Enterprise script

2005-05-01

QV Version: 7.0 English

Original release: April 2005

Contents

1. Introduction

1.1. About the course

This course is designed to introduce QlikView application developers to fundamental techniques for creating and maintaining QlikView load scripts using QlikView Enterprise. In this course, we will learn how to integrate data from various sources into a QlikView document in order to present the data in a comprehensive and understandable way to you and to others in your organisation.

The following questions will be addressed during the course:

- How do I interpret a project plan into a QlikView document?
- How do I access source data and integrate it with other data using QlikView?
- What are some of the techniques for developing data structures in QlikView?
- What tools are available to view the QlikView data structure in a document?
- How and why do I create data through the QlikView load script?
- What types of standards should I use to make my QlikView application efficient, understandable, and maintainable?
- How do I make my QlikView application secure?
- What do I do when I have a question?
- What do I do when I have a problem?

The course material is designed using a modular approach. Your instructor may choose to customize the specific material covered to suit the needs of their students. In other words, your particular training experience may include other topics, as well as eliminate some of the topics included in this training guide.

As with any product, to become an accomplished developer, you will need to work with real cases. This course will attempt to emulate a real business scenario by working through a project design approach, so that students will gain first-hand experience with creating a QlikView document out of a sample project requirements description. It may also help you to re-visit this material on your own some time in the future. The course material is designed so that it can be used with or without the guidance of an instructor.

1.2. Course data files

The course material includes the data files you need to create the course documents. Below is a short description of the data files included and how they are installed.

QWT	- Access database
EmpOff	- Excel file
Employees_New	- Excel file
Suppliers	- DIF file
Email	- Text file
Budget	- Excel file
SalesSecurity	- Text file
SalesInitials	- Text file
Access02	- Text file

1.3. Installation

You will find the files on the course CD, or other media provided by your instructor. When you run the installation program, the files will be installed in the directory: **C:\QlikView\ QlikView Training\ QVCourse_EnterpriseScript** unless you choose another directory.

Following installation, you will find a document with the name **QVE_Script_Course_Solution.qvw** in the folder **QVCourse_EnterpriseScript**. This document can be regarded as the solution to the exercises you will complete during the course.

1.4. Saving files

When you create a document of your own during the course, you may save it in the folder **QVCourse_EnterpriseScript**.

Any new data files you create during the course may be saved in the folder **QVCourse_EnterpriseScript \Files\Datasources** where the other data sources on which the final document is based can also be found.

1.5. Text format in the course material

Excerpts from the load script are written in the following typeface and size:

```
SQL SELECT
  CustomerID,
```

Keywords in the script are written in the following typeface and size:

Load

Commands, menus and buttons are written in the following typeface and size:

Edit Script

The names of sheets, list boxes, fields and expressions, etc. are written in the following typeface and size:

Salesperson

File names are written in the following typeface and size:

QVCourse_EntScr.qvw

2. The Qlik Wholesale Trade (QWT) Business Intelligence Project Plan

2.1. Introduction to the Project Plan

The QWT Business Intelligence (BI) Project Plan has been included in this course as an *example* of a project plan you may receive in your normal working environment. This is not meant to be a fully configured plan with time charts, responsibilities, etc. It is designed to provide you with an overall objective to be completed during the course. We will use the project plan as a guide for developing the load script required for the QlikView document deliverable. We will refer to the project plan document throughout the course, so you may want to keep the document open on your computer for easy access.

You will find the project plan in your course materials with the name **QWT BI Project Plan.doc**.

2.2. Project Plan Review

If you open the QWT Business Intelligence (BI) Project Plan document, you will find that it includes the following sections:

- **Key Measures**: here you will find some of the expressions that will be required in the QlikView document. Some of these calculations will be used in the load script, while others will be used in sheet objects (charts, text boxes, etc.) included in the user interface layout
- **Key Performance Indicators** (KPI's): this section includes high level KPI's that can be displayed through a dashboard perspective in the QlikView document.
- **Key Dimensions**: this section includes a list of some of the key dimensions that will be used throughout the application.
- **Trends**: provides a list of the important time dimension fields that will be required for historical analysis over time.
- **Key Selection Filters**: includes a list of the fields required in the QlikView document for selection and filtering of data.
- **Security**: contains the secured access requirements for the QlikView document.
- **Source Data Descriptions**: provides source data location and field level descriptions for each of the data sources.

Each of these sections may also include one or more Business Rules to help the application developer understand and deliver the appropriate features and functionality for the users of this QlikView document.

3. A short introduction to data structures

3.1. About data structures

To make things easier for you if have not done much work with databases previously, we will give a short introduction to the basics of data structures and databases. This introduction will hopefully help you to create QlikView documents based on your own data. If you are already familiar with these terms, you can skip this chapter.

3.2. Relational databases

Data is often stored in relational databases. Such databases include Microsoft SQL Server, Microsoft Access, Oracle, DB2, Sybase, Informix, and Teradata.

A relational database is based on a number of rules. One of these rules is that the information in the database is to be represented by the values stored in tables, which in turn constitute the database. Another rule is that the database must be able to handle at least one so-called structured language. An example of this is Structured Query Language (SQL), which is used to define and manipulate the data.

3.3. Other data structures

Another common format of data sources for QlikView is character-delimited text files. Text files must have a special structure for QlikView to be able to interpret them correctly without additional manipulation in the load script. The first line in such a text file is often, but not always, a field name, or label. The subsequent lines contain data belonging to the various fields. The fields in the file are delimited, or separated, by characters, usually commas, tabs or a semicolon. A standard text file is thus equivalent to a table with columns and rows.

4. QlikView Data Structures

4.1. Comparing database structures to QlikView data structures

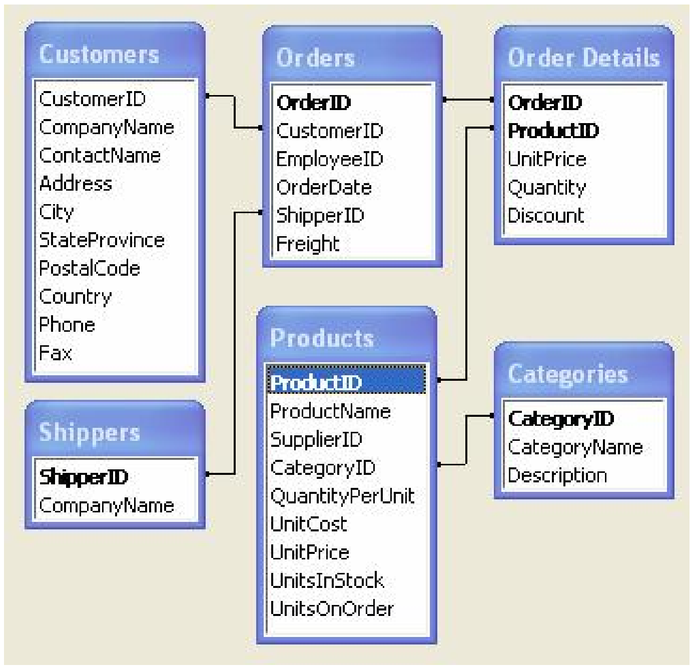


Figure 1. Table structure

The example above shows a data structure taken from the Access database that we will be working with in the course. The figure shows six tables that have defined relationships – or are associated - through common (key) fields. We will discover during the course that, unlike databases, QlikView does not allow explicit definitions of table relationships. QlikView does, however, automatically define table relationships – or *associations* – through like-named fields. In this example, the key fields are all named exactly the same in their respective tables. Of course, this is not always the case within a database, so we will explain during the course how to create the proper associations between tables in QlikView. We will, at the same time, learn how to *prevent* unwanted associations between tables in QlikView based on like-named fields. We will also learn how to associate other data, that may not necessarily be in database format (e.g. text files), to this data

The field *CustomerID* links the tables *Customers* and *Orders*. If you follow the arrows you will be able to see which fields link the whole structure. If two of the records in different tables have the same value in any of the common fields, they will be associated. Association in QlikView is essentially the same as the SQL *outer join*.

4.2. Data structures in QlikView

Each column from a data table, which is loaded into QlikView, becomes a field in the QlikView associative database (also referred to as the AQL database). Fields that appear in more than one table and have identical labels will be associated. Each field can be presented in the form of a list box in the QlikView document. Certain fields are not displayed; their only function is to link different tables. When you make a selection in a list box, QlikView searches the whole associative database for logical connections. As a result of this search the values associated with your selection are identified.

5. Loading Data into QlikView

In order to load data into QlikView, it is necessary to create instructions for data retrieval and handling. These instructions make up the bulk of the load script.

The script may specify instructions for how QlikView should interpret different data sets. QlikView can load and interpret the following types of data as input:

- Any type of character separated text files, e.g. comma separated files.
- The result of a database query, made by SQL via OLE DB/ODBC.
- Previously created QlikView-files (*binary files*).
- QlikView Data (qvd) files.
- Excel files in standard BIFF code.
- Fix format files.
- Dif files (common export format from AS/400).
- HTML tables
- XML tables

5.1. Script editing

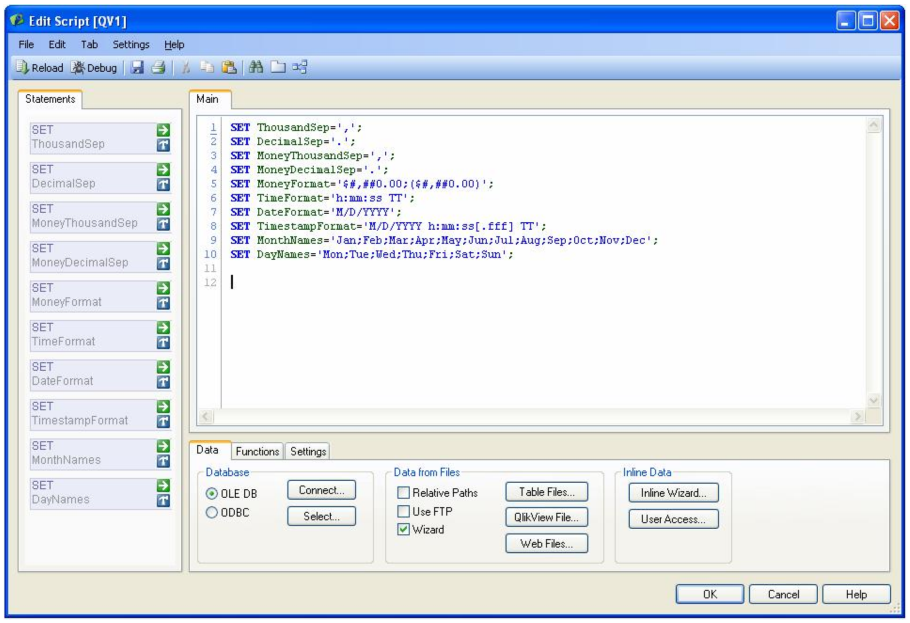
Let us now examine the **Edit Script** dialog, which can be used to generate, enter, and edit QlikView load script statements.

1. Start by creating a new document by selecting the command **New** from the **File** menu or by using the toolbar.

2. Choose **Edit Script** from the menu or toolbar.

The dialog shown below will appear on the screen. As you can see, there are numerous commands in the form of menu commands, toolbar buttons and dialog buttons. The edit window takes up the major part of the dialog though.

***TIP**:* For a complete description of all current dialogs and settings available through the Edit Script dialog, please refer to the document **ReferenceManual.pdf**. This reference document is typically installed during the standard QlikView installation.



```
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;Oct;Nov;Dec';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
```

Database: [✓ OLE DB] ODBC

Data from Files: Relative Paths, Use FTP, [✓ Wizard]

Figure 2. Edit Script dialog

Edit Script Toolbar

The toolbar contains the following controls:

Reload Executes the script, closes the **Edit Script** dialog box and opens the **Sheet Properties: Fields** page.

Debug Starts the script execution in the **Debugger**. The debugger searches for errors in the script. Every script statement can be monitored and the values of the variables can be examined while the script is executed.

Save Entire Document
Saves the active document in a file. Data, script and layout are saved.

Print Tab
Lets you print the contents of the currently active tab.

Cut Cuts out the selected script text and stores it in the Clipboard.

Copy Copies out the selected script text.

Paste Pastes the script text stored in the Clipboard back in.

Search Searches the script for the specified text string in the current tab only.

Add New Tab
Adds a new script tab. The script is executed tab by tab, from left to right.

Table Viewer
Displays the graphical table viewer for current data.

Many useful commands are also available in the five menus at the top:

Edit Script Menu Commands

The **FILE** menu has an option for saving the document without having to close the **Edit Script** dialog. Here you will also find the option for exporting the script as a script file (file extension .qvs) or as a print out.

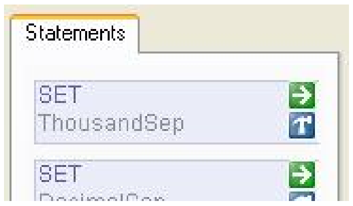
The **EDIT** menu holds all the commands necessary for editing the contents of the text edit pane. In addition to the commands for selecting, copying cutting and pasting of text, you will find the functions **Insert File** which is used for inserting a script file as well as **Find/Replace** which lets you search for specific text strings.

Many of these commands can be executed by means of the keyboard shortcuts (e.g. Ctrl+A will select all text).

The **TAB** menu contains the necessary commands for making it possible to manage the tabs of the script.

The **SETTINGS** menu includes the **Configure** command, which opens the **User Preferences: Editor** dialog where you can set font type and font color for the various text types of the script.

The commands of the **HELP** menu open the QlikView Help files. (For more information about the **Edit Script** dialog box, you can refer to the **Help** right now.)

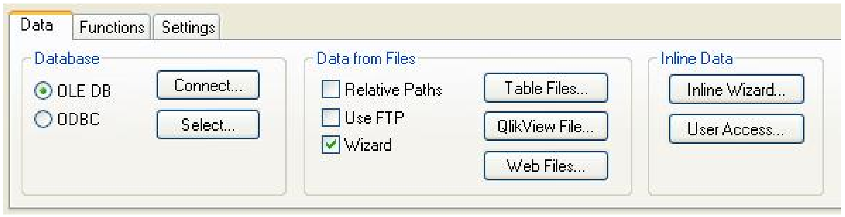


The **Statements** graph shows a box for each statement on the active script tab. The box outlines the most important features of the statement and provides an easier way to navigate the script.

Edit Script Tool Pane

The **Tool Pane** has three tab pages containing functions for script generation: Data, Functions, and Settings.

Data Page



The **Data Page** contains basic commands for loading data into QlikView.

The commands in the **Database** group are used to create a connection to and select fields from a data source. If you are using a commercial DBMS you may use ODBC or OLE DB as an interface between QlikView and the database.
OLE DB: Mark this alternative to access data bases through OLE DB.
ODBC Mark this alternative if you wish to access data bases through ODBC.

Use the **Connect...** button to open the **Data Link Properties** dialog box to select an OLE DB or ODBC data source, and generate the appropriate connect statement in the load script.
Use the **Select...** button to open the **Create Select Statement** dialog box to specify fields and tables from the chosen data source, and generate the appropriate select statement in the load script.

The commands in the **Data from Files** group are used for generating the **Load** script statements to read data from files.

Mark the **Relative Paths** check box to use relative paths instead of absolute paths for generated statements in the script.
Mark the **FTP** check box to use a dialog allowing you to select files from an ftp file server when you request **Table Files, QlikView Files**, or **Include** script statements.
Mark the **Wizard** check box to use the table file wizard when you click **Open** in the **Open Local Files** dialog.

Click on the **Table Files...** button to open the **Open Local Files** dialog box listing various text file formats, including Excel and QlikView Data (qvd) files. Selecting one or several files and pressing **OK** will generate one or several **load** statements based on the options selected in the wizard.

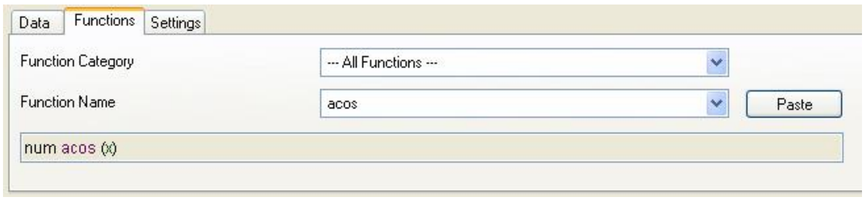
Click on the **QlikView File…** button to open the **Open QlikView File** dialog box listing QlikView files (*.qvw). Selecting a file and pressing **OK** will generate a **binary** statement. Only one binary statement is allowed in a QlikView load script, and it must be the first statement in the load script.

Click on the **Web Files…** button to open the **Table Files Wizard: Source** dialog box to enter a URL as a source for your data table.
The commands in the **Inline Data** group are used for generating the script statements to create data inline in the script.

Click on the **Inline Wizard…** button to open the **Inline Data Wizard** dialog box to assist you with creating a **Load Inline** statement using a spreadsheet type control.

Click on the **User Access…** button to open the **Access Restriction Table Wizard** dialog box to assist you with creating a special **Load Inline** statement to be used in a section access (application security).

Functions Page



num acos (x)

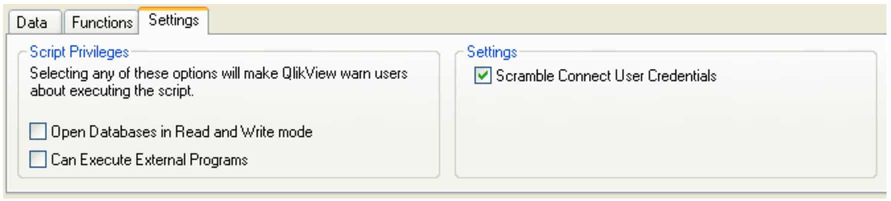
The **Functions** page can be used to help with generating QlikView functions to be used in script statements.

The **Function Category** dropdown box contains a list of function categories. Select a category in the list to see the corresponding functions in the **Function Name** list below.

The **Function Name** dropdown box contains a list of QlikView standard script functions. The list can be narrowed down by selecting a category in the **Function Category** list above.

Click on the **Paste** button once you have selected the function name you need. The function will be entered in the script at the current cursor position.

Settings Page



Script Privileges

Selecting any of these options will make QlikView warn users about executing the script.

☐ Open Databases in Read and Write mode

☐ Can Execute External Programs

Settings

☑ Scramble Connect User Credentials

The **Settings** page can be used to select certain privileges and settings in the load script.

Use the **Script Privileges** group to set the **execute** command and the **mode is write** qualifier in **select** statements. If your script contains these elements and you have not enabled these settings, the respective statements will fail. After enabling the use of either or both features the user will be prompted to approve the script the first time it is run on a computer. This check can be overridden by the **/nosecurity** command line switch or via a setting on the **Security** page of **User Preferences.**

Use the **Settings** group to set the **Scramble Connect User Credentials** setting on or off to control whether database user and password are scrambled on **connect** statements.

TIP: It is not a requirement that load script statements are created and stored within the QlikView document, but there must be a reference to them if they are stored in an external file. This is done by using the **Include** function available in the script editor.

5.2. Syntax

In this section, we will cover the most common statements (**connect**, **select**, **load**) in the script for identifying and loading data into QlikView. Each of these statements can be generated using wizards. We will practice this in upcoming sections, but first, let's look at some examples of these statements, and how and where they might be utilized in a QlikView load script.

We will also look at some of the options available for renaming a field, which is of great importance when working with QlikView. For complete and current details regarding script statement syntax, always refer to the QlikView Reference Manuals, or to the **Help** subsystem. All the script statements in this course are described in detail in Book 1 of the **Reference Manual** for QlikView 7.

5.3. Connect

The **connect** statement is used to establish a connection to a database through an ODBC or OLE DB interface. Once this connection is established, it is utilized until a new **connect** is defined. Multiple **connect** statements can be defined in a QlikView load script, but only one database connection can be open at any time.

If the **connect** statement is generated by the provided wizard any user ID and password provided will be generated with the scrambled **xuserid is / xpassword is** syntax, provided that **Scramble Connect User Credentials** is selected on the **General** page of the **User Preferences** dialog. If you enter the connect statement manually, the non-scrambled **userid is / password is** syntax must be used for providing user ID and password. Full scrambling is currently only possible for **ODBC connect** statements. Some parts of the **OLEDB connect** string cannot be scrambled.

The **codepage is** specifier can be used if you encounter problems with national characters in specific ODBC/OLE DB drivers.
If **mode is write** is specified in the *access_info* the connection will be opened in read-write mode. In all other cases the connection will be opened as read-only. The use of **mode is write** must be enabled in the Settings section of the **Edit Script** dialog .
If **ODBC** is placed before **connect**, the ODBC interface will be used, otherwise OLE DB will be used.

<u>Examples of **connect**:</u>

```
ODBC connect to [SQLDATA;database=SQL1] (UserId is
sa, Password is admin);

ODBC CONNECT TO [MS Access
Database;DBQ=data\sampledata.mdb];

ODBC connect to
[COSQL01;DATABASE=SALESDATA;Trusted_Connection=Yes];
```

The data source specified by this statement is used by all the subsequent **select** statements until a new **connect** statement is encountered.

5.4. Select

The SQL **select** statement is used to identify fields and tables to load from the current database connection. Any valid **select** statement can be used, but be aware that ODBC drivers can impose limitations on acceptable syntax for a particular database connection. Also, select statements can not utilize QlikView functions within the statement.

Field names and table names must be bracketed by quotes or square brackets if they contain spaces or special characters. When the script is automatically generated by QlikView, the quotation mark used is the one preferred by the ODBC driver specified in the data source definition of the data source in the **connect** statement.

Furthermore, several **select** statements can sometimes be concatenated into one through the use of a **union** operator:
selectstatement **union** *selectstatement*

The **select** statement is interpreted by the ODBC driver, so deviations from the general SQL syntax may occur depending on the capabilities of the ODBC drivers. For example:

- **as** is sometimes not allowed, i.e. *aliasname* must follow immediately after *fieldname*.
- **as** is sometimes compulsory if an *aliasname* is used.
- **distinct**, **as**, **where**, **group by**, **order by**, or **union** are sometimes not supported.
- The ODBC driver sometimes does not accept some types of quotation marks.

Examples of **select**:

```
SQL SELECT * FROM FACILITIES;

SQL SELECT DISTINCT
  I.AddressID,
   Name,
   Address,
   PostalCode
FROM
   [Invoice] I,
   [Address] A
WHERE
  I.InvoiceType is not null
   and I.InvoiceDate >= '2001-01-01'
  and I.AddressID = A.AddressID;
```

5.5. Load

The **Load** statement can load data through a number of different methods:

- Load directly from a text, Excel, qvd, xml, etc. file
- Load from a subsequent **select** or **load** statement. The subsequent **select** or **load** must immediately follow this **load** statement.
- Load from a previously loaded (resident) table
- Load directly from data in the load script through an **Inline load**
- Load from generated data

All QlikView script functions are available for use in a **load** statement.

Field names and table names must be bracketed by single quotes or square brackets if they contain spaces or special characters.

Examples of **load**:

```
Load * from 'c:\userfiles\data2.txt' (ansi, txt,
delimiter is '\t', embedded labels);

Load A, B, if(C>0,'+','-') as X, weekday(D) as Y;
Select A,B,C,D from Table1;

Load
 A, B, A*B+D as E
Resident tab1;

Load * Inline
[CatID, Category
0,Regular
1,Occasional
2,Permanent];

Load
 RecNo() as A, rand() as B
Autogenerate(10000);
```

5.6. Renaming a field

It is possible to rename one or more fields in the load script. It is also possible to name fields that have no name in the source data. There are multiple methods of doing this in the script:

- Rename using **as** in a **load** statement, which means that you rename a specific field in that specific statement. If you are using the Table Files Wizard to create a **load** statement, you can click on any field name in the Label area, and enter a new name. The generated **load** statement will include the **as** syntax automatically.
- Rename using **alias**, which means that you rename all the occurrences of those fields with the names specified in the script.
- Rename one or more existing fields using **Rename Field** statement. This statement can optionally use a mapping table, which stores the *oldname* to *newname* conversion data. We will discus mapping tables later in this course.

Example **as**:

```
Load
  Capital as Capital city,
  Cntry as Country,
  Pop as Population
from Country.csv (ansi, txt…
```

The syntax for an **alias** statement is:

Alias <fieldname> **as** <new fieldname>, <fieldname> **as** <new fieldname>,…

Example **alias**:

```
Alias ProdId as ProductID, Mon as Month, Cname as
Customer;
```

The syntax for a **rename field** statement is:

rename field*[***s***]* *(***using** *mapname | oldname* **to** *newname {, oldname* **to** *newname})*

where
mapname is the name of a previously loaded mapping table containing one or more pairs of old and new field names
oldname is the old field name and
newname is the new field name.

Note Both **rename field** and **rename fields** are allowed forms with no difference in effect.

Example **rename field** :

```
Rename field XAZ0007 to Sales;

FieldMap:
Mapping select oldnames, newnames from datadict;
Rename fields using FieldMap;
```

6. Data source files

In the first part of the course, we will load data from three different sources, according to our project plan document. The primary data will come from an Access database, named QWT. To this data, we will add tables from Excel spreadsheets and from a text file in DIF format, which has been extracted from an AS/400 system.

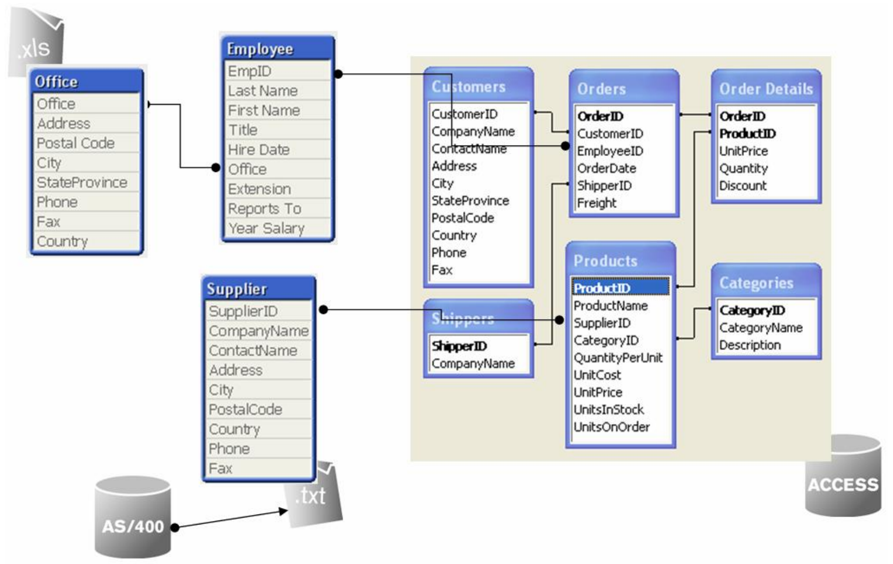


Figure 3. Data source files

The data sources are logically connected by common fields (a.k.a. key fields). In the case of the tables that contain information on the employees and the company's orders, we have the common fields *EmployeeID* and *EmpID*. However, one of the fields must be renamed for QlikView to associate these fields in our application. We also have *SupplierID* which is a common field in the QWT database and the table containing data on the suppliers (DIF). You may also notice that there are like named fields between the tables that we *not* want to associate. These fields will have to be renamed as well, to prevent an inadvertent QlikView association.

6.1. The order database

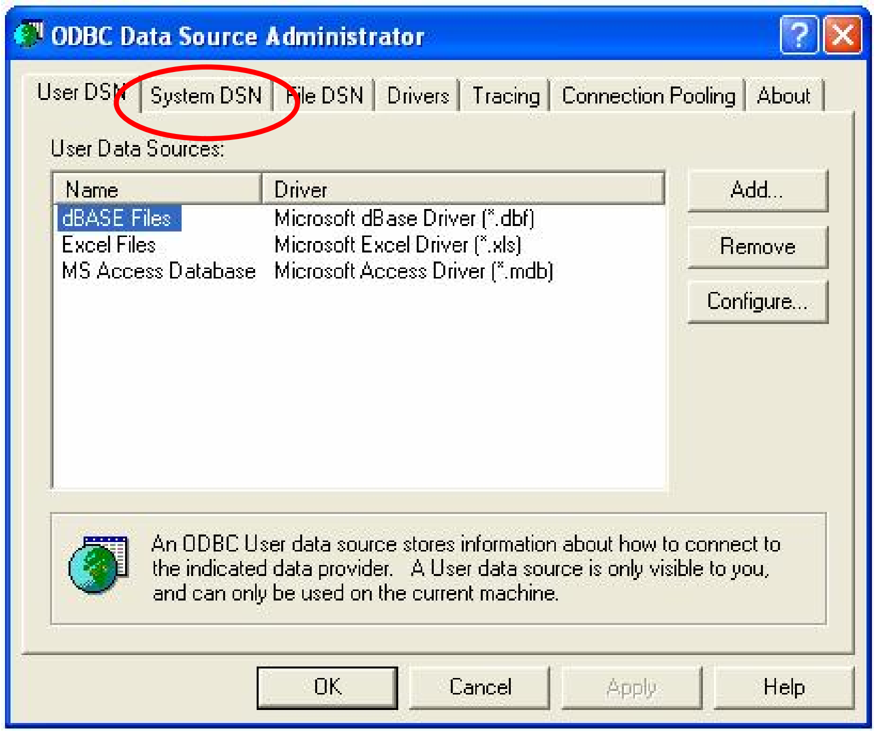
According to our project plan, and as can been gathered from figure 3, the QWT.mdb database contains the *Customers, Suppliers, Products, Categories, Orders, and Order Details* tables. We will load each of these tables, but first we need to create an ODBC connection for the database.

6.2. Creating an ODBC connection

In order to gain access to the database from QlikView, we need an ODBC data source. It is created via the **Control Panel … Administrative Tools … Data Sources (ODBC)** in Windows. Below is a guide through the various stages.

***TIP**:* It is also possible to define a data source through the Connect dialog in the QlikView Script Editor.

1. Run the ODBC Administrator Program through Control Panel to open the **ODBC – Data Source Administrator** dialog.



User Data Sources:

Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
MS Access Database	Microsoft Access Driver (*.mdb)

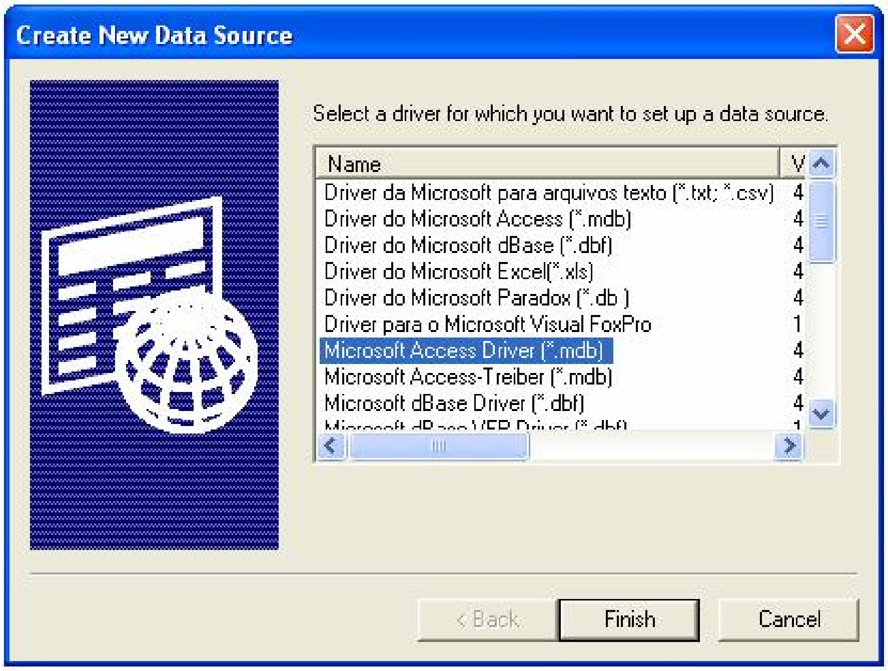
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Figure 4. ODBC – Data source administration

2. Determine whether you need to define a *User* or *System* level data source. A User data source will only be available to the current user on this machine. A system data source will be available to any user on this machine. For this course, we will add a System data source. To do this, click on the **System DSN** tab in the dialog.

3. Add a system data source by clicking on the button **Add…**

You must now select an appropriate driver for accessing your data source. A number of drivers can be seen in the figure below. If you are using a commercial database, but do not have its drivers installed, you will probably be able to find them on the installation disks for the database. If the installation disks are unavailable or the drivers are outdated or otherwise inappropriate, contact your supplier or look for the drivers on the Internet.



Select a driver for which you want to set up a data source.

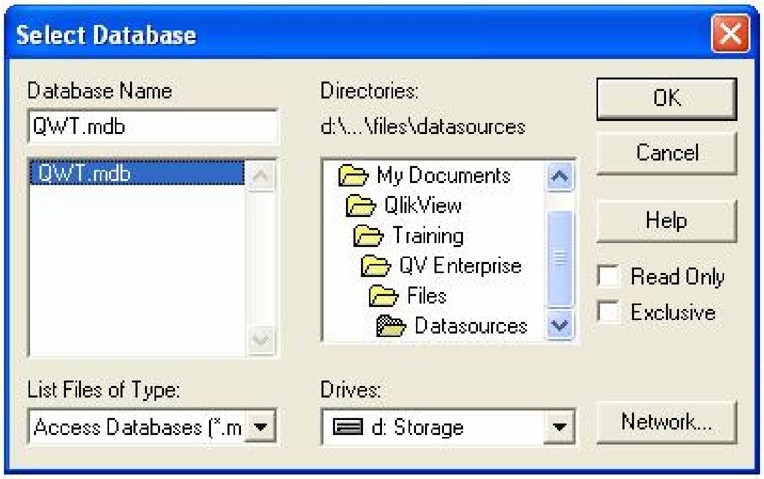
Name	V
Driver da Microsoft para arquivos texto (*.txt; *.csv)	4
Driver do Microsoft Access (*.mdb)	4
Driver do Microsoft dBase (*.dbf)	4
Driver do Microsoft Excel(*.xls)	4
Driver do Microsoft Paradox (*.db)	4
Driver para o Microsoft Visual FoxPro	1
Microsoft Access Driver (*.mdb)	4
Microsoft Access-Treiber (*.mdb)	4
Microsoft dBase Driver (*.dbf)	4

Figure 5. Selecting the driver

4. Select **Microsoft Access Driver (*.mdb)** and click **Finish**.

5. Click **Select**.

6. Navigate to the folder where the QWT database is stored (**DataSources**), and select **QWT.mdb.** Click **OK**.



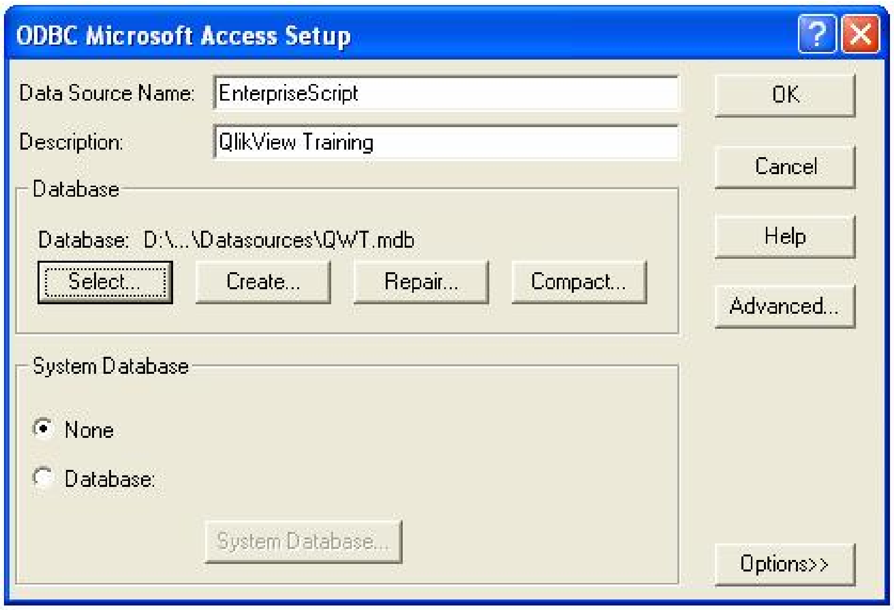
Database Name: QWT.mdb
QWT.mdb

Directories: d:\...\files\datasources
My Documents
QlikView
Training
QV Enterprise
Files
Datasources

List Files of Type: Access Databases (*.m
Drives: d: Storage

Figure 6. Selecting the data source

7. Enter a suitable title for your data source. It's recommended that you fill in the **Data Source Name** and **Description** according to the figure below. Note that your Database location may be located on a different drive.



Data Source Name: EnterpriseScript
Description: QlikView Training
Database: D:\...\Datasources\QWT.mdb
System Database: [✓ None] Database:

Figure 7. Naming the database

8. Click **OK** to accept the changes and close the dialog.

You should now be able to access the data source from QlikView. If for some reason the connection to the data source is broken, use the **Repair** button in the dialog above in order to re-establish the connection.

7. Creating the script

The script that we will write in this course loads data from an Access database. Fields from a number of tables will be loaded using **select** statements. The syntax used is standard SQL.

7.1. Script generation

The advantage of using the QlikView script editor is that many of the script statements are generated automatically by selecting the fields you want to load in the file wizards. It is often necessary to make some changes manually, e.g. to assign new field names. The script editor may also point out obvious errors through color-coding , e.g. unmatched parenthesis on a function.

7.2. Generating a script – step by step

1. Start QlikView Enterprise, if it is not already active.

2. Select **New** from the **File** menu or the appropriate button on the toolbar to create a new document.

3. Select **Document Properties** from the **Settings** menu, and open the **General** tab in the dialog. Make sure **Generate Logfile** is checked. This will generate a script execution log file every time the load script is run. You can also check the **User Preferences** option on the **Design** tab to **Always Use Logfiles for New Documents** so that this option will be selected for you automatically in the future.

4. Select **Save** from the **File** menu or the appropriate button on the toolbar to save a document. Navigate to the course directory and save your file with a suitable name under **Files**.

TIP: It is usually a good idea to save a new document prior to editing the script, so the correct relative paths can be generated. This will typically be required for portability of the QlikView document.

5. Select **Edit Script** from the **File** menu or the toolbar.

You have now created a new script file and, as you can see, it already contains some lines of script. These are the format variables, which are generated automatically by QlikView. The variables are based on the settings in your operating system regarding date, currency, time, etc.

6. Mark the **Relative Paths** checkbox, if it is not already checked. This will generate relative paths for **Data from Files** loads from the saved location of the QlikView document in our script statements. This will *not* apply to our **connect** statement below, but it will be useful later when we read from text files.

7. Select ODBC and click **Connect** to open the **Connect to Data Source** dialog.

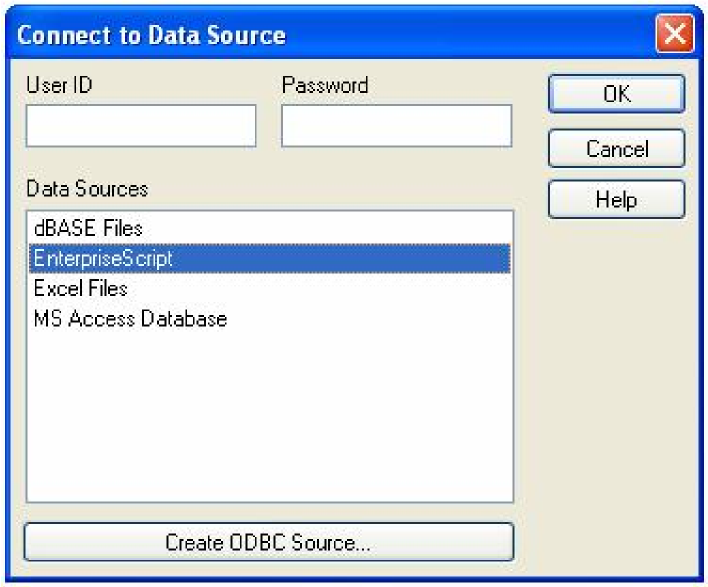


Figure 8. Selecting the data source

8. Select **EnterpriseScript** and click **OK**. You should now see the following statement added to your script (with an absolute path).

```
ODBC CONNECT TO
[EnterpriseScript;DBQ=Datasources\QWT.mdb];
```

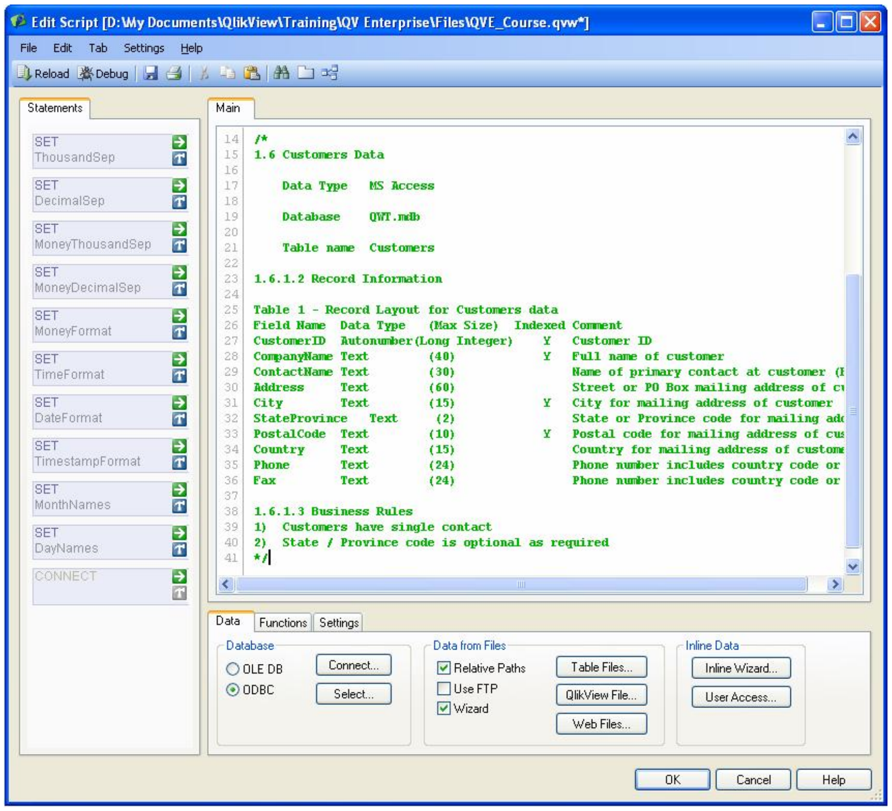
The path generated in the **connect** statement is actually the absolute path. It is suggested, however, that you change the path to a relative one, as above. It is much easier to move files from one computer to another when you use relative paths.

9. Next, we will add data from the Customers table. Before generating the select statement, let's add a comment to document the script as we create it.

QlikView allows three different comment types:

- **REM** preceding a statement will comment that statement up to its ending ;
- // will comment all text following it on a single line.
- **/* … */** will comment all text between the delimiters.

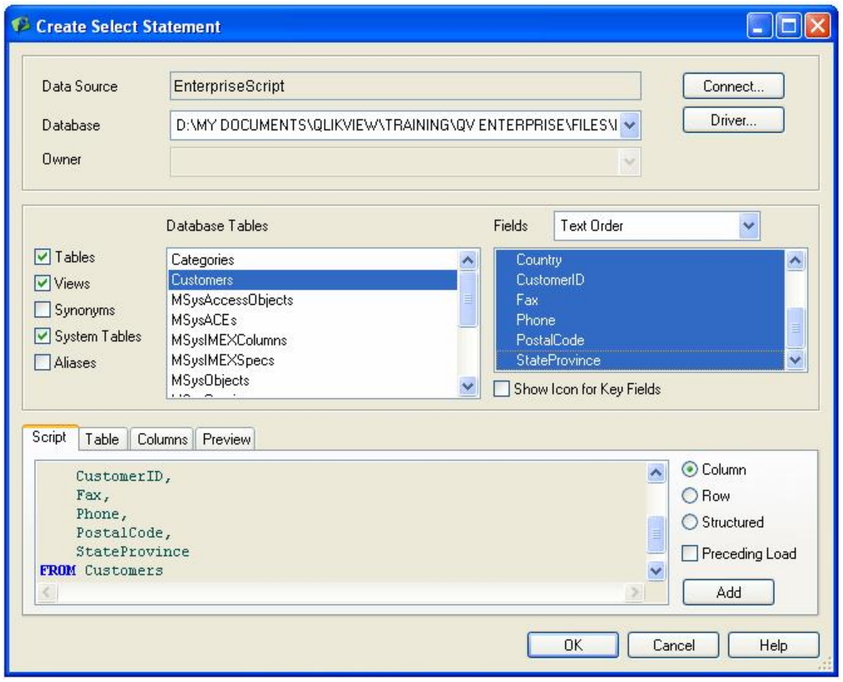
***TIP**:* Be sure not to use the // comment for an Include function, since that will only comment the initial line in the Include file.



```
/*
1.6 Customers Data

    Data Type   MS Access

    Database    QWT.mdb

    Table name  Customers

1.6.1.2 Record Information

Table 1 - Record Layout for Customers data
Field Name  Data Type   (Max Size)  Indexed Comment
CustomerID  Autonumber(Long Integer)    Y   Customer ID
CompanyName Text        (40)            Y   Full name of customer
ContactName Text        (30)                Name of primary contact at customer (I
Address     Text        (60)                Street or PO Box mailing address of cu
City        Text        (15)            Y   City for mailing address of customer
StateProvince   Text     (2)                State or Province code for mailing add
PostalCode  Text        (10)            Y   Postal code for mailing address of cus
Country     Text        (15)                Country for mailing address of custome
Phone       Text        (24)                Phone number includes country code or
Fax         Text        (24)                Phone number includes country code or

1.6.1.3 Business Rules
1)  Customers have single contact
2)  State / Province code is optional as required
*/
```

Figure 9. Adding comments to the load script

Feel free to add any type of comment you would like. The suggestion above used the text from the project plan document from the Customers table data description. (Some formatting was required to line up the columns in the record layout table).



Data Source: EnterpriseScript
Database: D:\MY DOCUMENTS\QLIKVIEW\TRAINING\QV ENTERPRISE\FILES\
Tables [✓] Views [✓] Synonyms [ ] System Tables [✓] Aliases [ ]
Database Tables: Categories, Customers, MSysAccessObjects, MSysACEs, MSysIMEXColumns, MSysIMEXSpecs, MSysObjects
Fields: Text Order — Country, CustomerID, Fax, Phone, PostalCode, StateProvince

```
    CustomerID,
    Fax,
    Phone,
    PostalCode,
    StateProvince
FROM Customers
```

Figure 9. Selecting all fields from the Customers table

10. Click **SELECT** in the **Edit Script** dialog, and select the table *Customers* from **Database Tables**.

11. Select all the fields in the table by marking them with the mouse. You can also select * to mark all the fields, but the names of the fields will then not be written out in the script, which will cause additional work later. In the frame under the table and field names you can preview the script that will be generated automatically.

12. Click **OK**. The selected tables and fields will now be included in the script. The select statement should look something like this:

```
Customers:
SQL SELECT
    Address,
    City,
    CompanyName,
    ContactName,
    Country,
    CustomerID,
    Fax,
    Phone,
    PostalCode,
    StateProvince
FROM Customers;
```

13. Add the table name label **Customers:** just before the SQL SELECT to properly label this logical table in QlikView. By default,

QlikView will use the table name as the name for the logical table, but it is good practice to always label your table loads so that you control how to reference them later. QlikView will automatically italicize the table name once you add the colen (:) after the name.

14. Click the **Save** icon in the **Edit Script** dialog. This will save your entire QlikView document, including the load script.

15. Now, repeat this procedure to load the fields from the tables *Shippers*, *Products* and *Categories* in the same way as above. Change the name of the field *CompanyName* to *Shipper* in the *Shipper* table so that shippers are not mixed in with our customer companies. Also do not load the field *UnitPrice* from the *Products* table, since we will be pulling that data from the *Order Details* table.

```
/* Shippers Table comments */

Shippers:
SQL SELECT
     ShipperID,
     CompanyName as Shipper
FROM Shippers;

/* Products Table comments */

Products:
SQL SELECT
    CategoryID,
    ProductID,
    ProductName,
    QuantityPerUnit,
    SupplierID,
    UnitCost,
    UnitsInStock,
    UnitsOnOrder
FROM Products;

/* Categories Table comments */

Categories:
SQL SELECT
    CategoryID,
    CategoryName,
    Description
 FROM Categories;
```

16. Click **SELECT** again and load the table *Orders*. Manually edit the script to copy the field *OrderDate* as shown below in order to generate new fields for the year, month and day. We will be loading this data with a preceding **load** statement in order to use QlikView date functions, including the formatting of **month**. (The difference

is that the month is represented as a number when using the **select** statement, and as a combination of text and number when using the **load** statement.) Note that the text strings used to represent the months is dependent on the regional settings in your operating system (as seen in the initial script statements). If your settings are in English, the months will be given in English.

According to the project plan, under the **Trends** section, we will need to eventually offer time analysis over Month, Quarter, and Year. The following script will provide the Year, the Month, and the day of the month. We will expand on this later to add Rolling Month & Quarter.

Notice that the **Load** statement has no **From** or **Resident** clause, since the source is the following **SQL SELECT** statement.

```
/* Orders Table comments */

Orders:
Load
    CustomerID,
    EmployeeID,
    Freight,
    OrderDate,
    Year(OrderDate) as Year,
    Month(OrderDate) as Month,
    Day(OrderDate) as Day,
    OrderID,
    ShipperID;

SQL SELECT * FROM Orders;
```

17. Finally, we will load the table *OrderDetails*. Here we will create a new field *NetSales* which is the first Key Measure identified in the project plan in the **Key Measures** section. NetSales is the result of a calculation based on UnitPrice*Quantity*(1-Discount). The load script is as follows:

```
/* Order Details Table comments */

Order_Details:
SQL SELECT
    OrderID,
    ProductID,
    Quantity,
    UnitPrice,
    UnitPrice*Quantity*(1-Discount) as NetSales
FROM `Order Details`;
```

***TIP**:* Spaces are not allowed in the explicit table label name.

Be sure to save your application – either in the **Edit Script** dialog, or in the QlikView interface through **File** … **Save**.

7.3. Your first basic script

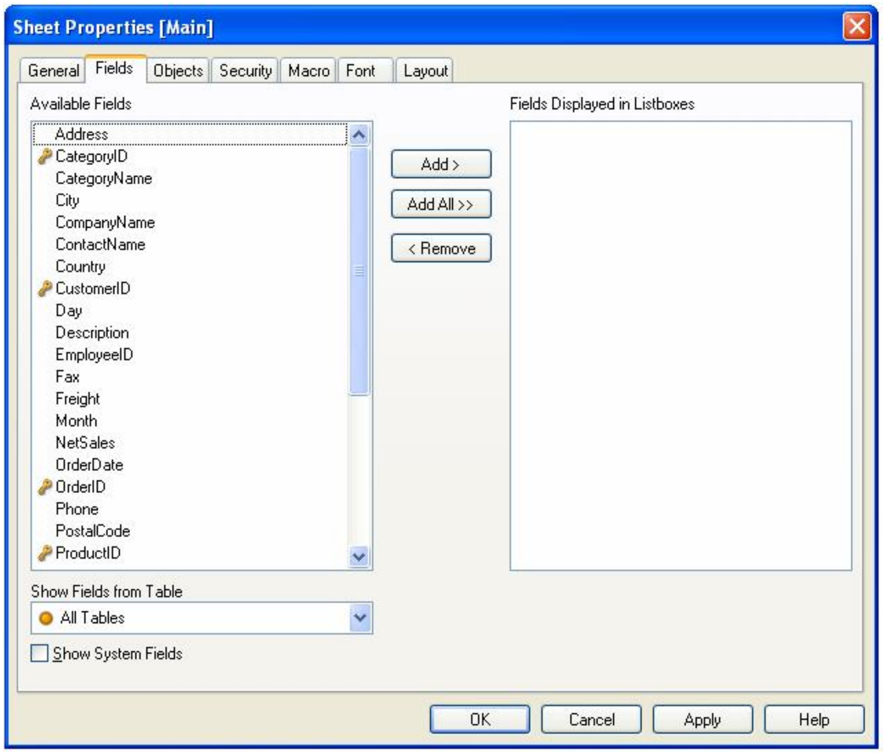
Your script should now look like this:

```
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;Oc
t;Nov;Dec';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';

ODBC CONNECT TO
[EnterpriseScript;DBQ=DATASOURCES\QWT.mdb];

 /* Customers Table comments */


Customers:
SQL SELECT
    Address,
    City,
    CompanyName,
    ContactName,
    Country,
    CustomerID,
    Fax,
    Phone,
    PostalCode,
    StateProvince
FROM Customers;

 /* Shippers Table comments */

 Shippers:
 SQL SELECT
     ShipperID,
     CompanyName as Shipper
 FROM Shippers;

 /* Products Table comments */

 Products:
 SQL SELECT
     CategoryID,
     ProductID,
    ProductName,
    QuantityPerUnit,
    SupplierID,
    UnitCost,
    UnitsInStock,
    UnitsOnOrder
FROM Products;

/* Categories Table comments */

Categories:
SQL SELECT
    CategoryID,
    CategoryName,
    Description
 FROM Categories;

/* Orders Table comments */

Orders:
Load
     CustomerID,
     EmployeeID,
     Freight,
     OrderDate,
     Year(OrderDate) as Year,
     Month(OrderDate) as Month,
     Day(OrderDate) as Day,
     OrderID,
     ShipperID;

SQL SELECT * FROM Orders;

/* Order Details Table comments */

Order_Details:
SQL SELECT
     OrderID,
     ProductID,
     Quantity,
     UnitPrice,
     UnitPrice*Quantity*(1-Discount) as NetSales
FROM `Order Details`;
```

7.4. Execution of the script

In order to eliminate the risk of your script being lost, for one reason or another, it is important to save it often – especially before running a load script. Then click on the **ReLoad** icon in the toolbar. The script will run and the data will be loaded into the application.

Select Fields in the **Layout** menu or **Settings … Sheet Properties … Fields** lists all the loaded fields:



Address
CategoryID
CategoryName
City
CompanyName
ContactName
Country
CustomerID
Day
Description
EmployeeID
Fax
Freight
Month
NetSales
OrderDate
OrderID
Phone
PostalCode
ProductID

Figure 10. The Sheet Properties: Fields dialog page

8. Data structure of the loaded data

In this chapter, we will learn about system fields that QlikView creates and maintains automatically, and that can be used to understand the internal structure of our QlikView document. We will also be introduced to a number of ways to monitor and analyze the structure of the QlikView data. These tools and techniques will be useful while we create the load script – and the document structure, and will be critical when trying to verify the integrity of a document, or to debug error behaviour.

8.1. System fields

During loading, six special fields are generated which contain information on the structure of the associative database in QlikView, i.e. they contain Meta data on the AQL database. These are called system fields, and we shall now see how they can be used when working with QlikView.

1. *$Field*	Shows the names of all the fields loaded
2. *$Table*	Shows the names of all the tables loaded
3. *$Rows*	Shows the number of rows in the tables
4. *$Fields*	Shows the number of fields in the various tables
5. *$FieldsNo*	Shows the positions of the fields in the tables (column number)
6. *$Info*	Shows the names of the information tables loaded

8.1.1. The system tab

When you are developing a document, a system sheet is very useful as it shows how the logical tables in the document are related to each other. It is good practice to create the system sheet as the first step after loading the data.

We will now create a system sheet:

1. Create a new sheet, either by clicking on the Add Sheet button in the Design toolbar or by selecting **Add Sheet** in the **Layout** menu.

2. Right-click on the new sheet, and select **Properties**. Enter the Title of the sheet as *System* on the **General** tab.

3. On the **Fields** tab, check the box **Show System Fields** and then select all the fields with a dollar sign, $, in front of them.

4. Click **Add**, and then **OK**.

Arrange the fields on the sheet and then select **Properties**, **General**, **Show Frequency** for the list box *$Field*, in order to be able to see how many times the various fields occur in the tables of the associative database. Under the **Sort** tab you can sort them in decreasing frequency to place the fields that occur most often at the top of the list.

8.1.2. Using system fields

If you select *CustomerID* in *$Field* you will see which tables the field appears in, along with other system fields information.

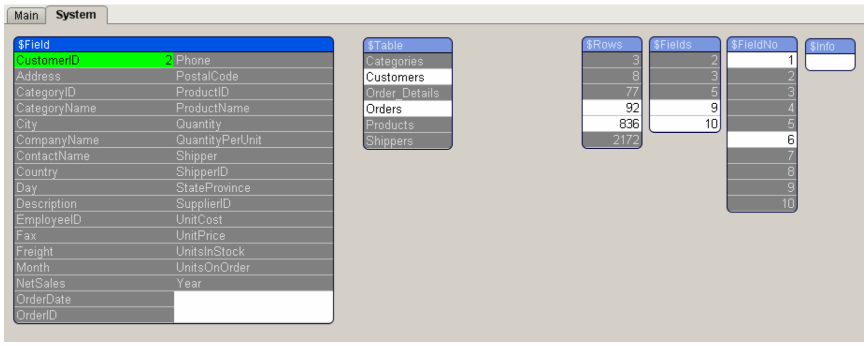


Figure 112. The result of selecting CustomerID in the list box $Field

8.2. The table viewer

Another way to display the logical structure of the available tables in a QlikView document and the connections between them is to use the integrated graphical table viewer. The command **File ... Table Viewer** (or <Ctrl>-T) opens a window displaying all the loaded tables and their connecting key fields. You may rearrange its components by clicking and dragging or by clicking **Auto-Layout**. Be sure to click on **OK** when done to save your layout.

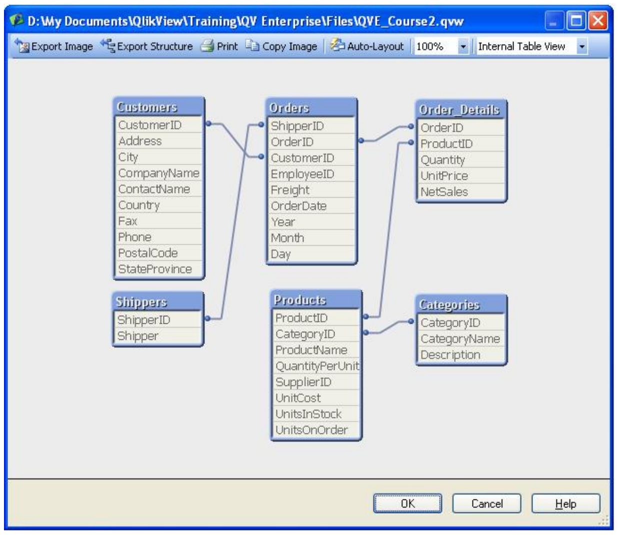


Figure 13. A sample table view

8.3. The system table

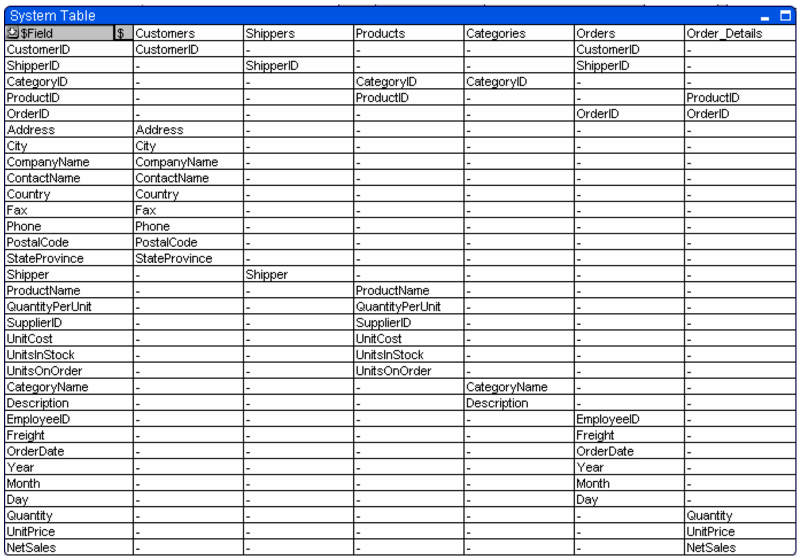
It is possible to create tables of various types in QlikView, containing widely different kinds of data, so why not use the same technique to investigate the relations between the tables in our database?

The system table is a pivot table that illustrates the relations and connections between tables and fields in QlikView's associative db.

We will proceed by creating such a table on our system sheet. Every system developer should have a system table. The more complicated the data structure, the greater the use of the table.

1. In your QlikView application, make sure you are on the *System* sheet.

2. Right-click in the sheet and select new sheet object, **System Table**. A Pivot Table will be created with the dimensions *$Field* and $*Table*. The expression in the chart will be *Only($Field)*. Both dimensions are sorted according to load order.



System Table

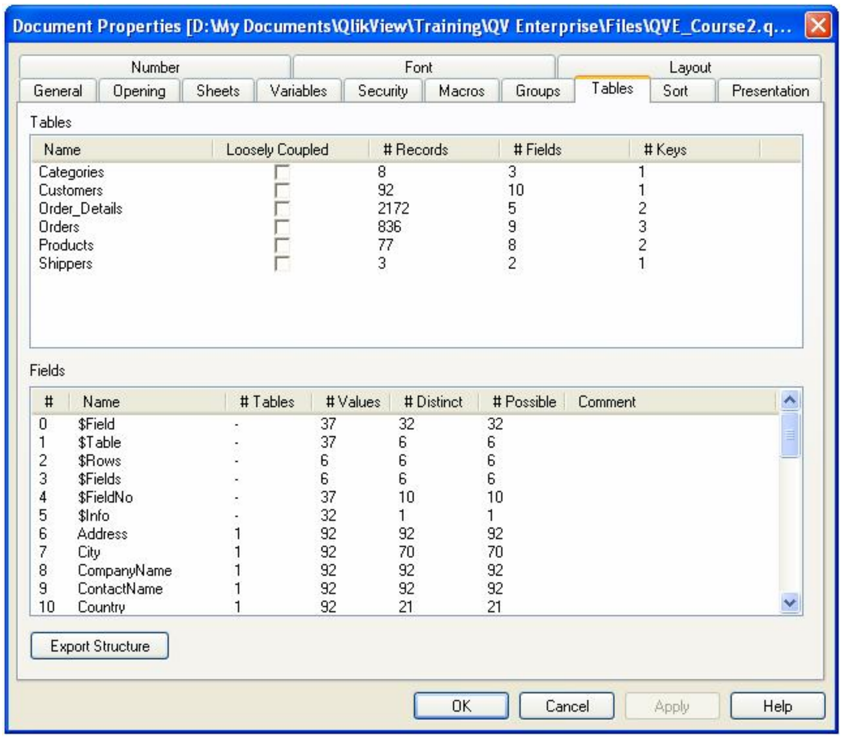
$Field $	Customers	Shippers	Products	Categories	Orders	Order_Details
CustomerID	CustomerID	-	-	-	CustomerID	-
ShipperID	-	ShipperID	-	-	ShipperID	-
CategoryID	-	-	CategoryID	CategoryID	-	-
ProductID	-	-	ProductID	-	-	ProductID
OrderID	-	-	-	-	OrderID	OrderID
Address	Address	-	-	-	-	-
City	City	-	-	-	-	-
CompanyName	CompanyName	-	-	-	-	-
ContactName	ContactName	-	-	-	-	-
Country	Country	-	-	-	-	-
Fax	Fax	-	-	-	-	-
Phone	Phone	-	-	-	-	-
PostalCode	PostalCode	-	-	-	-	-
StateProvince	StateProvince	-	-	-	-	-
Shipper	-	Shipper	-	-	-	-
ProductName	-	-	ProductName	-	-	-
QuantityPerUnit	-	-	QuantityPerUnit	-	-	-
SupplierID	-	-	SupplierID	-	-	-
UnitCost	-	-	UnitCost	-	-	-
UnitsInStock	-	-	UnitsInStock	-	-	-
UnitsOnOrder	-	-	UnitsOnOrder	-	-	-
CategoryName	-	-	-	CategoryName	-	-
Description	-	-	-	Description	-	-
EmployeeID	-	-	-	-	EmployeeID	-
Freight	-	-	-	-	Freight	-
OrderDate	-	-	-	-	OrderDate	-
Year	-	-	-	-	Year	-
Month	-	-	-	-	Month	-
Day	-	-	-	-	Day	-
Quantity	-	-	-	-	-	Quantity
UnitPrice	-	-	-	-	-	UnitPrice
NetSales	-	-	-	-	-	NetSales

Figure 14. A System table

8.4. Document Properties: Tables dialog page

This dialog page offers yet another way of looking at the data structure. All tables and fields included in the QlikView document are list, along with statistics on each entity.

Click on the **Export Structure** button to export a number of tab delimited text files containing this information. These files can then be imported back into QlikView – either this document or another document – for additional analysis.



Tables

Name	Loosely Coupled	# Records	# Fields	# Keys
Categories	☐	8	3	1
Customers	☐	92	10	1
Order_Details	☐	2172	5	2
Orders	☐	836	9	3
Products	☐	77	8	2
Shippers	☐	3	2	1

Fields

#	Name	# Tables	# Values	# Distinct	# Possible	Comment
0	$Field	-	37	32	32	
1	$Table	-	37	6	6	
2	$Rows	-	6	6	6	
3	$Fields	-	6	6	6	
4	$FieldNo	-	37	10	10	
5	$Info	-	32	1	1	
6	Address	1	92	92	92	
7	City	1	92	70	70	
8	CompanyName	1	92	92	92	
9	ContactName	1	92	92	92	
10	Country	1	92	21	21	

Figure 15. Document Properties: Tables dialog

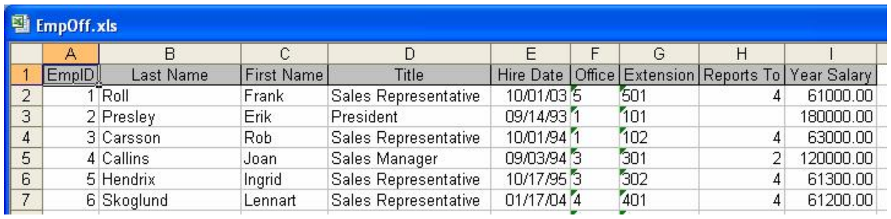
We will refer back to this dialog in a later chapter while discussing circular references in the data structure and how they can be tracked using the **Document Properties: Tables** dialog. Specifically, we will learn about the Loosely Coupled check boxes for Tables, and how they might be used.

9. Adding Text Data

According to our project plan, there are additional data tables to load. These tables are not in database format, so we will now start to use the Table Files wizard to create the QlikView Load statements. The sources this time are two Excel spreadsheets and a DIF file. These files contain data on the employees , offices, and suppliers. Let us start by looking at the data sources.

9.1. Employees

We will take data on the employees from the Excel file **EmpOff.xls** and the spreadsheet *Employee* (in the folder **DataSources**). We will first open the file in Excel, to take a look at its contents.



EmpOff.xls

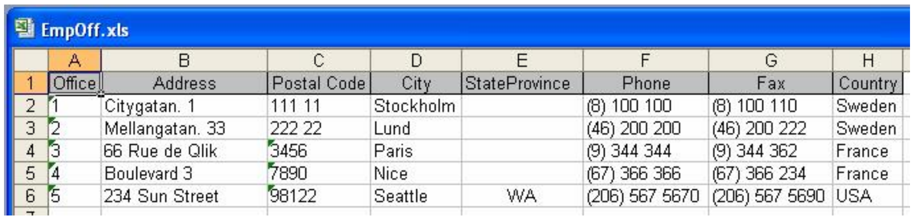
	A	B	C	D	E	F	G	H	I
1	EmpID	Last Name	First Name	Title	Hire Date	Office	Extension	Reports To	Year Salary
2	1	Roll	Frank	Sales Representative	10/01/03	5	501	4	61000.00
3	2	Presley	Erik	President	09/14/93	1	101		180000.00
4	3	Carsson	Rob	Sales Representative	10/01/94	1	102	4	63000.00
5	4	Callins	Joan	Sales Manager	09/03/94	3	301	2	120000.00
6	5	Hendrix	Ingrid	Sales Representative	10/17/95	3	302	4	61300.00
7	6	Skoglund	Lennart	Sales Representative	01/17/04	4	401	4	61200.00

Figure 16. The Employee table

The key in this table is the field *EmpID*, which will connect the table to the rest of the data we have loaded.

9.2. Offices

Data on the company's offices will also be taken from the Excel file **EmpOff.xls** but from the *Office* spreadsheet, which is the second spreadsheet in the Excel file **EmpOff.xls**.



EmpOff.xls

	A	B	C	D	E	F	G	H
1	Office	Address	Postal Code	City	StateProvince	Phone	Fax	Country
2	1	Citygatan. 1	111 11	Stockholm		(8) 100 100	(8) 100 110	Sweden
3	2	Mellangatan. 33	222 22	Lund		(46) 200 200	(46) 200 222	Sweden
4	3	66 Rue de Qlik	3456	Paris		(9) 344 344	(9) 344 362	France
5	4	Boulevard 3	7890	Nice		(67) 366 366	(67) 366 234	France
6	5	234 Sun Street	98122	Seattle	WA	(206) 567 5670	(206) 567 5690	USA

Figure 17. The Office table

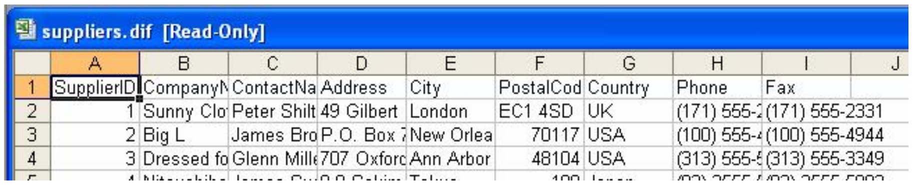
The key in this table is *Office* and the values in this field will be associated with values in the *Office* field in the *Employee* table.

9.3. Suppliers

Data on the company's suppliers will be taken from a DIF file called **suppliers.dif** (folder **DataSources**). This format is somewhat different, as you will see if you look at the file in a normal text editor, e.g. Notepad. The DIF file looks like this:

```
TABLE
0,1
"EXCEL"
VECTORS
0,30
""
TUPLES
0,9
""
DATA
0,0
""
-1,0
BOT
1,0
"SupplierID"
1,0
"CompanyName"
1,0
"ContactName"
1,0
"Address"
1,0
"City"
1,0
```

If we look at the same file in Excel, it is somewhat more intelligible:



suppliers.dif [Read-Only]

	A	B	C	D	E	F	G	H	I	J
1	SupplierID	CompanyN	ContactNa	Address	City	PostalCod	Country	Phone	Fax	
2	1	Sunny Clo	Peter Shilt	49 Gilbert	London	EC1 4SD	UK	(171) 555-2	(171) 555-2331	
3	2	Big L	James Bro	P.O. Box 7	New Orlea	70117	USA	(100) 555-4	(100) 555-4944	
4	3	Dressed fo	Glenn Mille	707 Oxford	Ann Arbor	48104	USA	(313) 555-5	(313) 555-3349	
5	4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

Figure 18. The DIF file Suppliers

We will be loading data from this file in a following chapter.

9.4. Script generation using the file wizard

We will now continue with the generation of our load script, by adding QlikView **Load** statements for the three files that we just looked at.

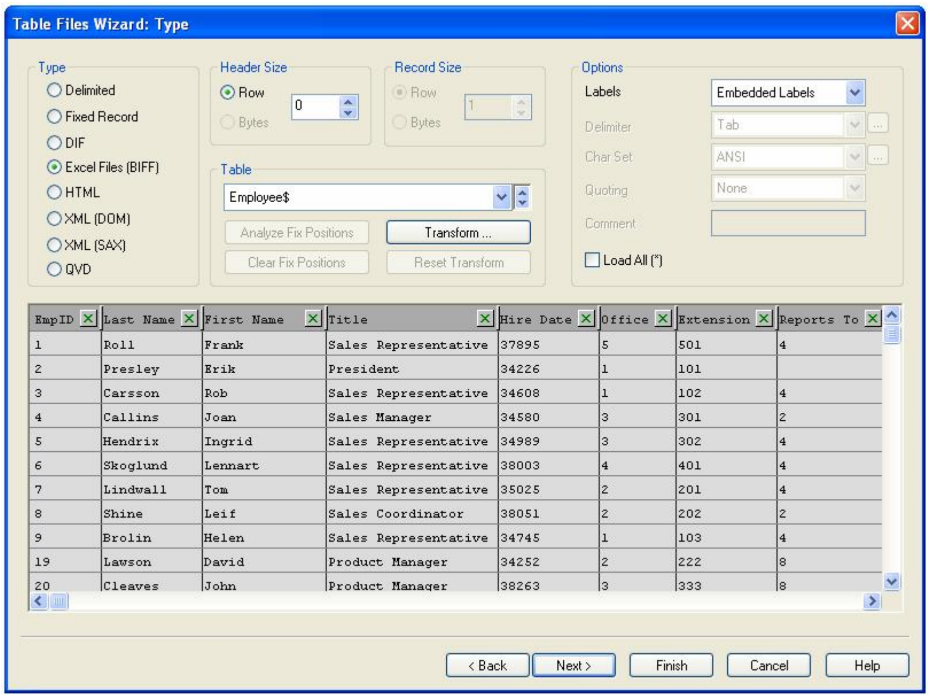
To create these statements, we will use the **Table File** wizard in the **Edit Script** dialog.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Make sure the **Relative Paths** check box is checked.

3. Ensure that the **Wizard** check box is checked and then click the **Table Files…** button to open the **Open Local Files** dialog.

4. Browse to the file **EmpOff.xls** in the folder **DataSources** and click **Open.**



Type: [✓ Excel Files (BIFF)]
Header Size: [✓ Row] 0
Table: Employee$
Labels: Embedded Labels
Delimiter: Tab
Char Set: ANSI
Quoting: None
Load All (*): ☐

EmpID	Last Name	First Name	Title	Hire Date	Office	Extension	Reports To
1	Roll	Frank	Sales Representative	37895	5	501	4
2	Presley	Erik	President	34226	1	101	
3	Carsson	Rob	Sales Representative	34608	1	102	4
4	Callins	Joan	Sales Manager	34580	3	301	2
5	Hendrix	Ingrid	Sales Representative	34989	3	302	4
6	Skoglund	Lennart	Sales Representative	38003	4	401	4
7	Lindwall	Tom	Sales Representative	35025	2	201	4
8	Shine	Leif	Sales Coordinator	38051	2	202	2
9	Brolin	Helen	Sales Representative	34745	1	103	4
19	Lawson	David	Product Manager	34252	2	222	8
20	Cleaves	John	Product Manager	38263	3	333	8

Figure 19. The File wizard

Check that the suggested settings in the wizard are correct. They should be as follows:

Type: Excel files (BIFF), **Table**: Employee$, **Embedded Labels**

5. Click on the field name *EmpID* and change the name to *EmployeeID.*

6. Click **Finish** to return to the **Edit Script** dialog and view the new **Load** statement generated for the Employee data..

7. Now, add your comments to this data load, and label the table as *Employee*. You can also delete the Directory ; statement, which is generated because of the Relative Paths specification. We will not need these statements in our script.

8. Also, according to our project plan, we need to provide an [*Employee Hire Year*] field. Add this field now, using the **year** function on the [*Hire Date*] field.

9. The script statements should look as follows:

```
/* Employee Table comments */

Employee:
Load
  EmpID as EmployeeID,
  [Last Name],
  [First Name],
  Title,
  [Hire Date],
  Year([Hire Date]) as [Employee Hire Year],
  Office,
  Extension,
  [Reports To],
  [Year Salary]
FROM Datasources\EmpOff.xls (biff, embedded labels,
table is [Employee$]);
```

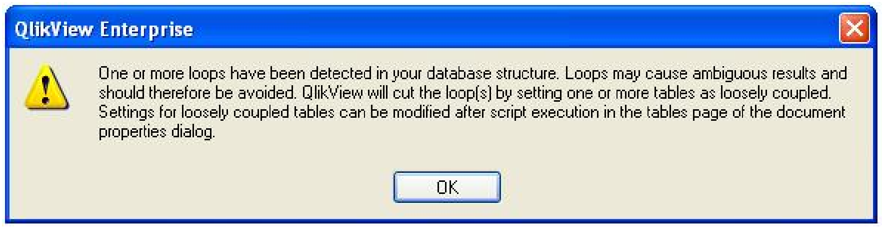
10. Now, follow the same procedure for the Office data. This table is located on the second spreadsheet of the Excel file **EmpOff.xls**. When you open the Table Wizard, be sure to change to the Excel spreadsheet *Office$* in the box labelled **Table** on the first page of the wizard.

11. Add the table comments, and label the Load statement as Office. The following statements should now be included in your script.

```
/* Office Table comments */

Office:
Load
  Office,
  Address,
  [Postal Code],
  City,
  StateProvince,
  Phone,
  Fax,
  Country
FROM Datasources\EmpOff.xls (biff, embedded labels,
table is [Office$]);
```

Note that the field names containing spaces are enclosed in square brackets (quotation marks " " can also be used), e.g. [Postal Code].

12. Click **OK** in the Script Editor, and save the document. Then reload the data from the menus or toolbar.


QlikView Enterprise

One or more loops have been detected in your database structure. Loops may cause ambiguous results and should therefore be avoided. QlikView will cut the loop(s) by setting one or more tables as loosely coupled. Settings for loosely coupled tables can be modified after script execution in the tables page of the document properties dialog.

Figure 20. Warning about circular references

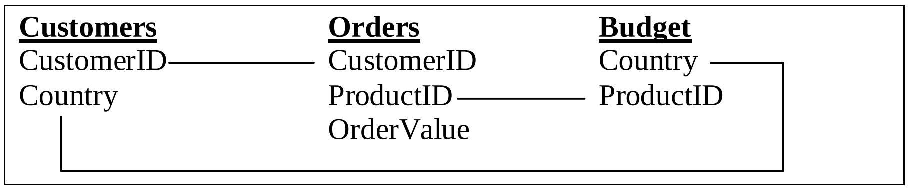
13. QlikView warns that circular references, or loops, have been found in the table structure. Click **OK**.

When working with complicated data structures containing many tables, it is possible to end up in a situation where the interpretation of the data is uncertain. QlikView has been developed in such a way that it can handle the most complicated structures and automatically interpret them correctly, but there are some limitations. It is important that you are aware of these limitations and know how to solve the problem of loops when they arise.

We will soon return to the generation of our script, but first let us look at loops and their consequences.

9.5. Circular references

Consider the following example, which consists of a simple data structure with three tables.



As you can see, it is possible to literally ”go around in circles.” In this example, it is easy to detect a circular reference, but it may be more difficult in complicated structures.

Data structures of this kind should be avoided, as they may lead to ambiguous interpretation of the data.

Unfortunately, circular references are quite common, and it is not unusual to come across them in working life. They are often due to poor database design, but in some cases they are unavoidable.

In some cases, a field (or a table) may have several roles, for example, a company may be both a supplier and a customer. The field (or table) must then be loaded into QlikView twice under different names. QlikView solves the problem of circular references by using a loosely coupled table. If QlikView finds a loop while executing the load script, a warning dialog will be shown and one or more tables will be set to loosely coupled. QlikView will attempt to make the longest table loosely coupled. This is often a transaction table. If you wish to deviate from the QlikView default, you can define the table to be loosely coupled using a **loosen table** statement in the script. It is also possible to change the settings for loosely coupled tables interactively after the execution of the script under the **Tables** tab in **Document Properties**. You can also determine which tables in your structure are set to loosely coupled by using the **Table Viewer** utility. Loosely coupled tables will show dotted lines as their connectors.

In order to avoid circular references and loosely coupled tables, you must rename the fields that cause the loops and remove the loosely coupled tables under the **Tables** tab in **Document Properties**.

9.6. Causes of circular references

Many times, circular structures will result from unintended key fields in the data load. In our QlikView document we received a warning for circular references because many fields in different tables had the same name. This results in QlikView field associations that should not occur. This can be seen by studying the system sheet.

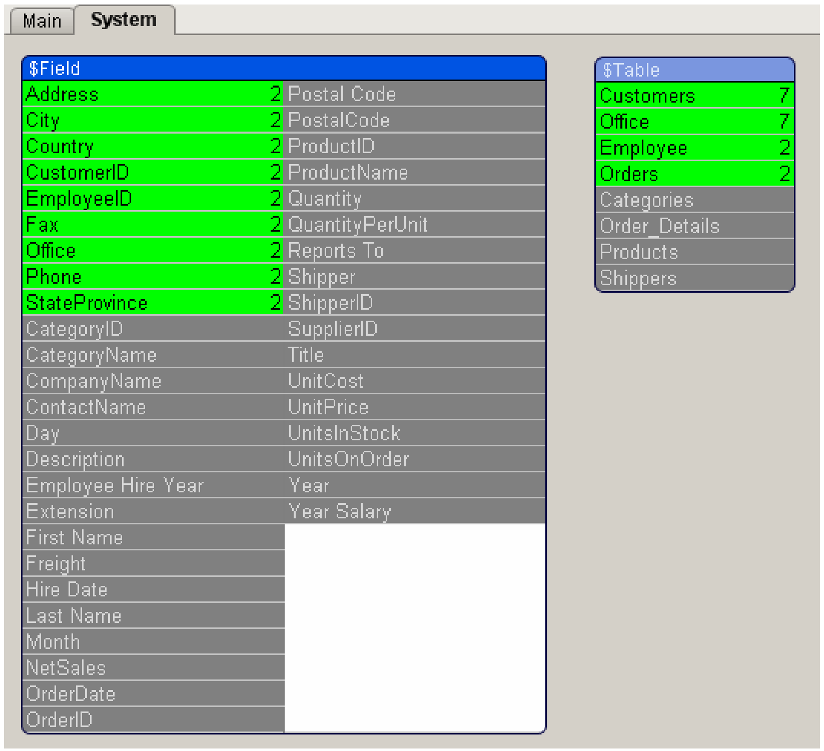
1. Go to the system sheet and right-click on the list box *$Table* and select **Properties.**

2. Select **Show Frequency** under the **General** tab and sort by frequency.

3. We can now see the frequency in both list boxes *$Field* and *$Table*. We will now select all those that have a frequency greater than 1 in *$Field* and then do the same for *$Table*.

4. Remove all the values in *$Field* that have the frequency 1, and then do the same in *$Table*. Continue in this way until it is no longer possible to remove values with the frequency 1.

If it is not possible to obtain the frequency 1 for all remaining possible tables in the list box *$Table*, then we have a circular reference somewhere in the application.



$Field			
Address	2	Postal Code	
City	2	PostalCode	
Country	2	ProductID	
CustomerID	2	ProductName	
EmployeeID	2	Quantity	
Fax	2	QuantityPerUnit	
Office	2	Reports To	
Phone	2	Shipper	
StateProvince	2	ShipperID	
CategoryID		SupplierID	
CategoryName		Title	
CompanyName		UnitCost	
ContactName		UnitPrice	
Day		UnitsInStock	
Description		UnitsOnOrder	
Employee Hire Year		Year	
Extension		Year Salary	
First Name			
Freight			
Hire Date			
Last Name			
Month			
NetSales			
OrderDate			
OrderID			

$Table	
Customers	7
Office	7
Employee	2
Orders	2
Categories	
Order_Details	
Products	
Shippers	

Figure 21. All the possible values have a frequency greater than 1

If we now look closer at the possible values in the list box *$Field*, we can see that some of these fields should probably not be connected. One example is the field *Fax*.

1. Select *Fax* in the list box *$Field*.

2. Check to see which tables the field *Fax* occurs in.

Fax is found in the tables *Customers* and *Office*. We don't want the fax numbers of our customers to be mixed up with the fax numbers of our own offices. Therefore, we should rename this field in the *Office* table, which was the table we loaded most recently. The other fields in this table should also be renamed.

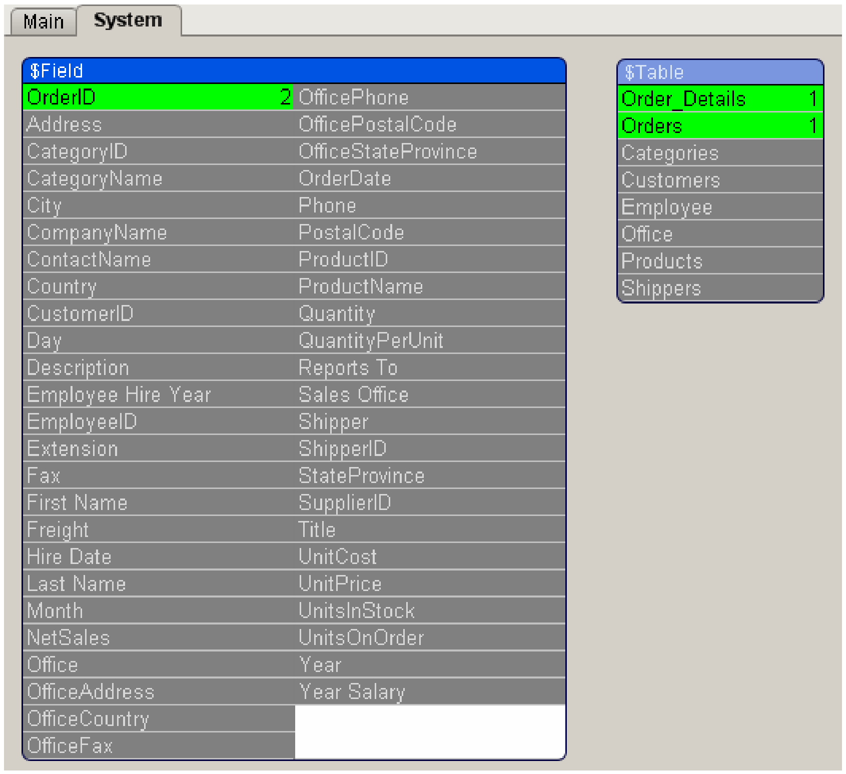
9.7. Renaming fields to avoid circular references

We can avoid circular references by modifying the script.

1. Open the **Edit Script** dialog, and rename the fields in the *Office* table as shown below. Note that we do not want to rename the intentional key field *Office*, since we still need a link to the *Employee* table. We will also use this opportunity to rename the *City* field to *[Sales Office]* as per our project plan.

```
Office:
Load
  Office,
   Address as OfficeAddress,
   [Postal Code] as OfficePostalCode,
   City as [Sales Office],
   StateProvince as OfficeStateProvince,
   Phone as OfficePhone,
   Fax as OfficeFax,
   Country as OfficeCountry
FROM Datasources\EmpOff.xls (biff, embedded labels,
table is [Office$]);
```

2. Click **OK** in the Script Editor, and save the document. Then reload the data from the menus or toolbar. There should be no warning of circular references, which means that they have all been removed.



$Field		
OrderID	2	OfficePhone
Address		OfficePostalCode
CategoryID		OfficeStateProvince
CategoryName		OrderDate
City		Phone
CompanyName		PostalCode
ContactName		ProductID
Country		ProductName
CustomerID		Quantity
Day		QuantityPerUnit
Description		Reports To
Employee Hire Year		Sales Office
EmployeeID		Shipper
Extension		ShipperID
Fax		StateProvince
First Name		SupplierID
Freight		Title
Hire Date		UnitCost
Last Name		UnitPrice
Month		UnitsInStock
NetSales		UnitsOnOrder
Office		Year
OfficeAddress		Year Salary
OfficeCountry		
OfficeFax		

$Table	
Order_Details	1
Orders	1
Categories	
Customers	
Employee	
Office	
Products	
Shippers	

Figure 22. The possible tables now have the frequency 1

As you can see in the figure above, there are no longer any fields that associate the tables *Office* and *Customers*. QlikView will automatically remove the loosely coupled designation for the *Orders* table once the circular reference has been corrected.

9.8. Loosely Coupled Tables

In a loosely coupled table, the associative logic in QlikView is internally disconnected. This means that selections in the non-key fields in one table do not affect other tables. In certain cases, this can be quite effective, although in our case, we do not want to implement this feature. In order to understand the concept of loosely coupled, consider the example below:

Below we can see table boxes created by three different tables.

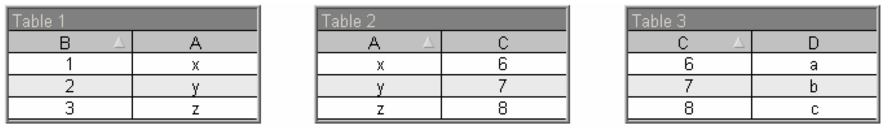


Table 1

B	A
1	x
2	y
3	z

Table 2

A	C
x	6
y	7
z	8

Table 3

C	D
6	a
7	b
8	c

If we choose the value 2 in field B, the following will occur:

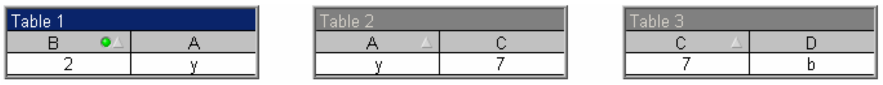


Table 1

B	A
2	y

Table 2

A	C
y	7

Table 3

C	D
7	b

The selection affects all the tables. Let us now retain this value, but instead make Table 2 loosely connected. This means that the logic between fields A and C is disconnected. The result is:

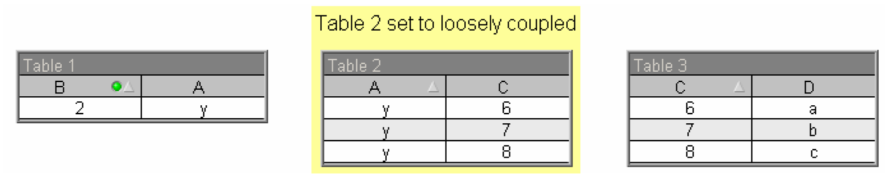


Table 2 set to loosely coupled

Table 1

B	A
2	y

Table 2

A	C
y	6
y	7
y	8

Table 3

C	D
6	a
7	b
8	c

Note that Table 2 shown above is a table box and not the actual table. The table box shows all the possible combinations of columns. As there is no logical connection between fields A and C, all the possible combinations of their values are shown.

As Table 2 is loosely connected, the selections made in Table 1 will not propagate to Table 3.

10. Exercises

1. Use the **Export Structure** button located on the **Document Properties: Tables** dialog to export the table structure data from your QlikView document.

2. Create a new QlikView document, and load data from the tables you exported in step 1.

3. Display the fields from this load on one or more sheets.

11. Creating data in QlikView

In our project plan, one of the **Key Dimensions** listed is *Sales Person*. Since there is no field included in our source data for *Sales Person*, we will need to generate this field in QlikView during the data load. In this chapter, we will practice Resident loads and conditional loads, as well as introduce the concept of creating multiple logical tables in QlikView based on a single source data table. We will also introduce script tabs.

11.1. Resident Load

In this section, we will learn how to create a new logical table in QlikView, based on a previously loaded (resident) table. We will also learn how to segregate the load script into separate tabs for easier reading and maintenance.

1. Open the **Edit Script** dialog.

2. Add a new tab to the script by clicking on the **Add New Tab** icon or selecting **Add Tab ...** from the **Tab** menu item. Name the new tab Sales Person.

3. We will now add another table load to the script, but this time, instead of using a wizard to create the code, we will copy existing code, and modify it. First, locate the *Employee* table load in the **Main** script tab. Copy (highlight and **<Ctrl>-C**, or **Edit ... Copy**) all the lines for this statement. Then switch back to the **Sales Person** tab, and Paste (**<Ctrl>-V**, or **Edit ... Paste**). You should now have a duplicate of the Employee table load in the new tab.

4. Now, edit the load statement as follows:
   - Change the table name to Sales_Person
   - Remove the rename of EmpID, and change to the QlikView field name of EmployeeID (we are no longer reading from the source data).
   - Remove the *[Hire Date]*, *Office*, *Extension*, *[Reports To]*, and *[Year Salary]* fields from the **load** statement.
   - Remove the comma from the Title field (it is now the last field in this load) and rename this field to *SalesTitle*.
   - Remove the **From** specification, since we will not be reading from a disk file for this load.
   - Add a **Resident** specification, pointing to the resident file we wish to load from, in the same location as the **From** was in.

5. Your script should now look as follows:

```
/* Sales Person Table comments */

Sales_Person:
Load
  EmployeeID,
   [Last Name],
   [First Name],
   Title as SalesTitle
Resident Employee;
```

6. Now we want to limit the load of all Employee data records to just those we can identify as a sales person. To do this, we need to make another change to the script code. First, remove the semicolon (;) located after **Resident Employee**. Next, add the where condition after the Resident line as follows:

```
where Left(Title,3) = 'Sal'
      OR Title = 'President';
```

7. Click **OK** in the Script Editor, and save the document. Then reload the data from the menus or toolbar.

8. Add list boxes for *[Last Name]*, *Title*, and *SalesTitle* to the **Main** sheet. **Select All** in the *SalesTitle* field, and notice that only some of the *[Last Name]* and *Title* values show as possible (white).

For a value of the original *Employee* table to be included in the new logical *Sales_Person* table, it must satisfy at least one of the conditions we have defined. The first condition is that the first three characters of the field *Title* must be *Sal*. The second condition is that the field be equal to *President.*

9. Save your document.

11.2. Synthetic key tables

It is undesirable to have multiple common keys across multiple tables in a QlikView data structure. This may cause QlikView to utilize complex keys (a.k.a. synthetic keys) to generate the connections in the data structure. Synthetic keys are generally resource heavy and may slow down calculations and, in extreme cases, over load an application. They also make an document harder to understand and maintain. There are some cases where synthetic keys cannot be avoided (e.g. Interval Match Tables), but, in general, synthetic keys should always be eliminated, if possible.

When we loaded the table that generates the field *SalesTitle*, we inadvertently created a synthetic key between the *Employee* and *Sales_Person* tables. The synthetic key is generated in a new synthetic key table *($ Syn 1 Table*) that can be seen from the **Table Viewer** utility under the **File** menu.

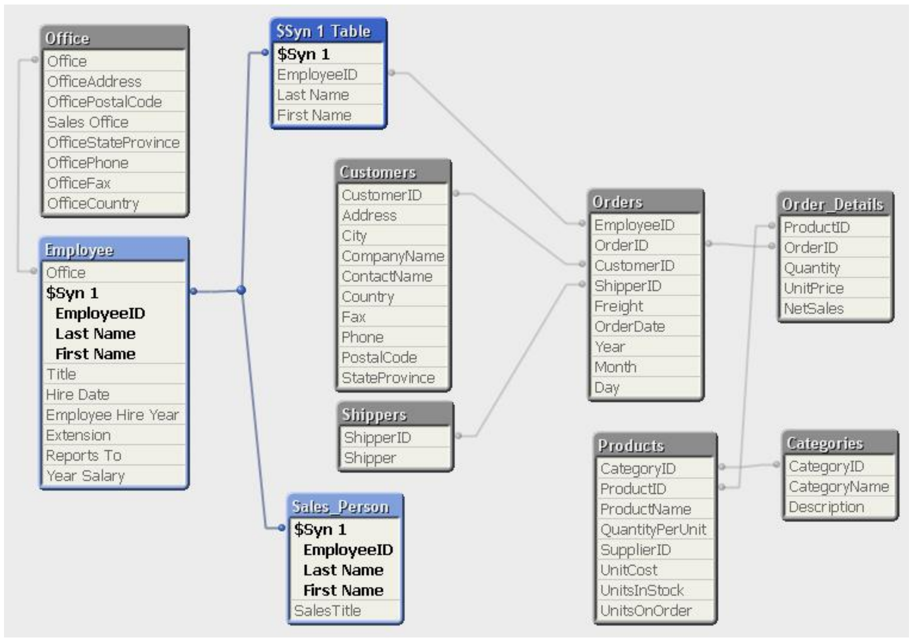


Figure 23. The Synthetic key tables as seen from Table Viewer

11.3. Removing the synthetic key table

As can be gathered from the **Table Viewer** we currently have a synthetic key table comprising the fields *First Name*, *Last Name* and *EmployeeID.* In this case, there is no reason why we need to use the name fields as an additional key between these tables, since *EmployeeID* serves this purpose already. To correct this situation, we will again employ the technique of renaming fields. In order to remove the redundant connections, we should rename the fields that we don't want to use as key fields. We may take this opportunity to introduce a further change by combining the fields *First Name* and *Last Name* into new *Name* fields by concatenation. Also, our project plan requires us to provide a Sales Person field consisting of first name and last name.

Make the following changes to the script lines that load the fields *First Name* and *Last Name* so that the script loading the two tables involved now reads:

```
Employee:
Load
    EmpID as EmployeeID,
      [First Name] & ' ' & [Last Name] as Name,
      [Last Name],
      [First Name],
      Title,
      [Hire Date],
      Year([Hire Date]) as [Employee Hire Year],
      Office,
      Extension,
      [Reports To],
      [Year Salary]
FROM Datasources\EmpOff.xls (biff, embedded
labels, table is [Employee$]);


…

Sales_Person:
Load
    EmployeeID,
    Name as [SalesPerson],
    Title as SalesTitle
Resident Employee
Where
    Left(Title,3) = 'Sal'
    OR Title = 'President';
```

After running the script, you can confirm that the complex key is gone by using the **Table Viewer** utility again. You can also add the new [Sales Person] list box to the Main sheet to view the results.

11.4. Advanced – Using Orders to determine Sales Person.

We just used a rather simplistic method that allowed us to sample the field values required for the field *SalesPerson*. This method is perfectly adequate, but we can obtain the same result by using a more elegant solution.

We begin by observing that all sales people are included in our sales data. It follows that they must occur in the field *EmployeeID* in the table *Orders*. We begin by creating a new field *EmployeeSales* in the script where *Orders* is being loaded. By referring to this field later in the script, we can insure that all employees that have been credited with sales will be listed in the *[Sales Person]* field.

1. To do this, enter the following script line in the *Orders* table **load** statement immediately after the loading of the *EmployeeID* field.

```
EmployeeID as EmployeeSales,
```

2. Next, change the **Where** condition in the *Sales_Person* table **load** statement to utilize the QlikView **exists** function. We will also add a unique *SalesPersonID* field to identify the SalesPerson.

```
Sales_Person:
Load
    EmployeeID,
    EmployeeID as SalesPersonID,
    Name as [SalesPerson],
    Title as SalesTitle
Resident Employee
Where
    Exists(EmployeeSales,EmployeeID);
```

The condition in the **Where** clause checks that the loaded data has matching values in the field *EmployeeSales*. As the name implies, the **exists** function can be used to check whether a specific field value exists in a specified field of the data loaded so far. Remember to be careful of the order of your script statements, since the reference field should be populated prior to checking for values. In this example, the *Orders* table must be loaded prior to the *Sales_Person* table for this condition to work properly. After completing these changes, and saving your document, you should be able to verify the functionality of your script.

12. Loading a DIF File

12.1. Text files in Data Interchange Format

The acronym DIF stands for **D**ata **I**nterchange **F**ormat and is used, for example, when transferring text files from the AS/400 environment. We will now add the *suppliers* table, which, as we saw earlier, is in DIF format, to our load script.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Position your cursor at the bottom of the **Main** tab, and add your comment for the *Suppliers* data table load.

3. Ensure that the **Wizard** check box is checked and then click the **Table Files…** button to open the **Open Local Files** dialog.

4. Browse to the file **suppliers.dif** in the folder **DataSources** and click **Open.**

5. Check that the wizard has interpreted the file format correctly.

Type: DIF, **Character Set**: OEM, **Embedded Labels**.

Note 1:
The correct character set for this file is OEM. Files created in Windows have ANSI as their standard character set. Macintosh uses the MAC character set. Sometimes, as in this case, other character sets are used when importing data from other computers. In this case, OEM is perhaps the most common. This character set is used in OS/2 and DOS, for example. If you specify the wrong character set, there is a risk that some characters will be lost, or that they will be interpreted incorrectly.

Note 2:
QlikView associates fields with the same name. If two tables have several fields in common, QlikView may create complex keys in order to associate the tables (see Synthetic key tables).

As in the *Office* table, **suppliers.dif** has a number of fields in common with the *Customers* table in the Access database. These fields should not be associated with each other, and some of the fields in **suppliers.dif** must be renamed. The only common field should be *SupplierID*, which is also found in the *Products* table.

As we will be renaming almost all the fields in **suppliers.dif**, we will use a special statement in QlikView that qualifies all the field names with the name of the table when they are loaded. The new field names will thus be *table name.field name*. As we want to rename all the fields except *SupplierID*, we will use *.

6. Click **Finish** to return to the Edit Script dialog and view the new Load statement generated for the Suppliers data.

7. Enter the following lines of code <u>before</u> the *suppliers* table **load**.

```
        Qualify *;
        Unqualify SupplierID;
```

As *SupplierID* is to be connected to another table, it should not have the table qualifier. This is specified using the statement **unqualify SupplierID.**

8. After the *Suppliers* table has been loaded, we must add the following statement so that all the fields loaded after this one do not have the table qualifier.

```
        Unqualify *;
```

You should now have the following lines in your script.

```
/* suppliers Table comments */

Qualify *;
Unqualify SupplierID;

suppliers:
Load
  SupplierID,
  CompanyName,
  ContactName,
  Address,
  City,
  PostalCode,
  Country,
  Phone,
  Fax
FROM Datasources\suppliers.dif (oem, dif, embedded
labels);

Unqualify *;
```

9. Click **OK** in the Script Editor, and save the document. Then reload the data from the menus or toolbar.

13. Key Fields

Key fields are fields that are common to one or more tables (associated fields). When a field occurs in more than one table, QlikView does not know which table to use in order to calculate the frequency of the data.

13.1. An example of problems that can arise

Assume that we have a table called *Orders* with 1000 different order numbers (*OrderID*). We also have table called *ForeignOrders*, which contains 200 order numbers. These numbers are also found in the first table.

The two tables will be associated via the common field *OrderID*. The problem arises when you want to know the exact number of unique *OrderID*. Is it 1000, 200 or 1200? We know, based on the information we have, that the correct answer is 1000 unique *OrderID*, but this is not at all clear to QlikView.

In this case, QlikView will look for a main table. It may choose the right one, but in most cases the program will have to guess.

As guesses can lead to serious consequences, QlikView has been designed so that it does not allow certain operations if there is any doubt as to which field is the correct one for calculating the frequency.

13.2. How does this affect you?

You should bear in mind the following limitations when working with key fields.

- In most cases, it is not possible to obtain information on frequency in a list box that shows associated fields. You will see that the checkbox **Show Frequency** is inactivated.
- In most cases, it will not be possible to use functions for calculating the frequency of associated fields in charts (Count, etc.) You must use a **Distinct** qualifier in these expressions.
- In general, it is good practice to avoid using key fields in list boxes and expressions. Key fields should be used for linking tables together, and not for displaying data in a QlikView document.

13.3. Solving the problem

There is a relatively simple solution to the problem of key fields and the calculation of data frequency. You load the field you want to use to calculate the frequency once again under another name.

The problem we described above can thus be solved in the following way:

```
Load
   …,
   OrderID,
   OrderID as OrderIDCount,
   …
from Order.xls (…);
```

You can now use the new field (not associated) in a list box, which shows the frequency or a chart with functions to calculate the frequency. The new field name can easily be concealed by giving it another label in order not to confuse users of the document.

13.4. Does the chart really show what I want it to?

When you create a chart in QlikView, it is naturally important to check that it really shows what you want it to show. Be careful to choose expressions that are suitable in each case. QlikView has a number of expressions. Below is a summary of these expressions together with what they show.

Salesperson	Article	CustomerNo	Quantity
Karl	A	10	100
Janne	A	101	200
Ola	B	10	250
Karl	B	111	350

Expression	Result
Total Count(Salesperson)	4
Total Count(Distinct Salesperson)	3
Total Count(Article)	4
Total Count(Distinct Article)	2
Num(Quantity)	4
Sum(Quantity)	900

14. Exercises

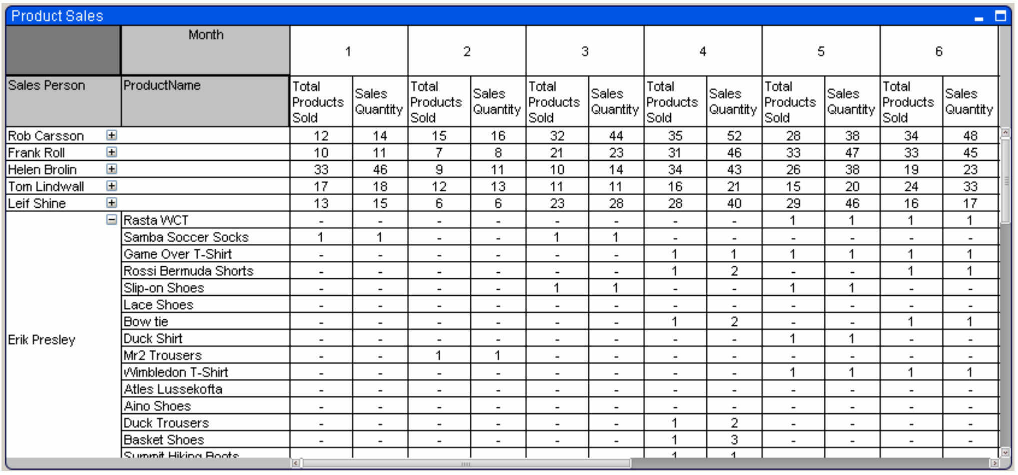
1. Modify the script in your QlikView document to include a field named *ProductIDCount*, based on the field *ProductID*. The new field will be used to produce the required **Key Measure** *Total Products Sold* as specified in our project plan.

2. Modify the script in your QlikView document to include a field named *CountOrderDate*, based on the field *OrderDate*. The new field will be used to count total number of Orders.

3. Create a pivot table chart, with the dimensions *SalesPerson*, *ProductName*, and *Month*. The expression should count the number of DISTINCT products sold, and should be labelled [Total Products Sold]. Show Partial Sums by *SalesPerson*, and *Month*.

4. Add a new expression for the total count of products sold, and label this [Sales Quantity].

5. Sort the *ProductName* and *SalesPerson* dimensions by the same expression as [Sales Quantity], in Descending order.


Product Sales

	Month	1		2		3		4		5		6	
Sales Person	ProductName	Total Products Sold	Sales Quantity	Total Products Sold	Sales Quantity	Total Products Sold	Sales Quantity	Total Products Sold	Sales Quantity	Total Products Sold	Sales Quantity	Total Products Sold	Sales Quantity
Rob Carsson		12	14	15	16	32	44	35	52	28	38	34	48
Frank Roll		10	11	7	8	21	23	31	46	33	47	33	45
Helen Brolin		33	46	9	11	10	14	34	43	26	38	19	23
Tom Lindwall		17	18	12	13	11	11	16	21	15	20	24	33
Leif Shine		13	15	6	6	23	28	28	40	29	46	16	17
Erik Presley	Rasta WCT	-	-	-	-	-	-	-	-	1	1	1	1
	Samba Soccer Socks	1	1	-	-	1	1	-	-	-	-	-	-
	Game Over T-Shirt	-	-	-	-	-	-	1	1	1	1	1	1
	Rossi Bermuda Shorts	-	-	-	-	-	-	1	2	-	-	1	1
	Slip-on Shoes	-	-	-	-	1	1	-	-	1	1	-	-
	Lace Shoes	-	-	-	-	-	-	-	-	-	-	-	-
	Bow tie	-	-	-	-	-	-	1	2	-	-	1	1
	Duck Shirt	-	-	-	-	-	-	-	-	1	1	-	-
	Mr2 Trousers	-	-	1	1	-	-	-	-	-	-	-	-
	Wimbledon T-Shirt	-	-	-	-	-	-	-	-	1	1	1	1
	Atles Lussekofta	-	-	-	-	-	-	-	-	-	-	-	-
	Aino Shoes	-	-	-	-	-	-	-	-	-	-	-	-
	Duck Trousers	-	-	-	-	-	-	1	2	-	-	-	-
	Basket Shoes	-	-	-	-	-	-	1	3	-	-	-	-
	Summit Hiking Boots							1	1				

Figure 24. The solution

15. Additional Time Dimension fields

There are two (2) additional time dimension fields required in our QlikView document according to the **Trends** section of our project plan. We need to add *Quarter* and *Rolling Month*. We will use a fairly simple technique for generating *Quarter* that will introduce the concepts of **Load Inline**, **autogenerate** and **Mapping** tables. For *Rolling Month*, we will introduce some additional QlikView date functions, as well as how to specify date formats.

15.1. Inline tables

You may remember that earlier in the course, we talked about a number of ways to load data in the QlikView script. We have already been introduced to many of these techniques, and now we will meet a new one.

In some cases, it may be advantageous to enter table data directly in the script. This is done with the aid of a so-called **load inline** statement.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Position your cursor near the top of the **Main** tab, prior to any table loads, but after the **SET** statements. Add a comment for loading Quarter data.

3. Add the following statement to the script:

```
Quarters:
Load * Inline [
Month, Quarter
1,Q1
2,Q1
3,Q1
4,Q2
5,Q2
6,Q2
7,Q3
8,Q3
9,Q3
10,Q4
11,Q4
12,Q4];
```

Notice that a load inline contains the field names and data enclosed by square brackets ([]). Also notice the field names are located on the first line, and that data values are separated by commas. The table entered in the script associates numeric months to the corresponding quarter. When we run the script, a new field (*Quarter*) is generated.

Click **OK** in the Script Editor, and save the document. Then reload the data from the menus or toolbar.

TIP: Inline tables can also be generated by means of the **Inline Wizard** that is opened from a button in the **Inline Data** group in the script editor.

15.2. Autogenerate tables

Another way to generate data in QlikView is to use the **autogenerate** clause on the **load** statement. Specifying **autogenerate** on a **load** statement will automatically generate a number of records. Only constants and parameter-free functions are allowed in an **autogenerate load**. Quite often, the **recno()** or **rowno()** functions are used to provide a unique number for each row.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Position your cursor after the *Quarters* tables load we just added in the previous step on the **Main** tab. Be sure to comment out that **load** statement, since we will be replacing it with another alternative. If you add **REM** in front of the table label, that will comment the entire statement.

3. Add the following statement to the script:

```
Quarters:
Load
   rowno() as Month,
   'Q' & Ceil(rowno()/3) as Quarter
Autogenerate(12);
```

The **rowno()** function will return the current row number of the QlikView logical table that is being created, starting with 1. The **Ceil** (ceiling) function will round the number passed up to the nearest integer. The **&** character is used for string concatenation in QlikView.

Click **OK** in the Script Editor, and save the document. Then reload the data from the menus or toolbar.

Remember to use only <u>one</u> of the *Quarters* **load** statements, and comment out the other.

15.3. Mapping tables

This table is useful, in that it links our Month data in the *Orders* table with the correct Quarter. However, the *Month* field is now a key field, and this will probably cause problems later. There are a few solutions to this, and we will investigate one of them now.

By changing our Quarters **load inline** table into a **mapping** table, we will be able to integrate the *Quarters* field into the same table as *Month* (the *Orders* table).

The **mapping** prefix is used on a join or select statement to create a mapping table. Tables read via **mapping load** or **mapping select** are treated differently from other tables. They will be stored in a separate area of memory and used only as mapping tables during script execution. After script execution they will be automatically dropped.

A mapping table must have two columns, the first one containing 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 input tables. When mapping tables are used to map a certain field value or expression, that value will be compared to the values in the first column of the mapping table. If found, the original value will be replaced by the corresponding value in the second column of the mapping table. If not found, no replacement is made.

The syntax is:
mapping *(load statement | select statement)*

Now, change the Quarters table load into a mapping load as follows:

```
Quarters_Map:
Mapping Load …
```

If you reload the data now, you will lose the *Quarters* table and field, since mapping tables are temporary. However, we can use the *Quarters_Map* table in our script (as long as we use it after it is defined in the script). To do this, we will use the **applymap** function.

The syntax is:
applymap(*'mapname', expr, [, defaultexpr]* **)**

The **applymap** function maps any expression on a previously loaded mapping table. *Mapname* is the name of a mapping table previously loaded by a **mapping load** or **mapping select** statement. The name must be quoted with single quotes. *Expr* is the expression whose result shall be mapped. *Defaultexpr* is an optional expression, which will be used as default mapping value if the mapping table does not contain any matching value for expr. If no default is provided, the value of *expr* is returned as is.

Add an **applymap** function to the *Orders* table, based on the numeric value of *Month*. This function should refer to the *Quarters_Map* table. The function should look as follows:

```
applymap('Quarters_Map',num(Month(OrderDate))) as
Quarter,
```

Click **OK** in the Script Editor, and save the document. Then reload the data from the menus or toolbar. Then add the field *Quarter* to the Main sheet, and check your work.

15.4. Rolling Month

We will complete our time dimension fields by creating a new field that makes every month unique. There are, of course, several ways to accomplish this. In this course, we will create the field *Rolling Month* by using QlikView date functions based on the *OrderDate* field, along with a date formatting function to provide the correct display format for our new month field. We will not discuss limiting the *Rolling Month* field values to the last 24 months, as that topic is discussed in the *Time Dimension* training module.

1. Open the **Edit Script** dialog from the menu or toolbar.
2. Locate the *Orders* table load statement on the Main tab.
3. Create a new field (*Rolling Month*) in the **Load** statement for the table *Orders*, as follows:

```
   date(monthstart(OrderDate),'MMM-YYYY') as [Rolling Month],
```

The **monthstart** function returns the first day of the month of the *OrderDate* value. The **date** function then formats this value into a 3-character month name, followed by a 4-digit year. Since QlikView stores this field as both a text string (the format we just specified) and as a numeric, it can be sorted numerically, as you would expect.
We will learn more about formatting functions in a later chapter.

4. The complete Orders table load statement should now look as follows. Be sure your script matches this.

```
Orders:
Load
   CustomerID,
   EmployeeID,
   EmployeeID as EmployeeSales,
   Freight,
   OrderDate,
   OrderDate as CountOrderDate,
   Year(OrderDate) as Year,
   Month(OrderDate) as Month,
   date(monthstart(OrderDate),'MMM-YYYY') as [Rolling Month],
   applymap('Quarters_Map',num(Month(OrderDate))) as Quarter,
   Day(OrderDate) as Day,
   OrderID,
   ShipperID;

SQL SELECT * FROM Orders;
```

5. Click **OK** and then save your file. Run the script.

6. Now, replace the *Month* dimension in the pivot table chart you created in the second set of exercises with the *Rolling Month* field. You will have to set the properties for this field again as they were set for *Month*.

***TIP**:* For more detailed information about utilizing time dimension data in QlikView documents, please consult the *Time Dimension* training module.

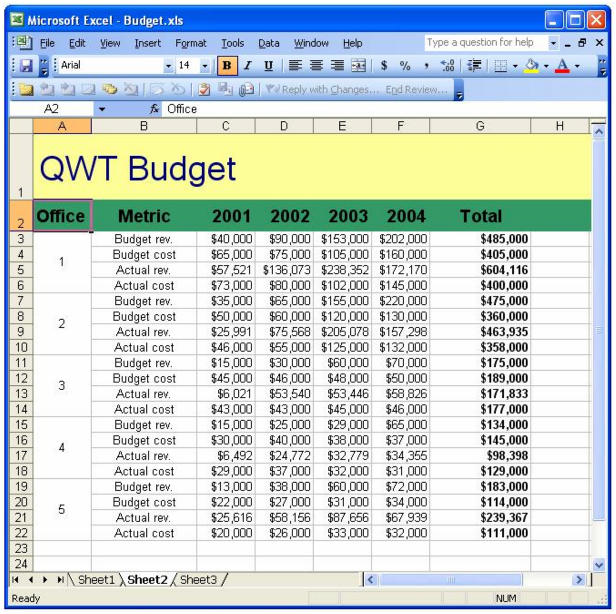
16. Crosstable and Generic Load

All the data we have loaded so far has been reasonably structured, and can be loaded through standard select or load statements. However, not all data is nicely formatted for a QlikView load. In this section we will explore some additional options available in QlikView to load non-standard data formats. An Excel file (**Budget.xls**) containing financial data can be found in the **DataSources** directory. This file contains data on two sheets. The first is very easily loaded into QlikView without any manipulation. On the second sheet, however, is data that you might typically find in an organization. This spreadsheet is formatted for viewing, and not necessarily for a QlikView load. Both sheets include fields giving the budgeted costs and income and the actual result for a number of recent years. We want to load the data on Sheet2 utilizing the **Crosstable** and **Generic Load** techniques.

16.1. Crosstable Load

Take a look at the Budget Excel file before loading the data. The data contained on Sheet2 is in a cross table format, meaning that it is a matrix of values between two lists of header data.

1. Open the **Budget.xls** file found in the **DataSources** directory. View Sheet2. This data format presents a number of challenges, but at a minimum, you should notice that a regular load into QlikView will produce fields for each Year, instead of year values in a single Year field, as we would like to have in our QlikView document.


Microsoft Excel - Budget.xls
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial
14
Reply with Changes... End Review...
A2
Office
Sheet1
Sheet2
Sheet3
Ready
NUM

QWT Budget

Office	Metric	2001	2002	2003	2004	Total
1	Budget rev.	$40,000	$90,000	$153,000	$202,000	**$485,000**
	Budget cost	$65,000	$75,000	$105,000	$160,000	**$405,000**
	Actual rev.	$57,521	$136,073	$238,352	$172,170	**$604,116**
	Actual cost	$73,000	$80,000	$102,000	$145,000	**$400,000**
2	Budget rev.	$35,000	$65,000	$155,000	$220,000	**$475,000**
	Budget cost	$50,000	$60,000	$120,000	$130,000	**$360,000**
	Actual rev.	$25,991	$75,568	$205,078	$157,298	**$463,935**
	Actual cost	$46,000	$55,000	$125,000	$132,000	**$358,000**
3	Budget rev.	$15,000	$30,000	$60,000	$70,000	**$175,000**
	Budget cost	$45,000	$46,000	$48,000	$50,000	**$189,000**
	Actual rev.	$6,021	$53,540	$53,446	$58,826	**$171,833**
	Actual cost	$43,000	$43,000	$45,000	$46,000	**$177,000**
4	Budget rev.	$15,000	$25,000	$29,000	$65,000	**$134,000**
	Budget cost	$30,000	$40,000	$38,000	$37,000	**$145,000**
	Actual rev.	$6,492	$24,772	$32,779	$34,355	**$98,398**
	Actual cost	$29,000	$37,000	$32,000	$31,000	**$129,000**
5	Budget rev.	$13,000	$38,000	$60,000	$72,000	**$183,000**
	Budget cost	$22,000	$27,000	$31,000	$34,000	**$114,000**
	Actual rev.	$25,616	$58,156	$87,656	$67,939	**$239,367**
	Actual cost	$20,000	$26,000	$33,000	$32,000	**$111,000**

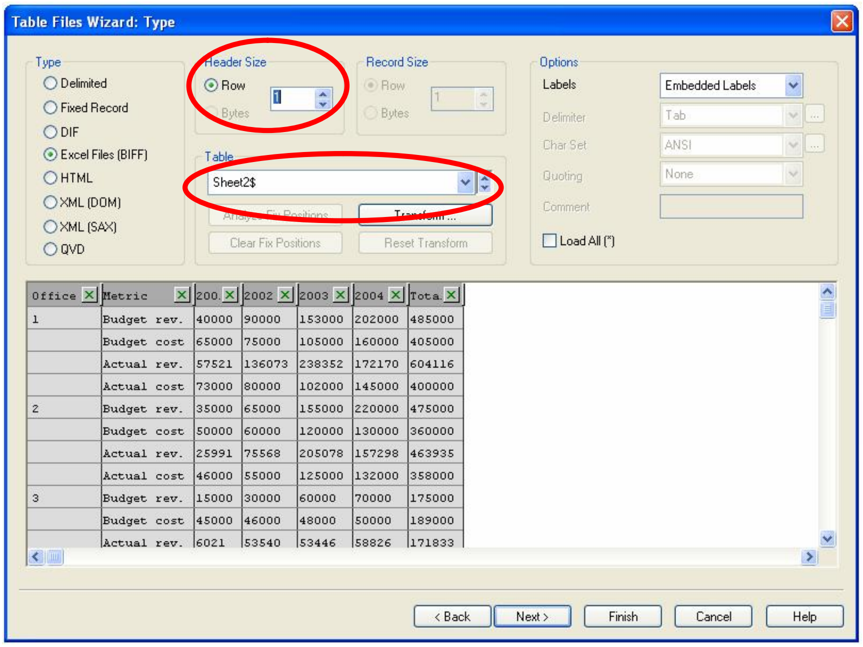
Figure 25. Budget spreadsheet data on Sheet2

2. Close the Excel file, and return to QlikView.

3. Open the **Edit Script** dialog.

4. Add a new tab to the script by clicking on the **Add New Tab** icon or selecting **Add Tab ...** from the **Tab** menu item. Name the new tab Budget/Actual.

5. First, we will use the **Table Files Wizard** to open the **Budget.xls** file located in the **Datasources** directory.



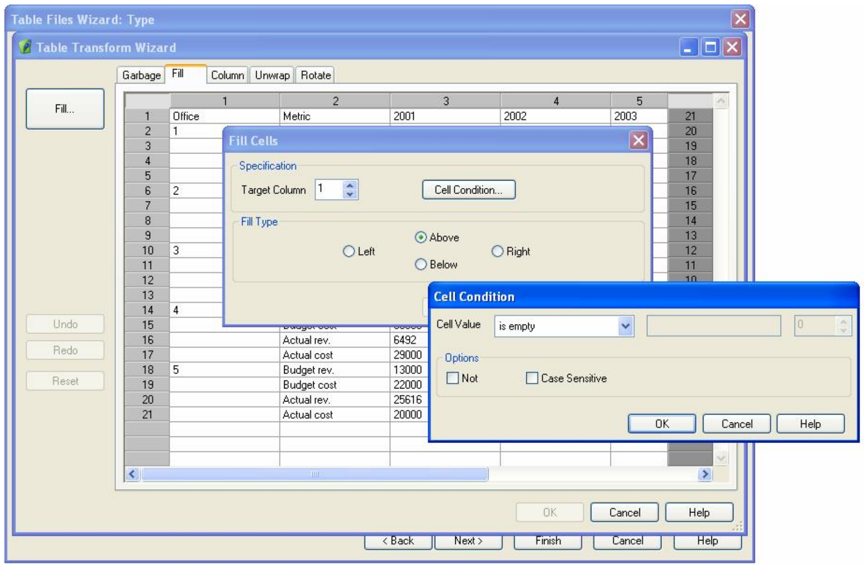
Type: [✓ Excel Files (BIFF)]
Header Size: [✓ Row] 1
Table: Sheet2$
Labels: Embedded Labels
Delimiter: Tab
Char Set: ANSI
Quoting: None

Office	Metric	200.	2002	2003	2004	Tota.
1	Budget rev.	40000	90000	153000	202000	485000
	Budget cost	65000	75000	105000	160000	405000
	Actual rev.	57521	136073	238352	172170	604116
	Actual cost	73000	80000	102000	145000	400000
2	Budget rev.	35000	65000	155000	220000	475000
	Budget cost	50000	60000	120000	130000	360000
	Actual rev.	25991	75568	205078	157298	463935
	Actual cost	46000	55000	125000	132000	358000
3	Budget rev.	15000	30000	60000	70000	175000
	Budget cost	45000	46000	48000	50000	189000
	Actual rev.	6021	53540	53446	58826	171833

Figure 26. Budget spreadsheet in Table File Wizard

Select **Sheet2** for **Table**. Notice that there is an extra header line in this data, so we need to increase the **Header Size** in order to read column Labels correctly. Change this value from 0 to **1** for **Row**.

6. Next, we need to fill in the empty cells for Office so that all the data connects properly. Click on the **Transform...** button on the **Table Files Wizard** dialog. Choose the **Fill** dialog sheet, and click on the **Fill...** button. Verify that the **Target Column** is set to **1**, and click on the **Cell Condition...** button. Verify that **Cell Value** is set to **is empty**, and click on **OK**. Click on **OK** for the **Fill Cells** dialog, and then click on **OK** for the **Fill** dialog.



	1	2	3	4	5
1	Office	Metric	2001	2002	2003
2	1				
3					
4					
5					
6	2				
7					
8					
9					
10	3				
11					
12					
13					
14	4				
15		Budget cost			
16		Actual rev.	6492		
17		Actual cost	29000		
18	5	Budget rev.	13000		
19		Budget cost	22000		
20		Actual rev.	25616		
21		Actual cost	20000		

Specification
Target Column: 1
Fill Type: ( ) Left [✓ Above] ( ) Below ( ) Right

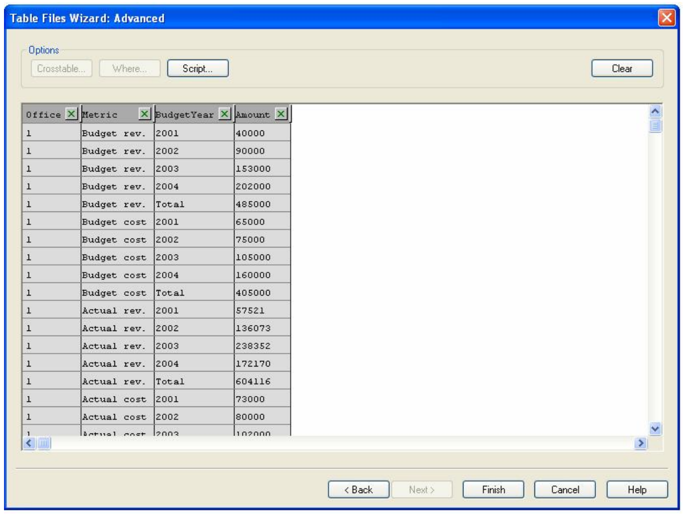
Cell Value: is empty
Options: [ ] Not [ ] Case Sensitive

Figure 27. Table File Wizard Transform …. Fill dialog

7. When you return to the **Table Files Wizard** dialog, you should notice that the cells for the Office column are now all filled in with values.

8. Next, we need to specify that this data is in a cross table format. Click on **Next…** to open the **Table Files Wizard: Advanced** dialog. Here, click on the **Crosstable…** button under **Options**.

9. You will then be prompted for **qualifying fields**. A **qualifying field** is simply a normal field, with data values in a column. A **qualifying field** will not be changed in a **crosstable load**. In this case, the fields *Office* and *Metric* are both qualifying fields. In the **crosstable load**, all qualifying fields must be placed before the attribute and data fields. Respond **No** when prompted for 2001 as a qualifying field. Enter **BudgetYear** for the name of the **attribute field**. Enter **Amount** for the name of the **data field**.



Office	Metric	BudgetYear	Amount
1	Budget rev.	2001	40000
1	Budget rev.	2002	90000
1	Budget rev.	2003	153000
1	Budget rev.	2004	202000
1	Budget rev.	Total	485000
1	Budget cost	2001	65000
1	Budget cost	2002	75000
1	Budget cost	2003	105000
1	Budget cost	2004	160000
1	Budget cost	Total	405000
1	Actual rev.	2001	57521
1	Actual rev.	2002	136073
1	Actual rev.	2003	238352
1	Actual rev.	2004	172170
1	Actual rev.	Total	604116
1	Actual cost	2001	73000
1	Actual cost	2002	80000
1	Actual cost	2003	102000

Figure 28. Crosstable Option complete

10. Click on **Finish…**to close the dialog.

You should now see the Crosstable load statement in your script. Remove the Directory statement, and add a table name of *Budget_Actual_S1* in front of the load. You may also want to remove the field names, and replace them with “*”. This will allow the script to read this spreadsheet even when the Year values change. Your script should now look as follows:

```
/* Budget Actual crosstable load comments */

Budget_Actual_S1:
CROSSTABLE(BudgetYear, Amount, 2)
Load
  *
FROM
  Datasources\Budget.xls (biff, header is line,
embedded labels, table is [Sheet2$], filters(
Replace(1, top, StrCnd(null))
));
```

11. Click **OK** and then save your file. Run the script.

12. Create a new sheet, either by clicking on the Add Sheet button in the Design toolbar or by selecting **Add Sheet** in the **Layout** menu.

13. Right-click on the new sheet, and select **Properties**. Enter the Title of the sheet as *Budget/Actual* on the **General** tab.

14. On the **Fields** tab, select the table *Budget_Actual_S1* and add all fields from this table. Notice that the Amount field contains all values for the Budget and Actual amounts, but we would prefer to have these values separated between different fields. To do this, we will need to implement another type of load against this data.

16.2. Generic Load

A set of generic data is one in which the field names are stored as field values in one column, while the field values themselves are stored in a second column. In our case, we have the *Metric* field (column), which contains the names of the fields we need, and the Amount field (column), which contains the all the values of the individual fields. One way to transform this data into separate fields, with their values is to use a **Generic Load** statement.

If you think of a cross table as data that has *values as fields*, then a generic table would have *fields as values*. We have seen how to change cross table data into standard data. Now we will see how to change generic data into standard data.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Position your cursor at the bottom of the Budget/Actual tab, and add the following statement to your script:

```
/* Budget Actual generic load comments */

Budget_Actual:
Generic Load
  Office,
  BudgetYear,
  Metric,
  Amount
Resident
  Budget_Actual_S1;

DROP TABLE Budget_Actual_S1;
```

Note that the order of the fields in a **Generic Load** are very important, since the last two fields listed will be used as the field names and values fields, respectively.

3. Click **OK** and then save your file. Run the script.

4. You can now add the fields *Actual cost*, *Actual rev.*, *Budget cost*, and *Budget rev.* to your layout.

The drawback to using a **Generic Load** is that it generally produces synthetic keys (multiple common keys between tables). You can see this by using the **Table Viewer**. Another limitation is that the resulting generic tables cannot be concatenated.

17. Include

It is possible to include files in the script that contain script or parts of a script. We will now see how this is done, and how we can use more advanced applications of this without writing anything directly in the script.

Let us first look at what we are going to add to the script.

Open the text file **Email.txt**, located in the Datasources directory using Notepad, or a similar tool. This file contains the following script.

```
Rem *** creates e-mail address;

LOAD
EmpID as EmployeeID,
IF((ord("First Name") >= 65 AND ord("First Name") <=
90), chr(ord("First Name")+32),
IF((Left("First Name",1)='Ä' OR Left("First
Name",1)='ä'), chr(97),
IF((Left("First Name",1)='Å' OR Left("First
Name",1)='å'), chr(97),
IF((Left("First Name",1)='Ö' OR Left("First
Name",1)='ö'), chr(111),Left("First Name",1)))))&
IF((ord("Last Name") >= 65 AND ord("Last Name") <=
90), chr(ord("Last Name")+32),
IF((Left("Last Name",1)='Ä' OR Left("First
Name",1)='ä'), chr(97),
IF((Left("Last Name",1)='Å' OR Left("First
Name",1)='å'), chr(97),
IF((Left("Last Name",1)='Ö' OR Left("First
Name",1)='ö'), chr(111), Left("Last Name",1)))))&
IF((ord(Right("Last Name",1)) >= 65 AND
ord(Right("Last Name",1)) <= 90), chr(Right("Last
Name",1))+32,
IF((Right("Last Name",1)='Ä' OR Left("First
Name",1)='ä'), chr(97),
IF((Right("Last Name",1)='Å' OR Left("First
Name",1)='å'), chr(97),
IF((Right("Last Name",1)='Ö' OR Left("First
Name",1)='ö'), chr(111), Right("Last Name",1)))))&
'@'&
IF(Office=1,'stockholm.se',
IF(Office=2,'lund.se',
IF(Office=3,'paris.fr',
IF(Office=4,'nice.fr','seattle.com')))) as "e-mail"
FROM datasources\empoff.xls(ansi, biff, embedded
labels, table is [Employee$]);
```

This is an example of quite a complicated **load** statement, which is mainly based on nested **if** statements. In this case, we want to create e-mail addresses from the information contained in our database. Many conditions must be satisfied, which leads to a complicated **load** statement that generates a new logical table containing two fields. We will load *EmployeeID* and the new field *e-mail*, the former giving us the association to the rest of the structure.

The **load** statement creates a signature consisting of the first letter of the first name, and the first and last letters of the last name. It also ensures that there are no capital letters in the signature, only lower case letters. Foreign (Swedish) letters, e.g. å, Å, ä, Ä, ö and Ö are also removed. Following the signature, @ is then inserted, followed by the appropriate server address. The latter is determined by the office in which the employee works.

We will now include the external file in our load script.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Position your cursor at the bottom of the **Main** tab.

3. Select **Include** from the **Edit** menu command.

4. Browse to the file **Email.txt** located in the Datasources directory, and click **Open**. The following line will be added to your script:

```
$(Include=datasources\Email.txt)
```

It is worth noting that there is no semicolon after the statement, but there can be one or more located within the included text file.

5. Click **OK** and save the QlikView document.

6. Run the script.

7. Add a new sheet to the layout, and name it *Employees*. Add the new field *e-mail* as a list box.

Of course, any part, or even the entire script can be located in an external Include file. There can also be multiple Include files in a script.

18. QlikView Data (QVD) Files

One of the most important changes affecting how scripts are written in QlikView is the introduction of QlikView Data (QVD) files. A QVD file contains a table of data exported from QlikView. QVD is a native QlikView format. It can only be written to and read from QlikView. The file format is optimized for speed when reading data from a QlikView script but it is also very compact. Reading data from a QVD file is typically 10-100 times faster than reading from other data sources.

18.1. QVD File Format

A QVD file contains exactly one table. Conceptually it is quite similar to any typed file (e.g. csv, dif, biff or fix). A QVD file is composed of three parts:

1. A well formed XML header (in UTF-8 char set) describing the fields in the table, the layout of the subsequent information and some other meta-data.
2. Symbol tables in a byte stuffed format.
3. Actual table data in a bit-stuffed format.

18.2. Use of QVD Files

QVD files can be used for many purposes. At least four major uses can be easily identified. In many cases two or more of them will be applicable at the same time.

- **Increasing Load Speed -** By buffering non-changing or slowly-changing parts of input data in QVD files, script execution can become considerably faster for large data sets. For large data sets it will thus be easier to meet reload time-window limitations. When developing applications it is often necessary to run the script repeatedly. By using QVD buffering in such situations repeated waiting times can be reduced significantly even if the data set is not that large.
- **Decreasing Load on Database Servers -** By buffering non-changing or slowly-changing parts of input data in QVD files, the amount of data fetched from external data sources can be greatly reduced. This reduces load on external databases and network traffic. When several QlikView scripts share the same data it is only necessary to load it once from the source database. The other applications can make use of the data from a QVD file.
- **Consolidating Data from Multiple QlikView Applications -** Consolidation of data from multiple QlikView applications is

possible with the help of QVD files. With the Binary script statement you can only load data from only one single QlikView application into another. With QVD files, a QlikView script can combine data from any number of QlikView applications. This opens up possibilities, e.g. for applications consolidating similar data from different business units etc.

- **Incremental Load -** In many common cases the QVD functionality can be used to facilitate incremental load, i.e. only loading new records from a growing database.

18.3. Creating QVD Files

QVD files can be created in three ways. We will explore the first two methods in this course.

1. Explicitly created and named from script by means of the **store** command. Simply state in the script that you want a previously read table or part of a resident table to be exported to an explicitly named filename at a location that you choose.
2. Automatically created and maintained from script. By preceding a **load** or **select** statement with the **buffer** prefix, QlikView will automatically create a QVD file which, during a later load, if certain conditions are met, will be used instead of the original data source when reloading data. The QVD file will have a cryptic name based on a hash of the **load/select** statement and normally reside in the Windows Application data folder.
3. Explicitly named and created from layout or via Automation. Data exported from the QlikView layout via GUI commands or Automation macros. In the GUI you will find QVD as one of the possible export formats under the **Export...** command, found on the object menu of most sheet objects.

18.4. Manual Creation of a QVD File in the Script

We will now create a QVD file in our load script by using the **store** statement. This statement will create an explicitly named QVD file.

The syntax for the **store** statement is:

store [**(*|**<field_list>**) from**] <table> **into** <file_name>;

Where:
<*table*> is a script labeled, resident, table.
<*file_name*> is interpreted similar to names in **load** statements, i.e. the **directory** statements apply.
Fields in the <*field list*> may be renamed using standard **as** syntax.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Locate the *Customers* table load statement on the Main tab.

3. Following the *Customers* table load statement, add the **store** statement as follows:

   ```
   store Customers into datasources/customers.qvd;
   ```

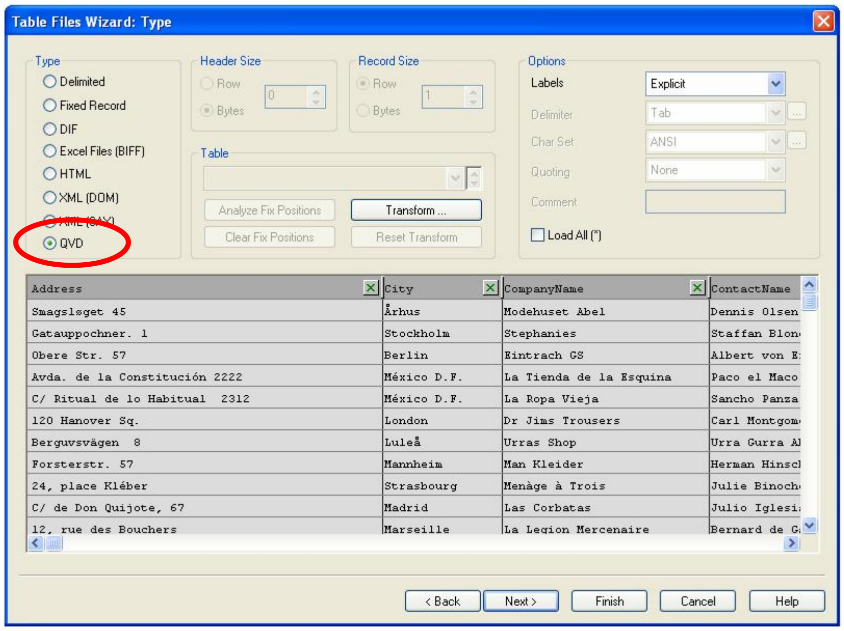
4. Click **OK** and then save your file. Run the script.

You will not notice any changes to the System sheet in your application. No new logical tables or fields exist. The **store** statement you just added has no effect on your current QlikView document, other than executing an additional script statement. Once this statement has executed, however, a new data file exists that may be read by this QlikView document, or any QlikView document with access to the folder which we placed the **customers.qvd** file. To test this, let's temporarily replace the *Customers* **select** statement with a new **load** statement from the **customers.qvd** file we just created.

5. Re-open the **Edit Script** dialog from the menu or toolbar.

6. Comment the existing *Customers* **select** statement <u>and</u> the **store** statement we just added.

7. Now add a new Table Files load statement using the **Table File Wizard**. Locate the **customers.qvd** file in the **Datasources** folder, and open.



Type: [✓ QVD]

Labels: Explicit

Delimiter: Tab

Char Set: ANSI

Quoting: None

Load All (*): ☐

Address	City	CompanyName	ContactName
Smagsløget 45	Århus	Modehuset Abel	Dennis Olsen
Gatauppochner. 1	Stockholm	Stephanies	Staffan Blon
Obere Str. 57	Berlin	Eintrach GS	Albert von E
Avda. de la Constitución 2222	México D.F.	La Tienda de la Esquina	Paco el Maco
C/ Ritual de lo Habitual 2312	México D.F.	La Ropa Vieja	Sancho Panza
120 Hanover Sq.	London	Dr Jims Trousers	Carl Montgom
Berguvsvägen 8	Luleå	Urras Shop	Urra Gurra A
Forsterstr. 57	Mannheim	Man Kleider	Herman Hinsc
24, place Kléber	Strasbourg	Menàge à Trois	Julie Binoch
C/ de Don Quijote, 67	Madrid	Las Corbatas	Julio Iglesi
12, rue des Bouchers	Marseille	La Legion Mercenaire	Bernard de G

Figure 29. QVD Table File Wizard

8. Notice the **Type** is correctly indicated as **QVD**. Select **Finish** to close the dialog.

9. You can remove the **Directory** statement, and add the table label *Customers* to the new load statement. Your script should now look as follows:

```
Customers:
Load
  Address,
  City,
  CompanyName,
  ContactName,
  Country,
  CustomerID,
  Fax,
  Phone,
  PostalCode,
  StateProvince
FROM Datasources\customers.qvd (qvd);
```

10. Click **OK** and then save your file. Run the script.

Again, you will not notice any changes to the System sheet in your application, since we did not change any tables – only the location and type from where we read the data. In a normal environment, you would probably also notice a big difference in the time it takes to read the *Customer* table, since a QVD file is read into QlikView extremely fast.

Of course, the drawback to this technique is that when our Customers database data changes, it will not be read into our QlikView document. We will address that in the next section.

18.5. Automatic Creation of a QVD File in the Script

QVD files can be also be created and maintained automatically via the **buffer** prefix. This prefix can be used on most **load** and **select** statements in the script. It indicates that a QVD file is used to cache/buffer the result of the statement. Some limitations exist, the most notable is that there must be either a file load or a **select** statement in “the bottom”. The name of the QVD file is a *calculated* name (a 160-bit hash of statement and other discriminating info, as hex) and is typically stored in the DATA folder:

C:\Document and Settings\%user%\Local Settings\Application Data\QlikTech\QlikView\Buffers

***TIP**:* To determine and/or change where QlikView will place QVD buffer files, as well as additional default folder locations, you can check under **Settings … User Preferences … Folders**.

The syntax of the prefix is:
buffer *[(option [,option])]* **load …**
or
buffer *[(option [,option])]* **select …**

where an *option* is either of the following:
- **incremental -** this enables the ability to read only part of an underlying file. Previous size of the file is stored in the XML header in the QVD file. This is particularly useful with log files. All records loaded at a previous occasion are read from the QVD file whereas the following new records are read from the original source and finally an updated QVD file is created.
- **stale (after)** *amount* **[(days | hours)] -** This is typically used with DB sources where there is no simple timestamp on the original data. Instead one specifies how old the QVD snapshot can be before it is replaced with a fresh read from the source file or database.

We will now revise our *Customers* table data load to use the automatic method of QVD file generation. We know from our project plan that *Customers* data is updated weekly, so we only need to read in updated data every 7 days. We will therefore change the script to add the correct QVD buffer prefix to our original *Customers* **select** statement.

1. Re-open the **Edit Script** dialog from the menu or toolbar.

2. Comment the *Customers* table **load** statement from QVD we added in the previous section, and uncomment the original *Customers* **select** statement.

3. Now add the prefix **buffer (stale after 7 days)** to the *Customers* **select** statement. Your script should now look as follows:

```
Customers:
buffer (stale after 7 days) SELECT
    Address,
    City,
    CompanyName,
    ContactName,
    Country,
    CustomerID,
    Fax,
    Phone,
    PostalCode,
    StateProvince
FROM Customers;
```

4. Click **OK** and then save your file. Run the script

When using the buffer prefix on load or select statements, no explicit statements for reading are necessary. QlikView will determine to which extent to use data from the QVD file or acquire data via the original **load** or **select** statement.
Regardless of the QVD method used, when no transformations are applied on the fields read (apart from renaming fields) the super-fast (optimized) reading mode will be used. You can determine what qvd mode was used by viewing the Script Execution Progress dialog,.

18.6. QVD File Script Functions

There are a number of new script functions that have been added for access to the data found in the XML header of a QVD file. These functions are described in the **File Functions** section of the QlikView Reference Manual. Here is a sampling of the new functions available:

QvdCreateTime(*filename* **) -** Returns the XML-header time stamp from a QVD file if any (otherwise NULL).

QvdNoOfRecords(*filename* **) -** Returns the number of records currently in a QVD file.

QvdNoOfFields(*filename* **) -** Returns the number of fields in a QVD file.

QvdFieldName(*filename* **) -** Returns the name of field number *field_no*, if it exists in a QVD file (otherwise NULL).

QvdTableName(*filename* **) -** Returns the name of the table contained in a QVD file.

19. Exercises

1. Modify the script in your QlikView document to use a **buffer** qvd load on the *Orders* and *Order_Details* tables.

2. Use the **stale** specification. Refer to the project plan to determine how often these tables need to be refreshed (i.e. how often is this data updated according to the business rules).

20. Advanced Scripting – Part 1

Two of the Key Performance Indicators, according to the project plan, require us to calculate the recent yearly change in NetSales by Customer. In this section, we will be introduced to a number of advanced scripting techniques in QlikView that we can use to provide this solution. Of course, as with most problems, there are multiple approaches to solving this problem. This particular approach will allow us to explore the concepts of:

- Join
- Aggregation
- Order by
- Previous (accessing data from the previous loaded record)
- Variables

20.1. Joining data from multiple tables

If we want to compare a Customer's *NetSales* between 2 previous years, we could do this simply by selecting individual years in the application interface, and comparing the chart results, or including *Year* as a dimension in a chart object calculating *NetSales*. This may be sufficient for some of our requirements, but to determine the overall Customer performance, we will create additional data describing yearly *NetSales* performance by Customer in the load script. We can then utilize this data to determine increase/decrease by Customer, as well as percentage change by Customer in our QlikView document.

We get the calculated field *NetSales* from the table *Order_Details*, but only for a certain year that is given in the field *OrderDate* and for a certain Customer in the field *CustomerID* in the table *Orders*. In order to use two tables together like this, we must begin by combining them into a single table. Here, the **Join** between tables can be performed against the source database, or by using a QlikView **Join** command. Since we already have the source data we need loaded into memory, we will use the QlikView **Join Load** statement against these two resident tables (*Order_Details* and *Orders*).

Let's begin by adding a new tab in our script called *Sales Change*.

6. Open the **Edit Script** dialog.

7. Add a new tab to the script by clicking on the **Add New Tab** icon or selecting **Add Tab ...** from the **Tab** menu item. Name the new tab Sales Change.

8. First, we need to load a new logical table based on the Orders resident table. We will name this new table *OrdersByYear_Source*.

```
/* Sales Change Source load comments */

OrdersByYear_Source:
Load
  CustomerID,
  OrderID,
  Year
RESIDENT
  Orders
Where
  Year = 2003 or Year = 2004; /*only load 2 years*/
```

We load the fields *CustomerID* and *Year* in order to group (aggregate) the data. The field *OrderID* will be used to link to the *Orders_Detail* table when we **Join** to that table. The **Where** clause will limit the data to just the order years 2003 and 2004.

Next, we will add the *NetSales* data to this table through a **Join Load**.

9. Add the following statement to your script:

```
Left Join (OrdersByYear_Source) Load
  OrderID,
  NetSales
RESIDENT
  OrderDetails;
```

Here we use a **Left Join** load because we only want to match on the orders loaded based on the **Where** clause we specified in the previous load. We need to make sure the field *OrderID* is included in this load to match the records from the *OrdersByYear_Source* table we are joining to. In QlikView, the default join behavior is a full outer join. Therefore, if there are no matching fields between the two joined tables, you will get a Cartesian product of the records. Since we are specifying *OrderID* in both tables, and we are specifying **Left**, only the records matching *OrderID* included in the *OrdersByYear_Source* table will be included. We include the *NetSales* field because that is what we want to add to the *OrdersByYear_Source* table we are creating.

10. Click **OK** and save the QlikView document.

11. Run the script.

12. Add a new sheet to the layout, and name it *Sales Change*. Add all fields from the *OrdersByYear_Source* table. Notice that all these fields are Key fields. Can you explain why?

20.2. Aggregating data

To group or aggregate data, we will use the **Group by** clause in the **Load** statement. In this case, we need to aggregate the data in the *OrdersByYear_Source* table by *CustomerID* and *Year*.

1. Open the **Edit Script** dialog.

2. Add the following statement to your script:

```
/* Aggregation by Customer comments */

OrdersByYear:
Load
  CustomerID,
  Year,
  sum(NetSales) as NetSalesByYear
RESIDENT
  OrdersByYear_Source
Group by
  CustomerID,Year;
```

Notice the aggregation function *sum(NetSales)* included in this statement. This function will be evaluated over all the potential combinations of the other fields in the **Load** (*CustomerID* and *Year*) statement. Typically, you will always **Group by** the fields in the **Load** that are not aggregation fields. This field is given a new name to differentiate it from *NetSales*. The *CustomerID* and *Year* fields will be used as Key fields (although in the next section we will drop the *Year* field.

TIP: To access additional information about aggregation functions available in the load script, please refer to the **QlikView Reference Manual**.

3. We can also **Drop** the temporary table we used for the join, since we no longer need this table.

```
DROP TABLE OrdersByYear_Source;
```

4. Click **OK** and save the QlikView document.

5. Run the script.

6. Add the field *NetSalesByYear* to the *Sales Change* sheet. Notice that if you select a *CustomerID*, you will generally see two values associated in this new field (the two years of *NetSales* aggregated to each year).

20.3. Calculating data Between Records

In order to calculate yearly change for each Customer, we can load this data again, using an **Order by** clause to sort the records properly, and a **Previous** function to access data from the previous record.

1. Open the **Edit Script** dialog.

2. Add the following statement to your script at the bottom of the *Sales Change* tab:

```
/* Sales Change by Customer comments */

Sales_Change:
Load
   CustomerID,
   NetSalesByYear,
   If(CustomerID = Previous(CustomerID),
      If(NetSalesByYear > Previous(NetSalesByYear),
         'UP','DOWN'),null()) as SalesChangeIndicator,
   If(CustomerID = Previous(CustomerID),
      NetSalesByYear - Previous(NetSalesByYear),
         null()) as SalesChangeAmt
RESIDENT
   OrdersByYear
Order by
   CustomerID,Year;
```

First, notice the **Order by** clause at the end of the statement. This will insure that the records are sorted properly – by *CustomerID* and *Year*. The default sort order used by QlikView is ascending, so the year 2003 record for a customer (if it exists) will be loaded first.

Now notice the first **If** function, used for the *SalesChangeIndicator* field. This is a nested If, in the format:

If(<condition>, [If(<condition>, then, else)], else)

The first <condition>: **CustomerID = Previous (CustomerID)** will compare the current record value for the field CustomerID against the previous input record that was not discarded due to a where clause. In other words, this condition checks to see if both records are for the same Customer. The second <condition>: **NetSalesByYear > Previous(NetSalesByYear)** will compare the current record value for the field *NetSalesByYear* against the previous input record value for *NetSalesByYear* . So, if the Customer is the same, and the *NetSalesByYear* is greater in 2004 than in 2003, the result in the field *SalesChangeIndicator* will be "UP". If the *CustomerID* is not the same, this field will be assigned a value of null().

The field *SalesChangeAmount* uses a similar **If** function, although not a nested **If**. In this case, if the Customer is the same between records, the difference between the *NetSalesByYear* in 2004 and *NetSalesByYear* in 2003 will be calculated, otherwise, the field will be set to null().

7. We can also **Drop** the temporary table we used for the aggregation, since we no longer need this table.

```
DROP TABLE OrdersByYear;
```

8. Click **OK** and save the QlikView document.

9. Run the script.

10. Add the fields *SalesChangeIndicator* and *SalesChangeAmount* to the *Sales Change* sheet. You may need to set the Number format for the *SalesChangeAmount* field to 2 decimals through the **Settings … Document Properties … Number** dialog.

You may also notice that the field *NetSalesByYear* is no longer relevant, since it is no longer connected to the *Year* data. We included this field in our final *Sales_Change* table for clarity (although it was not technically needed by QlikView), but we do not need this field once the load is completed. We can therefore **Drop** the field using the **Drop Field** statement.

```
DROP FIELD NetSalesByYear;
```

11. Click **OK** and save the QlikView document.

12. Run the script.

The field *NetSalesByYear* should now show as unavailable.

20.4. Using Variables in the Script

It is important to design your QlikView load scripts so that they are flexible and easy to maintain. One of the ways to help achieve this goal is to use script variables. Variables can be set to a constant – numeric or string - or can be assigned a value through an evaluated expression. Use the **SET** statement to assign a constant value, and the **LET** statement to assign a value through an evaluated expression. It is also a good idea to name variables uniquely so that they may be easily identified in the QlikView document. Variables defined in the load script are available in the document interface.

In this case, let us assign a value to the current year for NetSales comparison. Previously, we hard-coded this value to 2004 in the script. First, we will assign a constant value to the variable, and then we will change that to be dynamic based on the current system date.

1. Open the **Edit Script** dialog.

2. Position your cursor on the **Main** tab at the top. Add a new **SET** statement to define our *vCurrentFullYear* variable. In general, you should always assign variables at the beginning of the load script so that they are available throughout the script.

   ```
   SET vCurrentFullYear = 2004;
   ```

3. Now, move to the **Sales Change** tab. We need to edit the Where clause for the *OrdersByYear_Source* table. Change this as follows:

```
/* Sales Change Source load comments */

OrdersByYear_Source:
Load
  CustomerID,
  OrderID,
  Year
RESIDENT
  Orders
Where
  Year = $(vCurrentFullYear) - 1 or Year =
$(vCurrentFullYear); /*only load 2 years*/
```

QlikView will substitute the value of *vCurrentFullYear* in the script statement when it is run. In this case, the value will be 2004.

4. Click **OK** and save the QlikView document.

5. Run the script.

There should be no changes to the field values after the new script is run.

This will work fine until next year. Instead of requiring a script change every year to maintain this variable, we can assign a value to it using the system date as input. Since this assignment will require an expression evaluation, we will need to change our **SET** statement to a **LET** statement.

6. Open the **Edit Script** dialog.

7. Position your cursor on the **Main** tab at the top. Change the **SET** statement to define our *vCurrentFullYear* variable to a **LET** statement as follows.

   ```
   LET vCurrentFullYear = Year(today()) - 1;
   ```

This statement will evaluate the year of today's date (the run date for the script), and subtract 1 to obtain the current full year.

8. Click **OK** and save the QlikView document.

9. Run the script.

There should be no changes to the field values after the new script is run.

21. Exercises

In this exercise, we will take a break from script writing, and create a Straight Table chart using fields from the *Sales_Change* table we loaded in the previous section. The Straight Table will include built in QlikView graphics to provide a visual cue indicating change direction (up or down).

1. On the Sales Change sheet that was recently added in the document interface, choose **Layout ... New Sheet Object ... Chart**. Enter **Net Sales Change by Customer** as the **Window Title**. Select **Straight Table** for the **Chart Type**, and add the field *CompanyName* as the only **Dimension**. Label the dimension as **Customer**.

2. Add a NetSales expression [sum(NetSales)].

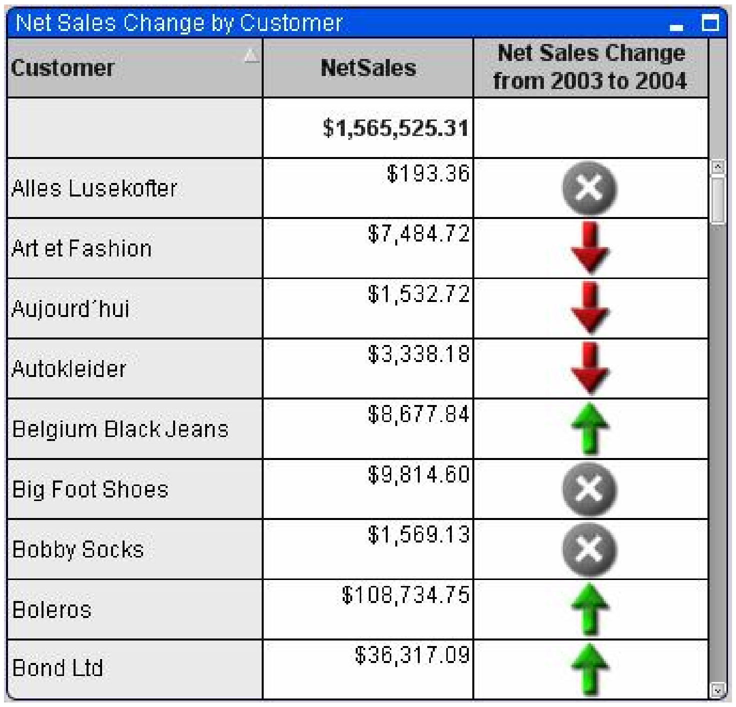
3. Add another expression as follows:

   ```
   If(SalesChangeIndicator =
   'UP','qmem://<bundled>/BuiltIn/arrow_n_g.png',

   If(SalesChangeIndicator =
   'DOWN','qmem://<bundled>/BuiltIn/arrow_s_r.png',

   'qmem://<bundled>/BuiltIn/cross.png'))
   ```

   ***TIP**:* Use the **Images** section of **Edit Expression** to paste the QlikView **qmem** function with the appropriate arrow/cross graphic.

   Label this expression with a dynamic string, including the Sales Change comparison years. Hint: use the *vCurrentFullYear* variable we created in the load script.

4. With a little formatting, your new chart object should look something like the one on the following page. If you see the **qmem** function instead of the graphic listed in the *Sales Change* column, be sure to specify **Image** as **Representation** and **Keep Aspect** as **Image Formatting** for the *Sales Change* expression.



Net Sales Change by Customer

Customer	NetSales	Net Sales Change from 2003 to 2004
	$1,565,525.31	
Alles Lusekofter	$193.36	✖
Art et Fashion	$7,484.72	⬇
Aujourd´hui	$1,532.72	⬇
Autokleider	$3,338.18	⬇
Belgium Black Jeans	$8,677.84	⬆
Big Foot Shoes	$9,814.60	✖
Bobby Socks	$1,569.13	✖
Boleros	$108,734.75	⬆
Bond Ltd	$36,317.09	⬆

Figure 30. Sales Change Straight Table chart

22. Advanced Scripting – Part 2

Another way to join data together from multiple tables is to use concatenation.

22.1. Concatenation

Automatic Concatenation

If the field names and the number of fields of two or more loaded tables are exactly the same, QlikView will automatically concatenate the results of the different **load** or **select** statements into one table.

Example:

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

The resulting logical 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.

Rules:

- The number and names of the fields must be exactly the same.
- The order of the fields listed in each statement is arbitrary
- The order of the two statements is arbitrary.

Forced Concatenation

If two or more tables do not have exactly the same set of fields, it is still possible to force QlikView 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 logical table.

Example:

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

The resulting logical 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.

Rules:

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

Prevent Concatenation

If two tables have the same set of fields and thus would normally be automatically concatenated, you can prevent the concatenation with the **noconcatenate** prefix. This statement prevents concatenation with any existing logical table with the same set of fields.

The syntax is:
noconcatenate *(loadstatement | selectstatement)*

Example:

```
Load A,B from file1.csv;
Noconcatenate load A,B from file2.csv;
```

Example

In our data, we have been provided with an additional set of new employees that are not yet contained in the EmpOff.xls file. In order to add this data, we need to modify our load script.

1. Open the **Edit Script** dialog.

2. Position your cursor on the **Main** tab directly after the *Employee* table has been loaded. We need to duplicate the fields we currently have for *Employee*, so we will not use the **Table Wizard** in this case. Instead, **copy** the *Employee* **load** statement, and **paste** the copied text after the original text.

3. Since the new file data format matches our first file, we only need to change the source of the data. Revise the From clause in the new load statement to read as follows:

```
FROM Datasources\Employees_New.xls (biff, embedded
labels, table is [Employee$]);
```

4. Click **OK** and save the QlikView document.

5. Run the script.

If you notice a number of new **Synthetic Keys**, or a new *$Table* value of *Employee-1*, you know something did not work correctly with automatic concatenation.

6. You can avoid a number of potential problems with automatic concatenation by using the **concatenate** prefix on **load** statements that you know should be concatenated. Add the **concatenate** prefix to the new *Employee* **load** statement, and specify the *Employee* table. This will always concatenate these two tables together, even if inadvertent script changes are made later to one of the loads, but not the other. The new Employee load statement should now begin as follows:

   ```
   concatenate (Employee) Load
   ```

7. You may have noticed that there are very few differences between our two *Employee* **load** statements. In fact, we can use another QlikView feature to load the same data in just a <u>single</u> **load** statement. By using a wildcard specification on the **From** file name, QlikView will automatically load from all files matching that specification, and concatenate the data into a single logical table for you. Since both our file names start with "Emp", and have the ".xls" file extension, we can use the wildcard "Emp*.xls" in the From clause. If we make this change, and comment the second Employee load statement, the script should now read as follows:

```
Employee:
Load
   EmpID as EmployeeID,
  [First Name] & ' ' & [Last Name] as Name,
  [Last Name],
  [First Name],
  Title,
  [Hire Date],
  Year([Hire Date]) as [Employee Hire Year],
  Office,
  Extension,
  [Reports To],
  [Year Salary]
FROM Datasources\Emp*.xls (biff, embedded labels,
table is [Employee$]);

REM Employee:
concatenate (Employee) Load
   EmpID as EmployeeID,
  [First Name] & ' ' & [Last Name] as Name,
  [Last Name],
  [First Name],
  Title,
  [Hire Date],
  Year([Hire Date]) as [Employee Hire Year],
```

```
    Office,
    Extension,
    [Reports To],
    [Year Salary]
FROM Datasources\Employees_New.xls (biff, embedded
labels, table is [Employee$]);
```

8. Save the revised script and the QlikView document. Then reload, and verify the *Employee* data has not changed.

9. As an optional exercise, you may want to try to determine why the employees listed in the Employees_New.xls file are not assigned email addresses (field *e-mail* is null for these employees). What do you need to do to correct this problem?

23. Security

Computer security is an important element in this course. QlikView documents often contain confidential information, and it is important to know how to prevent access by unauthorized users.

23.1. Access control

A QlikView document is an encrypted file consisting of a database, script, layout, etc. The file format itself may provide some intrinsic protection, as it is not possible to open the file if you do not have QlikView. It is also possible to include access levels in the load script.

23.2. Access levels

Each user of the QlikView document must be assigned an access level: ADMIN or USER. A user with ADMIN privileges can change everything in the document (subject to product limitations), whereas a person with USER privileges has restricted access. A user of QlikView Analyzer+ will be automatically restricted to USER privileges, regardless of section access settings. If no access level is assigned to a user in section access, the user cannot open the QlikView document.

For clarity, it may be useful to use other access levels, e.g. NONE. These will always be treated as “no access”.

23.3. Access control database

All access control is managed via text files, databases or **inline** clauses in the same way as data is normally handled by QlikView. The tables are loaded in the normal way, but in an access section that is initiated in the script by the statement **section access**.

***TIP**:* Be aware that all field names and values will be automatically converted to upper case in section access. This must be taken into consideration when using preceding load or resident load statements within section access.

If such an access section is declared in the load script, the part of the script that loads standard data must be preceded by the statement **section application**. There are a number of protected field names in the access control database, including: *USERID*, *PASSWORD*, *SERIAL*, *NTNAME*, *NTDOMAINSID*, *NTSID*, and *ACCESS*. Other user-defined fields may be added, e.g. *GROUP* or *DEPARTMENT*, to facilitate dynamic data reduction or for administration, but QlikView does not use the extra fields for limiting access to the document. None, all or any combination of the security fields may be loaded in the access section. If none of the security fields are loaded, all the users will have ADMIN rights.

It is thus not necessary to use USERID – a check can be made on SERIAL only, for example. This fact can be used for command-line reloads of access restricted documents.

- **ACCESS** A field that defines what access the user should have.
- **USERID** A field that should contain a user ID that has the privilege specified in the field ACCESS.
- **PASSWORD** A field that should contain an accepted password.
- **SERIAL** A field that should contain a number corresponding to the QlikView serial number. Example: 4900 2394 7113 7304
- **NTNAME** A field that should contain a string corresponding to a Windows NT Domain user name or group name.
- **NTDOMAINSID** A field that should contain a string corresponding to a Windows NT Domain SID.
  Example: S-1-5-21-125976590-467238106-1092489882
- **NTSID** A field that should contain a Windows NT SID.
  Example: S-1-5-21-125976590-467238106-1092489882-1378

QlikView will compare the QlikView serial number with the field SERIAL, the Windows NT User name and groups with NTNAME, the Windows NT Domain SID with NTDOMAINSID and the Windows NT SID with NTSID. It will further prompt for User ID and Password and compare these with the fields USERID and PASSWORD.

If the found combination of user ID, password and environment properties is found also in the Section Access table, then the document is opened with the corresponding access level. If not, QlikView will deny the user access to the document. If the user ID and/or the password are not entered correctly within three attempts the entire logon procedure must be repeated.

In the logon procedure, QlikView will first check SERIAL, NTNAME, NTDOMAINSID, and NTSID to see if this information is enough to grant the user access to the document. If so, QlikView will open the document without prompting for USERID and PASSWORD. If only some of the access fields are loaded, the appropriate of the above requirements are used.

All the fields listed in **Load** or **Select** statements in the section access must be written in UPPER CASE. Any field name containing lower case letters in the database will be converted to upper case before being read by the Load or Select statement. However, the USERID and the PASSWORD entered by the end-user opening the QlikView documents are case insensitive.

23.4. Inherited access restrictions

A **binary** statement will cause the access restriction to be inherited by the new QlikView document that contains the **binary** statement. A person with ADMIN rights to this new document may change the access rights of the new document by adding a new access section. A person with USER rights can execute the script and change the script (by adding their own data to the file loaded with the **binary** statement). A person with USER rights cannot change the access rights. This makes it possible for a database administrator to control user access, including those that start with a binary statement.

23.5. Section access in script

We will now add the necessary lines to our script to check the access rights of various users. It is generally good practice to place the script code for section access in the “hidden script” area.

Hidden Script

A hidden script is a password protected, hidden area of script code that is always executed prior to the standard script during a reload.

When choosing **Edit Hidden Script** from the **File** menu in the **Edit Script** dialog you will be prompted for a password, which will be required before giving access to the hidden script again. If it is the first time you access the hidden script in a document (thereby creating one), you will have to confirm the new password. After this the Hidden Script tab will appear to the left of all other script tabs and remain there until you close the document. Keep in mind the following aspects of hidden scripts if you choose to use them:

- If a hidden script is used, the **binary** command cannot be used in the normal script, since it must be the first statement executed in a document script.
- Tables generated by the hidden part of the script will not be represented by name in the *$Table* system field.
- The **Script Execution Progress** dialog will not be updated during the execution of a hidden script. No entries will be made in the log file, if used. Note that as of QlikView version 7, there is available a **Document Security** override to **Show Progress for**

Hidden Script. The progress information will also be written to the script execution log, if applicable.

- If the hidden script contains a **Section Access**, such a section will not be permitted in the normal script nor in a script starting with a binary load of the QlikView file containing the hidden script.

Adding Section Access

We will now add a basic section access in the hidden script area of our QlikView document.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Add a hidden script by selecting **File ... Edit Hidden Script ...** Use **qlikview** as the password for the hidden script. Then, add the following lines of script:

```
/* SECURITY section access comments */

Section Access;

Access01:
Load * Inline
[USERID, PASSWORD, ACCESS
Demo, User, User
Demo, Admin, Admin];

Section Application;
```

TIP: IT IS **REQUIRED** TO INCLUDE THE **Section Application** STATEMENT AFTER THE ACCESS LOAD IN ORDER TO PRODUCE A USABLE DOCUMENT.

This simple access check will require the user to identify himself/herself when opening a document.

The USERID *Demo* and PASSWORD *User* will prevent the user from accessing the load script if that security is set in Document Properties. A user belonging to the USER group is also denied access to the **Security** tab in the **Document Properties** and **Sheet Properties** dialogs. Also note that using QlikView Analyzer+ will automatically open in USER mode, regardless of the section access settings.

The USERID *Demo* together with the PASSWORD *Admin* will give the user the rights to make all changes to the document in QlikView Enterprise.

***TIP*: IT IS STRONGLY SUGGESTED TO SAVE A DOCUMENT WITH SECTION ACCESS AS A NEW FILE NAME *AFTER* RELOAD, BUT *BEFORE* ATTEMPTING TO CLOSE AND REOPEN THE QlikView DOCUMENT. IF LOGICAL ERRORS EXIST IN SECTION ACCESS, IT MAY NO LONGER BE POSSIBLE TO OPEN THE DOCUMENT ONCE IT IS RELOADED.**

3. Click **OK**.

4. Save your file, but ***DO NOT RELOAD YET***.

5. **Save As** a new filename (add "_secure" to your existing filename) and then run the script.

6. Exit QlikView and open your newly named document again. You will now see the following dialog box where you can enter your user ID and password.

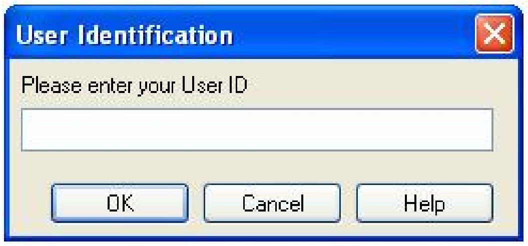


Figure 31. Dialog in which user ID is entered

23.6. Access control for certain commands

The settings under the **Security** tabs in the **Document Properties** dialog and the **Sheet Properties** dialog in the **Settings** menu allow the prevention of users from making certain menu commands and changing the layout. In order to use these settings as security measures, it is important, however, that the users have only USER rights. All those who have ADMIN rights can change the security settings at any time.

23.7. Further access control

It is easy to increase the security control to those users who we believe will require access to a specific document. By adding a field to the previously created **inline** table, we can connect it to a new two-columned table in which we specify the serial numbers that have access to the document. In this way, we have restricted access to a certain document even further.

The new two-columned table will be created in Notepad, or a similar text editor, and be called *Access*. We will save this new tab-delimited text file in the **DataSources** directory.

We proceed in the following way:

1. Add another field called *COMPUTER* to the previously existing *Access01* **inline** table. Include one or two identifiers for course computers (these will act as connected fields) as shown below:

```
/* SECURITY section access comments */

Section Access;

Access01:
Load * Inline
[USERID, PASSWORD, ACCESS, COMPUTER
Demo, User, User, Course1
Demo, Admin, Admin, Course2];

Section Application;
```

2. Open Notepad, or a similar text editor, and create a two-columned table with the fields *COMPUTER* and *SERIAL*. Include your own serial number as a field value; you will find this number under the **About QlikView** menu in QlikView. An example is shown below:

```
COMPUTER  SERIAL
Course1   2300 2394 7111 8000
Course2   2300 2394 7111 8001
```

3. Save this file as **Access02.txt**.

   It can be seen that only two licence numbers have access to the document we have created, one with USER rights and one with ADMIN rights. Also, note that we are adding restrictions to existing *USERID*s (Demo), and not replacing current restrictions.

4. Open the load script and click on the **Table Files** button. Load the newly created table **Access02.txt** after the **inline** table, and prior to

the **section application** statement, as shown below:

```
Access02:
Load
     COMPUTER,
     SERIAL
FROM Datasources\Access02.txt (ansi, txt,
delimiter is '\t', embedded labels);

Section Application;
```

5. Click **OK**.

6. Save your file and then run the script. If this works, save your file again (you may want to **Save As** again here) and **Exit** QlikView.

7. Open the program again and enter your USERID and PASSWORD.

Assuming that you have not entered your serial number in both records in the **Access02.txt** file, you may only log in as a USER or ADMIN. It would be easy to add a third line to the access control tables which always gave every serial number USER rights, assuming a valid user ID and password. To do this, we can add a third computer (e.g. Course3) and enter ***** as the value in the *SERIAL* field.

Unattended Command Line Reload Considerations

In order to create security access for a non-intervention command line reload process, you would enter the *SERIAL* registered to the user assigned to the reload process on the reload computer. Then connect this to *USERID* and *PASSWORD* with ***** as values (* here means all possible values, which would avoid the *USERID* and *PASSWORD* prompts). You should also set the *ACCESS* field to Admin for this user.

If you add the following record to the *Access01* **Load Inline** statement, this will allow the Course2 computer to be used for unattended reloads.

```
*,*,Admin,Course2]
```

23.8. Access restrictions on selected field values

QlikView secured access provides a feature to prevent users from viewing parts of the data in a document. This feature was primarily developed for QlikView Server, but can also be used in QlikView, with a few considerations.

The selection of values to be shown or hidden is controlled by having one or more fields with the same names in **section access** and **section application**. When the user has logged in, QlikView will copy the (upper case) selected values in **section access** to fields in **section application** with the same name. QlikView will permanently conceal, from the user, all the data excluded by this process.

All field names and values used to connect **section access** and **section application** must be written in upper case, since all field names and field values are, by default, converted to upper case in **section access**.

To utilize this feature, the option **Initial Data Reduction Based on Section Access** on the **Opening** page of the **Document Properties** dialog must be checked. If this feature is used in documents that are to be distributed by other means than via QlikView Server, the option **Prohibit Binary Load**, on the same page, must be selected in order to maintain the integrity of data protection.

***TIP**:* There is a known issue in Dynamic Data Reduction QlikView documents where a fully executed reload will override dynamic data reduction settings – allowing users to see "all" data. In order to mitigate this security risk, QlikView version 7 will prohibit data reloads in documents that have dynamic data reduction in effect.

Script Control Statements

Now we would like to add a different method for determining document security access using the *SalesPerson* field, along with data reduction. We will utilize the concept of script control statements to allow the administrator of this document to choose which security access method to use for the data reload. To do this, we will set a variable to specify the method, and then reference that variable in an **IF … THEN … ELSE** script control. Unlike the **If** function, typically used in a **Load** statement or a chart expression, the **IF … THEN … ELSE** script control allows one or more script statement to be included within the control parameters.

The syntax for the **IF … THEN … ELSE** control is:
if *condition* **then**
[statements]
{ **elseif** *condition* **then**
[statements] }
[**else**
[statements]]
end if

where
condition is a logical expression which can be evaluated as true or false.
statements is any group of one or more QlikView script statements.

***TIP**:* Since the **if..then** statement is a control statement, each of its four possible clauses (**if..then**, **elseif..then**, **else** and **end if**) must not cross a line boundary. They may be terminated by semicolon or end-of-line.

1. Open the load script and **Edit Hidden Script** (note that you must have Admin privileges to edit the hidden script). Verify that your cursor is positioned on the Hidden Script tab at the top.

2. First, we will add the script control variable, and the script control statements. Your Hidden Script tab should now look as follows (the new statements have been highlighted):

```
/* SECURITY section access comments */

/* Use for BASIC security access */
SET vSecureType = 'BASIC';
/* Comment following statement for BASIC security
access */

/* Use for SalesPerson and Data Reduction */
//SET vSecureType = 'DATA';

Section Access;

IF vSecureType = 'BASIC' THEN

Access01:
Load * Inline
[USERID, PASSWORD, ACCESS, COMPUTER
Demo, User, User, Course1
Demo, Admin, Admin, Course2
*,*,Admin,Course2];

Access02:
Load
   COMPUTER,
   SERIAL
FROM Datasources\Access02.txt (ansi, txt, delimiter
is '\t', embedded labels);

Section Application;

ELSE  /* 'DATA' Security Access */


END IF
```

We have not added any statements in the ‘DATA’ security access control part yet, so we need to make sure the statement that would set the **vSecureType = 'DATA'** is commented.

3. Click **OK**.

4. Save your file and then run the script. If this works, save your file again (you may want to **Save As** again here) and **Exit** QlikView.

5. Open the program again and enter your USERID and PASSWORD.

Nothing should have changed yet concerning access to your document.

Now, however, we will implement the second method using the *SalesPerson* field and data reduction.

Field value limitation in section access

We will now attempt to limit the amount of data shown in our QlikView document. We want to distribute the file to the employees involved in sales. Each salesperson will, however, not have access to data pertaining to their peers. Therefore, we will add a limitation to the script, which ensures that each person only has access to their own data.

We will utilize two text files for this purpose. The first file will establish the section access fields for each allowed user. The second text file will be loaded in section application, and limit the application data that each allowed user will be able to view once they open the QlikView document. Both text files are located in the **Datasources** directory.

We must also bear in mind that we need an administrator to manage the document. One of the salespersons is also the Sales Manager of the company, and he should naturally have access to the entire document in order to be able to assess the performance of each salesperson. We also have a Sales Coordinator in Lund who should have access to the data on all salespersons. This access can be implemented through use of a null field value specified in the connecting field when loaded in the **section access** section. You could use the * value for this field, as we used earlier for the USERID and PASSWORD fields, but it is generally preferable to use null for connecting fields, since this will allow access to ALL data, regardless of whether it has a logical connection to the connecting field.

1. Open the load script and **Edit Hidden Script** (note that you must have Admin privileges to edit the hidden script). Verify that your cursor is positioned on the Hidden Script tab.

2. Now, we will add the statements in the 'DATA' security access part. Here, we will add **Load** statements for the two text files, separated by the **Section Application** statement.

3. The first text file to load is **SalesSecurity.txt**, located in the **Datasources** directory. Label this logical table as *Access01*.

4. Then, add the **Section Application** statement.

5. Next, create a load statement for the **SalesInitials.txt** file, located in the **Datasources** directory. It is good practice to use the **upper** function against the connecting field (*SP*), since the value must be uppercase to match the value from section access. The new statements should look as follows:

```
ELSE  /* 'DATA' Security Access */

Access01:
Load
   [USERID],
   [ACCESS],
   SP /* Connecting field for data reduction */
FROM Datasources\SalesSecurity.txt (ansi, txt,
delimiter is '\t', embedded labels);

Section Application;

Access_Application:
Load
   upper(SP) as SP,  /* Connecting field for data
reduction */
   [SalesPerson]
FROM Datasources\SalesInitials.txt (ansi, txt,
delimiter is '\t', embedded labels);

END IF
```

6. Click **OK**.

7. Save your file and then run the script. If this works, save your file again (you may want to **Save As** again here) and **Exit** QlikView.

8. Open the program again and enter **James** as USERID.

You should see all data. Then, close QlikView, and log in as **Leif**. Notice that you can view data for this user, as well as the Sales Person Tom Lindwall.

24. Exercises

In this exercise, we will make sure that the ability to perform an unattended command line reload is possible, regardless of the security access method used in the script.

1. Make the script changes necessary to allow the SERIAL number registered on computer Course2 to open, reload and save this QlikView document, regardless of whether the ‘BASIC’ or ‘DATA’ method is used for security access.

25. Debugging

When making script changes, it can sometimes be difficult to find errors. QlikView therefore contains a script execution debugger to help you identify mistakes in your script.

Running the script in the debugger makes it much easier to find errors. It can also save a great deal of time. In the debugger, you can study each statement and check the values of the variables while the script is being executed.

The script is shown in the window in the upper half of the dialog. A yellow cursor shows how far execution has proceeded. **Breakpoints** can be inserted by clicking on a line number, and removed by clicking again. All breakpoints can be removed by clicking the **Clear** button. When a new breakpoint is encountered, execution is halted until the command is given to resume.

The current script statement is shown in the window in the middle of the dialog.

Status codes and error messages are shown in the lower left window. This is essentially the same information as that shown in the **Script Execution Progress** window when the script is run without the debugger.

The bottom right-hand window shows all the variables and their values. Values which have been changed, are shown in red.

The script can be run in three different modes:

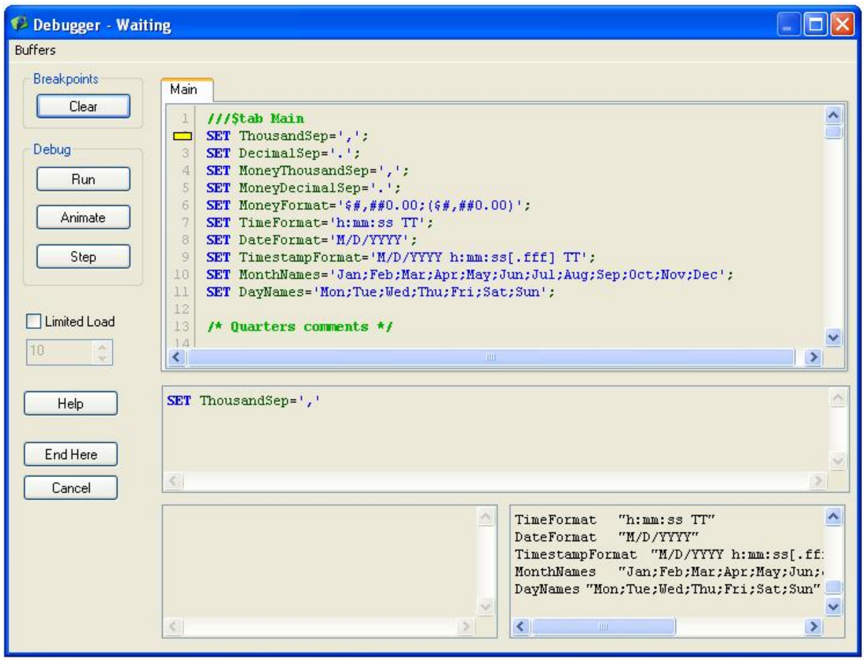
Run	This is the normal mode for script execution. The script is run to the end or until a breakpoint is encountered.
Animate	The script is run as described above, but with a short pause after each statement. This allows you to follow the execution more carefully.
Step	The script is executed one statement at a time.

To run the whole script, use one of the following methods:

Select **Limited Load** and enter a number in the window below. The number is the maximum number of records accepted for each **load** and **select** statement. This is a very practical way to limit the execution time when a script is being run on live data.

Click **End Here** to leave the debugger. Data that has all ready been loaded will be retained in QlikView.

Click **Cancel** to stop execution and to discard the loaded data.



```
///$tab Main
SET ThousandSep=',';
SET DecimalSep='.';
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='$#,##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;Oct;Nov;Dec';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';

/* Quarters comments */
```

```
SET ThousandSep=','
```

```
TimeFormat   "h:mm:ss TT"
DateFormat   "M/D/YYYY"
TimestampFormat  "M/D/YYYY h:mm:ss[.ff
MonthNames   "Jan;Feb;Mar;Apr;May;Jun;
DayNames "Mon;Tue;Wed;Thu;Fri;Sat;Sun"
```

Figure 32. The Debugger dialog

We will now try running the debugger on our script.

1. Open the **Edit Script** dialog from the menu or toolbar.

2. Click the **Debug** toolbar icon to open the **Debugger** dialog.

3. Insert breakpoints before **SQL Select** in tables *Products*, *Categories* and before **Load** in the *Sales_Person* table, by clicking on the line number in the script window.

   The breakpoints will be seen as red points.

4. See what happens when you click on the **Animate** button.

   Running the script in the various modes available in the debugger is like clicking Run in the script. When the script has been loaded, you must click **Close** to get to the **Select Fields** dialog.

5. Open the debugger again and **Run** the script with a **Limited Load** of 10 records. Not only can this be a useful tool for identifying errors and validating changes, but also as a way to create template applications with a small number of records in them.

25.1. The Script Execution Log File

At the beginning of the course, we set the **Generate Logfile** selection in **Document Properties**. Now we will take a look at the file that is generated during script execution.

The log file will have the same name as your QlikView document, but with a ".qvw "appended to it, and a file extension of .log (e.g. **QVE_Course.qvw.log**). The file will be located in the same directory as the QlikView document that is being reloaded.

The log file will generally contain all executed script statements, without the line or bracketed comments (REM statements are shown). It also includes the following information:

- Start execution timestamp
- Finished execution timestamp
- The number of fields and name of each field identified in a **load** or **select**, along with the number of records included in that **load** or **select**.
- The line number from the script
- The QlikView version running the script
- Any script execution errors that may have occurred
- Any Synthetic Keys that are created will be listed at the end of the log file.

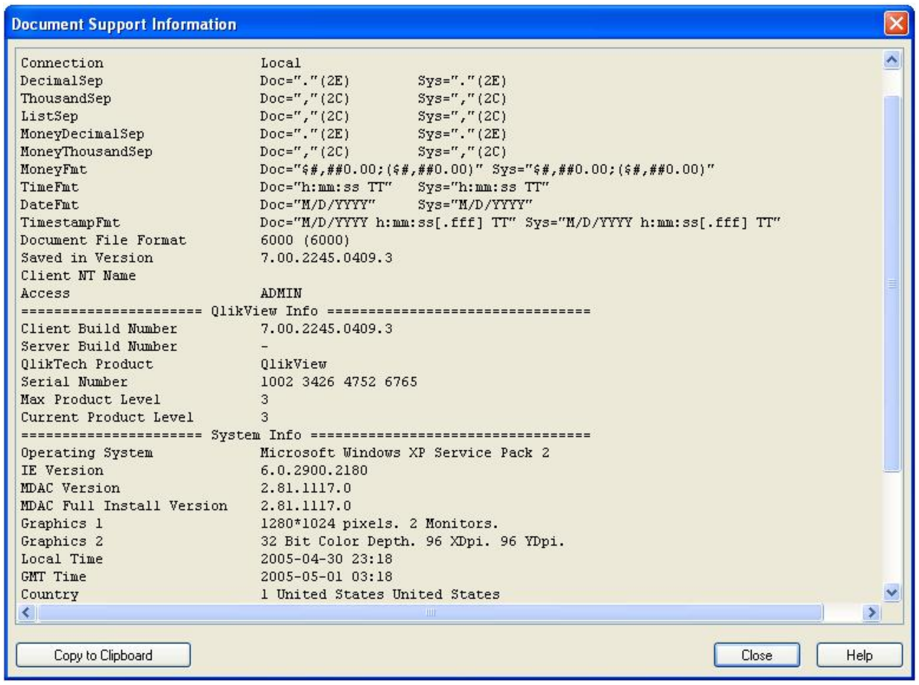
See if you can locate the log file from your application, and open it using Notepad, or a similar tool.

25.2. Reporting Bugs in QlikView

If you discover a bug in QlikView, it is important to report this behaviour to QlikTech or a certified QlikTech Partner. It is also critical to provide as much detailed information as possible to describe the error, and, hopefully, the ability for QlikTech developers to reproduce the behaviour. There are a number of things you can do to make the bug report an efficient and useful process.

1. Provide a clear description of the problem, including typical actions leading up to the error, along with any error messages that are displayed.

2. If it is possible to provide a sample QlikView document that can be used to demonstrate the error, this will be very helpful in resolving the issue.
3. If the error occurs during a reload, include the QlikView log file with the bug report. The log file is described in the preceding section of this course.
4. You should always create a *System* sheet in your QlikView documents to help determine the integrity of your data structure. This sheet can easily be "hidden" from users in a production application through the use of the **Show Sheet** setting in **Sheet Properties**.
5. All bug reports should include the **Document Support Information**. This data contains valuable information about the computer experiencing the problem, the version of QlikView being run, and settings in the QlikView document. It is available from the **Help** menu item, or by pressing **Ctrl-Shift-Q**. Use the **Copy to Clipboard** button to copy this information and paste in your Email or into a text document.



```
Connection                  Local
DecimalSep                  Doc="."(2E)        Sys="."(2E)
ThousandSep                 Doc=","(2C)        Sys=","(2C)
ListSep                     Doc=","(2C)        Sys=","(2C)
MoneyDecimalSep             Doc="."(2E)        Sys="."(2E)
MoneyThousandSep            Doc=","(2C)        Sys=","(2C)
MoneyFmt                    Doc="$#,##0.00;($#,##0.00)" Sys="$#,##0.00;($#,##0.00)"
TimeFmt                     Doc="h:mm:ss TT"   Sys="h:mm:ss TT"
DateFmt                     Doc="M/D/YYYY"     Sys="M/D/YYYY"
TimestampFmt                Doc="M/D/YYYY h:mm:ss[.fff] TT" Sys="M/D/YYYY h:mm:ss[.fff] TT"
Document File Format        6000 (6000)
Saved in Version            7.00.2245.0409.3
Client NT Name
Access                      ADMIN
====================== QlikView Info ================================
Client Build Number         7.00.2245.0409.3
Server Build Number         -
QlikTech Product            QlikView
Serial Number               1002 3426 4752 6765
Max Product Level           3
Current Product Level       3
====================== System Info ==================================
Operating System            Microsoft Windows XP Service Pack 2
IE Version                  6.0.2900.2180
MDAC Version                2.81.1117.0
MDAC Full Install Version   2.81.1117.0
Graphics 1                  1280*1024 pixels. 2 Monitors.
Graphics 2                  32 Bit Color Depth. 96 XDpi. 96 YDpi.
Local Time                  2005-04-30 23:18
GMT Time                    2005-05-01 03:18
Country                     1 United States United States
```

Figure 33. Document Support Information

25.3. QlikView Reference Materials

There are a number of QlikView Reference materials available to you. Many have been mentioned in this course already. It is important to familiarize yourself with these tools as you develop your skills to create efficient and effective QlikView documents.

1. **QlikView Reference Manual**: The Reference Manual documents are typically installed on your computer during the QlikView installation process. These include a Tutorial Guide for new users, along with detailed reference guides for script and layout development. You can also contact your QlikTech or certified QlikView Partner representative about ordering hard copy manuals.
2. **QlikView Help Subsystem**: The Help subsystem is available directly from most dialogs and menus. It includes a general contents, an index of available information, and a full text search capability. If invoked off a dialog, it will open context specific information.
3. **QlikView Example Documents**: These sample documents can be an invaluable source of information that demonstrate validated methods for developing QlikView documents. It can be quite useful to browse through these documents to get ideas of what can be accomplished through QlikView.
4. **QlikView Training Material:** This course document, along with other course documents offered by QlikTech can be used as a valuable reference guide that you can always refer back to after training is completed.

26. Appendix 1: Data Types in QlikView

QlikView can handle text strings, numbers, dates, times, timestamps and currencies. These can be sorted, shown in various formats and used in calculations. This means that dates, times and timestamps can be added and subtracted. This chapter is provided as background and reference information concerning how QlikView handles data types.

26.1. Data storage in QlikView

In order to understand how QlikView interprets data and formats numbers, you must first know how data is stored internally in the program. All the data loaded into QlikView is stored in two ways: as a text string and as numbers.

1. The text string is always used. It is this that is shown in the list boxes and other sheet objects. In the formatting of data in list boxes (number formatting) only the text string is affected.

2. Numbers are only used when the data can be interpreted as a valid number. All forms of numerical calculation and sorting can be used.

If several pieces of data with the same numerical value are loaded in the same field, they will be treated as multiple occurrences of the same value and will together be assigned the first text string encountered. If the numbers 1.0, 1 and 1.000 are loaded in this order, they will be given the numerical value 1, and the original text string 1.0.

26.2. Data containing information on data type

Fields that contain numbers of a defined data type, in a database loaded via ODBC, will be handled according to their respective data type in QlikView.

QlikView will remember the original format of the fields, even if it is changed in one of the number format dialogs. It is always possible to restore the original format by clicking on the **Default from input** button in the number format dialog.

QlikView uses the following standard formats for each type of number:

- **Integers, floating-point numbers**: standard format for numbers
- **Money**: standard format for currencies
- **Time, date, timestamp**: ISO standard

The standard settings for numbers and currencies are defined via variables, which are interpreted in the script, or via the settings of the operating system (Control Panel).

26.3. Data without information on data type

The handling of data that have no specific formatting information (e.g. data from text files or ODBC data with a general format) is more complicated. The final result depends on at least six factors.

1. The format of the data in the database
2. The settings of the operating system regarding numbers, time, date, etc. (Control Panel)
3. The use of variables for interpretation in the script
4. The use of interpretation functions in the script
5. The use of formatting functions in the script
6. The use of number format dialogs in the document

QlikView tries to interpret the input data as numbers, dates, times, etc. As long as the standard system settings are used, QlikView will interpret and format the data automatically. Thus the user need not change the script or the settings in QlikView. There is a simple way of checking if QlikView has interpreted the data correctly: numerical data are usually right-aligned in list boxes, while text strings are left-aligned.

The standard routine involves going through the following process until a suitable format is found. (Standard format includes, for example, decimal separator, the order of years, months and numbers, etc. in the operating system, i.e. in the Control Panel, or in some cases defined via the special variables for interpretation by the script.)

QlikView interprets data as:

1. a number according to the standard format for numbers
2. a date according to the standard format for dates
3. a timestamp according to the standard format for time and date
4. a time according to the standard format for time
5. a date according to the following format: yyyy-MM-dd

6. a timestamp 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. currencies according to the standard format for currencies

9. a number with '.' as decimal separator and ',' as a separator for thousands, assuming that neither the decimal separator nor the separator for thousands is set to ','

10. a number with ',' as decimal separator and '.' as a separator for thousands, assuming that neither the decimal separator nor the separator for thousands is set to '.'

11. a text string. This final test never fails: if it is at all possible to load the data, it is always possible to interpret it as a text string.

Interpretation problems may arise when data is loaded from text files. An incorrect decimal separator or the separator for thousands can lead to QlikView interpreting the numbers incorrectly. The first thing you should do is to check that the variables for number interpretation in the script are correctly defined, and that the system settings in the Control Panel are correct.

When QlikView has interpreted data as a date or time, it is possible to change the date and time format in the **Properties** dialog of the sheet object (under the **Number** tab). The overruling formatting is done on the document level in the **Document Properties: Number sheet**.

As there is no predefined format for data, a record can obviously contain values with different formats in a single field. For example, valid dates, integers and text may be found in one field. This data will not be formatted, but will be shown in list boxes in their original form.

When the number format dialog for such a field is opened for the first time, the format will be **Mixed**. Once you have changed the format it will be impossible for QlikView to revert to the original format for these field values, that is, if the **Survive Reload** box is checked when the script is run.

The **Default from input** button is thus not activated for this kind of field after the number format has been changed.

26.4. Dates and times

QlikView stores dates, times and timestamps as a date serial number for date. The serial number for dates is used for dates, times and timestamps and for arithmetical calculations based on units of date and time. Dates and times can thus be added and subtracted, and intervals can be compared, etc.

The numbers used for date and time are the values of the number of days that have passed since December 30, 1899. QlikView is thus compatible with the date system 1900 used by Microsoft Excel for Windows, Lotus 1-2-3 and other programs, between 1 March 1900 and 28 February 2100. Outside this time frame QlikView uses the same date system extrapolated with the aid of the Gregorian calendar, which is now a standard in the Western world.

The date serial number for time is a number between 0 and 1. The date serial number 0.00000 corresponds to 00:00:00, while 0.99999 corresponds to 23:59:59. Mixed numbers show both date and time: the date serial number 2.5 corresponds to 1 January 1900, 12.00 hours.

Data is displayed according to the format of the text string. The settings made in the Control Panel are used as standard. It is also possible to define the format for date and time via variables that are interpreted by the script or with the aid of a formatting function. Furthermore, it is also possible to reformat data in the **Properties** dialog of the sheet objects.

Example:

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

or vice versa

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

As mentioned previously, QlikView will follow a certain procedure for the interpretation of dates, times and other types of data. The final result will, however, be affected by a number of factors.

27. Appendix 2: The Final Script

```
///$tab HiddenScript
/* SECURITY section access comments */

/* Use for BASIC security access */
SET vSecureType = 'BASIC';
/* Comment following statement for BASIC security access */

/* Use for SalesPerson and Data Reduction */
//SET vSecureType = 'DATA';

Section Access;

IF vSecureType = 'BASIC' THEN

Access01:
Load * Inline
[USERID, PASSWORD, ACCESS, COMPUTER
Demo, User, User, Course1
Demo, Admin, Admin, Course2
*,*,Admin,Server,Course2];

/*
   UnComment following statement AFTER adding correct
   serial number in Access02.txt file
*/

REM Access02:
Load
      COMPUTER,
      SERIAL
FROM Datasources\Access02.txt (ansi, txt, delimiter is '\t',
embedded labels);

Section Application;

ELSE  /* 'DATA' Security Access */

Access01:
Load
      [USERID],
      '*' as SERIAL, /* Needed for command line user */
      [ACCESS],
      SP /* Connecting field for data reduction */
FROM Datasources\SalesSecurity.txt (ansi, txt, delimiter is
'\t', embedded labels);

/* Concatenate data for command line reload */

concatenate (Access01) Load * Inline
[USERID, SERIAL, ACCESS, SP
*,1002 3426 4752 6766,Admin,];

Section Application;

Access_Application:
Load
     upper(SP) as SP,  /* Connecting field for data
reduction */
     [Sales Person]
FROM Datasources\SalesInitials.txt (ansi, txt, delimiter is
'\t', embedded labels);

END IF /* Security Access */
///$tab Main
LET vCurrentFullYear = Year(today()) - 1;

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;Oct;Nov;Dec';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';

/* Quarters comments */

rem Quarters_Map:
Mapping Load * Inline [
Month, Quarter
1,Q1
2,Q1
3,Q1
4,Q2
5,Q2
6,Q2
7,Q3
8,Q3
9,Q3
10,Q4
11,Q4
12,Q4];

Quarters_Map:
Mapping Load
     rowno() as Month,
     'Q' & Ceil(rowno()/3) as Quarter
Autogenerate(12);

/* CONNECT to QWT database */

ODBC CONNECT TO [EnterpriseScript;DBQ=DATASOURCES\QWT.mdb];
```

```
/* Customers Table load using QVD buffer */

Customers:
buffer (stale after 7 days) SELECT
    Address,
    City,
    CompanyName,
    ContactName,
    Country,
    CustomerID,
    Fax,
    Phone,
    PostalCode,
    StateProvince
FROM Customers;

rem store Customers into datasources/customers.qvd;

rem Customers:
Load
      Address,
      City,
      CompanyName,
      ContactName,
      Country,
      CustomerID,
      Fax,
      Phone,
      PostalCode,
      StateProvince
FROM Datasources\customers.qvd (qvd);


/* Shippers Table comments */

Shippers:
SQL SELECT
      ShipperID,
      CompanyName as Shipper
FROM Shippers;

/* Products Table comments */

Products:
SQL SELECT
    CategoryID,
    ProductID,
    ProductName,
    QuantityPerUnit,
    SupplierID,
    UnitCost,
    UnitsInStock,
    UnitsOnOrder
FROM Products;
```

```
/* Categories Table comments */

Categories:
SQL SELECT
    CategoryID,
    CategoryName,
    Description
 FROM Categories;

/* Orders Table comments */

Orders:
buffer (stale after 1 days) Load
      CustomerID,
      EmployeeID,
      EmployeeID as EmployeeSales,
      Freight,
      OrderDate,
      OrderDate as CountOrderDate,
      Year(OrderDate) as Year,
      Month(OrderDate) as Month,
      date(monthstart(OrderDate),'MMM-YYYY') as [Rolling
Month],
      applymap('Quarters_Map',num(Month(OrderDate))) as
Quarter,
      Day(OrderDate) as Day,
      OrderID,
      ShipperID;

SELECT * FROM Orders;

/* Order Details Table comments */

Order_Details:
buffer (stale after 1 days) SQL SELECT
      OrderID,
      ProductID,
      ProductID as CountProductID,
      Quantity,
      UnitPrice,
      UnitPrice*Quantity*(1-Discount) as NetSales
FROM `Order Details`;

Employee:
Load
    EmpID as EmployeeID,
     [First Name] & ' ' & [Last Name] as Name,
     [Last Name],
     [First Name],
     Title,
     [Hire Date],
     Year([Hire Date]) as [Employee Hire Year],
     Office,
     Extension,
      [Reports To],
      [Year Salary]
FROM Datasources\Emp*.xls (biff, embedded labels, table is
[Employee$]);

REM Employee:
concatenate (Employee) Load
    EmpID as EmployeeID,
      [First Name] & ' ' & [Last Name] as Name,
      [Last Name],
      [First Name],
      Title,
      [Hire Date],
      Year([Hire Date]) as [Employee Hire Year],
      Office,
      Extension,
      [Reports To],
      [Year Salary]
FROM Datasources\Employees_New.xls (biff, embedded labels,
table is [Employee$]);


Office:
Load
    Office,
      Address as OfficeAddress,
      [Postal Code] as OfficePostalCode,
      City as [Sales Office],
      StateProvince as OfficeStateProvince,
      Phone as OfficePhone,
      Fax as OfficeFax,
      Country as OfficeCountry
FROM Datasources\EmpOff.xls (biff, embedded labels, table is
[Office$]);

/* Suppliers Table comments */

Qualify *;
Unqualify SupplierID;

suppliers:
Load
      SupplierID,
      CompanyName,
      ContactName,
      Address,
      City,
      PostalCode,
      Country,
      Phone,
      Fax
FROM Datasources\suppliers.dif (oem, dif, embedded labels);

Unqualify *;
```

```
/* Include external script file */

$(Include=datasources\email.txt)
///$tab Sales Person
/* Sales Person Table comments */

Sales_Person:
Load
    EmployeeID,
    EmployeeID as SalesPersonID,
    Name as [SalesPerson],
    Title as SalesTitle
Resident Employee
Where
    Exists(EmployeeSales,EmployeeID);
///$tab Sales Change
/*
   Load and Join Order data based on
   vCurrentFullYear variable setting
*/

OrdersByYear_Source:
Load
     CustomerID,
     OrderID,
     Year
RESIDENT
     Orders
Where
     Year = $(vCurrentFullYear) - 1 or Year =
$(vCurrentFullYear); /*only load 2 years*/

Left Join (OrdersByYear_Source) Load
     OrderID,
     NetSales
RESIDENT
     Order_Details;

/* Aggregation by Customer comments */

OrdersByYear:
Load
     CustomerID,
     Year,
     sum(NetSales) as NetSalesByYear
RESIDENT
     OrdersByYear_Source
Group by
     CustomerID,Year;

DROP TABLE OrdersByYear_Source;

/* Sales Change using Previous function */

SalesChange:
```

```
Load
      CustomerID,
      NetSalesByYear,
      If(CustomerID = Previous(CustomerID),
         If(NetSalesByYear > Previous(NetSalesByYear),
       'UP','DOWN'),null()) as SalesChangeIndicator,
      If(CustomerID = Previous(CustomerID),
         NetSalesByYear - Previous(NetSalesByYear),
       null()) as SalesChangeAmt
RESIDENT
      OrdersByYear
Order by
      CustomerID,Year;

DROP TABLE OrdersByYear;

DROP FIELD NetSalesByYear;

///$tab Budget/Result
Budget_Actual_S1:
CROSSTABLE(BudgetYear, Amount, 2)
Load
      *
FROM
      Datasources\Budget.xls (biff, header is line, embedded
labels, table is [Sheet2$], filters(
Replace(1, top, StrCnd(null))
));

Budget_Actual:
Generic Load
      Office,
      BudgetYear,
      Metric,
      Amount
Resident
      Budget_Actual_S1;

DROP TABLE Budget_Actual_S1;
```