

# 教程 - 脚本编写的后续步骤

Qlik Sense®

November 2022

版权所有 © 1993-2023 QlikTech International AB。保留所有权利。





---

<b>1 欢迎学习本教程！</b>	<b>5</b>
1.1 您将学到的内容	5
1.2 谁应当参加该课程	5
1.3 压缩包内容	5
1.4 本教程中的课程	6
1.5 延伸阅读和资源	6
<b>1 LOAD 和 SELECT 语句</b>	<b>7</b>
<b>2 转换数据</b>	<b>8</b>
2.1 使用 Crosstable 前缀	8
Crosstable 前缀	8
清除内存缓存	12
2.2 使用 Join 和 Keep 合并表格	12
Join	12
使用 Join	13
Keep	15
Inner	15
Left	17
Right	18
2.3 使用内部记录函数 Peek、Previous 和 Exists	19
Peek()	19
Previous()	19
Exists()	20
使用 Peek() 和 Previous()	20
使用 Exists()	22
2.4 匹配间隔和迭代加载	24
使用 IntervalMatch() 前缀	24
使用 While 循环和迭代加载 IterNo()	26
开放和封闭间隔	28
<b>3 数据清理</b>	<b>29</b>
3.1 映射表	29
规则：	29
3.2 Mapping 函数和语句	29
3.3 Mapping 前缀	29
3.4 ApplyMap() 函数	30
3.5 MapSubstring() 函数	32
3.6 Map ... Using	33
<b>4 处理层次结构数据</b>	<b>34</b>
4.1 Hierarchy 前缀	34
4.2 HierarchyBelongsTo 前缀	35
授权	36
<b>5 QVD 文件</b>	<b>39</b>
5.1 创建 QVD 文件	39
Store	40
5.2 从 QVD 文件读取数据	40
Buffer	42

---

---

5.3 谢谢！ .....	44
---------------	----

# 1 欢迎学习本教程！

欢迎学习本教程，我们将为您介绍 **Qlik Sense** 中的高级的脚本编译。

一旦您熟悉了脚本的基础知识，就可以在将数据加载到 **Qlik Sense** 中时开始对其执行更复杂的操作。例如，这可以包括使用交叉表转换数据、清理数据以及从称为 **QVD** 文件的 **Qlik** 数据文件中创建和加载数据。

## 1.1 您将学到的内容

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

## 1.2 谁应当参加该课程

您应当熟悉 **Qlik Sense** 中脚本编写的基础知识。也就是说，您已经使用脚本加载数据和操纵数据。

如果您还没有这样操作，我们建议您完成新手脚本编写教程。

您需要数据加载编辑器的访问权限并且应当得到在 **Qlik Sense Enterprise on Windows** 中加载数据的许可。

这些说明通常也适用于 **Qlik Sense Cloud Business**。

## 1.3 压缩包内容

您下载的压缩包中含有完成本教程所需的以下数据文件：

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

该软件包还包含高级脚本编写教程应用程序的副本。应用程序中的其他脚本部分包含您在本教程中创建的其他应用程序的脚本。您可将应用程序上传至应用中心。

我们建议按照教程中描述的方式自行构建应用程序，以最大限度地提高学习效果。此外，您还必须按照教程中的说明上传并连接到数据文件，数据加载才能正常工作。

但是，如果遇到问题，应用程序可能会帮助您排除故障。我们已经指出了与每节课相关联的脚本段。

## 1.4 本教程中的课程

根据您对于 **Qlik Sense** 的经验，完成本教程需要 **3-4** 小时。各主题设计为按顺序完成。但您也可以暂时离开，然后随时再回来继续学习。幸运的是，不会有测验。

转换数据

使用 **Crosstable** 前缀

使用 **Join** 和 **Keep** 合并表格

使用内部记录函数 **Peek**、**Previous** 以及 **Exists**

匹配间隔和迭代加载

数据清理

处理层次结构数据

QVD 文件

## 1.5 延伸阅读和资源

- [Qlik](#) 提供了各种各样的资源帮助您进行深入学习。
- [Qlik 在线帮助](#) 可供使用。
- 培训，包括免费的在线课程，可在 [Qlik Continuous Classroom](#) 获取。
- 讨论论坛、博客等可见于 [Qlik Community](#)。

# 1 LOAD 和 SELECT 语句

每一语句都将生成一个内部表格。可以使用 **LOAD** 和 **SELECT** 语句将数据加载到 **Qlik Sense**。**LOAD** 用于从文件加载数据, 而 **SELECT** 用于从数据库加载数据。

在本教程中, 您将使用来自文件的数据, 因此您将使用 **LOAD** 语句。

您还可以使用前导 **LOAD** 来操作所加载数据的内容。例如, 必须在 **LOAD** 语句中重命名字段, 而 **SELECT** 语句不允许更改字段名。

在将数据加载到 **Qlik Sense** 时应用以下规则:

- 在 **Qlik Sense** 中, **LOAD** 或 **SELECT** 语句生成的表格之间无任何区别。因此, 在加载多个表格时, 这些表格是由 **LOAD** 或 **SELECT** 语句加载, 抑或由这两者共同加载都无关紧要。
- 语句或数据库中原始表格的字段顺序是由 **Qlik Sense** 逻辑随机排列。
- 字段名区分大小写, 用于在数据表之间建立关联。因此, 有时必须重命名加载脚本中的字段以获得所需的数据模型。

## 2 转换数据

在应用程序中使用数据之前,可以在数据加载编辑器中转换和操作数据。

数据操作的其中一个优势是,您可以选择从文件中仅加载数据的子集(如某个表格中的几个选定列),以便更高效地处理数据。还可以多次加载数据以将原始数据分隔为多个新的逻辑表格。还可以从多个源加载数据并将其合并到 **Qlik Sense**。

下面的练习将向您展示如何使用 **Crosstable** 前缀加载数据。此外,您也将了解如何联接表格、使用内部记录函数(如 **Peek** 和 **Previous**)和使用 **While Load** 多次加载同一行。

### 2.1 使用 Crosstable 前缀

交叉表是常见的表格类型,特点是在两个标题数据正交列表之间显示值矩阵。只要您拥有数据的交叉表,均可使用 **Crosstable** 前缀转换数据并创建所需的字段。

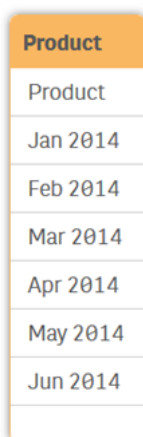
#### Crosstable 前缀

在下面的 *Product* 表中,每个月份必须使用一列,且每种产品必须使用一行。

Product 表						
Product	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

加载表时,输出是一个表,其中 *Product* 一个字段,月份中的每个一个字段。

具有 *Product* 字段的 *Product* 表,并且月份中的每个一个字段



Product
Product
Jan 2014
Feb 2014
Mar 2014
Apr 2014
May 2014
Jun 2014

如果您想要分析此数据,则可以轻松收集一个字段中的所有数字和另一个字段中的所有月份。在本例中,这是一个三列表,每个类别 (*Product*, *Month*, *Sales*) 有一列。



具有 *Product*、*Month* 和 *Sales* 字段的 *Product* 表格

Product
Product
Month
Sales

**Crosstable** 前缀可将数据转换成一个表格，其中一列为 *Month*，另一列为 *Sales*。另一种方法是使用字段名称并将其转换成字段值。

执行以下操作：

1. 新建应用程序并将其命名为高级脚本编写教程。
2. 在**数据加载编辑器**中新增脚本段。
3. 命名部分为 *Product*。
4. 在右侧菜单的 **AttachedFiles** 下，单击**选择数据**。
5. 上传然后选择 *Product.xlsx*。
6. 在**选择数据**自窗口中选择 *Product*。



确保选中了**字段名称**下面的**嵌入的字段名称**以包含您加载数据时表格字段的名称。

7. 单击**插入脚本**。

您的脚本应如下所示：

```
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. 单击**加载数据**。
9. 打开**数据模型查看器**。数据模型如下所示：

具有 *Product* 字段的 *Product* 表, 并且月份中的每个一个字段

Product
Product
Jan 2014
Feb 2014
Mar 2014
Apr 2014
May 2014
Jun 2014

10. 单击**数据加载编辑器**中的 *Product* 标签。
11. 在 LOAD 语句上方输入以下内容:  
`CrossTable(Month, Sales)`
12. 单击**加载数据**。
13. 打开**数据模型查看器**。数据模型如下所示:  
具有 *Product*、*Month* 和 *Sales* 字段的 *Product* 表格

Product
Product
Month
Sales

注意输入数据只有一列作为限定符字段;作为内部键(上例中的 *Product*)。但是, 您可以拥有多个此类字段。如果这样, 则必须在 **LOAD** 语句的属性字段前面列出了所有限定字段, 且 **Crosstable** 前缀的第三个参数必须用来定义限定字段的数量。您不能使用前置 **LOAD** 或将前缀放置在 **Crosstable** 关键字的前面。但是您可使用自动串联。

在 Qlik Sense 中的表格里, 您的数据形式如下:

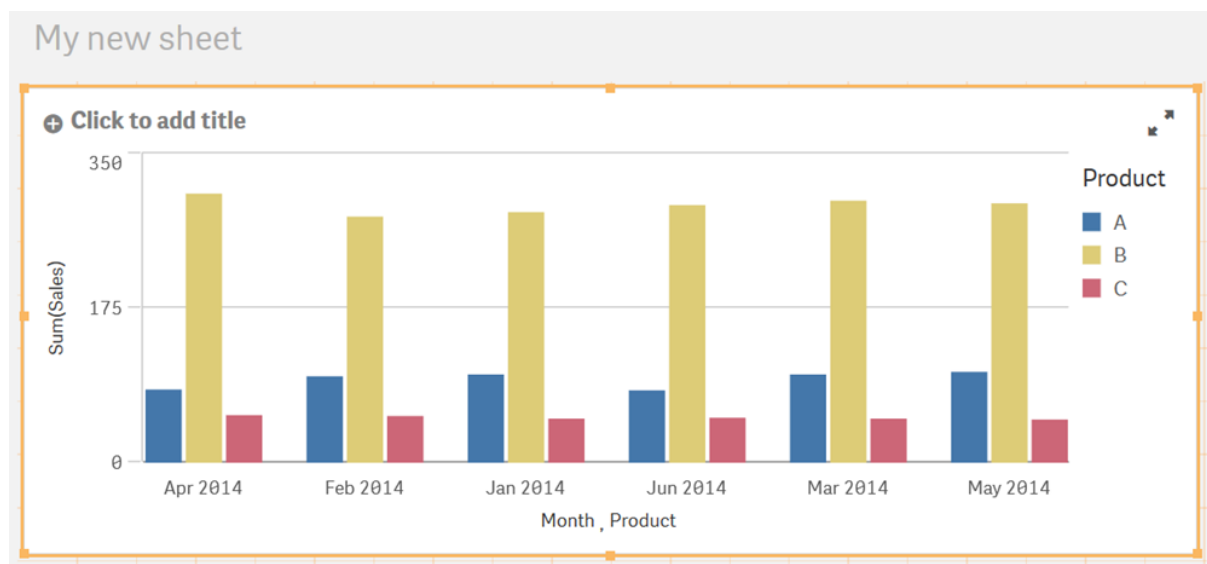
表格示出了使用 *Crosstable* 前缀加载的数据

My new sheet

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
C	Jan 2014	50

例如您现在可以使用数据 创建条形图：

条形图示出了使用 *Crosstable* 前缀加载的数据



要了解关于 *Crosstable* 的更多信息，请参阅 *Qlik Community* 中该文章：[Crosstable 加载](#)。这些行为将在 *QlikView* 的上下文中讨论。然而，逻辑同样适用于 *Qlik Sense*。

数字解释不适用于属性字段。这意味着如果您将月份用作列标题，则将不会自动对其进行解释。解决方法是使用 *Crosstable* 前缀创建一个临时表格，然后运行第二次通过它，使解释如下例所示：

注意这仅为示例。在 Qlik Sense 中没有附带的练习。

```
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;
```

## 清除内存缓存

可以删除为清除内存缓存而创建的表。当按照上一节所述加载到临时表格后,然后在不再需要时应将其删除。例如:

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

也可以放置字段。例如:

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

如您所见,关键字 TABLE 和 FIELD 可为单数或复数。

## 2.2 使用 Join 和 Keep 合并表格

将两个表格组合成一个表格的联接操作。组合后的表格记录是由原始表格的记录所组成,通常在将两个表格的记录合并到任何一个表格中时,一个常见值可用于一个或几个常见字段,这就是所谓的自然联接。在 Qlik Sense 中,联接可在脚本中创建,以生成逻辑表格。

联接已处于脚本中的表格是完全有可能的。Qlik Sense 逻辑随后无法看到单独的表格,但能看到联接结果,即单一的内部表格。某些情况下这是必要的,但也存在一些缺点:

- 加载表格往往会变得更大,这将使 Qlik Sense 运行得更慢。
- 一些信息可能会丢失:原始表格中的频率(记录数)可能不再可用。

Keep 功能可以在表格存入 Qlik Sense 之前将两个表格中的一个或两个缩减为表格数据的交集,旨在减少需要使用显式联接的情况。



在本文档中,术语“联接”通常指创建内部表格前所作的联接。但是,创建内部表格后所作的关联在本质上也是联接。

### Join

进行联接的最简单方法是在脚本中使用 Join 前缀,以联接内部表格与另一个命名表格或最后创建的表格。该联接是外部联接,创建两个表格的所有可能的数值组合。

示例:

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

结果内部表格包含字段 a、b、c 和 d。根据这两个表格的字段值不同,记录的数量也会有所不同。



联接的字段名称必须完全相同。联接的字段数可以是任意的。表格通常包含一个或几个共同字段。没有共同字段会致使表格生成笛卡儿积。所有字段均相同也是可能的,但通常没有意义。除非已在 **Join** 语句中指定先前加载表格的表格名称,否则 **Join** 前缀会使用先前最后创建的表格。因而此时两个语句的排列顺序并不是任意的。

## 使用 Join

Qlik Sense 脚本语言中的显式 **Join** 前缀可完全联接这两个表格。结果会生成一个表格。此类联接通常会导致非常大的表格。

执行以下操作：

1. 打开高级脚本编写教程应用程序。
2. 在**数据加载编辑器**中新增脚本段。
3. 调用部分 *Transactions*。
4. 在右侧菜单的 **AttachedFiles** 下,单击**选择数据**。
5. 上传然后选择 *Transactions.csv*。



确保选中了**字段名称**下面的**嵌入的字段名称**以包含您加载数据时表格字段的名称。

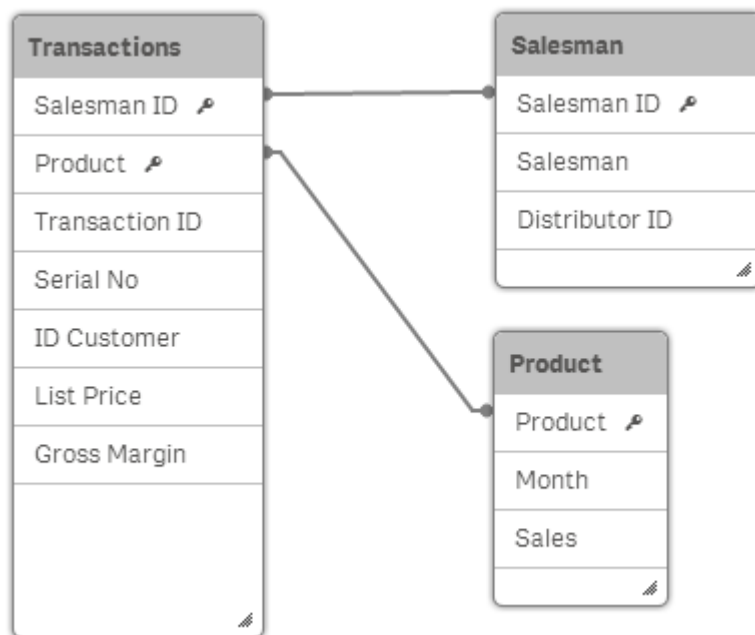
6. 在**选择数据自**窗口中,单击**插入脚本**。
7. 上传然后选择 *Salesman.xlsx*。
8. 在**选择数据自**窗口中,单击**插入脚本**。

您的脚本应如下所示：

```
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. 单击**加载数据**。
10. 打开**数据模型查看器**。数据模型如下所示：

数据模型: *Transactions*、*Salesman* 和 *Product* 表格



然而, 让 *Transactions* 和 *Salesman* 表格分离可能并非所需的结果。最好是将两个表格联接。

执行以下操作:

1. 要设置联接表的名称, 请在第一条 **LOAD** 语句上方添加以下行:  
Transactions:
2. 要联接 *Transactions* 和 *Salesman* 表格, 在第二个 **LOAD** 语句上方添加以下行:  
Join(Transactions)

您的脚本应如下所示:

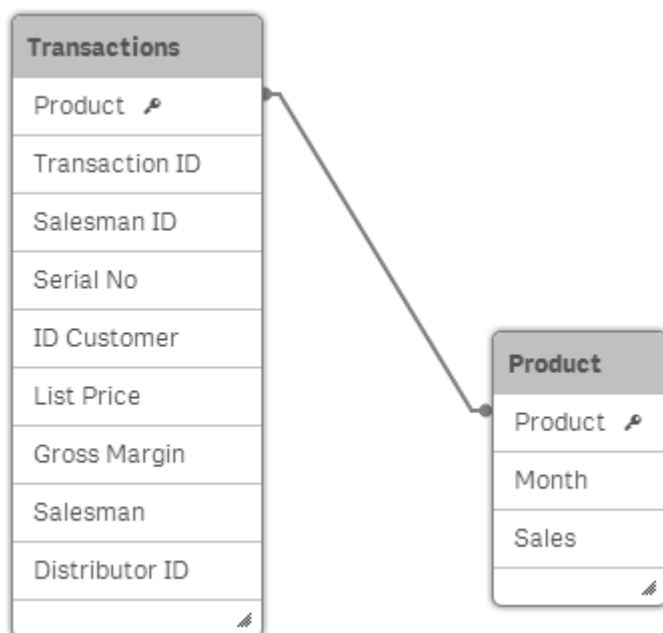
```

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. 单击**加载数据**。
4. 打开**数据模型查看器**。数据模型如下所示:

数据模型: *Transactions* 和 *Product* 表格



*Transactions* 和 *Salesman* 表格的所有字段现在都已合并成一个 *Transactions* 表格。



要了解有关何时使用 *Join* 的信息, 请参阅 *Qlik Community* 中的博客文章: [To Join or not to Join\( 联接或不联接\)](#)、[Mapping as an Alternative to Joining\( 将映射作为联接的替代方案\)](#)。这些行为将在 *QlikView* 的上下文中讨论。然而, 逻辑同样适用于 *Qlik Sense*。

## Keep

*Qlik Sense* 的其中一个主要功能就是使表格之间形成关联, 而不是联接这些表格, 这种关联可以大大减少使用内存, 提高处理速度并且灵活多变。*Keep* 功能旨在减少需要使用显式联接的情况。

两个 *LOAD* 或 *SELECT* 语句之间的 *Keep* 前缀有将两个表格中的一个或两个表格缩减为表格数据交集的效果, 然后才将其存储到 *Qlik Sense*。*Keep* 前缀必须是 *Inner*、*Left* 或 *Right* 关键字之一。选择表格记录的方法与相应联接方法相同。但是, 这两个表格并未联接, 而是分别命名后存储在 *Qlik Sense* 中。

## Inner

数据加载脚本中的 *Join* 和 *Keep* 前缀可以位于前缀 *Inner* 之后。

如果用于 *Join* 之前, 说明两个表格之间的联接应为内部联接。由此生成的表格所包含的两表格之间的组合必带有两表格的完整数据集。

如果用于 *Keep* 之前, 说明首先应使两个表格缩减为它们自身的共同交集, 然后才可在 *Qlik Sense* 中存储这些表格。

示例：

在这些示例中，我们使用源表格 *Table1* 和 *Table2*。

注意这些仅为示例。在 Qlik Sense 中没有附带的练习。

Table 1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

## Inner Join

首先，我们使用两个表格中合并的数据在表格上执行 Inner Join，从而使 *VTable* 仅包含一行，即两个表格中仅有的记录。

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

VTable

A	B	C
1	aa	xx

## Inner Keep

如果我们执行 Inner Keep，则会获得两个表格。这两个表格通过字段 *A* 关联。

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

VTab1

A	B
1	aa

VTab2

A	C
1	xx



## Left

数据加载脚本中的 **Join** 和 **Keep** 前缀可以位于前缀 **left** 之后。

如果用于 **Join** 之前, 说明两个表格之间的联接应为左联接。由此生成的表格所包含的两表格之间的组合仅带有第一个表格的完整数据集。

如果用于 **Keep** 之前, 说明首先应使第二个表格缩减为其与第一个表格间的共同交集, 然后才可在 **Qlik Sense** 中存储此表格。

示例:

在这些示例中, 我们使用源表格 *Table1* 和 *Table2*。

Table1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

首先, 我们在表格上执行 **Left Join**, 从而产生 *VTable*, 其中包含 *Table1* 中的所有行(与 *Table2* 中匹配的字段合并)。

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

VTable

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

如果我们执行 **Left Keep**, 则会获得两个表格。这两个表格通过字段 **A** 关联。

```
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 脚本语言中的 **Join** 和 **Keep** 前缀可以位于 **right** 前缀之后。

如果用于 **Join** 之前, 说明两个表格之间的联接应为右联接。生成的表格仅包含这两个表格的组合, 其中第二个表格带有完整数据集。

如果用于 **Keep** 之前, 说明首先应使第一个表格缩减为其与第二个表格间的共同交集, 然后才可在 Qlik Sense 中存储此表格。

示例:

在这些示例中, 我们使用源表格 *Table1* 和 *Table2*。

Table1

A	B
1	aa
2	cc
3	ee

Table2

A	C
1	xx
4	yy

首先, 我们在表格上执行 **Right Join**, 从而产生 *VTable*, 其中包含 *Table2* 中的所有行(与 *Table1* 中匹配的字段合并)。

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

VTable

A	B	C
1	aa	xx
4	-	yy

如果我们执行 **Right Keep**, 则会获得两个表格。这两个表格通过字段 **A** 关联。

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

VTab1

A	B
1	aa

VTab2

A	C
1	xx
4	yy

## 2.3 使用内部记录函数 Peek、Previous 和 Exists

当对当前记录的评估需要一个来自以前加载的数据记录的值时, 使用这些函数。

在此部分教程中, 我们将检测 **Peek()**、**Previous()** 和 **Exists()** 函数。

### Peek()

**Peek()** 用于在表格中返回已经加载行的字段值。可以将行号指定为表格。如果未指定行号, 将使用上次加载的记录。

语法:

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

行必须为整数。0 表示第一个记录, 1 表示第二个记录, 以此类推。负数表示从表格末端开始计算的顺序。-1 表示最后读取的记录。

如果未陈述行, 则假定为 -1。

**Tablename** 是表格标签, 不以冒号结束。如果未指定 **tablename**, 则假定为当前表格。如果用于 **LOAD** 语句之外或指向另外一个表格, 则必须包括 **tablename**。

### Previous()

**Previous()** 用于查找使用因 **where** 子句而未丢弃的以前输入记录的数据的 **expr** 表达式的值。在内部表格的首个记录中, 此函数将返回 **NULL** 值。

语法:

**Previous**(expression)

可以嵌套 `Previous()` 函数以访问能够进一步回滚的记录。数据直接从输入源获取，使其也可引用尚未载入 Qlik Sense 的字段，也就是说即使尚未将它们存储在相关的数据库中。

## Exists()

**Exists()** 用于确定是否已经将特定字段值加载到数据加载脚本中的字段。此函数用于返回 `TRUE` 或 `FALSE`，这样它可以用于 `LOAD` 语句或 `IF` 语句中的 **where** 子句。

语法：

`Exists(field [, expression ] )`

字段到目前为止必须存在于由脚本加载的数据中。*Expression* 是对字段值的表达式求值，以在指定字段中查找。如果省略，指定字段中的当前记录值将被假定。

## 使用 Peek() 和 Previous()

按它们的最简单形式，`Peek()` 和 `Previous()` 用于识别表格内的特定值。下面是您将在本练习中加载的 *Employees* 表中数据的示例。

来自员工表格的数据的示例

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

目前，这只能用于收集月份、雇用和终止的相关数据，因此我们要使用 `Peek()` 和 `Previous()` 函数为 *Employee Count* 和 *Employee Var* 添加字段，用于查看员工总数的每月差异。

执行以下操作：

1. 打开高级脚本编写教程应用程序。
2. 在数据加载编辑器中新增脚本段。
3. 调用部分 *Employees*。
4. 在右侧菜单的 **AttachedFiles** 下，单击 **选择数据**。
5. 上传然后选择 *Employees.xlsx*。



确保选中了 *Field names* 下面的 *Embedded field names* 以包含您加载数据时表格字段的名称。

6. 在 **选择数据** 窗口中，单击 **插入脚本**。

您的脚本应如下所示：

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

## 7. 修改脚本，使其如下所示：

```
[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);
```

Excel 工作表中 *Date* 字段里的日期的格式为 MM/DD/YYYY。要确保使用来自系统变量的格式正确解释日期，将 *Date* 函数应用至 *Date* 字段。

*Peek()* 函数用于识别为所定义字段加载的任何值。在该表达式中，我们将先确定 *rowno()* 是否等于 1。如果等于 1，则 *Employee Count* 将不存在，因此我们使用 *Hired* 减去 *Terminated* 的差额填充该字段。

如果 *rowno()* 大于 1，我们查看最后一个月的 *Employee Count*，然后使用该数字加上该月的 *Hired* 减去 *Terminated* 员工的差额。

另请注意，在 *Peek()* 函数中，我们使用 (-1)。这意味着 Qlik Sense 要查看当前记录上面的记录。如果未指定 (-1)，Qlik Sense 将假定您想要查看之前的记录。

## 8. 将以下内容添加至您脚本的末尾：

```
[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()* 函数用于识别为所定义字段加载的上一个值。在该表达式中我们将先确定 *rowno()* 是否等于 1。如果等于 1，我们知道 *Employee Var* 将不会存在，因为上一个月的 *Employee Count* 没有任何记录。因此我们只能为该值输入 0

如果 *rowno()* 大于 1，我们知道将会存在 *Employee Var*，因此我们要查看上一个月的 *Employee Count*，然后从当前月份的 *Employee Count* 减去该数字，得出 *Employee Var* 字段中的值。

您的脚本应如下所示：

```
[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. 单击加载数据。

在应用程序概述的新工作表中，使用 *Date*, *Hired*, *Terminated*, *Employee Count* 和 *Employee Var* 作为表的列创建表。最终生成的表格将如下所示：

在脚本中使用 *Peek* 和 *Previous* 后的表格

My new sheet

Date	Sum(Hired)	Sum(Terminated)	Sum([Employee Var])	Employee Count
<b>Totals</b>	<b>77</b>	<b>31</b>	<b>40</b>	
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()* 和 *Previous()* 可让用户在表格中针对性地定义行。两个函数之间的最大区别在于，*Peek()* 函数可让用户查看之前尚未加载到脚本的字段，而 *Previous()* 函数只能让用户查看之前加载的字段。*Previous()* 在 *LOAD* 语句的输入中操作，而 *Peek()* 在 *LOAD* 语句的输出中操作。（和 *RecNo()* 与 *RowNo()* 之间的差值相同）这意味着如果您使用 *Where* 子句，则这两个函数的作用不同。

因此，当您需要显示当前值与上一个值时更适合使用 *Previous()* 函数。在本例中，我们计算了每个月的员工变动。

当您针对之前尚未加载到表格的字段或如果您需要针对特定行时，更适合使用 *Peek()* 函数。如本例所示，我们通过查看上一个月的 *Employee Count*，然后加上当前月的雇用和终止员工之间的差额计算 *Employee Count*。请记住，*Employee Count* 不是原始文件中的一个字段



要了解关于 *Peek()* 和 *Previous()* 的更多信息，请参阅 *Qlik Community* 中的该文章：[Peek\(\) vs Previous\(\) - When to Use Each](#)。这些行为将在 *QlikView* 的上下文中讨论。然而，逻辑同样适用于 *Qlik Sense*。

## 使用 *Exists()*

*Exists()* 函数通常与脚本中的 *Where* 子句搭配使用以加载数据，如果已经在数据模型中加载相关数据。

在下面的示例中，我们也将使用 *Dual()* 函数将数值赋值给字符串。

执行以下操作：

1. 创建一个新应用程序并为其指定一个名称。
2. 在数据加载编辑器中新增脚本段。
3. 调用部分 *People*。

## 4. 输入以下脚本：

```
//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. 单击**加载数据**。

在脚本中，*Age* 和 *AgeBucket* 字段只有已在数据模型中加载 *PersonID* 后才加载。

注意，在 *AgeTemp* 表格中已经为 *PersonID* 11 和 12 列出年龄，但由于在数据模型( *People* 表格)中尚未加载这些 ID，因此它们已被 *Where Exists(PersonID)* 子句排除。该子句语法也可以这样书写：*Where Exists(PersonID, PersonID)*。

脚本的输出和以下类似：

在脚本中使用 *Exists* 后的表格

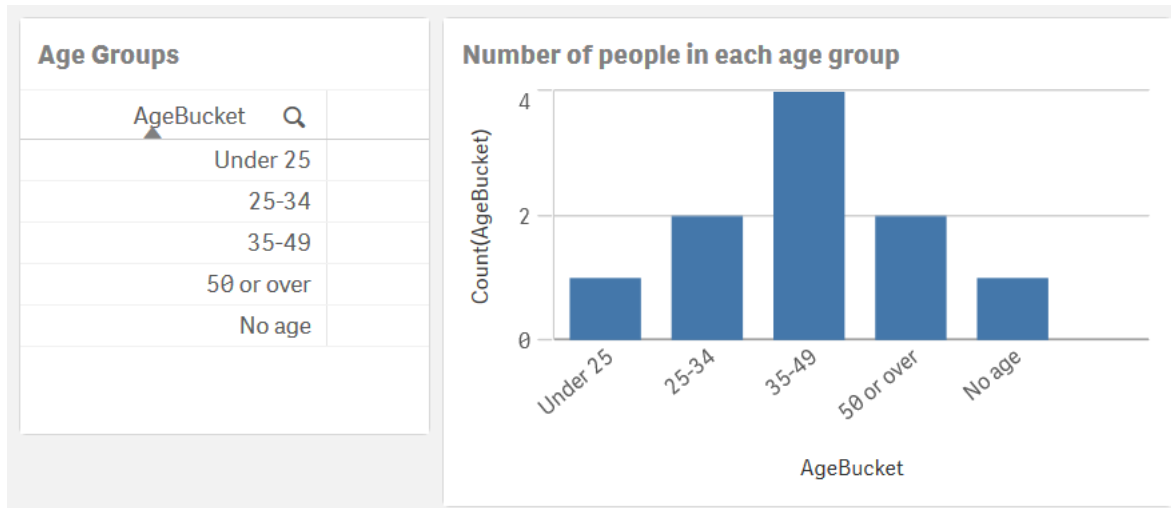
My new sheet

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

如果没有将 *AgeTemp* 表格中的任何 *PersonID* 加载到数据模型，则不会将 *Age* 和 *AgeBucket* 字段联接到 *People* 表格。使用 *Exists()* 函数可以帮助防止在数据模型中孤立记录/数据，即 *Age* 和 *AgeBucket* 字段没有任何相关的人员。

6. 创建一个新工作表并为其指定一个名称。
7. 打开新表格，然后单击**编辑工作表**。
8. 使用维度 *AgeBucket* 将标准表格添加至工作表，并将可视化命名为 *Age Groups*。

9. 使用维度 **AgeBucket** 和度量 **Count([AgeBucket])** 将条形图添加到表格。命名可视化 **Number of people in each age group**。
10. 将表格和条形图的属性添加到首选项，然后单击**完成**。  
现在，您的工作表应如下所示：  
具有依据年龄的分组的工作表



如果需要将数值赋值给字符串，**Dual()** 函数在脚本或图表表达式中很有用。

在以上脚本中，您必须拥有用于加载年龄的应用程序，并且您已决定将这些年龄存放在存储段中，这样您就可以根据年龄存储段和实际年龄创建可视化效果。存储段分为 25 岁以下，25 至 35 岁之间等。通过使用 **Dual()** 函数，可以向年龄存储段分配数值，稍后用来在列表框或图表中对年龄存储段进行排序。因此，如应用程序表格所示，排序后会将“无年龄”放在列表的末尾。



要了解关于 **Exists()** 和 **Dual()** 的更多信息，请参阅 **Qlik Community** 中的该文章：[Dual 和 Exists - 有用的函数](#)

## 2.4 匹配间隔和迭代加载

**LOAD** 或 **SELECT** 语句的 **Intervalmatch** 前缀用于链接离散数值和一个或多个时间间隔数字。此功能十分强大，如可用于生产环境。

### 使用 **IntervalMatch()** 前缀

最基本的间隔匹配是，在一个表格中有数字或日期(事件)列表，在另一个表格中有间隔列表时。目的是联接两个表格。一般情况下，这是一种多对多的关系，即间隔可以拥有属于它的多个日期，且一个日期可以属于多个间隔。要解决此问题，您需要在两个原始表格之间创建一个桥接表格。可以通过多种方法实现此操作。

在 **Qlik Sense** 中解决此问题的最简单方法是在 **LOAD** 或 **SELECT** 语句前面使用 **IntervalMatch()** 前缀。**LOAD/SELECT** 语句只能包含两个字段，“**From**”和“**To**”字段用于定义间隔。然后，**IntervalMatch()** 前缀将会生成加载间隔和之前加载数字字段之间的所有组合，用于指定为参数的前缀。



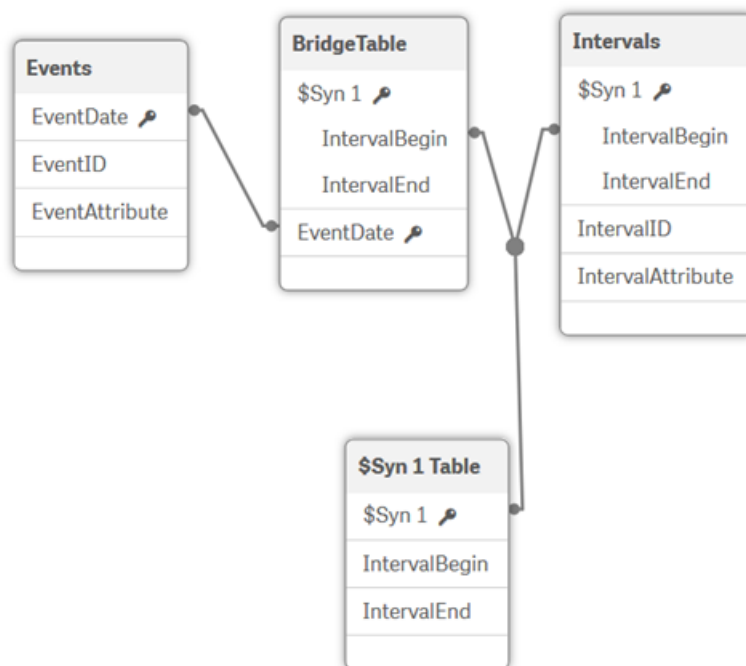
执行以下操作：

1. 创建一个新应用程序并为其指定一个名称。
2. 在**数据加载编辑器**中新增脚本段。
3. 调用部分 *Events*。
4. 在右侧菜单的 **AttachedFiles** 下，单击**选择数据**。
5. 上传然后选择 *Events.txt*。
6. 在**选择数据**自窗口中，单击**插入脚本**。
7. 上传然后选择 *Intervals.txt*。
8. 在**选择数据**自窗口中，单击**插入脚本**。
9. 在脚本中，命名第一个表格 *事件*，然后命名第二个表格 *Intervals*。
10. 在脚本末尾添加 **IntervalMatch** 创建第三个表格，以桥接前面两个表格：  

```
BridgeTable: IntervalMatch (EventDate) LOAD distinct IntervalBegin, IntervalEnd Resident Intervals;
```
11. 您的脚本应如下所示：  

```
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. 单击**加载数据**。
13. 打开**数据模型查看器**。数据模型如下所示：

数据模型: *Events*、*BridgeTable*、*Intervals* 和 *\$Syn1* 表格



包含复合关键字段( *IntervalBegin* 和 *IntervalEnd* 字段)的数据模型会将自己显示为 Qlik Sense 合成键。

基本表格如下：

- *Events* 表格只能包含每个事件的一个记录。
- *Intervals* 表格只能包含每个间隔的一个记录。
- 桥接表格只能包含事件和间隔的每个组合的一个记录，并且用于联接前面两个表格。

注意，如果间隔重叠，则事件可以属于多个间隔。当然，间隔也可以拥有属于它的多个事件。

从规范化和紧凑感方面来看，此数据模型是最佳模型。*Events* 表格和 *Intervals* 表格都保持不变，并包含原始数量的记录。在这些表格中进行的所有 Qlik Sense 计算操作，如 **Count** (*EventID*) 都能够正常运行，因此需要进行正确评估。



要了解关于 *IntervalMatch()* 的更多信息，请参阅 Qlik Community 中该文章：[Using IntervalMatch\(\)](#)

## 使用 While 循环和迭代加载 IterNo()

您可以使用 **While** 循环和 **IterNo()** 创建间隔的下限和上限之间的枚举值获得几乎完全相同的桥接表格。

可以使用 **While** 子句在 **LOAD** 语句内创建循环。例如：

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

该 LOAD 语句将针对每个输入记录进行循环并反复加载此语句, 只要 While 子句中的表达式值为 True。IterNo() 函数在第一次迭代中返回“1”, 在第二次迭代中返回“2”, 以此类推。

您拥有间隔的主键 IntervalID, 因此在脚本中的唯一区别是创建桥接表格的方式:

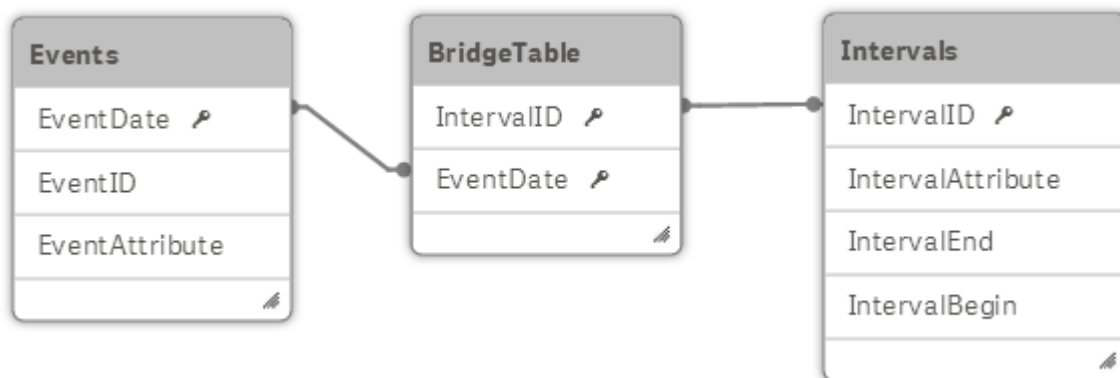
执行以下操作:

1. 将现有 Bridgetable 语句替换为以下脚本。

```
BridgeTable: LOAD distinct * where Exists(EventDate); LOAD IntervalBegin + IterNo() - 1
as EventDate, IntervalID Resident Intervals while IntervalBegin + IterNo() - 1
<= IntervalEnd;
```

2. 单击**加载数据**。
3. 打开**数据模型查看器**。数据模型如下所示:

数据模型: *Events*、*BridgeTable* 和 *Intervals* 表格

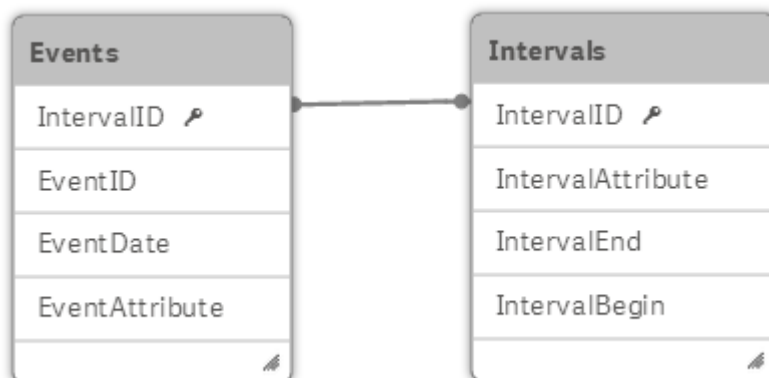


通常, 使用三个表格的解决办法是最好的, 因为它允许在间隔和事件之间存在多对多的关系。但是, 一种常见的情况是您知道事件只能属于一个单一的间隔。在该情况下, 桥接表格实际并不必要: *IntervalID* 可直接存储在事件表内。可以通过多种方法实现此操作, 但最有效的方法是联接 *Bridgetable* 表格和 *Events* 表格。

4. 将以下脚本添加至您脚本的末尾:

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

5. 单击**加载数据**。
6. 打开**数据模型查看器**。数据模型如下所示:

数据模型: *Events* 和 *Intervals* 表格

## 开放和封闭间隔

间隔是开放间隔还是封闭间隔由间隔中是否包含这些端点确定。

- 如果包含端点, 则该间隔为封闭间隔:  
 $[a,b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$
- 如果不包含端点, 则该间隔为开放间隔:  
 $]a,b[ = \{x \in \mathbb{R} \mid a < x < b\}$
- 如果包含一个端点, 则该间隔为半开放间隔:  
 $[a,b[ = \{x \in \mathbb{R} \mid a \leq x < b\}$

如果出现间隔重叠和数字属于一个以上间隔的情况, 则通常需要使用封闭间隔。

但是, 在某些情况下, 您不希望重叠间隔, 希望数字只属于一个间隔。因此, 如果一个端点是一个间隔的结束, 同时它也是下一个间隔的开始, 这就形成了一个间隔。拥有此值的数字将同时属于两个间隔。因此, 您应使用半开放间隔。

解决此问题的一种实用的解决办法是从所有间隔的结束值中减去一个非常小的数目, 从而创建封闭间隔, 但间隔不重叠。如果数字为日期, 则最简单的方法是使用 `DayEnd()` 函数返回一天的最后一毫秒:

```
Intervals: LOAD..., DayEnd(IntervalEnd - 1) as IntervalEnd From Intervals;
```

您也可以手动减去一个小的数目。如果这样做, 请确保减去的数目不能太小, 因为该操作将会四舍五入到 52 位有效的二进制数 (14 位十进制数)。如果您使用太小的数目, 差额将不会太明显, 这样将会用回原始数。

## 3 数据清理

有时候, 当您将源数据加载到 **Qlik Sense** 时, 不需要您在 **Qlik Sense** 应用程序中等待完成。**Qlik Sense** 提供了一系列函数和语句, 用于将数据转换成所需的格式。

当脚本运行时可以在 **Qlik Sense** 脚本中使用映射来替换或修改字段值或名称, 以便于可以使用映射清理数据, 并使它更加一致或替换部分或所有的字段值。

当您从不同表格加载数据时, 表示相同事物的字段值并不总是拥有一致的名称。由于缺乏一致性会妨碍关联, 因而需要解决此问题。这一问题通过创建映射表比较字段值可以得到完美解决。

### 3.1 映射表

通过 **Mapping** 或 **Mapping** 加载的表格的处理方式不同于其他表格。它们存储在内存的单独区域内, 并在脚本运行时仅用作映射表。在脚本运行后, 将自动删除这些表格。

规则:

- 映射表必须拥有列, 第一列包含比较值, 第二列包含所需的映射值。
- 两列必须命名, 且名称之间不存在关联。列名称不得与常规内部表格的字段名称相联系。

### 3.2 Mapping 函数和语句

本教程将详细讨论以下映射函数/语句:

- Mapping 前缀
- ApplyMap()
- MapSubstring()
- Map ... Using 语句
- Unmap 语句

### 3.3 Mapping 前缀

**Mapping** 前缀用于脚本中创建映射表。然后, 可将映射表与 **ApplyMap()** 函数、**MapSubstring()** 函数或 **Map ... Using** 语句搭配使用。

执行以下操作:

1. 创建一个新应用程序并为其指定一个名称。
2. 在**数据加载编辑器**中新增脚本段。
3. 调用部分 *Countries*。
4. 输入以下脚本:

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

**CountryMap** 表格存储两列：**Country** 和 **NewCountry**。**Country** 列用于存储在 **Country** 字段中以不同方式输入的国家/地区。**NewCountry** 列用于存储映射值的方式。该映射表将用于存储 **Country** 字段中一致的 **US** 国家/地区值。例如，如果 **U.S.A.** 存储在 **Country** 字段中，则会将其映射为 **US**。

## 3.4 ApplyMap() 函数

使用 **ApplyMap()** 可根据之前创建的映射表替换字段中的数据。在可以使用 **ApplyMap()** 函数之前需要加载映射表。您将加载的表格 **Data.xlsx** 中的数据如下：

数据表

ID	Name	Country	Code
1	John Black	U.S.A.	SDFGBS1DI
2	Steve Johnson	U.S.	2ABC
3	Mary White	美国	DJY3DFE34
4	Susan McDaniels	u	DEF5556
5	Dean Smith	US	KSD111DKFJ1

注意应以不同方式输入国家/地区。为了使 **Country** 字段保持一致，应先加载映射表，然后再使用 **ApplyMap()** 函数。

执行以下操作：

1. 在上面输入的脚本下面，选择并加载 **Data.xlsx**，然后插入脚本。
2. 在新创建的 **LOAD** 语句上方输入以下内容：

**Data:**

您的脚本应如下所示：

```
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. 修改包含 **Country** 的行，如下所示：

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

**ApplyMap()** 函数的第一个参数的映射名称应使用单引号括起来。第二个参数是包含要替换的数据的字段。

4. 单击**加载数据**。

最终生成的表格如下所示：

表格示出了使用 *ApplyMap()* 函数加载的数据

My new sheet

Click to add title				
ID	Name	Country	Code	
1	John Black	US	SDFGBS1DI	
2	Steve Johnson	US	2ABC	
3	Mary White	US	DJY3DFE34	
4	Susan McDaniels	u	DEF5556	
5	Dean Smith	US	KSD111DKFJ1	

*United States* 的不同拼写已更改为 *US*。如果存在一个拼写不正确的记录，则 *ApplyMap()* 函数不会更改该字段值。使用 *ApplyMap()* 函数，您可以使用第三个参数添加默认表达式，如果映射表不包含匹配值。

- 添加 'us' 作为 *ApplyMap()* 函数的第三个参数，用于处理当国家/地区输入不正确时的情况：  
`ApplyMap('CountryMap', Country, 'US') as Country,`

您的脚本应如下所示：

```
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);
```

- 单击**加载数据**。

最终生成的表格如下所示：

表格示出了使用 *ApplyMap* 函数加载的数据

My new sheet

Click to add title				
ID	Name	Country	Code	
1	John Black	US	SDFGBS1DI	
2	Steve Johnson	US	2ABC	
3	Mary White	US	DJY3DFE34	
4	Susan McDaniels	US	DEF5556	
5	Dean Smith	US	KSD111DKFJ1	



要了解关于 *ApplyMap()* 的更多信息，请参阅 *Qlik Community* 中该文章：[不要联接 - 改为使用 Applymap](#)

## 3.5 MapSubstring() 函数

MapSubstring() 函数可让您映射字段的一部分。

在由 ApplyMap() 创建的表格中, 我们现在要将数字写入作为文本, 因此将会使用 MapSubstring() 函数将数字数据替换为文本。

为此, 首先需要创建映射表。

执行以下操作:

1. 在 *CountryMap* 部分末尾但在 *Data* 部分前面添加以下脚本行。

```
CodeMap: MAPPING LOAD * INLINE [ F1, F2 1, one 2, two 3, three 4, four 5, five 11,
eleven ];
```

在 *CodeMap* 表中, 已经映射数字 1 至 5, 以及 11。

2. 在脚本的 *Data* 部分中, 修改 Code 语句, 如下所示:

```
MapSubString('CodeMap', Code) as Code
```

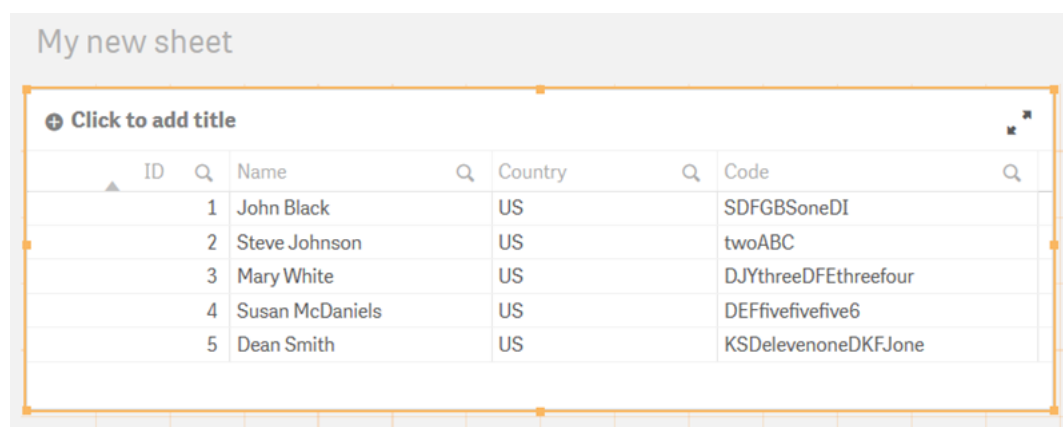
您的脚本应如下所示:

```
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. 单击**加载数据**。

最终生成的表格如下所示:

表格示出了使用 *MapSubString* 函数加载的数据



ID	Name	Country	Code
1	John Black	US	SDFGBSoneDI
2	Steve Johnson	US	twoABC
3	Mary White	US	DJYthreeDFEthreefour
4	Susan McDaniels	US	DEfffivefive6
5	Dean Smith	US	KSDelevenoneDKFJone

在 *Code* 字段中将数字字符替换为文本。如果数字类似 ID=3 和 ID=4 出现多次, 则文本也重复。ID=4. *Susan McDaniels* 在其代码中包含 6。由于在 *CodeMap* 表中不映射 6, 因此它保持不变。ID=5, *Dean Smith* 在其代码中包含 111。已经将其映射为“elevenone”。





要了解关于 `MapSubstring()` 的更多信息, 请参阅 Qlik Community 中该文章: [Mapping ... and not the geographical kind\(映射...而不是地理类型\)](#)

## 3.6 Map ... Using

`Map ... Using` 语句也可以用来将映射应用到字段。不过, 它的工作方式与 `ApplyMap()` 稍有不同。在 `ApplyMap()` 每次处理映射时都会遇到字段名称, `Map ... Using` 在处理映射会将值存储在内部表的字段名称中。

示例如下。假定我们在脚本中多次加载 `Country` 字段, 并希望在每次加载该字段时应用映射。可以使用 `ApplyMap()` 函数(如本教程前面所述), 或者可以使用 `Map ... Using`。

如果使用 `Map ... Using`, 则在将字段存储到内部表时将映射应用到该字段。因此, 在下例中, 已将映射应用到 `Data1` 表中的 `Country` 字段, 但不会应用到 `Data2` 表中的 `Country2` 字段。这是因为 `Map ... Using` 语句仅适用于名为 `Country` 的字段。当将 `Country2` 字段存储到内部表后, 其名称不再是 `Country`。如果您想要将映射应用到 `Country2` 表格, 则需要使用 `ApplyMap()` 函数。

`Unmap` 语句结束 `Map ... Using` 语句, 因此如果在执行 `Unmap` 语句后加载 `Country`, 将不会应用 `CountryMap`。

执行以下操作:

1. 将 `Data` 表格的脚本替换为以下内容:

```
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. 单击**加载数据**。

最终生成的表格如下所示:

表格示出了使用 `Map ... Using` 函数加载的数据

ID	Name	Country	Country2
1	John Black	US	U.S.A.
2	Steve Johnson	US	U.S.
3	Mary White	US	United States
4	Susan McDaniels	u	u
5	Dean Smith	US	US

## 4 处理层次结构数据

层次结构是所有商业智能解决方案的重要组成部分，用于介绍合理包含不同级别粒度的维度。有些层次结构非常简单和直观，而其他层次结构非常复杂，需要经过深思熟虑后才能进行正确建模。

从层次结构的顶层到底层，会员的信息越来越详细。例如，在拥有 **Market**、**Country**、**State** 和 **City** 级别的维度中，会员美洲出现在层次结构的顶层，会员美国出现在第二层，会员加州出现在第三层且旧金山出现在底层。加州比美国更加具体，而旧金山比加州更加具体。

在关系模型中存储层次结构是多种解决方案的共同挑战。可以通过以下多种方法来解决该挑战：

- 水平层次结构
- 邻接表模型
- 路径枚举法
- 嵌套集合模型
- 祖先列表

对于本教程，我们将创建一个祖先列表，因为它能够以可直接在查询中使用的方式显示层次结构。有关其他方法的更多信息，请参阅 [Qlik Community](#)。

### 4.1 Hierarchy 前缀

**Hierarchy** 前缀是放置在 **LOAD** 或 **SELECT** 语句前面的脚本命令，用于加载相邻节点表。**LOAD** 语句需要具有至少三个字段：**ID** 是节点的唯一密钥，涉及父级和名称。

前缀将加载的表格转换成扩展的节点表格；表格含有许多其他列；层次结构的每一层都拥有一个表格。

执行以下操作：

1. 创建一个新应用程序并为其指定一个名称。
2. 在**数据加载编辑器**中新增脚本段。
3. 调用部分 *Wine*。
4. 在右侧菜单的 **AttachedFiles** 下，单击**选择数据**。
5. 上传然后选择 *Winedistricts.txt*。
6. 在**选择数据自**窗口中，取消选中 *Lbound* 和 *Rbound* 字段，从而不加载它们。
7. 单击**插入脚本**。
8. 在 **LOAD** 语句上方输入以下内容：

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

您的脚本应如下所示：

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

9. 单击**加载数据**。
10. 使用**数据模型查看器**的**预览**部分查看生成的表格。

生成的扩展节点表拥有与其源表数量相同的记录：每个节点一条记录。扩展节点表非常实用，因为它满足在关系模型中分析层次结构的许多要求：

- 所有节点名称都存在于一个表格中的同一列中，因此可用于搜索。
- 此外，已将不同的节点级别扩展到每个表格中的一个字段；可用于向下钻取组或透视表中的维度的字段。
- 此外，已将不同的节点级别扩展到每个表格中的一个字段；字段可用于向下钻取组。
- 可使它包含节点的唯一路径，从而按正确的顺序列出所有祖先节点。
- 可使它包含节点的深度，即到根节点的距离。

最终生成的表格如下所示：

表格示出了使用 *Hierarchy* 前缀加载的数据的示例

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		



要了解关于层级的更多信息，请参阅 *Qlik Community* 中该文章：[层级](#)

## 4.2 HierarchyBelongsTo 前缀

和 *Hierarchy* 前缀相似，*HierarchyBelongsTo* 前缀是放置在 **LOAD** 或 **SELECT** 语句前面的脚本命令，用于加载相邻节点表。

另外，**LOAD** 语句需要具有至少三个字段：**ID** 是节点的唯一密钥，涉及父级和名称。前缀将加载的表格转换成祖先节点表格，该表格拥有列出作为单独记录的祖先和后代的每个组合。因此，可以很轻松地找到特定节点的所有祖先或所有后代。

执行以下操作：

1. 修改**数据加载编辑器**中的 *Hierarchy* 语句，使其为如下所示：  
**HierarchyBelongsTo (NodeID, ParentID, NodeName, BelongsToID, BelongsTo)**
  2. 单击**加载数据**。
  3. 使用**数据模型查看器**的**预览**部分查看生成的表格。
- 祖先节点表可满足在关系模型中分析层次结构的许多要求：

- 如果节点 **ID** 表示单个节点，则上级 **ID** 表示层次结构的整个树和子树。
- 存在的所有节点名称均可在角色中作为节点和树，且可用于搜索。

- 可以使它包含节点深度与祖先节点深度之间的深度差，即到子树根节点的距离。
- 最终生成的表格如下所示：

表格示出了使用 *HierarchyBelongsTo* 前缀加载的数据

My new sheet

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

授权

通常在很少情况下，使用层次结构进行授权。其中一个示例是组织层次结构。每位经理都应有权查看与其自己的部门有关的所有信息，包括其所有子部门。但是，他们不得有权查看其他部门。

组织层次结构示例



这意味着将会允许不同的人查看组织的不同子树。授权表可能如下所示：

授权表格

ACCESS	NTNAME	PERSON	POSITION	PERMISSIONS
USER	ACME\JRL	"John"	CPO	HR
USER	ACME\CAH	Carol	CEO	CEO
USER	ACME\JER	James	Director Engineering	Engineering
USER	ACME\DBK	Diana	CFO	Finance
USER	ACME\RNL	Bob	COO	Sales
USER	ACME\LFD	Larry	CTO	Product

在此例中，允许 *Carol* 查看 *CEO* 及以下职位的任何相关信息；允许 *Larry* 查看 *Product* 组织；只允许 *James* 查看 *Engineering* 组织。

示例：

层次结构通常存储在相邻的节点表中。在本例中，要解决此问题，可以使用 `HierarchyBelongsTo` 加载相邻节点表格并命名上级字段 *Tree*。

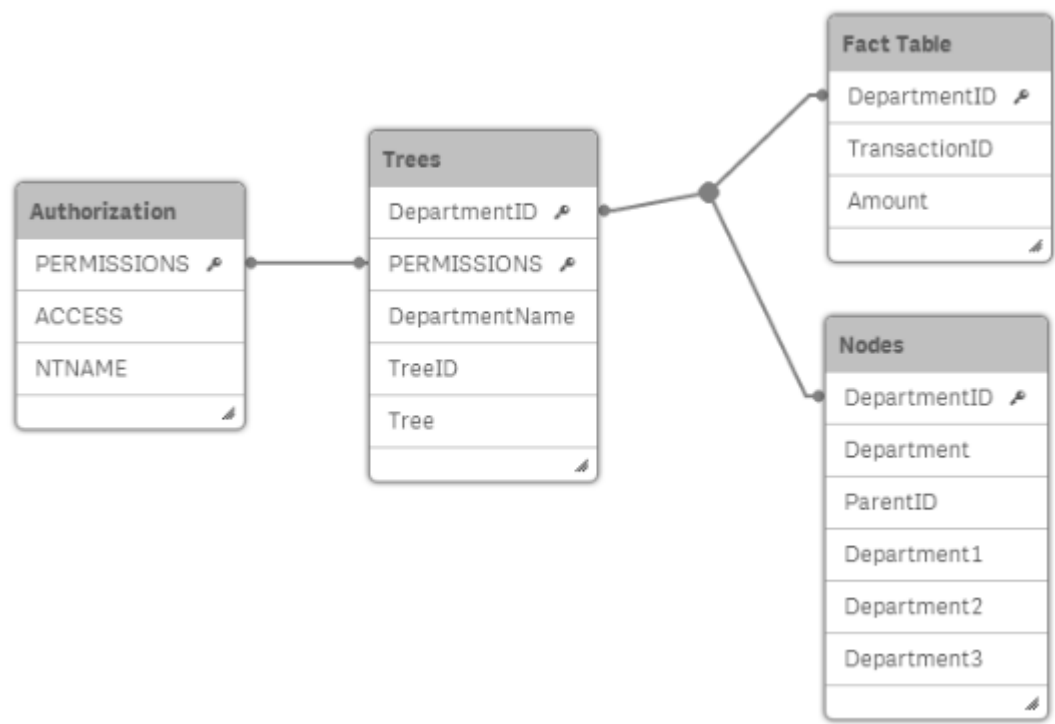
如果您希望使用 `Section Access`，则需要加载 *Tree* 的大写副本并将此称为新字段 *PERMISSIONS*。最后，您需要加载授权表。这两个最后步骤可使用以下脚本行完成。注意 `TempTrees` 表是通过 `HierarchyBelongsTo` 语句创建的表。

注意这仅为示例。在 `Qlik Sense` 中没有附带的练习。

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

此示例将生成以下数据模型：

数据模型: *Authorization*、*Trees*、*Fact* 和 *Nodes* 表格



## 5 QVD 文件

QVD (QlikView Data) 文件是包含从 Qlik Sense 或 QlikView 中所导出数据的表格的文件。QVD 是本地 Qlik 格式, 只能由 Qlik Sense 或 QlikView 写入和读取。当从 Qlik Sense 脚本中读取数据时, 该文件格式可提升速度, 同时又非常紧凑。从 QVD 文件读取数据通常比从其他数据源读取快 10 到 100 倍。

QVD 文件可以用两种模式读取: 标准(快速)和优化(超快)。所选模式由 Qlik Sense 脚本引擎自动确定。尽管字段可以重命名, 但仅当全部加载字段在无任何转换的形式下读取(操作字段公式)时才可以使用优化模式。导致 Qlik Sense 解压记录的 Where 子句也将禁用优化加载。

QVD 文件正好包含一个数据表, 由三个部分组成:

- 在表格中描述字段的 XML 标题(UTF-8 字符集)、后续信息的布局以及一部分其他元数据。
- 字节填充格式符号表。
- 位填充格式实际表。

QVD 文件具有许多用途。可轻易识别四种主要的用途。在任何给定的情况下都可以应用不只一个:

- 提高数据加载速度  
通过缓冲 QVD 文件中输入数据的不改变或缓慢改变部分, 执行大型数据集脚本可变得相当快。
- 降低数据库服务器上的加载量  
从外部数据源提取的数据量也可大幅度降低。这可减少外部数据库的工作量和网络流量。而且, 当几个 Qlik Sense 脚本共享相同的数据时, 只需要将这些数据从源数据库加载到 QVD 文件一次即可。其他应用程序可以利用此 QVD 文件中相同的数据。
- 合并多个 Qlik Sense 应用程序数据。  
使用 Binary 脚本语句, 可能只能从单个 Qlik Sense 应用程序将数据加载到另一个应用程序, 但使用 QVD 文件, Qlik Sense 脚本可以合并任何数量的 Qlik Sense 应用程序数据。这使在一个应用程序中合并不同企业单位的相似数据等成为可能。
- 增量加载  
在许多常见情况下, QVD 功能可用于简化增量加载, 通过独家从不断扩大的数据库中加载新记录。

### 5.1 创建 QVD 文件

可通过两种方法创建 QVD 文件:

- 在 Qlik Sense 脚本中使用 Store 命令可进行显式创建并命名。  
在脚本中说明要将先前读取的表格或其部分导出到您选定位置上的一个明确命名文件。
- 从脚本中自动创建并维护。  
通过为 load 或 select 语句加入 Buffer 前缀, Qlik Sense 将自动创建 QVD 文件, 但重新加载数据时需使用某些条件代替初始数据源。

在结果 QVD 文件之间没有不同, 如在读取速度方面。

## Store

该脚本创建显式命名的 QVD、CSV 或 txt 文件。

### 语法：

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

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

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

### 示例：

```
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';
```

### 执行以下操作：

1. 打开高级脚本编写教程应用程序。
2. 单击 *Product* 脚本段。
3. 将以下内容添加至脚本的末尾：

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

您的脚本应如下所示：

```
Crosstabe(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. 单击加载数据。

*Product.qvd* 文件现在将会出现在文件列表中。

此数据文件是 **Crosstable** 脚本的结果且也是一个包含三列的表格，每个类别都拥有一列 (Product, Month, Sales)。此数据文件现在可用于替换整个 *Product* 脚本部分。

## 5.2 从 QVD 文件读取数据

可以通过以下方法由 Qlik Sense 读入或访问 QVD 文件：

- 加载 QVD 文件作为显式数据源。QVD 文件可由 Qlik Sense 脚本中的 load 语句引用，与其他类型的文本文件一样 (csv、fix、dif、biff 等)。



示例：

```
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);
```

- 自动加载缓冲 QVD 文件。当在 `load` 或 `select` 语句上使用 `buffer` 前缀时，无需明确说明读取的语句。Qlik Sense 将自行确定使用 QVD 文件数据的程度这一点同通过原始 `LOAD` 或 `SELECT` 语句获取数据相反。
- 通过脚本访问 QVD 文件。许多脚本函数(都以 QVD 开头)都可用于检索在 QVD 文件的 XML 标题中发现的数据的不同信息。

执行以下操作：

1. 在 *Product* 脚本段中注释掉整个脚本。

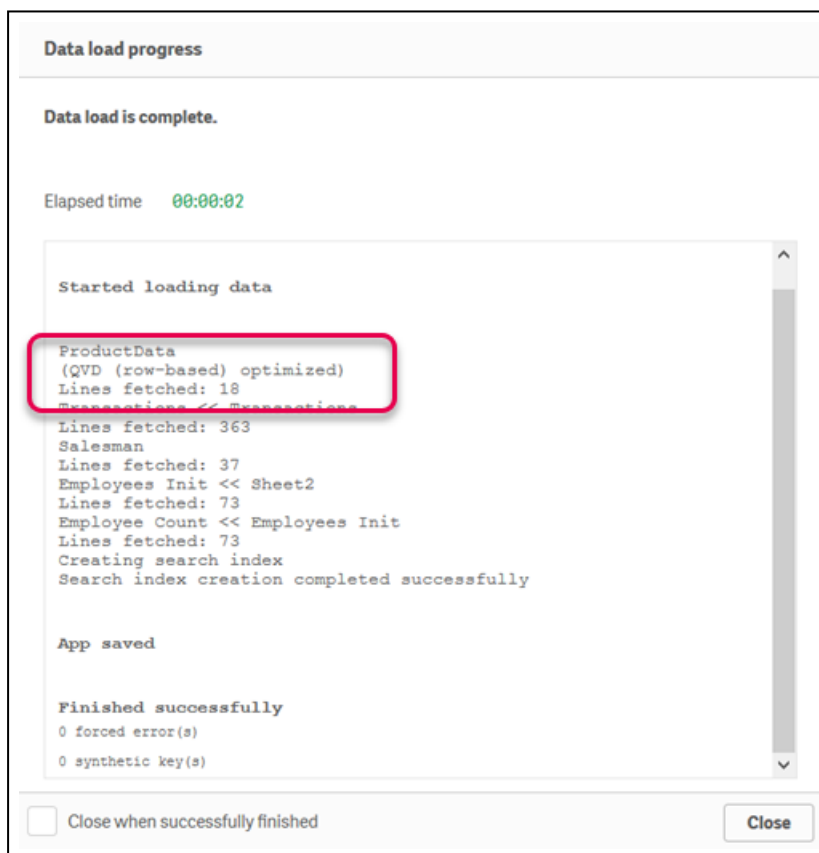
2. 输入以下脚本：

```
Load * from [lib://AttachedFiles/ProductData.qvd](qvd);
```

3. 单击**加载数据**。

将数据从 QVD 文件加载。

数据加载进度窗口





要了解有关将 QVD 文件用于增量加载的信息, 请参阅 *Qlik Community* 中的该文章: [Qlik 增量加载的概览](#)

## Buffer

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

### 语法:

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

如果未使用任何选项, 则首次执行脚本时创建的 QVD 缓冲将无限期使用。

### 示例:

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

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

**Amount** 即指定时间周期的数字。可能要使用 **Decimals**。如果省略, 则假定单位为天数。

**stale after** 选项通常与数据库源一起使用, 数据库源在初始数据上并无简单的时间戳。**stale after** 子句仅陈述自 QVD 缓冲创建时间计起的时间周期, 此后其即被视为无效。QVD 缓冲在此之前会被用作数据源, 在此之后则使用原始数据源。QVD 缓冲文件随后会自动更新, 同时新周期开始。

### 示例:

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

## Incremental

**incremental** 选项可实现仅读取基础文件的一部分。文件先前大小存储在 QVD 文件中的 **XML** 标题中。这对日志文件特别有用。上一步载入的全部记录都可从 QVD 文件读取, 而后续新记录可从原始数据源读取, 这样就可创建一个更新的 QVD 文件。

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

### 示例:

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

QVD 缓冲通常在创建脚本的应用程序内整个脚本执行过程不再引用时移除, 或者在创建脚本的应用程序不再存在时移除。如果您希望保留缓冲区的内容作为 QVD 或 CSV 文件, 则应使用 **Store** 语句。

### 执行以下操作:

1. 创建一个新应用程序并为其指定一个名称。
2. 在**数据加载编辑器**中新增脚本段。
3. 在右侧菜单的 **AttachedFiles** 下, 单击**选择数据**。
4. 上传然后选择 *Cutlery.xlsx*。
5. 在**选择数据**自窗口中, 单击**插入脚本**。
6. 注释掉 `load` 语句中的字段, 并将 `load` 语句更改为以下内容:

```
Buffer LOAD *
```

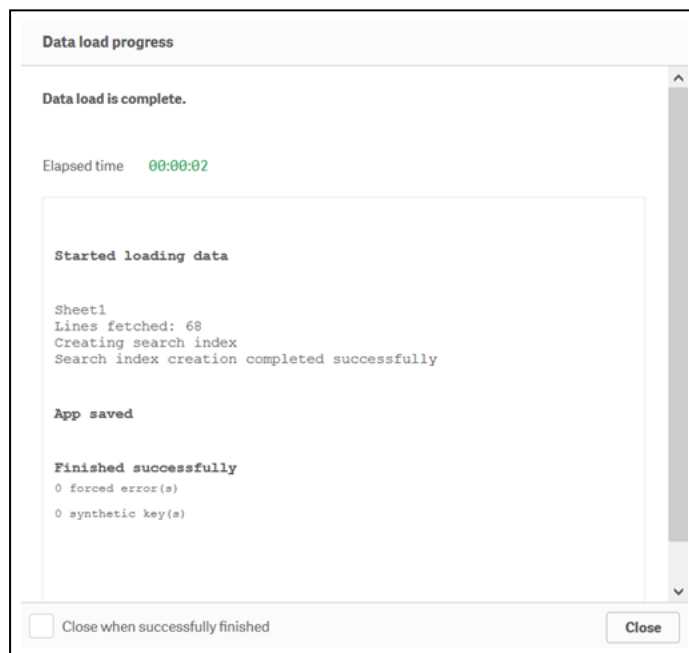
您的脚本应如下所示:

```
Buffer LOAD * //      "date", //      item, //      quantity FROM  
[lib://AttachedFiles/Cutlery.xlsx] (ooxml, embedded labels, table is Sheet1);
```

7. 单击**加载数据**。

第一次加载数据时, 将从 *Cutlery.xlsx* 进行加载。

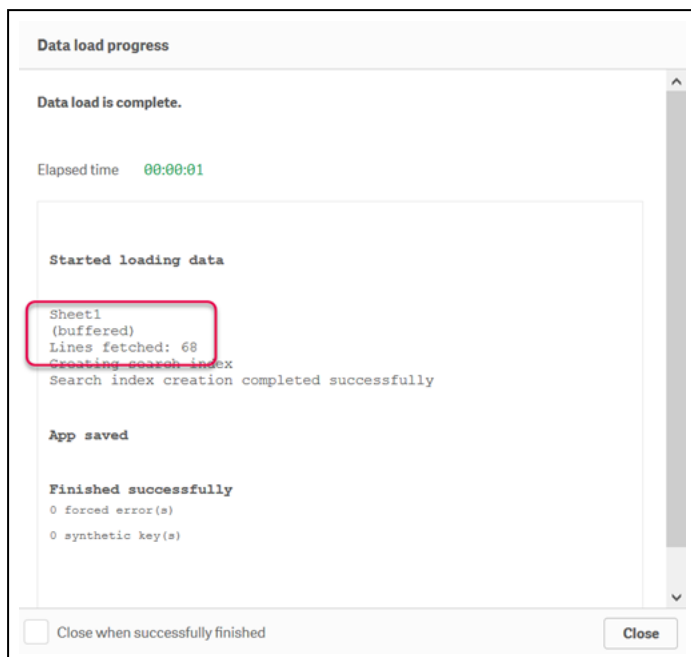
数据加载进度窗口



`Buffer` 语句还创建 QVD 文件并将其存储在 Qlik Sense 中。在 Qlik Sense Enterprise on Windows 部署中, 它存储在 Qlik Sense 服务器上的目录中。

8. 再次单击**加载数据**。
9. 这次是从第一次加载数据时由 `Buffer` 语句创建的 QVD 文件中加载数据。

数据加载进度窗口



## 5.3 谢谢！

现在, 您已完成本教程的学习, 希望您已获得有关 Qlik Sense 的脚本编译的更多知识。请访问我们的网站, 了解有关提供进一步培训的更多信息。