```

StateName - チャート関数

StateName() は使用されているビジュアライゼーションの並列ステートの名前を返します。
StateName は、例えば、ビジュアライゼーションのステートが変更されたことを反映する動的テキストやカラーを使ってビジュアライゼーションを作成するのに使用できます。この関数はチャート数式で使用できますが、数式が参照するステートを特定するためには使用できません。

構文:

```
StateName ()
```

Example 1:

```
      動的テキスト
='Region - ' & if(StateName() = '$', 'Default', StateName())
```

Example 2:

```
      動的カラー
if(StateName() = 'Group 1', rgb(152, 171, 206),
      if(StateName() = 'Group 2', rgb(187, 200, 179),
            rgb(210, 210, 210)
      )
)
```

8.26 テーブル関数

テーブル関数は、現在読み込まれているデータテーブルに関する情報を返します。テーブル名の指定がなく、関数が **LOAD** ステートメント内で使用されている場合、現在のテーブルと判断されます。

すべての関数は、データロードスクリプトで使用できますが、**NoOfRows** は唯一チャート式でも使用できます。

テーブル関数の概要

関数の中には、概要の後に詳細が示されているものもあります。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

FieldName

FieldName スクリプト関数は、ロード済みのテーブルに含まれる指定の番号の項目名を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

```
FieldName (field_number ,table_name)
```

FieldNumber

FieldNumber スクリプト関数は、ロード済みのテーブルに含まれる指定の項目の番号を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

```
FieldNumber (field_name ,table_name)
```

NoOfFields

NoOfFields スクリプト関数は、ロード済みのテーブルに含まれる項目数を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

```
NoOfFields (table_name)
```

NoOfRows

NoOfRows 関数は、ロード済みのテーブルに含まれる行数 (レコード数) を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

```
NoOfRows (table_name)
```

NoOfTables

このスクリプト関数は、これまでにロードされたテーブル数を返します。

```
NoOfTables()
```

TableName

このスクリプト関数は、指定した番号のテーブル名を返します。

```
TableName(table_number)
```

TableNumber

このスクリプト関数は、指定したテーブルの番号を返します。最初のテーブルの番号は 0 です。

table_name が存在しない場合は、NULL を返します。

```
TableNumber(table_name)
```

この例では、ロードされたテーブルと項目についての情報を持つテーブルを作成します。

まず、サンプル データをロードします。このセクションに記載されているテーブル関数の説明に使用される 2 つのテーブルが作成されます。

```
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 ;
```

次に、ロードしたテーブルを **NoOfTables** 関数を使って繰り返し処理してから、**NoOfFields** 関数を使って各テーブルの項目を繰り返し処理し、テーブル関数を使って情報をロードします。

```
//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;
```

この結果、Tables は次のようになります。

Load table

Table	TableNo	TableRows	Field	FieldNo
Characters	0	26	Alpha	1
Characters	0	26	Num	2
ASCII	1	191	Num	1
ASCII	1	191	AsciiAlpha	2
ASCII	1	191	AsciiNum	3

FieldName

FieldName スクリプト関数は、ロード済みのテーブルに含まれる指定の番号の項目名を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

構文:

```
FieldName(field_number ,table_name)
```

引数:

引数

引数	説明
field_number	参照する項目の項目番号。
table_name	参照する項目が含まれているテーブル。

```
LET a = FieldName(4,'tab1');
```

FieldNumber

FieldNumber スクリプト関数は、ロード済みのテーブルに含まれる指定の項目の番号を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

構文:

```
FieldNumber(field_name ,table_name)
```

引数:

引数

引数	説明
field_name	項目の名前。
table_name	項目が含まれているテーブルの名前。

field_name 項目が、table_name に存在しない場合、もしくは table_name が存在しない場合は、この関数は 0 を返します。

```
LET a = FieldNumber('Customer','tab1');
```

NoOfFields

NoOfFields スクリプト関数は、ロード済みのテーブルに含まれる項目数を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

構文:

```
NoOfFields(table_name)
```

引数:

引数

引数	説明
table_name	テーブルの名前。

```
LET a = NoOfFields('tab1');
```

NoOfRows

NoOfRows 関数は、ロード済みのテーブルに含まれる行数 (レコード数) を返します。**LOAD** ステートメントで使用されている場合、ロード中のテーブルを参照することはできません。

構文:

```
NoOfRows(table_name)
```

引数:

引数

引数	説明
table_name	テーブルの名前。

```
LET a = NoOfRows('tab1');
```

8.27 三角関数と双曲線関数

このセクションでは、三角関数と双曲線関数の演算子について説明します。これらのすべての関数では、引数は計算結果がラジアンで表された角度になる数式で、**x** は実数として解釈されます。

角度はすべてラジアンで表します。

すべての関数は、データロードスクリプトおよびチャート式の両方で使用できます。

cos
x の余弦です。結果は、-1 から1 の間の数値になります。

```
cos( x )
```

acos
x の逆コサイン。関数は、-1≤**x**≤1 の場合にのみ定義されます。結果は、0 からπの間の数値になります。

```
acos( x )
```

sin
x の正弦です。結果は、-1 から1 の間の数値になります。

```
sin( x )
```

asin
x の逆サイン。関数は、-1≤**x**≤1 の場合にのみ定義されます。結果は、-π/2 からπ/2 の間の数値になります。

```
asin( x )
```

tan
x の正接です。結果は実数です。

```
tan( x )
```

atan
x の逆タンジェント。結果は、-π/2 からπ/2 の間の数値になります。

```
atan( x )
```

atan2
逆正接関数の2 次元一般化です。原点と、**x** 座標と**y** 座標で表される点との間の角度を返します。結果は、-π から+π の間の数値になります。

```
atan2( y,x )
```

cosh

x の双曲線余弦。結果は正の実数です。

```
cosh( x )
```

sinh

x の双曲線正弦。結果は実数です。

```
sinh( x )
```

tanh

x の双曲線正接。結果は実数です。

```
tanh( x )
```

acosh

x の逆双曲線余弦。結果は正の実数です。

```
acosh( x )
```

asinh

x の逆双曲線正弦。結果は実数です。

```
asinh( x )
```

atanh

x の逆双曲線正接。結果は実数です。

```
atanh( x )
```

以下のスクリプト コードはサンプル テーブルをロードし、その後、値を三角関数と双曲線関数で計算した結果を含むテーブルをロードします。

```
SampleData:
LOAD * Inline
[Value
-1
0
1];

Results:
Load *,
cos(Value),
acos(Value),
sin(Value),
asin(Value),
tan(Value),
atan(Value),
atan2(Value, Value),
cosh(Value),
sinh(Value),
tanh(Value)
```

```
RESIDENT SampleData;

Drop Table SampleData;
```

8.28 ウィンドウ関数

ウィンドウ関数は、複数の行の値を使用して計算を実行し、各行の値を個別に生成します。ウィンドウ関数は、テーブル全体が読み取られた場合にのみ計算できます。

ウィンドウ関数を使用して、次の操作を実行できます。

- 行内の個々の数値と、列内の平均値、最大値、または最小値を比較します。
- 列内またはテーブル全体内の個々の値のランクを計算します。

ウィンドウ関数はテーブル内のレコード数を変更しませんが、集計関数または関係関数、および範囲関数と同様のタスクを実行できます。

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Window
Window 関数は複数の行から計算を実行し、各行の値を個別に生成します。

```
Window - スクリプト関数(input_expr, [partition1, partition2, ...], [sort_type, [sort_expr]],[filter_expr], [start_expr,end_expr])[row_window_size])
```

WRank
WRank 関数は、**Window** 内でランキング計算を実行します。

```
WRank - スクリプト関数([TOTAL] expr[, mode[, fmt]])
```

Window - スクリプト関数

Window() は複数の行から計算を実行し、各行の値を個別に生成します。

Window 関数を使用して、次の操作を実行できます。

- 行内の個々の数値と、列内の平均値、最大値、または最小値を比較します。
- 列内またはテーブル全体内の個々の値のランクを計算します。

Window 関数はテーブル内のレコード数を変更しませんが、集計関数、関係関数、範囲関数と同様のタスクを実行できます。

Window 関数では、テーブルに追加するために作業しているテーブルの LOAD ステートメント内にキャッシュが必要です。例:

```
[Transactions]:
Load
	*,
	Window(avg(Expression1),[Num]);
LOAD
	TransLineID,
```

```
        TransID,
        "Num",
        Dim1,
        Dim2,
        Dim3,
        Expression1,
        Expression2,
        Expression3
FROM [lib://AttachedFiles/transactions.qvd] (qvd);
```

Window は、丸めや基本的な数値演算などの一般的な関数をサポートします。例:

```
Load *, Round(Window(Sum(Salary),Department)) as SumSalary
Load *, Window(Sum(Salary),Department) + 5 as SumSalary
```

Window 関数のスライディング ウィンドウを定義できます。これにより、現在の行に **Window** 関数を適用するときに使用される行の数が設定されます。例えば、Window を前の 3 行と後続の 3 行に設定できます。

構文:

```
Window (input_expr, [partition1, partition2, ...], [sort_type, [sort_expr]],
[filter_expr], [start_expr,end_expr])
```

戻り値データ型:LOAD ステートメントによって作成された結果テーブルに新しい項目が追加されました。

引数:

引数

引数	説明

input_expr	関数によって計算され、返された入力式です。Median(Salary) など、集計に基づく任意の式である必要があります。例:
	`Window(Median(Salary)) as MedianSalary`
	入力は、集計が適用されない項目名とすることもできます。この場合、**Window** はその項目に **Only()** 関数が適用されるかのように扱います。例:
	`Window(Salary,Department) as WSalary`
	
	必要に応じて、入力式を使用してパーティションを定義できます。パーティションは、**group by** 句によるグループ化と同じですが、結果が新しい列として入力テーブルに追加される点が異なります。パーティションによって入力テーブルのレコード数が減ることはありません。複数のパーティション項目を定義できます。
	例:
	`LOAD`
	`Window(Max(Sales), City, 'ASC', OrderDate, Sales > 300)`
	`+ AddMonths(OrderDate,-6) as MAX_Sales_City_Last_6_Mos,`
	`Window(Avg(Sales), City, 'ASC', OrderDate, City = 'Portland')`
	`+ AddMonths(OrderDate,-6) as Avg_Sales_Portland_Last_6_Mos,`
	`Window(Max(Sales), City, 'ASC', OrderDate, Sales > 300)`
	`+ AddMonths(OrderDate,-12) as MAX_Sales_City_Last_12_Mos;`
	`LOAD`
	`City,`
	`Sales,`
	`OrderDate`
	`FROM [lib://AttachedFiles/Sales Data.xlsx]`
	`(ooxml, embedded labels, table is [Sales Data]);`

partition1, partition2	input_expr の後は、任意の数のパーティションを定義できます。パーティションは、集計を適用する組み合わせを定義する項目です。集計は各パーティションに個別に適用されます。例: `Window(Avg(Salary), Unit, Department, Country) as AvgSalary` 上記では、パーティションは *Unit*、*Department*、*Country* です。 パーティションは必須ではありませんが、項目を適切にウィンドウ処理するために必要です。

引数	説明
sort_type, [sort_expr]]	必要に応じて、ソートタイプとソート数式を指定します。sort_type には、次の 2 つの値のいずれかを指定できます。 • ASC: 昇順ソート。 • DESC: 降順ソート。 sort_type を定義する場合は、ソート式を定義する必要があります。これは、パーティション内の行の順序を決定する数式です。 例: `Window(RecNo(), Department, 'ASC', Year)` 上記の例では、パーティション内の結果が *Year* 項目で昇順にソートされます。 *ソートタイプとソート式は、主に **RecNo** 関数と **WRank** 関数でのみ必要となります。*
filter_expr	必要に応じて、フィルター式を追加します。これは、レコードを計算に含めるかどうかを決定するブール式です。 このパラメータは完全に省略できます。その結果、フィルターは存在しないはずです。 例: `Window(avg(Salary), Department, 'ASC', Age, EmployeeID=3 Or EmployeeID=7) as wAvgSalary) as wAvgSalaryIfEmpIs3or7`

[start_expr,end_expr]	必要に応じて、スライディング ウィンドウ機能の引数を設定します。スライディング ウィンドウには 2 つの引数が必要です。
	• 開始の数式: ウィンドウに含める現在の行より前の行数。
	• 終了の数式: ウィンドウに含める現在の行より後の行数。
	例えば、前の 3 行、現在の行、次の行を含める場合は、次のようになります。
	`Window(concat(Text(Salary),'-'), Department, 'ASC', Age, Year>0, -3, 1) as WSalaryDepartment`
	先行するすべての行、または後続のすべての行を表示するには、**Unbounded()** 関数を使用できます。例えば、前の行、現在の行、次の行をすべて含める場合は、次のようになります。
	`Window(concat(Text(Salary),'-'), Department, 'ASC', Age, Year>0, UNBOUNDED(), 1) as WSlidingSalaryDepartment`
	例えば、現在の行から3 行目以降のすべての行を含める場合は、次のようになります。
	`Window(concat(Text(Salary),'-'), Department, 'ASC', Age, Year>0, 3, UNBOUNDED()) as WSlidingSalaryDepartment`

例 - 集計を含む項目の追加

例: 集計を含む項目の追加

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load
*,
Window(Avg(transaction_amount),customer_id) as AvgCustTransaction;

Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, M, Orange
3752, 20180916, 5.75, 1, 5646471, S, Blue
3753, 20180922, 125.00, 7, 3036491, L, Black
3754, 20180922, 484.21, 13, 049681, XS, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

結果

集計を含む項目を追加した結果

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	AvgCustTransaction
3750	20180830	23.56	2	2038593	L	赤色	103.43
3751	20180907	556.31	6	203521	M	オレンジ	266.775
3752	20180916	5.75	1	5646471	S	青	42.935
3753	20180922	125.00	7	3036491	L	ブラック	64.21

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	AvgCustTransaction
3754	20180922	484.21	13	049681	XS	赤色	273.88
3756	20180922	59.18	2	2038593	M	青	103.43
3757	20180923	177.42	21	203521	XL	ブラック	266.775
3758	20180924	153.42	14	2038593	L	赤色	103.43
3759	20180925	7.42	5	203521	M	オレンジ	266.775
3760	20180925	80.12	18	5646471	M	青	42.935
3761	20180926	3.42	7	3036491	XS	ブラック	64.21
3763	20180926	63.55	12	049681	S	赤色	273.88
3763	20180927	177.56	10	2038593	L	青	103.43
3764	20180927	325.95	8	203521	XL	ブラック	266.775

例 - 特定の値でフィルタリングされた集計を含む項目の追加

例: 特定の値でフィルタリングされた集計を含む項目の追加

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load
*,
Window(Avg(transaction_amount),customer_id, color_code = 'Blue') as AvgCustTransaction;

Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, M, Orange
3752, 20180916, 5.75, 1, 5646471, S, Blue
3753, 20180922, 125.00, 7, 3036491, L, Black
3754, 20180922, 484.21, 13, 049681, XS, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
```

```
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

結果

特定の値でフィルタリングされた集計を含む広告項目を追加した結果

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	AvgCustTransaction
3750	20180830	23.56	2	2038593	L	赤色	-
3751	20180907	556.31	6	203521	M	オレンジ	-
3752	20180916	5.75	1	5646471	S	青	42.94
3753	20180922	125.00	7	3036491	L	ブラック	-
3754	20180922	484.21	13	049681	XS	赤色	-
3756	20180922	59.18	2	2038593	M	青	118.4
3757	20180923	177.42	21	203521	XL	ブラック	-
3758	20180924	153.42	14	2038593	L	赤色	-
3759	20180925	7.42	5	203521	M	オレンジ	-
3760	20180925	80.12	18	5646471	M	青	42.94
3761	20180926	3.42	7	3036491	XS	ブラック	-
3763	20180926	63.55	12	049681	S	赤色	-
3763	20180927	177.56	10	2038593	L	青	118.4
3764	20180927	325.95	8	203521	XL	ブラック	-

例 - スライディング ウィンドウを使用した項目の追加

例: スライディング ウィンドウを使用した項目の追加

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load
*,
Window(Avg(transaction_amount),customer_id, 'ASC', -1, 1, 0, 1) as AvgCustTransaction;

Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, M, Orange
3752, 20180916, 5.75, 1, 5646471, S, Blue
3753, 20180922, 125.00, 7, 3036491, L, Black
3754, 20180922, 484.21, 13, 049681, XS, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

結果

特定の値でフィルタリングされた集計を含む広告項目を追加した結果

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	AvgCustTransaction
3750	20180830	23.56	2	2038593	L	赤色	41.37
3751	20180907	556.31	6	203521	M	オレンジ	366.865
3752	20180916	5.75	1	5646471	S	青	42.935
3753	20180922	125.00	7	3036491	L	ブラック	64.21

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	AvgCustTransaction
3754	20180922	484.21	13	049681	XS	赤色	273.88
3756	20180922	59.18	2	2038593	M	青	106.3
3757	20180923	177.42	21	203521	XL	ブラック	92.42
3758	20180924	153.42	14	2038593	L	赤色	165.49
3759	20180925	7.42	5	203521	M	オレンジ	166.685
3760	20180925	80.12	18	5646471	M	青	80.12
3761	20180926	3.42	7	3036491	XS	ブラック	3.42
3763	20180926	63.55	12	049681	S	赤色	177.56
3763	20180927	177.56	10	2038593	L	青	63.55
3764	20180927	325.95	8	203521	XL	ブラック	325.95

制限事項

Window には次の制限があります。

- **Window** はリソースを大量に消費する機能であり、特にメモリの消費量が大きくなります。
- **Window** は Qlik Sense Mobile には対応していません。
- チャートの数式は **Window** をサポートしていません。
- **Window** 関数を他の **Window** 関数内にネストすることはできません。
- **Window** は集計関数内では使用できません。
- **Window** はテーブル全体をスキャンできる必要があります。
- スライディング ウィンドウ機能を使用する場合、**WRank()**、**RecNo()**、**RowNo()** は **Window** と一緒に使用できません。

WRank - スクリプト関数

WRank() は、ロードスクリプトでテーブルの行を評価し、それぞれの行に対して、ロードスクリプトで評価された項目の値の相対位置を示します。この関数はテーブルの評価時に、結果を現在のパーティションに含まれるその他の行の結果と比較して、セグメント内の現在の行のランキングを返します。

テーブルのパーティション

	Region	Country	Population	Rank(Population)
Column segment #1	Americas	Mexico	128,932,753	2
	Americas	Canada	37,742,154	3
	Americas	United States of America	331,002,651	1
Column segment #2	Europe	Sweden	10,099,265	4
	Europe	United Kingdom	67,886,011	2
	Europe	France	65,273,511	3
	Europe	Germany	83,783,942	1

WRank は **Window** 関数でのみ使用できます。**Window** 関数には、ソートタイプとソート数式が含まれている必要があります。ソート数式にはランキングが適用されます。

構文:

```
WRank ([mode[, fmt]])
```

戻り値データ型: dual

引数:

引数

引数	説明
mode	必要に応じて、関数の結果の数値表現を指定します。
fmt	必要に応じて、関数の結果のテキスト表現を指定します。
TOTAL	テーブルが1軸の場合、またはスクリプトの前に **TOTAL** 修飾子が付加されている場合は、関数は列全体に沿って評価されます。テーブルまたはテーブルに相当するアイテムに複数の縦軸が含まれる場合、現在のパーティションには、項目ソート順の最後の軸を表示する列を除く、すべての軸列の現在行と同じ値を持つ行のみが含まれます。

ランキングは、dual 値として返されます。行ごとに固有のランキングがある場合、1から現在のパーティション内の行数を示す整数になります。

複数の行がランキングを共有する場合は、テキストおよび数値表現を **mode** および **fmt** のパラメータで制御できます。

mode

最初の引数 **mode** は、次のいずれかの値になります。

mode 値

値	説明
0 (デフォルト)	共有グループ内のすべての順位がランキング全体の中間値以下に入る場合、いずれの行も共有グループ内の最低順位を取得します。 共有グループ内のすべての順位が順位付け全体の中間値以上に入る場合は、いずれの行も共有グループ内の最高順位を取得します。 共有グループ内の順位がランキング全体の中間値をまたく場合は、いずれの行もパーティション全体の最高順位と最低順位の平均に相当する値を取得します。

値	説明
1	すべての行における最低順位。
2	すべての行における平均順位。
3	すべての行における最高順位。
4	最初の行における最低順位、その後は行ごとに1ずつ増加。

fmt

2番目の引数 **fmt** は、次の値を取ることができます。

fmt 値

値	説明
0 (デフォルト)	すべての行における低い値 **-** 高い値 (例: 3 - 4)。
1	すべての行における低い値。
2	最初の行における低い値、その後のグループ内の行は空白。

mode 4 と**fmt** 2 の行の順序は、テーブル項目のロード順で決定されます。

例 - ランク付けされた項目の追加

例: ランク付けされた項目の追加

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load
*,
Window(WRank(0),customer_id, 'Desc', transaction_amount) as TransactionRanking;

Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, M, Orange
3752, 20180916, 5.75, 1, 5646471, S, Blue
3753, 20180922, 125.00, 7, 3036491, L, Black
3754, 20180922, 484.21, 13, 049681, XS, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
```

```
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

結果

ランク付けされた項目を追加した結果

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	TransactionRanking
3750	20180830	23.56	2	2038593	L	赤色	4-4
3751	20180907	556.31	6	203521	M	オレンジ	1-1
3752	20180916	5.75	1	5646471	S	青	2-2
3754	20180922	484.21	13	049681	XS	赤色	1-1
3756	20180922	59.18	2	2038593	M	青	3-3
3753	20180922	125.00	7	3036491	L	ブラック	1-1
3757	20180923	177.42	21	203521	XL	ブラック	3-3
3758	20180924	153.42	14	2038593	L	赤色	2-2
3759	20180925	7.42	5	203521	M	オレンジ	4-4
3760	20180925	80.12	18	5646471	M	青	1-1
3763	20180926	63.55	12	049681	S	赤色	2-2
3761	20180926	3.42	7	3036491	XS	ブラック	2-2
3764	20180927	325.95	8	203521	XL	ブラック	2-2
3763	20180927	177.56	10	2038593	L	青	1-1

例 - 1 桁の結果に対して fmt を使用してランク付けされた項目を追加する

例: 1 桁の結果に対して fmt を使用してランク付けされた項目を追加する

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load
*,Window(WRank(0,1),customer_id, 'Desc', transaction_amount) as TransactionRanking;

Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, M, Orange
3752, 20180916, 5.75, 1, 5646471, S, Blue
3753, 20180922, 125.00, 7, 3036491, L, Black
3754, 20180922, 484.21, 13, 049681, XS, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

結果

1 桁の結果に対して fmt を使用してランク付けされた項目を追加した結果

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	TransactionRanking
3750	20180830	23.56	2	2038593	L	赤色	4
3751	20180907	556.31	6	203521	M	オレンジ	1
3752	20180916	5.75	1	5646471	S	青	2
3754	20180922	484.21	13	049681	XS	赤色	1
3756	20180922	59.18	2	2038593	M	青	3
3753	20180922	125.00	7	3036491	L	ブラック	1
3757	20180923	177.42	21	203521	XL	ブラック	3
3758	20180924	153.42	14	2038593	L	赤色	2
3759	20180925	7.42	5	203521	M	オレンジ	4
3760	20180925	80.12	18	5646471	M	青	1

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	TransactionRanking
3763	20180926	63.55	12	049681	S	赤色	2
3761	20180926	3.42	7	3036491	XS	ブラック	2
3764	20180927	325.95	8	203521	XL	ブラック	2
3763	20180927	177.56	10	2038593	L	青	1

例 - 複数のパーティションを持つランク付けされた項目を追加する

例: 複数のパーティションを持つランク付けされた項目を追加する

ロード スクリプト

データ ロード エディタで新しいタブを作成し、次のデータをインライン ロードとしてロードします。結果を確認するには、以下の Qlik Sense のテーブルを作成します。

```
Transactions:
Load
*,Window(WRank(0,1),customer_id, size, color_code, 'Desc', transaction_amount) as
TransactionRanking;

Load * Inline [
transaction_id, transaction_date, transaction_amount, transaction_quantity, customer_id, size,
color_code
3750, 20180830, 23.56, 2, 2038593, L, Red
3751, 20180907, 556.31, 6, 203521, M, Orange
3752, 20180916, 5.75, 1, 5646471, S, Blue
3753, 20180922, 125.00, 7, 3036491, L, Black
3754, 20180922, 484.21, 13, 049681, XS, Red
3756, 20180922, 59.18, 2, 2038593, M, Blue
3757, 20180923, 177.42, 21, 203521, XL, Black
3758, 20180924, 153.42, 14, 2038593, L, Red
3759, 20180925, 7.42, 5, 203521, M, Orange
3760, 20180925, 80.12, 18, 5646471, M, Blue
3761, 20180926, 3.42, 7, 3036491, XS, Black
3763, 20180926, 63.55, 12, 049681, S, Red
3763, 20180927, 177.56, 10, 2038593, L, Blue
3764, 20180927, 325.95, 8, 203521, XL, Black
];
```

結果

1 桁の結果に対して fmt を使用してランク付けされた項目を追加した結果

transaction_id	transaction_date	transaction_amount	transaction_quantity	customer_id	サイズ	color_code	TransactionRanking
3750	20180830	23.56	2	2038593	L	赤色	2
3751	20180907	556.31	6	203521	M	オレンジ	1
3752	20180916	5.75	1	5646471	S	青	1
3754	20180922	484.21	13	049681	XS	赤色	1
3756	20180922	59.18	2	2038593	M	青	1
3753	20180922	125.00	7	3036491	L	ブラック	1
3757	20180923	177.42	21	203521	XL	ブラック	2
3758	20180924	153.42	14	2038593	L	赤色	1
3759	20180925	7.42	5	203521	M	オレンジ	2
3760	20180925	80.12	18	5646471	M	青	1
3763	20180926	63.55	12	049681	S	赤色	1
3761	20180926	3.42	7	3036491	XS	ブラック	1
3764	20180927	325.95	8	203521	XL	ブラック	1
3763	20180927	177.56	10	2038593	L	青	1

制限事項

WRank には次の制限があります。

- fmt 値が 0 で、二重結果のテキスト部分を **WRank** に使用する場合は、**Text()** と **Window(WRank)** を使用する必要があります。例: `Text(Window(WRank(0), Unit, 'DESC', Age)) as UnitWRankedByAgeText`。

9 ファイル システム アクセス制御

セキュリティ上の理由から、Qlik Sense の標準モードでは、データ ロード スクリプトまたは関数、変数に含まれるパスはサポートされていません。これは、ファイル システムが露呈されてしまうからです。

QlikView ではこうしたファイル システム パスがサポートされていますが、QlikView ロード スクリプトを再利用するために、標準モードを無効にして、レガシー モードを使用することができます。

ただし、標準モードを無効にすると、ファイル システムが露呈するためセキュリティ リスクが生じます。

標準モードの無効化 (page 1486)

9.1 ODBC および OLE DB データ接続ベースでファイルに接続する場合のセキュリティ面

ファイルベースのドライバを使用する ODBC および OLE DB データ接続では、接続文字列に、接続したデータ ファイルへのパスが表示されます。このパスは、接続の編集時にデータ選択ダイアログまたは特定の SQL クエリに表示されます。標準モードとレガシー モードの両方で起こります。

データ ファイルへのパスの表示に不安を感じる場合は、可能であればフォルダ データ接続を使用してデータ ファイルに接続することをお勧めします。

9.2 標準モードの制限

標準モードでは、使用できない/制限があるステートメントや変数、関数がいくつかあります。データ ロード スクリプトでサポートされていないステートメントを使用すると、ロード スクリプト実行時にエラーが生じる可能性があります。エラー メッセージは、スクリプト ログ ファイルで確認できます。サポートされていない変数と関数を使用しても、エラーメッセージやログファイルの入力は行われません。代わりに、関数は NULL を返します。

データ ロード スクリプトの編集中には、変数やステートメント、関数がサポートされていないことを通知するメッセージは表示されません。

システム変数

システム変数

変数	標準 モード	レガシー モード	定義
Floppy	対応していません	サポート対象	見つかった最初のフロッピー ドライブのドライブ文字を返します。通常は *a:* です。

変数	標準 モード	レガシー モード	定義
CD	対応していません	サポート対象	見つかった最初の CD-ROM ドライブのドライブ文字を返します。CD-ROM が見つからない場合は、*c:* が返されます。
QvPath	対応していません	サポート対象	Qlik Sense 実行可能ファイルへの参照文字列を返します。
QvRoot	対応していません	サポート対象	Qlik Sense 実行可能ファイルのルートディレクトリを返します。
QvWorkPath	対応していません	サポート対象	現在の Qlik Sense アプリへの参照文字列を返します。
QvWorkRoot	対応していません	サポート対象	現在の Qlik Sense アプリのルートディレクトリを返します。
WinPath	対応していません	サポート対象	Windows への参照文字列を返します。
WinRoot	対応していません	サポート対象	Windows のルートディレクトリを返します。
$(include=...)	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**Include/Must_Include** 変数は、スクリプトにインクルードしてスクリプト コードとして評価する必要があるテキストが格納されたファイルを指定します。データの追加には使用されません。スクリプト コードの一部を別のテキスト ファイルに保存して、複数のアプリで再利用することができます。これはユーザー定義変数です。

一般的なスクリプト ステートメント

一般的なスクリプト ステートメント

ステートメント	標準 モード	レガシー モード	定義
Binary	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	別のアプリからデータをロードするには、**binary** ステートメントが使用されます。
Connect	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**CONNECT** ステートメントは、Qlik Sense が OLE DB/ODBC インターフェースから一般的なデータベースにアクセスする方法を定義する際に使用します。ODBC の場合、まず ODBC アドミニストレータを使用して、データ ソースを指定する必要があります。
Directory	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**Directory** ステートメントは、新たな **Directory** ステートメントが作成されるまで、後続の **LOAD** ステートメントのどのディレクトリでデータ ファイルを検索するか定義します。
Execute	対応していません	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**Execute** ステートメントはその他のプログラムの実行に使用しますが、Qlik Sense ではデータのロードを行います。例えば、必要な変換を行う場合などです。
LOAD from ...	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**LOAD** ステートメントは、ファイル、スクリプトで定義されたデータ、事前にロードされたテーブル、Web ページ、後続の **SELECT** ステートメントの結果、または自動生成されたデータから項目をロードします。

ステートメント	標準 モード	レガシー モード	定義
Store into ...	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**Store** ステートメントは、QVD、Parquet、CSV、または TXT ファイルを作成します。

スクリプト制御ステートメント

スクリプト制御ステートメント

ステートメント	標準 モード	レガシー モード	定義
For each... filelist mask/dirlist mask	サポートされている入力: ライブラリ接続を使用したパス 返されたアウトプット: ライブラリ接続名	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス 返されたアウトプット: ライブラリ接続またはファイル システム パス (入力に応じて)	構文 filelist mask は、**filelist mask** に一致する現在のディレクトリ内にある、すべてのファイルのコンマ区切りリストを生成します。構文 **dirlist mask** は、ディレクトリ名のマスクに一致する現在のディレクトリ内にある、すべてのディレクトリのコンマ区切りリストを生成します。

ファイル関数

ファイル関数

関数	標準 モード	レガシー モード	定義
Attribute()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	異なるメディア ファイルのメタタグの値をテキストとして返します。
ConnectString()	返されたアウトプット: ライブラリ接続名	ライブラリ接続名または実際の接続 (入力に応じて)	ODBC または OLE DB 接続のアクティブな接続文字列を返します。
FileDir()	返されたアウトプット: ライブラリ接続名	返されたアウトプット: ライブラリ接続またはファイル システム パス (入力に応じて)	**FileDir** 関数は、現在読み取り中のテーブル ファイルのディレクトリ パスを文字列で返します。
FilePath()	返されたアウトプット: ライブラリ接続名	返されたアウトプット: ライブラリ接続またはファイル システム パス (入力に応じて)	**FilePath** 関数は、現在読み取り中のテーブル ファイルのフル パスを文字列で返します。

関数	標準 モード	レガシー モード	定義
FileSize()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**FileSize** 関数は、filename ファイル のサイズをバイト数で表した整数を返します。filenameが指定されていない場合は、現在読み取り中のテーブル ファイルのサイズを返します。
FileTime()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	**FileTime** 関数は、指定されたファイルの最後に更新された日付と時刻をUTC フォーマットで返します。ファイルが指定されていない場合、関数は現在読み込まれているテーブル ファイルの最後に更新された日付と時刻を返します。
GetFolderPath()	対応していません	返されたアウトプット: 絶対パス	**GetFolderPath** 関数は、Microsoft Windows *SHGetFolderPath* 関数の値を返します。この関数は、Microsoft Windows フォルダの名前を入力として返し、フォルダのフルパスを返します。
QvdCreateTime()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	このスクリプト関数は、QVD ファイルに含まれた XML ヘッダーの日付と時刻を返します (ない場合は NULL を返します)。タイムスタンプでは、時刻は UTC で提供されます。
QvdFieldName()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	このスクリプト関数は、QVDファイルの項目番号**fieldno**の名前を返します。項目が存在しない場合は、NULLを返します。

関数	標準 モード	レガシー モード	定義
QvdNoOfFields()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	このスクリプト関数は、QVD ファイル内の項目数を返します。
QvdNoOfRecords()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	このスクリプト関数は、QVD ファイル内に含まれるレコードの数を返します。
QvdTableName()	サポートされている入力: ライブラリ接続を使用したパス	サポートされている入力: ライブラリ接続またはファイル システムを使用したパス	このスクリプト関数は、QVD ファイルに保存されているテーブルの名前を返します。

システム関数

システム関数

関数	標準 モード	レガシー モード	定義
DocumentPath()	対応していません	返されたアウトプット: 絶対 パス	この関数は、現在の Qlik Sense アプリへの完全なパスを含む文字列を返します。
GetRegistryString()	対応していません	サポート対象	指定されたレジストリの path にある指定されたレジストリの key の値を返します。この関数は、チャートとロードスクリプトで使用できます。

9.3 標準 モードの無効化

ファイルの絶対 パスや相対 パス、ならびにライブラリ接続を参照するQlikViewロード スクリプトを再利用するために、標準 モードを無効にして、レガシー モードを設定することができます。

ただし、標準 モードを無効にすると、ファイル システムが露呈するためセキュリティ リスクが生じます。

Qlik Sense

Qlik Sense では、**[標準 モード]** プロパティを使用して、QMC で標準 モードを無効にできます。

Qlik Sense Desktop

Qlik Sense Desktop では、*Settings.ini* で標準/レガシー モードを設定できます。

デフォルトのインストール先を使って Qlik Sense Desktop をインストールした場合、*Settings.ini* は *C:\Users\{user}\Documents\Qlik\Sense\Settings.ini* 内にあります。選択したフォルダーに Qlik Sense Desktop をインストールした場合、*Settings.ini* はインストール パスの *Engine* フォルダー内にあります。

次の手順を実行します。

1. テキスト エディタで *Settings.ini* を開きます。
2. *StandardReload=1* を *StandardReload=0* に変更します。
3. ファイルを保存して、Qlik Sense Desktop を起動します。

Qlik Sense Desktop は現在、レガシー モードで実行されています。

設定

StandardReload で利用できる設定は、以下のとおりです。

- 1 (標準モード)
- 0 (レガシー モード)

10 チャートレベルのスクリプト作成

チャートデータを変更する場合、いくつかのステートメントで構成される Qlik Sense スクリプトのサブセットを使用します。ステートメントは、正規のスクリプト ステートメントまたはスクリプト制御 ステートメントのどちらかになります。先頭にプレフィックスが付くステートメントもあります。

一般に正規 ステートメントは、何らかの形でデータの操作に使用されます。これらのステートメントはスクリプト内で何行でも記述できますが、必ずセミコロン「;」で終了する必要があります。

通常、制御 ステートメントはスクリプト実行の流れを制御するために使用されます。制御 ステートメントの各節は 1 つのスクリプト行に収める必要があり、セミコロン「;」または改行 コードで終了する必要があります。

プレフィックスは、必要に応じて正規 ステートメントに適用できますが、制御 ステートメントには適用できません。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。ただし、ステートメントで使用される項目名と変数名は大文字と小文字が区別されます。

このセクションでは、チャートデータを変更する際に使用するスクリプトのサブセットで使用可能なすべてのスクリプト文、制御文、プレフィックスのアルファベット順のリストを見ることができます。

10.1 制御文をコントロールする

チャートデータを変更する場合、いくつかのステートメントで構成される Qlik Sense スクリプトのサブセットを使用します。ステートメントは、正規のスクリプト ステートメントまたはスクリプト制御 ステートメントのどちらかになります。

通常、制御 ステートメントはスクリプト実行の流れを制御するために使用されます。制御 ステートメントの各節は 1 スクリプト行に収める必要があり、セミコロンまたは改行 コードで終了する必要があります。

プレフィックスは制御文に適用されることはありません。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。

チャート修飾の制御文の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Call

call 制御 ステートメントは、事前に **sub** ステートメントで定義されているサブルーチンを呼び出します。

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

Do..loop

do..loop 制御 ステートメントはスクリプト反復構文で、論理条件が満たされるまで、1 つまたは複数のステートメントを実行します。

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

End

End スクリプト キーワードは、**If** 節、**Sub** 節、**Switch** 節を閉じるために使用されます。

Exit

Exit スクリプト キーワードは **Exit Script** ステートメントの一部ですが、**Do** 節、**For** 節、**Sub** 節から抜けるためにも使用されます。

Exit script

この制御ステートメントは、スクリプトの実行を停止します。スクリプト内の任意の場所に挿入できます。

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

For..next

for..next 制御ステートメントは、カウンタ付きのスクリプト反復構文です。**for** と**next** で囲まれたループ内のステートメントは、カウンタ変数の初期値と最終値で指定された回数分実行されます。

```
For..next  counter = expr1 to expr2 [ stepexpr3 ]

[statements]

[exit for [ ( when | unless ) condition ]

[statements]

Next [counter]
```

For each ..next

for each..next 制御ステートメントは、コンマ区切りリストの各値に対して、1 つまたは複数のステートメントを実行するスクリプト反復構文です。**for** と**next** で囲まれたループ内のステートメントは、リストの各値で指定された回数分実行されます。

```
For each..next var in list

[statements]

[exit for [ ( when | unless ) condition ]

[statements]

next [var]
```

If..then

if..then 制御ステートメントは、1 つ以上の論理条件に応じて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

> ***if..then*** *ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、使用可能な 4 つの節 (**if..then**、**elseif..then**、**else**、**end if**) が行をまたぐことはできません。*

```
If..then..elseif..else..end if condition then
```

```
  [ statements ]

{ elseif condition then

  [ statements ] }

[ else

  [ statements ] ]

end if
```

Next
Next スクリプト キーワードは、**For** ループを閉じるために使用されます。

Sub
sub..end sub 制御ステートメントは、**call** ステートメントで呼び出されるサブルーチンを定義します。

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

Switch
switch 制御ステートメントは、数式の値に基づいて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

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

To
To スクリプト キーワードは、次のスクリプト ステートメントで使用されます。

Call

call 制御ステートメントは、事前に **sub** ステートメントで定義されているサブルーチンを呼び出します。

構文:

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

引数:

引数

引数	説明
name	サブルーチンの名前。
paramlist	サブルーチンに送られる実パラメータのコンマ区切りのリスト。リスト内の各アイテムは、項目名、変数、または任意の数式です。

call ステートメントで呼び出されるサブルーチンは、スクリプトの実行中に先に出現する **sub** ステートメントで定義される必要があります。

パラメータはサブルーチンにコピーされます。**call** ステートメントのパラメータが数式ではなく変数の場合、パラメータはサブルーチンが終了したときにコピーして戻されます。

制限事項:

- **call** ステートメントは、制御ステートメントであり、セミコロンまたは改行コードで終わるため、行をまたぐことはできません。
- 例えば [`if..then`] 制御文内の [`Sub..end sub`]を使用してサブルーチンを定義すると、同じ制御文内からはサブルーチンしか呼び出すことはできません。

Do..loop

do..loop 制御ステートメントはスクリプト反復構文で、論理条件が満たされるまで、1 つまたは複数のステートメントを実行します。

構文:

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

> ***do..loop*** *ステートメントは制御ステートメントであり、セミコロンまたは改行コードで終わっているため、使用可能な 3 つの節 (**do**、**exit do**、**loop**) が行をまたぐことはできません。*

引数:

引数

引数	説明
condition	True または False の評価を実施する論理式。
statements	1 つ以上の Qlik Sense スクリプトステートメントのグループ。
while / until	**while** または **until** 条件節は、**do..loop** ステートメントに 1 つだけ必要です (例えば、**do** あるいは **loop** の後)。各条件は、初出の場合に限り解釈されますが、ループ内に出現した場合は毎回評価されます。
exit do	**exit do** 節がループ内で出現した場合、スクリプトの実行はループの終了を示す **loop** 節の後の最初のステートメントに移ります。**exit do** 節は、**when** や **unless** サフィックスを使用して条件を付けることができます。

End

End スクリプトキーワードは、**If** 節、**Sub** 節、**Switch** 節を閉じるために使用されます。

Exit

Exit スクリプト キーワードは **Exit Script** ステートメントの一部ですが、**Do** 節、**For** 節、**Sub** 節から抜けるためにも使用されます。

Exit script

この制御ステートメントは、スクリプトの実行を停止します。スクリプト内の任意の場所に挿入できます。

構文:

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

exit script ステートメントは、制御ステートメントであり、セミコロンまたは改行コードで終わるため、行をまたぐことはできません。

引数:

引数

引数	説明
condition	True または False の評価を実施する論理式。
when / unless	**exit script** ステートメントは、**when** や**unless** 節をオプションで使用して、条件を付けることができます。

```
//Exit script
Exit Script;

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

For..next

for..next 制御ステートメントは、カウンタ付きのスクリプト反復構文です。**for** と**next** で囲まれたループ内のステートメントは、カウンタ変数の初期値と最終値で指定された回数分実行されます。

構文:

```
For counter = expr1 to expr2 [ step expr3 ]

[statements]

[exit for [ ( when | unless ) condition ]

[statements]
```

```
Next [counter]
```

数式 *expr1*、*expr2*、および *expr3* は、ループが最初に挿入される際に評価されます。カウンタ変数の値はループ内のステートメントで変更できますが、これは良いプログラミングとは言えません。

exit for 節がループ内で出現した場合、スクリプトの実行はループの終了を示す **next** 節の後の最初のステートメントに移ります。**exit for** 節は、**when** や **unless** サフィックスを使用して条件を付けることができます。

> ***for..next*** *ステートメントは制御ステートメントであり、セミコロンまたは改行コードで終わっているため、使用可能な 3 つの節 (**for..to..step**、**exit for**、**next**) が行をまたぐことはできません。*

引数:

引数

引数	説明
counter	変数名。*counter* が **next** の後に指定されている場合は、対応する **for** の後に検出されるものと同じ変数名である必要があります。
expr1	ループが実行される *counter* 変数の最初の値を判定する数式。
expr2	ループが実行される *counter* 変数の最後の値を判定する数式。
expr3	ループが実行されるたびに *counter* 変数の増分を示す値を判定する数式。
condition	True または False の評価を実施する論理式。
statements	1 つ以上の Qlik Sense スクリプトステートメントのグループ。

For each..next

for each..next 制御ステートメントは、コンマ区切りリストの各値に対して、1 つまたは複数のステートメントを実行するスクリプト反復構文です。**for** と **next** で囲まれたループ内のステートメントは、リストの各値で指定された回数分実行されます。

構文:
現在のディレクトリ内のファイルとディレクトリ名のリストの生成を可能にする特殊構文です。

```
for each var in list
```

```
[statements]
```

```
[exit for [ ( when | unless ) condition ]
```

```
[statements]
```

```
next [var]
```

引数:

引数

引数	説明
var	ループ実行のたびに、リストから新しい値を取得するスクリプト変数名。**var** が **next** の後に指定されている場合は、対応する **for each** の後に検出されるものと同じ変数名である必要があります。

var 変数の値は、ループ内のステートメントで変更できますが、これは良いプログラミングとは言えません。

exit for 節がループ内で出現した場合、スクリプトの実行はループの終了を示す **next** 節の後の最初のステートメントに移ります。**exit for** 節は、**when** や **unless** サフィックスを使用して条件を付けることができます。

> ***for each..next*** *ステートメントは制御ステートメントであり、セミコロンまたは改行コードで終わっているため、使用可能な 3 つの節 (**for each**、**exit for**、**next**) が行をまたぐことはできません。*

構文:

```
list := item { , item }
```

```
item := constant | (expression) | filelist mask | dirlist mask |
fieldvaluelist mask
```

引数

引数	説明
constant	任意の数値または文字列。スクリプトに直接書き込まれた文字列は単一引用符で囲む必要があります。単一引用符で囲まれていない文字列は、変数として解釈され、変数の値が使用されます。数字は単一引用符で囲む必要はありません。
expression	任意の式。
mask	有効なファイル名の文字や、標準的なワイルドカード文字 ***** と **?** を含むファイル名またはディレクトリ名のマスク。 絶対ファイル パスや lib:// パスを使用できます。
condition	True または False の評価を実施する論理式。
statements	1 つ以上の Qlik Sense スクリプト ステートメントのグループ。
filelist mask	この構文は、ファイル名のマスクに一致する現在のディレクトリ内にある、すべてのファイルのコンマ区切りリストを生成します。 *この引数は、標準モードのライブラリ接続のみをサポートします。*

引数	説明
dirlist mask	この構文は、ディレクトリ名のマスクに一致する現在のフォルダ内にある、すべてのディレクトリのコンマ区切りリストを生成します。  *この引数は、標準モードのライブラリ接続のみをサポートします。*
fieldvaluelist mask	この構文は、Qlik Sense にすでにロードされた項目の値を使って繰り返されます。

 *Qlik Web ストレージプロバイダコネクタとその他の DataFile 接続は、ワイルドカード (***** および **?**) の使用に対応していません。*

Example 1: ファイルのリストのロード

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

Example 2: ファイル リストをディスクに作成

この例では、フォルダにあるすべての Qlik Sense 関連ファイルのリストをロードしています。

```
sub DoDir (Root)
   for each Ext in 'qvw', 'qva', 'qvo', 'qvs', 'qvc', 'qvf', 'qvd'

      for each File in filelist (Root&'/*.' &Ext)

         LOAD
            '$(File)' as Name,
            FileSize( '$(File)' ) as Size,
            FileTime( '$(File)' ) as FileTime
         autogenerate 1;

      next File

   next Ext
   for each Dir in dirlist (Root&'/*' )

      call DoDir (Dir)

   next Dir

end sub

call DoDir ('lib://DataFiles')
```

Example 3: 項目の値を使って繰り返し

この例は、ロードされた値のリストである FIELD を使って繰り返され、新しい項目 NEWFIELD を生成します。FIELD の 1 つの値につき、2 つの NEWFIELD レコードが作成されます。

```
load * inline [
FIELD
one
two
three
];

FOR Each a in FieldValueList('FIELD')
LOAD '$(a)' &'-'&RecNo() as NEWFIELD AutoGenerate 2;
NEXT a
```

この結果、テーブルは次のようになります。

Example table

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

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

if..then 制御ステートメントは、1 つ以上の論理条件に応じて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

通常、制御ステートメントはスクリプト実行の流れを制御するために使用されます。チャートの数式では、代わりに **if** 条件付き関数を使用してください。

構文:

```
If  condition then

  [ statements ]

{ elseif condition then

  [ statements ] }

[ else

  [ statements ] ]
```

```
end if
```

if..then ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、使用可能な 4 つの節 (**if..then**、**elseif..then**、**else**、**end if**) が行をまたぐことはできません。

引数:

引数

引数	説明
condition	True か False で評価できる論理式です。
statements	1 つ以上の Qlik Sense スクリプト ステートメントのグループ。

Example 1:

```
if a=1 then
      LOAD * from abc.csv;

      SQL SELECT e, f, g from tab1;
end if
```

Example 2:

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

Example 3:

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

Next

Next スクリプト キーワードは、**For** ループを閉じるために使用されます。

Sub..end sub

sub..end sub 制御ステートメントは、**call** ステートメントで呼び出されるサブルーチンを定義します。

構文:

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

引数はサブルーチンにコピーされ、**call** ステートメントで対応する実パラメータが変数名の場合は、サブルーチンの終了後、コピーして戻されます。

サブルーチンに **call** ステートメントで渡される実パラメータよりも仮パラメータが多い場合は、余分なパラメータは NULL に初期化され、サブルーチン内でローカル変数として使用できます。

引数:

引数

引数	説明
name	サブルーチンの名前。
paramlist	サブルーチンの仮パラメータの変数名のコンマ区切りリスト。これはサブルーチン内の変数として使用できます。
statements	1 つ以上の Qlik Sense スクリプトステートメントのグループ。

制限事項:

- **sub** ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、2 つの節 (**sub**、**end sub**) が行をまたぐことはできません。
- 例えば [`if..then`] 制御文内の [`Sub..end sub`]を使用してサブルーチンを定義すると、同じ制御文内からはサブルーチンしか呼び出すことはできません。

Example 1:

```
Sub INCR (I,J)

I = I + 1

Exit Sub when I < 10

J = J + 1

End Sub

Call INCR (X,Y)
```

Example 2: - パラメータ転送

```
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)
```

上記の結果、サブルーチン内でローカルに A は 1、B は 4、C は NULL に初期化されます。

サブルーチンを終了する際、グローバル変数 A は 2 を値として取得します (サブルーチンからコピーして返されます)。2 番目の実パラメータ“(X+1)*2” は変数ではないため、コピーして返されません。最後に、グローバル変数 C はサブルーチン呼び出しの影響を受けません。

Switch..case..default..end switch

switch 制御ステートメントは、数式の値に基づいて異なるパスに従うようスクリプトを強制実行させるスクリプト選択構文です。

構文:

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

> ***switch** ステートメントは制御文であり、セミコロンまたは改行コードで終わっているため、使用可能な 4 つの節 (**switch**、**case**、**default**、**end switch**) が行をまたぐことはできません。*

引数:

引数

引数	説明
expression	任意の式。
valuelist	比較される数式の値のコンマ区切りのリスト。スクリプトの実行は、値リストの値が数式の値と等しい最初のグループのステートメントで続行されます。値リストの各値は、任意の数式の場合があります。**case** 節で一致しない場合は、**default** 節 (指定した場合) のステートメントが実行されます。
statements	1 つ以上の Qlik Sense スクリプトステートメントのグループ。

```
Switch I

Case 1

LOAD '$(I): CASE 1' as case autogenerate 1;

Case 2

LOAD '$(I): CASE 2' as case autogenerate 1;

Default
```

```
LOAD '$(I): DEFAULT' as case autogenerate 1;

End Switch
```

To

To スクリプト キーワードは、次のスクリプト ステートメントで使用されます。

10.2 プレフィックス

プレフィックスは、必要に応じて正規ステートメントに適用できますが、制御ステートメントには適用できません。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。ただし、ステートメントで使用される項目名と変数名は大文字と小文字が区別されます。

チャート修飾プレフィックスの概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

Add
スクリプト内の任意の **LOAD** または **SELECT** ステートメントに **Add** プレフィックスを追加して、別のテーブルにレコードを追加するように指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。**Add** プレフィックスは **Map** ステートメントでも使用できます。

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

Replace
Replace プレフィックスをスクリプト内の任意の **LOAD** または **SELECT** ステートメントに追加して、ロードされたテーブルを別のテーブルに置き換えるように指定できます。また、このステートメントを部分的なリロードで実行する必要があることも指定します。**Replace** プレフィックスは **Map** ステートメントでも使用できます。

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

Add

チャート修正の文脈において、**Add** プレフィックスは **LOAD** と共に使用され、*HC1* テーブルに値を追加するために、Qlik Associative Engine により計算されたハイパーキューブを表します。1 つまたは複数の列を指定することができる。欠測値は、Qlik Associative Engine で自動的に埋められます。

構文:

```
Add loadstatement
```

この例では、インラインステートメントから [*Date*] と [*Sales*] のカラムに 2 行を追加しています。

```
Add Load
x as Dates,
y as Sales
```

```
Inline
[
Dates,Sales
2001/09/1,1000
2001/09/10,-300
]
```

Replace

チャート修正の文脈では、**Replace** プレフィックスは、*HC1* テーブルのすべての値をスクリプトで定義された計算値で変更します。

構文:

```
Replace loadstatement
```

この例では、列 z のすべての値を x と y の和で上書きしています。

```
Replace Load
x+y as z
Resident HC1;
```

10.3 一般的なステートメント

一般に正規ステートメントは、何らかの形でデータの操作に使用されます。これらのステートメントはスクリプト内で何行でも記述できますが、必ずセミコロン「;」で終了する必要があります。

スクリプトのキーワードは、いずれも小文字と大文字の組み合わせが可能です。ただし、ステートメントで使用される項目名と変数名は大文字と小文字が区別されます。

チャート修飾の一般的制御文の概要

それぞれの関数についての説明は、概要の後に表示されます。また、構文内の関数名をクリックすると、その関数の詳細を確認できます。

LOAD

チャート修正の文脈では、**LOAD** ステートメントにより、スクリプトで定義されたデータ、または以前にロードされたテーブルからハイパーキューブに追加データをロードします。分析接続からデータをロードすることもできます。

> ***LOAD** ステートメントは、**Replace** または **Add** プレフィックスを持つか、リジェクトされます。*

```
Add | Replace Load [ distinct ] fieldlist

(

inline data [ format-spec ] |

resident table-label
```

```
) | extension pluginname.functionname([script] tabledescription)]

[ where criterion | while criterion ]

[ group by groupbyfieldlist ]

[order by orderbyfieldlist ]
```

Let
let ステートメントは、**set** ステートメントを補完し、スクリプト変数を定義する際に使用します。**let** ステートメントでは、**set** ステートメントとは逆に、変数に代入する前に、スクリプトの実行時に「=」の右側の数式が評価されます。

```
Let variablename=expression
```

Set
set ステートメントは、スクリプト変数を定義する際に使用します。これらは、文字列、パス、ドライバなどの代入に使用されます。

```
Set variablename=string
```

Put
Put ステートメントを使い、ハイパーキューブの数値を設定します。

HCValue
HCValue ステートメントを使用し、指定の列の行にある値を読み出します。

Load

チャート修正の文脈では、**LOAD** ステートメントにより、スクリプトで定義されたデータ、または以前にロードされたテーブルからハイパーキューブに追加データをロードします。分析接続からデータをロードすることもできます。

> ***LOAD** ステートメントは、**Replace** または **Add** プレフィックスを持つか、リジェクトされます。*

構文:

```
Add | Replace LOAD fieldlist

(

inline data [ format-spec ] |

resident table-label

) | extension pluginname.functionname([script] tabledescription)]

[ where criterion | while criterion ]

[ group by groupbyfieldlist ]
```

```
[order by orderbyfieldlist ]
```

引数:

引数

引数	説明
fieldlist	*fieldlist ::= (* | field{, * | field })* ロードする項目のリスト。項目リストとして * を使用すると、テーブルのすべての項目が指定されます。 *field ::= (fieldref | expression) [***as** *aliasname]* 項目定義には、リテラル、既存項目への参照、または数式を含める必要があります。 *fieldref ::= (fieldname |@fieldnumber |@startpos:endpos [* ***I*** | ***U*** | ***R*** | ***B*** | **T**])* *fieldname* は、テーブル内の項目名と同じテキストです。項目名にスペースなどが含まれる場合は、ストレート二重引用符または角括弧で囲む必要があります。明示的に表現できない項目名については、次のような表記規則を使用します。 @*fieldnumber* は、区切り記号付きテーブル ファイルの項目番号を表します。「@」が前に付いた正の整数でなければなりません。常に 1 から項目の数まで、番号が振られています。 @*startpos*:*endpos* は、固定長レコードが含まれるファイル内の項目の開始および終了位置を表します。位置はどちらも正の整数でなければなりません。2 つの番号の前に「@」を付け、コロン (:) で区切る必要があります。常に 1 から位置の数までの番号が付けられます。最後の項目で、**n** は終了位置として使用されます。 • @*startpos*:*endpos* の直後に **I** か **U** の文字が続く場合は、バイトの読み取りは符号付き (**I**) バイナリまたは符号なし (**U**) の整数 (Intel のバイト順) と解釈されます。読み取られる位置の数は、1、2、または 4 です。 • @*startpos*:*endpos*の直後に文字 **R** が続く場合は、読み取られるバイトはバイナリの実数 (IEEE 32 ビットまたは 64 ビットの浮動小数点) として解釈されます。読み取られる位置の数は、4 または 8 です。 • @*startpos*:*endpos* の直後に文字 **B** が続く場合は、読み取られるバイトは COMP-3 標準に従った BCD (Binary Coded Decimal) 数として解釈されます。任意のバイト数を指定できます。 *expression* は、同じテーブルにある 1 つまたは複数の項目に基づいた数値関数または文字列関数です。詳細については、数式の構文を参照してください。 項目に新しい名前を割り当てるには、**as** を使用します。

引数	説明
inline	スクリプト内でデータを入力し、ファイルからロードしない場合は、**inline** を使用します。 *data ::= [text]* **inline** 節を使用してい入力するデータは、二重引用符または角括弧で囲む必要があります。括弧で囲まれたテキストは、ファイルのコンテンツと同じ方法で解釈されます。そのため、テキストファイルで新しい行を挿入する場合と同様に、**inline** 節のテキストについても Enter キーを押します。列の数は最初の行で定義されています。 *format-spec ::= (fspec-item {, fspec-item })* この書式指定は、括弧に囲まれた複数の書式指定アイテムのリストで構成されます。詳細については、「*書式指定アイテム (page 165)*」を参照してください。
resident	事前にロード済みのテーブルからデータをロードする場合は、**resident** を使用します。 *table label* は、元のテーブルを作成した **LOAD** ステートメントの前に配置されるラベルです。ラベルの最後にはコロン (:) を記述します。

引数	説明
extension	分析接続からデータをロードすることができます。サーバーサイド拡張 (SSE) プラグインで定義されている関数を呼び出す、またはスクリプトを評価する **extension** 節を使用する必要があります。 SSE プラグインに単一のテーブルを送ることができます。単一のデータテーブルが返されます。返す項目名が SSE プラグインで指定されていない場合、項目には Field1, Field2 などの名前が付けられます。 `Extension pluginname.functionname( tabledescription );` • SSE プラグインの関数を使用したデータのロード *tabledescription ::= (table { ,tablefield})* テーブルの項目を記述していない場合、項目はロードの順で使用されます。 • SSE プラグイン内のスクリプト評価によるデータのロード *tabledescription ::= (script, table { ,tablefield})* **テーブル項目定義におけるデータ型の扱い** データ型は、分析接続で自動的に検出されます。データに数値が含まれず、少なくとも 1 個の非 NULL テキスト文字列が含まれる場合、その項目はテキストとみなされます。それ以外の場合は数値とみなされます。 **String()** または **Mixed()** で項目名を囲むことで、データ型を強制的に指定できます。 • **String()** は項目をテキストに指定します。項目が数値の場合、デュアル値のテキスト部分が抽出され、変換は実行されません。 • **Mixed()** は項目をデュアルに指定します。 **拡張**テーブル項目定義以外では、**String()**、**Mixed()** は使用できず、テーブル項目定義では他の Qlik Sense 関数を使用できません。
where	**where** 節は、レコードを選択に含めるかどうかを示します。*criterion* が True の場合は選択が含まれます。 *criterion* は論理式です。
while	**while** は、レコードを繰り返し読み取るかどうかを示す節です。*criterion* が True の場合は、同じレコードが読み取られます。通常、**while** 節には **IterNo( )** 関数が含まれていなければなりません。 *criterion* は論理式です。
group by	データを集計 (グループ化) すべき項目を定義するには、**group by** 節を使用します。集計項目は、ロードする数式に何らかの方法で含めてください。集計項目以外の項目は、ロードした数式に含まれる集計関数の外部で使用できます。 *groupbyfieldlist ::= (fieldname { ,fieldname })*

引数	説明
order by	**order by** 節は、**load** ステートメントで処理される前に、常駐テーブルのレコードをソートします。1 つ以上の項目の昇順または降順で、常駐テーブルをソートできます。最初に数値、次に各国の照合順でソートされます。この節は、データソースが常駐テーブルの場合に限り使用できます。 順序項目は、常駐テーブルをソートする項目を指定します。項目は、名前または常駐テーブル内での番号 (最初の項目が番号 1) で指定できます。 orderbyfieldlist ::= fieldname [sortorder] { , fieldname [sortorder] } *sortorder* は、昇順の *asc* または降順の *desc* のどちらかになります。*sortorder* を指定しない場合は、*asc* と見なされます。 *fieldname*、*path*、*filename*、*aliasname* は、それぞれの名前を示すテキスト文字列です。ソーステーブルのフィールドは *fieldname* として使用できます。ただし、as 節 (*aliasname*) を使用して作成された項目は範囲外になり、同じ **load** ステートメント内では使用できません。

Let

let ステートメントは、**set** ステートメントを補完し、スクリプト変数を定義する際に使用します。**let** ステートメントでは、**set** ステートメントとは逆に、変数に代入する前に、スクリプトの実行時に「=」の右側の数式が評価されます。

構文:

```
Let variablename=expression
```

例と結果:

例	結果

`Set x=3+4;`
 	
`Let y=3+4;`
 	
`z=$(y)+1;`
	$(x) は '3+4' として評価されます
	
	$(y) は '7' として評価されます
	
	$(z) は '8' として評価されます
	
	Set ステートメントと**Let** ステートメントの違いに注意してください。**Set** ステートメントは文字列「3+4」を変数に割り当てますが、**Let** ステートメントは文字列を評価して 7 を変数に割り当てます。

Let T=now( );	$(T) には現在の時刻の値が渡されます。

Set

set ステートメントは、スクリプト変数を定義する際に使用します。これらは、文字列、パス、ドライバなどの代入に使用されます。

構文:

```
Set variablename=string
```

Example 1:

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

Example 2:

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

Example 3:

```
Set BudgetYear=2012;
```

Put

put ステートメントを使い、ハイパーキューブの数値を設定します。

列へのアクセスは、ラベルにより行うことができます。また、宣言順で列や行にアクセスすることも可能です。詳しくは下記の例をご覧ください。

構文:

```
put column(position)=value
```

Example 1:

列へのアクセスは、ラベルにより行うことができます。

この例では、[*Sales*] のラベルが付いた列の最初の位置に 1 の値を設定します。

```
Put Sales(1) = 1;
```

Example 2:

メジャー用の #hc1.measure 形式を使用する宣言順でメジャー列にアクセスできます。

この例では、ソートされた最終ハイパーキューブの 10 番目の位置で値 1000 を設定します。

```
Put #hc1.measure.2(10) = 1000;
```

Example 3:

軸の #hc1.dimension 形式を使用して、宣言順で軸の行にアクセスできます。

この例では、定数Piの値を 3 番目に宣言された軸の 5 行目に配置しています。

```
Put #hc1.dimension.3(5) = Pi();
```

そのような軸または式が、値またはラベルに存在しない場合、列が見つからなかったことを示すエラーが返されます。列のインデックスが範囲外の場合、エラーは一切表示されません。

HCValue

HCValue 機能を使用し、指定の列の行にある値を読み出します。

構文:

```
HCValue(column,position)
```

Example 1:

この例では、ラベルが [売上]である列の最初の位置の値を返します。

```
HCValue(Sales,1)
```

Example 2:

この冷では、ソートされたハイパーキューブの 10 番目の位置で値を返します。

```
HCValue(#hc1.measure2,10)
```

Example 3:

この例では、3 番目の軸の 5 行目の値を返します。

```
HCValue(#hc1.dimension.3,5)
```

そのような軸または式が、値またはラベルに存在しない場合、列が見つからなかったことを示すエラーが返されます。列のインデックスが範囲外の場合、NULL が返されます。

11 Qlik Sense が対応していない QlikView 関数とステートメント

QlikView のロードスクリプトとチャートの数式で利用できる大部分の関数とステートメントは、Qlik Sense にも対応していますが、以下で記載するように、例外がいくつかあります。

11.1 Qlik Sense が対応していないスクリプトステートメント

Qlik Sense が対応していない QlikView スクリプトステートメント

ステートメント	コメント
Command	代わりに、**SQL** を使用します。
InputField	

11.2 Qlik Sense が対応していない関数

以下の一覧は、Qlik Sense が対応していない QlikView スクリプト関数とチャート関数を示しています。

- **GetCurrentField**
- **GetExtendedProperty**
- **Input**
- **InputAvg**
- **InputSum**
- **MsgBox**
- **NoOfReports**
- **ReportComment**
- **ReportId**
- **ReportName**
- **ReportNumber**

11.3 Qlik Sense が対応していないプレフィックス

この一覧には、QlikView が対応していない Qlik Sense のプレフィックスが記載されています。

- **Bundle**
- **Image_Size**
- **Info**

12 Qlik Senseでの使用が推奨されていない関数とステートメント

QlikView のロードスクリプトとチャートの数式で利用できる大部分の関数とステートメントは、Qlik Sense でもサポートされていますが、一部の関数とステートメントの Qlik Sense での使用は推奨されていません。以前のバージョンの Qlik Sense で使用可能であって、廃止される可能性のある関数とステートメントもあります。

互換性の理由から、それらの関数とステートメントは動作するようになっていますが、今後のバージョンではサポートされなくなる可能性があるため、このセクションの推奨事項に従ってコードを更新することをお勧めします。

12.1 Qlik Sense での使用が推奨されていないスクリプトステートメント

以下のテーブルでは、Qlik Sense での使用が推奨されていないスクリプトステートメントが含まれています。

推奨されていないスクリプトステートメント

ステートメント	推奨事項
Command	代わりに、**SQL** を使用します。
CustomConnect	代わりに、**Custom Connect** を使用します。

12.2 Qlik Sense での使用が推奨されていないスクリプトステートメントパラメータ

以下のテーブルでは、Qlik Sense での使用が推奨されていないスクリプトステートメントパラメータについて記載しています。

推奨されていないスクリプトステートメントパラメーター

ステートメント	パラメータ
Buffer	次のパラメータの代わりに、**Incremental** を使用します。 • **Inc** (非推奨) • **Incr** (非推奨)

ステートメント	パラメータ
LOAD	以下のパラメータのキーワードは、QlikView ファイル変換ウィザードで生成されます。データがリロードされた場合、機能は維持されますが、Qlik Sense では、以下のパラメータを使用してステートメントを生成するためのガイド付きサポートやウィザードは提供していません。 • **Bottom** • **Cellvalue** • **Col** • **Colmatch** • **Colsplit** • **Colxtr** • **Compound** • **Contain** • **Equal** • **Every** • **Expand** • **Filters** • **Intarray** • **Interpret** • **Length** • **Longer** • **Numerical** • **Pos** • **Remove** • **Rotate** • **Row** • **Rowcnd** • **Shorter**

12.3 Qlik Sense での使用が推奨されていない関数

以下のテーブルでは、Qlik Sense での使用が推奨されていないスクリプト関数とチャート関数について記載しています。

推奨されていない関数

関数	推奨事項
NumAvg **NumCount** **NumMax** **NumMin** **NumSum**	代わりに、範囲関数を使用します。 *範囲関数 (page 1307)*
Color() **QliktechBlue** **QliktechGray**	代わりに、その他のカラー関数を使用します。**QliktechBlue()** は **RGB(8, 18, 90)** に、**QliktechGray** は **RGB(158, 148, 137)** に置き換えると、同じ色にすることができます。 *カラー関数 (page 548)*
QlikViewVersion	代わりに、**EngineVersion** を使用します。 *EngineVersion (page 1452)*
ProductVersion	代わりに、**EngineVersion** を使用します。 *EngineVersion (page 1452)*
QVUser	
Year2Date	代わりに、**YearToDate** を使用します。
Vrank	代わりに、**Rank** を使用します。
WildMatch5	代わりに、**WildMatch** を使用します。

ALL 修飾子

QlikView では、**ALL** 修飾子を数式の前に配置します。これは **{1} TOTAL** を使用するとの同じです。この場合、チャートの軸や現在の選択を無視して、ドキュメント内の項目のすべての値に対して計算を行います。ドキュメントの論理状態に関係なく、常に同じ値が返されます。**ALL** 修飾子が使用された場合、**ALL** 修飾子はそれ自体で SET を定義するので、SET 数式は利用できません。**ALL** 修飾子は、このバージョンの Qlik Sense では機能しますが、今後のバージョンではサポートされなくなる可能性があります。