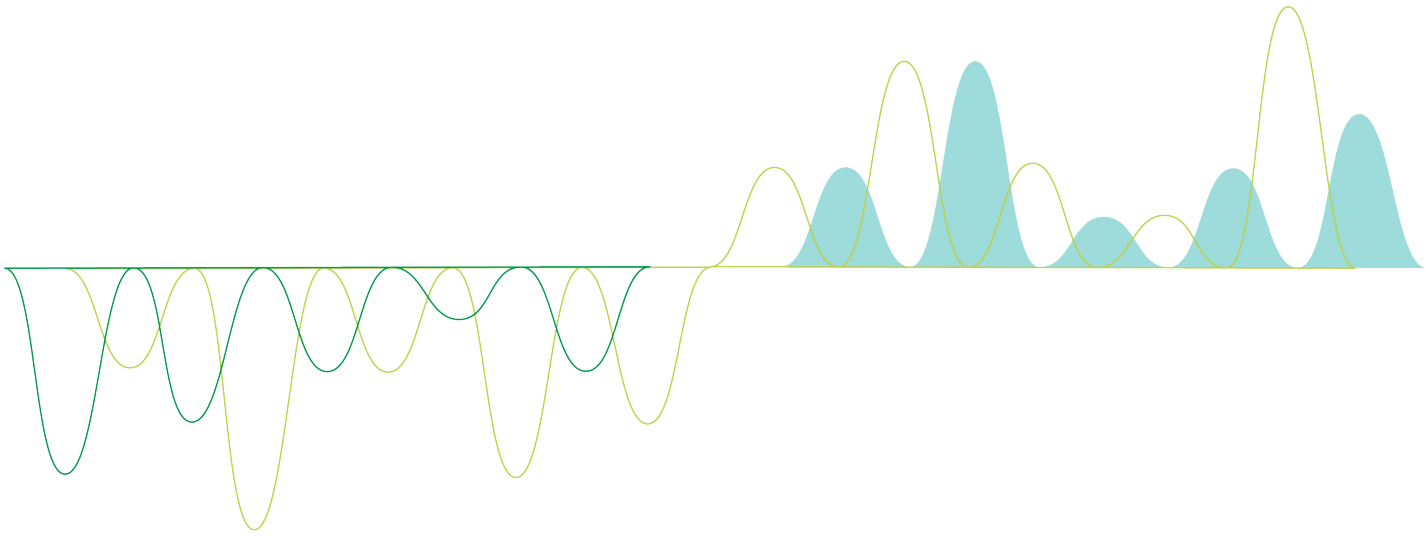
チュートリアル - スクリプトでの次のステップ

Qlik Sense®
May 2022
Copyright © 1993-2022 QlikTech International AB. All rights reserved.



© 2022 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

Contents

1 チュートリアルへようこそ!

このチュートリアルでは、Qlik Sense で高度なスクリプトを作成する方法をご紹介します。

スクリプト作成の基本を習得すると、Qlik Sense にデータをロードする際に、より高度な操作を実行できるようになります。これにはたとえば、クロス集計を使用したデータの変換、データのクレンジング、QVD ファイルとして知られる Qlik データファイルからのデータの作成およびロードなどが含まれます。

1.1 学習内容

After completing this tutorial, you should be comfortable with loading data using some of the more advanced scripting functions in Qlik Sense.

1.2 対象ユーザー

Qlik Sense のスクリプト作成の基本を習得している必要があります。具体的には、スクリプトを使用し、データをロードして操作したことがある方が対象です。

未経験の方は、初心者向けチュートリアルを完了することを推奨します。

データロードエディターへのアクセス権があり、Qlik Sense Enterprise on Windows でデータをロードできる必要があります。

上記の指示は通常、Qlik Sense Cloud Business にも適用されます。

1.3 パッケージ内容

ダウンロードした zip パッケージには、チュートリアルを完了するために必要な次のデータファイルが含まれています:

- *Cutlery.xlsx*
- *Data.xlsx*
- *Events.txt*
- *Employees.xlsx*
- *Intervals.txt*
- *Product.xlsx*
- *Salesman.xlsx*
- *Transactions.csv*
- *Winedistricts.txt*

また、パッケージには *高度なスクリプトチュートリアル* アプリも含まれています。アプリ内の追加のスクリプトセクションに、このチュートリアルで作成するその他のアプリのスクリプトが含まれています。ハブにアプリをアップロードできます。

学習効果を最大限に高めるため、チュートリアルの説明どおりに自分自身でアプリを構築することを推奨します。また、データロードを機能させるには、チュートリアルの説明どおりに、データファイルをアップロードしてデータファイルに接続する必要があります。

ただし、問題が発生した場合は、アプリがトラブルシューティングに役立つ場合があります。各レッスンにどのスクリプトセグメントが関連しているのかを示しました。

1.4 このチュートリアルでのレッスン

Qlik Sense の使用経験により異なりますが、このチュートリアルの所要時間は 3 ~ 4 時間です。トピックは、順番に完了するよう構成されています。ただし、トピックをとばして、後で戻ることもできます。幸い、テストはありません。

データの変換
Crosstable プレフィックスの使用
Join と Keep を使用したテーブルの結合
レコード間関数の使用: Peek、Previous、Exist
間隔の一致と反復ロード
データクレンジング
階層データの処理
QVD ファイル

1.5 追加の資料とリソース

- Qlik では、さらなる詳細情報を提供する、広範なリソースをご用意しています。
- Qlik Sense オンライン ヘルプ を利用できます。
- 無料のオンライン コースを含むトレーニングは、Qlik Continuous Classroom で利用できます。
- ディスカッション フォーラム、ブログなどは、Qlik Community にあります。

1 LOAD とSELECT ステートメント

LOAD および SELECT ステートメントを使用すると、Qlik Sense にデータをロードできます。これらの各ステートメントにより、内部テーブルが生成されます。LOAD はファイルからのデータのロードに使用されるのに対し、SELECT はデータベースからのデータのロードに使用されます。

このチュートリアルでは、ファイルのデータを使用するため、LOAD ステートメントを使用します。

先行する LOAD を使用して、ロードするデータのコンテンツを操作することもできます。たとえば、LOAD ステートメントで項目名を変更する必要があり、SELECT ステートメントでの項目名の変更を許可しないようにします。

以下の規則はデータを Qlik Sense にロードする際に適用されます。

- Qlik Sense では、LOAD ステートメントで生成されるテーブルとSELECT ステートメントで生成されるテーブルに違いはありません。つまり、複数のテーブルをロードする場合、LOAD とSELECT のいずれか一方のステートメントを使用するか、これら2 つを組み合わせて使用するかは問題にはなりません。
- Qlik Sense ロジックでは、ステートメント内またはデータベース内の元のテーブルの項目の順序は重要ではありません。
- 項目名は大文字小文字が区別され、データテーブル間の関連付けに使用されます。このため、適切なデータモデルを作成するには、ロードスクリプトで項目名を変更する必要がある場合があります。

2 データの変換

アプリでデータを使用する前に、データ ロード エディターでデータを変換および操作できます。

データ操作のメリットの 1 つは、ファイルからデータのサブセットのみをロードするという選択ができることです。つまり、テーブルからいくつかの列だけを選択してより効率的にデータを処理できます。また、データを複数回に渡りロードして、生データをいくつかの新しい論理テーブルに分化することもできます。さらに、複数のソースからデータをロードして、Qlik Sense にある 1 つのテーブルに結合することも可能です。

次の演習では、Crosstable プレフィックスを使用してデータをロードする方法について説明します。また、テーブルの結合、レコード間関数 (Peek および Previous) の使用、While Load を使って同じ行を何回もロードする方法についても学習します。

2.1 Crosstable プレフィックスの使用

クロス集計は、ヘッダー データが直交する 2 つのリストに値のマトリックスを持つ一般的なテーブルの種類で、データのクロス集計を使用する場合は、常に Crosstable プレフィックスを使用して、データの変換や希望する項目の作成ができます。

Crosstable プレフィックス

次の *Product* テーブルには、Month (月) ごとに 1 列、Product (製品) ごとに 1 行あります。

Product テーブル

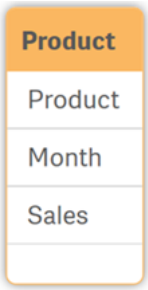
製品	Jan 2014	Feb 2014	Mar 2014	Apr 2014	May 2014	Jun 2014
A	100	98	100	83	103	82
B	284	279	297	305	294	292
C	50	53	50	54	49	51

このテーブルをロードすると、*Product* とそれぞれの月に 1 項目ずつ存在するテーブルが生成されます。

Product 項目と、月ごとに 1 項目ずつある Product テーブル

このデータを分析する場合は、すべての数値が 1 つの項目にあり、別の項目にすべての月がある方がはるかに容易です。この例の場合、各 *カテゴリ*(*Product, Month, Sales*) ごとに 1 つの列を持つ 3 列構成のテーブルです。

Product、*Month*、*Sales* の各項目がある *Product* テーブル



Crosstable プレフィックスは、データを 1 列が *Month*、もう1 つが *Sales* を表すテーブルに変換します。要するに、項目名を取得して項目値に変換します。

次の手順を実行します。

1. 新しいアプリを作成し、*高度なスクリプト チュートリアル*という名前を付けます。
2. **データ ロード エディター**で新しいスクリプト セクションを追加します。
3. *Product* セクションに名前を付けます。
4. 右のメニューの **[AttachedFiles]** で、**[データを選択]** をクリックします。
5. *Product.xlsx* をアップロードして選択します。
6. **[Select data from]** (データの選択元) ウィンドウで、*Product* テーブルを選択します。

   > *[項目名]* で *[埋め込まれた項目名]* が選択されていることを確認します。このオプションが選択されていると、データをロードする際にテーブルの項目名が含まれます。

7. **[スクリプトを挿入]** をクリックします。
   これで、スクリプトは次のようになります。

   ```
   LOAD     Product,     "Jan 2014",     "Feb 2014",     "Mar 2014",     "Apr 2014",
   "May 2014",     "Jun 2014" FROM [lib://AttachedFiles/Product.xlsx]  (ooxml, embedded
   labels, table is Product);
   ```

8. **[データのロード]** をクリックします。
9. **[データ モデル ビューア]** を開きます。データ モデルは次のようになります。

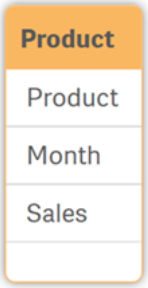
Product 項目 と、月 ごとに 1 項目 ずつある Product テーブル

10. **データ ロード エディター**で、[*Product*] タブをクリックします。
11. LOAD ステートメントの上に次の内容を入力します。

```
CrossTable(Month, Sales)
```

12. [**データのロード**] をクリックします。
13. [**データ モデル ビューア**] を開きます。データ モデルは次のようになります。

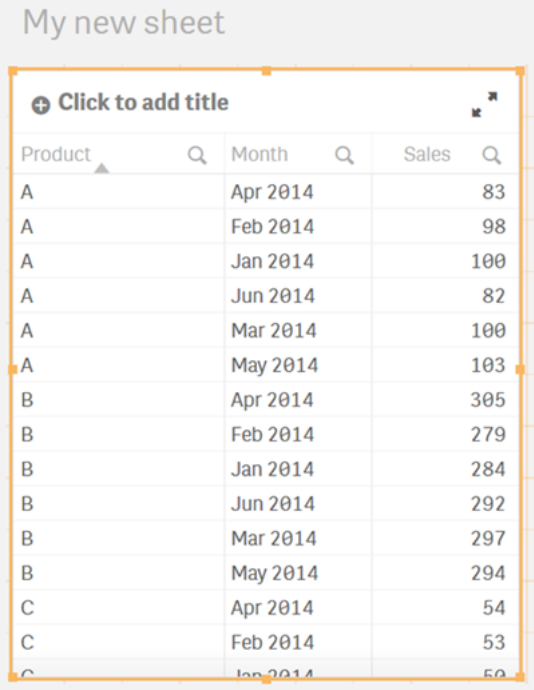
Product、Month、Sales の各項目がある Product テーブル



通常入力データには、内部キー (上の例では *Product*) として機能する修飾子項目の列が 1 つだけあります。ですが、ここでは複数持つことが可能です。その場合は、LOAD ステートメントの属性項目の前にすべての補助項目をリストし、Crosstable プレフィックスの 3 番目のパラメータを補助項目の数を定義するために使用する必要があります。Crosstable キーワードの前に先行する LOAD またはプレフィックスを持つことはできません。ただし、自動連結を使用できます。

Qlik Sense 内のテーブルで、データは次のようになります。

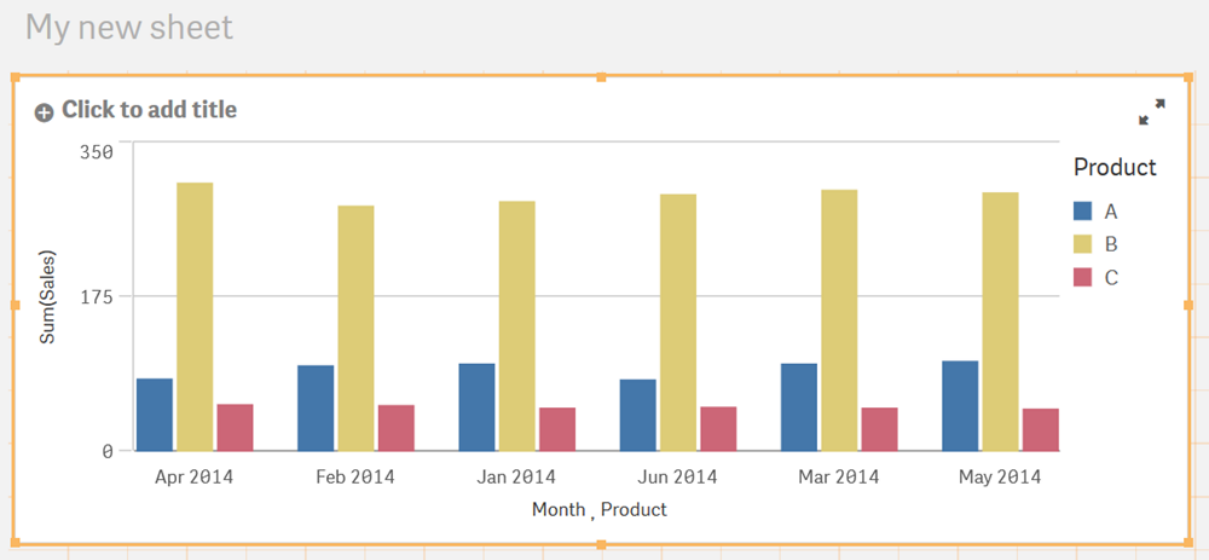
Crosstable プレフィックスを使用してロードしたデータのテーブル



Product	Month	Sales
A	Apr 2014	83
A	Feb 2014	98
A	Jan 2014	100
A	Jun 2014	82
A	Mar 2014	100
A	May 2014	103
B	Apr 2014	305
B	Feb 2014	279
B	Jan 2014	284
B	Jun 2014	292
B	Mar 2014	297
B	May 2014	294
C	Apr 2014	54
C	Feb 2014	53

これで、たとえば、データを使用して棒グラフを作成できるようになりました。

Crosstable プレフィックスを使用してロードしたデータの棒グラフ



Crosstable について詳しくは、Qlik Community の次のブログ投稿を参照してください。「Crosstable ロード」。動作は QlikView の状況に応じて説明されていますが、ロジックは Qlik Sense にもそのまま当てはまります。

属性項目では、数値解釈は行われません。これは、列のヘッダーが月である場合、自動的に解釈されないことを意味します。この回避策としては、Crosstable プレフィックスを使用して一時的なテーブルを作成し、次の例のように 2 回目のパススルーを実行して解釈を行います。

これはあくまで例です。Qlik Sense で行う付随する演習はありません。

```
tmpData: Crosstable (MonthText, Sales) LOAD Product, [Jan 2014], [Feb 2014], [Mar 2014], [Apr 2014], [May 2014], [Jun 2014]            FROM ...    Final: LOAD Product, Date(Date#(MonthText,'MMM YYYY'),'MMM YYYY') as Month, Sales Resident tmpData; Drop Table tmpData;
```

メモリキャッシュのクリア

作成するテーブルを削除し、メモリキャッシュをクリアできます。前のセクションのようにデータを一時的なテーブルにロードした場合、不要になったら削除しなければなりません。例:

```
DROP TABLE Table1, Table2, Table3, Table4; DROP TABLES Table1, Table2, Table3, Table4;
```

項目を削除することもできます。例:

```
DROP FIELD Field1, Field2, Field3, Field4; DROP FIELDS Field1, Field2, Field3, Field4; DROP FIELD Field1 from Table1; DROP FIELDS Field1 from Table1;
```

ご覧のとおり、キーワードTABLE とFIELD は単数形でも複数形でも使用できます。

2.2 Join とKeep を使用したテーブルの結合

結合とは 2 つのテーブルを1 つにまとめる操作です。結合により生成されるテーブルのレコードは元のテーブルのレコードを組み合わせたもので、通常、生成されたテーブル内の特定の組み合わせに関係する2 つのレコードが1 つ以上の共通項目に共通の値を持っています。そのため、これを「自然結合」と呼びます。Qlik Sense では、結合はスクリプトで実行され、論理テーブルが作成されます。

スクリプト内で事前にテーブルを結合することができます。その場合、Qlik Sense ロジックは、個々のテーブルではなく、結合の結果である単一の内部テーブルを処理するようになります。この処理が必要な場合もありますが、次のような欠点があります。

- ロードされたテーブルのサイズが大きくなり、Qlik Sense の動作が遅くなることがあります。
- 元のテーブル内のレコード数を取得できなくなり、情報の一部が失われる可能性があります。

Keep 機能は、Qlik Sense にテーブルを保存する前に、2 つのテーブルの片方または両方をテーブル データの共通部分に縮小させる効果を持ち、明示的な参加を使用する必要がある場合の数を減らすために設計されています。

このマニュアルでは、結合という用語は、内部テーブルが作成される前に行われる結合という意味で使用されています。しかし、内部テーブルが作成された後で行われる関連付けも、基本的には結合です。

Join

結合を作成する最も簡単な方法は、スクリプトで Join プレフィックスを使用することです。これは、内部テーブルを別の名前のテーブルまたは最後に作成されたテーブルに結合します。この結合は外部結合になり、2 つのテーブルからのすべての可能な値の組み合わせを作成します。

```
LOAD  a, b, c from table1.csv; join LOAD a, d from table2.csv;
```

その結果の内部テーブルには、項目 a、b、c、d が含まれています。レコード数は、2 つのテーブルの項目値によって異なります。

結合される項目の名前は、完全に同じである必要があります。結合される項目数は任意です。通常、テーブルには 1 つまたは複数の共通する項目があります。共通する項目が 1 つもない場合、テーブルのデカルト積は生成されません。すべての項目が共通することも考えられますが、通常これは意味がありません。Join ステートメントで、先にロードされたテーブルの名前が指定されていない限り、Join プレフィックスは最後に作成されたテーブルを使用します。そのため、2 つのステートメントの順序を任意に指定することはできません。

Join の使用

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。このような結合を行うと、テーブルが非常に大きくなることがよくあります。

次の手順を実行します。

1. *高度なスクリプト チュートリアル アプリ*を開きます。
2. **データ ロード エディター**で新しいスクリプト セクションを追加します。
3. このセクションを *Transactions* と呼びます。
4. 右のメニューの **[AttachedFiles]** で、**[データを選択]** をクリックします。
5. *Transactions.csv* をアップロードして選択します。

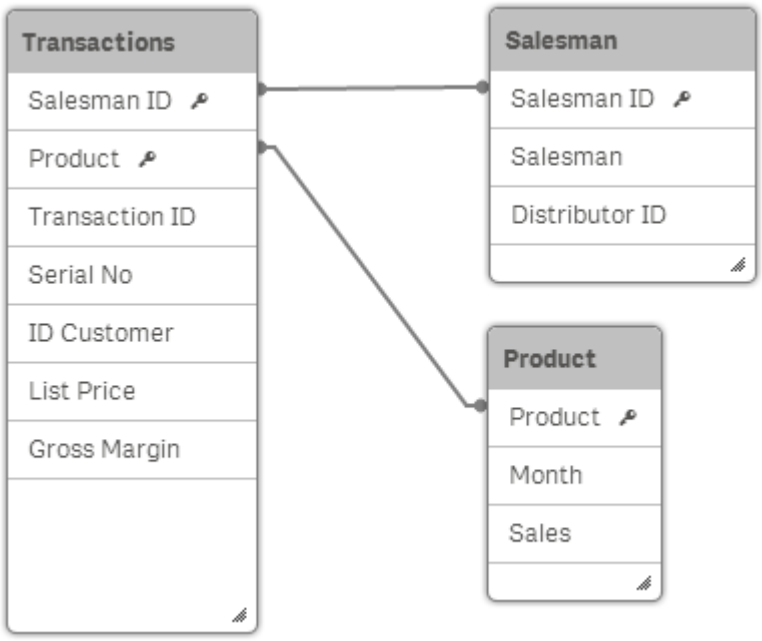
   [項目名] で *[埋め込まれた項目名]* が選択されていることを確認します。このオプションが選択されていると、データをロードする際にテーブルの項目名が含まれます。

6. **[Select data from]** (データの選択元) ウィンドウで、**[スクリプトを挿入]** をクリックします。
7. *Salesman.xlsx* をアップロードして選択します。
8. **[Select data from]** (データの選択元) ウィンドウで、**[スクリプトを挿入]** をクリックします。
   これで、スクリプトは次のようになります。

   ```
   LOAD     "Transaction ID",     "Salesman ID",     Product,     "Serial No",     "ID
   Customer",     "List Price",     "Gross Margin" FROM
   [lib://AttachedFiles/Transactions.csv] (txt, codepage is 28591, embedded labels,
   delimiter is ',', msq);  LOAD     "Salesman ID",     Salesman,     "Distributor ID" FROM
   [lib://AttachedFiles/Salesman.xlsx] (ooxml, embedded labels, table is Salesman);
   ```

9. **[データのロード]** をクリックします。
10. **[データ モデル ビューア]** を開きます。データ モデルは次のようになります。

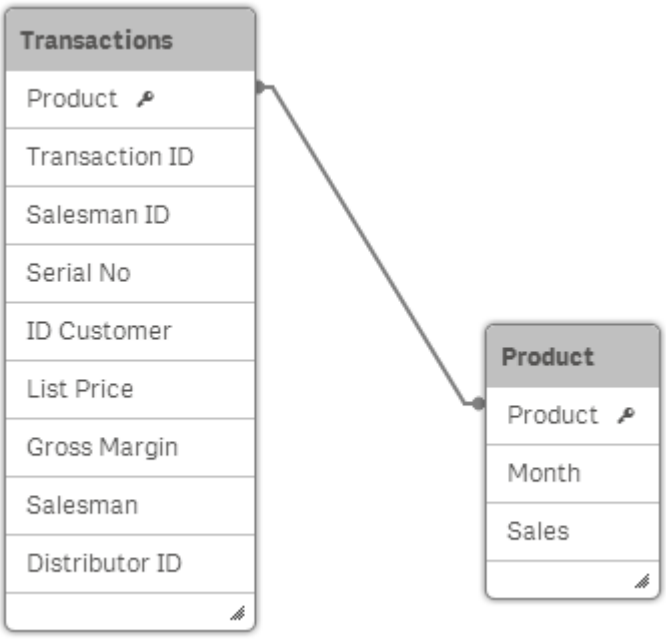
データ モデル:Transactions、Salesman、および Product テーブル



ただし、*Transactions* と *Salesman* のテーブルを分離すると、必要な結果が得られないことがあります。この 2 つのテーブルは結合した方がよいでしょう。

次の手順を実行します。

1. 結合したテーブルの名前を設定するには、最初の LOAD ステートメントの上に次の行を追加します。
   ```
   Transactions:
   ```
2. *Transactions* と *Salesman* テーブルを結合するには、2 番目の LOAD ステートメントの上に次の行を追加します。
   ```
   Join(Transactions)
   ```
   これで、スクリプトは次のようになります。
   ```
   Transactions: LOAD     "Transaction ID",     "Salesman ID",     Product,     "Serial
   No",     "ID Customer",     "List Price",     "Gross Margin" FROM
   [lib://AttachedFiles/Transactions.csv] (txt, codepage is 28591, embedded labels,
   delimiter is ',', msq);  Join(Transactions) LOAD     "Salesman ID",     Salesman,
   "Distributor ID" FROM [lib://AttachedFiles/Salesman.xlsx] (ooxml, embedded labels, table
   is Salesman);
   ```
3. **[データのロード]** をクリックします。
4. **[データ モデル ビューア]** を開きます。データ モデルは次のようになります。
   データ モデル: Transactions および Product テーブル



これで *Transactions* および *Salesman* テーブルのすべての項目が、単一の *Transactions* テーブルに結合されました。

Join をいつ使用するかについて詳しくは、Qlik Community の次のブログ投稿を参照してください。「To Join or not to join」(結合するかしないか)、「Mapping as an ALternative to Joining」(結合の代わりにマッピングする)。動作は QlikView の状況に応じて説明されていますが、ロジックは Qlik Sense にもそのまま当てはまります。

Keep

Qlik Sense の主な機能の 1 つは、テーブルを結合する代わりにテーブル間に関連付けを作成できることです。これにより、メモリの使用量が削減され、速度が向上し、柔軟性が大幅に高まります。Keep の機能は、明示的な結合の使用回数を減らす目的で設計されています。

2 つの LOAD または SELECT ステートメント間の Keep プレフィックスは、Qlik Sense にテーブルが保存される前に、2 つのテーブルの片方または両方をテーブル データの共通部分に縮小させます。Keep プレフィックスの前には、Inner、Left、または Right のいずれかのキーワードを置く必要があります。テーブルからのレコードの選択は、対応する結合と同じ方法で行われます。ただし、2 つのテーブルは結合されず、個別のテーブルとしてそれぞれ異なる名前で Qlik Sense に保存されます。

Inner

データ ロード スクリプトの Join および Keep プレフィックスの前に、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、2 つのテーブル間の結合が内部結合であると指定されます。結果のテーブルには、両方のテーブルのすべてのデータ セットのうち、2 つのテーブル間で共通するものだけが含まれます。

Keep の前に使用すると、Qlik Sense に保存される前に、2 つのテーブルは互いの共通部分に縮小されます。

これらの例では、*Table1* と *Table2* のソース テーブルを使用しています。

これらはあくまで例です。Qlik Sense で行う付随する演習はありません。

Table 1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

Inner Join

まず、これらのテーブルで Inner Join を実行すると、*VTable* が作成されます。このテーブルには 2 つのテーブルのデータを結合した結果、1 行だけ、両方のテーブルに存在するレコードだけが表示されます。

```
VTable: SELECT * from Table1; inner join SELECT * from Table2;
```

VTable

A	B	C
1	aa	xx

Inner Keep

それに対して、Inner Keep を実行すると、2 つのテーブルが残ります。これらの 2 つのテーブルは、共通項目 *A* を介して関連付けられます。

```
VTab1: SELECT * from Table1; VTab2: inner keep SELECT * from Table2;
```

VTab1

A	B
1	aa

VTab2

A	C
1	xx

Left

データ ロード スクリプトの Join および Keep プレフィックスの前に、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、2 つのテーブル間の結合が左結合であると指定されます。結果のテーブルには、1 つ目のテーブルのすべてのデータ セットと 2 つのテーブルの共通するものだけが含まれます。

Keep の前に使用すると、Qlik Sense に保存される前に、2 つ目のテーブルが 1 つ目のテーブルとの共通部分に縮小されます。

これらの例では、*Table1* と *Table2* のソース テーブルを使用しています。

Table1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

まず、これらのテーブルで Left Join を実行すると、*VTable* が生成されます。このテーブルには、*Table1* のすべての行に加え、*Table2* の一致行の項目が含まれています。

```
VTable: SELECT * from Table1; left join SELECT * from Table2;
```

VTable

A	B	C
1	aa	xx
2	cc	-
3	ee	-

それに対して、Left Keep を実行すると、2 つのテーブルが残ります。これらの 2 つのテーブルは、共通項目 *A* を介して関連付けられます。

```
VTab1: SELECT * from Table1; VTab2: left keep SELECT * from Table2;
```

VTab1

A	B
1	aa
2	cc
3	ee

VTab2

A	C
1	xx

Right

Qlik Sense スクリプト言語内の Join および Keep プレフィックスの前に、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、2 つのテーブル間の結合が右結合であると指定されます。結果のテーブルには、2 つ目のテーブルのすべてのデータと2 つのテーブルで共通するものだけが含まれます。

Keep の前に使用すると、Qlik Sense に保存される前に、1 つ目のテーブルが 2 つ目のテーブルとの共通部分に縮小されます。

これらの例では、*Table1* と *Table2* のソース テーブルを使用しています。

Table1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

まず、これらのテーブルで Right Join を実行すると、*VTable* が生成されます。このテーブルには、*Table2* のすべての行に加え、*Table1* の一致行の項目が含まれています。

```
VTable: SELECT * from Table1; right join SELECT * from Table2;
```

VTable

A	B	C
1	aa	xx
4	-	yy

それに対して、Right Keep を実行すると、2 つのテーブルが残ります。これらの 2 つのテーブルは、共通項目 *A* を介して関連付けられます。

```
VTab1: SELECT * from Table1; VTab2: right keep SELECT * from Table2;
```

VTab1

A	B
1	aa

VTab2

A	C
1	xx
4	yy

2.3 レコード間関数の使用:Peek、Previous、および Exists

これらの関数は、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に使用します。

チュートリアルのここでは、Peek()、Previous()、および Exists() 関数を検証します。

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。

構文:

```
Peek(fieldname [ , row [ , tablename ] ] )
```

row は整数にする必要があります。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 が指定されていない場合は、-1 として処理されます。

Tablename は、末尾にコロンが付いていない、テーブルのラベルです。*tablename* 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、*tablename* が含まれている必要があります。

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(expression)
```

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ができます。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 (つまり、関連するデータベースに保存されていなくても) 項目を参照することもできます。

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。

構文:

```
Exists(field [, expression ] )
```

項目は、スクリプトによってこれまでにロードされたデータの中になければなりません。*Expression* は、指定した項目において、検索対象項目値として評価される数式です。省略されると、指定された項目の現在のレコード値が使用されます。

Peek() および Previous() の使用

もっとも簡単な形式では、Peek() および Previous() はテーブル内で特定の値を指定するために使用されます。この演習でロードする *Employees* テーブル内のサンプルデータを示します。

従業員テーブルのサンプル データ

Date	Hired	Terminated
1/1/2011	6	0
2/1/2011	4	2
3/1/2011	6	1
4/1/2011	5	2

現在、これは月、採用、退職などのデータのみを集計しており、Peek() とPrevious() 関数を使ってこれに *Employee Count* と *Employee Var* の項目を追加して、従業員総数の月ごとの変化を確認します。

次の手順を実行します。

1. *高度なスクリプト チュートリアル* アプリを開きます。
2. **データ ロード エディター**で新しいスクリプトセクションを追加します。
3. このセクションを *Employees* と呼びます。
4. 右のメニューの **[AttachedFiles]** で、**[データを選択]** をクリックします。

5. *Employees.xlsx* をアップロードして選択します。

   [Field names] で *[Embedded field names]* が選択されていることを確認します。このオプションが選択されていると、データをロードする際にテーブルの項目名が含まれます。

6. **[Select data from]** (データの選択元) ウィンドウで、**[スクリプトを挿入]** をクリックします。
   これで、スクリプトは次のようになります。

   ```
   LOAD        "Date",     Hired,     Terminated FROM [lib://AttachedFiles/Employees.xlsx]
   (ooxml, embedded labels, table is Sheet1);
   ```

7. スクリプトを、次のように修正します。
   ```
   [Employees Init]: LOAD     rowno() as Row,     Date(Date) as Date,     Hired,
   Terminated,     If(rowno()=1, Hired-Terminated,  peek([Employee Count], -1)+(Hired-
   Terminated)) as [Employee Count] FROM [lib://AttachedFiles/Employees.xlsx]
   embedded labels, table is Sheet1);
   ```
   Date シートの Excel 項目の日付は、MM/DD/YYYY の書式です。日付がこの書式を使用してシステム変数から正しく解釈されていることを確認するには、Date 関数を *Date* 項目に適用します。

   Peek() 関数が、定義済みの項目にロードされた値を特定します。この数式では、最初に rowno() が 1 と等しいかどうかを確認します。1 と等しい場合、*Employee Count* は存在しないため、*Hired - Terminated* の差をその項目に入力します。

   rowno() が 1 より大きい場合は、前月の *Employee Count* を確認し、その数値をその月の *Hired - Terminated* の値に加算します。

   Peek() 関数に (-1) を使用していることに注意してください。これは、Qlik Sense に現在のレコードを上回るレコードを確認するよう指示しています。(-1) が指定されていない場合は、Qlik Sense は前のレコードを確認することを意味します。

8. 次の内容をスクリプトの最後に追加します。
   ```
   [Employee Count]: LOAD  Row,    Date,   Hired,  Terminated,     [Employee Count],       If(rowno
   ()=1,0,[Employee Count]-Previous([Employee Count])) as [Employee Var] Resident
   [Employees Init] Order By Row asc; Drop Table [Employees Init];
   ```

   Previous() 関数が、定義済みの項目にロードされた最新の値を特定します。この数式では、最初に rowno() が 1 と等しいかどうかを確認します。1 と等しい場合は、前月の *Employee Count* のレコードが存在せず、それゆえ Employee Var は存在しないため、値として単に 0 を入力します。

   rowno()が 1 より大きい場合、*Employee Var*が存在していることを示すので、先月の *Employee Count* を確認し、その数値を現在の月の *Employee Count* から差し引いて *Employee Var* 項目の値を作成します。

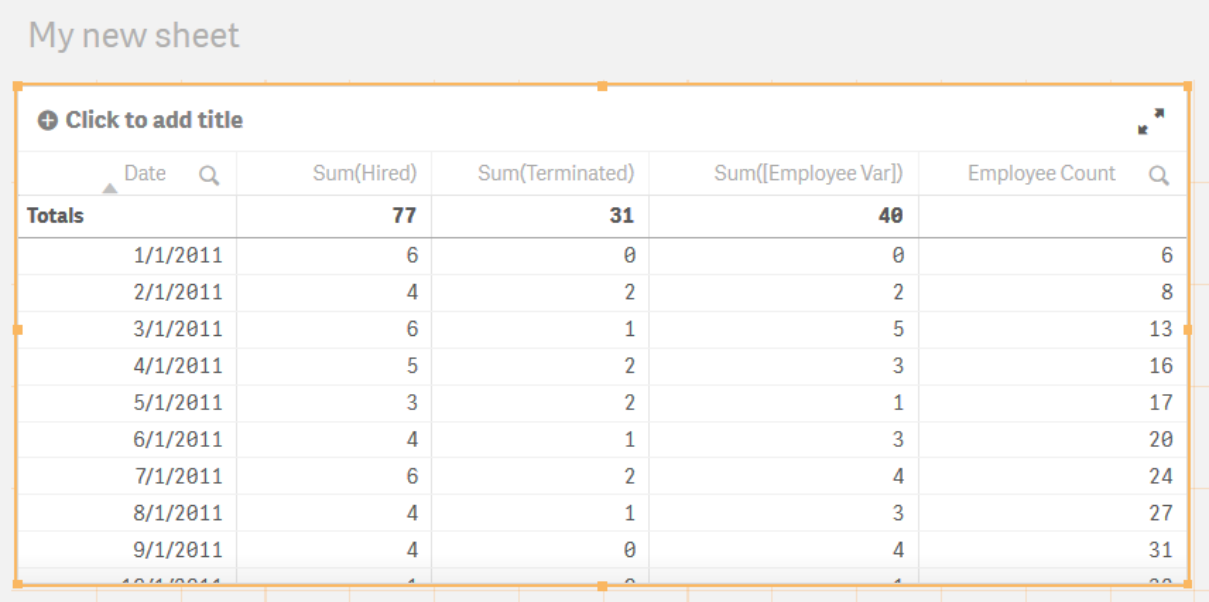
   これで、スクリプトは次のようになります。

   ```
   [Employees Init]: LOAD     rowno() as Row,     Date(Date) as Date,     Hired,
   Terminated,     If(rowno()=1, Hired-Terminated,  peek([Employee Count], -1)+(Hired-
   Terminated)) as [Employee Count] FROM [lib://AttachedFiles/Employees.xlsx] (ooxml,
   embedded labels, table is Sheet1);  [Employee Count]: LOAD     Row,     Date,     Hired,
       Terminated,     [Employee Count],     If(rowno()=1,0,[Employee Count]-Previous
   ([Employee Count])) as [Employee Var] Resident [Employees Init] Order By Row asc;       Drop
   ```

```
Table [Employees Init];
```

9. **[データのロード]** をクリックします。
   アプリ概要の新しいシートで、テーブルの列として *Date, Hired, Terminated, Employee Count* および *Employee Var* を使用してテーブルを作成します。この結果、テーブルは次のようになります。

   スクリプトで Peek と Previous を使用した後のテーブル



Date	Sum(Hired)	Sum(Terminated)	Sum([Employee Var])	Employee Count
Totals	**77**	**31**	**40**	
1/1/2011	6	0	0	6
2/1/2011	4	2	2	8
3/1/2011	6	1	5	13
4/1/2011	5	2	3	16
5/1/2011	3	2	1	17
6/1/2011	4	1	3	20
7/1/2011	6	2	4	24
8/1/2011	4	1	3	27
9/1/2011	4	0	4	31

Peek() とPrevious() を使用すると、テーブル内の定義済み行を対象にすることができます。この 2 つの関数の最大の違いは、Peek() 関数ではユーザーがスクリプトにあらかじめロードされていない項目を参照することができますが Previous() 関数はロード済みの項目しか参照できない点です。Previous() は、LOAD ステートメントへの入力で値を返し、Peek() は LOAD ステートメントの出力で値を返します。(RecNo() とRowNo() の違いと同じ。) これは、Where 節を使用すると、2 つの関数が異なる動作をすることを意味します。

Previous() 関数は、ユーザーが現在の値と前の値を表示する必要がある場合に、より適しています。例では、月によって異なる従業員の分散状況を計算しています。

Peek() 関数は、ユーザーがテーブルに事前にロードされていない項目を対象とする場合、または特定の行を対象とする必要がある場合のいずれにも、より適しています。これは、前月の *Employee Count* をピークすることで *Employee Count* を計算し、今月の雇用者と退職者数の差異を加算している例で示しています。*Employee Count* は、オリジナル ファイルの項目ではないことに注意が必要です。

> *Peek() とPrevious() を使用するタイミングについて詳しくは、Qlik Community の次のブログ投稿を参照してください。「Peek() vs Previous() - When to Use Each」動作は QlikView の状況に応じて説明されていますが、ロジックは Qlik Sense にもそのまま当てはまります。*

Exists() の使用

Exists() 関数は、スクリプトにおいて Where 節とともに使用され、データモデルにすでにロードされている関連データがある場合にデータをロードします。

次の例では、Dual() 関数を使用して文字列に数値を割り当てています。

次の手順を実行します。

1. 新しいアプリを作成し、名前を付けます。
2. **データ ロード エディター**で新しいスクリプト セクションを追加します。
3. このセクションを *People* と呼びます。
4. 次のスクリプトを入力します。

```
//Add dummy people data PeopleTemp: LOAD * INLINE [ PersonID, Person 1, Jane 2, Joe 3,
Shawn 4, Sue 5, Frank 6, Mike 7, Gloria 8, Mary 9, Steven, 10, Bill ];   //Add dummy age
data AgeTemp: LOAD * INLINE [ PersonID, Age 1, 23 2, 45 3, 43 4, 30 5, 40 6, 32 7, 45 8,
54 9, 10, 61 11, 21 12, 39 ];   //LOAD new table with people People: NoConcatenate LOAD
   PersonID,     Person Resident PeopleTemp;   Drop Table PeopleTemp;   //Add age and
age bucket fields to the People table Left Join (People) LOAD     PersonID,     Age,    If
(IsNull(Age) or Age='', Dual('No age', 5),       If(Age<25, Dual('Under 25', 1),          If
(Age>=25 and Age <35, Dual('25-34', 2),            If(Age>=35 and Age<50, Dual('35-49' , 3),
  If(Age>=50, Dual('50 or over', 4)          ))))) as AgeBucket Resident AgeTemp Where
Exists(PersonID);   DROP Table AgeTemp;
```

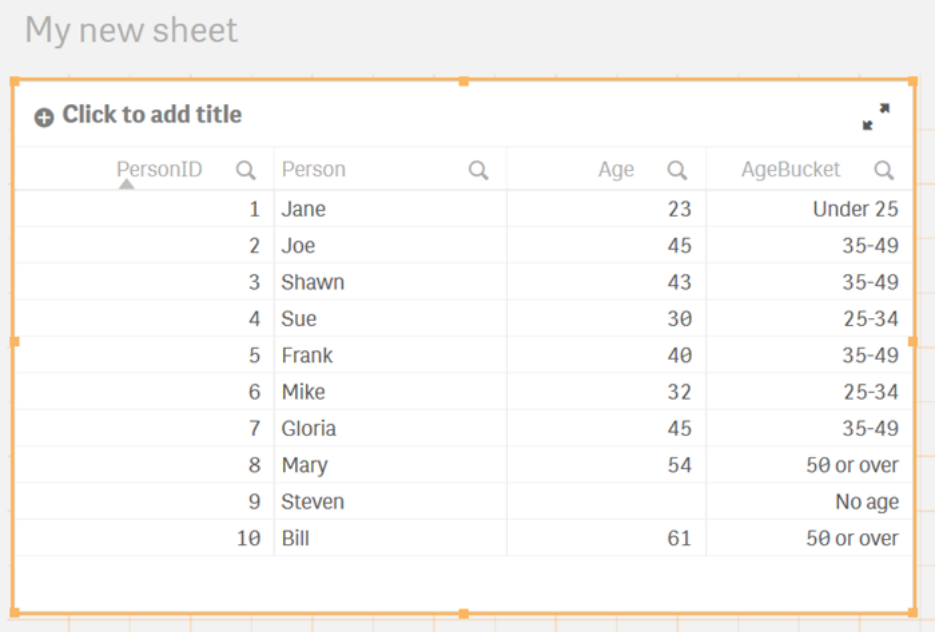
5. **[データのロード]** をクリックします。

   スクリプトでは、*Age* および *AgeBucket* 項目がロードされるのは、*PersonID* がすでにデータ モデルにロードされている場合に限られます。

   AgeTemp テーブルには年齢のリスト (*PersonID* 11 と 12) がありますが、これらの ID はデータ モデル (*People* テーブル) にはロードされておらず、Where Exists(PersonID) 節によって除外されていることに注意してください。この節はまた、のように書くこともできます。Where Exists(PersonID, PersonID) をクリックします。

   スクリプトの出力は次のようになります。

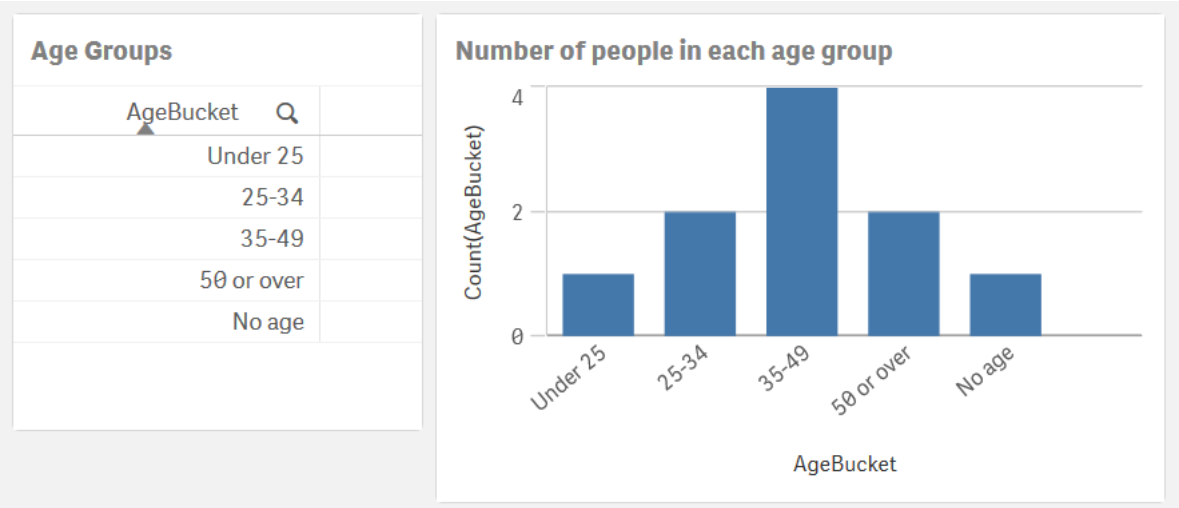
   スクリプトで Exists を使用した後のテーブル



PersonID	Person	Age	AgeBucket
1	Jane	23	Under 25
2	Joe	45	35-49
3	Shawn	43	35-49
4	Sue	30	25-34
5	Frank	40	35-49
6	Mike	32	25-34
7	Gloria	45	35-49
8	Mary	54	50 or over
9	Steven		No age
10	Bill	61	50 or over

PersonID が *AgeTemp* データモデルにロード済みのテーブルにない場合、*Age* と *AgeBucket* 項目は *People* テーブルには結合されません。Exists()関数を使用すると、データモデル内に孤立したレコード/データが残りません。つまり、*Age* と *AgeBucket* 項目に関連する従業員はいないことになります。

6. 新しいシートを作成し、名前を付けます。
7. その新しいシートを開き、**[シートの編集]** をクリックします。
8. 軸 *AgeBucket* を含むシートに標準的なテーブルを追加し、ビジュアライゼーションに「*Age Groups*」という名前を付けます。
9. 軸 *AgeBucket* とメジャー *Count([AgeBucket])* を持つシートに棒グラフを追加します。ビジュアライゼーションに *Number of people in each age group* という名前を付けます。
10. テーブルと棒グラフのプロパティを希望に応じて調整し、**[完了]** をクリックします。
シートは次のようになります。
年齢別にグループ化したシート



Dual() 関数は、文字列に数値を割り当てる必要があるスクリプトやチャートの数式で役に立ちます。

上記のスクリプトでは、年齢をロードしているアプリケーションがあり、この年齢をバケットに入れ、この年齢バケットと実年齢に基づくビジュアライゼーションを作成します。従業員 25 歳未満、25 ~ 35 歳といったバケットがあります。Dual() 関数を使用すると、後でリストボックスやチャート内で年齢バケットをソートするために便利な数値を、年齢バケットに割り当てることができます。つまりアプリシートでは、ソートによってリストの終端に「No age (年齢なし)」が追加されます。

Exists() と Dual() について詳しくは、Qlik Community の次のブログ投稿を参照してください。「Dual & Exists - Useful Functions」(Dual と Exists - 便利な関数)

2.4 間隔の一致と反復ロード

Intervalmatch プレフィックスを LOAD または SELECT ステートメントで使用すると、不連続の数値が 1 つ以上の数値間隔にリンクされます。これはとても有用な機能で、実稼働環境などで使用できます。

IntervalMatch() プレフィックスの使用

もっとも基本的な間隔一致は、1 つのテーブルに数値や日付 (イベント) のリストが存在し、2 つ目のテーブルに間隔のリストがある場合です。目的は、この 2 つのテーブルをリンクさせることです。一般的に、これは多対多の関係です。つまり、間隔には多数の日付が属し、日付は多数の間隔に属する場合があります。これを解決するには、2 つのオリジナル テーブル間の橋渡しとなるブリッジ テーブルを作成する必要があります。これには、さまざまな方法があります。

Qlik Sense におけるこの問題を解決するもっとも簡単は方法は、IntervalMatch() プレフィックスを LOAD または SELECT ステートメントの前に配置する方法です。LOAD/SELECT ステートメントには、間隔を定義する From と To という2 つの項目のみを含める必要があります。IntervalMatch() プレフィックスは、プレフィックスのパラメータで指定された、ロード済みの間隔と前にロードされた数値項目間のすべての組み合わせを生成します。

次の手順を実行します。

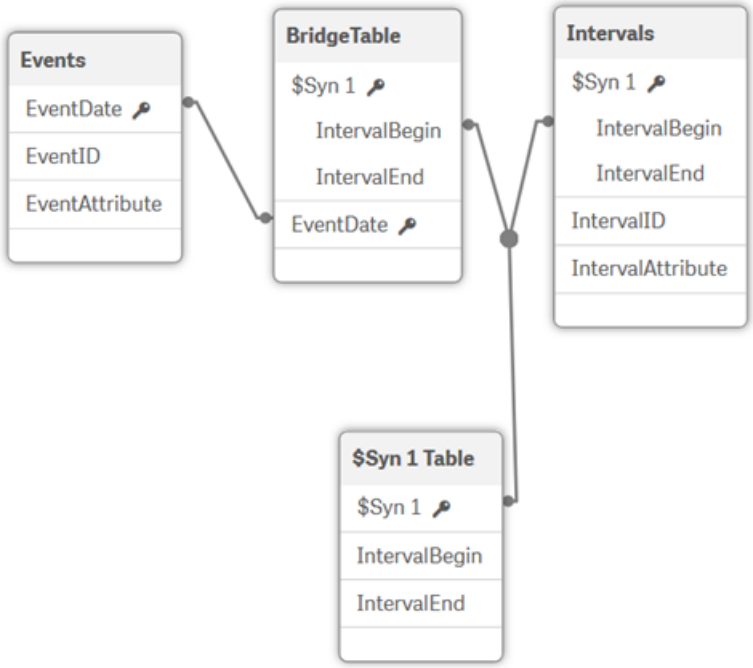
1. 新しいアプリを作成し、名前を付けます。
2. **データ ロード エディター**で新しいスクリプトセクションを追加します。
3. このセクションを *Events* と呼びます。
4. 右のメニューの **[AttachedFiles]** で、**[データを選択]** をクリックします。
5. *Events.txt* をアップロードして選択します。
6. **[Select data from]** (データの選択元) ウィンドウで、**[スクリプトを挿入]** をクリックします。
7. *Intervals.txt* をアップロードして選択します。
8. **[Select data from]** (データの選択元) ウィンドウで、**[スクリプトを挿入]** をクリックします。
9. スクリプトで、1 つ目のテーブルに *Events*、2 つ目のテーブルに *Intervals* という名前を付けます。
10. スクリプトの最後に IntervalMatch を追加して、最初の 2 つのテーブルをリンクさせる 3 つ目のテーブルを作成します。

    ```
    BridgeTable: IntervalMatch (EventDate) LOAD distinct IntervalBegin, IntervalEnd Resident Intervals;
    ```
11. これで、スクリプトは次のようになります。

```
Events: LOAD     EventID,     EventDate,     EventAttribute FROM
[lib://AttachedFiles/Events.txt]  (txt, utf8, embedded labels, delimiter is '\t', msq);
 Intervals: LOAD     IntervalID,     IntervalAttribute,     IntervalBegin,
IntervalEnd FROM [lib://AttachedFiles/Intervals.txt]  (txt, utf8, embedded labels,
delimiter is '\t', msq);   BridgeTable: IntervalMatch (EventDate) LOAD distinct
IntervalBegin, IntervalEnd Resident Intervals;
```

12. **[データのロード]** をクリックします。
13. **[データ モデル ビューア]** を開きます。データ モデルは次のようになります。

    データ モデル: Events、BridgeTable、Intervals、および $Syn1 テーブル



データ モデルには複合キー (*IntervalBegin* と *IntervalEnd* 項目) が含まれ、これは Qlik Sense 合成キーとして現れます。

基本的なテーブルは、

- イベントごとに正確に 1 つのレコードを含む、*Events* テーブル。
- 間隔ごとに正確に 1 つのレコードを含む、*Intervals* テーブル。
- イベントと間隔の組み合わせごとに正確に 1 つのレコードを含み、上記 2 つのテーブルをリンクさせているブリッジ テーブル。

間隔の重複によって、イベントに複数の間隔が付属している場合があります。そして 1 つの間隔に複数のイベントが付属している場合もあります。

このデータ モデルは、データの正規化やコンパクト化という意味においては最適です。*Events* テーブルと *Intervals* テーブルは、ともに不変で元の数と同じレコード数が含まれています。Count(EventID) など、これらのテーブルで行われる Qlik Sense のすべての計算は、正しく動作し、評価されます。

IntervalMatch() について詳しくは、Qlik Community の次のブログ投稿を参照してください。
IntervalMatch() の使用

While ループの使用とIterNo() の反復ロード

間隔の下限と上限間の可算値を生成するWhile ループとIterNo() を使用すると、ブリッジ テーブルと同様の結果が得られます。

LOAD ステートメント内のループは、次のようにWhile 節を使って作成できます。例:

```
LOAD Date, IterNo() as Iteration From … While IterNo() <= 4;
```

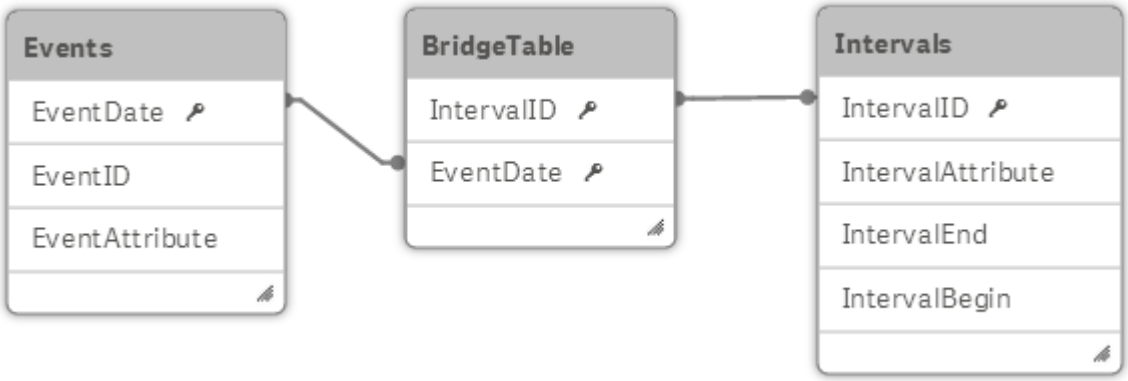
このようなLOAD ステートメントは、各入力レコードを何度もループし、While 節の数式がtrue である限りロードは何度も続きます。IterNo() 関数は、最初の反復では「1」、2 回目の反復では「2」というように値を返します。

間隔にはIntervalID というプライマリキーがあるため、スクリプトにおける唯一の違いはブリッジ テーブルを作成する方法です。

次の手順を実行します。

1. 既存のBridgetable ステートメントを、次のスクリプトと置き換えます。
   ```
   BridgeTable: LOAD distinct * Where Exists(EventDate); LOAD IntervalBegin + IterNo() - 1
   as EventDate, IntervalID     Resident Intervals     While IntervalBegin + IterNo() - 1
   <= IntervalEnd;
   ```
2. **[データのロード]** をクリックします。
3. **[データ モデル ビューア]** を開きます。データ モデルは次のようになります。

   データ モデル: Events、BridgeTable、および Intervals テーブル



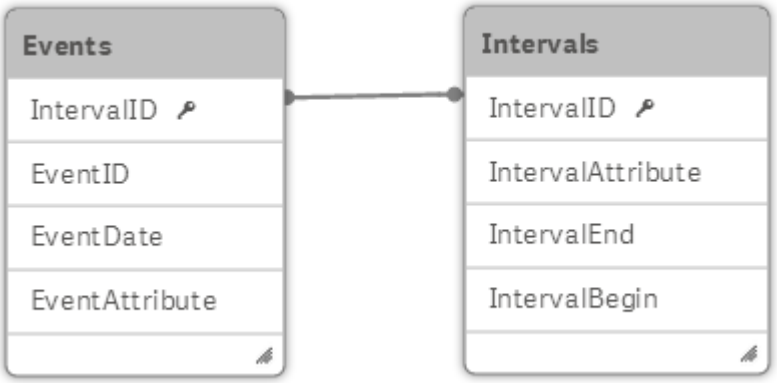
   一般に、3 つのテーブルを有するソリューションが最善です。これは間隔とイベント間に多対多の関係が許容されるからです。ですが多くの場合、1 つのイベントには単一の間隔が付属します。この場合、ブリッジ テーブルは必要ではありません。*IntervalID* は、イベント テーブルに直接保存されます。これを実現するにはいくつかの方法がありますが、もっとも有用な方法はBridgetable と*Events* テーブルを結合することです。

4. 次のスクリプトを、スクリプトの最後に追加します。

```
Join (Events) LOAD EventDate, IntervalID Resident BridgeTable;   Drop Table BridgeTable;
```

5. **[データのロード]** をクリックします。
6. **[データ モデル ビューア]** を開きます。データ モデルは次のようになります。

データ モデル: Events および Intervals テーブル



開区間と閉区間

間隔が開いているまたは閉じているかどうかは、エンドポイントが間隔に含まれているかどうかで判断します。

- エンドポイントが含まれている場合、それは閉区間です。
  $[a,b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$
- エンドポイントが含まれていない場合、それは開区間です。
  $]a,b[= \{x \in \mathbb{R} \mid a < x < b\}$
- 1 つのエンドポイントが含まれている場合、それは半開区間です。
  $[a,b[= \{x \in \mathbb{R} \mid a \leq x < b\}$

間隔が重複している場合や数値が 2 つ以上の間隔に付属している場合、通常は閉区間を使用する必要があります。

しかし、間隔を重複させずに、1 つの間隔にのみ数値を付属させたい場合もあります。よって、1 つのポイントが 1 つの間隔の終端にあり、同時に次の間隔の冒頭にある場合は問題になります。この値を持つ数値は、両方の間隔に属することになります。つまり、このような場合は半開区間を使用します。

この問題の現実的な解決策は、すべての間隔の終点地から少量を差し引き、閉じているが重複しない間隔を作成することです。数値が日付だった場合、日付の最後のミリ秒を返す DayEnd() 関数を使用するのが、もっとも簡単な方法です。

```
Intervals: LOAD…, DayEnd(IntervalEnd – 1) as IntervalEnd From Intervals;
```

手動で少量を差し引くこともできます。これを実施すると、値が 2 進 52 桁の有効数字 (10 進 14 桁) に四捨五入されるため、差し引いた量が小さすぎないことを確認してください。小さすぎる値を使用すると、差異に意味がなくなり、元の数値が使用されます。

3 データ クレンジング

Qlik Sense にロードするソース データが、必ずしも Qlik Sense アプリで必要でない場合があります。Qlik Sense は、データを希望する形式に変換するための関数とステートメントのホストを提供しています。

Qlik Sense スクリプトを実行する際にマッピングを使用すると、項目値や名前の置換または変更ができます。つまり、マッピングによりデータをより整合性のあるデータに整理したり、項目値の一部または全部と置換することが可能になります。

データを異なるテーブルからロードする際、同じことを表す項目値に、一貫した名前が付いているとは限りません。この一貫性の欠如は関連性を見落とす原因にもなるため、対処が必要です。この問題は、項目値を比較するマッピング テーブルを作成することで解決できます。

3.1 マッピング テーブル

Mapping load または Mapping select からロードされたテーブルは、他のテーブルとは異なる方法で処理されます。これらのテーブルは メモリの別の領域に保存され、スクリプトの実行時にマッピング テーブルとしてのみ使用されます。スクリプトが実行されると、これらのテーブルは自動的に削除されます。

ルール:

- マッピング テーブルは、1 列目に比較値、2 列目にマッピング値を含む 2 列構成でなければなりません。
- 2 つの列に名前を付ける必要がありますが、名前と列に含まれる値に関連性はありません。これらの列の名前は、通常の内部テーブルの項目名とは一切関係がありません。

3.2 Mapping 関数とステートメント

このチュートリアルでは、次のマッピング関数/ステートメントについて説明します。

•Mapping プレフィックス

• ApplyMap()

• MapSubstring()

•Map … Using ステートメント

•Unmap ステートメント

3.3 マッピング プレフィックス

Mapping プレフィックスは、マッピング テーブルを作成するためにスクリプトで使用します。マッピング テーブルは、ApplyMap() 関数または MapSubstring() 関数、Map … Using ステートメントと共に使用できます。

次の手順を実行します。

1. 新しいアプリを作成し、名前を付けます。
2. **データ ロード エディター**で新しいスクリプト セクションを追加します。

3. このセクションを *Countries* と呼びます。
4. 次のスクリプトを入力します。

```
CountryMap: MAPPING LOAD * INLINE [ Country, NewCountry U.S.A., US U.S., US United
States, US United States of America, US ];
```

CountryMap テーブルには、2 つの列(*Country* および *NewCountry*) が保存されます。*Country* 列には、様々な表記方法で *Country* 項目に入力された国が保存されます。*NewCountry* 列には、値をマッピングする方法に関する情報が保存されます。このマッピング テーブルは、一貫性のある*US* という国の値を *Country* 項目に保存するために使用されます。つまり、*U.S.A.* と *Country* 項目に保存されても、それは *US* としてマッピングされます。

3.4 ApplyMap() 関数

ApplyMap() を使用して、前に作成したマッピング テーブルに基づいて項目のデータを置き換えます。マッピング テーブルは、ApplyMap() 関数を使用する前にロードする必要があります。ロードする *Data.xlsx* テーブルのデータは次のようになります。

データ テーブル

ID	名前	国	コード
1	John Black	USA	SDFGBS1DI
2	Steve Johnson	U.S.	2ABC
3	Mary White	米国	DJY3DFE34
4	Susan McDaniels	u	DEF5556
5	Dean Smith	US	KSD111DKFJ1

国が様々な表記方法で入力されているのが分かります。国項目に一貫性を持たせるために、マッピング テーブルをロードし、**ApplyMap()** 関数を使用します。

次の手順を実行します。

1. 上で作成したスクリプトの下で、*Data.xlsx* を選択してロードし、スクリプトを挿入します。
2. 新しく作成した LOAD ステートメントの上に以下を入力します。

```
Data:
```

これで、スクリプトは次のようになります。

```
CountryMap: MAPPING LOAD * INLINE [     Country, NewCountry     U.S.A., US     U.S., US
   United States, US     United States of America, US ];         Data: LOAD     ID,     Name,
   Country,     Code FROM [lib://AttachedFiles/Data.xlsx] (ooxml, embedded labels, table
is Sheet1);
```

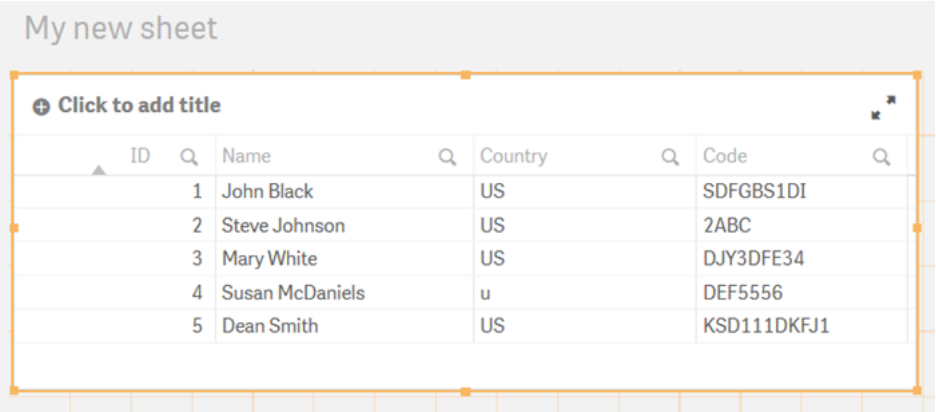
3. `Country,` を含む行を次のように修正します。

```
ApplyMap('CountryMap', Country) as Country,
```

ApplyMap() 関数の最初のパラメータには、単一引用符で囲まれたマップ名があります。2 番目のパラメータは、置換されるデータを含む項目です。

4. **[データのロード]** をクリックします。
   この結果、テーブルは次のようになります。

   ApplyMap() 関数を使用してロードしたデータのテーブル



   様々なつづりで表記された *United States* が、すべて *US* に変換されます。1 つのレコードはつづりが間違っているため、ApplyMap() 関数はこれを項目値に変更していません。ApplyMap() 関数を使用する際に、3 番目のパラメータを使ってマッピング テーブルに一致する値がない場合のデフォルトの数式を追加することができます。

5. 'US' を ApplyMap() 関数の 3 番目のパラメータとして追加し、国名が正しく入力されなかった場合に対応します。

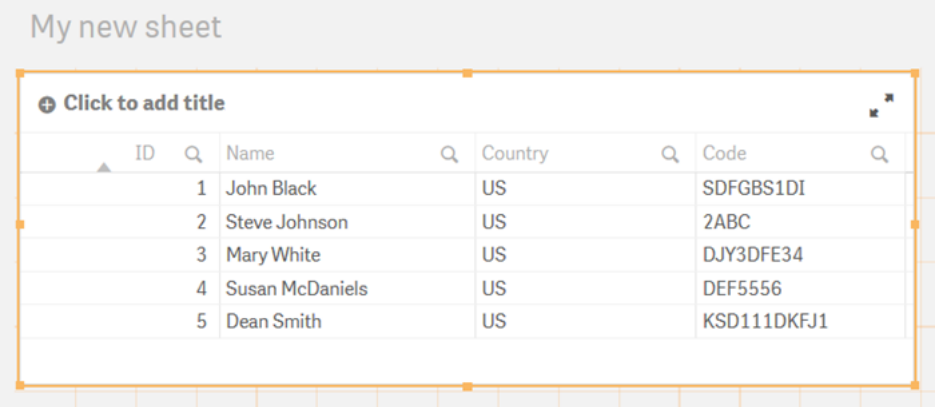
```
ApplyMap('CountryMap', Country, 'US') as Country,
```

   これで、スクリプトは次のようになります。

```
CountryMap: MAPPING LOAD * INLINE [     Country, NewCountry     U.S.A., US     U.S., US
   United States, US     United States of America, US ];  Data: LOAD     ID,     Name,
  ApplyMap('CountryMap', Country, 'US') as Country,     Code FROM
[lib://AttachedFiles/Data.xlsx] (ooxml, embedded labels, table is Sheet1);
```

6. **[データのロード]** をクリックします。
   この結果、テーブルは次のようになります。

   ApplyMap 関数を使用してロードしたデータのテーブル



ApplyMap() について詳しくは、*Qlik Community* の次のブログ投稿を参照してください。「Don't join - use Applymap instead」*(結合しない - Applymap* で代用*)*

3.5 MapSubstring() 関数

MapSubstring() 関数は、項目の部分的なマッピングを可能にします。

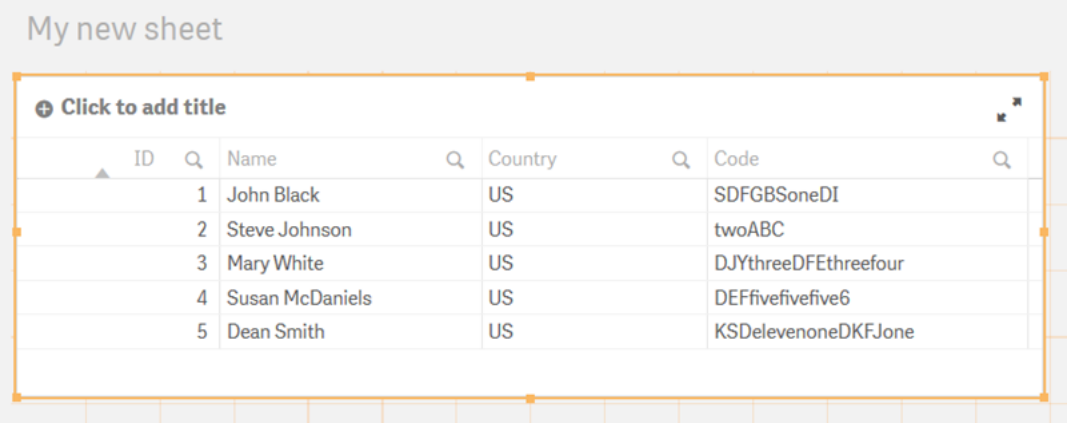
ApplyMap() で作成したテーブルに数値を文字列として記入したい場合、MapSubstring() 関数を使って数値を文字列に置換できます。

これを実行するには、まずマッピング テーブルを作成する必要があります。

次の手順を実行します。

1. *CountryMap* セクションの末尾、かつ *Data*セクションの前に次のスクリプト行を追加します。
   ```
   CodeMap: MAPPING LOAD * INLINE [ F1, F2 1, one 2, two 3, three 4, four 5, five 11, eleven ];
   ```
   *CodeMap*テーブルでは、数値の 1 ~ 5 と 11 がマッピングされます。
2. スクリプトの *Data* セクションで、Code ステートメントを次のように修正します。
   ```
   MapSubString('CodeMap', Code) as Code
   ```
   これで、スクリプトは次のようになります。
   ```
   CountryMap: MAPPING LOAD * INLINE [     Country, NewCountry     U.S.A., US     U.S., US     United States, US     United States of America, US ];  CodeMap: MAPPING LOAD * INLINE [ F1, F2 1, one 2, two 3, three 4, four 5, five 11, eleven ];  Data: LOAD     ID,     Name,     ApplyMap('CountryMap', Country, 'US') as Country,     MapSubString('CodeMap', Code) as Code FROM [lib://AttachedFiles/Data.xlsx] (ooxml, embedded labels, table is Sheet1);
   ```
3. **[データのロード]** をクリックします。
   この結果、テーブルは次のようになります。

   MapSubString 関数を使用してロードしたデータのテーブル



数字が *Code* 項目で文字列に置換されています。ID=3 および ID=4 のように数字が 1 回以上現れる場合は、テキストもまた繰り返されます。ID=4.*Susan McDaniels* は、そのコードに 6 がありました。6 は *CodeMap* テーブルにマッピングされていないので、変更されません。ID=5、*Dean Smith* のコードには 111 がありました。これは、'elevenone' としてマッピングされています。

MapSubstring() について詳しくは、Qlik Community の次のブログ投稿を参照してください。
「Mapping ... and not the geographical kind」(地理学ではないマッピング)

3.6 Map ... Using

Map ... Using ステートメントもまた、項目へのマッピングに使用できます。ただし、ApplyMap() とは機能が少し異なります。ApplyMap() は項目名が発生するたびにマッピングを実行しますが、Map ... Using は、値が内部テーブルの項目名に保存された場合にマッピングを行います。

例を挙げて見てみましょう。スクリプトで *Country* 項目を複数回ロードしており、項目がロードされる度にマッピングを実行したい場合を想定します。このチュートリアルですでに説明したように ApplyMap() 関数が使用可能であり、Map ... Using を使用することもできます。

Map ... Using が使用されると、項目が内部テーブルに保存される際にマップが項目に適用されます。つまり、以下の例のように Data1 テーブルの *Country* 項目に対するマッピングが実行されますが、*Data2* テーブルの *Country2* 項目に対するマッピングは実行されません。これは、Map ... Using ステートメントが *Country* という名前の項目にのみ適用されることが理由です。*Country2* 項目が内部テーブルに保存されると、*Country* という名前ではなくなります。*Country2* テーブルに対するマッピングを実行したい場合は、ApplyMap() 関数を使用する必要があります。

Unmap ステートメントは Map ... Using ステートメントを終了させます。つまり *Country* が Unmap ステートメントの後にロードされると、*CountryMap* は適用されません。

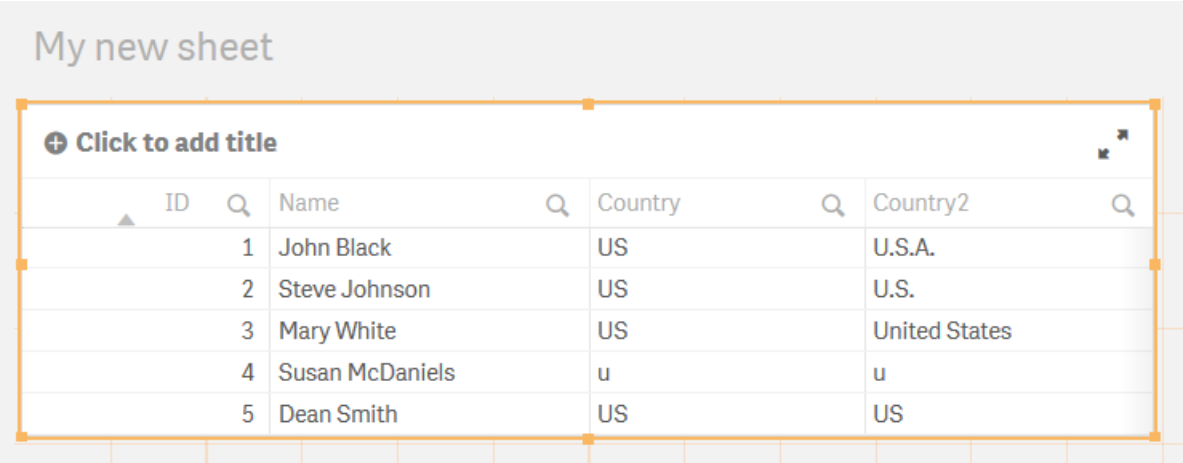
次の手順を実行します。

1. *Data* テーブルのスクリプトを次のスクリプトで置き換えます。

```
Map Country Using CountryMap; Data1:     LOAD     ID,     Name,     Country FROM
[lib://AttachedFiles/Data.xlsx] (ooxml, embedded labels, table is Sheet1);
LOAD     ID,     Country as Country2 FROM [lib://AttachedFiles/Data.xlsx] (ooxml,
embedded labels, table is Sheet1); UNMAP;
```

2. **[データのロード]** をクリックします。
   この結果、テーブルは次のようになります。

   Map ... Using 関数を使用してロードしたデータのテーブル



4 階層データの処理

階層構造は、すべてのビジネス インテリジェンス ソリューションの重要な部分で、異なる粒度レベルを含む軸を説明するために使用します。シンプルで直感的なものもあれば、正しくモデルを構築するためによく考える必要がある複雑なものもあります。

メンバーは階層の最上部から最下部に向かって、段階的に詳細になっていきます。例えば、軸に市場、国、州、市というレベルがある場合、南北アメリカ大陸は階層の最上部に位置し、米国は 2 番目のレベル、カリフォルニアは 3 番目、サンフランシスコは最下部のレベルに表示されます。カリフォルニアは米国よりも範囲が狭く、サンフランシスコはカリフォルニアよりさらに詳細度が増しています。

リレーショナル モデルにおける階層の保存は、複数のソリューションがある場合、よく問題になります。これには、いくつかのアプローチがあります。

- 横方向の階層
- 隣接リスト モデル
- 経路列挙方法
- 入れ子集合モデル
- 先祖リスト

このチュートリアルの目的においては、クエリーで直接使用できる形式の階層を提示するために、先祖リストを作成します。その他のアプローチの詳細については、Qlik Community を参照してください。

4.1 Hierarchy プレフィックス

Hierarchy プレフィックスはスクリプト コマンドで、隣接するノード テーブルをロードする LOAD または SELECT ステートメントの前に置きます。LOAD ステートメントには、少なくとも 3 つの項目 (ノードの一意のキーである ID、親ノードへの参照、名前) が必要です。

プレフィックスはロードされたテーブルを展開ノード テーブルに変換します。テーブルには多数の追加列 (階層の各レベルごとに 1 つ) が含まれています。

次の手順を実行します。

1. 新しいアプリを作成し、名前を付けます。
2. **データ ロード エディター**で新しいスクリプト セクションを追加します。
3. このセクションを *Wine* と呼びます。
4. 右のメニューの **[AttachedFiles]** で、**[データを選択]** をクリックします。
5. *Winedistricts.txt* をアップロードして選択します。
6. **[Select data from]** (データの選択元) ウィンドウで、*Lbound* と *RBound* 項目を選択解除してロードされないようにします。
7. **[スクリプトを挿入]** をクリックします。
8. LOAD ステートメントの上に次の内容を入力します。

```
Hierarchy (NodeID, ParentID, NodeName)
```

これで、スクリプトは次のようになります。

```
Hierarchy (NodeID, ParentID, NodeName) LOAD NodeID, ParentID, NodeName FROM [lib://AttachedFiles/Winedistricts.txt] (txt, utf8, embedded labels, delimiter is '\t', msq);
```

9. **[データのロード]** をクリックします。
10. 結果テーブルを確認するには、**データ モデル ビューア**の **[プレビュー]** セクションを使用してください。

    結果の展開 ノード テーブルには、ソース テーブルとまったく同じ数のレコード(ノードごとに 1 つ) があります。展開 ノード テーブルは、リレーショナル モデルの階層を解析する、次のような多数の要件を満たしているため非常に実用的です。

    - すべてのノード名は 1 つの同じ列に存在するため、検索に使用できます。
    - さらに、さまざまなノード レベルがそれぞれ 1 つの項目 (ドリルダウン グループで、またはピボット テーブルの軸として使用可能な項目) に展開されます。
    - さらに、異なるノード レベルがそれぞれ 1 つの項目に展開されるため、項目をドリルダウン グループで使用することができます。
    - ノード固有のパスを含むように作成して、すべての祖先を正しい順番でリストすることができます。
    - ノードの深度 (ルートからの距離) も含めるよう作成することができます。

    この結果、テーブルは次のようになります。

    Hierarchy プレフィックスを使用してロードしたサンプル データのテーブル

My new sheet

NodeID	ParentID	NodeName	NodeName1	NodeName2	NodeName3	NodeName4	NodeName5	NodeName6
289	288	Bas-Médoc	The World	Europe	France	Bordeaux	Médoc	Bas-Médoc
290	289	Listrac	The World	Europe	France	Bordeaux	Médoc	Bas-Médoc
291	289	Pauillac	The World	Europe	France	Bordeaux	Médoc	Bas-Médoc
292	289	Saint-Estèphe	The World	Europe	France	Bordeaux	Médoc	Bas-Médoc
293	289	Saint-Julien	The World	Europe	France	Bordeaux	Médoc	Bas-Médoc
294	288	Haut-Médoc	The World	Europe	France	Bordeaux	Médoc	Haut-Médoc
295	294	Margaux	The World	Europe	France	Bordeaux	Médoc	Haut-Médoc

階層について詳しくは、Qlik Community の次のブログ投稿を参照してください。「階層」

4.2 HierarchyBelongsTo プレフィックス

Hierarchy プレフィックス同様、HierarchyBelongsTo プレフィックスはスクリプト コマンドで、隣接するノード テーブルをロードする LOAD または SELECT ステートメントの前に置きます。

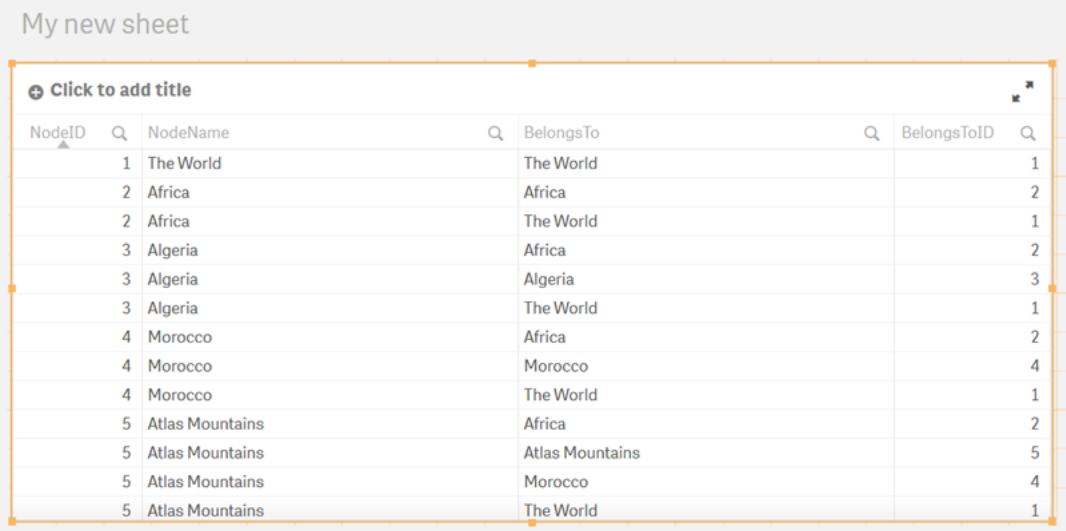
ここでもまた、LOAD ステートメントには少なくとも 3 つの項目 (ノードの一意のキーである ID、親 ノードへの参照、名前) が必要です。プレフィックスはロードされたテーブルを展開 ノード テーブルに変換します。テーブルにはあらゆる先祖の組み合わせおよび、個別レコードとして子孫がリストされています。よって、特定 ノードの先祖または子孫はすべて簡単に検索できます。

次の手順を実行します。

1. **データ ロード エディター**で、Hierarchy ステートメントを次のように修正します。
   ```
   HierarchyBelongsTo (NodeID, ParentID, NodeName, BelongsToID, BelongsTo)
   ```
2. **[データのロード]** をクリックします。
3. 結果テーブルを確認するには、**データ モデル ビューア**の **[プレビュー]** セクションを使用してください。
   先祖テーブルは、リレーショナル モデルの階層を解析する、次のような多数の要件を満たしています。
   - ノードIDが単一ノードを表す場合、先祖 ID 階層のツリー全体およびサブツリーを表します。
   - すべてのノード名は、ノードとしての役割およびツリーとして役割の両方に存在し、両方とも検索に使用できます。
   - ノード深度とサブツリーのルートからの距離である先祖深度の間の深度差異を含むよう、作成することもできます。

   この結果、テーブルは次のようになります。

   HierarchyBelongsTo プレフィックスを使用してロードしたデータのテーブル



NodeID	NodeName	BelongsTo	BelongsToID
1	The World	The World	1
2	Africa	Africa	2
2	Africa	The World	1
3	Algeria	Africa	2
3	Algeria	Algeria	3
3	Algeria	The World	1
4	Morocco	Africa	2
4	Morocco	Morocco	4
4	Morocco	The World	1
5	Atlas Mountains	Africa	2
5	Atlas Mountains	Atlas Mountains	5
5	Atlas Mountains	Morocco	4
5	Atlas Mountains	The World	1

承認

階層を承認に使用することは、珍しいことではありません。1 つの例としては、組織的な階層が挙げられます。各マネージャーが、すべてのサブ部門含め、自分の部門に関連するあらゆる権限を持っている必要があります。ですが、必ずしも他の部門を管理する権限を持つべきではない、というわけではありません。

組織的な階層の例

これは、閲覧を許可される組織内のサブツリーは人によって異なるということを意味します。認証テーブルは、例えば、以下のようになります。

承認テーブル

ACCESS	NTNAME	PERSON	POSITION	PERMISSIONS
ユーザー	ACME\JRL	John	CPO	HR
ユーザー	ACME\CAH	Carol	CEO	CEO
ユーザー	ACME\JER	James	エンジニアリング部長	エンジニアリング
ユーザー	ACME\DBK	Diana	CFO	財務
ユーザー	ACME\RNL	Bob	COO	売上
ユーザー	ACME\LFD	Larry	CTO	製品

この場合、*Carol* は *CEO* とその下の関係するすべての組織を閲覧でき、*Larry* は *Product* 組織を、*James* は *Engineering* 組織のみを閲覧できます。

多くの場合、階層は隣接ノードテーブルに保存されます。この例では、これを解決するために、HierarchyBelongsTo を使用して隣接ノードテーブルをロードし、先祖項目 *Tree* に名前を付けます。

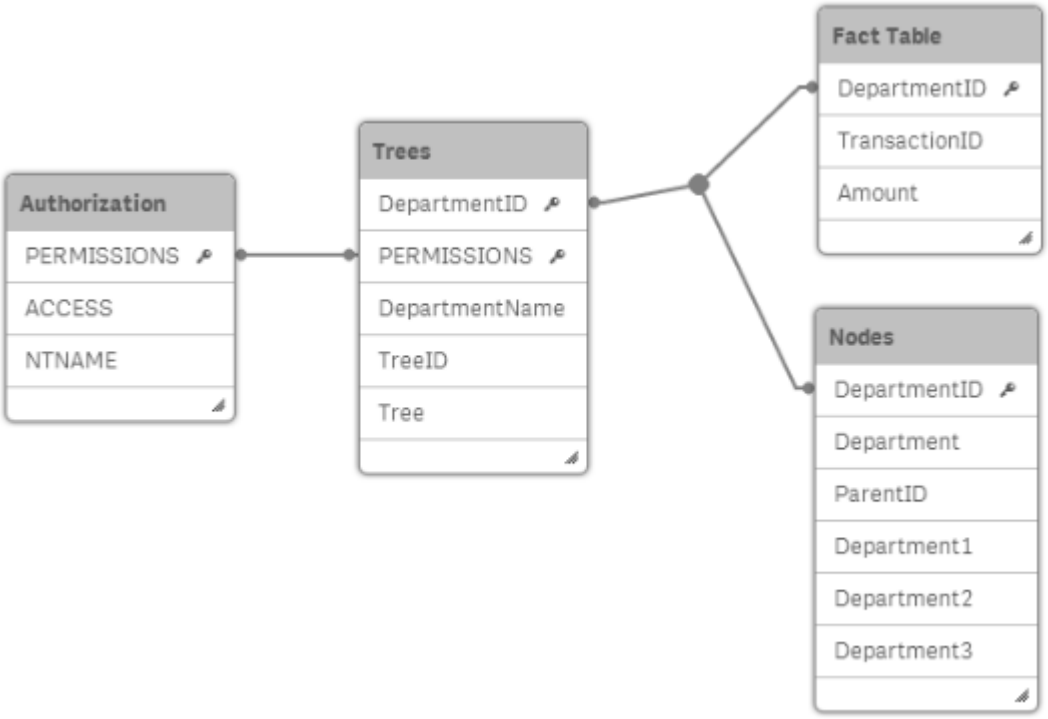
Section Access を使用する場合は、*Tree* の大文字コピーをロードし、この新しい項目を *PERMISSIONS* とします。最後に、承認テーブルをロードする必要があります。最後の 2 つのステップは、次のスクリプト行を使用して実行することが可能です。TempTrees テーブルは HierarchyBelongsTo ステートメントによって作成されるテーブルです。

これはあくまで例です。Qlik Sense で行う付随する演習はありません。

```
Trees: LOAD *,     Upper(Tree) as PERMISSIONS     Resident TempTrees; Drop Table TempTrees;
Section Access; Authorization: LOAD     ACCESS,         NTNAME,         UPPER(Permissions) as PERMISSIONS Fro
Organization; Section Application;
```

この例では、次のデータモデルが生成されます。

データモデル:Authorization、Trees、Fact、および Nodes テーブル



5 QVD ファイル

QVD (QlikView Data) ファイルは、Qlik Sense または QlikView からエクスポートされたデータのテーブルを含むファイルです。QVD はネイティブの Qlik 形式で、Qlik Sense または QlikView でしか書き込みと読み出しを行えません。ファイル形式は、Qlik Sense スクリプトからデータを読み取る際の速度について最適化されていますが、それでも非常にコンパクトです。QVD ファイルからのデータの読み取りは、他のデータソースから読み取る場合よりも、一般に 10 ~ 100 倍速くなっています。

QVD ファイルは、標準 (高速) と最適化 (超高速) の 2 つのモードで読み取ることができます。使用されるモードは、Qlik Sense スクリプトエンジンによって自動的に決定されます。最適化モードは、ロードされたすべての項目が変換 (項目に対して実行される式) なしに読み取られる場合にのみ使用できます。ただし、項目名は変更することができます。Where 節は、Qlik Sense にレコードを解凍させ、最適化されたロードを無効にします。

QVD ファイルは、厳密に 1 つのデータテーブルを保持し、次の 3 つの部分で構成されます。

- テーブル内の項目や、後続情報およびその他のメタデータのレイアウトを記述する XML ヘッダー (UTF-8 文字セット)。
- バイト埋め込み形式のシンボル テーブル。
- ビット埋め込み形式の実際のテーブル データ。

QVD ファイルの用途はさまざまですが、主に 4 つの目的で使用されます。複数の目的を一度に達成できる場合もあります。

- データロード速度の向上

  QVD ファイルの入力データの変更されない部分または変化が遅い部分をバッファリングすることで、大きなデータセットに対するスクリプトの実行が大幅に高速化します。

- データベース サーバーの負荷の減少

  外部データソースから取得するデータ量を大幅に削減できます。これにより、外部データベースおよびネットワーク トラフィックの負荷が減少します。さらに、複数の Qlik Sense スクリプトが同じデータを共有する場合は、ソース データベースから QVD ファイルに一度データをロードするだけで済みます。他のアプリケーションは、この QVD ファイルから同じデータを使用できます。

- 複数の Qlik Sense アプリケーションからのデータ統合

  Binary スクリプト ステートメントでは、単一の Qlik Sense アプリケーションからのみ別のアプリケーションにデータをロードできます。しかし、QVD ファイルを使用すれば、Qlik Sense スクリプトは、任意の数の Qlik Sense アプリケーションからデータを統合できます。これにより、さまざまな部署の類似データを統合するなど、アプリケーションの可能性が広がります。

- 増分ロード

  多くの場合、QVD の機能を活用することで、増大するデータベースから新しいレコードだけをロードする増分ロードを簡単に実行できます。

5.1 QVD ファイルの作成

QVD ファイルは 2 つの方法で作成できます。

- Qlik Sense スクリプトで Store コマンドを使用した明示的な作成と命名。
  スクリプトで、以前に読み取ったテーブルまたはその一部を、選択した場所にある明示的に指定されたファイルにエクスポートするように記述します。
- スクリプトからの自動作成とメンテナンス。
  load または select ステートメントの前に Buffer プレフィックスを付けると、Qlik Sense は自動的に QVD ファイルを作成します。この QVD ファイルは一定の条件下で、データのリロード時に元のデータソースの代わりに使用できます。

読み込み時間など、生成された QVD ファイルに違いはありません。

Store

このスクリプト関数は、明示的に命名された QVD、CSV、txt ファイルを作成します。

構文:

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

項目のエクスポートは 1 つのデータテーブルからのみ行えます。複数のテーブルの項目がエクスポートされる場合、エクスポートされるデータテーブルを作成するためには、事前にスクリプトで明示的に結合されている必要があります。

テキストの値は UTF-8 形式で CSV ファイルに出力されます。区切り文字を指定できます (**LOAD** を参照)。CSV ファイルへの store ステートメントは、BIFF エクスポートをサポートしていません。

例:

```
Store mytable into [lib://AttachedFiles/xyz.qvd]; Store * from mytable into
[lib://FolderConnection/xyz.qvd]; Store myfield from mytable into
'lib://FolderConnection/xyz.qvd'; Store myfield as renamedfield, myfield2 as renamedfield2
from mytable into [lib://AttachedFiles/xyz.qvd]; Store mytable into
'lib://FolderConnection/myfile.txt'; Store * from mytable into
'lib://FolderConnection/myfile.csv';
```

次の手順を実行します。

1. *高度なスクリプト チュートリアル* アプリを開きます。
2. *Product* スクリプト セクションをクリックします。
3. 次の内容をスクリプトの最後に追加します。

   ```
   Store * from Product into [lib://AttachedFiles/ProductData.qvd](qvd);
   ```

   これで、スクリプトは次のようになります。

   ```
   CrossTable(Month, Sales) LOAD     Product,     "Jan 2014",     "Feb 2014",     "Mar
   2014",     "Apr 2014",     "May 2014" FROM [lib://AttachedFiles/Product.xlsx]  (ooxml,
   embedded labels, table is Product);  Store * from Product into
   [lib://AttachedFiles/ProductData.qvd](qvd);
   ```

4. **[データのロード]** をクリックします。
   これで *Product.qvd* ファイルがファイル リストに表示されるようになります。

Crosstableスクリプトの結果であるこのデータファイルは、各カテゴリ(Product, Month, Sales) ごとに 1 つの列を持つ3 列構成のテーブルです。このデータファイルは、*Product* スクリプトセクション全体を置換するために使用できます。

5.2 QVD ファイルからのデータのロード

Qlik Sense は次の方法で QVD ファイルの読み込みやアクセスを実行します。

- 明示的なデータソースとしての QVD ファイルのロード。QVD ファイルは、その他のテキストファイル (csv、fix、dif、biff など) と同様に、Qlik Sense スクリプトの load ステートメントで参照できます。

  例:

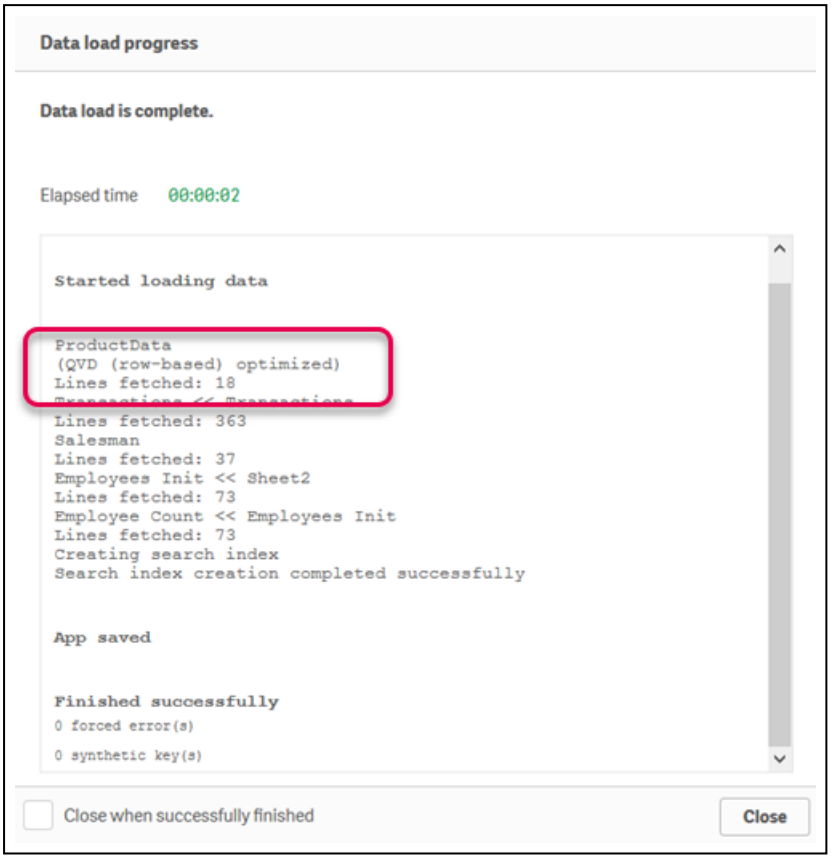
  ```
  LOAD * from 'lib://FolderConnection/xyz.qvd' (qvd); LOAD fieldname1, fieldname2 from
  [lib://FolderConnection/xyz.qvd] (qvd); LOAD fieldname1 as newfieldname1, fieldname2 as
  newfieldname2 from [lib://AttachedFiles/xyz.qvd](qvd);
  ```

- バッファ済みの QVD ファイルの自動ロード。load または select ステートメントで buffer プレフィックスを使用する場合、読み取り用の明示的なステートメントは必要ありません。Qlik Sense は元の LOAD ステートメントまたは SELECT ステートメントを使用してデータを取得するのではなく、QVD ファイルからどの範囲のデータを使用するかを決定します。
- スクリプトからの QVD ファイルへのアクセス。(QVD で始まる) いくつかのスクリプト関数を使用して、QVD ファイルの XML ヘッダーにあるデータのさまざまな情報を取得できます。

次の手順を実行します。

1. *Product* スクリプトセクションのスクリプト全体をコメントアウトします。
2. 次のスクリプトを入力します。
   ```
   Load * from [lib://AttachedFiles/ProductData.qvd](qvd);
   ```
3. **[データのロード]** をクリックします。
   データは QVD ファイルからロードされます。

データロード進捗状況 ウィンドウ



増分ロードでの QVD ファイルの使用について詳しくは、Qlik Community の次のブログ投稿を参照してください。「Overview of Qlik Incremental Loading」(Qlik 増分ロードの概要)

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。

```
Buffer load * from MyTable;
```

stale [after] amount [(days | hours)]

Amount は期間を指定する数字で、Decimals 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データベース ソースで使用します。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。

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

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、古いデータが変更されたり削除されている場合、増分 ロードは使用できませんのでご注意 ください。

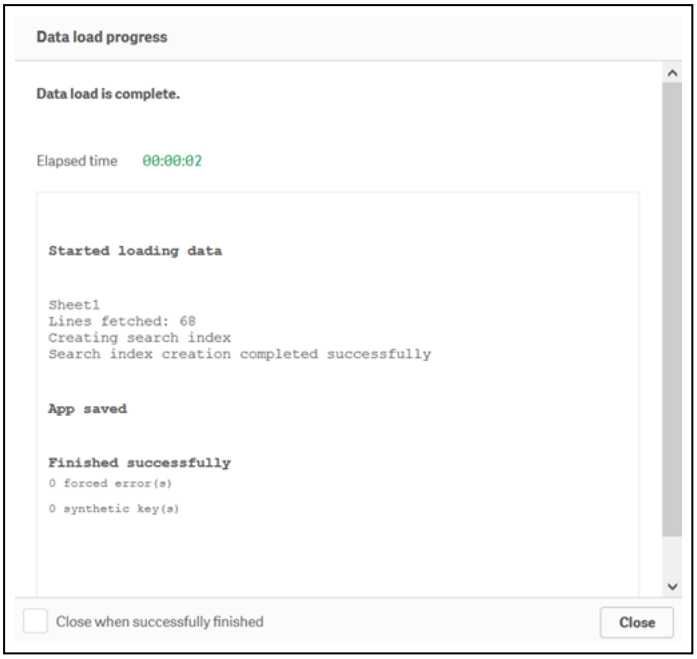
```
Buffer (incremental) load * from MyLog.log;
```

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。Store ステートメントは、バッファのコンテンツを QVD または CSV ファイルとして保持したい場合に使用します。

次の手順を実行します。

1. 新しいアプリを作成し、名前を付けます。
2. **データ ロード エディター**で新しいスクリプト セクションを追加します。
3. 右のメニューの **[AttachedFiles]** で、**[データを選択]** をクリックします。
4. *Cutlery.xlsx* をアップロードして選択します。
5. **[Select data from]** (データの選択元) ウィンドウで、**[スクリプトを挿入]** をクリックします。
6. Load ステートメントの項目をコメント アウトして、Load ステートメントを次のように変更します。
   ```
   Buffer LOAD *
   ```
   これで、スクリプトは次のようになります。
   ```
   Buffer LOAD * //     "date", //     item, //     quantity FROM
   [lib://AttachedFiles/Cutlery.xlsx] (ooxml, embedded labels, table is Sheet1);
   ```
7. **[データのロード]** をクリックします。
   初めてデータをロードするときは、*Cutlery.xlsx* からロードされます。

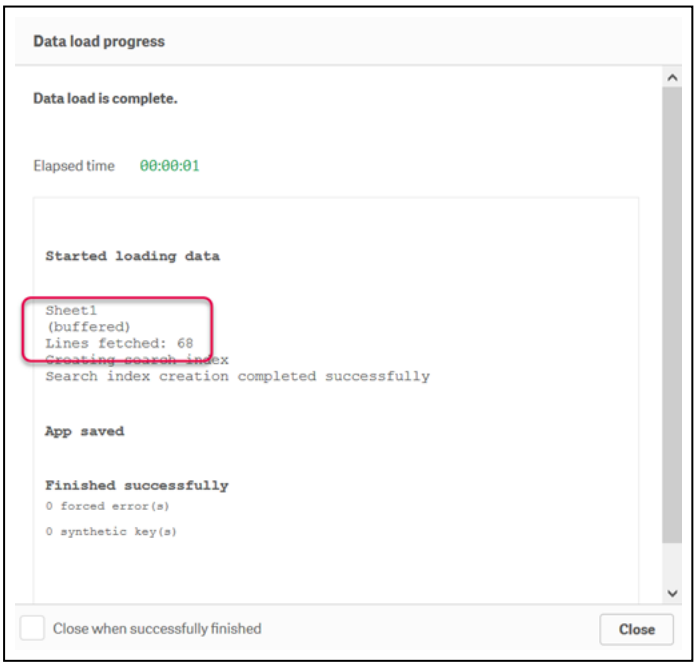
データロード進捗状況ウィンドウ



Buffer ステートメントは QVD ファイルを作成し、このファイルを Qlik Sense に保存します。Qlik Sense Enterprise on Windows 展開では、Qlik Sense サーバー上のディレクトリに保存されます。

8. **[データのロード]** を再度クリックします。
9. 今回は、初めてデータをロードしたときに Buffer ステートメントによって作成された QVD ファイルからデータがロードされます。

   データロード進捗状況ウィンドウ



5.3 お疲れ様でした!

以上でチュートリアルは終了です。ここでご紹介した基本情報をQlik Senseでのスクリプト作成にお役立てください。ウェブサイトでは、ご利用可能なトレーニングについての情報を公開しています。ぜひご覧ください。