Loading and Modeling Data

Qlik® Sense

1.1

Copyright © 1993-2015 QlikTech International AB. All rights reserved.

Copyright © 1993-2015 QlikTech International AB. All rights reserved.

Qlik®, QlikTech®, Qlik® Sense, QlikView®, Sense™ and the Qlik logo are trademarks which have been registered in multiple countries or otherwise used as trademarks by QlikTech International AB. Other trademarks referenced herein are the trademarks of their respective owners.

Contents

1 About this document

When you have created a Qlik Sense app, the first step is to load the app with data to explore and analyze. This document describes how to use the data load editor to create your data load script and how to view the resulting data model in the data model viewer. You will also learn how to manage security using the section access function, and how large data sets can be accesses using Direct Discovery.

Make sure to see the Concepts in Qlik Sense guide to learn more about the fundamental concepts related to the different topics presented.

Additionally, an introduction and best practices for data modeling in Qlik Sense are presented.

For detailed reference regarding script functions and chart functions, see the Qlik Sense online help.

This document is derived from the online help for Qlik Sense. It is intended for those who want to read parts of the help offline or print pages easily, and does not include any additional information compared with the online help.

Please use the online help or the other documents to learn more.

The following documents are available:

- Concepts in Qlik Sense
- Working with Apps
- Creating Visualizations
- Discovering and Analyzing
- Data Storytelling
- Publishing Apps, Sheets and Stories
- Script Syntax and Chart Functions Guide
- Qlik Sense Desktop

You find these documents and much more at help.qlik.com.

2 Quick data load

Quick data load is available only in Qlik Sense Desktop.

You can add another data file, such as a Microsoft Excel spreadsheet or a comma delimited text file into your app quickly, by clicking **Quick data load** in the menu.

See: *How to prepare Excel files for loading with Qlik Sense (page 21)*

You can also drop a data file on the Qlik Sense Desktop window to start a quick data load.

Do the following:

1. Select how to load the data file.
   Add data - this will add another data file to the data you have already loaded.
   Replace data - this will remove all data you have loaded previously in the app.

   *If you select **Replace data**, this can affect visualizations you have created. If the new data does not contain fields with the same names, the visualizations may need to be updated.*

2. Select which data file to load.
   After selecting a file, the **Select data from** dialog is opened.
3. Select tables and fields to load.
4. When you are done selecting fields, click **Load data**.
   The **Select data from** dialog is closed, and the data is imported. The data load progress dialog provides the results of the data load.
   If data was loaded with warnings, see the Troubleshooting section below for more information.

If the app did not contain a sheet, a new sheet is created, which you can start editing to add data visualizations.

2.1 Troubleshooting

If you have loaded several files, it is possible that you receive a warning after loading the data. The two most common warnings are:

Synthetic keys	If two tables contain more than one common field, Qlik Sense creates a synthetic key to resolve the linking.
Circular references	If you have loaded more than two tables, the common fields may cause a loop in the data structure.

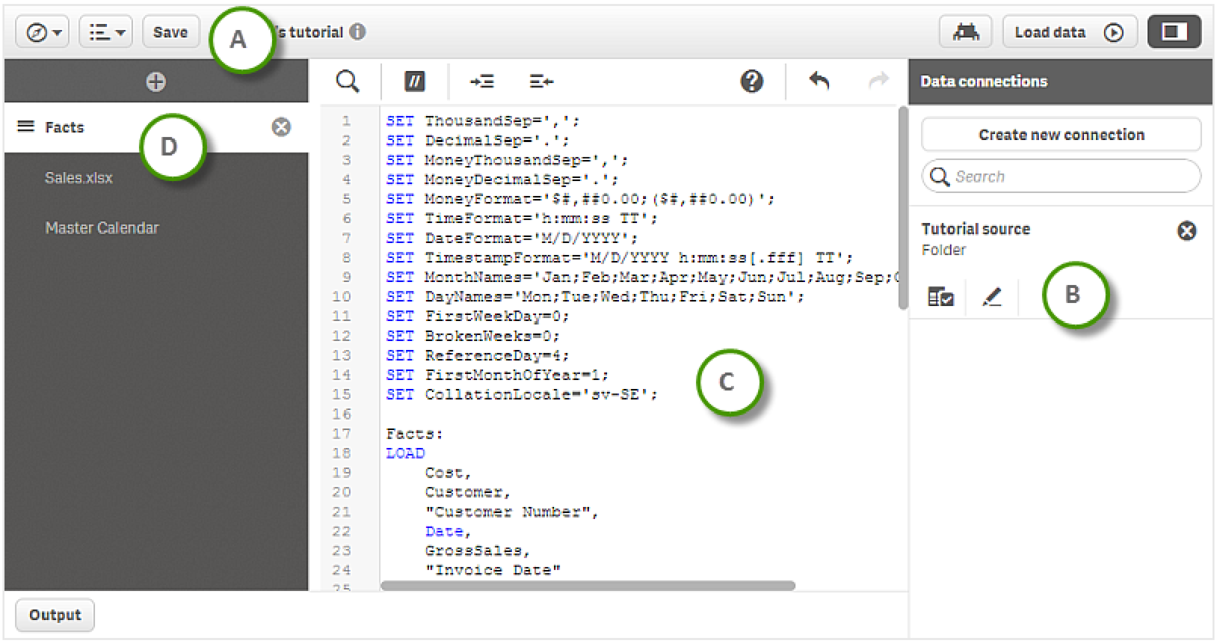
Both issues can be resolved in the data load editor.

3 Using the data load editor

This section describes how to use the data load editor to create or edit a data load script that can be used to load your data model into the app.

The data load script connects an app to a data source and loads data from the data source into the app. When you have loaded the data it is available to the app for analysis. When you want to create, edit and run a data load script you use the data load editor.

A script can be typed manually, or generated automatically. Complex script statements must, at least partially, be entered manually.



```
SET ThousandSep=',';
SET DecimalSep='.';
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='$#,##0.00;($#,##0.00)';
SET TimeFormat='h:mm:ss TT';
SET DateFormat='M/D/YYYY';
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET FirstWeekDay=0;
SET BrokenWeeks=0;
SET ReferenceDay=4;
SET FirstMonthOfYear=1;
SET CollationLocale='sv-SE';

Facts:
LOAD
    Cost,
    Customer,
    "Customer Number",
    Date,
    GrossSales,
    "Invoice Date"
```

A	Toolbar with the most frequently used commands for the data load editor: navigation menu, global menu, **Save**, (debug) and **Load data** . The toolbar also displays the save and data load status of the app.	*Data load editor toolbars (page 44)*
B	Under **Data connections** you can save shortcuts to the data sources (databases or remote files) you commonly use. This is also where you initiate selection of which data to load.	*Connect to data sources (page 12)*

C	You can write and edit the script code in the text editor. Each script line is numbered and the script is color coded by syntax components. The text editor toolbar contains commands for **Search and replace**, **Help mode**, **Undo**, and **Redo**. The initial script already contains some pre-defined regional variable settings, for example `SET ThousandSep=`, that you generally do not need to edit.	*Edit the data load script (page 36)*
D	Divide your script into sections to make it easier to read and maintain. The sections are executed from top to bottom.	*Organizing the script code (page 38)*
E	**Output** displays all messages that are generated during script execution.	

3.1 Quick start

If you want to load a file or tables from a database, you need to complete the following steps in **Data connections**:

1. **Create new connection** linking to the data source (if the data connection does not already exist).
2. Select data from the connection.

When you have completed the select dialog with **Insert script**, you can select **Load data** to load the data model into your app.

3.2 Opening the data load editor

You can open the data load editor from the app overview, sheet view or the data model viewer.

Do the following:

- Click in the toolbar and select **Data load editor**.

The data load editor opens.

3.3 Connect to data sources

Data connections in the data load editor provide a way to save shortcuts to the data sources you commonly use: databases, local files or remote files. **Data connections** lists the connections you have saved in alphabetical order. You can use the search/filter box to narrow the list down to connections with a certain name or type.

The following types of connections exist:

- Standard connectors:
  - **ODBC** database connections.
  - **OLE DB** database connections.

  - **Folder** connections that define a path for local or network file folders.
  - **Web file** connections used to select data from files located on a web URL.
- Custom connectors:
  With custom developed connectors you can connect to data sources that are not directly supported by Qlik Sense. Custom connectors are developed using the QVX SDK or supplied by Qlik or third-party developers. In a standard Qlik Sense installation you will not have any custom connectors available.

You can only see data connections that you own, or have been given access rights to read or update. Please contact your Qlik Sense system administrator to acquire access if required.

Creating a new data connection

Do the following:

1. Click **Create new connection**.
2. Select the type of data source you want to create from the drop-down list.
   The settings dialog, specific for the type of data source you selected, opens.
3. Enter the data source settings and click **Save** to create the data connection.

The data connection is now created with you as the default owner. If you want other users to be able to use the connection in a server installation, you need to edit the access rights of the connection in Qlik Management Console.

The connection name will be appended with your user name and domain to ensure that it is unique.

*If **Create new connection** is not displayed, it means you do not have access rights to add data connections. Please contact your Qlik Sense system administrator to acquire access if required.*

Deleting a data connection

Do the following:

1. Click ⊗ on the data connection you want to delete.
2. Confirm that you want to delete the connection.

The data connection is now deleted.

If ⊗ is not displayed, it means you do not have access rights to delete the data connection. Please contact your Qlik Sense system administrator to acquire access if required.

Editing a data connection

Do the following:

1. Click on the data connection you want to edit.
2. Edit the data connection details. Connection details are specific to the type of connection.
3. Click **Save**.

The data connection is now updated.

If you edit the name of a data connection, you also need to edit all existing references (lib://) to the connection in the script, if you want to continue referring to the same connection.

If is not displayed, it means you do not have access rights to update the data connection. Please contact your Qlik Sense system administrator if required.

Inserting a connect string

Connect strings are required for **ODBC**, **OLE DB** and custom connections.

Do the following:

- Click on the connection for which you want to insert a connect string.

A connect string for the selected data connection is inserted at the current position in the data load editor.

You can also insert a connect string by dragging a data connection and dropping it on the position in the script where you want to insert it.

Selecting data from a data connection

If you want to select data to load in your app, do the following:

1. **Create new connection** linking to the data source (if the data connection does not already exist).
2. Select data from the connection.

Referring to a data connection in the script

You can use the data connection to refer to data sources in statements and functions in the script, typically where you want to refer to a file name with a path.

The syntax for referring to a file is *'lib://(connection_name)/(file_name_including_path)'*

Example 1: Loading a file from a folder data connection

This example loads the file *orders.csv* from the location defined in the MyData data connection.

```
LOAD * FROM 'lib://MyData/orders.csv';
```

Example 2: Loading a file from a sub-folder

This example loads the file *Customers/cust.txt* from the DataSource data connection folder. Customers is a sub-folder in the location defined in the MyData data connection.

```
LOAD * FROM 'lib://DataSource/Customers/cust.txt';
```

Example 3: Loading from a web file

This example loads a table from the PublicData web file data connection, which contains the link to the actual URL.

```
LOAD * FROM 'lib://PublicData' (html, table is @1);
```

Example 4: Loading from a database

This example loads the table Sales_data from the MyDataSource database connection.

```
LIB CONNECT TO 'MyDataSource';
LOAD *;
SQL SELECT * FROM `Sales_data`;
```

Where is the data connection stored?

Connections are stored using the Qlik Sense Repository Service.You can manage data connections with the Qlik Management Console in a Qlik Sense server deployment. The Qlik Management Console allows you to delete data connections, set access rights and perform other system administration tasks.

In Qlik Sense Desktop, all connections are saved in the app without encryption.This includes possible details about user name, password, and file path that you have entered when creating the connection. This means that all these details may be available in plain text if you share the app with another user. You need to consider this while designing an app for sharing.

ODBC data connections

You can create a data connection to select data from an ODBC data source that has already been created and configured in the **ODBC Data Source Administrator** dialog in Windows Control Panel.

Creating a new ODBC data connection

Do the following:

1. Click **Create new connection** and select **ODBC**.
   The **Create new connection (ODBC)** dialog opens.
2. Select which data source to use from the list of available data sources, either **User DSN** or **System DSN**.
   System DSN connections can be filtered according to **32-bit** or **64-bit**.

For **User DSN** sources you need to specify if a 32-bit driver is used with **Use 32-bit connection**.

3. Add **Username** and **Password** if required by the data source.
4. If you want to use a name different from the default DSN name, edit **Name**.
5. Click **Save**.

The connection is now added to the **Data connections**, and you can connect to, and select data from the connected data source.

Editing an ODBC data connection

Do the following:

1. Click ✎ on the **ODBC** data connection you want to edit.
   The **Edit connection (ODBC)** dialog opens.
2. You can edit the following properties:
   Select which data source to use from the list of available data sources, either **User DSN** or **System DSN**.
   Username
   Password
   Name
3. Click **Save**.

The connection is now updated.

⚠ *The settings of the connection you created will not be automatically updated if the data source settings are changed. This means you need to be careful about storing user names and passwords, especially if you change settings between integrated Windows security and database login in the DSN.*

Best practices when using ODBC data connections

Moving apps with ODBC data connections

If you move an app between Qlik Sense sites/Qlik Sense Desktop installations, data connections are included. If the app contains ODBC data connections, you need to make sure that the related ODBC data sources exist on the new deployment as well. The ODBC data sources need to be named and configured identically, and point to the same databases or files.

Connecting to non-ANSI encoded data files

If you are experiencing problems with non-ANSI encoded data files when using an ODBC data connection, you can try importing the data files with a folder connection, which supports more options for handling character codes.

Security aspects when connecting to file based ODBC data connections

ODBC data connections using file based drivers will expose the path to the connected data file in the connection string. The path can be exposed when the connection is edited, in the data selection dialog, or in certain SQL queries.

If this is a concern, it is recommended to connect to the data file using a folder data connection if it is possible.

OLE DB data connections

You can create a data connection to select data from an OLE DB data source .

Creating a new OLE DB data connection

Do the following:

1. Click **Create new connection** and select **OLE DB**.
2. Select **Provider** from the list of available providers.
3. Type the name of the **Data source** to connect to. This can be a server name, or in some cases, the path to a database file. This depends on which **OLE DB** provider you are using.

   Example:

   If you selected Microsoft Office 12.0 Access Database Engine OLE DB Provider, enter the file name of the Access database file, including the full file path:
   C:\Users\{user}\Documents\Qlik\Sense\Apps\Tutorial source files\Sales.accdb

   > *If connection to the data source fails, a warning message is displayed.*

4. Select which type of credentials to use if required:
   - **Windows integrated security**: With this option you use existing Windows credentials of the user running the Qlik Sense service.
   - **Specific user name and password**: With this option you need to enter **User name** and **Password**.

   If the data source does not require credentials, leave **User name** and **Password** empty.
5. If you want to test the connection, click **Load** and then **Select database...** to use for establishing the data connection.

   > *You are still able to use all other available databases of the data source when selecting data from the data connection.*

6. If you want to use a name different to the default provider name, edit **Name**.

   > *You cannot use the following characters in the connection name: \ / : * ? " ' < > |*

7. Click **Save**.

   > *The **Save** button is only enabled if connection details have been entered correctly, and the automatic connection test was successful.*

The connection is now added to the **Data connections**, and you can connect to, and select data from the connected OLE DB data source if the connection string is correctly entered.

Editing an OLE DB data connection

Do the following:

1. Click ✎ on the **OLE DB** data connection you want to edit.
   The **Edit connection (OLE DB)** dialog opens.
2. You can edit the following properties:
   - **Connection string** (this contains references to the **Provider** and the **Data source**)
   - **Username**
   - **Password**
   - **Name**
3. Click **Save**.

The connection is now updated.

Best practices when using OLE DB data connections

Security aspects when connecting to file based OLE DB data connections

OLE DB data connections using file based drivers will expose the path to the connected data file in the connection string. The path can be exposed when the connection is edited, in the data selection dialog, or in certain SQL queries.

If this is a concern, it is recommended to connect to the data file using a folder data connection if it is possible.

Troubleshooting data selection problems

If you are not able to select data from the data connection, please verify that the connection string is correctly designed, and that you use appropriate credentials to log on.

Folder data connections

You can create a data connection to select data from files in a folder, either on a physical drive or a shared network drive.

On a Qlik Sense server installation, the folder needs to be accessible from the system running the Qlik Sense engine, that is, the user running the Qlik Sense service. If you connect to this system using another computer or touch device, you cannot connect to a folder on your device, unless it can be accessed from the system running the Qlik Sense engine.

Creating a new folder data connection

Do the following:

1. Click **Create new connection** and select **Folder**.
   The **Create new data connection (folder)** dialog opens. When you install Qlik Sense, a working directory is created, named *C:\Users\{user}\Documents\Qlik\Sense\Apps*. This is the default directory selected in the dialog.

   *If the **Folder** option is not available, you do not have access right to add folder connections. Contact your Qlik Sense system administrator.*

2. Enter the **Path** to the folder containing the data files. You can either:
   - Select the folder
   - Type a valid local path (example: *C:\data\MyData\*)
   - Type a UNC path (example: *\\myserver\filedir\*).
3. Enter the **Name** of the data connection you want to create.
4. Click **Save**.

The connection is now added to the **Data connections**, and you can connect to, and select data from files in the connected folder.

Editing a folder data connection

Do the following:

1. Click ✎ on the folder data connection you want to edit.
   The **Edit connection (folder)** dialog opens.

   If ✎ is disabled, you do not have access right to edit folder connections. Contact your Qlik Sense system administrator.

2. You can edit the following properties:
   Path
   Name
3. Click **Save**.

The connection is now updated.

Web file data connections

You can create a data connection to select data from files residing on a web server, accessed by an URL address, typically in HTML or XML format.

Creating a new web file data connection

Do the following:

1. Click **Create new connection** and select **Web file**.
   The **Select web file** dialog opens.
2. Enter the **URL** to the web file.

3. Enter the **Name** of the data connection you want to create.
4. Click **Save**.

The connection is now added to the **Data connections**, and you can connect to, and select data from the web file.

Editing a web file data connection

Do the following:

1. Click ✎ on the web file data connection you want to edit.
   The **Select web file** dialog opens.
2. You can edit the following properties:
   URL
   Name
3. Click **Save**.

The connection is now updated.

Loading data from files

Qlik Sense can read in data from files in a variety of formats:

- Text files, where data in fields is separated by delimiters such as commas, tabs or semicolons (comma-separated variable (CSV) files).
- dif files (Data Interchange Format).
- fix files (fixed record length).
- HTML tables.
- Excel files (except password protected Excel files).
- xml files.
- Qlik native QVD and QVX files.

In most cases, the first line in the file holds the field names.

Files are loaded using a **LOAD** statement in the script. **LOAD** statements can include the full set of script expressions.

To read in data from another Qlik Sense app, you can use a **Binary** statement.

Example:

```
directory c:\databases\common;
LOAD * from TABLE1.CSV (ansi, txt, delimiter is ',', embedded labels);
LOAD fieldx, fieldy from TABLE2.CSV (ansi, txt, delimiter is ',', embedded labels);
```

Loading data from a file in the data load editor

Instead of typing the statements manually in the data load editor, you can use the **Select data** dialog to select data to load. Do the following:

1. Open the data load editor.
2. Create a **Folder** data connection, if you don't have one already. The data connection should point to the directory containing the data file you want to load.
3. Click on the data connection to open the data selection dialog.

Now you can select data from the file and insert the script code required to load the data.

> *You can also use the ODBC interface to load a Microsoft Excel file as data source. In that case you need to create an **ODBC** data connection instead of a **Folder** data connection.*

How to prepare Excel files for loading with Qlik Sense

If you want to load Microsoft Excel files into Qlik Sense, there are many functions you can use to transform and clean your data in the data load script, but it may be more convenient to prepare the source data directly in the Microsoft Excel spreadsheet file. This section provides a few tips to help you prepare your spreadsheet for loading it into Qlik Sense with minimal script coding required.

Use column headings

If you use column headings in Excel, they will automatically be used as field names if you select **Embedded field names** when selecting data in Qlik Sense. It is also recommended that you avoid line breaks in the labels, and put the header as the first line of the sheet.

Formatting your data

It is easier to load an Excel file into Qlik Sense if the content is arranged as raw data in a table. It is preferable to avoid the following:

- Aggregates, such as sums or counts. Aggregates can be defined and calculated in Qlik Sense.
- Duplicate headers.
- Extra information that is not part of the data, such as comments. The best way is to have a column for comments, that you can easily skip when loading the file in Qlik Sense.
- Cross-table data layout. If, for instance, you have one column per month, you should, instead, have a column called “Month” and write the same data in 12 rows, one row per month. Then you can always view it in cross-table format in Qlik Sense.
- Intermediate headers, for example, a line saying “Department A” followed by the lines pertaining to Department A. Instead, you should create a column called “Department” and fill it with the appropriate department names.
- Merged cells. List the cell value in every cell, instead.
- Blank cells where the value is implied by the previous value above. You need to fill in blanks where there is a repeated value, to make every cell contain a data value.

Use named areas

If you only want to read a part of a sheet, you can select an area of columns and rows and define it as a named area in Excel. Qlik Sense can load data from named areas, as well as from sheets.

Typically, you can define the raw data as a named area, and keeping all extra commentary and legends outside the named area. This will make it easier to load the data into Qlik Sense.

Remove password protection

Password protected files are not supported by Qlik Sense.

Loading data from databases

You can load data from commercial database systems into Qlik Sense through the following connectors:

- **Standard connectors** using the Microsoft ODBC interface or OLE DB. To use ODBC, you must install a driver to support your DBMS and you must configure the database as an ODBC data source in the **ODBC Data Source Administrator** in Windows **Control Panel**.
- **Custom connectors**, specifically developed to load data from a DBMS into Qlik Sense.

Loading data from an ODBC database

The easiest way to get started loading data from a database, such as Microsoft Access or any other database that can be accessed through an ODBC data source, is by using the data selection dialog in the data load editor.

To do this, do the following:

1. You need to have an ODBC data source for the database you want to access. This is configured in the **ODBC Data Source Administrator** in Windows**Control Panel**. If you do not have one already, you need to add it and configure it, for example pointing to a Microsoft Access database.
2. Open the data load editor.
3. Create an **ODBC** data connection, pointing to the ODBC connection mentioned in step 1.
4. Click on the data connection to open the data selection dialog.

Now you can select data from the database and insert the script code required to load the data.

ODBC

To access a DBMS (DataBase Management System) via ODBC, it is necessary to have installed an ODBC driver for the DBMS in question. The alternative is to export data from the database into a file that is readable to Qlik Sense.

Normally, some ODBC drivers are installed with the operating system. Additional drivers can be bought from software retailers, found on the Internet or delivered from the (DBMS) manufacturer. Some drivers are redistributed freely.

The ODBC interface described here is the interface on the client computer. If the plan is to use ODBC to access a multi-user relational database on a network server, additional DBMS software that allows a client to access the database on the server might be needed. Contact the DBMS supplier for more information on the software needed.

Adding ODBC drivers

An ODBC driver for your DBMS(DataBase Management System) must be installed for Qlik Sense to be able to access your database. Please refer to the documentation for the DBMS that you are using for further details.

64-bit and 32-bit versions of ODBC configuration

A 64-bit version of the Microsoft Windows operating system includes the following versions of the Microsoft Open DataBase Connectivity (ODBC)Data Source Administrator tool (*Odbcad32.exe):*

- The 32-bit version of the *Odbcad32.exe* file is located in the *%systemdrive%\Windows\SysWOW64* folder.
- The 64-bit version of the *Odbcad32.exe* file is located in the *%systemdrive%\Windows\System32* folder.

Creating ODBC data sources

An ODBC data source must be created for the database you want to access. This can be done during the ODBC installation or at a later stage.

*Before you start creating data sources, a decision must be made whether the data sources should be **User DSN** or **System DSN** (recommended). You can only reach user data sources with the correct user credentials. On a server installation, typically you need to create system data sources to be able to share the data sources with other users.*

Do the following:

1. Open *Odbcad32.exe*.
2. Go to the tab **System DSN** to create a system data source.
3. Click **Add**.

   The **Create New Data Source** dialog appears, showing a list of the ODBC drivers installed.
4. If the correct ODBC driver is listed, select it and click **Finish**.

   A dialog specific to the selected database driver appears.
5. Name the data source and set the necessary parameters.
6. Click **OK**.

OLE DB

Qlik Sense supports the OLE DB(Object Linking and Embedding, Database) interface for connection to external data sources. A great number of external databases can be accessed via OLE DB.

Logic in databases

Several tables from a database application can be included simultaneously in the Qlik Sense logic. When a field exists in more than one table, the tables are logically linked through this key field.

When a value is selected, all values compatible with the selection(s) are displayed as optional. All other values are displayed as excluded.

If values from several fields are selected, a logical AND is assumed.

If several values of the same field are selected, a logical OR is assumed.

In some cases, selections within a field can be set to logical AND.

3.4 Select data to load

You can select which fields to load from files or database tables and which views of the data source you want by using the interactive **Select data** dialog. The dialog shows different selection or transformation options depending on the type of file or database you use as data source. As well as selecting fields, you can also rename fields in the dialog. When you have finished selecting fields, you can insert the generated script code to your script.

You open **Select data** by clicking on a data connection in the data load editor.

Selecting data from a database

In order to select data from a database, you start by clicking on an ODBC or OLE DB data connection in the data load editor. This is where you select the fields to load from the database tables or views of the data source. You can select fields from several databases, tables and views in one session.

Selecting database

Do the following:

1. Select a **Database** from the drop-down list.
2. Select **Owner** of the database.

The list of **Tables** is populated with views and tables available in the selected database.

Selecting tables and views

The table list will include tables, views, synonyms, system tables and aliases from the database.

If you want to select all fields of a table, do the following:

- Mark the checkbox next to the table name.

If you want to select specific fields from a **Table**, do the following:

- Click on the table name (not in the checkbox).
  Fields will be updated with the available table content, and you can continue by selecting fields.

Tables with all fields selected are indicated with a checkmark in the checkbox, while tables with some fields selected are indicated with a blue square in the checkbox and the number of selected fields to the right.

Selecting fields

Fields lists all fields available in the selected **Table**. You can **Filter** for fields in the list by typing part of the field name in the text box.

You can display the fields in either of the following views:

- **Data preview**, which presents data in horizontal column layout and preview data loaded. Primary key fields are indicated by 🔑.
- **Metadata**, which presents data in a vertical list layout with field metadata, such as primary keys and data type, displayed.

Do the following:

- Select the fields you want to include using the checkbox next to each field name.

When you have selected the fields to include from the selected table, you can continue to select fields from other tables in the same database, or pick another database and select tables and fields from that database.

Selections summary provides an overview of the databases you have selected and the number of selected tables and fields.

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields so that they are different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them (in either file) to have identical names to load them as one field.

You cannot rename fields in the data selection wizard at the same time as you filter for fields by searching. You have to erase the search string in the text box first.

It is not possible to rename two fields in the same table so that they have identical names.

Do the following:

- Click on the field header you want to rename, type the new name and press *Enter*.

The field is renamed, and the script preview is updated.

Renaming a field corresponds to using **`as`** *in a field definition in a LOAD statement.*

*If you rename fields in a table, a LOAD statement will be inserted automatically regardless of the **Include LOAD statement** setting.*

Previewing script

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Including LOAD statement

If **Include LOAD statement** is selected, SELECT statements are generated with preceding LOAD statements using the SELECT statements as input.

If you rename fields in a table, a LOAD statement will be inserted automatically regardless of this setting.

Inserting in the script

When you have finished selecting fields and want to generate your **LOAD**/**SELECT** statements in the script, do the following:

- Click **Insert script**.

The data selection window is closed, and the LOAD /SELECT statements are inserted in the script in accordance with your selections.

Selecting data from a delimited table file

You can load data from a delimited table file in formats such as CSV, TXT, TAB, PRN, MEM or SKV, or files with custom delimiter symbols. When you have selected fields, you can insert the script code required to load the fields into the script.

To start selecting data, do the following:

1. Click on a folder data connection in the data load editor.
2. Select file from the drop-down list of available files in the folder and click **Select**.

The selection dialog is displayed with **Fields** updated with preview data.

Setting file options

The preview data is formatted using settings derived from your file, but you may need to adjust the file options to suit your purpose:

File format	Set to **Delimited** or **Fixed record**. When you make a selection, the select data dialog will adapt to the selected file format.

Field names	Set to specify if the table contains **Embedded field names** or **No field names**.
Delimiter	Set the **Delimiter** character used in your table file.
Quoting	Set to specify how to handle quotes: **None** = quote characters are not accepted **Standard** = standard quoting (quotes can be used as first and last characters of a field value) **MSQ** = modern style quoting (allowing multi-line content in fields)
Header size	Set to the number of lines to omit as table header.
Character set	Set to character set used in the table file.
Comment	Data files can contain comments between records, denoted by starting a line with one or more special characters, for example //. Specify one or more characters to denote a comment line. Qlik Sense does not load lines starting with the character(s) specified here.
Ignore EOF	Select **Ignore EOF** if your data contains EOF characters as part of field value.

Preview data is formatted according to the options you have set.

Selecting fields

Fields lists all fields available to select. You can do one of the following:

- Select which fields to include using the checkbox next to each field name.
- Select all fields by using the **Select all fields** checkbox.

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields so that they are different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them (in either file) to have identical names to load them as one field.

Do the following:

- Click on the field header you want to rename, type the new name and press *Enter*.

The field is renamed, and the script preview is updated.

*Renaming a field corresponds to using `as` in a field definition in a **LOAD** statement.*

Previewing the script

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Inserting the script

When you have finished selecting fields and want to generate your **LOAD**/**SELECT** statements in the script, do the following:

- Click **Insert script**.

The **Select data from** window is closed, and the LOAD statements are inserted in the script in accordance with your selections.

Selecting data from a Microsoft Excel file

You can load from data from a selected Microsoft Excel file. When you have selected fields, you can insert the script code required to load the fields into the script.

To start selecting data, do the following:

1. Click on a **Folder** data connector in the data load editor.
2. Select file from the drop-down list of available files in the folder and click **Select**.

Select data from is displayed and the list of **Tables** is populated with the sheets and named areas available in the selected Microsoft Excel file.

*You can also use a Microsoft Excel file as data source using the ODBC interface. In that case you need to use an **ODBC** data connection instead of a **Folder** data connection.*

Selecting tables

The **Tables** list includes all sheets and named areas of the selected Microsoft Excel file.

If you want to select all the fields in a table, do the following:

- Check the box next to the table name.

If you want to select specific fields from a table, do the following:

- Click the table name (not in the checkbox).
  Fields is updated with the available table content, and you can select the fields you require.

Tables that have all columns selected are indicated with a checkmark in the checkbox, while tables with some fields selected are indicated with a square in the checkbox and the number of selected fields to the right of the table name.

Selecting fields

Fields lists all fields available in the selected table. You may need to specify the following settings:

Field names:

- **Embedded field names**, if field names (headers) are stored in the first line of data.
- **No field names**, if there are no field names.

Header size:

- Set to the number of lines to ignore when loading data.

Do the following:

- Select the fields to include using the checkbox next to each field name.

When you have selected the fields to include from the selected table, you can continue to select fields from other tables in the same file.

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields so that they are different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them (in either file) to have identical names to load them as one field.

Do the following:

- Click on the field header you want to rename, type the new name and press *Enter*.

The field is renamed, and the script preview is updated if the field is selected.

Renaming a field corresponds to using ***`as`*** *in a field definition in a* ***`LOAD`*** *statement.*

Previewing the script

The statements that will be inserted are displayed in the script preview , which you can choose to hide by clicking **Preview script**.

Inserting the script

When you have finished selecting fields and want to generate your **LOAD**/**SELECT** statements in the script, do the following:

- Click **Insert script**.

The **Select data from** window is closed, and the **LOAD** statements are inserted in the script in accordance with your selections.

Selecting data from a fixed record table file

You can select data from fixed record data files where each record (row of data) contains a number of columns with a fixed field size, usually padded with spaces or tab characters.

Example: Fixed record table file

```
Item       Id          Price
Watch      001         2.75
Ball       002         3.25
```

To select data from a fixed record file, do the following:

1. Click on a **Folder** data connection in the data load editor.
2. Select the file from the drop-down list of available files in the folder and click **Select**.
   Select data from is displayed with **Fields** updated with preview data.
3. Set **File format** to **Fixed record**.
   The preview data will adapt to the fixed record format.

Setting file options

The preview data is formatted using settings derived from your file, but you may need to adjust the file options to suit your purpose:

Field names	Set to specify if the table contains **Embedded field names** or **No field names**.
Header size	Set **Header size** to the number of lines to omit as table header.
Character set	Set to the character set used in the table file.
Tab size	Set to the number of spaces that one Tab character represents in the table file.
Record line size	Set to the number of lines that one record spans in the table file. Default is 1.

Preview data is formatted according to the options you have set.

Setting field break positions

You can set the field break positions in two different ways:

- Enter the field break positions separated by commas manually in **Field break positions**. Each position marks the start of a field.

  Example: 1,12,24

- Enable **Field breaks** to edit field break positions interactively in the field data preview. **Field break positions** is updated with the selected positions. You can:
  - Click in the field data preview to insert a field break.
  - Click on a field break to delete it.
  - Drag a field break to move it.

Selecting fields

Fields lists all fields available to select. You can do one of the following:

- Select the fields to include using the checkbox next to each field name.
- **Select all fields**

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields so that they are different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them (in either file) to have identical names to load them as one field.

Do the following:

- Click on the field header you want to rename, type the new name and press *Enter*.

The field is renamed, and the script preview is updated.

Renaming a field corresponds to using `as` *in a field definition in a* `LOADS` *statement.*

Previewing the script

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Inserting the script

When you have finished selecting fields and want to generate your **LOAD**/**SELECT** statements in the script, do the following:

- Click **Insert script**.

The **Select data from** window is closed, and the LOAD statements are inserted in the script in accordance with your selections.

Selecting data from a QVD or QVX file

You can load data from QVD files created by Qlik Sense and QVX files created by a custom connector. When you have selected which fields to load, you can insert the script code required to load the fields into the script.

To select data, do the following:

1. Click on a **Folder** data connection in the data load editor.
2. Select a QVD or QVX file from the drop-down list of available files in the folder and click **Select**.

Select data from is displayed with **Fields** updated with preview data.

Selecting fields

Fields lists all fields available to select. You can either:

- Select the fields to include by marking the checkbox next to each field name.
- Click **Select all fields**

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields so that they are different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them (in either file) to have identical names to load them as one field.

Do the following:

- Click on the field header you want to rename, type the new name and press *Enter*.

The field is renamed, and the script preview is updated.

*Renaming a field corresponds to using `as` in a field definition in a **LOAD** statement.*

Previewing the script

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Inserting the script

When you have finished selecting fields and want to generate your **LOAD** statements in the script, do the following:

- Click **Insert script**.

The **Select data from** window is closed, and the **LOAD** statements are inserted in the script in accordance with your selections.

Selecting data from an HTML file

You can load data from tables in HTML files, that is, tables that are encoded with the <TABLE> element.

The **Select data from** dialog opens when you click on a **Folder** or **Web file** connection in the data load editor. This is where you select the fields to load from the tables of the HTML file. You can select fields from several tables.

Selecting tables

Tables lists all tables in the HTML file.

If you want to select all fields of a table, do the following:

- Mark the checkbox next to the table name.

If you want to select specific fields of a table, do the following:

- Click on the table name (not in the checkbox).
  Fields will be updated with the available table content, and you can continue selecting fields.

Tables with all fields selected are indicated with a checkmark in the checkbox, while tables with some fields selected are indicated with a square in the checkbox and the number of selected fields to the right.

Selecting fields

Fields lists all fields available in the selected **Table**. You can **Filter** for fields in the list by typing part of the field name in the text box.

You cannot rename fields in the data selection wizard at the same time as you filter for fields by searching. You have to erase the search string in the text box first.

It is not possible to rename two fields in the same table so that they have identical names.

You may need to specify the following settings:

Field names:

- **Embedded field names**, if field names (field headers) are stored in the first line of data.
- **No field names**, if there are no field names.

Character set:

Do the following:

- Select the fields to include by marking the checkbox next to each field name.

When you have selected the fields to include from the selected table, you can continue to select fields from other tables in the same file.

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields so that they are different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them (in either file) to have identical names to load them as one field.

You cannot rename fields in the data selection wizard at the same time as you filter for fields by searching. You have to erase the search string in the text box first.

It is not possible to rename two fields in the same table so that they have identical names.

Do the following:

- Click on the field header you want to rename, type the new name and press *Enter*.

The field is renamed, and the script preview is updated.

*Renaming a field corresponds to using `as` in a field definition in a **LOAD** statement.*

Previewing the script

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Inserting the script

When you have finished selecting fields and want to generate your **LOAD** statements in the script, do the following:

- Click **Insert script**.

The **Select data from** window is closed, and the **LOAD** statements are inserted in the script in accordance with your selections.

Selecting data from an XML file

You can load data that is stored in XML format.

Do the following:

1. Click on a **Folder** or **Web file** connection in the data load editor.
2. Select the XML file you want to load data from.

The **Select data from** dialog is displayed with **Fields** updated with preview data. This is where you select the fields to load from the tables of the file. You can select fields from several tables.

Selecting tables and fields

Tables lists all tables available to select. When you select a table, **Fields** is updated with the available fields. By default, all data is selected.

- If you want to exclude a table or a field, mark the checkbox next to the table or field name.

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields so that they are different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them (in either file) to have identical names to load them as one field.

Do the following:

- Click on the field header you want to rename, type the new name and press *Enter*.

The field is renamed, and the script preview is updated when you select something else.

Renaming a field corresponds to using `as` *in a field definition in a* ***LOAD*** *statement.*

Previewing the script

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Inserting the script

When you have finished selecting fields and want to generate your **LOAD** statements in the script, do the following:

- Click **Insert script**.

The **Select data from** window is closed, and the **LOAD** statements are inserted in the script in accordance with your selections.

Selecting data from a KML file

You can load geographical data that is stored in KML format to use when creating a map visualization.

Do the following:

1. Click on a **Folder** or **Web file** connection in the data load editor.
2. Select the KML file you want to load data from.

The **Select data from** dialog is displayed with **Fields** updated with preview data. This is where you select the fields to load from the tables of the file. You can select fields from several tables.

Selecting tables and fields

Tables lists all tables available to select. When you select a table, **Fields** is updated with the available fields. By default, all data is selected.

- If you want to exclude a table or a field, mark the checkbox next to the table or field name.

Renaming fields

You can rename fields. This is particularly useful in the following two cases:

- If you load two files containing a field with the same name, they will by default be linked and treated as one field in Qlik Sense. If you want to load them as separate fields, rename the fields to be different.
- If you load two files containing a field that should be read as one field, but has different names in the respective files, you can rename them to have identical names to load them as one field.

Do the following:

- Click on the field header you want to rename, type the new name and press Enter.

The field is renamed, and the script preview is updated when you select something else.

*Renaming a field corresponds to using `as` in a field definition in a **Load** statement.*

Previewing the script

The statements that will be inserted are displayed in the script preview, which you can choose to hide by clicking **Preview script**.

Inserting the script

When you have finished selecting fields and want to generate your **LOAD** statements in the script, do the following:

- Click **Insert script**.

The **Select data from** window is closed, and the **LOAD** statements are inserted in the script in accordance with your selections.

3.5 Edit the data load script

You write the script in the text editor of the data load editor. Here you can make manual changes to the **LOAD**/**SELECT** statements you have generated with the data selection pop-ups, and type new script.

The script, which must be written using the Qlik Sense script syntax, is color coded to make it easy to distinguish the different elements. Comments are highlighted in green, whereas Qlik Sense syntax keywords are highlighted in blue. Each script line is numbered.

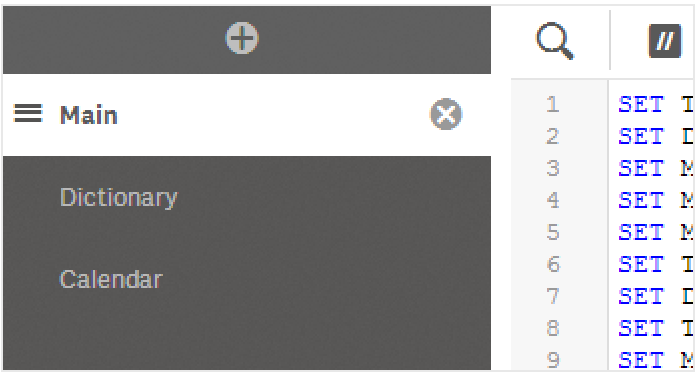
There are a number of functions available in the editor to assist you in developing the load script:

Detailed syntax help	There are two ways to access detailed syntax help for a Qlik Sense syntax keyword: • Click ? in the toolbar to enter syntax help mode. In syntax help mode you can click on a syntax keyword (marked in blue and underlined) to access syntax help. *You cannot edit the script in syntax help mode.* • Place the cursor inside or at the end of the keyword and press Ctrl+H.
Auto-complete	If you start to type a Qlik Sense script keyword, you get an auto-complete list of matching keywords to select from. The list is narrowed down as you continue to type. For example, type *month*. Do the following: • Select a keyword in the list by clicking on it or by pressing Enter.
Tooltips	When you type an open parenthesis after a Qlik Sense script function, a tooltip displays the syntax of the function, including parameters, return value types and additional statements.
Prepared test script	You can insert a prepared test script that will load a set of inline data fields. You can use this to quickly create a data set for test purposes. Do the following: • Press *Ctrl+00* The test script code is inserted into the script.

Indenting code	You can indent the code to increase readability. Do the following: 1. Select one or several lines to change indentation. 2. Click to indent the text (increase indentation) or, click to outdent the text (decrease indentation). *You can also use keyboard shortcuts:* *Tab (indent)* *Shift+Tab (outdent)*
Search and replace	You can search and replace text throughout script sections. Click in the toolbar to open the **Search and replace** panel.
Selecting all code	You can select all code in the current script section. Do the following: • Press *Ctrl+A* All script code in the current section is selected.

Organizing the script code

You can divide your script into sections to organize the structure. The script is executed in the order of the sections from top to bottom. The script must contain at least one section.



Creating a new script section

You can insert new script sections to organize your code.

Do the following:

- Click .

The new section is inserted after the currently selected section.

Deleting a script section

You can delete a script section, including all code in the section.

Deleting a script section cannot be undone.

Do the following:

- Click ⊗ next to the section tab to delete it. You need to confirm the deletion.
  The section is now deleted.

Renaming a script section

You can rename a script section.

Do the following:

1. Click on the section name and type to edit the name.
2. Press Enter or click outside the section when you are finished.

The section is now renamed.

Rearranging script sections

You can rearrange the order of sections to change the script execution order.

Do the following:

1. Select the section you want to move.
2. Put the cursor on the ≡ drag bars and drag the section to rearrange the order.

The sections have now been rearranged.

Commenting in the script

You can insert comments and remarks in the script code, or deactivate parts of the script code by using comment marks. All text on a line that follows to the right of // (two forward slashes) will be considered a comment and will not be executed when the script is run.

The data load editor toolbar contains a shortcut to commenting or uncommenting code. The function works as a toggle, if the selected code is not commented it will be commented, and vice versa.

Commenting

Do the following:

1. Select one or more lines of code that are not commented out, or place the cursor at the beginning of a line.
2. Click //, or press Ctrl+K.

The selected code is now commented out.

Uncommenting

Do the following:

1. Select one or more lines of code that are commented out, or place the cursor at the beginning of a commented line.
2. Click //, or press Ctrl+K.

The selected code is now not commented out.

There are more ways to insert comments in the script code:

- *Using the* ***Rem*** *statement.*
- *Enclosing a section of code with /* and */.*

Example:

```
Rem This is a comment ;

/* This is a comment
   that spans two lines */

// This is a comment as well
```

Searching in the load script

You can search and replace text throughout script sections.

Searching

Open the data load editor. Do the following:

1. Click 🔍 in the toolbar.
   The search drop-down dialog is displayed.
2. In the search box, type the text you want to find. You need to type at least two characters.
   The search results are highlighted in the current section of the script code. Also, the number of text instances found is indicated next to the section label.
3. You can browse through the results by clicking ‹ and › .

Click 🔍 in the toolbar to close the search dialog.

Also, you can select ***Search in all sections*** *to search in all script sections. The number of text instances found is indicated next to each section label. The search function is not case sensitive.*

Replacing

Do the following:

1. Click 🔍 in the toolbar.
   The search drop-down dialog is displayed.
2. Type the text you want to find in the search box.
3. Type the replacement text in the replace box and click **Replace**.
4. Click > to find next instance of search text and do one of the following:
   - Click **Replace** to replace text.
   - Click > to find next.

Click 🔍 in the toolbar to close the search dialog.

*You can also click **Replace all in section** to replace all instances of the search text in the current script section. The replace function is case sensitive, and replaced text will have the case given in the replace field. A message is shown with information about how many instances that were replaced.*

Saving the load script

When you save the script the entire app is saved, but data is not automatically reloaded.

Do the following:

- Click **Save** in the data load editor toolbar to save the script.

The script is now saved, but the app will still contain old data from the previous reload, which is indicated in the toolbar. If you want to update the app with new data, click **Load data** ▶ in the data load editor toolbar.

When you save the script, it is automatically checked for syntax errors. Syntax errors are highlighted in the code, and all script sections containing syntax errors are indicated with ⓘ next to the section label.

The script is automatically saved to the app when data is loaded.

3.6 Debug the data load script

You can use the debugging utilities in the data load editor to step through the execution of your script by using breakpoints, which will enable you to inspect variable values and output from the script execution. You can select if you want to view any or all of **Output**, **Variables** and **Breakpoints**.

To show the debug panel, do the following:

- Click 🐞 in the data load editor toolbar.
  The debug panel is opened at the bottom of the data load editor.

You can't create connections, edit connections, select data, save the script or load data while you are running in debug mode, that is, from you have started debug execution until the script is executed or execution has been ended

Debug toolbar

The data load editor debug panel contains a toolbar with the following options to control the debug execution:

Limited load	Enable this to limit how many rows of data to load from each data source. This is useful to reduce execution time if your data sources are large. Enter the number of rows you want to load. *This only applies to physical data sources. Automatically generated and Inline loads will not be limited, for example.*
▶	Start or continue execution in debug mode until the next breakpoint is reached.
⏯	Step to the next line of code.
■	End execution here. If you end before all code is executed, the resulting data model will only contain data up to the line of code where execution ended.

Output

Output displays all messages that are generated during debug execution. You can select to lock the output from scrolling when new messages are displayed by clicking 🔓 .

Additionally, the output menu (☰) contains the following options:

Clear	Click this to delete all output messages.
Select all text	Click this to select all output messages.
Scroll to bottom	Click this to scroll to the last output message.

Variables

Variables lists all reserved variables, system variables and variables defined in the script, and displays the current values during script execution.

Setting a variable as favorite

If you want to inspect specific variables during execution, you can set them as favorites. Favorite variables are displayed at the top of the variable list, marked by a yellow star. To set a variable as favorite, do the following:

- Click on the ★ next to a variable.
  The ★ is now yellow, and the variable moved to the top of the variable list.

Filtering variables

You can apply a filter to show only a selected type of variables by using the following options in the variables menu (☰):

Show all variables	Click this to show all types of variables.
Show system variables	Click this to show system variables. System variables are defined by Qlik Sense, but you can change the variable value in the script.
Show reserved variables	Click this to show reserved variables. Reserved variables are defined by Qlik Sense and the value cannot be changed.
Show user defined variables	Click this to show user defined variables. User defined variables are variables that you have defined in the script.

Breakpoints

You can add breakpoints to your script to be able to halt debug execution at certain lines of code and inspect variable values and output messages at this point. When you have reached a breakpoint, you can choose to stop execution, continue until the next breakpoint is reached, or step to the next line of code. All breakpoints in the scripts are listed, with a reference to section and line number.

Adding a breakpoint

To add a breakpoint at a line of code , do one of the following:

- In the script, click in the area directly to the right of the line number where you want to add a breakpoint.
  A ⊖ next to the line number will indicate that there is a breakpoint at this line.

You can add breakpoints even when the debug panel is closed.

Deleting breakpoints

You can delete a breakpoint by doing either of the following:

- In the script, click on a ⊖ next to the line number.
- In the list of breakpoints, click ⊗ next to a breakpoint.

You can also click ☰ and select **Delete all** to delete all breakpoints from the script.

Enabling and disabling breakpoints

When you create a breakpoint it is enabled by default, which is indicated by ✓ next to the breakpoint in the list of breakpoints. You can enable and disable individual breakpoints by selecting and deselecting them in the list of breakpoints.

You also have the following options in the breakpoints menu (☰):

- **Enable all**
- **Disable all**

3.7 Run the script to load data

Click **Load data** in the toolbar to run the script and reload data in the app. The app is automatically saved before loading the data.

The **Data load progress** dialog is displayed, and you can **Abort** the load. When the data load has completed, the dialog is updated with status (**Finished successfully** or **Data load failed**) and a summary of possible errors and warnings, such as for synthetic keys. The summary is also displayed in **Output**, if you want to view it after the dialog is closed.

*If you want the **Data load progress** dialog to always close automatically after a successful execution, select **Close when successfully finished** .*

3.8 Data load editor toolbars

The toolbars allow you to perform global actions on your data load script, such as undo/redo, debug, and search/replace. You can also click **Load data** to reload the data in your app.

Main toolbar

⊘	Navigation menu with the following options: **App overview** **Data model viewer** **Open hub**
☰	Menu with the following options: **Quick data load** - load another data file into the app - only available in Qlik Sense Desktop. **Help** **About**

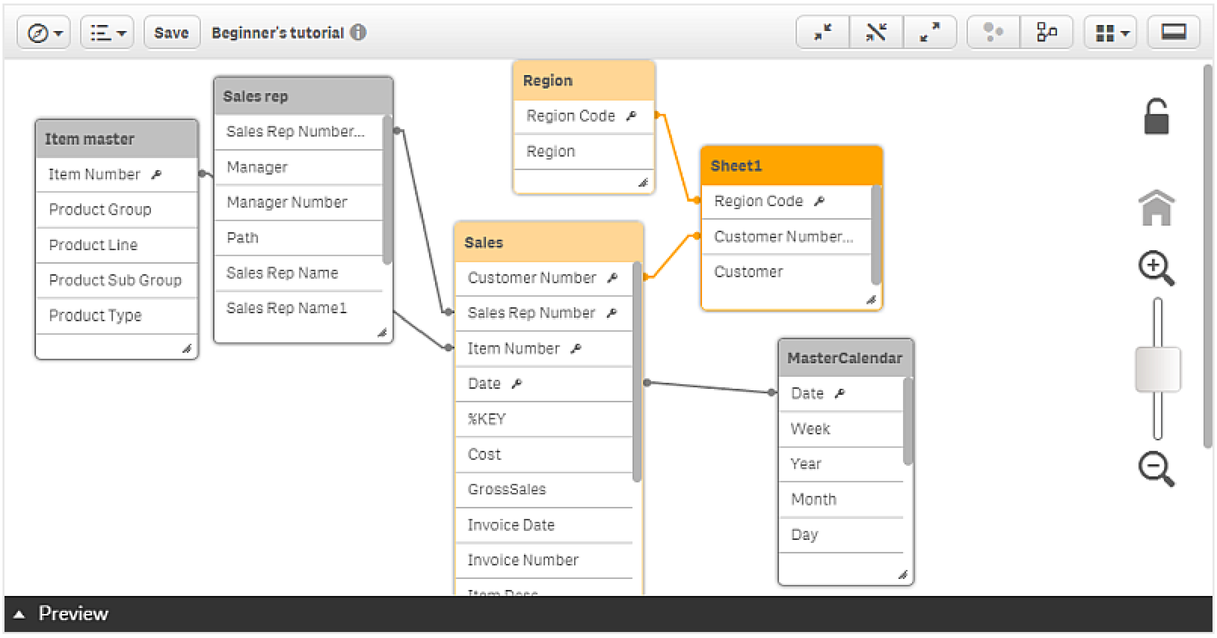
🐞	Debug the script
Load data ▶	Execute the script and reload data. The app is automatically saved before reloading.
◨	Toggle **Data connections** view.

Editor toolbar

🔍	Search and replace text in the script.
//	Comment/uncomment
⇥	Indent
⇤	Outdent
?	Activate syntax help mode. In help mode you can click on a syntax keyword (marked in blue) in the editor to access detailed syntax help. *It is not possible to edit the script in help mode.*
↶	Undo the latest change in the current section (multiple step undo is possible). This is equivalent to pressing Ctrl+Z.
↷	Redo the latest **Undo** in the current section. This is equivalent to pressing Ctrl+Y.

4 Viewing the data model

The data model viewer provides you with an overview of the data structure of the app. You can preview data in the tables and fields in the data model viewer. You can also create dimensions and measures on-the-fly.



In the data model viewer, each data table is represented by a box, with the table name as title and with all fields in the table listed. Table associations are shown with lines, with a dotted line indicating a circular reference. When you select a table or a field, the highlighting of associations instantly gives you a picture of how fields and tables are related.

You can change the zoom level, by clicking , or using the slider. Click to restore zoom level to 1:1.

4.1 Views

You can select to view either:

- **Internal table view** - the Qlik Sense data model including synthetic fields
- **Source table view** - the data model of the source data tables

4.2 Moving and resizing tables in the data model viewer

Moving tables

You can move the tables by dragging them on the canvas. The table positions will be saved when the app is saved.

You can lock the table layout (positions and sizes), by clicking in the right part of the canvas. To unlock the table layout, click .

It is also possible to arrange the layout automatically using the options under in the toolbar:

- **Grid layout** - to arrange the tables in a grid.
- **Auto layout** - to arrange the tables to fit in the window.
- **Restore layout** - to revert to the latest saved layout state.

Resizing tables

You can adjust the display size of a table with the red arrow in the bottom right corner of a table. Display size will not be saved when the app is saved.

You can also use the automatic display size options in the toolbar:

Collapse all - to minimize all tables to show the table name only.

Show linked fields - to reduce the size of all tables to show the table name and all fields with associations to other tables.

Expand all - to maximize all tables to show all fields in the table.

4.3 Preview of tables and fields in the data model viewer

In the data model viewer, you can get a preview of any data table in a panel at the bottom of the screen. In the preview, you can quickly inspect the contents of a table or field. You can also add dimensions and measures quickly if you select a field.

A preview of a table shows the 10 first table fields and their values, and a preview of a field shows an individual field with its 10 first values. Additionally, metadata such as information density and tags are displayed in the preview panel.

You can show and hide the preview panel in two ways:

- Click in the data model viewer toolbar
- Click the **Preview** header

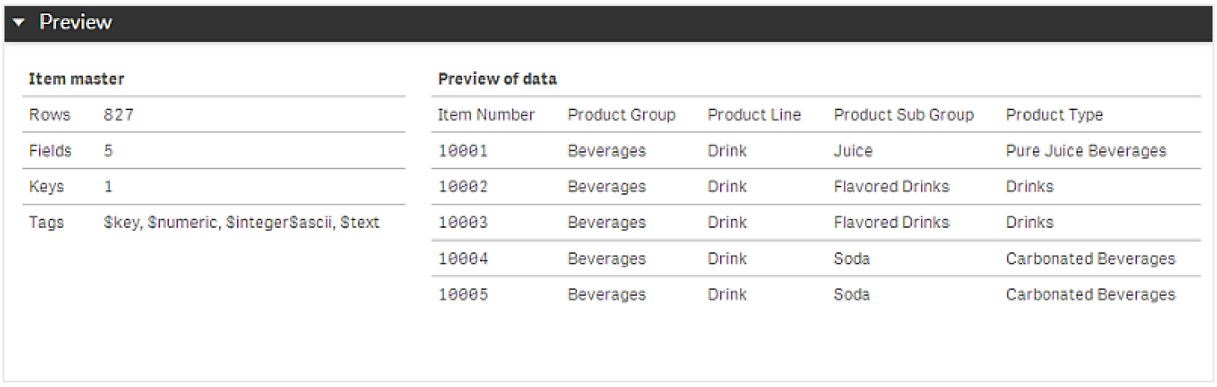
Direct Discovery data is not displayed in the preview .

Showing a preview of a table

Do the following:

- Click a table header in the data model viewer.

The preview panel is displayed with fields and values of the selected table.



Item master	
Rows	827
Fields	5
Keys	1
Tags	$key, $numeric, $integer$ascii, $text

Preview of data

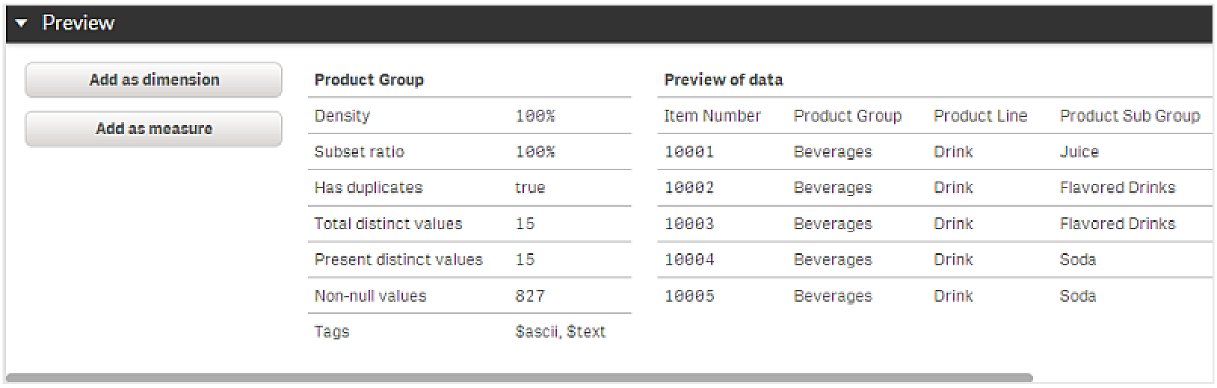
Item Number	Product Group	Product Line	Product Sub Group	Product Type
10001	Beverages	Drink	Juice	Pure Juice Beverages
10002	Beverages	Drink	Flavored Drinks	Drinks
10003	Beverages	Drink	Flavored Drinks	Drinks
10004	Beverages	Drink	Soda	Carbonated Beverages
10005	Beverages	Drink	Soda	Carbonated Beverages

Showing a preview of a field

Do the following:

- Click a table field in the data model viewer.

The preview panel is displayed with the selected field and its values.



Product Group	
Density	100%
Subset ratio	100%
Has duplicates	true
Total distinct values	15
Present distinct values	15
Non-null values	827
Tags	$ascii, $text

Preview of data

Item Number	Product Group	Product Line	Product Sub Group
10001	Beverages	Drink	Juice
10002	Beverages	Drink	Flavored Drinks
10003	Beverages	Drink	Flavored Drinks
10004	Beverages	Drink	Soda
10005	Beverages	Drink	Soda

4.4 Creating a master dimension from the data model viewer

When you are working with an unpublished app, you can create master dimensions so that they can be reused. Users of a published app will have access to the master dimensions in their library, but will not be able to modify them. The data model viewer is not available in a published app.

Do the following:

1. In the data model viewer, select a field and open the **Preview** panel.
2. Click **Add as dimension**.
   The **Create new dimensions** dialog opens, with the selected field. The name of the selected field is also used as the default name of the dimension.
3. Change the name if you want to, and optionally add a description and tags.
4. Click **Add dimension**.
5. Click **Done** to close the dialog.

The dimension is now saved to the master items tab of the assets panel.

*You can quickly add several dimensions as master items by clicking **Add dimension** after adding each dimension. Click **Done** when you have finished.*

Direct Discovery tables are indicated by in the data model viewer.

4.5 Creating a master measure from the data model viewer

When you are working with an unpublished app, you can create master measures so that they can be reused. Users of a published app will have access to the master measures in their library, but will not be able to modify them.

Do the following:

1. In the data model viewer, select a field and open the **Preview** panel.
2. Click **Add as measure**.
   The **Create new measure** dialog opens, with the selected field. The name of the selected field is also used as the default name of the measure.
3. Enter an expression for the measure.
4. Change the name if you want to, and optionally add a description and tags.
5. Click **Create**.

The measure is now saved to the master items tab of the assets panel.

4.6 Data model viewer toolbar

In the data model viewer, you find the following tools in the toolbar at the top of the screen:

	Navigation menu with the following options: **App overview** **Data load editor** **Open hub**
	Menu with the following options: **Help** **About**

Save	Save changes.
	Click the info icon to show or hide the app details.
	Internal table view - the Qlik Sense data model including synthetic fields.
	Source table view - the data model of the source data tables.
	Collapse all tables to show the table name only.
	Reduce the size of all tables to show the table name and all fields with associations to other tables.
	Expand all tables to show all fields.
	Layout menu with the following options: **Grid layout** **Auto layout** **Restore layout**
	Open and close the preview pane.

5 Managing security with section access

You can use section access in the data load script to handle security. In this way, a single file can be used to hold the data for a number of users or user groups. Qlik Sense will use the information in the section access for authentication and authorization, and dynamically reduce the data, so that users only see their own data.

The security is built into the file itself, which means a downloaded file is also protected, to some extent. However, if security demands are high, downloading of files and offline use should be prevented, and files should be published by the Qlik Sense server only. As all data is kept in one file, the size of this file can potentially be very large.

5.1 Sections in the script

Access control is managed through one or several security tables loaded in the same way as Qlik Sense normally loads data. This makes it possible to store these tables in a normal database. The script statements managing the security tables are given within the access section, which in the script is initiated by the statement **Section Access**.

If an access section is defined in the script, the part of the script loading the app data must be put in a different section, initiated by the statement **Section Application**.

Example:

```
Section Access;
LOAD * inline
[ACCESS,USERID
ADMIN, A
USER,U ];
Section Application;
LOAD... ... from... ...
```

Section access system fields

The access levels are assigned to users in one or several tables loaded within the section access. These tables can contain several different user-specific system fields, typically USERID, and the field defining the access level, ACCESS. All section access system fields will be used for authentication or authorization. The full set of section access system fields is described below.

None, all, or any combination of the security fields may be loaded in the access section. Therefore, it is not necessary to use USERID – an authorization can be made using other fields, for example, serial number only.

ACCESS	Defines what access the corresponding user should have.
USERID	Contains a string corresponding to a Qlik Sense user name. Qlik Sense will get the log-on information from the Proxy and compare it to the value in this field.

GROUP	Contains a string corresponding to a group in Qlik Sense. Qlik Sense will resolve the user supplied by the proxy against this group.
OMIT	Contains the name of the field that is to be omitted for this specific user. Wildcards may be used and the field may be empty. An easy way of doing this is to use a subfield.

You should not apply OMIT on key fields, as this will change the underlying data structure. This may create logical islands and calculation inconsistencies.

Qlik Sense will compare the user supplied by the proxy with UserID and resolve the user against groups in the table. If the user belongs to a group that is allowed access, or the user matches, they will get access to the app.

As the same internal logic that is the hallmark of Qlik Sense is also used in the access section, the security fields can be put in different tables. All the fields listed in **LOAD** or **SELECT** statements in the section access must be written in UPPER CASE. Convert any field name containing lower case letters in the database to upper case using the **Upper** function before reading the field by the **LOAD** or **SELECT** statement.

A wildcard, *, is interpreted as all (listed) values of this field, that is. a value listed elsewhere in this table. If used in one of the system fields (USERID, GROUP) in a table loaded in the access section of the script, it is interpreted as all (also not listed) possible values of this field.

When loading data from a QVD file, the use of the upper function will slow down the loading speed.

If you have enabled section access, you cannot use the section access system field names listed here as field names in your data model.

Example:

In this example, only users in the finance group can open the document.

ACCESS	GROUP
USER	Finance

5.2 Dynamic data reduction

Qlik Sense supports the dynamic data reduction feature, by which some of the data in an app can be hidden from the user, based on the section access login:

- Fields (columns) can be hidden by using the system field OMIT.
- Records (rows) can be hidden by linking the section access data with the real data: The selection of values to be shown/excluded is controlled by having one or more fields with common names in section access and section application. After user login, Qlik Sense will attempt to copy the selections in fields in section access to any fields in section application with exactly the same field names (the field names must be written in UPPER CASE). After the selections have been made, Qlik Sense will permanently hide all data excluded by these selections from the user.

All field names used in the transfer described above and all field values in these fields must be upper case, because all field names and field values are, by default, converted to upper case in section access.

Example:

```
section access;
LOAD * inline [
ACCESS, USERID,REDUCTION, OMIT
USER, AD_DOMAIN\ADMIN,*,
USER, AD_DOMAIN\A,1,
USER, AD_DOMAIN\B, 2,NUM
USER, AD_DOMAIN\C, 3, ALPHA
USER, INTERNAL\SA_SCHEDULER,*,
];
section application;
T1:
LOAD *,
NUM AS REDUCTION;
LOAD
Chr( RecNo()+ord('A')-1) AS ALPHA,
RecNo() AS NUM
AUTOGENERATE 3;
```

The field REDUCTION (upper case) now exists in both section access and section application (all field values are also upper case). The two fields would normally be totally different and separated, but using section access, these fields will link and reduce the number of records displayed to the user.

The field OMIT, in section access, defines the fields that should be hidden from the user.

The result will be:

- User ADMIN can see all fields and only those records other users can see in this example when REDUCTION is 1,2, or 3.
- User A can see all fields, but only those records connected to REDUCTION=1.
- User B can see all fields except NUM, and only those records connected to REDUCTION=2.
- User C can see all fields except ALPHA, and only those records connected to REDUCTION=3.

The INTERNAL\SA_SCHEDULER account user is required to enable reload of the script in a Qlik Management Console task.

5.3 Inherited access restrictions

A binary load will cause the access restrictions to be inherited by the new Qlik Sense app. A person with ADMIN rights to this new app may change the access rights of this new app by adding a new access section. A person with USER rights can execute the script and change the script, thus adding their own data to the binary loaded file. A person with USER rights cannot change the access rights. This makes it possible for a database administrator to also control the user access to binary loaded Qlik Sense apps.

6 Accessing large data sets with Direct Discovery

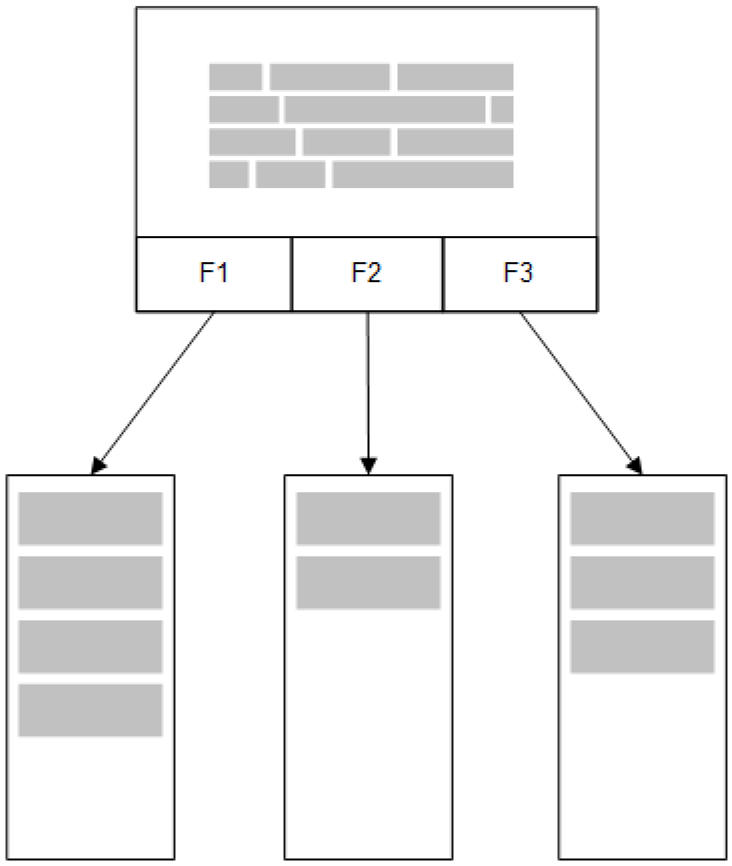
Direct Discovery expands the associative capabilities of the Qlik Sense in-memory data model by providing access to additional source data through an aggregated query that seamlessly associates larger data sets with in-memory data. Direct Discovery enhances business users' ability to conduct associative analysis on big data sources without limitations. Selections can be made on in-memory and Direct Discovery data to see associations across the data sets with the same Qlik Sense association colors - green, white, and gray. Visualizations can analyze data from both data sets together.

Data is selected for Direct Discovery using a special script syntax, **DIRECT QUERY**. Once the Direct Discovery structure is established, Direct Discovery fields can be used along with in-memory data to create Qlik Sense objects. When a Direct Discovery field is used in a Qlik Sense object, an SQL query is run automatically on the external data source.

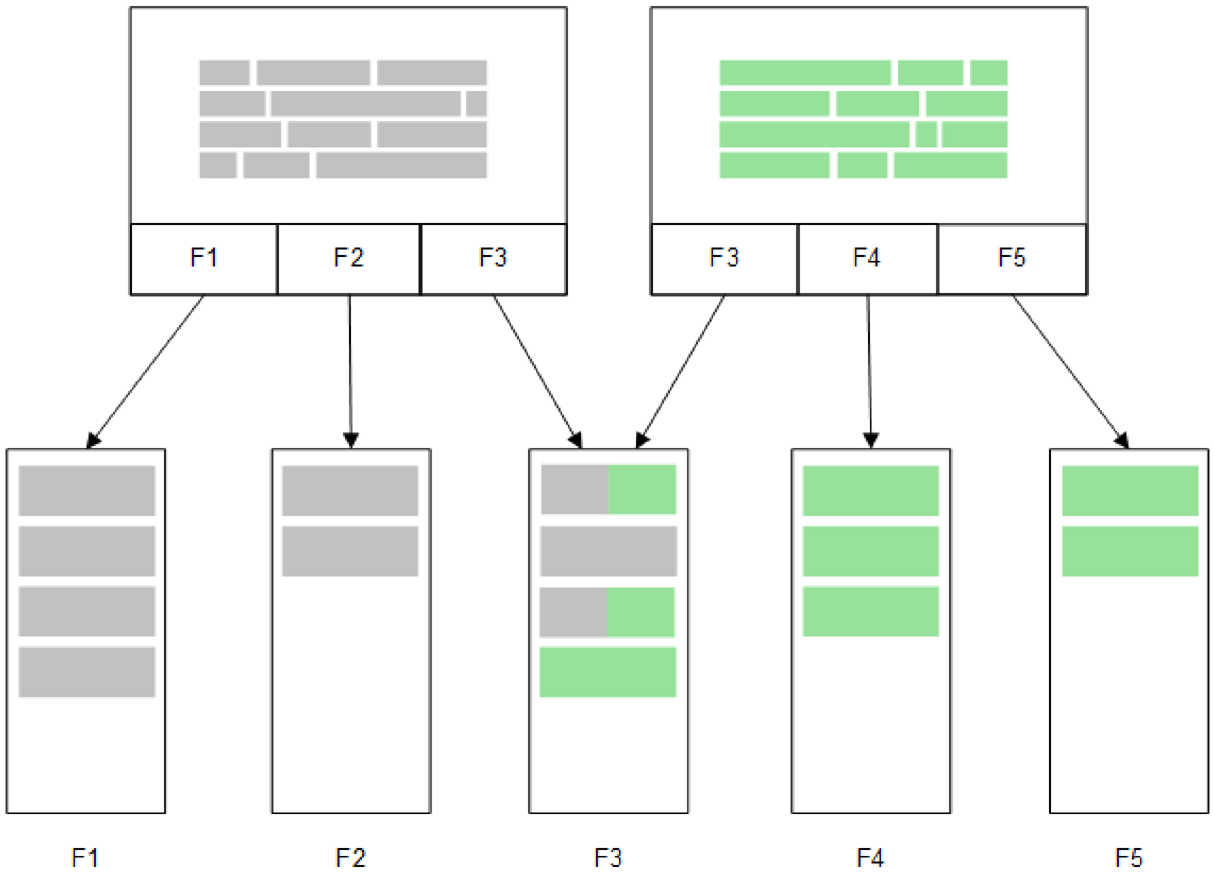
6.1 Differences between Direct Discovery and in-memory data

In-memory model

In the Qlik Sense in-memory model, all unique values in the fields selected from a table in the load script are loaded into field structures, and the associative data is simultaneously loaded into the table. The field data and the associative data is all held in memory.

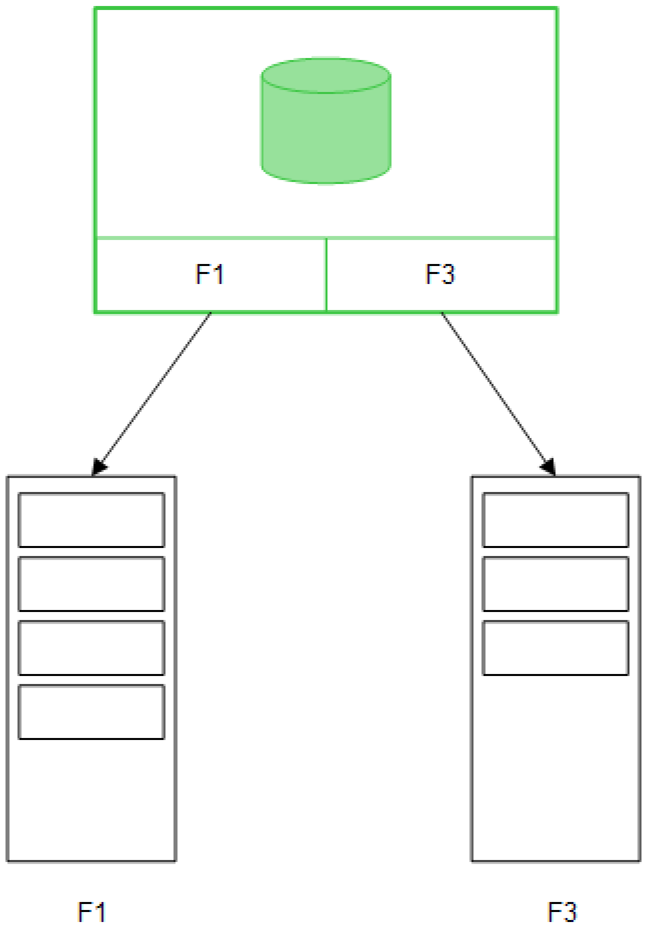


A second, related table loaded into memory would share a common field, and that table might add new unique values to the common field, or it might share existing values.

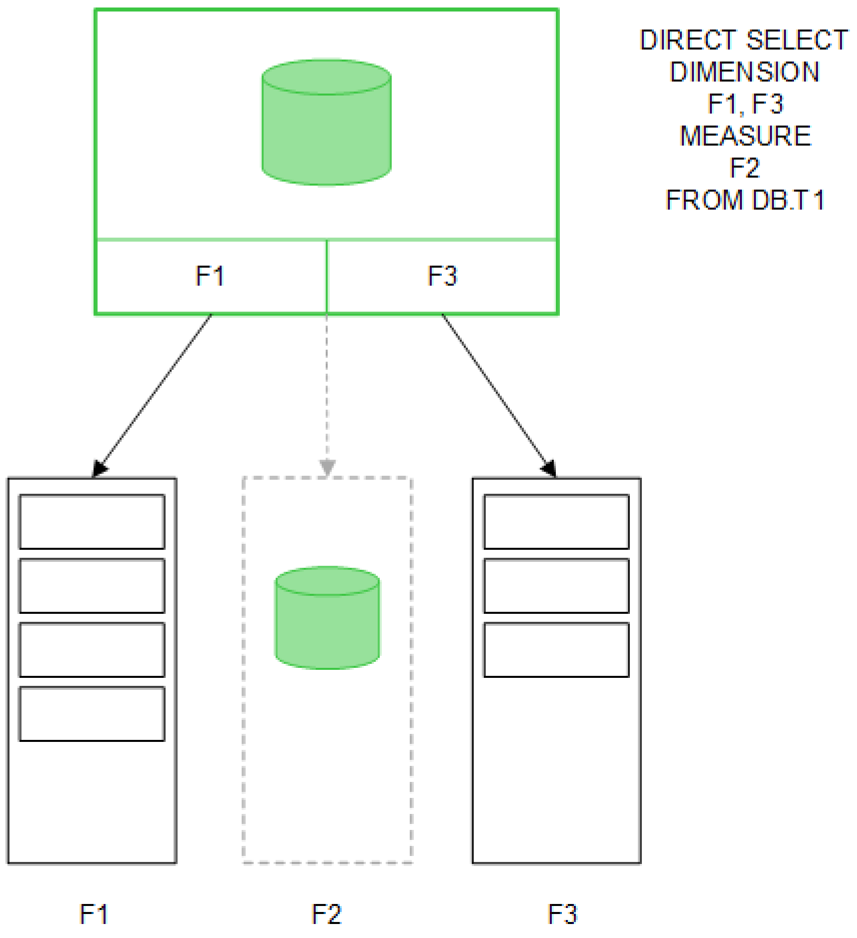


Direct Discovery

When table fields are loaded with a Direct Discovery**LOAD** statement (**Direct Query**), a similar table is created with only the **DIMENSION** fields. As with the In-memory fields, the unique values for the **DIMENSION** fields are loaded into memory. But the associations between the fields are left in the database.



MEASURE field values are also left in the database.



Once the Direct Discovery structure is established, the Direct Discovery fields can be used with certain visualization objects, and they can be used for associations with in-memory fields. When a Direct Discovery field is used, Qlik Sense automatically creates the appropriate SQL query to run on the external data. When selections are made, the associated data values of the Direct Discovery fields are used in the WHERE conditions of the database queries.

With each selection, the visualizations with Direct Discovery fields are recalculated, with the calculations taking place in the source database table by executing the SQL query created by Qlik Sense. The calculation condition feature can be used to specify when visualizations should be recalculated. Until the condition is met, Qlik Sense does not send queries to recalculate the visualizations.

Performance differences between in-memory fields and Direct Discovery fields

In-memory processing is always faster than processing in source databases. Performance of Direct Discovery reflects the performance of the system running the database processing the Direct Discovery queries.

It is possible to use standard database and query tuning best practices for Direct Discovery. All of the performance tuning should be done on the source database. Direct Discovery does not provide support for query performance tuning from the Qlik Sense app. It is possible, however, to make asynchronous, parallel calls to the database by using the connection pooling capability. The load script syntax to set up the pooling capability is:

```
SET DirectConnectionMax=10;
```

Qlik Sense caching also improves the overall user experience. See *Caching and Direct Discovery (page 61)* below.

Performance of Direct Discovery with **DIMENSION** fields can also be improved by detaching some of the fields from associations. This is done with the **DETACH** keyword on **DIRECT QUERY**. While detached fields are not queried for associations, they are still part of the filters, speeding up selection times.

While Qlik Sense in-memory fields and Direct Discovery**DIMENSION** fields both hold all their data in memory, the manner in which they are loaded affects the speed of the loads into memory. Qlik Sense in-memory fields keep only one copy of a field value when there are multiple instances of the same value. However, all field data is loaded, and then the duplicate data is sorted out.

DIMENSION fields also store only one copy of a field value, but the duplicate values are sorted out in the database before they are loaded into memory. When you are dealing with large amounts of data, as you usually are when using Direct Discovery, the data is loaded much faster as a **DIRECT QUERY** load than it would be through the **SQL SELECT** load used for in-memory fields.

Differences between data in-memory and database data

DIRECT QUERY is case-sensitive when making associations with in-memory data. Direct Discovery selects data from source databases according to the case-sensitivity of the database. If a database is not case-sensitive, a Direct Discovery query might return data that an in-memory query would not. For example, if the following data exists in a database that is not case-sensitive, a Direct Discovery query of the value "Red" would return all four rows.

ColumnA	ColumnB
red	one
Red	two
rED	three
RED	four

An in-memory selection of "Red," on the other hand, would return only:

```
Red two
```

Qlik Sense normalizes data to an extent that produces matches on selected data that databases would not match. As a result, an in-memory query may produce more matching values than a Direct Discovery query. For example, in the following table, the values for the number "1" vary by the location of spaces around them:

ColumnA	ColumnB

' 1'	space_before
'1'	no_space
'1 '	space_after
'2'	two

If you select "1" in a **Filter pane** for ColumnA, where the data is in standard Qlik Sense in-memory, the first three rows are associated:

' 1'	space_before
'1'	no_space
'1 '	space_after

If the **Filter pane** contains Direct Discovery data, the selection of "1" might associate only "no_space". The matches returned for Direct Discovery data depend on the database. Some return only "no_space" and some, like SQL Server, return "no_space" and "space_after".

Caching and Direct Discovery

Qlik Sense caching stores selection states of queries in memory. As the same types of selections are made, Qlik Sense leverages the query from the cache instead of querying the source data. When a different selection is made, an SQL query is made on the data source. The cached results are shared across users.

Example:

1. User applies initial selection.
   SQL is passed through to the underlying data source.
2. User clears selection and applies same selection as initial selection.
   Cache result is returned, SQL is not passed through to the underlying data source.
3. User applies different selection.
   SQL is passed through to the underlying data source.

A time limit can be set on caching with the **DirectCacheSeconds** system variable.Once the time limit is reached, Qlik Sense clears the cache for the Direct Discovery query results that were generated for the previous selections.Qlik Sense then queries the source data for the selections and recreates the cache for the designated time limit.

The default cache time for Direct Discovery query results is 30 minutes unless the **DirectCacheSeconds** system variable is used.

6.2 Direct Discovery field types

Within Direct Discovery, there are three types of data fields: DIMENSION, MEASURE, and DETAIL. The types are set on data fields when the Direct Discovery selection is made using the **Direct Query** statement in the load script.

All Direct Discovery fields can be used in combination with in-memory fields. Typically, fields with discrete values that will be used as dimensions should be loaded with the DIMENSION keyword, whereas numeric data that will be used in aggregations only should be marked as MEASURE fields. DIMENSION fields cannot be used in object expressions.

The following table summarizes the characteristics and usage of the Direct Discovery field types:

Field Type	**In memory?**	**Forms association?**	**Used in chart expressions?**
DIMENSION	Yes	Yes	Yes
MEASURE	No	No	Yes
DETAIL	No	No	No

DIMENSION fields

DIMENSION fields are loaded in memory and can be used to create associations between in-memory data and the data in Direct Discovery fields. Direct DiscoveryDIMENSION fields are also used to define dimension values in charts.

MEASURE fields

MEASURE fields, on the other hand, are recognized on a "meta level." MEASURE fields are not loaded in memory (they do not appear in the data model viewer).The purpose is to allow aggregations of the data in MEASURE fields to take place in the database rather than in memory. Nevertheless, MEASURE fields can be used in expressions without altering the expression syntax. As a result, the use of Direct Discovery fields from the database is transparent to the end user.

The following aggregation functions can be used with MEASURE fields:

- **Sum**
- **Avg**
- **Count**
- **Min**
- **Max**

DETAIL fields

DETAIL fields provide information or details that you may want to display, but not use in chart expressions. DETAIL fields can only be used in **Count** aggregations, and can only be viewed in a **Table**. Fields designated as DETAIL commonly contain data that cannot be aggregated in any meaningful way, like comments.

Any field can be designated as a DETAIL field.

6.3 Data sources supported in Direct Discovery

Qlik SenseDirect Discovery can be used against the following data sources, both with 32-bit and 64-bit connections;

- ODBC/OLEDB data sources - All ODBC/OLEDB sources are supported, including SQL Server, Teradata and Oracle.
- Custom connectors which support SQL – SAP SQL Connector, Custom QVX connectors for SQL compliant data stores.

Both 32-bit and 64-bit connections are supported.

SAP

For SAP, Direct Discovery can be used only with the Qlik SAP SQL Connector, and it requires the following parameters in **SET** variables:

```
SET DirectFieldColumnDelimiter=' ';
SET DirectIdentifierQuoteChar=' ';
```

SAP uses OpenSQL, which delimits columns with a space rather than a comma, so the above set statements cause a substitution to accommodate the difference between ANSI SQL and OpenSQL.

Google Big Query

Direct Discovery can be used with Google Big Query and requires the following parameters in the set variables:

```
SET DirectDistinctSupport=false;
SET DirectIdentifierQuoteChar='[]';
SET DirectIdentifierQuoteStyle='big query'
```

Google Big Query does not support **SELECT DISTINCT** or quoted column/table names and has non-ANSI quoting configuration using '[ ]'.

MySQL and Microsoft Access

Direct discovery can be used in conjunction with MySQL and Microsoft Access but may require the following parameters in the set variables due to the quoting characters used in these sources:

```
SET DirectIdentifierQuoteChar='``';
```

6.4 Limitations when using Direct Discovery

Supported data types

All data types are supported in Direct Discovery, though there may be cases in which specific source data formats need to be defined to Qlik Sense. This can be done on the load script by using the "SET Direct...Format" syntax. The following example demonstrates how to define the date format of the source database that is used as the source for Direct Discovery:

Example:

```
SET DirectDateFormat='YYYY-MM-DD';
```

There are also two script variables for controlling how the Direct Discovery formats money-type values in the generated SQL statements:

```
SET DirectMoneyFormat (default '#.0000')
SET DirectMoneyDecimalSep (default '.')
```

The syntax for these two variables is the same as for **MoneyFormat** and **MoneyDecimalSep**, but there are two important differences in the usage:

- This is not a display format, so it should not include currency symbols or thousands separators.
- The default values are not driven by the locale but are hard wired to the values. (Locale-specific formats include the currency symbol.)

Direct Discovery can support the selection of extended Unicode data by using the SQL standard format for extended character string literals (N'<extended string>') as required by some databases, such as SQL Server. This syntax can be enabled for Direct Discovery with the script variable **DirectUnicodeStrings** . Setting this variable to "true" enables the use of "N" in front of the string literals.

Security

The following security best practices should be taken into consideration when using Direct Discovery:

- All of the users using the same application with the Direct Discovery capability use the same connection. Authentication pass-through and credentials-per-user are not supported.
- Section Access is supported in server mode only.
- It is possible to execute custom SQL statements in the database with a NATIVE keyword expression, so the database connection set up in the load script should use an account that has read-only access to the database.
- Direct Discovery has no logging capability, but it is possible to use the ODBC tracing capability.
- It is possible to flood the database with requests from the client.
- It is possible to get detailed error messages from the server log files.

Qlik Sense functionality that is not supported

Because of the interactive and SQL syntax-specific nature of Direct Discovery, several features are not supported:

- Advanced calculations (Set Analysis, complex expressions)
- Calculated dimensions
- Comparative Analysis (Alternate State) on the objects that use Direct Discovery fields
- Direct Discovery**MEASURE** and **DETAIL** fields are not supported in the search tool
- Binary load from an application that is accessing a Direct Discovery table
- Loop and Reduce
- Synthetic keys on the Direct Discovery table
- Table naming in script does not apply to the Direct Discovery table
- The use of wild card * character after **DIRECT QUERY** keyword on the load script (`DIRECT QUERY *`)
- Oracle database tables with LONG datatype columns are not supported.
- Big integers in scientific notation, outside range [-9007199254740990, 9007199254740991], can cause rounding errors and undefined behavior.

6.5 Multi-table support in Direct Discovery

You can use Direct Discovery to load more than one table or view using ANSI SQL join functionality. In a single chart, all measures must be derived from the same logical table in Qlik Sense, but this can be a combination of several tables from source linked via join statements. However, you can use dimensions sourced from other tables in the same chart.

For example, you can link the tables loaded with Direct Discovery using either a **Where** clause or a **Join** clause.

Linking Direct Discovery tables with a **Where** clause

In this example script, we load data from the database AW2012. The tables Product and ProductSubcategory are linked with a **Where** clause using the common ProductSubCategoryID field.

```
      Product_Join:
DIRECT QUERY
DIMENSION
   [ProductID],
   [AW2012].[Production].[Product].[Name] as [Product Name],
   [AW2012].[Production].[ProductSubcategory].[Name] as [Sub Category Name],
   Color,
   [AW2012].[Production].[Product].ProductSubcategoryID as [SubcategoryID]
MEASURE
   [ListPrice]
FROM [AW2012].[Production].[Product],
     [AW2012].[Production].[ProductSubcategory]
WHERE [AW2012].[Production].[Product].ProductSubcategoryID =
      [AW2012].[Production].[ProductSubcategory].ProductSubcategoryID   ;
```

Linking Direct Discovery tables with **Join On** clauses

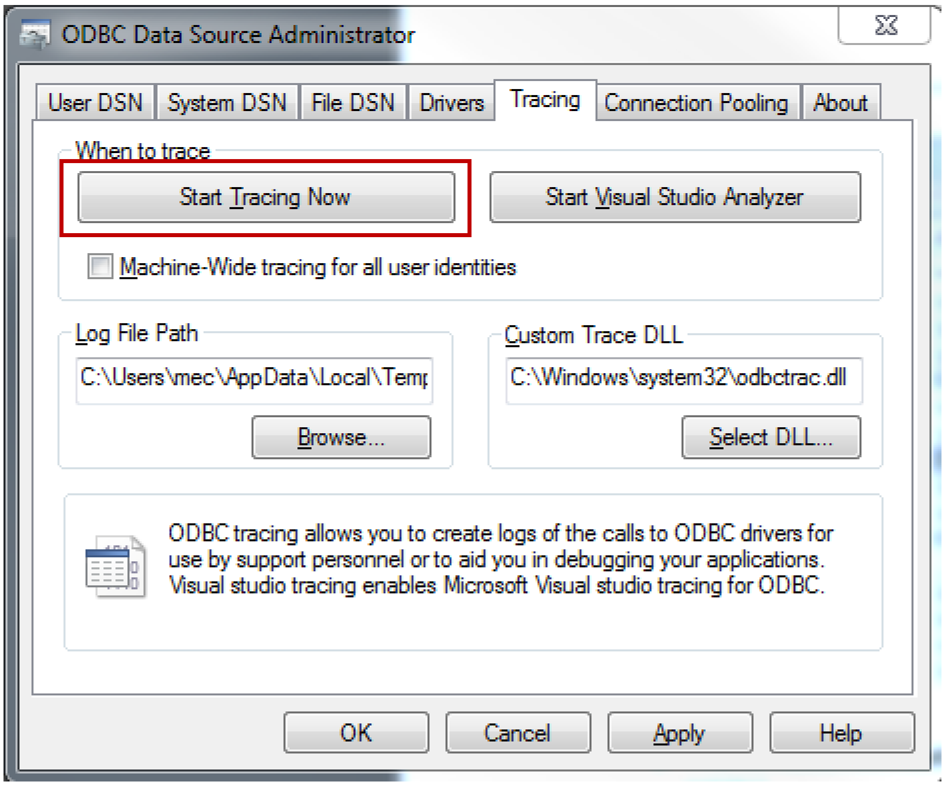
You can also use **Join On** clauses to link Direct Discovery tables. In this example statement we join the SalesOrderHeader table to the SalesOrderDetail table via the SalesOrderID field, and also join the Customer table to the SalesOrderHeader table via the Customer ID field.

In this example we create measures from the same logical table, which means we can use them in the same chart. For example, you can create a chart with SubTotal and OrderQty as measures.

```
Sales_Order_Header_Join:
DIRECT QUERY
DIMENSION
   AW2012.Sales.Customer.CustomerID as CustomerID,
   AW2012.Sales.SalesOrderHeader.SalesPersonID as SalesPersonID,
   AW2012.Sales.SalesOrderHeader.SalesOrderID as SalesOrderID,
   ProductID,
   AW2012.Sales.Customer.TerritoryID as TerritoryID,
   OrderDate,
   NATIVE('month([OrderDate])') as OrderMonth,
   NATIVE('year([OrderDate])') as OrderYear
MEASURE
   SubTotal,
   TaxAmt,
   TotalDue,
   OrderQty
DETAIL
   DueDate,
   ShipDate,
   CreditCardApprovalCode,
   PersonID,
   StoreID,
   AccountNumber,
   rowguid,
   ModifiedDate
FROM AW2012.Sales.SalesOrderDetail
   JOIN AW2012.Sales.SalesOrderHeader
   ON (AW2012.Sales.SalesOrderDetail.SalesOrderID =
       AW2012.Sales.SalesOrderHeader.SalesOrderID)
   JOIN AW2012.Sales.Customer
   ON(AW2012.Sales.Customer.CustomerID =
      AW2012.Sales.SalesOrderHeader.CustomerID);
```

6.6 Logging Direct Discovery access

Direct DiscoverySQL statements passed to the data source can be recorded in the trace files of the database connection. For a standard ODBC connection, tracing is started with the **ODBC Data Source Administrator**:



C:\Users\mec\AppData\Local\Temp

C:\Windows\system32\odbctrac.dll

ODBC tracing allows you to create logs of the calls to ODBC drivers for use by support personnel or to aid you in debugging your applications. Visual studio tracing enables Microsoft Visual studio tracing for ODBC.

The resulting trace file details SQL statements generated through the user selections and interactions.

7 Introduction to data modeling

This introduction serves as a brief presentation of how you can load data into Qlik Sense to provide background for the topics in this section, presenting how to achieve basic data loading and transformation.

Qlik Sense uses a data load script, which is managed in the data load editor, to connect to and retrieve data from various data sources. In the script, the fields and tables to load are specified. It is also possible to manipulate the data structure by using script statements and expressions.

During the data load, Qlik Sense identifies common fields from different tables (key fields) to associate the data. The resulting data structure of the data in the app can be monitored in the data model viewer. Changes to the data structure can be achieved by renaming fields to obtain different associations between tables.

After the data has been loaded into Qlik Sense, it is stored in the app. The app is the heart of the program's functionality and it is characterized by the unrestricted manner in which data is associated, its large number of possible dimensions, its speed of analysis and its compact size. The app is held in RAM when it is open.

Analysis in Qlik Sense always happens while the app is not directly connected to its data sources. So, to refresh the data, you need to reload the script.

7.1 Understanding data structures

Data loading statements

Data is loaded by **LOAD** or **SELECT** statements. Each of these statements generates an internal table. A table can always be seen as a list of something, each record (row) then being a new instance of the object type and each field (column) being a specific attribute or property of the object.

Rules

The following rules apply when loading data into Qlik Sense:

- Qlik Sense does not make any difference between tables generated by a **LOAD** or a **SELECT** statement. This means that if several tables are loaded, it does not matter whether the tables are loaded by **LOAD** or **SELECT** statements or by a mix of the two.
- The order of the fields in the statement or in the original table in the database is arbitrary to the Qlik Sense logic.
- Field names are used in the further process to identify fields and making associations. These are case sensitive, which often makes it necessary to rename fields in the script.

Execution of the script

For a typical **LOAD** or **SELECT** statement the order of events is roughly as follows:

1. Evaluation of expressions
2. Renaming of fields by **as**
3. Renaming of fields by **alias**

4. Qualification of field names
5. Mapping of data if field name matches
6. Storing data in an internal table

Fields

Fields are the primary data-carrying entities in Qlik Sense. A field typically contains a number of values, called field values. In database terminology we say that the data processed by Qlik Sense comes from data files. A file is composed of several fields where each data entry is a record. The terms file, field and record are equivalent to table, column and row respectively. The Qlik Sense AQL logic works only on the fields and their field values.

Field data is retrieved by script via **LOAD**, **SELECT** or **Binary** statements. The only way of changing data in a field is by re-executing the script. The actual field values can not be manipulated by the user from the layout or by means of automation. Once read into Qlik Sense they can only be viewed and used for logical selections and calculations.

Field values consist of numeric or alphanumeric (text) data. Numeric values actually have dual values, the numeric value and its current, formatted text representation. Only the latter is displayed in sheet objects etc.

The content of a field can be represented in a filter pane.

Field tags

There are three different types of system tags, script generated system tags that cannot be altered by the user, script generated system tags that can be altered in the script and system tags that are set interactively by the user. System tags are always preceded by a $ sign.

The following system tags are automatically generated at the end of script generation. These cannot be changed by the user:

- $system - denotes a system field.
- $key - denotes a key field.
- $keypart - denotes that the field is part of one or more synthetic keys.
- $syn - denotes a synthetic key.

The following tags are also automatically generated at the end of script generation, but may be altered or overridden using script.

- $hidden - denotes a hidden field.
- $numeric - all (non-NULL) values in the field are numeric.
- $integer - all (non-NULL) values in the field are integers.
- $text - no values in the field are numeric.
- $ascii - field values contain only standard ascii characters.
- $date - all (non-NULL) values in the field can be interpreted as dates (integers).
- $timestamp - all (non-NULL) values in the field can be interpreted as time stamps.

The following tags can be enabled and disabled by the user:

- $dimension - denotes a field recommended for use in chart dimensions, filter panes, etc.
- $measure - denotes a field recommended for use in expressions.

The user can also add custom tags in the script. These custom tags may not use the same name as any system tag.

System fields

In addition to the fields extracted from the data source, system fields are also produced by Qlik Sense. These all begin with "$" and can be displayed like ordinary fields in a visualization, such as a filter pane or a table. System fields are typically created during script execution, and are primarily used as an aid in app design.

System fields are not included in field lists in the assets panel or the Expression editor. If you want to use a system field, you need to reference it by typing it manually.

Example: =$Field

Available system fields

The following system fields are available:

$Table	Displays all internal tables loaded by the script.
$Field	Displays the fields that are read from the tables.
$Fields	Represent the number of fields in different tables.
$FieldNo	Displays the position of the fields in the tables.
$Rows	Displays the number of rows in the tables.
$Info	Displays names of info tables, if they have been included in the app.

Logical tables

Each **LOAD** or **SELECT** statement generates a table. Normally, Qlik Sense treats the result of each one of these as one logical table. However, there are a couple of exceptions from this rule:

- If two or more statements result in tables with identical field names, the tables are concatenated and treated as one logical table.
- If a **LOAD** or **SELECT** statement is preceded by any of the following qualifiers, data is altered or treated differently:

concatenate	This table is concatenated with (added to) another named table or with the last previously created logical table.
crosstable	This table is converted from crosstable format to column format.
generic	This table is split into several other logical tables.

info	This table is loaded not as a logical table, but as an information table containing links to external info such as files, sounds, URLs, etc.
intervalmatch	The table (which must contain exactly two columns) is interpreted as numeric intervals, which are associated with discrete numbers in a specified field.
join	This table is joined by Qlik Sense with another named table or with the last previously created logical table, over the fields in common.
keep	This table is reduced to the fields in common with another named table or with the last previously created logical table.
mapping	This table (which must contain exactly two columns) is read as a mapping table, which is never associated with other tables.
semantic	This table is loaded not as a logical table, but as a semantic table containing relationships that should not be joined, e.g. predecessor, successor and other references to other objects of the same type.

When the data has been loaded, the logical tables are associated.

Table names

Qlik Sense tables are named when they are stored in the Qlik Sense database. The table names can be used, for example for,**LOAD** statements with a **resident** clause or with expressions containing the **peek** function, and can be seen in the *$Table* system field in the layout.

Tables are named in accordance with the following rules:

1. If a label immediately precedes a **LOAD** or **SELECT** statement the label is used as table name. The label must be followed by a colon.

   Example:

   ```
   Table1:
   LOAD a,b from c.csv;
   ```

2. If no label is given, the file name or table name immediately following the keyword **FROM** in the **LOAD** or **SELECT** statement is used. A maximum of 32 characters is used. The extension is skipped if the file name is used.
3. Tables loaded inline are named INLINExx, where xx is a number. The first inline table will be given the name *INLINE01*.
4. Automatically generated tables are named AUTOGENERATExx, where xx is a number. The first autogenerated table is given the name *AUTOGENERATE01*.
5. If a table name generated according to the rules above should be in conflict with a previous table name, the name is extended with -x , where x is a number. The number is increased until no conflict remains. For example, three tables could be named *Budget*, *Budget-1* and *Budget-2*.

There are three separate domains for table names: **section access**, **section application** and mapping tables. Table names generated in **section access** and **section application** are treated separately. If a table name referenced is not found within the section, Qlik Sense searches the other section as well. Mapping tables are treated separately and have no connection whatsoever to the other two domains of table names.

Table labels

A table can be labeled for later reference, for example by a **LOAD** statement with a **resident** clause or with expressions containing the **peek** function. The label, which can be an arbitrary string of numbers or characters, should precede the first **LOAD** or **SELECT** statement that creates the table. The label must be followed by a colon "**:**".

Labels containing blanks must be quoted using single or double quotation marks or square brackets.

Example 1:

```
Table1:
LOAD a,b from c.csv;
LOAD x,y from d.csv where x=peek('a',y,'Table1');
```

Example 2: Table label containing a blank

```
[All Transactions]:
SELECT * from Transtable;
LOAD Month, sum(Sales) resident [All Transactions] group by Month;
```

Associations between logical tables

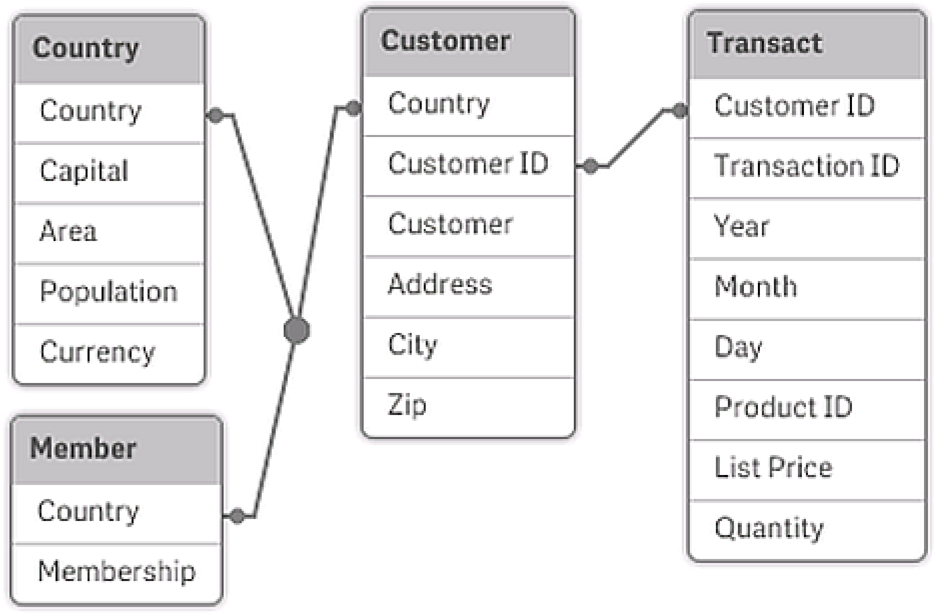
A database can have many tables. Each table can be considered as a list of something; each record in the list represents an instance of an object of some type.

Example:

If two tables are lists of different things, for example if one is a list of customers and the other a list of invoices, and the two tables have a field such as the customer number in common, this is usually a sign that there is a relationship between the two tables. In standard SQL query tools the two tables should almost always be joined.

The tables defined in the Qlik Sense script are called logical tables. Qlik Sense makes associations between the tables based on the field names, and performs the joins when a selection is made, for example selecting a field value in a filter pane.

This means that an association is almost the same thing as a join. The only difference is that the join is performed when the script is executed - the logical table is usually the result of the join. The association is made after the logical table is created - associations are always made between the logical tables.



Four tables: a list of countries, a list of customers, a list of transactions and a list of memberships, which are associated with each other through the fields Country and CustomerID.

Qlik Sense association compared to SQL natural outer join

A Qlik Sense association resembles a SQL natural outer join. The association is however more general: an outer join in SQL is usually a one-way projection of one table on another. An association always results in a full (bidirectional) natural outer join.

Frequency information in associating fields

There are some limitations in the use of most associating fields, that is, fields that are common between two or more tables. When a field occurs in more than one table, Qlik Sense has a problem knowing which of the tables it should use for calculating data frequencies.

Qlik Sense analyzes the data to see if there is a non-ambiguous way to identify a main table to count in (sometimes there is), but in most cases the program can only make a guess. Since an incorrect guess could be fatal (Qlik Sense would appear to make a calculation error) the program has been designed not to allow certain operations when the data interpretation is ambiguous for associating fields.

Limitations for associating fields

1. It is not possible to display frequency information in a filter pane showing the field.
2. Statistics boxes for the field show n/a for most statistical entities.
3. In charts, it is not possible to create expressions containing functions that depend on frequency information (such as Sum, Count functions, and Average) on the field, unless the **Distinct** modifier is activated. After each reload, Qlik Sense will scan all chart expressions to see if any ambiguities have occurred as a result of changes in data structures. If ambiguous expressions are found, a warning dialog will be shown and the expression will be disabled. It will not be possible to enable the expression until the problem has been corrected. If a log file is enabled, all ambiguous expressions will be listed in the log.

Workaround

There is a simple way to overcome these limitations. Load the field an extra time under a new name from the table where frequency counts should be made. Then use the new field for a filter pane with frequency, for a statistics box or for calculations in the charts.

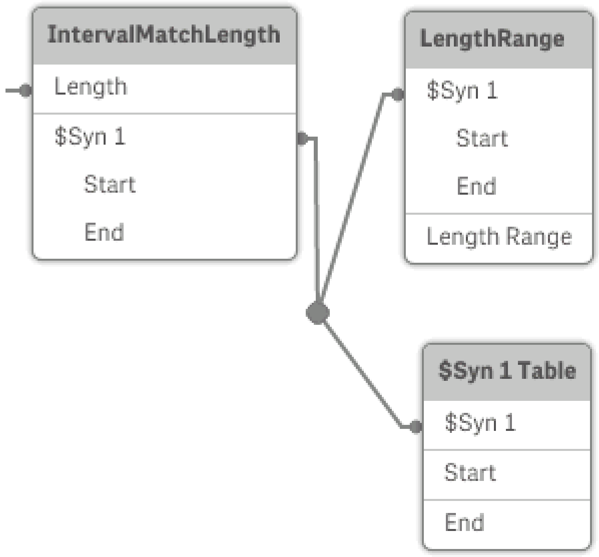
Synthetic keys

When two or more internal tables have two or more fields in common, this implies a composite key relationship. Qlik Sense handles this by creating synthetic keys automatically. These keys are anonymous fields that represent all occurring combinations of the composite key.

If you receive a warning about synthetic keys when loading data, it is recommended that you review the data structure in the data model viewer. You should ask yourself whether the data model is correct or not. Sometimes it is, but often enough the synthetic key is there due to an error in the script.

Multiple synthetic keys are often a symptom of an incorrect data model, but not necessarily. However, a sure sign of an incorrect data model is if you have synthetic keys based on other synthetic keys.

When the number of synthetic keys increases, depending on data amounts, table structure and other factors, Qlik Sense may or may not handle them gracefully, and may end up using excessive amount of time and/or memory. In such a case you need to re-work your script by removing all synthetic keys.



Handling synthetic keys

If you need to avoid synthetic keys, there are a number of ways for solving this in the data load script:

- Check that only fields that logically link two tables are used as keys.
    - Fields like “Comment”, “Remark” and “Description” may exist in several tables without being related, and should therefore not be used as keys.

    - Fields like “Date”, “Company” and “Name” may exist in several tables and have identical values, but still have different roles (Order Date/Shipping Date, Customer Company/Supplier Company). In such cases they should not be used as keys.
- Make sure that redundant fields aren’t used – that only the necessary fields connect. If for example a date is used as a key, make sure not to load year, month or day_of_month of the same date from more than one internal table.
- If necessary, form your own non-composite keys, typically using string concatenation inside an AutoNumber script function.

Data types in Qlik Sense

Qlik Sense can handle text strings, numbers, dates, times, timestamps, and currencies correctly. They can be sorted, displayed in a number of different formats and they can be used in calculations. This means, for example, that dates, times, and timestamps can be added to or subtracted from each other.

Data representation inside Qlik Sense

In order to understand data interpretation and number formatting in Qlik Sense, it is necessary to know how data is stored internally by the program. All of the data loaded into Qlik Sense is available in two representations: as a string and as a number.

1. The string representation is always available and is what is shown in the list boxes and the other sheet objects. Formatting of data in list boxes (number format) only affects the string representation.
2. The number representation is only available when the data can be interpreted as a valid number. The number representation is used for all numeric calculations and for numeric sorting.

If several data items read into one field have the same number representation, they will all be treated as the same value and will all share the first string representation encountered. Example: The numbers 1.0, 1 and 1.000 read in that order will all have the number representation 1 and the initial string representation 1.0.

Number interpretation

When you load data containing numbers, currency, or dates, it will be interpreted differently depending on whether the data type is defined or not. This section describes how data is interpreted in the two different cases.

Data with type information

Fields containing numbers with a defined data type in a database loaded using ODBC will be handled by Qlik Sense according to their respective formats. Their string representation will be the number with an appropriate formatting applied.

Qlik Sense will remember the original number format of the field even if the number format is changed for a measure under **Number formatting** in the properties panel.

The default formats for the different data types are:

- integer, floating point numbers: the default setting for number
- currency: the default setting for currency
- time, date, timestamp: ISO standard formatting

The default settings for number and currency are defined using the script number interpretation variables or the operating system settings (**Control Panel**).

Data without type information

For data without specific formatting information from the source (for example, data from text files or ODBC data with a general format) the situation becomes more complicated. The final result will depend on at least six different factors:

1. The way data is written in the source database
2. The operating system settings for number, time, date and so on. (**Control Panel**)
3. The use of optional number-interpreting variables in the script
4. The use of optional interpretation functions in the script
5. The use of optional formatting functions in the script
6. The number formatting controls in the document

Qlik Sense tries to interpret input data as a number, date, time, and so on. As long as the system default settings are used in the data, the interpretation and the display formatting is done automatically by Qlik Sense, and the user does not need to alter the script or any setting in Qlik Sense. There is an easy way to find out if the input data has been correctly interpreted: numeric values are right-aligned in list boxes, whereas text strings are left-aligned.

By default, the following scheme is used until a complete match is found. (The default format is the format such as the decimal separator, the order between year, month and day, and so on, specified in the operating system, that is, in the **Control Panel**, or in some cases from the special number interpretation variables in the script.

Qlik Sense will interpret the data as:

1. A number in accordance with the default format for numbers.
2. A date according to the default format for date.
3. A timestamp according to the default format for time and date.
4. A time according to the default format for time.
5. A date according to the following format: yyyy-MM-dd.
6. A time-stamp according to the following format: YYYY-MM-DD hh:mm[:ss[.fff]].
7. A time according to the following format: hh:mm[:ss[.fff]].
8. Money according to the default format for currency.
9. A number with '.' as decimal separator and ',' as thousands separator, provided that neither the decimal separator nor the thousands separator are set to ','.
10. A number with ',' as decimal separator and '.' as thousands separator, provided that neither the decimal separator nor the thousands separator are set to ‘.'.
11. A text string. This last test never fails: if it is possible to read the data, it is always possible to interpret it as a string.

When loading numbers from text files, some interpretation problems may occur, for example, an incorrect thousands separator or decimal separator may cause Qlik Sense to interpret the number incorrectly. The first thing to do is to check that the number-interpretation variables in the script are correctly defined and that the system settings in the **Control Panel** are correct.

When Qlik Sense has interpreted data as a date or time, it is possible to change to another date or time format in the properties panel of the visualization.

Since there is no predefined format for the data, different records may, of course, contain differently formatted data in the same field. It is possible for example, to find valid dates, integers, and text in one field. The data will therefore, not be formatted, but shown in its original form.

Date and time interpretation

Qlik Sense stores each date, time, and timestamp found in data as a date serial number. The date serial number is used for dates, times and timestamps and in arithmetic calculations based on date and time entities. Dates and times can thus be added and subtracted, intervals can be compared, and so on.

The date serial number is the (real valued) number of days passed since December 30, 1899, that is, the Qlik Sense format is identical to the 1900 date system used by Microsoft Excel and other programs, in the range between March 1, 1900 and February 28, 2100. For example, 33857 corresponds to September 10, 1992. Outside this range, Qlik Sense uses the same date system extended to the Gregorian calendar.

The serial number for times is a number between 0 and 1. The serial number 0.00000 corresponds to 00:00:00, whereas 0.99999 corresponds to 23:59:59. Mixed numbers indicate the date and time: the serial number 2.5 represents January 1, 1900 at 12:00 noon.

The data is, however, displayed according to the format of the string. By default, the settings made in the **Control Panel** are used. It is also possible to set the format of the data by using the number interpretation variables in the script or with the help of a formatting function. Lastly,it is also possible to reformat the data in the properties sheet of the sheet object.

Example 1:

1997-08-06	is stored as:	35648
09:00	is stored as:	0.375
1997-08-06 09:00	is stored as:	35648.375

and the other way around.

35648	with number format 'D/M/YY' is shown as:	6/8/97
0.375	with number format 'hh.mm' is shown as:	09.00

Qlik Sense follows a set of rules to try to interpret dates, times, and other data types. The final result, however, will be affected by a number of factors as described here.

Example 2:

These examples assume the following default settings:

- Number decimal separator: .
- Short date format: YY-MM-DD
- Time format: hh:mm

The following table shows the different representations when data is read into Qlik Sense without the special interpretation function in the script:

Source data	**Qlik Sense default interpretation**	**'YYYY-MM-DD' date format**	**'MM/DD/YYYY' date format**	**'hh:mm' time format**	**'# ##0.00' number format**
0.375	0.375	1899-12-30	12/30/1899	09:00	0.38
33857	33857	1992-09-10	09/10/1992	00:00	33 857.00
97-08-06	97-08-06	1997-08-06	08/06/1997	00:00	35 648.00
970806	970806	4557-12-21	12/21/4557	00:00	970 806.00
8/6/97	8/6/97	8/6/97	8/6/97	8/6/97	8/6/97

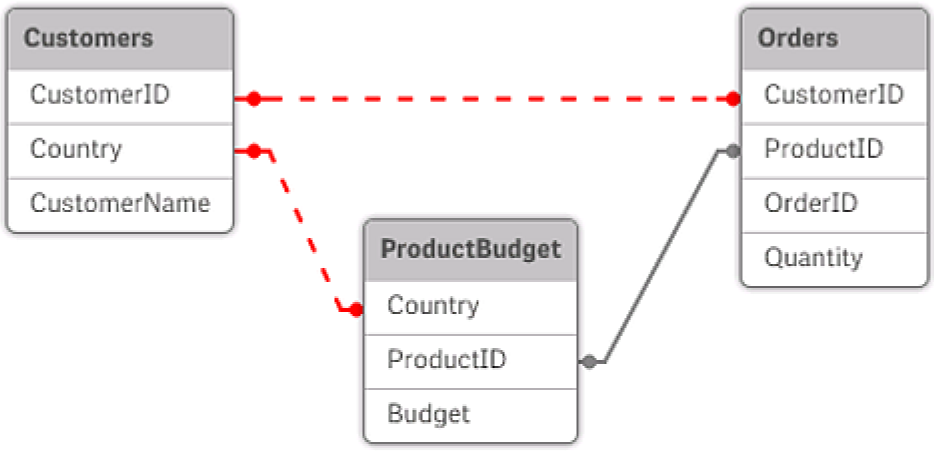
The following table shows the different representations when data is read into Qlik Sense using the date#(A, 'M/D/YY') interpretation function in the script:

Source data	**Qlik Sense default interpretation**	**'YYYY-MM-DD' date format**	**'MM/DD/YYYY' date format**	**'hh:mm' time format**	**'# ##0.00' number format**
0.375	0.375	0.375	0.375	0.375	0.375
33857	33857	33857	33857	33857	33857
97-08-06	97-08-06	97-08-06	97-08-06	97-08-06	97-08-06
970806	970806	970806	970806	970806	970806
8/6/97	8/6/97	1997-08-06	08/06/1997	00:00	35 648.00

7.2 Understanding circular references

If there are circular references ("loops") in a data structure, the tables are associated in such a way that there is more than one path of associations between two fields.

This type of data structure should be avoided as much as possible, since it might lead to ambiguities in the interpretation of data.



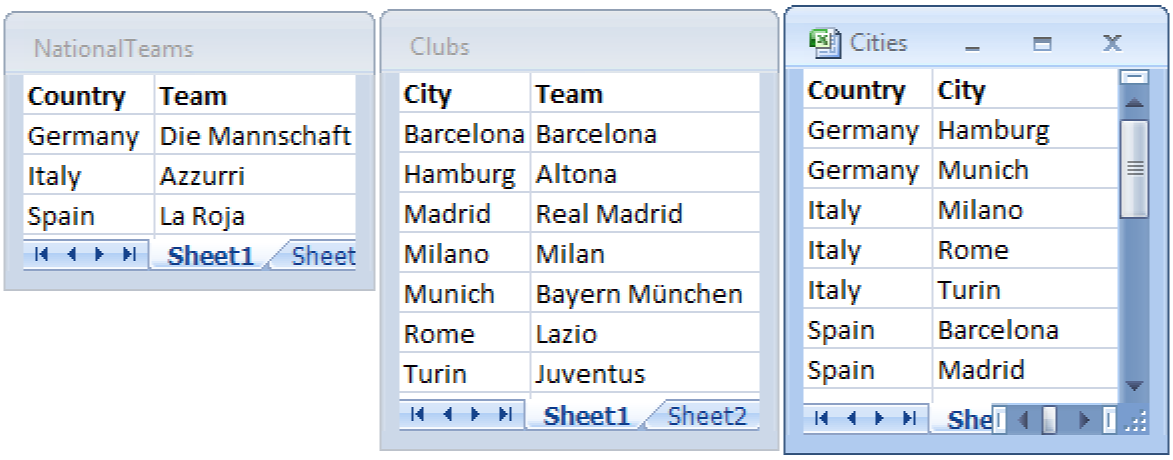
Three tables with a circular reference

Qlik Sense solves the problem of circular references by breaking the loop with a loosely coupled table. When Qlik Sense finds circular data structures while executing the load script, a warning dialog will be shown and one or more tables will be set as loosely coupled. Qlik Sense will typically attempt to loosen the longest table in the loop, as this is often a transaction table, which normally should be the one to loosen. In the data model viewer, loosely-coupled tables are indicated by the red dotted links to other tables.

Example:

Data is loaded from three tables that include the following:

- Names of some national soccer teams
- Soccer clubs in some cities
- Cities of some European countries



NationalTeams

Country	Team
Germany	Die Mannschaft
Italy	Azzurri
Spain	La Roja

Clubs

City	Team
Barcelona	Barcelona
Hamburg	Altona
Madrid	Real Madrid
Milano	Milan
Munich	Bayern München
Rome	Lazio
Turin	Juventus

Cities

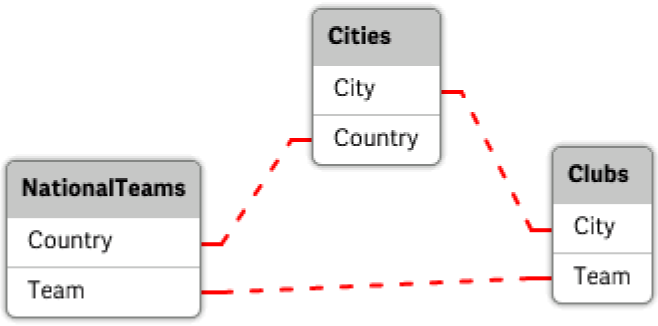
Country	City
Germany	Hamburg
Germany	Munich
Italy	Milano
Italy	Rome
Italy	Turin
Spain	Barcelona
Spain	Madrid

View of the source data tables

This data structure is not very good, since the field name *Team* is used for two different purposes: national teams and local clubs. The data in the tables creates an impossible logical situation.

When loading the tables into Qlik Sense, Qlik Sense determines which of the data connections that is least important, and loosens this table.

Open the **Data model viewer** to see how Qlik Sense interprets the relevance of the data connections:



View of the circular references as noted by red dotted lines

The table with cities and the countries they belong to is now loosely coupled to the table with national teams of different countries and to the table with local clubs of different cities.

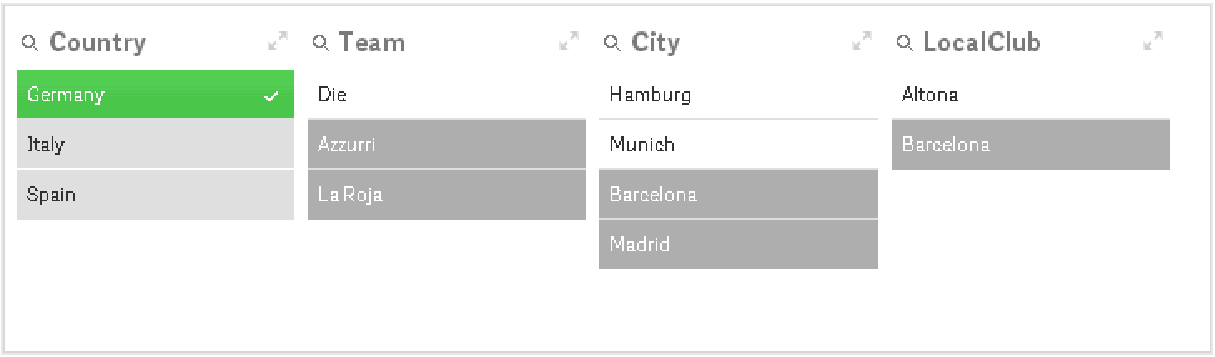
Solving circular references

When circular references occur, you need to edit the data load script by assigning a unique name to one of the fields with identical names.

Do the following:

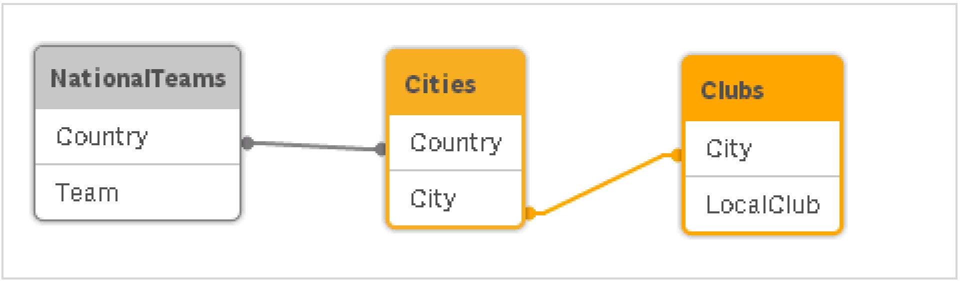
1. Open the data load editor.
2. Edit the **LOAD** statement for one of the duplicate field names.
   In this example, the **LOAD** statement of the table that holds the local teams and their cities would include with a new name for *Team*, for example *LocalClub*. The updated **LOAD** statement reads:
   ```
   LOAD City, Team as LocalClub
   ```
3. Click ⦿ in the toolbar to reload data in the app.

You now have logic that works throughout all the tables. In this example, when *Germany* is selected, the national team, the German cities and the local clubs of each city are associated:



Country	Team	City	LocalClub
Germany ✓	Die	Hamburg	Altona
Italy	Azzurri	Munich	Barcelona
Spain	La Roja	Barcelona	
		Madrid	

When you open the **Data model viewer**, you see that the loosely coupled connections are replaced with regular connections:



7.3 Renaming fields

Sometimes it is necessary to rename fields in order to obtain the desired associations. The three main reasons for renaming fields are:

1. Two fields are named differently although they denote the same thing:
    - The field *ID* in the *Customers* table
    - The field *CustomerID* in the *Orders* table

    The two fields denote a specific customer identification code and should both be named the same, for example *CustomerID*.
2. Two fields are named the same but actually denote different things:
    - The field *Date* in the *Invoices* table
    - The field *Date* in the *Orders* table

    The two fields should preferably be renamed, to for example *InvoiceDate* and *OrderDate*.
3. There may be errors such as misspellings in the database or different conventions on upper- and lowercase letters.

Since fields can be renamed in the script, there is no need to change the original data. There are two different ways to rename fields as shown in the examples.

Example 1: Using the alias statement

The **LOAD** or **SELECT** statement can be preceded by an **alias** statement.

```
Alias ID as CustomerID;
LOAD * from Customer.csv;
```

Example 2: Using the as specifier

The **LOAD** or **SELECT** statement can contain the **as** specifier.

```
LOAD ID as CustomerID, Name, Address, Zip, City, State from Customer.csv;
```

7.4 Concatenating tables

Automatic concatenation

If the field names and the number of fields of two or more loaded tables are exactly the same, Qlik Sense will automatically concatenate the content of the different statements into one table.

Example:

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

The resulting internal table has the fields a, b and c. The number of records is the sum of the numbers of records in table 1 and table 2.

The number and names of the fields must be exactly the same. The order of the two statements is arbitrary.

Forced concatenation

Even if two or more tables do not have exactly the same set of fields, it is still possible to force Qlik Sense to concatenate the two tables. This is done with the **concatenate** prefix in the script, which concatenates a table with another named table or with the last previously created table.

Example:

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

The resulting internal table has the fields a, b and c. The number of records in the resulting table is the sum of the numbers of records in table 1 and table 2. The value of field b in the records coming from table 2 is NULL.

*The number and names of the fields must be exactly the same. Unless a table name of a previously loaded table is specified in the **concatenate** statement the **concatenate** prefix uses the last previously created table. The order of the two statements is thus not arbitrary.*

Preventing concatenation

If the field names and the number of fields of two or more loaded tables are exactly the same, Qlik Sense will automatically concatenate the content of the different statements into one table. This is possible to prevent with a **noconcatenate** statement. The table loaded with the associated **LOAD** or **SELECT** statement will then not be concatenated with the existing table.

Example:

```
LOAD a, b, c from table1.csv;
```

```
noconcatenate LOAD a, b, c from table2.csv;
```

7.5 Loading data from a previously loaded table

You can use the **Resident** predicate in a **LOAD** statement to load data from a previously loaded table. This is useful when you want to perform calculations on data loaded with a **SELECT** statement where you do not have the option to use Qlik Sense functions, such as date or numeric value handling.

Example:

In this example, the date interpretation is performed in the **Resident** load as it can't be done in the initial **Crosstable LOAD**.

```
PreBudget:
Crosstable (Month, Amount, 1)
LOAD Account,
   Jan,
   Feb,
   Mar,
…
From Budget;

Budget:
Noconcatenate
LOAD
   Account,
   Month(Date#(Month,’MMM’)) as Month,
   Amount
Resident PreBudget;

Drop Table PreBudget;
```

*A common case for using **Resident** is where you want to use a temporary table for calculations or filtering. Once you have achieved the purpose of the temporary table, it should be dropped using the **Drop table** statement.*

Resident or preceding LOAD?

In most cases, the same result can be achieved by using a preceding **LOAD** instead, that is, a **LOAD** statement that loads from the **LOAD** or **SELECT** statement below, without specifying a source qualifier such as **From** or **Resident** that you would normally do. A preceding **LOAD** is generally the faster option, but there are some cases where you need to use a **ResidentLOAD** instead:

- If you want to use the **Order_by** clause to sort the records before processing the **LOAD** statement.
- If you want to use any of the following prefixes, in which cases preceding **LOAD** is not supported:
  - **Crosstable**
  - **Join**
  - **Intervalmatch**

Preceding LOAD

The preceding **LOAD** feature allows you to load a table in one pass, but still define several successive transformations. Basically, it is a **LOAD** statement that loads from the **LOAD** or **SELECT** statement below, without specifying a source qualifier such as **From** or **Resident** that you would normally do. You can stack any number of **LOAD** statements this way. The statement at the bottom will be evaluated first, then the statement above, and so on until the top statement has been evaluated.

You can achieve the same result using **Resident**, but in most cases a preceding **LOAD** will be faster.

Another advantage of preceding load is that you can keep a calculation in one place, and reuse it in **LOAD** statements placed above.

*The following prefixes cannot be used in conjunction with preceding **LOAD**:**Join**, **Crosstable** and **Intervalmatch**.*

Example 1: Transforming data loaded by a SELECT statement

If you load data from a database using a **SELECT** statement, you cannot use Qlik Sense functions to interpret data in the **SELECT** statement. The solution is to add a **LOAD** statement, where you perform data transformation, above the **SELECT** statement.

In this example we interpret a date stored as a string using the Qlik Sense function **Date#** in a **LOAD** statement, using the previous **SELECT** statement as source.

```
LOAD Date#(OrderDate,’YYYYMMDD’) as OrderDate;
SQL SELECT OrderDate FROM … ;
```

Example 2: Simplifying your script by reusing calculations

In this example we use a calculation more than once in the script:

```
LOAD  ...,
   Age( FromDate + IterNo() – 1, BirthDate ) as Age,
   Date( FromDate + IterNo() – 1 ) as ReferenceDate
   Resident Policies
      While IterNo() <= ToDate - FromDate + 1 ;
```

By introducing the calculation in a first pass, we can reuse it in the Age function in a preceding **LOAD**:

```
LOAD ..., ReferenceDate,
   Age( ReferenceDate, BirthDate ) as Age;
LOAD *,
   Date( FromDate + IterNo() – 1 ) as ReferenceDate
   Resident Policies
      While IterNo() <= ToDate - FromDate + 1 ;
```

8 Best practices for data modeling

This section describes a number of different ways you can load your data into the Qlik Sense app, depending on how the data is structured and which data model you want to achieve.

In general, the way you load data into the app can be explained by the Extract, Transform and Load process:

1. Extract
   The first step is to extract data from the data source system. In the script, you use **SELECT** or **LOAD** statements to define this. The differences between these statements are:
   - **SELECT** is used to select data from an ODBC data source or OLE DB provider. The **SELECT** SQL statement is evaluated by the data provider, not by Qlik Sense.
   - **LOAD** is used to load data from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically
2. Transform
   The transformation stage involves manipulating the data using script functions and rules to derive the desired data model structure. Typical operations are:
   - Calculating new values
   - Translating coded values
   - Renaming fields
   - Joining tables
   - Aggregating values
   - Pivoting
   - Data validation
3. Load
   In the final step, you run the script to load the data model you have defined into the app.

Your goal should be to create a data model that enables efficient handling of the data in Qlik Sense. Usually this means that you should aim for a reasonably normalized star schema or snowflake schema without any circular references, that is, a model where each entity is kept in a separate table. In other words a typical data model would look like this:

- a central fact table containing keys to the dimensions and the numbers used to calculate measures (such as number of units, sales amounts, and budget amounts).
- surrounding tables containing the dimensions with all their attributes (such as products, customers, categories, calendar, and suppliers) .

In many cases it is possible to solve a task, for example aggregations, either by building a richer data model in the load script, or by performing the aggregations in the chart expressions. As a general rule, you will experience better performance if you keep data transformations in the load script.

It's good practice to sketch out your data model on paper. This will help you by providing structure to what data to extract, and which transformations to perform.

Each table in your data model usually corresponds to either a **SELECT** or **LOAD** statement. The differences between these statements are:

- **SELECT** is used to select data from an ODBC data source or OLE DB provider. The **SELECT** SQL statement is evaluated by the data provider, not by Qlik Sense.
- **LOAD** is used to load data from a file, from data defined in the script, from a previously loaded table, from a web page, from the result of a subsequent **SELECT** statement or by generating data automatically

8.1 Guidelines for data and fields

There are certain conventions and limitations you need to be aware of when working with Qlik Sense. For example: the upper limit for data tables and fields as well as maximum amount of loaded data in Qlik Sense.

Guidelines for amount of loaded data

The amount of data that can be loaded into Qlik Sense is primarily limited by the amount of primary memory of the computer.

Upper limits for data tables and fields

Be aware when building very large apps that a Qlik Sense app cannot have more than 2,147,483,648 distinct values in one field.

The number of fields and data tables as well as the number of table cells and table rows that can be loaded, is limited only by RAM.

Recommended limit for load script sections

The recommended maximum number of characters to be used per load script section is 50,000 characters.

Conventions for number and time formats

In many interpretation and formatting functions it is possible to set the format for numbers and dates by using a format code. This topic describes the conventions used to format a number, date, time or time stamp. These conventions apply both to script and chart functions.

Number formats

- To denote a specific number of digits, use the symbol "0" for each digit.
- To denote a possible digit, use the symbol "#". If the format contains only # symbols to the left of the decimal point, numbers less than 1 begin with a decimal point.
- To mark the position of the thousands separator or the decimal separator, use the applicable thousands separator and the decimal separator.

The format code is used for defining the positions of the separators. It is not possible to set the separator in the format code. Use the **DecimalSep** and **ThousandSep** variables for this in the script.
It is possible to use the thousand separator to group digits by any number of positions, for example, a format string of "0000-0000-0000" (thousand separator="-") could be used to display a twelve-digit part number as "0012-4567-8912".

Examples:

# ##0	describes the number as an integer with a thousands separator.
###0	describes the number as an integer with a thousands separator.
0000	describes the number as an integer with at least four digits. For example, the number 123 will be shown as 0123.
0.000	describes the number with three decimals.
0.0##	describes the number with at least 1 decimal and at most three decimals.

Special number formats

Qlik Sense can interpret and format numbers in any radix between 2 and 36 including binary, octal and hexadecimal. It can also handle roman formats.

Binary format	To indicate binary format the format code should start with (bin) or (BIN).
Octal format	To indicate octal format the format code should start with (oct) or (OCT).
Hexadecimal format	To indicate hexadecimal format the format code should start with (hex) or (HEX). If the capitalized version is used A-F will be used for formatting (for example 14FA). The non-capitalized version will result in formatting with a-f (for example 14fa). Interpretation will work for both variants regardless of the capitalization of the format code.
Decimal format	The use of (dec) or (DEC) to indicate decimal format is permitted but unnecessary.
Custom radix format	To indicate a format in any radix between 2 and 36 the format code should start with (rxx) or (Rxx) where xx is the two-digit number denoting the radix to be used. If the capitalized R is used letters in radices above 10 will be capitalized when Qlik Sense is formatting (for example 14FA). The non-capitalized r will result in formatting with non-capital letters (for example 14fa). Interpretation will work for both variants regardless of the capitalization of the format code. Note that (r02) is the equivalent of (bin), (R16) is the equivalent of (HEX), and so on.
Roman format	To indicate roman numbers the format code should start with (rom) or (ROM). If the capitalized version is used capital letters will be used for formatting (for example MMXVI). The non-capitalized version will result in formatting with lower cap letters (mmxvi). Interpretation will work for both variants regardless of the capitalization of the format code. Roman numbers are generalized with minus sign for negative numbers and 0 for zero. Decimals are ignored with roman formatting.

Examples:

num(199, '(bin)')	returns	11000111
num(199, '(oct)')	returns	307
num(199, '(hex)')	returns	c7
num(199, '(HEX)')	returns	C7
num(199, '(r02)')	returns	11000111
num(199, '(r16)')	returns	c7
num(199, '(R16)')	returns	C7
num(199, '(R36)')	returns	5J
num(199, '(rom)')	returns	cxcix
num(199, '(ROM)')	returns	CXCIX

Dates

You can use the following symbols to format a date. Arbitrary separators can be used.

D	To describe the day, use the symbol "D" for each digit.
M	To describe the month number, use the symbol "M". • Use "M" or "MM" for one or two digits. • "MMM" denotes short month name in letters as defined by the operating system or by the override system variable **MonthNames** in the script. • "MMMM" denotes long month name in letters as defined by the operating system or by the override system variable **LongMonthNames** in the script.
Y	To describe the year, use the symbol "Y" for each digit.
W	To describe the weekday, use the symbol "W". • "W" will return the number of the day (for example 0 for Monday) as a single digit. • "WW" will return the number with two digits (e.g. 02 for Wednesday). • "WWW" will show the short version of the weekday name (for example Mon) as defined by the operating system or by the override system variable **DayNames** in the script. • "WWWW" will show the long version of the weekday name (for example Monday) as defined by the operating system or by the override system variable **LongDayNames** in the script.

Examples: (with 31st March 2013 as example date)

YY-MM-DD	describes the date as 13-03-31.

YYYY-MM-DD	describes the date as 2013-03-31.
YYYY-MMM-DD	describes the date as 2013-Mar-31.
DD MMMM YYYY	describes the date as 31 March 2013.
M/D/YY	describes the date as 3/31/13.
W YY-MM-DD	describes the date as 6 13-03-31.
WWW YY-MM-DD	describes the date as Sat 13-03-31.
WWWW YY-MM-DD	describes the date as Saturday 13-03-31.

Times

You can use the following symbols to format a time. Arbitrary separators can be used.

h	To describe the hours, use the symbol "h" for each digit.
m	To describe the minutes, use the symbol "m" for each digit.
s	To describe the seconds, use the symbol "s" for each digit.
f	To describe the fractions of a second, use the symbol "f" for each digit.
tt	To describe the time in AM/PM format, use the symbol "tt" after the time.

Examples: (with 18.30 as example time):

hh:mm	describes the time as 18:30
hh.mm.ss.ff	describes the time as 18.30.00.00
hh:mm:tt	describes the time as 06:30:pm

Time stamps

The same notation as that of dates and times above is used in time stamps.

Examples: (with 31th March 2013 18.30 as example time stamp):

YY-MM-DD hh:mm	describes the time stamp as 13-03-31 18:30
M/D/Y hh.mm.ss.ffff	describes the time stamp as 3/31/13 18.30.00.0000

8.2 Working with QVD files

A QVD (QlikView Data) file is a file containing a table of data exported from Qlik Sense. QVD is a native Qlik format and can only be written to and read by Qlik Sense or QlikView. The file format is optimized for speed when reading data from a script but it is still very compact. Reading data from a QVD file is typically 10-100 times faster than reading from other data sources.

QVD files can be read in two modes: standard (fast) and optimized (faster). The selected mode is determined automatically by the script engine. Optimized mode can only be employed when all loaded fields are read without any transformations (formulas acting upon the fields), although the renaming of fields is allowed. A **where** clause causing Qlik Sense to unpack the records will also disable the optimized load.

A QVD file holds exactly one data table and consists of three parts:

1. Header.

   If the QVD file was generated with QlikView the header is a well-formed XML header (in UTF-8 char set) describing the fields in the table, the layout of the subsequent information and other metadata.

2. Symbol tables in a byte-stuffed format.
3. Actual table data in a bit-stuffed format.

Purpose of QVD files

QVD files can be used for many purposes. At least four major uses can be easily identified. More than one may apply in any given situation:

Increasing load speed

By buffering non-changing or slowly-changing blocks of input data in QVD files, script execution becomes considerably faster for large data sets.

Decreasing load on database servers

The amount of data fetched from external data sources can also be greatly reduced. This reduces the workload on external databases and network traffic. Furthermore, when several scripts share the same data, it is only necessary to load it once from the source database into a QVD file. Other apps can make use of the same data through this QVD file.

Consolidating data from multiple apps

With the **binary** script statement, data can be loaded from a single app into another app, but with QVD files a script can combine data from any number of apps. This makes it possible for apps to consolidate similar data from different business units, for example.

Incremental load

In many common cases,, the QVD functionality can be used for incremental load by only loading new records from a growing database.

Creating QVD files

A QVD file can be created in two ways:

1. Explicit creation and naming using the **store** command in the script. State in the script that a previously-read table, or part thereof, is to be exported to an explicitly-named file at a location

of your choice.

2. Automatic creation and maintenance from script. By preceding a **LOAD** or **SELECT** statement with the **buffer** prefix, Qlik Sense will automatically create a QVD file, which under certain conditions, can be used instead of the original data source when reloading data.

There is no difference between the resulting QVD files, with regard to reading speed, and so on.

Reading data from QVD files

A QVD file can be read into or accessed by the following methods:

1. Loading a QVD file as an explicit data source. QVD files can be referenced by a **LOAD** statement in the script, just like any other type of text files (csv, fix, dif, biff etc).

Example:

```
LOAD * from xyz.qvd (qvd);
LOAD Name, RegNo from xyz.qvd (qvd);
LOAD Name as a, RegNo as b from xyz.qvd (qvd);
```

2. Automatic loading of buffered QVD files. When using the **buffer** prefix on **LOAD** or **SELECT** statements, no explicit statements for reading are necessary. Qlik Sense will determine the extent to which it will use data from the QVD file as opposed to acquiring data using the original **LOAD** or **SELECT** statement.
3. Accessing QVD files from the script. A number of script functions (all beginning with **qvd**) can be used for retrieving various information on the data found in the XML header of a QVD file.

8.3 Using QVD files for incremental load

The incremental load task is very often used with data bases. An incremental load only loads new or changed records from the database, all other data should already be available in the app. It is almost always possible to use incremental load with QVD files.

The basic process is described below:

1. Load new data from the database table.
   This is a slow process, but only a limited number of records are loaded.
2. Load old data from the QVD file.
   Many records are loaded, but this is a much faster process.
3. Create a new QVD file.
4. Repeat the procedure for every table loaded.

The following examples show cases where incremental load is used, however, a more complex solution might be necessary, if the source database requires.

- Append only (typically used for log files)
- Insert only (no update or delete)

- Insert and update (no delete)
- Insert, update and delete

Here are the outlines of solutions for each of the these cases. You can read QVD files in either optimized mode or standard mode. (The method employed is automatically selected by the Qlik Sense engine depending on the complexity of the operation.) Optimized mode is about 10 times faster than standard mode, or about 100 times faster than loading the database in the ordinary fashion.

Append only

The simplest case is the one of log files; files in which records are only appended and never deleted. The following conditions apply:

- The database must be a log file (or some other file in which records are appended and not inserted or deleted) which is contained in a text file (ODBC, OLE DB or other databases are not supported).
- Qlik Sense keeps track of the number of records that have been previously read and loads only records added at the end of the file.

Example:

```
Buffer (Incremental) Load * From LogFile.txt (ansi, txt, delimiter is '\t', embedded labels);
```

Insert only (no update or delete)

If the data resides in a database other than a simple log file, the append approach will not work. However, the problem can still be solved with a minimum amount of extra work. The following conditions apply:

- The data source can be any database.
- Qlik Sense loads records inserted in the database after the last script execution.
- A ModificationTime field (or similar) is required for Qlik Sense to recognize which records are new.

Example:

```
QV_Table:
SQL SELECT PrimaryKey, X, Y FROM DB_TABLE
WHERE ModificationTime >= #$(LastExecTime)#
AND ModificationTime < #$(BeginningThisExecTime)#;

Concatenate LOAD PrimaryKey, X, Y FROM File.QVD;
STORE QV_Table INTO File.QVD;
```

The hash signs in the SQL WHERE clause define the beginning and end of a date. Check your database manual for the correct date syntax for your database.

Insert and update (no delete)

The next case is applicable when data in previously loaded records may have changed between script executions. The following conditions apply:

- The data source can be any database.
- Qlik Sense loads records inserted into the database or updated in the database after the last script

execution
- A ModificationTime field (or similar) is required for Qlik Sense to recognize which records are new.
- A primary key field is required for Qlik Sense to sort out updated records from the QVD file.
- This solution will force the reading of the QVD file to standard mode (rather than optimized), which is still considerably faster than loading the entire database.

Example:

```
QV_Table:
SQL SELECT PrimaryKey, X, Y FROM DB_TABLE
WHERE ModificationTime >= #$(LastExecTime)#;

Concatenate LOAD PrimaryKey, X, Y FROM File.QVD
WHERE NOT Exists(PrimaryKey);

STORE QV_Table INTO File.QVD;
```

Insert, update and delete

The most difficult case to handle is when records are actually deleted from the source database between script executions. The following conditions apply:

- The data source can be any database.
- Qlik Sense loads records inserted into the database or updated in the database after the last script execution.
- Qlik Sense removes records deleted from the database after the last script execution.
- A field ModificationTime (or similar) is required for Qlik Sense to recognize which records are new.
- A primary key field is required for Qlik Sense to sort out updated records from the QVD file.
- This solution will force the reading of the QVD file to standard mode (rather than optimized), which is still considerably faster than loading the entire database.

Example:

```
Let ThisExecTime = Now( );

QV_Table:
SQL SELECT PrimaryKey, X, Y FROM DB_TABLE
WHERE ModificationTime >= #$(LastExecTime)#
AND ModificationTime < #$(ThisExecTime)#;

Concatenate LOAD PrimaryKey, X, Y FROM File.QVD
WHERE NOT EXISTS(PrimaryKey);

Inner Join SQL SELECT PrimaryKey FROM DB_TABLE;

If ScriptErrorCount = 0 then
STORE QV_Table INTO File.QVD;
Let LastExecTime = ThisExecTime;
End If
```

8.4 Combining tables with Join and Keep

A join is an operation that uses two tables and combines them into one. The records of the resulting table are combinations of records in the original tables, usually in such a way that the two records contributing to any given combination in the resulting table have a common value for one or several common fields, a so-called natural join. In Qlik Sense, joins can be made in the script, producing logical tables.

It is possible to join tables already in the script. The Qlik Sense logic will then not see the separate tables, but rather the result of the join, which is a single internal table. In some situations this is needed, but there are disadvantages:

- The loaded tables often become larger, and Qlik Sense works slower.
- Some information may be lost: the frequency (number of records) within the original table may no longer be available.

The **Keep** functionality, which has the effect of reducing one or both of the two tables to the intersection of table data before the tables are stored in Qlik Sense, has been designed to reduce the number of cases where explicit joins needs to be used.

In this documentation, the term join is usually used for joins made before the internal tables are created. The association, made after the internal tables are created, is however essentially also a join.

Joins within an SQL SELECT statement

With some ODBC drivers it is possible to make a join within the **SELECT** statement. This is almost equivalent to making a join using the **Join** prefix.

However, most ODBC drivers are not able to make a full (bidirectional) outer join. They are only able to make a left or a right outer join. A left (right) outer join only includes combinations where the joining key exists in the left (right) table. A full outer join includes any combination. Qlik Sense automatically makes a full outer join.

Further, making joins in **SELECT** statements is far more complicated than making joins in Qlik Sense.

Example:

```
SELECT DISTINCTROW
[Order Details].ProductID, [Order Details].
UnitPrice, Orders.OrderID, Orders.OrderDate, Orders.CustomerID
FROM Orders
RIGHT JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID;
```

This **SELECT** statement joins a table containing orders to a fictive company, with a table containing order details. It is a right outer join, meaning that all the records of *OrderDetails* are included, also the ones with an *OrderID* that does not exist in the table *Orders*. Orders that exist in *Orders* but not in *OrderDetails* are however not included.

Join

The simplest way to make a join is with the **Join** prefix in the script, which joins the internal table with another named table or with the last previously created table. The join will be an outer join, creating all possible combinations of values from the two tables.

Example:

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

The resulting internal table has the fields a, b, c and d. The number of records differs depending on the field values of the two tables.

*The names of the fields to join over must be exactly the same. The number of fields to join over is arbitrary. Usually the tables should have one or a few fields in common. No field in common will render the cartesian product of the tables. All fields in common is also possible, but usually makes no sense. Unless a table name of a previously loaded table is specified in the **Join** statement the **Join** prefix uses the last previously created table. The order of the two statements is thus not arbitrary.*

Keep

The explicit **Join** prefix in the data load script performs a full join of the two tables. The result is one table. In many cases such joins will results in very large tables. One of the main features of Qlik Sense is its ability to make associations between tables instead of joining them, which reduces space in memory, increases speed and gives enormous flexibility. The keep functionality has been designed to reduce the number of cases where explicit joins need to be used.

The **Keep** prefix between two **LOAD** or **SELECT** statements has the effect of reducing one or both of the two tables to the intersection of table data before they are stored in Qlik Sense. The **Keep** prefix must always be preceded by one of the keywords **Inner**, **Left** or **Right**. The selection of records from the tables is made in the same way as in a corresponding join. However, the two tables are not joined and will be stored in Qlik Sense as two separately named tables.

Inner

The **Join** and **Keep** prefixes in the data load script can be preceded by the prefix **Inner**.

If used before **Join**, it specifies that the join between the two tables should be an inner join. The resulting table contains only combinations between the two tables with a full data set from both sides.

If used before **Keep**, it specifies that the two tables should be reduced to their common intersection before being stored in Qlik Sense.

Example:

In these examples we use the source tables Table1 and Table2:

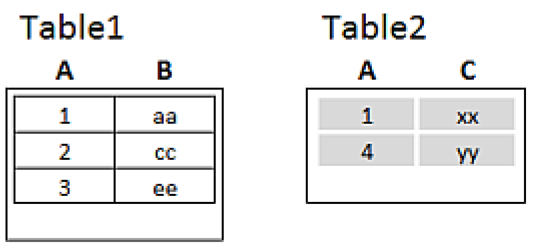


Table1

A	B
1	aa
2	cc
3	ee

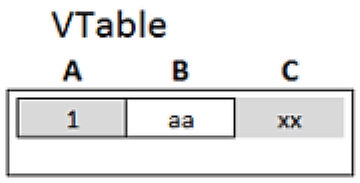
Table2

A	C
1	xx
4	yy

Inner examples source tables

First, we perform an **Inner Join** on the tables, resulting in VTable, containing only one row, the only record existing in both tables, with data combined from both tables.

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



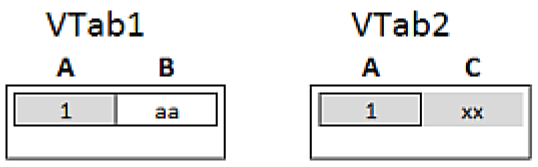
VTable

A	B	C
1	aa	xx

Inner Join example

If we perform an **Inner Keep** instead, you will still have two tables. The two tables are of course associated via the common field A.

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



VTab1

A	B
1	aa

VTab2

A	C
1	xx

Inner Keep example

Left

The **Join** and **Keep** prefixes in the data load script can be preceded by the prefix **left**.

If used before **Join**, it specifies that the join between the two tables should be a left join. The resulting table only contains combinations between the two tables with a full data set from the first table.

If used before **Keep**, it specifies that the second table should be reduced to its common intersection with the first table before being stored in Qlik Sense.

Example:

In these examples we use the source tables Table1 and Table2:

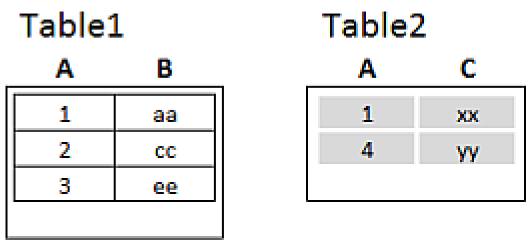


Table1

A	B
1	aa
2	cc
3	ee

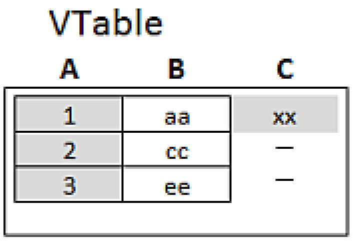
Table2

A	C
1	xx
4	yy

Left examples source tables

First, we perform a **Left Join** on the tables, resulting in VTable, containing all rows from Table1, combined with fields from matching rows in Table2.

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



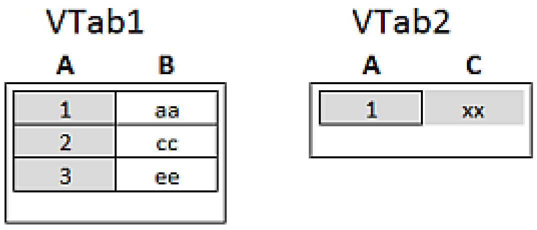
VTable

A	B	C
1	aa	xx
2	cc	–
3	ee	–

Left Join example

If we perform an **Left Keep** instead, you will still have two tables. The two tables are of course associated via the common field A.

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



VTab1

A	B
1	aa
2	cc
3	ee

VTab2

A	C
1	xx

Left Keep example

Right

The **Join** and **Keep** prefixes in the data load script can be preceded by the prefix **right**.

If used before **Join**, it specifies that the join between the two tables should be a right join. The resulting table only contains combinations between the two tables with a full data set from the second table.

If used before **Keep**, it specifies that the first table should be reduced to its common intersection with the second table before being stored in Qlik Sense.

Example:

In these examples we use the source tables Table1 and Table2:

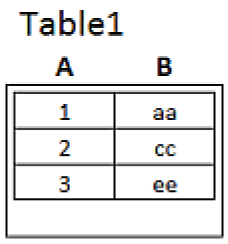


Table1

A	B
1	aa
2	cc
3	ee

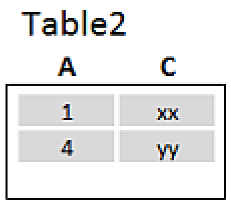


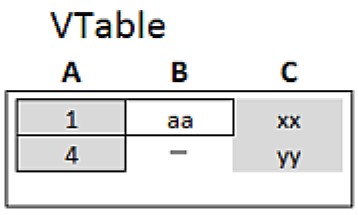
Table2

A	C
1	xx
4	yy

Right examples source tables

First, we perform a **Right Join** on the tables, resulting in VTable, containing all rows from Table2, combined with fields from matching rows in Table1.

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



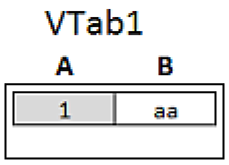
VTable

A	B	C
1	aa	xx
4	–	yy

Right Join example

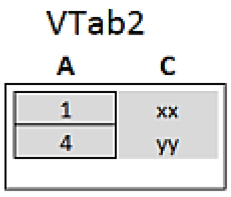
If we perform an **Left Keep** instead, you will still have two tables. The two tables are of course associated via the common field A.

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



VTab1

A	B
1	aa



VTab2

A	C
1	xx
4	yy

Right Keep example

8.5 Using mapping as an alternative to joining

The **Join** prefix in Qlik Sense is a powerful way of combining several data tables in the data model. One disadvantage is that the combined tables can become large and create performance problems. An alternative to **Join** in situations where you need to look up a single value from another table is to use mapping instead. This can save you from loading unnecessary data that slows down calculations and potentially can create calculation errors, as joins can change the number of records in the tables.

A mapping table consists of two columns; a comparison field (input) and a mapping value field (output).

In this example we have an table of orders (Orders), and need to know the country of the customer, which is stored in the customer table (Customers).

Orders data table

OrderID	OrderDate	ShipperID	Freight	CustomerID
12987	2007-12-01	1	27	3
12988	2007-12-01	1	65	4
12989	2007-12-02	2	32	2
12990	2007-12-03	1	76	3

Customers data table

CustomerID	Name	Country	...
1	DataSales	Spain	...
2	BusinessCorp	Italy	...
3	TechCo	Germany	...
4	Mobecho	France	...

In order to look up the country (Country) of a customer, we need a mapping table that looks like this:

CustomerID	Country
1	Spain
2	Italy
3	Germany
4	France

The mapping table, which we name MapCustomerIDtoCountry, is defined in the script as follows:

```
    MapCustomerIDtoCountry:
    Mapping LOAD CustomerID, Country From Customers ;
```

The next step is to apply the mapping, using the **ApplyMap** function, when loading the order table:

```
Orders:
    S *,
        ApplyMap('MapCustomerIDtoCountry', CustomerID, null()) as Country
        From Orders ;
```

The third parameter of the **ApplyMap** function is used to define what to return when the value is not found in the mapping table, in this case **Null()**.

The resulting table will look like this:

OrderID	OrderDate	ShipperID	Freight	CustomerID	Country
12987	2007-12-01	1	27	3	Germany
12988	2007-12-01	1	65	4	France
12989	2007-12-02	2	32	2	Italy
12990	2007-12-03	1	76	3	Germany

8.6 Working with cross tables

A cross table is a common type of table featuring a matrix of values between two orthogonal lists of header data. It could look like the table below.

Example 1:

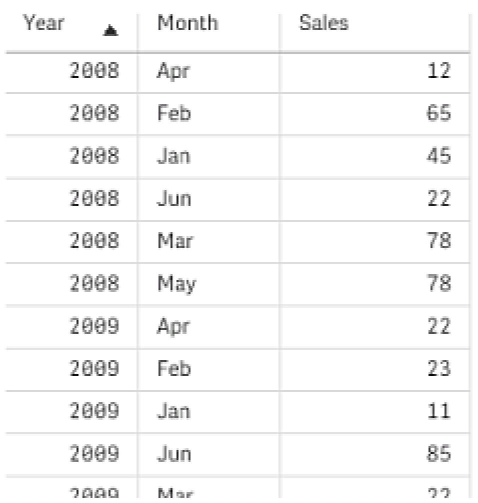
Year	Jan	Feb	Mar	Apr	May	Jun
2008	45	65	78	12	78	22
2009	11	23	22	22	45	85
2010	65	56	22	79	12	56
2011	45	24	32	78	55	15
2012	45	56	35	78	68	82

If this table is simply loaded into Qlik Sense, the result will be one field for *Year* and one field for each of the months. This is generally not what you would like to have. You would probably prefer to have three fields generated: one for each header category (*Year* and *Month*) and one for the data values inside the matrix.

This can be achieved by adding the **crosstable** prefix to the **LOAD** or **SELECT** statement, for example:

```
crosstable (Month, Sales) LOAD * from ex1.xlsx;
```

This creates the following result in Qlik Sense:



Year ▲	Month	Sales
2008	Apr	12
2008	Feb	65
2008	Jan	45
2008	Jun	22
2008	Mar	78
2008	May	78
2009	Apr	22
2009	Feb	23
2009	Jan	11
2009	Jun	85
2009	Mar	22

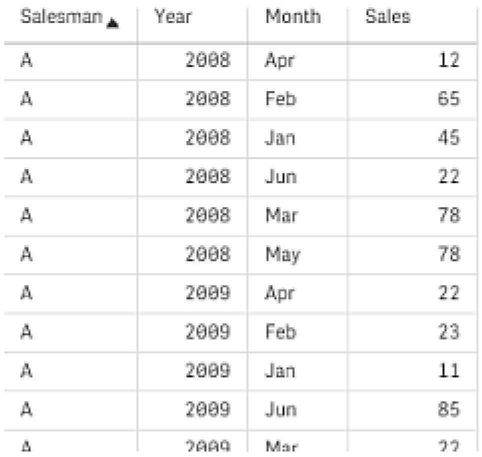
The cross table is often preceded by a number of qualifying columns, which should be read in a straightforward way. In this case there is one qualifying column, Year:

Example 2:

Salesman	Year	Jan	Feb	Mar	Apr	May	Jun
A	2008	45	65	78	12	78	22
A	2009	11	23	22	22	45	85
A	2010	65	56	22	79	12	56
A	2011	45	24	32	78	55	15
A	2012	45	56	35	78	68	82
B	2008	57	77	90	24	90	34
B	2009	23	35	34	34	57	97
B	2010	77	68	34	91	24	68
B	2011	57	36	44	90	67	27
B	2012	57	68	47	90	80	94

In this case there are two qualifying columns to the left, followed by the matrix columns. The number of qualifying columns can be stated as a third parameter to the **crosstable** prefix as follows:

```
crosstable (Month, Sales, 2) LOAD * from ex2.xlsx;
```
This creates the following result in Qlik Sense:



Salesman ▲	Year	Month	Sales
A	2008	Apr	12
A	2008	Feb	65
A	2008	Jan	45
A	2008	Jun	22
A	2008	Mar	78
A	2008	May	78
A	2009	Apr	22
A	2009	Feb	23
A	2009	Jan	11
A	2009	Jun	85
A	2009	Mar	22

8.7 Generic databases

A generic database is a table in which the field names are stored as field values in one column, while the field values are stored in a second. Generic databases are usually used for attributes of different objects.

Look at the example GenericTable below. It is a generic database containing two objects, a ball and a box. Obviously some of the attributes, like color and weight, are common to both the objects, while others, like diameter, height, length and width are not.

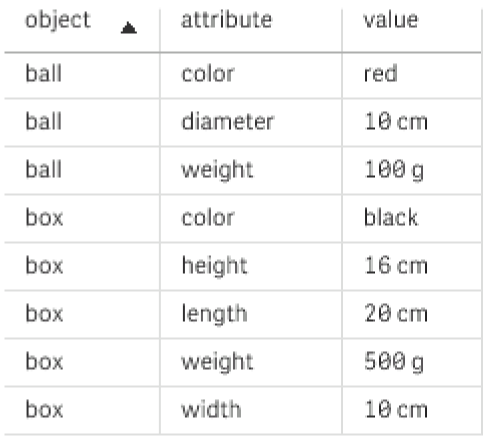
GenericTable

object	attribute	value
ball	color	red
ball	diameter	10 cm
ball	weight	100 g
box	color	black
box	height	16 cm
box	length	20 cm
box	weight	500 g
box	width	10 cm

On one hand it would be awkward to store the data in a way giving each attribute a column of its own, since many of the attributes are not relevant for a specific object.

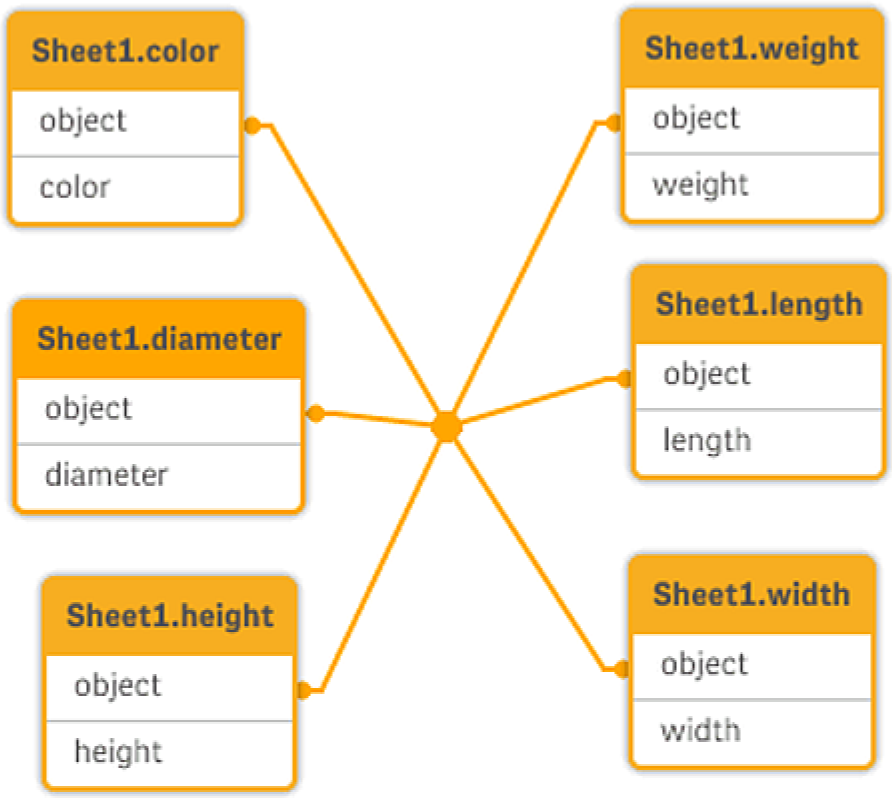
On the other hand, it would look messy displaying it in a way that mixed lengths, colors and weights.

If this database is loaded into Qlik Sense using the standard way and display the data in a table it looks like this:



object ▲	attribute	value
ball	color	red
ball	diameter	10 cm
ball	weight	100 g
box	color	black
box	height	16 cm
box	length	20 cm
box	weight	500 g
box	width	10 cm

However, if the table is loaded as a generic database, column two and three will be split up into different tables, one for each unique value of the second column:



The syntax for doing this is simple:

Example:

```
Generic SELECT* from GenericTable;
```

It does not matter whether a **LOAD** or **SELECT** statement is used to load the generic database.

8.8 Matching intervals to discrete data

The **intervalmatch** prefix to a **LOAD** or **SELECT** statement is used to link discrete numeric values to one or more numeric intervals. This is a very powerful feature which can be used, for example, in production environments as shown in the example below.

Example:

Look at the two tables below. The first table shows the start and end of production of different orders. The second table shows some discrete events. How can we associate the discrete events with the orders, so that we know, for example, which orders were affected by the disturbances and which orders were processed by which shifts?

Table OrderLog

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

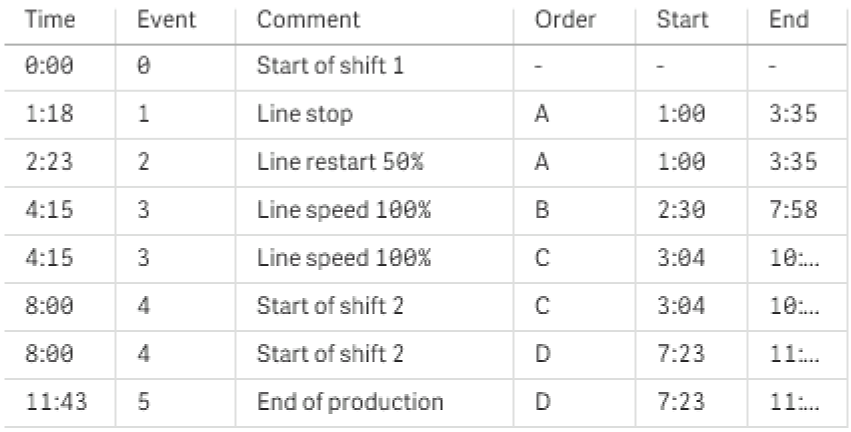
Table EventLog

Time	Event	Comment
00:00	0	Start of shift 1
01:18	1	Line stop
02:23	2	Line restart 50%
04:15	3	Line speed 100%
08:00	4	Start of shift 2
11:43	5	End of production

First, load the two tables as usual and then link the field *Time* to the intervals defined by the fields *Start* and *End*:

```
SELECT * from OrderLog;
SELECT * from EventLog;
Intervalmatch (Time) SELECT Start,End from OrderLog;
```

You can now create a table in Qlik Sense as below:



Time	Event	Comment	Order	Start	End
0:00	0	Start of shift 1	-	-	-
1:18	1	Line stop	A	1:00	3:35
2:23	2	Line restart 50%	A	1:00	3:35
4:15	3	Line speed 100%	B	2:30	7:58
4:15	3	Line speed 100%	C	3:04	10:...
8:00	4	Start of shift 2	C	3:04	10:...
8:00	4	Start of shift 2	D	7:23	11:...
11:43	5	End of production	D	7:23	11:...

We can now easily see that mainly order *A* was affected by the line stop but that the reduced line speed affected also orders *B* and *C*. Only the orders *C* and *D* were partly handled by *Shift 2*.

Note the following points when using **intervalmatch**:

- Before the **intervalmatch** statement, the field containing the discrete data points (*Time* in the example above) must already have been read into Qlik Sense. The **intervalmatch** statement does not read this field from the database table!

- The table read in the **intervalmatch LOAD** or **SELECT** statement must always contain exactly two fields (*Start* and *End* in the example above). In order to establish a link to other fields you must read the interval fields together with additional fields in a separate **LOAD** or **SELECT** statement (the first **SELECT** statement in the example above).
- The intervals are always closed, that is, the end points are included in the interval. Non-numeric limits render the interval to be disregarded (undefined) while NULL limits extend the interval indefinitely (unlimited).
- The intervals may be overlapping and the discrete values will be linked to all matching intervals.

Using the extended **intervalmatch** syntax to resolve slowly changing dimension problems

The extended **intervalmatch** syntax can be used for handling of the well-known problem of slowly changing dimensions in source data.

Sample script:

```
SET NullInterpret='';

IntervalTable:
LOAD Key, ValidFrom, Team
FROM 'lib://dataqv/intervalmatch.xlsx' (ooxml, embedded labels, table is IntervalTable);

Key:
LOAD
Key,
ValidFrom as FirstDate,
date(if(Key=previous(Key),
previous(ValidFrom) - 1)) as LastDate,
Team
RESIDENT IntervalTable order by Key, ValidFrom desc;

drop table IntervalTable;

Transact:
LOAD Key, Name, Date, Sales
FROM 'lib://dataqv/intervalmatch.xlsx' (ooxml, embedded labels, table is Transact);

INNER JOIN intervalmatch (Date,Key) LOAD FirstDate, LastDate, Key RESIDENT Key;
```

The **nullinterpret** statement is only required when reading data from a table file since missing values are defined as empty strings instead of NULL values.

Loading the data from *IntervalTable* would result in the following table:

Key	FirstDate	Team
000110	2011-01-21	Southwest
000110		Northwest
000120		Northwest
000120	2013-03-05	Southwest
000120	2013-03-05	Northwest
000120	2013-01-06	Southwest

The **nullasvalue** statement allows NULL values to map to the listed fields.

Create *Key*, *FirstDate*, *LastDate*, (attribute fields) by using **previous** and **order by** and thereafter the *IntervalTable* is dropped having been replaced by this key table.

Loading the data from *Transact* would result in the following table:

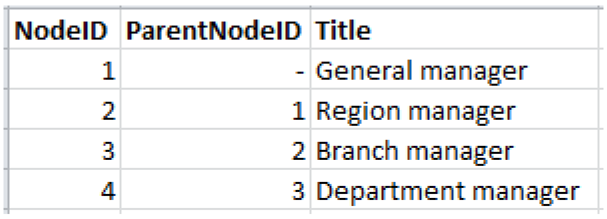
Key	Name	Date	Sales
000110	Spengler Aaron	2009-08-18	100
000110	Spengler Aaron	2009-12-25	200
000110	Spengler Aaron	2011-02-03	300
000110	Spengler Aaron	2011-05-05	400
000120	Ballard John	2011-06-04	500
000120	Ballard John	2013-01-20	600
000120	Ballard John	2013-03-10	700
000120	Ballard John	2013-03-13	800
000120	Ballard John	2013-09-21	900

The **intervalmatch** statement preceded by the **inner join** replaces the key above with a synthetic key that connects to the *Transact* table resulting in the following table:

Key	Team	Name	FirstDate	LastDate	Date	Sales
000110	Northwest	Spengler Aaron		2011-01-20	2009-08-18	100
000110	Northwest	Spengler Aaron		2011-01-20	2009-12-25	200
000110	Southwest	Spengler Aaron	2011-01-21		2011-02-03	300
000110	Southwest	Spengler Aaron	2011-01-21		2011-05-05	400
000120	Northwest	Ballard John		2013-01-05	2011-06-04	500
000120	Southwest	Ballard John	2013-01-06	2013-03-04	2013-01-20	600
000120	Southwest	Ballard John	2013-03-05		2013-03-10	700
000120	Southwest	Ballard John	2013-03-05		2013-03-13	800
000120	Southwest	Ballard John	2013-03-05		2013-09-21	900

8.9 Hierarchies

Unbalanced *n*-level hierarchies are often used to represent among other things, geographical or organizational dimensions in data. These types of hierarchies are usually stored in an adjacent nodes table, that is, in a table where each record corresponds to a node and has a field that contains a reference to the parent node.



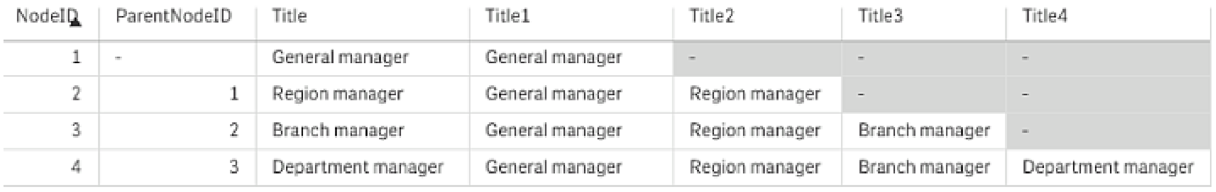
NodeID	ParentNodeID	Title
1	-	General manager
2	1	Region manager
3	2	Branch manager
4	3	Department manager

In such a table, the node is stored on one record only but can still have any number of children. The table may of course contain additional fields describing attributes for the nodes.

An adjacent nodes table is optimal for maintenance, but difficult to use in everyday work. Instead, in queries and analysis, other representations are used. The expanded nodes table is one common representation, where each level in the hierarchy is stored in a separate field. The levels in an expanded nodes table can easily be used e.g. in a tree structure.The **hierarchy** keyword can be used in the data load script to transform an adjacent nodes table to an expanded nodes table.

Example:

```
Hierarchy (NodeID, ParentNodeID, Title, 'Manager') LOAD
    NodeID,
    ParentNodeID,
    Title
FROM 'lib://data/hierarchy.txt' (txt, codepage is 1252, embedded labels, delimiter is ',', msq);
```



NodeID	ParentNodeID	Title	Title1	Title2	Title3	Title4
1	-	General manager	General manager	-	-	-
2	1	Region manager	General manager	Region manager	-	-
3	2	Branch manager	General manager	Region manager	Branch manager	-
4	3	Department manager	General manager	Region manager	Branch manager	Department manager

A problem with the expanded nodes table is that it is not easy to use the level fields for searches or selections, since a prior knowledge is needed about which level to search or select in. The ancestors table is a different representation that solves this problem. This representation is also called a bridge table.

The ancestors table contains one record for every child-ancestor relation found in the data. It contains keys and names for the children as well as for the ancestors, that is, every record describes which node a specific node belongs to. The **hierarchybelongsto** keyword can be used in the data load script to transform an adjacent nodes table to an ancestors table.

8.10 Dollar-sign expansions

Dollar-sign expansions are definitions of text replacements used in the script or in expressions. This process is known as expansion - even if the new text is shorter. The replacement is made just before the script statement or the expression is evaluated. Technically it is a macro expansion.

The expansion always begins with '$(' and ends with ') ' and the content between brackets defines how the text replacement will be done. To avoid confusion with script macros we will henceforth refer to macro expansions as dollar-sign expansions.

Dollar-sign expansions can be used with either of:

- variables
- parameters
- expressions

A dollar-sign expansion is limited in how many expansions it can calculate. Any expansion over 1000 will not be calculated.

Dollar-sign expansion using a variable

When using a variable for text replacement in the script or in an expression, the following syntax is used:

```
$ (variablename)
```

$*(variablename)* expands to the value in the variable. If *variablename* does not exist, the expansion will result in an empty string.

For numeric variable expansions, the following syntax is used:

```
$ (#variablename)
```

It always yields a valid decimal-point representation of the numeric value of the variable, possibly with exponential notation (for very large/small numbers). If *variablename* does not exist or does not contain a numeric value, it will be expanded to *0* instead.

Example:

```
SET DecimalSep=',';
LET X = 7/2;
```

The dollar-sign expansion **$(X)** will expand to *3,5* while **$(#X)** will expand to *3.5*.

Example:

```
Set Mypath=C:\MyDocs\Files\;
...
LOAD * from $(MyPath)abc.csv;
```

Data will be loaded from *C:\MyDocs\Files\abc.csv*.

Example:

```
Set CurrentYear=1992;
...
SQL SELECT * FROM table1 WHERE Year=$(CurrentYear);
```

Rows with Year=1992 will be selected.

Example:

```
Set vConcatenate = ;
For each vFile in FileList('.\*.txt')
   Data:
   $(vConcatenate)
   LOAD * FROM [$(vFile)];
   Set vConcatenate = Concatenate ;
Next vFile
```

In this example, all .txt files in the directory are loaded using the **Concatenate** prefix. This may be required if the fields differ slightly, in which case auto-concatenation does not work. The vConcatenate variable is initially set to an empty string, as the **Concatenate** prefix cannot be used on the first load. If the directory contains three files named *file1.txt*, *file2.txt* and *file3.txt*, the **LOAD** statement will during the three iterations expand to:

```
LOAD * FROM[.\file1.txt];
Concatenate LOAD * FROM[.\file2.txt];
Concatenate LOAD * FROM[.\file3.txt];
```

Dollar-sign expansion using parameters

Parameters can be used in dollar-sign expansions. The variable must then contain formal parameters, such as $1, $2, $3 etc. When expanding the variable, the parameters should be stated in a comma separated list.

Example:

```
Set MUL='$1*$2';
Set X=$(MUL(3,7)); // returns '3*7' in X

Let X=$(MUL(3,7)); // returns 21 in X
```

If the number of formal parameters exceeds the number of actual parameters only the formal parameters corresponding to actual parameters will be expanded. If the number of actual parameters exceeds the number of formal parameters the superfluous actual parameters will be ignored.

Example:

```
Set MUL='$1*$2';
Set X=$(MUL); // returns '$1*$2' in X

Set X=$(MUL(10)); // returns '10*$2' in X

Let X=$(MUL(5,7,8)); // returns 35 in X
```

The parameter $0 returns the number of parameters actually passed by a call.

Example:

```
set MUL='$1*$2 $0 par';
set X=$(MUL(3,7)); // returns '3*7 2 par' in X
```

Dollar-sign expansion using an expression

Expressions can be used in dollar-sign expansions. The content between the brackets must then start with an equal sign:

```
$(=expression )
```

The expression will be evaluated and the value will be used in the expansion.

Example:

```
$(=Year(Today())); // returns a string with the current year.

$(=Only(Year)-1); // returns the year before the selected one.
```

File inclusion

File inclusions are made using dollar-sign expansions. The syntax is:

```
$(include=filename )
```

The above text will be replaced by the content of the file specified after the equal sign. This feature is very useful when storing scripts or parts of scripts in text files.

Example:

```
$(include=C:\Documents\MyScript.qvs);
```

8.11 Using quotation marks in the script

You can use quotation marks in script statements in a number of different ways.

Inside LOAD statements

In a **LOAD** statement the following symbols should be used as quotation marks:

	Description	Symbol	ASCII	Example
Field names	double quotation marks	" "	34	"string"
	square brackets	[ ]	91, 93	[string]
	grave accents	` `	96	`string`
String literals	single quotation marks	' '	39	'string'

In SELECT statements

For a **SELECT** statement interpreted by the ODBC driver, it may be slightly different. Usually, you should use the straight double quotation marks (Alt + 0034) for field and table names, and the straight single quotation marks (Alt + 0039) for literals, and avoid using grave accents. However, some ODBC drivers not only accept grave accents as quotation marks, but also prefer them. In such a case, the generated **SELECT** statements contain grave accent quotation marks.

Microsoft Access quotation marks example

Microsoft Access ODBC Driver 3.4 (included in Microsoft Access 7.0) accepts the following quotation marks when analyzing the **SELECT** statement:

Field names and table names:	`[ ]` `" "` `` ` ` ``
String literals:	`' '`

Other databases may have different conventions.

Outside LOAD statements

Outside a **LOAD** statement, in places where Qlik Sense expects an expression, double quotation marks denote a variable reference and not a field reference. If you use double quotation marks, the enclosed string will be interpreted as a variable and the value of the variable will be used.

Out-of-context field references and table references

Some script functions refer to fields that have already been created, or are in the output of a **LOAD** statement, for example **Exists()** and **Peek()**. These field references are called out-of-context field references, as opposed to source field references that refer to fields that are in context, that is, in the input table of the **LOAD** statement.

Out-of-context field references and table references should be regarded as literals and therefore need single quotation marks.

Difference between names and literals

The difference between names and literals becomes clearer comparing the following examples:

Example:

```
'Sweden' as Country
```

When this expression is used as a part of the field list in a **LOAD** or **SELECT** statement, the text string "*Sweden*" will be loaded as field value into the Qlik Sense field "*Country*".

Example:

```
"land" as Country
```

When this expression is used as a part of the field list in a **LOAD** or **SELECT** statement, the content of the database field or table column named "*land*" will be loaded as field values into the Qlik Sense field "*Country*". This means. that *land* will be treated as a field reference.

Difference between numbers and string literals

The difference between numbers and string literals becomes clearer comparing the following examples.

Example:

```
'12/31/96'
```

When this string is used as a part of an expression, it will in a first step be interpreted as the text string "12/31/96", which in turn may be interpreted as a date if the date format is ‘MM/DD/YY’. In that case it will be stored as a dual value with both a numeric and a textual representation.

Example:

```
12/31/96
```

When this string is used as a part of an expression, it will be interpreted numerically as 12 divided by 31 divided by 96.

8.12 Data cleansing

When loading data from different tables, note that field values denoting the same thing are not always consistently named. Since this lack of consistency is not only annoying, but also hinders associations, the problem needs to be solved. This can be done in an elegant way by creating a mapping table for the comparison of field values.

Mapping tables

Tables loaded via **mapping load** or **mapping select** are treated differently from other tables. They will be stored in a separate area of the memory and used only as mapping tables during script execution. After the script execution they will be automatically dropped.

Rules:

- A mapping table must have two columns, the first one containing the comparison values and the second the desired mapping values.
- The two columns must be named, but the names have no relevance in themselves. The column names have no connection to field names in regular internal tables.

Using a mapping table

When loading several tables listing countries, you may find that one and the same country has several different names. In this example, the U.S.A. are listed as US, U.S., and United States.

To avoid the occurrence of three different records denoting the United States in the concatenated table, create a table similar to that shown and load it as a mapping table.

The entire script should have the following appearance:

```
CountryMap:
Mapping LOAD x,y from MappingTable.txt
(ansi, txt, delimiter is ',', embedded
labels);
Map Country using CountryMap;
LOAD Country,City from CountryA.txt
(ansi, txt, delimiter is ',', embedded labels);
LOAD Country, City from CountryB.txt
(ansi, txt, delimiter is ',', embedded labels);
```

The **mapping** statement loads the file *MappingTable.txt* as a mapping table with the label *CountryMap*.

The **map** statement enables mapping of the field *Country* using the previously loaded mapping table *CountryMap*.

The **LOAD** statements load the tables *CountryA* and *CountryB*. These tables, which will be concatenated due to the fact that they have the same set of fields, include the field *Country*, whose field values will be compared with those of the first column of the mapping table. The field values US, U.S., and United States will be found and replaced by the values of the second column of the mapping table, i.e. *USA*.

The automatic mapping is done last in the chain of events that leads up to the field being stored in the Qlik Sense table. For a typical **LOAD** or **SELECT** statement the order of events is roughly as follows:

1. Evaluation of expressions
2. Renaming of fields by as
3. Renaming of fields by alias
4. Qualification of table name, if applicable
5. Mapping of data if field name matches

This means that the mapping is not done every time a field name is encountered as part of an expression but rather when the value is stored under the field name in the Qlik Sense table.

To disable mapping, use the **unmap** statement.

For mapping on expression level, use the **applymap** function.

For mapping on substring level, use the **mapsubstring** function.

8.13 Wild cards in the data

It is also possible to use wild cards in the data. Two different wild cards exist: the star symbol, interpreted as all values of this field, and an optional symbol, interpreted as all remaining values of this field.

The star symbol

The star symbol is interpreted as all (listed) values of this field, that is, a value listed elsewhere in this table. If used in one of the system fields (*USERID, PASSWORD, NTNAME* or *SERIAL*) in a table loaded in the access section of the script, it is interpreted as all (also not listed) possible values of this field.

The star symbol is not allowed in information files. Also, it cannot be used in key fields, that is, fields used to join tables.

There is no star symbol available unless explicitly specified.

OtherSymbol

In many cases a way to represent all other values in a table is needed, that is, all values that were not explicitly found in the loaded data. This is done with a special variable called **OtherSymbol**. To define the **OtherSymbol** to be treated as "all other values", use the following syntax:

```
SET OTHERSYMBOL=<sym>;
```

before a **LOAD** or **SELECT** statement. <sym> may be any string.

The appearance of the defined symbol in an internal table will cause Qlik Sense to define it as all values not previously loaded in the field where it is found. Values found in the field after the appearance of the **OtherSymbol** will thus be disregarded.

In order to reset this functionality use:

```
SET OTHERSYMBOL=;
```

Example:

Table Customers

CustomerID	Name
1	ABC Inc.
2	XYZ Inc.
3	ACME INC
+	Undefined

Table Orders

CustomerID	Name
1	1234
3	1243
5	1248
7	1299

Insert the following statement in the script before the point where the first table above is loaded:

```
SET OTHERSYMBOL=+;
```

Any reference to a *CustomerID* other than 1, 2 or 3, e.g. as when clicking on *OrderID 1299* will result in *Undefined* under *Name*.

***OtherSymbol** is not intended to be used for creating outer joins between tables.*

8.14 NULL value handling

When no data can be produced for a certain field as a result of a database query and/or a join between tables, the result is normally a NULL value.

The Qlik Sense logic treats the following as real NULL values:

- NULL values returned from an ODBC connection
- NULL values created as a result of a forced concatenation of tables in the data load script
- NULL values created as a result of a join made in the data load script
- NULL values created as a result of the generation of field value combinations to be displayed in a table

*It is generally impossible to use these NULL values for associations and selections, except when the **NullAsValue** statement is being employed.*

Text files per definition cannot contain NULL values.

Associating/selecting NULL values from ODBC

It is possible to associate and/or select NULL values from an ODBC data source. For this purpose a script variable has been defined. The following syntax can be used:

```
SET NULLDISPLAY=<sym>;
```

The symbol <sym> will substitute all NULL values from the ODBC data source on the lowest level of data input. <sym> may be any string.

In order to reset this functionality to the default interpretation, use the following syntax:

```
SET NULLDISPLAY=;
```

*The use of **NULLDISPLAY** only affects data from an ODBC data source.*

If you wish to have the Qlik Sense logic interpret NULL values returned from an ODBC connection as an empty string, add the following to your script before any **SELECT** statement:

```
SET NULLDISPLAY=";
```

Here '' is actually two single quotation marks without anything in between.

Creating NULL values from text files

It is possible to define a symbol, which when it occurs in a text file or an **inline** clause will be interpreted as a real NULL value. Use the following statement:

```
SET NULLINTERPRET=<sym>;
```

The symbol <sym> is to be interpreted as NULL. <sym> may be any string.

In order to reset this functionality to the default interpretation, use:

```
SET NULLINTERPRET=;
```

*The use of **NULLINTERPRET** only affects data from text files and inline clauses.*

Propagation of NULL values in expressions

NULL values will propagate through an expression according to a few logical and quite reasonable rules.

Functions

The general rule is that functions return NULL when the parameters fall outside the range for which the function is defined.

Example:

asin(2)	returns	NULL
log(-5)	returns	NULL
round(A,0)	returns	NULL

As a result of the above follows that functions generally return NULL when any of the parameters necessary for the evaluation are NULL.

Example:

sin(NULL)	returns	NULL
chr(NULL)	returns	NULL
if(NULL, A, B)	returns	NULL
if(True, NULL, A)	returns	NULL
if(True, A, NULL)	returns	A

The exception to the second rule are logical functions testing for type.

Example:

isnull(NULL)	returns	True (-1)
isnum(NULL)	returns	False (0)

Arithmetic and string operators

If NULL is encountered on any side of these operators NULL is returned.

Example:

A	+	NULL	returns	NULL
A	-	NULL	returns	NULL
A	/	NULL	returns	NULL

A	*	NULL	returns	NULL
NULL	/	A	returns	NULL
0	/	NULL	returns	NULL
0	*	NULL	returns	NULL
A	&	NULL	returns	A

Relational operators

If NULL is encountered on any side of relational operators special rules apply.

Example:

NULL	rel.op	NULL	returns	NULL
A	<>	NULL	returns	True (-1)
A	<	NULL	returns	False (0)
A	<=	NULL	returns	False (0)
A	=	NULL	returns	False (0)
A	>=	NULL	returns	False (0)
A	>	NULL	returns	False (0)

9 Troubleshooting - Loading and modeling data

This section describes problems that can occur when loading and modeling data in Qlik Sense. The possible causes are described and you are presented with actions to solve the problems.

9.1 A data connection stops working after SQL Server is restarted

Possible cause:

If you create a data connection to a SQL Server, and then restart the SQL Server, the data connection may stop working, and you are not able to select data. Qlik Sense has lost connection to the SQL Server and was not able to reconnect.

Action:

Qlik Sense: Close the app, and open it again from the hub.

Qlik Sense Desktop: Close all apps and restart Qlik Sense Desktop.