

P r o f e s s i o n a l   E x p e r t i s e   D i s t i l l e d

# QlikView Scripting

Your comprehensive guide to scripting powerful  
QlikView applications

**Matt Floyd**

**[PACKT]** enterprise   
PUBLISHING professional expertise distilled

# QlikView Scripting

Your comprehensive guide to scripting powerful  
QlikView applications

**Matt Floyd**



BIRMINGHAM - MUMBAI

# QlikView Scripting

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: November 2013

Production Reference: 1181113

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78217-166-9

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Dan Daube ([dandaube@gmail.com](mailto:dandaube@gmail.com))

# Credits

**Author**

Matt Floyd

**Project Coordinator**

Amey Sawant

**Reviewers**

Ralf Becher

Steve Dark

Anthony Perozzo

Deepak Vadithala

**Copy Editors**

Alisha Aranha

Sarang Chari

Dipti Kapadia

Sayanee Mukherjee

Karuna Narayanan

Lavina Pereira

Laxmi Subramanian

**Acquisition Editors**

Edward Gordan

Antony Lowe

**Proofreader**

Christopher Smith

**Commissioning Editors**

Nikhil Chinnari

Poonam Jain

**Indexer**

Monica Ajmera Mehta

**Technical Editors**

Manan Badani

Krutika Parab

Hardik B. Soni

**Production Coordinator**

Alwin Roy

**Cover Work**

Alwin Roy

# About the Author

**Matt Floyd** has worked in the software industry since 2000, and has held career titles ranging from Project Manager to Technical Writer to Business Intelligence analyst. His career has spanned many industries, including environmental, healthcare, pharmaceutical, and insurance.

Matt's hands-on experience with QlikView started in 2008 with documenting an OEM-healthcare-related QlikView application for several years. Since then, he has been a technical writer, implementation engineer, consultant, and developer on QlikView projects. His passion for QlikView stems from his fascination of discovery through data, and the art, science, and power of data visualization. He is the founder of the Atlanta QlikView User Group and lives with his family in metropolitan Atlanta. Matt can be followed on his blog where he covers mostly QlikView-related Business Intelligence topics at [floyd matt .com](http://floyd matt .com).

---

Thank you and much love to my wife, Beth, and daughters Audrey and Hope for their understanding and support during the writing process. And thank you to Dan Daube for his creative and technical work on the cover photo, and to my family and friends, who encouraged and cheered me on.

Much gratitude goes out as well to the entire Packt Publishing team, who have been so professional and helpful; thank you all for your kindness, suggestions, and patience. I'm honored to have super QlikView leaders as expert reviewers on this book: Ralf Becher, Steve Dark, Anthony Perozzo, and Deepak Vadithala. Your reviews were superb and sharpened my ideas and clarified the text to a high degree. Thank you to you all and to all the QlikView experts and my wonderful colleagues who are always ready to share insightful business intelligence tips and tricks. I hope this book adds a little bit to the ongoing QlikView discussion. Thank you so much for reading.

---

# About the Reviewers

**Ralf Becher** has worked as an IT systems architect and IT consultant since 1989 in the areas of banking, insurance, logistics, automotive, and retail. He and his partners founded TIQ Solutions in 2004.

The Leipzig-based company specializes in modern, quality-assured data management. Since 2004, it has been helping its customers process, evaluate, and maintain the quality of company data, helping them introduce, implement, and improve complex solutions in the fields of data architecture, data integration, data migration, master data management, metadata management, data warehousing, and business intelligence.

Ralf is an internationally recognized QlikView expert with a strong position in the QlikCommunity. He started working with QlikView in 2006 and has worked on QlikView add-on solutions for data quality and data integration, especially for connectivity in the Java and Big Data realms. He runs his QlikView data integration blog at <http://tiqview.tumblr.com/>.

**Steve Dark** was an SQL Server / MS ASP developer who built web-based reporting solutions for 10 years until he was shown a demo of QlikView. Soon after this revelation, Steve left his previous employer to set up Quick Intelligence, a consultancy focusing entirely on QlikView and delivering business intelligence solutions. Preferring to stay at the coalface, Steve spends the majority of his time with clients, building QlikView applications, managing QlikView deployments, and running projects.

He will never tire of showing QlikView to new users and seeing that "jaw-drop moment".

Steve stays active on QlikCommunity and other social media sites by sharing his enthusiasm for QlikView and assisting other users. Through his blog he shares tutorials, examples, and insights on QlikView (read them at <http://www.quickintelligence.co.uk/qlikview-blog/>).

Steve has also been a technical reviewer on *QlikView 11 for Developers*, by Barry Harmsen and Miguel Garcia, and *QlikView 11 for Developers Cookbook*, by Stephen Redmond. Both of these titles were published by Packt Publishing.

---

Getting your scripting right is always three quarters of the effort required on producing a top-class QlikView solution. I would like to thank Matt for writing this book so that this side of QlikView development also has the reference it deserves.

---

**Anthony Perozzo** is a software developer residing in the south of Belgium. He has been interested in IT sciences from a young age and has been working with many programming languages such as Java, .NET, PHP, HTML/JS, and Objective-C.

Anthony has worked in various areas. He worked for more than eight years in the domain of healthcare IT as a Clinical Software Director (mainly for infectious diseases, such as HIV, tuberculosis, and hepatitis) and has travelled to many developing countries for his work. During his missions, he made various web applications for the medical community and collaborated with NGOs, medical specialists, and well-known organizations on Business Intelligence and reporting tools (both in-house and QlikView).

Currently he is facing completely new challenges as he is working on in-house financial solutions (using SAP tools and Java) as a Project Manager with a worldwide leading company.

During his spare time, Anthony likes travelling around the world and making mobile apps for iPhone and iPad (Giltonwe Apps). He also likes flying his various drones.

**Deepak Vadithala** is a BI consultant and database developer who has been building BI/reporting applications since 2005. He has worked through many successful QlikView and SQL Server implementations, right from their inception through implementation; his experience and skills range from application development, UI design, to database/system administration. Deepak has experience working in the investment banking, retail, media, advertising, and research sectors.

He currently holds the QlikView Designer, QlikView Developer, and MCITP (SQL Server) certifications.

Deepak stays active on QlikCommunity and other social media sites by sharing tips and tricks about QlikView. His blog posts contain technical video tutorials, and he conducts technical quizzes that provide an insight on QlikView (read them at <http://www.QlikShare.com/>). You can also follow him on Twitter at @dvadithala where he tweets about technology.



# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read, and search across Packt's entire library of books.

## Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

## Instant Updates on New Packt Books

Get notified! Find out when new books are published by following [@PacktEnterprise](https://twitter.com/PacktEnterprise) on Twitter, or the *Packt Enterprise* Facebook page.

# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Connecting to ODBC/OLE DB Data Sources</b>	<b>5</b>
Installing drivers	5
Configuring ODBC	7
Connecting to the database from QlikView	7
Supported databases	8
Summary	8
<b>Chapter 2: Creating the Script</b>	<b>9</b>
Basics of data modeling	9
Accessing the Script Editor	10
Hidden scripts	11
The Script Editor features and commands	12
The File menu	12
The Edit menu	13
The Insert menu	13
The Tab menu	15
The Tools menu	15
The Help menu	16
The Tools pane	16
Script commenting	17
Creating the Select statement	17
Connecting to the database	18
Building the Select statement	18
Running your script and viewing results	20
Organizing tabs in scripts	20
Summary	21

---

<b>Chapter 3: Data Transformations in Flat Files</b>	<b>23</b>
<b>Why data transformation?</b>	<b>23</b>
<b>Using the Transformation Wizard</b>	<b>24</b>
Removing garbage from data files	24
Using the Fill function	27
Unwrapping data from repeating columns	28
Column manipulation	29
Transformation script results	30
Rotating tables	30
<b>Working with cross tables</b>	<b>32</b>
<b>Working with generic tables</b>	<b>33</b>
<b>Working with hierarchies</b>	<b>35</b>
<b>Summary</b>	<b>38</b>
<b>Chapter 4: Script Features and Functions</b>	<b>39</b>
<b>Number interpretation variables</b>	<b>39</b>
<b>CONNECT, LOAD, and other statement ordering</b>	<b>40</b>
<b>Script segmentation via tabs</b>	<b>41</b>
<b>Renaming fields</b>	<b>42</b>
The AS specifier	42
The Alias statement	43
The Rename Field statement	43
The QUALIFY statements	44
<b>Regular statements and script control statements</b>	<b>47</b>
<b>Script expressions</b>	<b>48</b>
<b>Quotation marks</b>	<b>48</b>
<b>Master calendar placement</b>	<b>50</b>
<b>Summary</b>	<b>50</b>
<b>Chapter 5: Basic Data Model and Table Viewer</b>	<b>51</b>
<b>What is the Table Viewer?</b>	<b>51</b>
<b>Star and Snowflake Schemas</b>	<b>54</b>
<b>Data model best practices</b>	<b>56</b>
<b>Exporting images and structures</b>	<b>56</b>
<b>Viewing internal and source tables</b>	<b>57</b>
<b>Composite/synthetic tables and keys</b>	<b>58</b>
<b>Eliminating synthetic keys and tables</b>	<b>58</b>
<b>Previewing records in tables</b>	<b>59</b>
<b>Information density and subset ratios</b>	<b>60</b>
<b>Summary</b>	<b>61</b>

---

<b>Chapter 6: Advanced Scripting and Data Model Optimization</b>	<b>63</b>
Circular references	63
Fixing circular references and removing synthetic tables	64
Using mapping tables to rename fields	67
Table combining and concatenation	69
The IntervalMatch function	70
Data islands	72
Using metadata	73
Summary	75
<b>Chapter 7: QlikView Data Files</b>	<b>77</b>
Defining a QVD	77
Advantages of QVDs	77
Creating QVDs	78
Full loads of QVDs	79
QVD incremental loads	80
Creating QVDs with QlikView Publisher	83
Loading from QVDs	84
Viewing content of QVDs	84
Managing reload or no data opens of QVWs	85
Summary	85
<b>Chapter 8: Debugging</b>	<b>87</b>
QlikView script debugger	87
Using the Exit Script function	89
Using syntax checker	90
Common QlikView script errors	90
Debugging using logfiles	92
Using find/replace in debugging	94
Removing or partially loading data	94
Summary	95
<b>Chapter 9: Layout Tips for Developers</b>	<b>97</b>
Your KPI story	97
Layout consistency	99
Layout best practices	100
Optimizing your QlikView application user interface	101
Using themes for a consistent look	102
Creating a new QlikView theme	102
Applying themes	103
Containers	104

*Table of Contents*

---

<b>Dimension limits</b>	<b>105</b>
<b>Color alerts and calculated colors</b>	<b>107</b>
Using visual cues	109
<b>Alternate states</b>	<b>110</b>
<b>Set analysis</b>	<b>112</b>
Set analysis syntax	112
Set identifier	112
Set operators	113
Set modifiers	114
<b>Summary</b>	<b>115</b>
<b>Index</b>	<b>117</b>

---

# Preface

QlikView is a powerful business intelligence and data discovery platform that allows people to quickly develop visualization applications with relevant data for business users. The relative ease of QlikView development—including backend scripting—allows applications to be developed rapidly and with more collaboration in application development by business users. QlikView applications allow business users to explore associations in multisource data and use a state-of-the-art dashboard and chart graphics that engage users in an interactive experience.

*QlikView Scripting* offers QlikView developers a rich discussion of scripting topics, from basics to advanced concepts, features, and functions. The book allows developers to quickly gain confidence in understanding and expanding their QlikView scripting knowledge and serves as a springboard for even more advanced topics in QlikView scripting.

## What this book covers

*Chapter 1, Connecting to ODBC/OLE DB Data Sources*, provides basic information on how to connect to various data sources.

*Chapter 2, Creating the Script*, discusses the basics of starting to build a script and corresponding data model, including load statements, selection types, basic script organization, and major elements of scripting.

*Chapter 3, Data Transformations in Flat Files*, describes the methods of transforming data contained in data files such as Excel spreadsheets and text files. There is extensive coverage of the Transformation Wizard, with scenarios on shaping up data files for import and scenarios with generic tables and cross tables.

*Chapter 4, Script Features and Functions*, gives an overview of the basic script syntax and structure of QlikView scripts, along with an overview of the Script Editor.

*Chapter 5, Basic Data Model and Table Viewer*, discusses the theory of associative data modeling and the use of the QlikView Table Viewer.

*Chapter 6, Advanced Scripting and Data Model Optimization*, describes other advanced concepts in QlikView scripting such as link tables, mapping tables, and concatenation.

*Chapter 7, QlikView Data Files*, gives an overview of the creation and use of QlikView data files or QVDs.

*Chapter 8, Debugging*, discusses debugging QlikView scripts, eliminating circular references, and the use of the debugger and logfiles.

*Chapter 9, Layout Tips for Developers*, describes basic QlikView object layout tips and design, as well as a couple of useful advanced object usage settings (alternate states and set analysis).

## What you need for this book

This book can be read independently of any software, but you will get more from the book if you follow along where examples are provided and use some of the code snippets and examples in a test QlikView application. The QVW files provided can only be opened by users with the full QlikView license.

The book was written with QlikView Version 11 in mind, and you can download the free version at QlikTech's website: [www.qlikview.com](http://www.qlikview.com) (go to **Support | Downloads**). Other useful software to have is Excel to view some source data files. In *Chapter 1, Connecting to ODBC/OLE DB Data Sources*, connecting to databases is covered (and instructions and tips are given), but don't worry if you don't have Microsoft Access. Download and install any RDBMS software (MySQL, Oracle, and so on) to practice connecting to a database; but this is not essential for the rest of the book.

## Who this book is for

*QlikView Scripting* is targeted at basic to intermediate developers, with some knowledge of QlikView applications and a basic understanding of QlikView scripting and data extraction and manipulation. Advanced users can also use this book as a reference guide and teaching aid. QlikView project team members such as business users, data/ETL professionals, project managers, and systems analysts can also benefit from an understanding of the structure and challenges of writing an efficient and useful QlikView application.

## Conventions


In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.


Code words in text are shown as follows: "You can also rename incoming data fields by using the `Alias` statement."

A block of code is set as follows:

```
Load ID as ClientID, Name, Address, Postal Code, City, State from
Clients.csv;
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Navigate to and click on the **Data | Table Files** button and in the **Open Local Files** window, navigate to the location of your copy of the `Hospital_General_Information_Chap3.xls` Excel file."

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).



## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books — maybe a mistake in the text or the code — we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## Questions

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.

# 1

## Connecting to ODBC/OLE DB Data Sources

This chapter gives basic information on how to connect to various database sources in QlikView. New users of QlikView will often create a new QlikView document (application or **QVW (QlikView Worksheet)**) by navigating to **File | New** from the QlikView toolbar. This command opens the **Select data source** wizard, where the user may browse to a locally stored Excel file and follow the wizard steps to extract that data into the new QVW.

In this chapter, we will take a step past Excel and other static file extractions and explore connecting QlikView to a database. The basic steps for connecting QlikView to a database are: driver installation of a supported database, configuring ODBC (if using an ODBC driver), and connecting via the QlikView Script Editor dialog. In addition to exploring database connections, we will also briefly discuss supported databases as well as exciting connectors that will help you connect to other data providers such as Salesforce, LinkedIn, Twitter, SAP, and Facebook using custom connectors.

### Installing drivers

As in any data extraction tool, it is necessary to have **Object Linking and Embedding Database (OLE DB)** or **Open Database Connectivity (ODBC)** installed on the computer or server in order to access a database. The type of connection is determined by the database to which you are connecting.

Microsoft Windows comes pre-installed with several OLE DB/ODBC drivers. For other database connections, you should visit the appropriate database vendor site's download section to get the latest ODBC driver for that database.

OLE DB drivers are most commonly used to connect to Microsoft Access databases, and this driver is usually installed with the Microsoft Windows operating system. Many OLE DB connectors are also available for other **Relational Database Management Systems (RDBMS)** such as **PostgreSQL**, **DB2**, and **Firebird**, among others. OLE DB is reported to be superior to ODBC for QlikView applications (speed, connectivity, efficiency), so consider using OLE DB if possible. An easy way to check whether or not the computer already has the OLE DB connector installed is from within QlikView. In QlikView, navigate to **File | Edit Script**. In the Script Editor dialog's **Data** tab, select **OLE DB** in the drop-down list and click **Connect**. The **Data Link Properties** dialog should appear if an OLE DB driver is installed. If a driver is not installed, you will receive an error message indicating that the OLE DB driver is not installed and a connection cannot be made to the database.

In practical use, most databases (except Microsoft Access) connect through the more widely used ODBC drivers available. Though both 32-bit and 64-bit ODBC drivers can be used with QlikView, when installing a new ODBC driver, select the proper type to work with your version of QlikView. If you are using the 32-bit QlikView application, install the 32-bit driver only. If you are using the 64-bit version of QlikView, you can use either the 32-bit or 64-bit ODBC driver. If the database you are connecting to is a 64-bit database (and you are connecting from a 64-bit QlikView version), be sure to use the 64-bit ODBC driver so you gain speed and efficiency from the 64-bit connection.

Check if your computer or server already has the desired driver installed by navigating to the Windows Control Panel (**Start | Settings | Control Panel**). In **Control Panel**, open the **Administrative Tools** panel, then locate the **Data Sources(ODBC)** icon, and find the ODBC driver in the **System DSN** tab. Click on **Add** in the **ODBC Data Source Administrator** dialog box.

If you cannot find the Data Sources icon, search for ODBC in **Control Panel** to display the **Administrative Tools** window. If you can't find the **Administrative Tools** window, it is likely that no OLE DB/ODBC drivers are installed at all, and you must visit the RDBMS vendor download site in order to install the necessary driver.

When you have installed the proper vendor ODBC driver on the computer or server that will be running QlikView queries, continue the setup by configuring the ODBC DSN information.

## Configuring ODBC

Depending on what type of driver you are using (32 bit or 64 bit), navigate and open the correct administrative setup tool to configure your ODBC connection:

- The 32-bit version of the `Odbcad32.exe` file is located at `%systemdrive%\Windows\System32`
- The 64-bit version of the `Odbcad64.exe` file is located at `%systemdrive%\Windows\SysWow64`

The **ODBC Data Source Administrator** dialog displays and allows you to choose the database to use with QlikView. To add a new connection, navigate to the **System DSN** tab in the **ODBC Data Source Administrator** dialog, select **Add**, then select the ODBC driver for the database to which you are connecting (remember, if your RDBMS is not listed, you must install a driver for it).

After selecting the desired ODBC driver, click on **Finish**. A dialog for the selected database driver is displayed that allows you to enter the name of your data source and other connection parameters. Click on the **OK** button when you finish entering the connection information, and the new database connection name is displayed in the **System DSN** tab.

## Connecting to the database from QlikView

Once the DSN information has been created, continue setting up the connection in QlikView, using the QlikView Script Editor. Open a QlikView document and access the Script Editor (by navigating to **File | Edit Script**, or by doing *Ctrl + E*). In the Script Editor dialog's **Data** tab, select ODBC from the drop-down list and click on **Connect** (if the driver installed is 64 bit, deselect the **Force 32 bit** checkbox). Select the database name in the **Connect to Data Source** dialog and click on **OK**. Enter a password for the data source if prompted. Once connected, notice that QlikView inserts a connect statement into the script, such as `ODBC CONNECT TO [DATABASE NAME ; DBQ=DBNAME] ;`.

From this point, the database is connected to QlikView and you may now create `SELECT` statements to start building your script. If you are not connected to the database when moving forward, you will be prompted for the data source again. In the Script Editor window, press *Enter* once or twice to advance to the next line after the connect statement, and click on the **Select** button to select the database tables. Note that while you can connect to multiple databases in a QlikView document, only one database is connected at a time. Each successive `Connect` statement disconnects the previous connection. You can also use the `Disconnect` command to explicitly terminate the most current database connection. In the next chapter, we will continue building the QlikView script.

## Supported databases

QlikView supports most common databases, and any that use ODBC or OLE DB connections, including the most common database systems: Oracle, MS Access, MS SQL Server, Teradata, PostgreSQL, MySQL, DB2, Sybase, Netezza, and Informix. Connectors are also available via the QlikView Expressor tool, which adds ETL functionality but is not covered in this book.

QlikTech also offers connectors to a number of Software as a Service (SaaS) packages, such as J.D. Edwards, SAP, and Salesforce (additional licensing fees apply). A connector to Informatica (ETL) is also available for purchase. Third-party connectors to LinkedIn, Facebook, Twitter, and others are available for purchase as well.

## Summary

In this chapter, we have discussed database connectivity driver installation, ODBC and OLE DB configuration, and connecting to databases from QlikView. We have also briefly touched on supported databases and other supported connectors to hosted data such as Salesforce, SAP, and LinkedIn. In the next chapter, we will cover the fundamentals of creating QlikView scripts.

# 2

## Creating the Script

A script is the heart of any QlikView application. The script is a code that controls data connections, extractions, modification, association, and storage. This chapter will give you the basics of building a QlikView script and corresponding data model, including load statements, script types, Script Editor commands, and basic script organization.

The QlikView script is a block of code written in a special SQL-like QlikView syntax language, which controls the various parts of the data extraction, transformations, and loading (storage) of data from databases and files to the QlikView application or of data exported to the QlikView storage files (QVDs). The script is created and edited in the **Edit Script** dialog in QlikView and is usually arranged in tabular format for organization purposes. There are many commands forming the QlikView syntax and rules for forming scripts. In this chapter, we will visit some basic elements of data modeling and script building.

### Basics of data modeling

A data model is a logical (or conceptual) depiction of how data is related and queried in a system. In a traditional relational database management system (RDBMS), entity relationship data models reign supreme: individual database tables are created to maximize the efficiency when entering, editing, and storing data. Data is typically not repeated in a relational database, and this eases data input and edits – but makes it much more difficult to retrieve data (query) from the database. Retrieving data from a relational database requires the user or developer to have knowledge of the data structure (in which tables the data resides).

Dimensional modeling (or associative modeling in QlikView-speak) is a more efficient way to arrange data tables that results in a more natural means of querying data. Dimensional modeling in QlikView is the creation of a new data structure (data model) from existing databases and files containing data you want to query. QlikTech, the makers of QlikView, likes to say that this method of associative data modeling works in the way the human brain works – forming relationships between data and allowing questions to be asked starting from any point in the system. This method of modeling is the output of the QlikView script, and it is essential that you take time to sketch out the data model prior to starting scripting. Preparation is the key to any QlikView project, and time invested in designing your data model will pay off in reducing troubleshooting and testing time later on.

QlikView forms the associative data model by automatically joining tables that have identical field names and the same case. This can cause problems by inadvertent joins on fields that have no relationship, but this can be remedied through reassigning names to fields (aliasing or mapping) or concatenating information to combine fields into existing tables.

## Accessing the Script Editor

The main work area in QlikView, and arguably the most important part of the QlikView application, is the **Script Editor** (also called the **Edit Script** dialog). This window (or dialog) can be accessed from an open QlikView document either by clicking the **Edit Script** icon, navigating to the **File | Edit Script** command in the QlikView command bar, or using the *Ctrl + E* keyboard shortcut.

When opening a new QlikView document, the **Create Script** wizard will display, prompting you through a step-by-step method (**Select Data Source**) to extract data from an Excel document. Usually, you will want to bypass the **Create Script** wizard. To do so, click on **Cancel**, then select the **Edit Script** icon, **File | Edit Script** command, or *Ctrl + E* to open the Script Editor window.

The Script Editor window displays. The main part of the Script Editor is the large scripting pane, where scripts are created and edited. By default, the script is already populated with common settings that describe how QlikView should handle numeric values, currency, time, date, timestamp, month, and days.

## Hidden scripts

The **Hidden Script** feature in QlikView is useful when you want to hide portions or all of your QlikView script from users. This feature is typically used to hide the Section Access portion of a load script or when hiding the entire script for security reasons.

Hidden scripts can be created from the Script Editor dialog by selecting the **File | Create Hidden Script** menu command. The **New Hidden Script Password** dialog box appears, prompting you to enter a password for the hidden script and confirm the password. When you click on **OK**, a new script tab will appear with a tiny key icon on it, indicating a password-protected hidden script. To add new tabs to the hidden script portion of your script, select the hidden script tab and click on the **New Tab** icon. This will create additional hidden script tabs. If you want to create non-hidden script tabs, select an unprotected script tab and click on the **New Tab** icon.

When you have the hidden script tab displayed (you've already entered the hidden script password during the QlikView session), you can also manage the hidden script using a couple more commands: navigating to **File | Remove Hidden Script** will remove the hidden script entirely from your QlikView script, and navigating to **File | Change Hidden Script Password** allows you to enter and confirm a new hidden script password.

Once you have created and saved the QlikView document with the hidden script, you can edit the main script as usual (selecting **File | Edit Script** from the main QlikView application), but the hidden script portion of the script is not displayed until the **File | Edit Hidden Script** command is selected from the Script Editor window.

Adding a hidden script can be a good idea when you want to protect the editing or viewing of section access or to protect intellectual property, but it comes with some downsides and caveats:

- Binary loading will not be allowed in any applications with a hidden script.
- The hidden script is always at the left-most side of the scripting dialog and cannot be moved. Scripts always run in tab order, so the hidden script (and all hidden script tabs) will be executed before the main (unprotected) script runs.
- The progress information displayed in the script execution window will not be displayed when a hidden script is run.
- If the log information is enabled for the script, no logging will be recorded if a hidden script exists.
- Section access in the main script will not be allowed if section access exists in the hidden script.



## The Script Editor features and commands

The QlikView Script Editor is the main workplace when developing the scripting elements of your QlikView application. Some useful features in the Script Editor include the following menus:

### The File menu

The **File** menu contains commands regarding saving the script and document, reloads, importing scripts, hidden scripts, and access to the Table Viewer.

- **Save Entire Document, Save Entire Document As..., and Save External Script File:** These commands are available from the **File** command menu and the **Save Entire Document** command is available as an icon in the toolbar. The commands save the corresponding files and allow you to save the document or script file as an alternatively named file. The **Save External Script File** command allows you to save the entire script as a `.qvs` file that can later be imported into other QlikView documents. Save early and often; the best practice is to save (**Save Entire Document**) before each reload after editing the script in any way. You can set up QlikView to automatically save the script before reloading data in the **User Preferences** dialog box, accessed from the main QlikView screen by navigating to the **Settings | User Preferences | Save** menu command and checking the **Save Before Reload** checkbox. Let's have a look at some of the commands:
- **Reload:** This is available from the **File** command menu or the toolbar icon. This command closes the Script Editor, reloads the script, and shows the progress of the script execution and any errors encountered. If the script reload process is cancelled and the QlikView application already contains previously loaded data, QlikView will ask if you want to reload *old* data or not. The best practice is to accept the reload old data option, so sheet objects and charts display with data.
- **Open External Script File:** This command is a handy way to import previously saved script files (the `.qvs` files) and text (`.txt`) files containing scripts. When importing script files using this command, the newly imported script is added in a new script tab, with the name of the script file as the new tab name. This new script can be edited as needed once imported.

- **Table Viewer:** The Table Viewer is a fundamental tool for helping to create QlikView applications. The Table Viewer is also available (and easier to access) from both the icon toolbar in the Script Editor and the main icon toolbar in the main QlikView application. This tool allows you to graphically view the table structure, relationships, metadata, and associations of source and model tables in your application. The Table Viewer serves as the main troubleshooting and visualization tool for your data model.

## The Edit menu

The **Edit** menu contains basic script editing and *tidying* commands that make the script easier to read, or disable (via commenting) blocks of script code. Let's have a look at the commands:

- **Undo, Redo, Cut, Copy, Paste, Delete, Select All, Find/Replace, Indent, Outdent:** These are basic editing tools as found in many software packages such as Microsoft Word.
- **Clear Entire Script:** This clears the active (open) script tab and removes the script tab (and cannot be undone). This command does not affect any other script tabs. You can also delete/clear a script tab by using the **Tab | Remove** command.
- **Upper Case, Lower Case, Invert Case, Capitalize:** These commands control the manipulation of text case. These commands can be useful in code organization and changing the case of case-sensitive script areas (some areas of QlikView scripting are case sensitive, such as accounts and roles in Section Access, which should be uppercase).
- **Comment, Uncomment:** These are very useful commands that allow you to add descriptive text comments (ignored by the script compiler) or disable portions of the script by using comments. You can also use other commenting commands, discussed in the next section, *Script commenting*.

## The Insert menu

The **Insert** menu contains powerful commands to insert many types of script elements into the script code, from variables to including files and connecting statements. Let's have a look at the commands:

- **Set Statement:** This command allows you to create and define `SET` statements using custom variables or many predefined variables (system variables, error variables, and so on.).

- **Environment Variables:** This command inserts the standard list of environment variables that are usually present in a script when a new QlikView document is created (money, time, date, and so on). These statements can be edited once inserted into the script.
- **Script Files:** This command allows you to browse for specific script files (.sql, .txt, or .qvs) and insert the contents of the script file at the cursor location.
- **Include Statement:** This command inserts a reference to a specific file (.sql, .txt, or .qvs). It differs from **Insert | Script Files** by not inserting the content directly into the QlikView script, but only referencing this file. The include file may contain a reusable script, query, connection string, or subroutine you wish to use in your script. The beauty of the include file command is that the data is reusable and does not have to be edited in QlikView itself. It is powerful when using connection strings, e-mail lists, and similar information that may change often.
- **Domain SID:** This command inserts the domain security ID into the script. This value is the identification number of the NT domain in which the user is logged in. This value may be used to set up section access when restricting users by NTDOMAINSID.
- **Test Script:** This command inserts a sample test script to illustrate the functioning of the QlikView script execution, objects, and debugging.
- **Load Statement:** This command allows for insertion of a load statement from either an external file (spreadsheet, text, XML, QVD, and so on) or an inline wizard that allows for creation of data internally in a table.
- **Section Access:** This command inserts section access information either from the Publisher Authorization location published by the QlikView Management Console (usually in a format similar to `http://localhost:4780/QMS/AuthTable`) or inline section access. If using Publisher Authorization, be sure to make the URL on the server a trusted site in Internet Explorer. If using inline section access, a wizard displays, allowing you to enter the necessary information and what type of section access is desired (username, NT domain, and so on).
- **Connect Statement, Disconnect Statement:** The **Connect** command opens the **Connect to Data Source** dialog, where the data source can be selected with the connection credentials. This will insert a connect string into the script. The **Disconnect** command will insert a disconnect command into the script, which will disconnect QlikView from the data source.

## The Tab menu

The **Tab** menu is helpful in creating script tabs, moving tabs to the left and right (promoting and demoting tabs) as well as renaming, merging, and removing tabs. Remember that tabs are a way of organizing and ordering script execution, and the script execution runs from the top of the left-most tab to the bottom of the right-most tab. Let's have a look at the commands:

- **Add Tab, Insert Tab at Cursor:** These commands add a new script tab at the end of the tab set or at a place determined by where the cursor is in the script, respectively.
- **Rename:** This command allows you to rename the active script tab.
- **Promote, Demote:** These commands move tabs to the left or right, respectively. Tab locations control when the script in that tab will be executed.
- **Merge with Previous:** This command merges the current active tab script with the previous (tab to the left) tab script information. It appends the current tab script information to the bottom of the previous tab script.
- **Remove:** This command removes the current active tab and all the script information it contains.

## The Tools menu

The **Tools** menu assists with setting up the correct ODBC connection, setting the Script Editor preferences, and providing a quick syntax check feature. Let's have a look at the commands:

- **ODBC Administrator 64 bit, ODBC Administrator 32 bit:** These commands open the correct **ODBC Data Source Administrator** dialog that allows you to set up the correct DSN information for the data source connection. Select the proper dialog based on your environment conditions and database.
- **Editor Preferences:** This command opens the **User Preferences** dialog that allows you to configure the Script Editor features, such as text font and size, help features, shortcuts, default scripting engine, and other style elements.
- **Syntax Check:** This command runs the QlikView syntax checker on the script code.

## The Help menu

The **Help** menu displays help information for the specific section of QlikView that is active.

## The Tools pane

The lower section of the Script Editor window is referred to as the Tools pane. This section has four tabs: **Data**, **Functions**, **Variables**, and **Settings**. The Tools pane of the Script Editor is where data sources are defined and connection strings are created, functions and values are created and edited, and other system settings are made.

- The **Data** tab: This tab allows you to create connections to databases and other data sources, connection strings, select statements, and universal file location settings (FTP or relative paths). Connections to ODBC and OLE DB data sources can be made from here as well as custom data sources if you have installed the proper .dll connector as required. A connector that serves as a way to create a QlikView Server administrative tool, `QVSAAdminDataProvider.dll`, is also provided. This QVW metadata extractor can be used alongside, instead of, or together with other tools such as the QlikView Governance Dashboard, available for free from QlikMarket ([market.QlikView.com](http://market.QlikView.com)).
- The **Functions** tab: From this tab, you can find a function type suitable for your application and paste it into the script. The **Functions** tab also contains the correct syntax to aid you in its use.
- The **Variables** tab: The custom variables you have created in the application can be found here and pasted into the script. You can also view and paste system variables into the script from this tab.
- The **Settings** tab: In the **Settings** tab, you can set script privileges to read and write from/to databases and execute external programs (turning these on decreases security and will cause an alert to be triggered, warning users). In this tab, you can also enable the scrambling of the connection credentials in the connect line of the script (inserted into the script when you click on the **Connect** button in the **Data** tab and select the data source and enter credentials).

## Script commenting

Script commenting is a powerful way to organize, describe, and disable/enable lines or an entire section of your QlikView script. For instance, it is QlikView's best practice to include an explanatory comment before each `Load` script for a table. Adding comments to code is also necessary for later code review and editing by other QlikView developers. Often, code created one month ago will be difficult to understand during later months when being reviewed (even by the same developer!). Commented code will be marked by QlikView as green text. Following are some of the several methods of script commenting available to you:

- Using `rem` (`REM`, `Rem`, or `rem`—it's case insensitive) before a line
- Two forward slashes (`//`) before a line of code comments out that line in the script
- One forward slash and asterisk before the line and one asterisk and forward slash after the lines (`/*...<string>...*/`) to be commented out will comment out the entire enclosed code block
- Highlight the code you wish to mark as a comment or uncomment, right-click, and select either **Comment** or **Uncomment** from the shortcut menu
- The highlighted text may be commented out and uncommented by choosing the **Edit | Comment** or **Edit | Uncomment** tabs, respectively, in the Script Editor command toolbar

## Creating the Select statement

Assuming you have any database on your computer or server (Access, MySQL, Oracle, or otherwise—these illustrations use the free Microsoft Access 2010 database, `Northwind1`, on a system running the 32-bit Office 2010), be sure you have the correct driver on your machine; if you are running the 32 bit Microsoft Office 2010 and accessing the newer format of the Access `.accdb` file, you will need to download and install the 32-bit *Microsoft Access Database Engine 2010 Redistributable* driver. Let's walk through creating a brief script in QlikView's Script Editor window.

## Connecting to the database

To get to the **Select Statement** dialog, you have to first connect to a database. The following example illustrates a connection to a Microsoft Access 2010 .accdb database file. Modify as needed for your database or data source type.

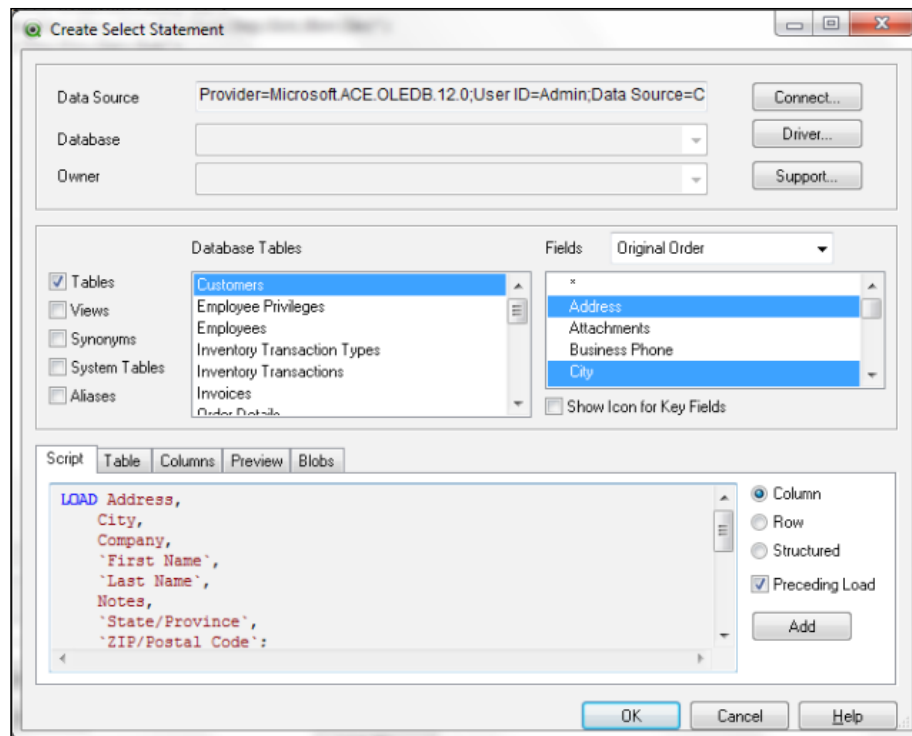
1. Start by opening a new QlikView document (click on **Cancel** when the **Select Data Source** wizard appears). Go to the Script Editor (navigating to **File | Edit Script** from the main QlikView toolbar).
2. In the **Data** tab in the **Tools** pane of Script Editor, select **OLE DB** and check the **Force 32 bit DB** checkbox (we want QlikView to look for installed 32-bit connections).
3. Establish the connection to the database and insert the connect statement into the script by clicking on the **Connect** button on the **Data** tab in the Tools pane, and proceeding through the **Data Link** dialog.
4. Select **Microsoft Office 12.0 Access Database Engine**. On older systems or to access older Access .mdb databases, select the **Microsoft Jet 4.0 OLE DB** driver.
5. Click on the **Connection** tab and enter the full pathname and filename of the .accdb database. Also, enter any connection login credentials if needed.
6. Test the data connection. If successful, click on **OK**, and the connect statement will display in the script window.
7. Click the **Select** button in the **Tools** pane to proceed to the **Create Select Statement** dialog. The **Create Select Statement** dialog will be populated with the database tables and fields for your selection. If the connection was not successful, troubleshoot the connection credentials and path/filename.

## Building the Select statement

Once we've connected to the database, the **Create Select Statement** dialog appears and allows you to customize and narrow your selection statement by selecting various tables and files.

By default, the first table in the list is selected and all columns (*select all* or *select star* with an asterisk) are selected. Each `select` statement can load information from one table and multiple columns (database fields).

In this example, we will select (*Ctrl* + click) from the **Customers** table the following columns: **Address**, **City**, **Company**, **Country/Region**, **E-mail address**, **First Name**, **Last Name**, **Mobile Phone**, **State/Province**, and **ZIP/Postal Code**. Leave checked the **Preceding Load** checkbox and click on **OK**.



The Load/Select statement displays in the script area in the Script Editor dialog. Note that the QlikView syntax Load statement appears above the SQL Select statement, as shown in the following screenshot:

```

11
12 OLEDB CONNECT32 TO [Provider=Microsoft.ACE.OLEDB.12.0;Us
13
14 LOAD Address,
15     City,
16     Company,
17     'Country/Region',
18     'E-mail Address',
19     'First Name',
20     'Last Name',
21     'Mobile Phone',
22     'State/Province',
23     'ZIP/Postal Code';
24 SQL SELECT Address,
25     City,
26     Company,
27     'Country/Region',
28     'E-mail Address',
29     'First Name',
30     'Last Name',
31     'Mobile Phone',
32     'State/Province',
33     'ZIP/Postal Code'
34 FROM Customers;
35

```



## Running your script and viewing results

Click on **Save** and then on the **Reload** button. The **Script Execution Progress** dialog appears briefly and the Script Editor closes, with the main QlikView screen (sheet) open to the **Sheet Properties** dialog. This is the sequence you will see each time after reloading a script. It is fine to click on **Cancel** in the **Sheet Properties** dialog, unless you would like to add one or more fields to the existing sheet.

Click on **Add All** and then on **OK** to display all the available fields on the sheet. All fields are displayed as listboxes on the main QlikView sheet. Select the **Layout | Rearrange Sheet Objects** menu command to organize the listboxes. Make some selections and see how the data changes based on the selections. Click on the **Clear** button to reset the selections.

## Organizing tabs in scripts

Now that you've seen how to connect to a data source and load fields from database tables, you may be wondering about how to organize the tabs in the QlikView Script Editor window. It will help to put some thought into the script architecture up front: how will the tabs be organized –by function or by the application tab? How about by data source? How will you handle variables? mapping tables?

It's up to you, and it will probably be a hybrid approach. There are two main schools of thought in the QlikView community about tab organization: some say organize them by data source and some say organize them by QlikView application's (QVW) user interface sheet tabs (for instance, each sheet tab will have a corresponding script tab).

The best way to organize and add tabs is likely the division along data source lines, with no consideration of the final user interface, QVW. Remember that tab order is important and connections and selections must flow in a logical order (promote or demote tabs as needed and add new tabs using the **Tab** menu). You may want to add other tabs as well, such as mapping tables, variables, clean-up procedures, and exit script. Whatever you do, make your tab names and tab order very descriptive and logical, and add plentiful comments to describe various parts of the script for easier maintenance later on.

## Summary

In this chapter, you've learned what the various QlikView Script Editor commands do, why to use hidden scripts, script and tab organization, and how to connect to a data source and create a basic load statement. In later chapters, we will build on this knowledge and get our hands dirty by adding more complex elements to our script. In the next chapter, we will discuss working with flat files and data transformations.



# 3

## Data Transformations in Flat Files

Original data is rarely in a state such that it can be used directly in QlikView without any modification. QlikView is powerful enough to allow modification of data when the data sources are flat files such as Excel or text files.

Flat files are common data sources for QlikView documents. Since we can't always control how the data is formatted and arranged in the flat files, we have plenty of options when it comes to transforming this data into something we can use in our QlikView application. Most of these transformations are executed in the **File Wizard: Transform** dialog (also referred to as the **Transformation Wizard**).

This chapter describes the modification of data through the various QlikView transformation functions that can be used with flat files such as Excel, CSV, TXT, XML, or HTML.

### Why data transformation?

Flat files are usually not perfectly formatted for use in QlikView; they may contain junk rows, columns, or fields; have blank fields; have repeating columns requiring **unwrapping**; require rows and columns to be rotated or transposed; or have table types considered to be of cross table or generic types. Fortunately, when connecting to a flat data file, the **Transformation Wizard** allows you to manipulate the data before loading it into your QlikView document.

## Using the Transformation Wizard

The **Transformation Wizard** can be accessed from the Script Editor, when you are connecting to a flat file such as a table file. Once you have selected the file, click on the **Enable Transformation Step** button to start the **Transformation Wizard**.

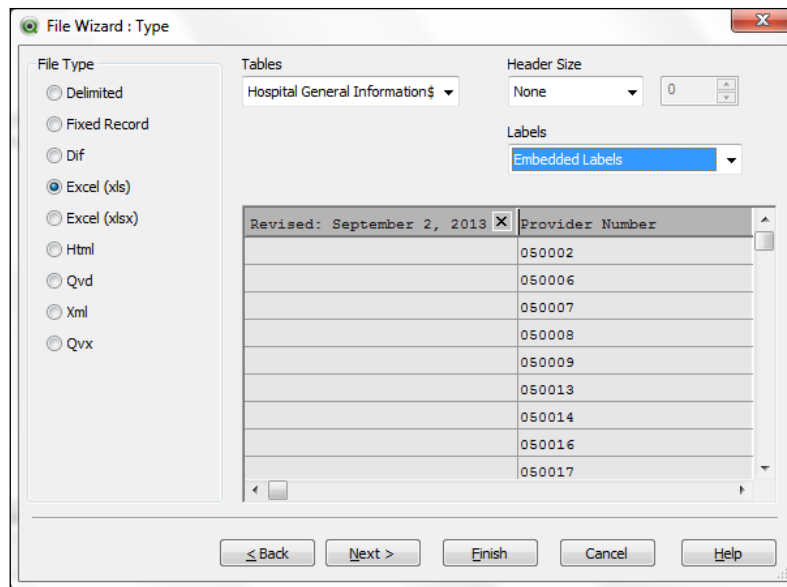
## Removing garbage from data files

Let's start using the **Transformation Wizard** by opening a .xls file and removing some of the garbage (unnecessary) rows and columns from the data. This example uses a spreadsheet available from Packt Publishing, located in the **Downloads** section for this book title. Visit the Packt Publishing website and download the `Hospital_General_Information_Chap3.xls` spreadsheet (this data is adapted from multiple healthcare and hospital spread sheets publicly available at <http://www.data.gov>):

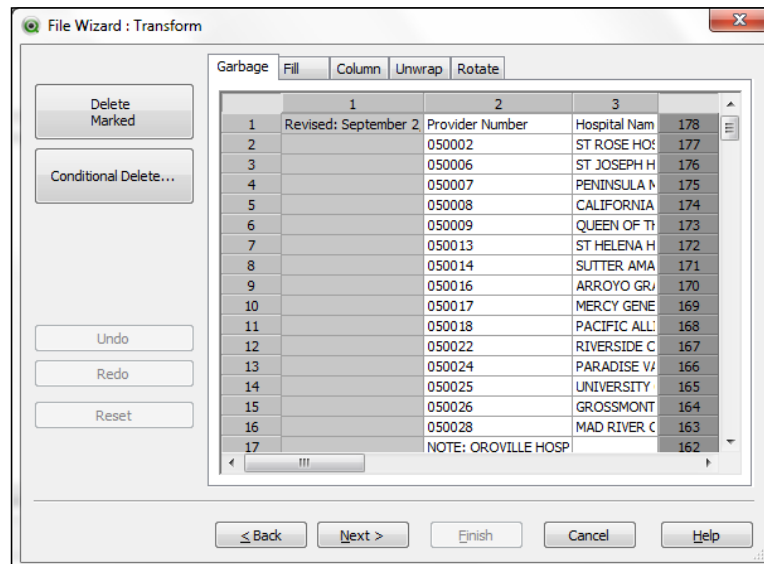
1. From the Script Editor, navigate to and click on the **Data | Table Files** button and in the **Open Local Files** window, navigate to the location of your copy of the `Hospital_General_Information_Chap3.xls` Excel file. The following screenshot illustrates the data fields of this .xls file:

	A	B	C	D	E	F	G	H	I
1	Revised: September 2, 2013	<b>Provider Number</b>	<b>Hospital Name</b>	<b>City</b>	<b>State</b>	<b>Provider Number</b>	<b>Hospital Name</b>	<b>City</b>	<b>State</b>
2		050002	ST ROSE HOSPITAL	HAYWARD	CA	050382	CITRUS VALLEY MEDICAL C	COVINA	CA
3		050006	ST JOSEPH HOSPITAL	EUREKA	CA	050385	PALM DRIVE HOSPITAL	SEBASTOP	CA
4		050007	PENINSULA MEDICAL C	BURLINGAM	CA	050390	HEMET VALLEY MEDICAL C	HEMET	CA
5		050008	CALIFORNIA PACIFIC M	E SAN FRANCIS	CA	050393	DOWNNEY REGIONAL MEDIC	DOWNNEY	CA
6		050009	QUEEN OF THE VALLEY	NAPA	CA	050394	COMMUNITY MEMORIAL HO	VENTURA	CA
7		050013	ST HELENA HOSPITAL	SAINT HELE	CA	050396	SANTA BARBARA COTTAGE	SANTA BARE	CA
8		050014	SUTTER AMADOR HOSP	JACKSON	CA	050397	COALINGA REGIONAL MEDIC	COALINGA	CA
9		050016	ARROYO GRANDE COM	ARROYO GR	CA	050407	CHINESE HOSPITAL	SAN FRANCI	CA
10		050017	MERCY GENERAL HOSF	SACRAMEN	CA	050411	KAISER FOUNDATION HOSPI	HARBOR CIT	CA
11		050018	PACIFIC ALLIANCE MEDI	LOS ANGEL	CA	050414	MERCY HOSPITAL OF FOLS	FOLSOM	CA
12		050022	RIVERSIDE COMMUNITY	RIVERSIDE	CA	050417	SUTTER COAST HOSPITAL	CRESCENT	CA
13		050024	PARADISE VALLEY HOS	NATIONAL C	CA	050423	PALO VERDE HOSPITAL	BLYTHE	CA
14		050025	UNIVERSITY OF CALIF	OF SAN DIEGO	CA	050424	SCRIPPS GREEN HOSPITAL	LA JOLLA	CA
15		050026	GROSSMONT HOSPITAL	LA MESA	CA	050425	KAISER FOUNDATION HOSPI	SACRAMENT	CA
16		050028	MAD RIVER COMMUNITY	ARCATA	CA	050426	WEST ANAHEIM MEDICAL C	ANAHEIM	CA
17		NOTE: OROVILLE HOSPITAL NEW FACILITY SCHEDULED 2/2015							
18		050029	OROVILLE HOSPITAL - MAIN FACILITY						
19		TBD	OROVILLE HOSPITAL - C	OROVILLE	CA	050434	COLUSA REGIONAL MEDICA	COLUSA	CA
20		050036	BAKERSFIELD MEMORI	BAKERSFIE	CA	050435	FALLBROOK HOSPITAL	FALLBROOK	CA
21		050038	SANTA CLARA VALLEY	SAN JOSE	CA	050438	HUNTINGTON MEMORIAL HO	PASADENA	CA

2. In the **File Wizard: Type** dialog, leave the default **Excel (xls)** file type radio button enabled. The **Tables** and **Header Size** drop-down fields should remain the same, but choose **Embedded Labels** in the **Labels** dropdown.



- Click on the **Next** button, and then on the **Enable Transformation Step** button to open the **Transformation Wizard**. The tab in the wizard is open to **Garbage**, allowing us to remove junk data. Select the first column header (as we don't need a revision data column). The column is highlighted. Remove the column by clicking on the **Delete Marked** button. The first column is removed.



4. If you are not re-using this flat file again, you could delete the row with the NOTE: OROVILLE HOSPITAL... text in it, by selecting that row header (row 17 in the Transformation Editor window of the previous illustration), and clicking on the **Delete Marked** button. But a better way to delete this row is to set up a conditional rule, so if the note entry changes often, it will always be deleted upon reload. If you wish to delete all the possible rows marked Note, use the **Conditional Delete** function as follows:

Specify Row Condition

Condition

☒ Compare with value    ☐ Compare with column    ☐ Range    ☐ All Rows

Column

1    contains    NOTE    0

Options

☒ Case Sensitive    ☐ Not

Conditions (AND)

Add    Remove

RowCnd(CellValue, 1, StrCnd(contains, 'NOTE', case))

OK    Cancel    Help

5. Click on the **Conditional Delete** button, then enable **Compare with value**, then select **Column 1, contains (or starts with)**, and enter the word **NOTE** in capital letters. Select the **Case Sensitive** checkbox, and click on the **Add** button. The condition code appears in the textbox. Note that you can add other conditions after clicking on **Add**, or select a range of rows or other columns or conditions in this dialog as well. Click on **OK** to make your changes immediately appear in the **Transformation Wizard**.



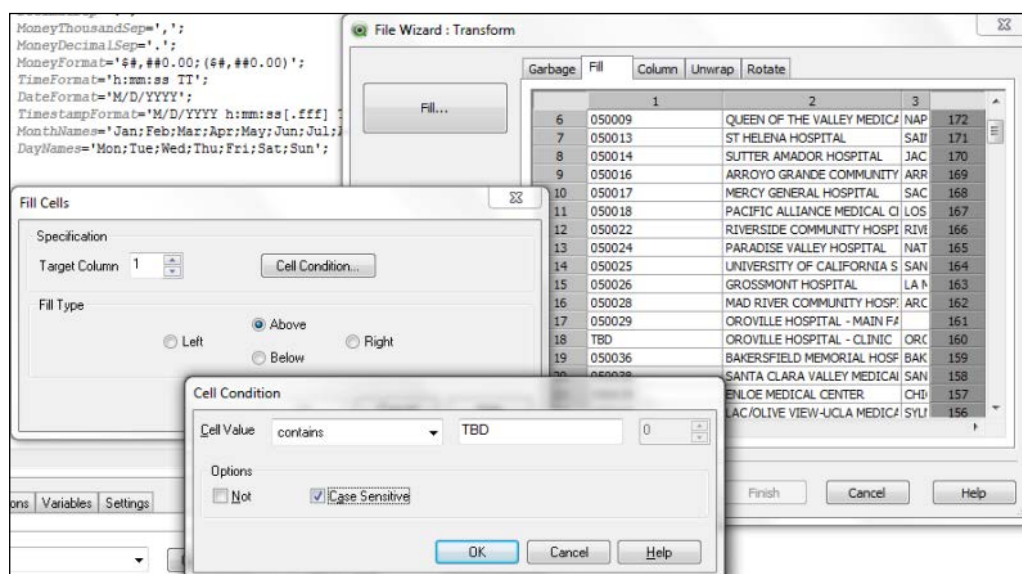
#### Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

## Using the Fill function

The **Fill** function allows you to fill in field data from data cells adjoining the target field. In this example, note row 18 (**Oroville Hospital - Clinic**) has a value **TBD** for the provider ID column. Assume that we know from the business requirements that all fields marked **TBD** are for hospitals that are child facilities of the main hospital facilities. Let's remove any fields marked **TBD** and replace the value with the value of the field directly above it (in this case, the provider ID of the parent facility).

1. Select the **Fill** tab in the **Transformation Wizard**, and click on the **Fill** button. In the **Fill Cells** dialog, leave **Target Column** with the value **1**, and click on the **Above** file type radio button. Click on the **Cell Condition** button, select **contains** in the **Cell Value** listbox, and enter the text **TBD** in the textbox. Select **Case Sensitive** so we don't inadvertently delete any fields that may contain that string in a hospital name. Click on **OK** in the **Cell Condition** dialog and then on **OK** in the **Fill Cells** dialog as shown in the following screenshot:



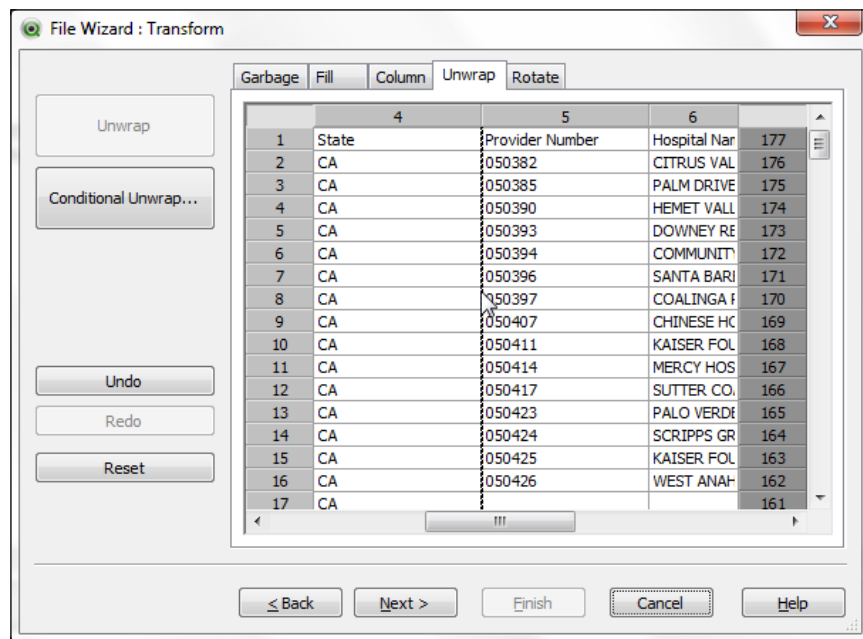
2. Note that both the **Oroville Hospital - Clinic** and **Oroville Hospital - Main** rows now have the same provider ID number **050029**.
3. Continue filling data fields in the same manner for the blank **City** and **State** fields in columns **3** and **4**, respectively. Those columns will need to pull in the data from the cell directly below the target field.



## Unwrapping data from repeating columns

The Excel file we are using for this example has repeating columns (1 to 4 and 5 to 8). We want to transform this data by moving all the columns from 5 to 8 and appending them to their corresponding columns from 1 to 4. We will accomplish this by using the **Unwrap** feature of the **Transformation Wizard**:

1. From the **Unwrap** tab, move the cursor to the row area between columns 4 and 5 (the end of the first set of rows ending with column 4, and the beginning of the repeating set of rows starting at column 5). The cursor changes into a vertical line, and a dashed vertical line appears.



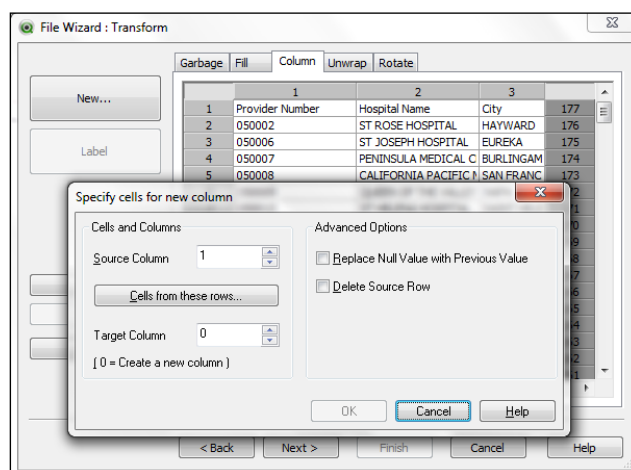
2. Click on the vertical column separator, and the line turns blue. Click on **Unwrap**, and columns 5 to 8 are appended to columns 1 to 4 at the last column.
3. Clean up the data by removing the second set of column headers, which is now located halfway down the table. Do this by selecting the **Garbage** tab and clicking on the **Conditional Delete** button.
4. Click on the **Compare with value** radio button, then select **Column 1**, **contains**, and enter the text `Provider`. Click on **Add**, and the condition code is entered in the textbox.

- Now, set another condition to work within the text condition you just set. Select the **Range** radio button and set the **From** row as **2** rows from the top (skipping the first row) and the **To** row as **1** from the bottom. Click on the **Select** button and leave the default selection scheme as select **1** row at a time, skip no rows. Click on **OK** and then on **Add** to add the query code.
- Click on **OK** to run these two conditions. The additional column headers are removed, but the first column headers remain untouched.
- Clean up any blank rows remaining by using the **Garbage** tab's conditional deletion functionality, by deleting empty rows.
- Once finished cleaning up, click on **Next** and then on **Finish** in the **Transformation Wizard**. Your custom transformation script is inserted into the script pane in the Script Editor. Note that you can always edit the resulting code at any time rather than starting over with the transformation.

## Column manipulation

Now that we've unwrapped the columns, we can further manipulate columns if we want to. Select the **Column** tab in the **Transformation Wizard**. In this tab, you can move or copy data from one column to an existing column or a new column. In this example, we will copy the provider ID column (column 1) to a new column we will create using this wizard (column 5). Duplication of columns within QlikView may be useful in building expressions later when designing sheet objects. This may also be done in the script itself by loading the same column twice and renaming the second column with an alias.

- From the **Columns** tab, click on the **New** button. The **Specify cells for new column** dialog is displayed:



2. In this case, set the **Source Column** value to 1, and leave **Target Column** at 0. Setting **Target Column** to zero automatically creates a new column after any used columns (column 5 is created in this example). Click on the button **Cells from these rows** to open the **Specify Row Condition** dialog.
3. Select the **All Rows** radio button. You could also set this option to move all rows that contain a zero, as all provider IDs start with a zero. Click on **Add** to create the condition query, and then click on **OK**. Advanced options are available here as well if you want to delete the source column, or fill in null values with previous field values (in the same column) when moving data. We are not interested in either of those options, so click on **OK** in the **Specify cells for new column** dialog.
4. Column 1 (**Provider Name**) is now copied to the new column 5—even the column header (since we moved all rows as the condition to move). There is now an issue in that two columns are named identically, so in the next step, we will change the column label for the newly added row.
5. From the **Column** tab in **Transformation Wizard**, highlight column 5 (the duplicate **Provider Name**). Click on the **Label** button and enter a new name in the textbox, such as `Provider Name 2`. Click on **OK**, and the new column name appears in the table pane.

## Transformation script results

When you click on **Finish** in the **Transformation Wizard**, the script code that is generated appears in the Script Editor. Note that it's easier to let the **Transformation Wizard** generate this code, and if anything needs to be tweaked a bit, such as changing a row position or column label name, it can be done in the script itself.

## Rotating tables

It can be useful in data modeling to rotate a table in any direction or transpose (swap) rows and columns. Let's try it using the **rotate** function of the **Transformation Wizard**. This is very similar to the **crosstable** function of the **Transformation Wizard**, but doesn't allow for aggregation of column data. This example uses a spreadsheet available from Packt Publishing, in the **Downloads** section for this book. Visit this site and download the `US_Oil_Imports_Chap3.xls` spreadsheet. From the Script Editor, click on the **Table Files** button and, in the **Open Local Files** window, navigate to your copy of the file `US_Oil_Imports_Chap3.xls` Excel file (this file is adapted from the publicly available file at <http://www.data.gov>), and click on **Next**. The following screenshot illustrates the data fields of this `.xls` file:

	A	B	C	D	E	F	G	H
1	<b>Data 1: U.S. Total Crude Oil Imports of Selected Countries (Thousands of Barrels per Day)</b>							
2	Sourcekey	Region	9/15/2012	10/15/2012	11/15/2012	12/15/2012	1/15/2013	2/15/2013
3	MTTIMUS2	All_Regions	10533	10088	10103	9610	10042	9235
4	MTTIMUSPG2	Persian Gulf	2071	2141	2103	1750	1798	1831
5	MTTIMXX2	OPEC	4268	4186	4195	3554	3850	3094
6	MTTIMIZ2	Iraq	461	593	489	462	419	529
7	MTTIMUSKU2	Kuwait	310	287	276	254	389	255
8	MTTIMUSSA2	Saudia	1291	1257	1325	1032	979	1032
9	MTTIMUSVE2	Venezuela	1035	951	1070	1092	898	601
10	MTTIMUSV2	Non-OPEC	6264	5902	5908	6056	6193	6141

1. In the **File Wizard: Type** dialog box, choose the **Selected\_Imports\$** tab in the **Tables** drop-down list (this will select the correct tab in the Excel file), then set the **Header Size** value to **1** line (this will remove the table title in row 1). Then, choose **Embedded Labels** in the **Labels** drop-down list.
2. Click on **Next**, and then click on the **Enable Transformation Step** button. Select the **Rotate** tab, then click on the **Left** button to see the table rotate counter clockwise. Click on **Right** and the table moves back to the original orientation. In this fashion, you can rotate the table to whatever orientation makes sense for your needs. Click on **Undo** to return the table to the original orientation.
3. With the table in original orientation, click on the **Transpose** button. The table rows and columns swap positions. This orientation may be useful and is close to the crosstable function discussed in the next section. Click on **Next** twice and then on **Finish** to insert your transposition code into the script if desired. The transposed table can also be viewed in the wizard **Results** table window, just before clicking on **Finish**. Note that the month dates have now swapped places with the **Region** and **Sourcekey** codes. This can be used in limited situations, but crosstable functionality (see the next section) may be more useful, however.

Result				
Sourcekey	MTTIMUS2	MTTIMUSPG2	MTTIMXX2	MTTIMIZ2
Region	All_Regions	Persian Gulf	OPEC	Iraq
9/15/2012	10533	2071	4268	461
10/15/2012	10088	2141	4186	593
11/15/2012	10103	2103	4195	489
12/15/2012	9610	1750	3554	462

## Working with cross tables

Cross tables are a common table type that create problems in aggregation and analysis when trying to use them in the QlikView data model. The `US_Oil_Imports_Chap3.xls` spreadsheet we worked with in the preceding rotation exercise is a cross table, because the dates are stored as dimensions and you can't properly aggregate all month data in the present format. We can convert this table into a normal table with the `crosstable` function of the **Transformation Wizard** as follows:

1. In the Script Editor, click on the **Table Files** button and navigate to the `US_Oil_Imports_Chap3.xls` Excel file. Click on **Next**, and then choose the **Selected\_Imports\$** tab in the **Tables** drop-down list (this will select the correct tab in the Excel file). Then set the **Header Size** to **1** line (this will remove the table title in row 1). Then, choose **Embedded Labels** in the **Labels** drop-down list, and click on **OK**.
2. Click on **Next** twice, and then click on the **Crosstable** button. In the **Crosstable** dialog box, select **2** for the **Qualifier Fields** (since we have 2 rows describing the region dimensions). In the **Attribute** field, enter `Date` (note the color coding of the rows and column areas), and in the **Data** field, enter `Barrels`. Click on **OK**.
3. The **Results** tab in the **Transformation Wizard** allows you to view the results of the `crosstable` function. Note that the dates are now rolled up in one dimension.

Result			
Sourcekey	Region	Date	Barrels
MTTIMUS2	All_Regions	9/15/2012	10533
MTTIMUS2	All_Regions	10/15/2012	10088
MTTIMUS2	All_Regions	11/15/2012	10103
MTTIMUS2	All_Regions	12/15/2012	9610
MTTIMUS2	All_Regions	1/15/2013	10042
MTTIMUS2	All_Regions	2/15/2013	9235
MTTIMUSPG2	Persian Gulf	9/15/2012	2071
MTTIMUSPG2	Persian Gulf	10/15/2012	2141
MTTIMUSPG2	Persian Gulf	11/15/2012	2103
MTTIMUSPG2	Persian Gulf	12/15/2012	1750
MTTIMUSPG2	Persian Gulf	1/15/2013	1798

4. Click on **Finish**, and the code is inserted into the script pane of Script Editor. Note in the code the prefix line to the Load statement, named CrossTable.

```
Imports:
CrossTable(Date, Barrels, 2)
LOAD * FROM
C:\QlikView_Scripting\US_Oil_Imports_Chap3.xls
(biff, embedded labels, header is 1 lines, table is [Selected_Imports$]);
```

## Working with generic tables

Generic tables are very common tables; they are also called **standard tables**. These tables have less organization and have many different dimensions and measure units. Each dimension or unit may have its own field. In this example, we will review a scripting solution to create separate tables of each dimension, and then combine the tables into one fact table:

1. From the supplementary material to this book available on the Packt Publishing website, download and open `Generic_Table_Loading_Chap3.qvw`. This QlikView application is a prototype tool for tracking software defects (noting software defect groups, priorities, and stages of development). Open the Script Editor and note the inline table load that builds a generic table.

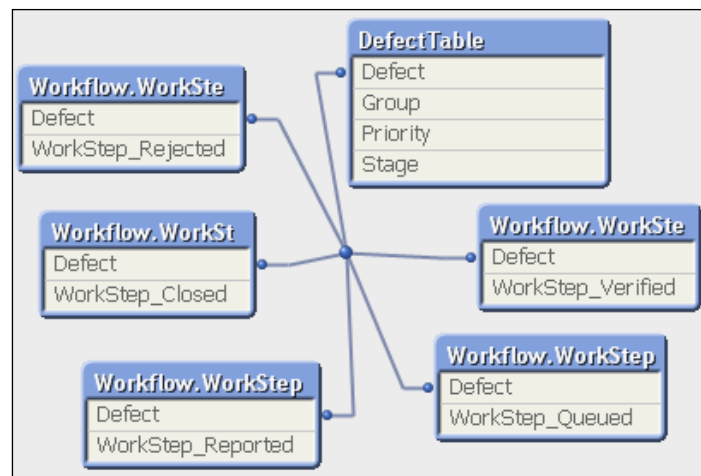
```
DefectTable:
LOAD * INLINE [
Defect, Group, Priority, Stage
1, Reports, High, Queued
2, Data, Low, Reported
3, Data, High, Closed
4, Object, Medium, Rejected
5, Security, High, Reported
6, System, Medium, Rejected
7, Security, High, Closed
8, Object, Low, Queued
9, Data, Medium, Closed
10, Requirements, Medium, Verified
];
```

2. Next, note the `GENERIC` identifier before the `LOAD` statement that follows the above inline table creation for a resident load of the already loaded `DefectTable`.

Workflow:

```
GENERIC LOAD Defect, 'WorkStep_' & Stage, 'x' RESIDENT  
DefectTable;
```

3. The effect of generic loading is that many new QlikView tables are created to handle each new dimension (see the following illustration from the Table Viewer):



4. This result may be fine in many circumstances, but one very useful tip is to clean up these tables and concatenate them into one larger fact table. To do this, place this code in the script after the generic load (modify it for your own example):

```
FOR t = 0 to NoOfTables()  
  TableList:  
    LOAD TableName($(t)) as Tablename AUTOGENERATE 1  
    WHERE WildMatch(TableName($(t)), 'Workflow.*');  
NEXT t  
FOR t = 1 to FieldValueCount('Tablename')  
  LET vTable = FieldValue('Tablename', $(t));  
  LEFT JOIN (DefectTable) LOAD * RESIDENT $(vTable);  
  DROP TABLE $(vTable);  
NEXT t  
DROP TABLE TableList;
```

- After reloading, access Table Viewer once again, to see the following lone concatenated fact table:

DefectTable
Defect
Group
Priority
Stage
WorkStep_Queued
WorkStep_Reported
WorkStep_Closed
WorkStep_Rejected
WorkStep_Verified

## Working with hierarchies

The **hierarchy** function of QlikView is powerful for creating parent-child relationships between two or more values. A quick example of a hierarchy may be: the Solar System is the parent to the child values Sun and Planet, and Planet is the parent to the child Earth. With the hierarchy functions set, QlikView can easily use this data to create tree-view tables.

Let's consider another example using a hospital surgery department structure:

- In QlikView, create a new QlikView document, and click on the **Table Files** button in Script Editor. Navigate to the `QlikView_Scripting` folder and open the file `Hierachy_Chap3.xls` as seen in the following screenshot. This file is a typical adjacent-nodes table used in setting up a parent-child relationship in a data model.

	A	B	C	D
1	PARENT_ID	CHILD_ID	PARENT	CHILD
2		0		Memorial Hospital
3	0	1	Memorial Hospital	Surgical Department
4	1	2	Surgical Department	Cardiology
5	2	3	Cardiology	Cardio Trauma
6	3	4	Cardio Trauma	Dr. Audrey Louise
7	3	5	Cardio Trauma	Dr. Hope Xinlan
8	3	6	Cardio Trauma	Dr. Beth Jackson

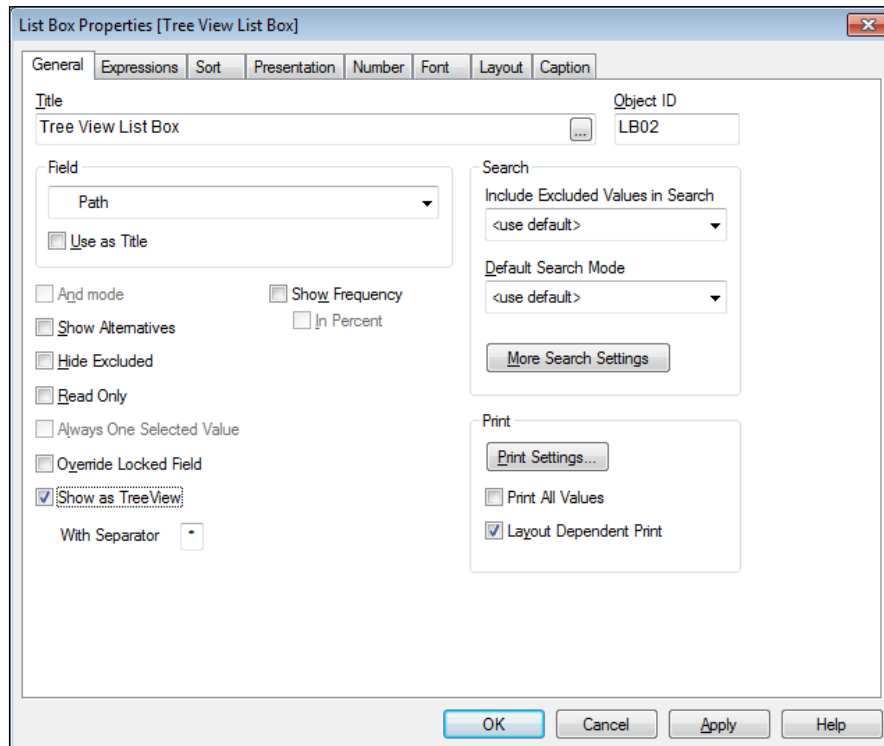


- In the **File Wizard: Type** dialog, select **Sheet1\$** in the **Tables** drop-down list, **None** in the **Header Size** field, and **Embedded Labels** in the **Labels** field. Click on **Next** twice and then on the **Hierarchy** button to display the **Hierarchy Parameters** dialog.
- Set up the parameters in this dialog as shown in the following screenshot. The first three parameters are required, but the rest of them are not (but you will need them to add the tree-view listbox). Note that the **Parent Name**, **Path Name**, and **Depth Name** fields can be named anything you like. The **Path Delimiter** value can be any character as well (be sure to add single quotes as seen). To make the path easier to read, add leading and trailing spaces before the character (in this case an asterisk):

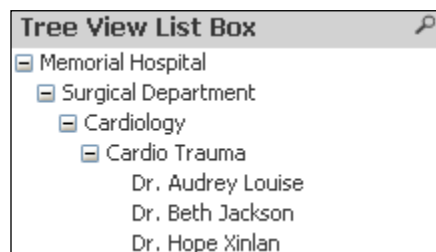
- Click on **OK** and then on **Finish**. In Script Editor, your new **HIERARCHY Load** statement appears. Click on **Save** and then on **Reload**. In the QlikView sheet, create a new table box with all the child and parent fields in it, and examine the layout.

Path	CHILD	CHILD1	CHILD2	CHILD3	CHILD4	CHILD5
Memorial Hospital	Memorial Hospital	Memorial Hospital	-	-	-	-
Memorial Hospital * Surgical Department	Surgical Department	Memorial Hospital	Surgical Department	-	-	-
Memorial Hospital * Surgical Department * Cardiology	Cardiology	Memorial Hospital	Surgical Department	Cardiology	-	-
Memorial Hospital * Surgical Department * Cardiology * Cardio Trauma	Cardio Trauma	Memorial Hospital	Surgical Department	Cardiology	Cardio Trauma	-
Memorial Hospital * Surgical Department * Cardiology * Cardio Trauma * Dr. Audrey...	Dr. Audrey Louise	Memorial Hospital	Surgical Department	Cardiology	Cardio Trauma	Dr. Audrey Louise
Memorial Hospital * Surgical Department * Cardiology * Cardio Trauma * Dr. Beth Ja...	Dr. Beth Jackson	Memorial Hospital	Surgical Department	Cardiology	Cardio Trauma	Dr. Beth Jackson
Memorial Hospital * Surgical Department * Cardiology * Cardio Trauma * Dr. Hope Xi...	Dr. Hope Xinlan	Memorial Hospital	Surgical Department	Cardiology	Cardio Trauma	Dr. Hope Xinlan

5. Now, set up a tree view listbox on the sheet by selecting **Layout | New Sheet Object | List Box** from the menu bar, and set up the parameters as in the following illustration. Make note of the **With Separator** field: it must match exactly what you set up in the HIERARCHY Load statement (in this case, a leading space, asterisk, and a trailing space, but this time with no single quotes):



6. The final result of the tree view listbox for your hierarchy is displayed on the sheet. If it does not work properly, check your **Path Delimiter** and **TreeView Separator** values to ensure they match.



## Summary

In this chapter, we learned how to transform data contained in flat files, so that it can be properly used in QlikView. We've covered all aspects of the **Transformation Wizard**: removing garbage rows and columns; moving or copying columns; rotating data; unwrapping data files; working with generic tables, cross tables, and hierarchies. In the next chapter, we will explore common scripting structures, useful features, and scripting tips.

# 4

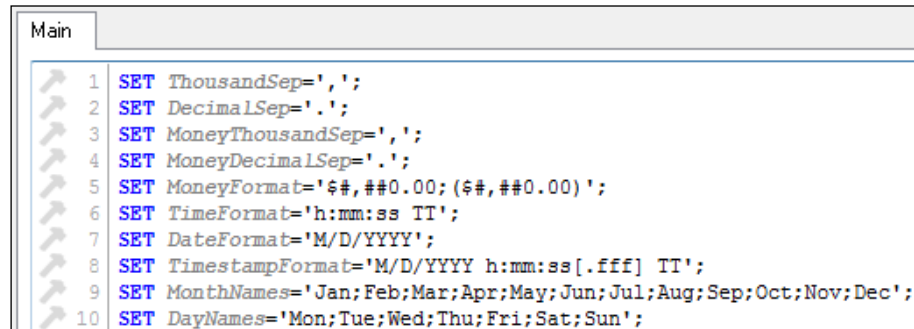
## Script Features and Functions

This chapter discusses basic script structure, features, and the functional areas of a QlikView script. A QlikView script contains several sections that are mandatory (number interpretation variables, script expressions, and load scripts) and some that are optional (master calendar and variables). One common feature you will often notice in scripts is the renaming of field functions (`Alias`, `AS`, `Rename Field`/`Rename Table`, and `QUALIFY`). The renaming of fields is a typical feature of scripts and pervasive enough to be included in this chapter of major script features. Other topics in this chapter include a discussion of script expressions and statements, use of quotation marks, and master calendar placement.

### Number interpretation variables

One of the first things you will notice while opening the Script Editor, for a new QlikView document, is the number interpretation variables. Most of these variables (which control how numbers, currency, time, and dates are displayed and handled) populate by default when creating a new QlikView document. When a QlikView script is run, these variables override any similar settings from the computer operating system.

The following is an example of number interpretation variables in a script. Each `Set` statement defines a variable, such as a comma ( , ) for the variable `ThousandSep`, which separates numeric thousands.



```
1 SET ThousandSep=',';
2 SET DecimalSep='.';
3 SET MoneyThousandSep=',';
4 SET MoneyDecimalSep='.';
5 SET MoneyFormat='$#,##0.00;($#,##0.00)';
6 SET TimeFormat='h:mm:ss TT';
7 SET DateFormat='M/D/YYYY';
8 SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';
9 SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
10 SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
```

Number interpretation variables can be edited (click on the gray hammer icon to the left of the `Set` statement corresponding to each variable). You can add more number interpretation variables by using the **Insert | Set Statement** menu command in the Script Editor.

## Connect, Load, and other statement ordering

In the *Creating the Select statement* section in *Chapter 2, Creating the Script*, we connected to a database and established the `Connect` statement in the script. You will notice that the first major feature of most scripts, after the initial number interpretation and any variable settings, is the `Connect` statement (if you are connecting to a database). If you are not connecting to a database, a `Connect` statement is not present. In the case of not connecting to a database, you are probably using table files or other files for the data source, so only the `Load` statement is present (with no `Connect` statement). The general order and definition of these statements are:

- **Connect:** This statement establishes a link to a database via an ODBC or OLE DB driver. The `Connect` statements are only required when a database source is present and needed for data extraction. The connection will cease upon a `DISCONNECT` statement, upon a new `Connect` statement, or at the end of the script execution.

- **Load:** This statement inserts fields from an external file (such as Excel, text, HTML, and QVD), from data defined in the script, an inline table, a previously loaded table, previously loaded fields, the result of a following `Select` statement, or by generating data automatically.
- **SQL Select:** This statement performs a full SQL query of the fields and tables to load from the current database connection. The `Select` statement is not needed if loading from a file, inline, or script-generated table.

The following screenshot shows an illustration of a typical script `Load` statement, including the `Connect` and `Select` statements. In this example, the `Connect` statement is in the format `Connect32`, which forces a 32-bit connection on a 64-bit system. Note, there is also a `Connect64` format, which forces a 64-bit connection. Also, note the prefix to the `Connect` function: possible prefix values are `ODBC`, `OLEDB` (default, if no prefix is used), and `Custom` (for custom database drivers or connectors). At the end of the `Connect/Load/Select` block of code, there is a `Store` statement, which stores the extracted information into a QVD file.

```

13
14 OLEDB CONNECT32 TO [Provider=Microsoft.ACE.OLEDB.12.0;User ID=Admin;]
15
16 Customer_Table:
17 LOAD Address,
18     City,
19     Company,
20     `Country/Region`,
21     `First Name`,
22     `Home Phone`,
23     ID,
24     `Job Title`,
25     `Last Name`,
26     `State/Province`,
27     `ZIP/Postal Code`;
28 SQL SELECT *
29 FROM Customers;
30
31 STORE Customer_Table into C:\Sample\Customers.qvd(qvd);

```

## Script segmentation via tabs

In the *Organizing tabs in scripts* section in *Chapter 2, Creating the Script*, various strategies were described for dividing the script among tabs in the Script Editor. There are many preferences when choosing the script architecture, but the one most accepted, in the QlikView community, is dividing script tabs by data source and also creating additional tabs for the master calendar, variables, mapping tables, and clean-up procedures. Remember that scripts are executed from top to bottom and from left to right so the tab scripts are executed in the correct order.

One related note about segmentation of the script (and associated QVWs) is that a common architecture is to have a three-tiered QlikView environment:

- Tier 1 application files are the data connection layer. The QVW connects to the database and runs the SQL queries of database tables to generate QVDs.
- Tier 2 is the transformation layer where transformations are performed by tier 2 QVWs with data from tier 1 QVDs. This information is stored in tier 2 data model QVDs.
- Tier 3 is the GUI/presentation layer, which uses tier 2 QVDs for the data model.

Adding, removing, and other scripting tab manipulation can be achieved via the **Tab** menu in the Script Editor. See the *The Tab menu* section in *Chapter 2, Creating the Script*, for details on these tab commands.

## Renaming fields

To build the associations between data fields, QlikView primarily uses field names as the main identifying factor. There are times when, as a developer, you may want to rename the fields in order to correctly link or delink data fields, as in the following examples:

- **Field linking:** If two data fields should be associated but are assigned different names, you can rename the fields with identical names so QlikView treats them as the same. Tables with identical named fields are linked. An important note is that field names are case sensitive in QlikView, and this is critical to field linking/associations.
- **Field delinking:** If two data fields do not contain the same information but share identical names, you will need to rename one of the data fields in order to delink or break the association. The tables with the different data field names will then be delinked.

There are a few methods we can use to rename data fields: the **AS** specifier, **Alias**, **Rename Fields**, and **QUALIFY**.

## The AS specifier

The **AS** specifier can be used directly inside a **Load** statement, such as in the following query:

```
Load ID as ClientID, Name, Address, Postal Code, City, State from
Clients.csv;
```

This method is commonly used because of the ease in using the specifier on a case-by-case basis, directly in the `Load` statement. In the previous example, the data source field named `ID` will be renamed `ClientID` in the resulting data output. This can be helpful when, for instance, many tables have various ID fields, such as shipper ID or product ID. In this case, we only want IDs associated with clients to be associated with each other, and to break any possible links between tables with any other fields of the name `ID`.

## The Alias statement

You can also rename incoming data fields by using the `Alias` statement. This statement precedes the `Load` statement, as follows:

```
Alias ID as ClientID;  
Load * from Clients.csv;
```

In this example, incoming data fields from the data source `Clients.csv` with the name `ID` are renamed `ClientID`. Again, this is useful for establishing or breaking links with tables. The downside of using an `Alias` statement is that the statement can be used far before the actual `Load` statement it affects, which can get lost when reviewing code and debugging by other developers. Another downside is that you cannot use a resident load (a previously loaded QlikView table) that references the original data field name—it must refer to the aliased definition of the field name. This can be a confusing aspect of using the `Alias` statement.

## The Rename Field statement

The `Rename Field` statement can be used to rename one or more QlikView fields after they have been loaded by a `Load` statement. The `Rename Field` statement is positioned after the `Load` statement, and may also use a mapping table for ease of maintenance. The one caveat is that you cannot rename two fields to the same name using the `Rename Field` statement. If attempted, only the first instance of the renaming will take place; all other instances will be ignored. Use the `Rename Field` statement as in the basic example, as follows:

```
Rename field ID to ClientID;
```

In this example, any field of `ID` will be renamed to `ClientID`.



If you use a mapping table (for example, an Excel file with original and new field names), then the format is:

```
Map_Table:
Mapping LOAD DATA_FIELD_NAME, DISPLAY_FIELD_NAME FROM [..\QV\mapping_
names.xls] (biff, embedded labels, table is Sheet1$);
Rename Fields using Map_Table;
```

In this example, you define a mapping table (Map\_Table) as the contents of an external Excel spreadsheet (mapping\_names.xls). The spreadsheet contains the original data field names and the new display names of the fields you wish to use in the QlikView application. In this example, the contents of the Excel file defining the mapping load is illustrated in the following screenshot:

A	B
Data_Field_Name	Display_Field_Name
CID	Customer ID
PID	Product ID
STID	Ship To ID
Code	ZIP
Zone	Sales Area
Resource	Salesperson

Using mapping tables in combination with the `Rename Field` statement is a useful way to ease the maintenance of field names. Since the field names are maintained in one place, developers do not have to search the code for statements such as `AS`, `Alias`, and `QUALIFY`. If you wish to rename the fields, change the new field name in the Excel spreadsheet, save the spreadsheet, and reload the script.

## The QUALIFY statements

At times, you will want to have more control over the automatic association built into QlikView, and a good approach is to fully define, or qualify, field names with both the table name and the field name. The `QUALIFY` statement is used to accomplish this full naming in the format `tablename.fieldname` (a period, or full stop, separates the table name and field name). Using the `QUALIFY` statement allows you to avoid automatic linking of tables using identical field names, because differing fields will not have identical field names using the `QUALIFY` statement. The `QUALIFY` statement can be used as in the following illustration:

```
QUALIFY ID, Name;
[Customers]:
LOAD
```

```
    ID,  
    Name  
FROM  
C:\Samples\Customers.qvd  
(qvd);  
UNQUALIFY *;
```

The result of the preceding code is that the new `Customers` table will contain `Customers.ID` and `Customers.Name`. The data is extracted from the file `Customers.qvd`. The `UNQUALIFY` statement effectively disables any further qualification of data fields coming in after it.

Here's another example, where QlikView forces autoconcatenation because the field names are the same:

```
Customers:  
Load ID, Name  
From C:\Qlikview\Clients.qvd (qvd);  
  
Products:  
Load ID, Name  
from C:\Qlikview\Products.qvd (qvd);
```

In the previous example, QlikView will create one `Customers` table (with the fields `ID` and `Name`) by auto-concatenating both tables (even though we explicitly named the second table `Products`), since the second table has the same field names as the first table. This table will contain data from both the `Clients.qvd` and `Products.qvd` files.

And here's an example using `QUALIFY`, in which we qualify the `ID` field and then concatenate:

```
QUALIFY ID;  
Customers:  
Load ID, Name  
From C:\Qlikview\Clients.qvd (qvd);  
Concatenate Load ID, Name  
From C:\Qlikview\Products.qvd (qvd);  
UNQUALIFY *;
```

The result of the aforementioned Load statement will be a new Customers table with three data fields: Customers.ID, Products.ID, and Name. The two loads are concatenated into one QlikView table (Customers). The Name field has the same name in each table and will not be qualified, since it is not named in the QUALIFY statement. Only the ID data field from each QVD is fully qualified with the QlikView table name and field name. In the case of Products.ID, this qualified QlikView table name is taken from the Products.qvd name, since the second Qlikview table was not named.

The **wildcard** character is useful when using the QUALIFY statement (note the double quotes), as illustrated in the following example. The wildcard can be used when it is impractical to list every field name that must be qualified.

```
QUALIFY "*ID", Name;  
[Customers]:  
Load ClientID, Name  
From C:\Qlikview\Clients.qvd (qvd);  
Load ShippingID, Name  
From C:\Qlikview\ShipTo.qvd (qvd);  
UNQUALIFY *;
```

The result of the preceding code is to qualify any fields ending in ID (ClientID and ShippingID), as well as the Name field. The fields contained in the new Customers table are Customers.ClientID, Customers.Name, ShipTo.ShippingID, and ShipTo.Name.

A useful tip is to use the wildcard when qualifying to initially qualify all fields (thus making all fields uniquely named), then specifically unqualify only the fields you know you will need to use as the linkage between tables, such as in the following example. In this example, we have employed the wildcard in both the QUALIFY statement and the UNQUALIFY statement (note the double quotes in the UNQUALIFY statement).

```
QUALIFY *;  
UNQUALIFY "Key_*";  
Customers:  
LOAD  
    ClientID AS Key_ClientID,  
    Name,  
    ShippingID AS Key_ShippingID,  
    State,  
    Sales_Person  
FROM C:\Qlikview\Clients.qvd (qvd);
```

## Regular statements and script control statements

QlikView scripts consist of a number of statements and expressions. A statement can be either a **regular script** statement or a **script control** statement.

Regular script statements are used for data handling and manipulation, and are the most common forms of script statements. These statements can be written in multiple rows of the script, but must always be terminated with a semicolon (;).

Examples of regular script statements include, but are not limited to:

- Select
- Load
- Join
- Keep
- Concatenate
- Map, Unmap
- Store
- Set

Control statements are optional statements typically used for controlling the logical flow of the script execution, such as in loops. Each clause of a control statement must be kept inside one script line and may be terminated by a semicolon.

Examples of script control statements include:

- if...then...elseif...else...end if
- For...next
- For each...next
- Do...loop
- Switch...case...default...end switch
- Exit script
- Sub...end sub
- Call

One general rule about statements is that prefixes may be applied to some regular statements but not to control statements. The exception to this rule is that the `When` and `Unless` prefixes can be used as suffixes to a few control statement clauses.

Script clause keywords can use any combination of lowercase and uppercase characters, but remember that field and variable names used in the script statements are case sensitive.

Script control statements control the logical flow of the script. They are optional features and can assist when using loops and `if...then` (conditional) statements.

## Script expressions

QlikView chart expressions are used in the frontend chart objects in QlikView, but script expressions are used in the QlikView script itself. Script expressions consist of scripting functions, fields, and operators. The most used script functions consist of aggregation functions and inter-record functions. You may also see statistical functions and financial functions (but these functions are more common in the frontend chart expressions). Other common functions include counter, date and time, conditional, numeric, interpretation, and string functions.

A note about QlikView terminology: the term **fields** corresponds to data columns in a table, in which are contained **records** (which correspond to rows in a table). Fields and records are contained in files (also known as QVDs). Operators are categorized as numeric operators, string operators, logical operators, relational operators, and bit operators.

Script expressions return either a number, a string, or both. Logical script expressions (functions) and operators return either zero (0) for false or -1 for true. Expressions can also convert strings to numbers and numbers to strings.

## Quotation marks

The use of quotation marks in the QlikView syntax language can be tricky, and depends on whether you are referencing fields, values, or variables. Usually, quotations are used for field values inside of a `Load` statement. Note that the use of quotation marks inside `Select` statements may vary, depending on the type of data source and ODBC/OLE DB driver (Oracle, MySQL, DB2, and so on may interpret quotation marks differently). There are some basic rules to follow for using quotation marks in QlikView `Load` statements:

- Single quotes are used to reference literal values. Literal values are strings of numbers or text that will be used as field values.
- Source data field names inside `Load` statements commonly use double quotes. The field names are usually found to the left of the `AS` specifier in a `Load` statement.
- Square brackets can be used instead of double quotes.

A common mistake that developers make is when using a field name contains a character such as a space or dash. In this case, use a double quote or bracket.

For example, if you have a data source field name with a space in it called "Client ID" then

```
Load Client ID
```

will cause a script error since QlikView expects an `AS` specifier or a comma after the word `Client`. The correct syntax is to use brackets or double quotes, such as:

```
Load [Client ID]
```

Or:

```
Load "Client ID"
```

What happens if you use a single quote? If you write this in your script:

```
Load 'Client ID'
```

The text string `Client ID` will be interpreted by QlikView as a literal field value.

When using quotation marks outside a `Load` statement, double quotes signal QlikView to look for a variable, not a source data field name. Outside of a `Load` statement, the string inside double quotes will be interpreted as a variable reference, and the value of the variable (if it exists) will be used.

Out-of-context field references (generated as a result of a `Load` statement) and table references such as the parameters in `Exists`, `NoOfRows()` and `Peek()`, are literal values and need single quotes.

QlikView also allows use of any of the quotation marks or brackets in many places, such as:

- The `Set` statement definitions (to the right of the equals sign)
- Aliases in the `Load` statement (to the right of `AS`)
- File names, URLs, or external table names
- Definitions regarding inline tables
- The first parameter of `Peek()` or `Exists()` inside `Load` statements

## Master calendar placement

One script concept that should be mentioned is master calendars. They are special time-period tables that are linked to fact tables. A master calendar is powerful in fact tables because it does not rely on the dates in the fact table itself as there may be dates for which there is no data. For instance, there may be days on which no goods are sold, and there will be no date available for selection if no data exists.

The master calendar builds out a full calendar, usually by looking (peeking) at the minimum transaction date in the fact table, then autopopulating dates for a range ending at the last transaction date (or the current date). The master calendar is typically the first or second tab in a script, and should ideally be located on its own tab, for easy reference and editing.

Master calendars can be tricky to code, but there are third-party providers of master calendar script creation utilities. Unfortunately, space does not allow us to delve into master calendars in this book, so we encourage developers to study the master calendar coding techniques on their own.

## Summary

By the end of this chapter, you should be able to describe elements of a script, common practices for renaming fields, the `Connect` and `Load` statement functions, script segmentation with tabs, regular and control script statements, and the use of quotation marks in QlikView scripts. In the next chapter, we will review some basic fundamentals of data modeling, as well as use of the QlikView Table Viewer.

# 5

## Basic Data Model and Table Viewer

This chapter describes associative data modeling and how the QlikView Table Viewer can assist you in creating a logical, efficient data model. The QlikView Table Viewer can assist you in documenting and analyzing QlikView applications (exporting data model images and structure), finding connection issues that create synthetic keys and tables, identifying opportunities to conform to the desired star schema model, and previewing data and statistics of data inside QlikView tables.

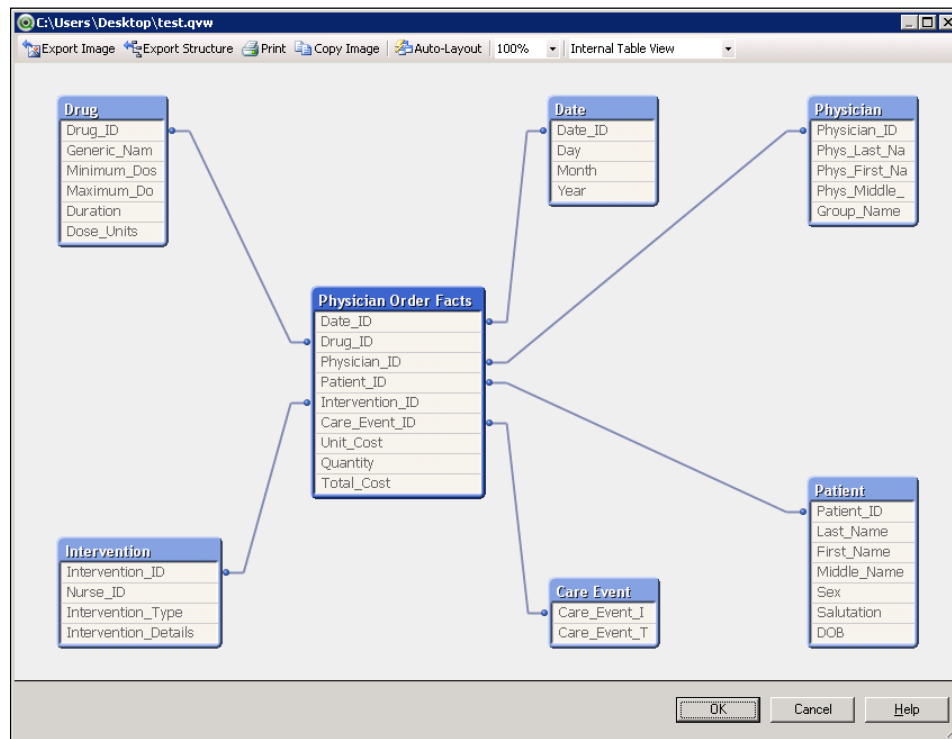
### What is the Table Viewer?

The QlikView Table Viewer is a graphical tool that QlikView developers use to visualize the QlikView data model (the structure, organization, and connections of data elements) of the currently open QlikView application or QVW.

The Table Viewer is accessible from either the Script Editor window or from the main application window of QlikView. To open the Table Viewer tool, you can either select the Table Viewer icon from the menu bar or press *Ctrl + T* on the keyboard, when a QlikView application is open.



When the data model is opened in Table Viewer (as shown in the following screenshot), QlikView tables are displayed as boxes containing rows. These rows correspond to data fields (think of them as columns in a relational database). The name of the QlikView table is displayed in the blue title box on each table, and the field names as defined in the QlikView script are contained in the rows.



QlikView is said to be associative, so the data fields with identical names are automatically linked together with blue connector lines. As a note, you can specifically tell QlikView to not automatically link data fields by either renaming fields as described in *Chapter 4, Script Features and Functions*, or using the `QUALIFY` statement.

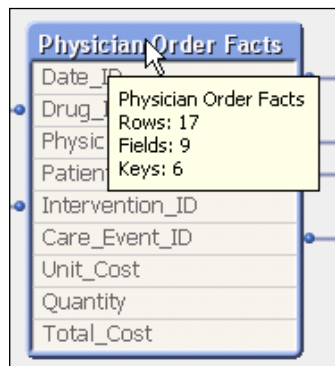
The tables can be moved around the Table Viewer space (by clicking on the table header and then dragging it) and the resulting arrangement is saved when you click on **OK**. This does not affect the connections and data model tables. An alternate method to the developer arranging the tables, viewed in the data model layout in Table Viewer, is to select the **Auto-Layout** command from the menu bar.

This method leads to quirky results, so a best practice is to manually arrange the tables in Table Viewer (if you do at all; many developers do not bother too much with Table Viewer organization, but you should always consult Table Viewer as a best practice to better understand the associations made by your QlikView script). Clicking on **Cancel** in Table Viewer will close Table Viewer without saving the new organization and view of the tables.

Much about a QlikView application's tables and associations can be learned by a thoughtful tour of Table Viewer:

- View data field and table associations by clicking on the table dialog title bar.
- View data field information for tables by moving the mouse over a table dialog title bar. Information such as table name, number of rows, fields, and keys is shown. We didn't do this in the following example, but you can also add comments to a table tool tip by using the `Comment Tables` statement anywhere in the script, such as:

```
Comment table Physician Order Facts with example
```

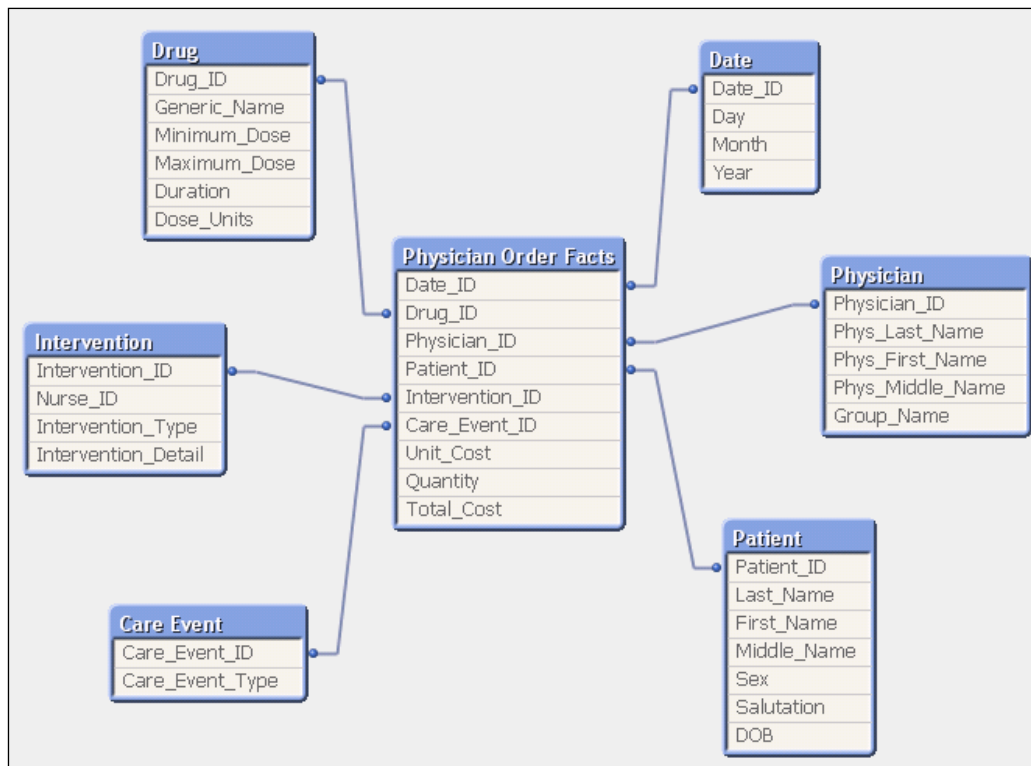


- View associations (connections) between data fields in multiple tables by clicking on a data field in any table.
- Preview data (limited preview of 1000 rows) by right-clicking on a table, and then clicking on **Preview**. See the *Previewing records in tables* section for more information on how to preview the table data.
- View information about each data field by hovering over any data field in a table. See the *Information density and subset ratios* section, for more information on how to view data field information.

## Star and Snowflake Schemas

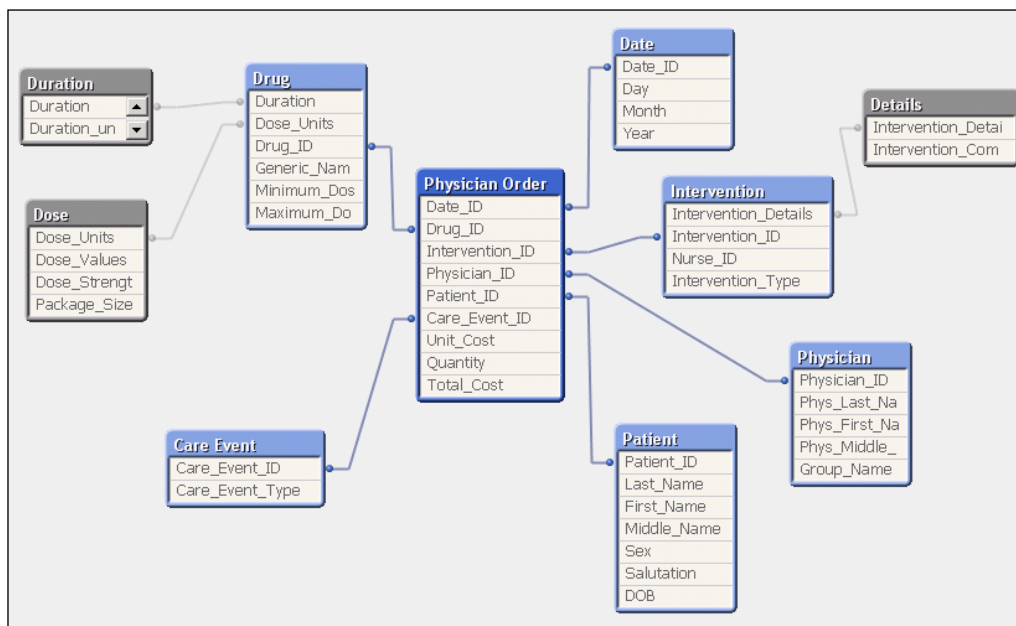
A **star schema** data models typically contain one (or sometimes, two) main fact table(s) (containing events or measures), with dimensional (supporting attributes) tables joined to it via primary keys. The resulting data model (see the following screenshot for a depiction of a star schema) has a hub and spoke, or star-like appearance when visualized (such as in Table Viewer). Star schemas are commonly used for **Online Analytical Processing (OLAP)** purposes, mostly for the high speed that they have to offer. QlikView best practices dictate that developers should model data into a star schema, for the greatest amount of efficiency and speed. Star schemas are highly denormalized data models, free of the limitations of traditional, normalized **Online Transactional Processing (OLTP)** source systems used for creating, inserting, deleting, and modifying records (these are transactions). Typical normalized databases used for OLTP are more strict in terms of data quality (such as no repeating data being stored), but are slower to respond to queries.

The following diagram is an example of a data model in a star schema format:



A **snowflake schema** (sometimes called a centipede schema) is an extension of the star schema. In the snowflake schema, the data model may have one or more fact tables, with connected dimension tables, but will also have secondary dimension tables radiating from one or more primary dimension tables. In the following screenshot of a snowflake schema, there is only one fact table (**Physician Order**), and two of the dimension tables (**Drug** and **Intervention**) also have secondary dimension tables attached to them. Pure star schemas in large systems or companies are somewhat rare; snowflake schemas are the more commonly encountered scenarios due to multiple fact tables and more complex and multiple underlying data sources.

The following diagram is an example of a data model in a snowflake schema format:



In cases where there is a snowflake schema, it may be useful to combine the tables via the **CONCATENATE** statement. This will add rows from one table to another. The **CONCATENATE** statement can be used to combine either fact tables or dimension tables. You can also **JOIN** the tables based on the key field.

More about the pros and cons of manipulating your data model and making it conform to the star or snowflake schema is discussed in the next section.

## Data model best practices

QlikView best practices dictate that, where possible, a star schema is desirable. When not a star schema, the smallest possible snowflake schema is fine, especially limiting the fact tables to one if possible. A scenario could exist, however, where there is only one large fact table containing many dimensions.

In the case of encountering one large single fact table with few or no dimensional tables, there is generally no added benefit of breaking apart the fact table and configuring it into a star (with only measures/events in the fact table and dimensions supporting those measures in many separate dimensional tables). This could be helpful when dealing with large amounts of data that consume large amounts of memory, but in most circumstances it is not helpful.

Similarly, if you have a data model with a snowflake schema, working to consolidate the data and build a star schema is usually not going to cause increased performance/decreased time of query in most cases. This does change with very large data sets, and in those cases, it is worthwhile to spend the time to consolidate and collapse as much as you can, to conform to the star schema.

One of the key considerations is limiting the number of connections required for QlikView to access data, so when working with large data sets, consolidate your data and limit the number of dimension tables, as well as consolidate/concatenate multiple fact tables.

It is worth considering that if QlikView gives you acceptable performance with smaller data sets without conforming to a star schema, you can weigh the consequences and perhaps continue using your current model, but keep an eye on the scalability and data creep.

## Exporting images and structures

The Table Viewer can export images or structures of the existing data model in a QlikView document. These images or structures can be useful in documentation and analysis of a QlikView application. The Table Viewer menu bar, as shown in the following screenshot, can be used for any of the exporting or print commands.



The Table Viewer menu commands help you with the exporting, copying, or printing of data model images and structures, as described in the following lists. Note that the **Internal Table View** and **Source Table View** drop-down list options are discussed in the next section, *Viewing internal and source tables*.

- **Export Image:** This menu command allows you to export the image to either a .png or .bmp file. The default file type is .png, and the default location is the current QlikView document directory.
- **Export Structure:** This menu command allows you to export the current data model to three text files (one each for tables, fields, and mappings). These tables can be imported back into QlikView for analysis on relationships and statistics.
- **Print:** This menu command allows you to print a graphical representation of the current QlikView data model, as currently depicted in Table Viewer.
- **Copy Image:** This menu command allows you to copy the current data model as shown in Table Viewer to the clipboard. This allows pasting the copied data model into a document, e-mail, or graphic editing software.
- **Auto-Layout:** This menu command allows you to automatically arrange the tables and connections in the data model. This can be a useful starting place for you to then manually clean up the data model and continue moving tables. Note that tables should be arranged prior to exporting, printing, or copying the data model image.
- **Zoom:** This menu command allows you to zoom in or out of the data model image in Table Viewer.

## Viewing internal and source tables

The Table Viewer allows you to, use the **Internal Table View / Source Table View** drop-down list options in the Table Viewer menu bar, and to view both the internal (QlikView) tables created in the script and the source tables.

The default selection upon opening Table Viewer is the Internal Table view. This view displays the data model as created in the QlikView script, with all the tables and connections represented. The alternate view is the Source Table view, and this view displays the data tables as they are loaded.

The main difference between the two views is the presence or absence of synthetic tables and data fields that are automatically added by QlikView in order to make sense of the data. In the Internal view, the synthetic tables or fields display. In the Source view, synthetic fields are not present but are represented as multiple connections between tables and data fields. See the next section, for more information on synthetic tables and keys.

## Composite/synthetic tables and keys

If tables with identically named data fields are loaded into QlikView (and you do not `QUALIFY` or rename the fields to unique names), the data fields will automatically associate. The data fields that two tables have in common will link as key fields in their respective tables. This is the associative nature of QlikView.

When you have more than one pair of identical data fields (keys) spanning two or more tables, QlikView creates synthetic keys. When this scenario arises, QlikView creates another table (a **synthetic table**), and adds it to the data model: the `$$syn` table. This synthetic table contains a synthetic key that is a composite of all the combinations of the multiple key fields connecting the tables. This synthetic field, denoted again by the `$$syn` symbol, is also placed in the original data fields connected by multiple fields. This new synthetic field is called a **synthetic key**.

Synthetic keys are undesirable, because they sometimes indicate poor data modeling and this can cause performance problems. Synthetic keys by themselves may not always cause obvious problems. They do not always lead to circular references. They can cause performance problems when there are many synthetic tables with multiple keys each. At times, QlikTech reports that synthetic keys can cause erratic query results. If you have a synthetic keys and tables present in your data model, try to eliminate them if possible.

## Eliminating synthetic keys and tables

Synthetic keys and tables can be removed from the data model in one of the following ways:

- Renaming fields to unique names , using `Alias`, `AS`, or `Rename Fields`.
- Using the `QUALIFY` statement to name fields with unique `table.field` names.
- Commenting out one of the connected table's key fields.

- Using the `Autonumber` statement to create unique key fields. This is often the best solution in terms of memory usage and the simplest to implement. Please note `Autonumber` may be problematic in applications generating the QVD files for use in other QlikView applications.
- Using the `Hash128` function to create unique key fields.

## Previewing records in tables

A useful tool in Table Viewer is the ability to view the data inside QlikView tables in the data model. To view data (limited to the first 1000 rows in the table), left-click on the table title bar, and click on **Preview**. A dialog box displays the data fields in the selected table. This is useful when you are building a script, so you can check the outcomes for a particular change.

In the following example, we are previewing the data fields in the `Physician Order Facts` table. This table has 17 rows, all of which are displayed in the dialog box.

The screenshot shows the QlikView Table Viewer interface. On the left, a data model diagram displays four tables: **Drug**, **Date**, **Physician**, and **Physician Order Facts**. The **Physician Order Facts** table is selected, and its fields are listed in the center: `Date_ID`, `Drug_ID`, `Physician_ID`, `Patient_ID`, `Intervention_ID`, `Care_Event_ID`, `Unit_Cost`, `Quantity`, and `Total_Cost`. On the right, a 'Dialog' window displays the first 17 rows of data from the **Physician Order Facts** table.

Physician_ID	Patient_ID	Care_Event_ID	Intervention_ID	Drug_ID	Date_ID	Unit_Cost
1002	24005	1	2	33043	2456518.5	23
1003	24006	2	4	44963	2456518.5	23
1004	24007	1	6	37873	2456518.5	54
1005	24008	3	3	2393	2456518.5	12
1006	24009	5	2	43409	2456518.5	1
1007	24010	2	4	76034	2456518.5	21
1008	24011	2	6	12435	2456518.5	16
1009	24012	1	5	67543	2456518.5	25
1010	24013	2	6	90554	2456518.5	12
1011	24014	1	7	33445	2456518.5	56
1012	24015	1	6	32467	2456518.5	88
1013	24016	1	9	56774	2456518.5	99
1014	24017	6	12	65774	2456518.5	76
1015	24018	3	3	3226	2456518.5	32
1016	24019	2	4	46678	2456518.5	12
1017	24020	4	5	33346	2456518.5	12
1018	24021	2	3	78886	2456518.5	34

If you want to go beyond previewing data in tables using Table Viewer, you can also view data inside individual QVD files. There is a great third-party tool for doing just that: QViewer, or EasyQlik is a very fast and small footprint standalone file viewer for analyzing the QVD files.

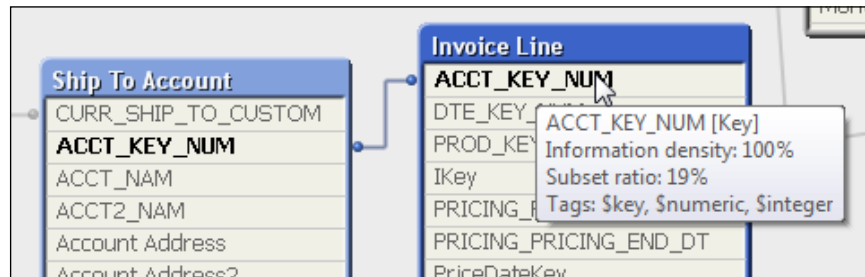


QlikView can view all data inside a QVD file (or partial data for large files), creation date stamps, field duplicates, null values, and count basic aggregations. QViewer/EasyQlik can also help you view data inside a resident table, with an addition of a small code block inside the script. The free version is limited to previewing 50,000 rows per table, and the full price version is unlimited. QViewer/EasyQlik is available for download at <http://www.easyqlik.com/>.

## Information density and subset ratios

You can view information about each data field in a table in Table Viewer by placing your cursor on any data field in any table. A tool tip displays (as shown in the following screenshot) with information about the data field, containing:

- **Information density:** This displays the percentage of rows in the table with values (none are null).
- **Subset ratio:** This displays the percentage of distinct values in this table for this data field (out of all possible distinct values for the data field contained in all tables).
- **Field comments** (if any): These are added by the use of the `Comment Field` statement, as follows:  
`Comment fieldname with 'any comment';`
- **Field tags:** These may be auto-populated from the system (denoted by a preceding dollar sign \$). Custom field tags may also be added and edited by navigating to **Settings | Document Properties... | Tables | Fields | Edit Tags | Add**. An example of the use of the field tags is to mark fields as dimensions or measures.



## Summary

In this chapter, you have learned the various ways that the QlikView Table Viewer can assist you in identifying problems in the data model, such as synthetic keys and nonconformity to a star schema. You will also now understand how the Table Viewer allows for previewing of data and documentation of data models. In the next chapter, we will discuss advanced scripting and data modeling tips.



# 6

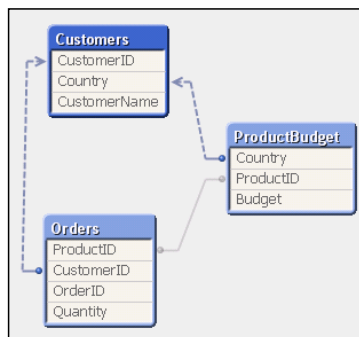
## Advanced Scripting and Data Model Optimization

Scripting syntax contains many different functions and statements, all of which help developers to refine data models. In this chapter, we will discuss circular references and how they are handled, using key (link) tables, mapping tables, concatenation of tables, interval matching, data islands, and metadata.

### Circular references

Circular references are a common issue that occurs when setting up a data model; these circular links occur if there are logical loops in a data structure because three or more tables have more than one path of association. These circular references have no clear start and finish, and make for unpredictable results.

QlikView provides an automatic solution when confronted with a circular reference – the software designates a **loosely coupled table** to one or more of the tables to break the logic loop. See the following diagram for an example of a loosely coupled table; note that the `Customers` table is connected to the other tables via a dotted line. This denotes that the `Customers` table is loosely coupled.



QlikView sets the loosely coupled table as the largest table in the loop (most often, a fact table). QlikView developers can also override this automatic setting of the loosely coupled table by declaring the table that should be considered as loosely coupled in the script, such as in the following code where we select a different table to be loosely coupled:

```
Orders:
Select * from Master_Data;
Loosen table Orders;
```

The preceding `Loosen` statement instructs QlikView to loosen the `Orders` table in order to break the circular reference.

Users can also set the defaults for the handling of loosely coupled tables in the **Loosely Coupled** checkboxes on the **Tables** tab under the **Document Properties** menu for each table in the application.

## Fixing circular references and removing synthetic tables

Circular references and synthetic tables can be corrected in a number of ways. There are many ways to correct a circular reference or synthetic table such as:

- Concatenating two tables together
- Renaming data fields
- Creating a link table

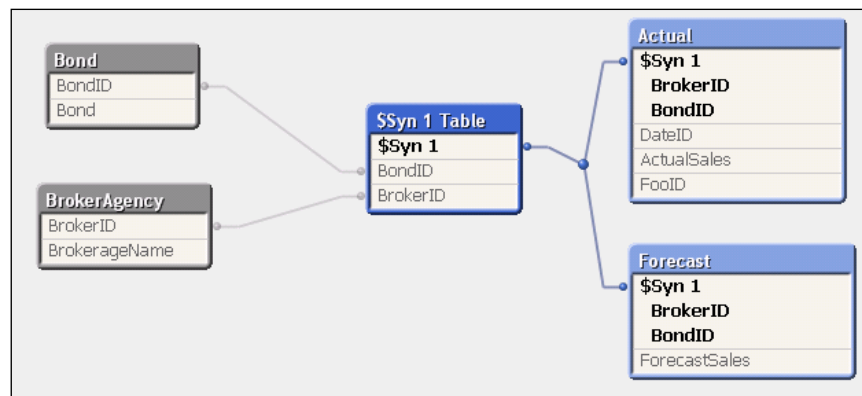
We have discussed renaming data fields in an *Chapter 4, Script Features and Functions*, (`Rename`, `AS`, `Alias`, and `QUALIFY`). `Concatenate` is discussed in a section, *Table combining and concatenation*, later in this chapter. In this section, we will describe using a **key table** (also called a link table) to fix circular references. Key tables work best when you don't want to use the `Concatenate` function, as and when you wish to leave the tables intact and separated. Note that `Concatenate` should, typically, only be used to fix circular references when all identically named facts (or dimensions) in the tables to be concatenated indeed have exactly the same measures or attributes.

Here's a quick example of a data model using a financial services company. In the first section, we do an inline load of brokerages' IDs and names to create the BrokerAgency table, then an inline load for the bond IDs and municipal bond names to create the Bond table, as shown in the following code:

```
BrokerAgency:
Load * Inline [
    BrokerID, BrokerageName
    100, FBSBrokerage
    200, WellsFargo
    300, UBS
    400, NorthGeorgia
];

Bond:
Load * Inline [
    BondID, Bond
    1000, LA
    1001, StLouis
    1002, Atlanta
    1003, NewYork
    1004, Detroit
];
```

In the next section, we will create the tables for Actual and Forecast. Since these two tables will automatically create a synthetic join—because they share two identical data field names—we will create our own key and key table to have complete control over how the script reacts. The following diagram shows how we would get a synthetic key if we proceeded without a key table:



The following code, starting with Hash128, is where we create a unique key from the two shared keys in the table, BrokerID and BondID. Here, we could have used Autonumber, AutonumberHash128, or AutonumberHash256, but we prefer using Hash128 for a good compromise of **low collision probability** and **portability** when using shared QVD instances for several QlikView applications (in other words, the key will work reliably).

```
Actual:
LOAD
    Hash128(BrokerID, BondID) AS ActualForecastID, *
    INLINE [
        DateID, BrokerID, BondID, ActualSales
        269, 100, 1000, 5000
        269, 200, 1001, 10000
        269, 200, 1002, 40000
        270, 300, 1003, 33000
        270, 200, 1004, 25000
        271, 400, 1004, 45000
        271, 100, 1001, 300000
    ];

Forecast:
LOAD
    Hash128(BrokerID, BondID) AS ActualForecastID, *
    INLINE [
        BrokerID, BondID, ForecastSales
        100, 1001, 150000
        200, 1001, 10000
        300, 1003, 30000
        400, 1004, 50000
    ];
```

In the next section, we will build the key table. We will name it with a descriptive name (we prefer adding the word `Bridge` in the name), and start with adding fields from the resident table, `Actual`, that we just added, as shown in the following code:

```
ActualForecastBridgeKey:
Load Distinct
    ActualForecastID,
    BrokerID,
    BondID
Resident
    Actual;
```

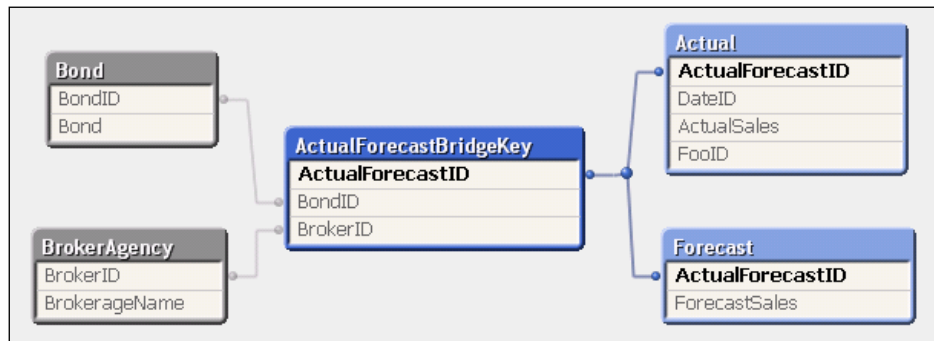
Then, we continue adding to the new key table from the other resident table, Forecast, by doing a Join on the new table (ActualForecastBridgeKey), and load the data fields shared between the two fact tables, including the new key (ActualForecastID) we created, as shown in the following code:

```
Join (ActualForecastBridgeKey)
Load Distinct
    ActualForecastID,
    BrokerID,
    BondID
Resident
    Forecast;
```

Because we added the two shared data fields into the new key table, we can drop them from the two original fact tables as shown in the following code:

```
DROP Fields BrokerID, BondID From Actual;
DROP Fields BrokerID, BondID From Forecast;
```

The final result is a link table, instead of a synthetic table, as shown in the following diagram:



## Using mapping tables to rename fields

Creating user-friendly data field names is the key to helping both developers and end users alike in deciphering the QlikView application. As discussed earlier, you can use renaming techniques on *non-key fields* such as AS, RENAME, or ALIAS. An easier, more consolidated, and easy-to-update method is to use a mapping table. This file may be stored as an external file for quick editing, and is easier for developers to consult and understand.



The main idea of a mapping table is to provide a two-column list of the existing field names and desired field names. The mapping table, in this case, will be an external .xls Excel file. You can also use an inline table, if desired.

To create a list of the existing data field names in your script, you can either comb over your script carefully or do this with a very easy method: On the sheet in your QlikView application, right-click on a blank area, click on **Select Fields** from the menu, and then click on **OK**. Choose to show **System Fields**, then move **\$Field** to the right window to select it (be careful so as not to choose **\$Fields**—plural). A listbox displays all the existing field names. Make sure you **Clear** all the selections in the document (all field names in the listbox should have a white background) and right-click on the list. Go to **Copy to Clipboard | Possible Values**. Open Excel, label the first column (cell **A1**) as *Existing*, and then paste the values into that column (cell **A2**). In the next column (cell **B1**), enter *Desired*, and then enter your desired data field names, corresponding to each of the source data names.

Save the Excel file as *Mapping.xls* to your desired location (in this example, we will use the location as *C:\Users\External*), and make a note of the location for reference in the script later. In the script, you can add the following code:

```
MappingTable:
MAPPING LOAD
    Existing,
    Desired
FROM C:\Users\External\Mapping.xls
(biff, embedded labels, table is [Sheet1$])
WHERE len(Desired)>0;
```

The **WHERE** clause in the preceding code instructs QlikView to only look for non-null values in the *Desired* field using the *len>0* (character length is greater than zero) function.

We can now instruct QlikView to use the new table called *MappingTable*. Add the following code to the end of the script after the **MAPPING LOAD** block of code discussed, as shown in the following code:

```
RENAME FIELDS USING MappingTable;
```

Note that all field names that do not have a data field name under the *Existing* column in the mapping table will not be renamed.

## Table combining and concatenation

In this book, we have emphasized the importance of developing the data model in the star schema format (if possible) and avoiding synthetic tables and circular references. Besides using link tables, key tables, and some of the other renaming strategies discussed in this book and elsewhere, one of the best ways to avoid these issues is to combine tables where it makes sense to do so.

As a note of clarification, the word `Concatenate` in QlikView-speak describes adding the table's rows onto another table. A QlikView `Join`, however, is best described as appending a table's columns onto another table. Most of the time, developers will need to tell QlikView explicitly when to concatenate and join.

Because QlikView is associative, there are times when the software automatically concatenates data fields from two or more separately loaded tables. This automatic concatenation happens when the number and names of the table columns are exactly the same. QlikView will automatically concatenate one statement with another, as the following statements illustrate:

```
Toys:
Load Product_Name, Product_ID, Category from Toys.csv;

Electronics:
Load Product_ID, Product_Name, Category from Electronics.csv;
```

These two statements are essentially treated as one since both the tables have identical columns (data fields) and number of columns (three columns). All the toy and electronic products are combined into one table – the first listed table. The `Electronics` table will not appear in the data model.

If you want to prevent this automatic concatenation, you must rename fields or use the `Noconcatenate` statement. This will prevent the automatic concatenation of tables even if they have identical names and number of data fields. The following code is an example of the `Noconcatenate` statement:

```
Toys:
Load Product_ID, Product_Name, Category from Toys.csv;
Noconcatenate Load Product_ID, Product_Name, Category from
Electronics.csv
```

In most cases, QlikView developers will need to lay out instructions for QlikView to follow for performing concatenations. QlikView calls this forced concatenation, and it is necessary when the tables that have to be combined do not have the same number or names of columns.

A **forced concatenation** is done using the Concatenate prefix (before Load) in the script. This will concatenate the table following the Concatenate statement to the table created immediately before this statement. Here's an example of a forced concatenation:

```
Toys:
Load Product_ID, Product_Name, Category from Toys.csv;
Concatenate Load Product_ID, Category from Electronics.csv;
```

Because we did not specify the table name to concatenate it to in the Concatenate statement (Concatenate TableName is the best practice), the statement appends the rows from Electronics.csv onto the last table created (Toys). The resulting internal Toys table has the Product\_ID, Product\_Name, and Category fields. The number of records in the resulting table is the sum of the number of records in Toys.csv and Electronics.csv. The value of Product\_Name in the records coming from Electronics.csv is null.

## The IntervalMatch function

IntervalMatch is a useful QlikView function that can be used to match a numeric value in one table, such as date, timestamp, score, percent, or any number to an interval or duration in another table. This can be useful in examples such as defining student grades, sales quarters, scoring and metric evaluation, and productivity.

IntervalMatch saves time and memory: because joins are not needed for IntervalMatch, the times to reload are shorter because memory demands and processing needs are less.

The use of IntervalMatch is done after setting up a table containing interval durations (and some other defining attributes such as test scores, percentages, and corresponding grade marks of 90-100, A), as well as an events table (with test scores, number of calls, sales figures, or other numeric measures). IntervalMatch will generate all the different combinations of intervals and events. IntervalMatch creates a table of its own, as well as a synthetic table with all the combinations of intervals and events.

Here's a sample scenario of a sales call center that has targeted call quotas for the salespersons, where lower number of outgoing calls per day is bad (performance) and higher number of outgoing calls per day is good. Load this script into QlikView and run it to examine the output, as shown in the following code:

```
Performance:
LOAD * INLINE [
    Calls_Min, Calls_Max, Performance
    0, 20, Poor
```

```

    21, 30, Fair
    31, 40, Good
    41, 50, Excellent
    51, 100, Outstanding
];

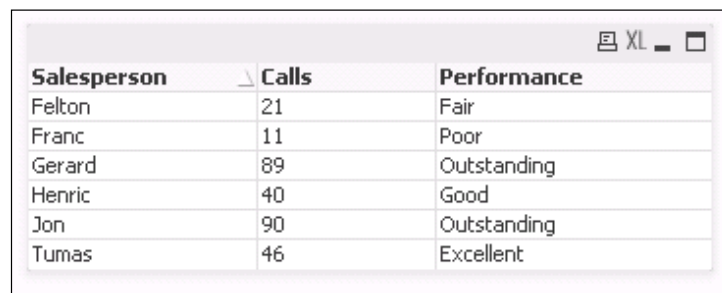
Actual_Sales_Calls:
LOAD * INLINE [
    Calls, Salesperson
    40, Henric
    89, Gerard
    21, Felton
    46, Tumas
    90, Jon
    11, Franc
];

IntervalBridge:
IntervalMatch(Calls) LOAD
Calls_Min, Calls_Max
RESIDENT Performance;

```

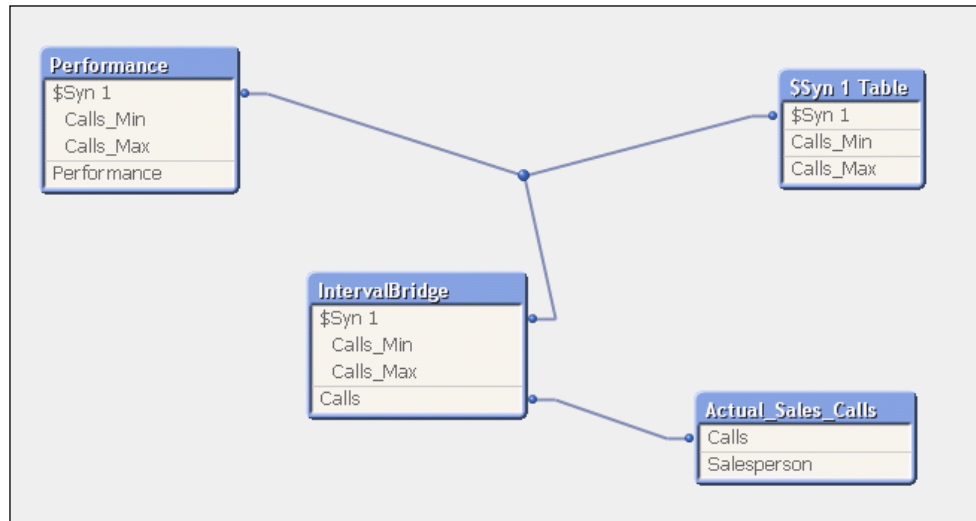
The first Load statement defines the Performance table (this is the Intervals table). The second Load statement defines the Actual\_Sales\_Calls table (actual sales calls per day – the events table). The third section is the IntervalMatch section that ties the Calls data field in Actual\_Sales\_Calls to the Calls\_Min and Calls\_Max metrics in the Performance table.

When this script is run, a synthetic key is created (as designed – do not correct it), and you may create a table box in the application with the name of the salesperson, number of calls, and performance score in it as shown in the following screenshot:



Salesperson	Calls	Performance
Felton	21	Fair
Franc	11	Poor
Gerard	89	Outstanding
Henric	40	Good
Jon	90	Outstanding
Tumas	46	Excellent

And here is the resulting data model. Note the synthetic keys present on **Internal Table View** in Table Viewer. This is the correct functionality – a synthetic table is created; leave this \$Syn table in place as shown in the following diagram:



## Data islands

From time to time, you may want to load data that is a separate table, disconnected from any other field in your data model. This is called a **data island**. Data islands can be created from any data source and are often created as an inline table. Data islands can be used to reference dates, table headers, or whatever you need to be disconnected from most of the existing data in the data model.

One useful example of data islands is to aid in the construction of table charts in the presentation layer of your document, for instance, to allow you to create rows containing measured values rather than dimensions. Let's use a call center example where we track customer representative telephone calls and call times. In the script, you load an inline table named `Call Table` with an inline load statement and one column, `Metrics`. Inside of this table you will specify three records: `Call Quantity`, `Call Duration`, and `Call Center Productivity`, as shown in the following code:

```
[Call Table]:  
LOAD * INLINE [  
  Metrics  
  Call Quantity  
  Call Duration  
  Call Center Productivity  
];
```

Then, we can add some data about the call center, via an inline load:

```
call center, via an inline load:
[Call Center Log]:
LOAD * INLINE [
dateid, calls, call_time_minutes
1, 1000, 200
2, 1500, 300
3, 2000, 400
];
```

From this point, it's a matter of creating a listbox in the frontend of QlikView, and selecting the dimension name of `Metrics` from the **Field** list in the listbox properties **General** tab. This will create the listbox with the three measures you would like to report. The expression for this listbox is dependent on the corresponding row in the data island. In this table, the expression can be set by clicking on the **Add** button in the **Expressions** tab and entering the following expression:

```
If (Metrics = 'Call Quantity', sum(calls), if (Metrics =
'Call Duration', sum(call_time_minutes), sum(calls)/sum(call_time_
minutes)))
```

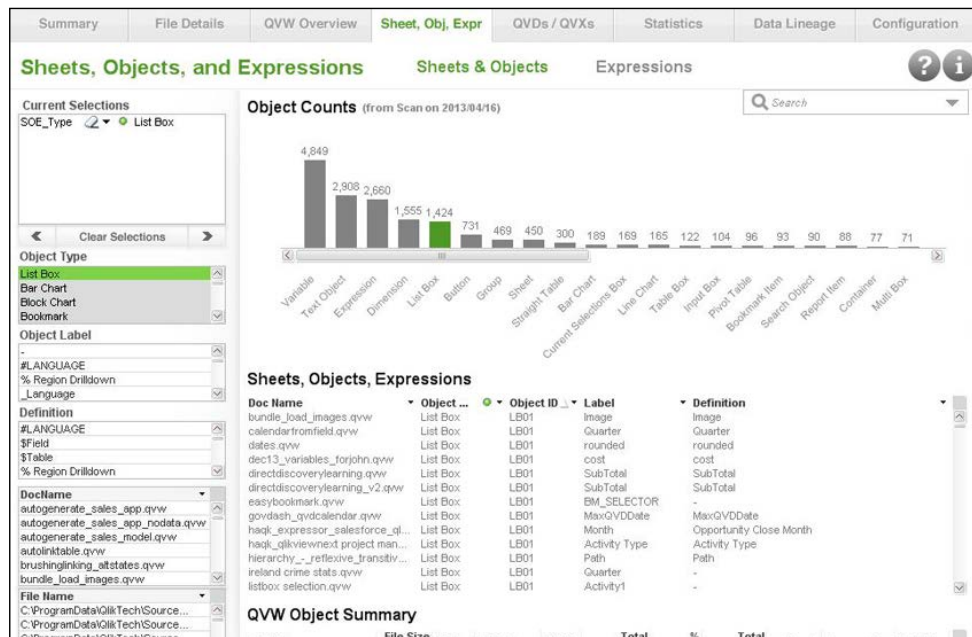
The preceding code will sum all calls if the metric name is equal to `Call Quantity`. If the metric name is `Call Duration`, the row will display the sum of the call time minutes. If the row is anything else (`Call Center Productivity`, in this case), the row will display the sum of all calls divided by the sum of the call time minutes

## Using metadata

Starting with QlikView 10, the ability to add metadata was added as a feature in the software. Metadata is data about data, and for QlikView applications this means information about system- and developer-created tags, as well as objects, QlikView documents, data lineage as it flows from the source to QVD to QVW, and many other bits of information.

The ability for QlikView to create, store, and read metadata has now been built into the core framework of QlikView. How does this help the developers or users? Metadata automatically built into any QlikView document, when combined with the QlikView Governance Dashboard, can be a great help to developers and QlikView administrators.

The following screenshot is an example of the QlikView Governance Dashboard application:



The QlikView Governance Dashboard application contains loads of information and charts on usage trends, job runtimes, QlikView documents accessed and by whom and when, QlikView documentation updates, object/chart counts, expressions in the QlikView documents, data reloads, stored and accessed QVD, and data lineage. Developers and administrators can use the dashboard to test and monitor their applications, as well as expose underlying faults in the data model, chart expressions, or data source issues that require attention.

Previously for applications like the QlikView Governance Dashboard (which is a free download from QlikMarket, <http://market.QlikView.com/>), application developers had to patch together various monitoring applications to access metadata and other system QlikView data.

Other uses of metadata may be using tags and object information to create a searchable QlikView application portal, allowing end users to search for key performance indicators or other chart information created by developers and business intelligence architects.

## Summary

By the end of this chapter, the reader should have an overview of the advanced QlikView scripting concepts and also be able to identify opportunities for data model optimization. In the next chapter, we will discuss data storage in QlikView data files, which are also called QVDs.





# 7

## QlikView Data Files

This chapter describes the creation and use of **QlikView Data Files (QVDs)**. QVDs are important in most QlikView application projects as they provide a means of batch extractions from a data source. This decreases the database and network load. Once a base QVD is generated, an incremental load script can ensure that only the new or updated records are saved to the QVD (and the deleted records are removed) when the script is run again. QVDs also save reload time and space as they are both highly compressed and lightning fast when used as a load source in QlikView queries.

### Defining a QVD

A QlikView data file, commonly called a QVD, is a flat data file that is written in a proprietary and native QlikView format, typically created by a QlikView document. QVDs are data storage files and contain one table as defined in the QlikView script. The file extension of the QVD files is `.qvd`. The use of QVD files is optional, but they are commonly used in complex QlikView applications that load data from databases or very large external data file sources due to reasons including faster speed, less overhead, and increased portability. Simple QlikView applications (for example, applications that only access data from an Excel spreadsheet) may not need to use QVD files as they provide little advantage in speed or portability.

### Advantages of QVDs

QVD files offer many advantages to your QlikView applications, including the following:

- **Faster load times for large data sets:** QVDs load many times faster than any other data source.

- **Incremental loading of QVDs from an active and expanding database:** Incremental loading is a sustainable way of loading data into your QlikView applications, and it reduces the load time dramatically as compared to a full load.
- **Less overhead on databases and networks:** Periodic loads from the data sources into the QVDs can be performed at off-peak times for short durations.
- **Portability across QlikView applications:** Each QVD can be used in many QlikView applications. Binary loads (where entire data models and data sets from one QVW can be imported into another QVW) can only be done once as per the QlikView application. An unlimited number of QVDs and other data sources can be used in any QVW.
- **Reduced file size of extracted data:** Compared to an external file, a QVD file with millions of rows is a fraction of the size, due to the QlikView compression methodology.

## Creating QVDs

QVDs are most commonly created using the `STORE` command in the load script. Each QVD is equal to one logical table (named and created in the script), such as in this example:

```
STORE Sales into C:\QlikView\QVDs\Sales.qvd(qvd);
```

A variable name can be set up previous to this `STORE` statement that defines the QVD storage location. The variable can be created in the following format:

```
set vSaveQVD = C:\QlikView\QVDs\;
```

Once created, the variable can be used as in the following `STORE` statement. The dollar sign (\$) is called dollar sign expansion in QlikView, and it instructs QlikView to evaluate the `vSaveQVD` variable, in this case, to return the location of the saved QVD directory path.

```
STORE Sales into $(vSaveQVD)Sales.qvd(qvd);
```

QVDs must always be created first using a full load, but can be maintained and updated by either doing another full load or performing an incremental load. An incremental load can be a much faster load (in some cases, radically decreasing the load times depending on the data source and data set) and can add new records, or update or delete them (in the data source) in the QVD – depending on how the load script is configured.

## Full loads of QVDs

Full loads are always the starting point for the initial QVD creation, and they may be used at any time that is appropriate for you. Full loads include the following tasks:

- Initial load of QVD of data from data source
- Total refresh of data on a scheduled basis (for example, weekly and monthly)
- Small data sources with fast loads of QVDs
- Fallback load scenario after an incremental load fails

The full-reload process is relatively simple, usually without a date filter, but it can have any selection criteria via the `where` clause, as shown in the following example:

```
tablename:
SQL SELECT Id,
...
FROM tablename;
Store tablename into $(vSaveQVD)QVDName.qvd (qvd);
```

In the preceding script, `tablename` is the logical table in QlikView, `vSaveQVD` is a variable that is set up with the saved QVD directory as done in the preceding section, and `QVDName` is any QVD name you want to use.

Here is a full load example from an Excel spreadsheet, where we have named the logical `Sales` table, loaded data from the `Orders.xlsx` spreadsheet (and from the `Orders` tab in the spreadsheet), and stored the data as a QVD named `Sales.qvd`, in the location defined by the `vSaveQVD` variable:

```
Sales:
LOAD InvoiceID,
     ProductID,
     CustomerID,
     Sales,
     Date_Order
FROM
$(vSourceData)Orders.xlsx
(ooxml, embedded labels, table is Orders);
STORE Sales into $(vSaveQVD)Sales.qvd(qvd);
```

## QVD incremental loads

Incremental loads are defined as loading only new or updated records from the database into an established QVD. Incremental loads are useful because they run much faster than full loads, particularly for large data sets from databases.

At a higher level, incremental loads follow these general steps:

1. Create a QVD via a full load.
2. Periodically run the incremental load by loading the new or updated data from the database into a table in the QlikView script.
3. Load in and concatenate the old data currently in the existing QVD to the newly created table in the script.
4. Create a new QVD file containing the old and new data, and repeat this for each QVD that you need to update incrementally.

The following is an example of an incremental load that uses an Excel file for the data source and handles (not deletes) incremental loads for updates and new data. In this particular case, we are using the embedded QlikView `QVDCreateDate` function (this is a time stamp updated by QlikView each time the QVD is created or updated). In the Excel file are two tabs, `Orders` and `More_Orders`. The documentation is in the code (using comment notation `//` or `/*` for clarity and allowing you to use the full code in a script).

We need to set several variables first. So, we will set a variable for the directory location of the source data files (`vSourceData`) and a directory location for the QVD file that we created and saved (`vSaveQVD`). Then we create a variable that is the date of creation that is embedded into the QVD from QlikView (the embedded date is `QVDCreateTime`; the new variable we create from that is `vQVDCreateDate`). We do this so that we can reformat it with the `date` function and call this variable `vUpdatedDate`.

```
*/

set vSourceData= 'C:\QlikView\DataFiles\';
set vSaveQVD = 'C:\QlikView\QVDs\';
Let vQVDCreateDate = QVDCreateTime('$ (vSaveQVD) Sales.qvd' );
Let vUpdatedDate = date('$ (vQVDCreateDate) ');

/*
```

The following code block is the first full data load. After it loads once, we can comment it out, so that only the incremental loads run after the QVD is established. Other ways to accomplish this may be by checking to see if the QVD already exists and doing a full load only if it does not exist. This full load extracts all data from the `Orders.xlsx` worksheet and stores it as a QVD file (`Sales.qvd`). The locations of the Excel QVD files that are to be saved are represented by the variables we set earlier.

```
//START OF FULL LOAD

Sales:
LOAD InvoiceID,
      ProductID,
      CustomerID,
      Sales,
      Updated

FROM
$(vSourceData)BrokerSales.xlsx
(ooxml, embedded labels, table is Orders);

STORE Sales into $(vSaveQVD)BrokerSales.qvd(qvd);

//END OF FULL LOAD

/*
```

Next, we load the updated or new records from the source data. Note that we are pulling this data from a new tab in Excel (`More_Orders`). This tab has the new and updated records. We use the `NoConcatenate` function because we don't want anything concatenated yet.

```
*/

Incremental:
NoConcatenate
LOAD InvoiceID,
      ProductID,
      CustomerID,
      Sales,

/*
```

Using the following date function sets the data in the `Updated` column to explicitly appear as a date:

```
*/  
  
    date(Updated) as Updated  
  
FROM $(vSourceData)Orders.xlsx (ooxml, embedded labels, table is More_  
Orders)  
  
/*
```

The next line selects only those records with updated or new records—this is where the column in the data source Excel sheet, `Updated`, is later than the variable value we created from the `QVDCreateDate` in the QVD. This does not address deletions.



Note that in a live database environment, the time between the querying of the database and updating of the QVD could lead to some rows being lost. In this case, you could rewrite the date stamp part of the script for the update time or persist the date before the data load is executed. This example is fine for reading from the Excel files with less robust usage and updates.

```
*/  
  
where Updated >= $(vUpdatedDate);  
  
/*  
  
Take the information from the QVD and concatenate it with the new  
table we just made (consisting of the new updates or additions), then  
store it back into the BrokerSales.qvd  
  
*/  
  
Concatenate(Incremental)  
LOAD InvoiceID,  
      ProductID,  
      CustomerID,  
      Sales,  
  
/*
```

Using the following `date` function sets the data in the `Updated` column to explicitly appear as a date. Note that because we are changing the format of the field in this QVD that is read, the data read will be in a standard mode. One solution would be to change the field format using the `date` function when the original QVD file is created, and read the field without any function. This would get you to the optimized load.

```
*/

    date(Updated) as Updated
FROM
$(vSaveQVD)BrokerSales.qvd (qvd)

/*
```

We use the `where not exists` clause because we don't want to pull in any rows from the existing QVD with the same `Invoice_ID` values as the rows we just pulled in from the data source. Note that it is critical that the ID used here is unique to each row.

```
*/

    where not exists(InvoiceID);

/*
```

Then we store the new concatenated (old, new, and updated data) table into the `BrokerSales.qvd` file. Lastly, the `Incremental` table is dropped.

```
*/

STORE Incremental into $(vSaveQVD)BrokerSales.qvd(qvd);
drop table Incremental;
```

## Creating QVDs with QlikView Publisher

QlikView administrators with the proper admin credentials may now (with QlikView 11) generate QVDs directly from the QlikView Management Console by inserting the QVD generation script as a task in **System | Supporting Tasks | QVD Generation**. Administrators should enter information in the **Basics** and **Parameters** fields, and click on **Apply**. The task will need to be set up and run to generate the QVD.

The benefit of creating a QVD in this way is avoiding the maintenance of separate QVW files solely to create QVDs.



## Loading from QVDs

A QVD file is queried as a data source by QlikView, and the `Load` statement in the QlikView script is all that is necessary to read the QVD data into memory, such as in the following examples.

Note that QVDs can be read in two modes: optimized (fastest) and standard (fast). The mode is selected automatically via QlikView and is determined by the presence of any data transformations (and formulas or functions acting on the fields). Only in a straight direct read (no transformations) is the optimized mode used. The optimized mode is 10 times faster than the standard mode (though the standard mode is much faster than reading from a database or flat file).

The following `Load` statement extracts all columns and rows from the `Sales.qvd` file:

```
LOAD * from Sales.qvd (qvd);
```

The next `LOAD` statement loads only `Name` and `ID` from the `Broker.qvd` file.

```
LOAD Name, ID from Broker.qvd (qvd);
```

And the final example loads the columns and uses the `AS` statement to rename those fields (`Name` is changed to `Broker_Name` and `ID` is changed to `Broker_ID`).

```
LOAD Name AS Broker_Name, ID AS Broker_ID from Broker.qvd (qvd);
```

## Viewing content of QVDs

Since QVDs are proprietary format files, you would typically open and view the values contained in a QVD by opening the QVD in QlikView. You can, however, use Notepad or WordPad to view the XML header containing the file attributes of a particular QVD.

It has become common among QlikView customers to install Easyqlik, a third-party QVD viewer tool available at <http://www.easyqlik.com> to view the contents of QVDs (the free version allows you to view thousands of rows, and an upgrade offers more features). Another QVD viewer that is available is Q-Eye. It is available at <http://www.etl-tools.com>.

If you can't install tools like this on your workstation or server, one of the easiest ways to quickly preview the QVD content is to create a new QVW document and drag a QVD onto the QlikView window. This opens the **File Wizard:Type** dialog and allows a limited preview of the columns and rows of data. Clicking on **Cancel** in the wizard closes the QVW and does not create any file.

If you want to examine all the columns and rows in a QVD, create a quick throwaway QlikView document, load the QVD, and then create a table box with all the data fields in it. Note that if your QVD is very large, you should do a partial reload, restrict via dimension limits, create a condition on the expression, or create a calculated limitation for the expression that would limit the rows that are returned.

## Managing reload or no data opens of QVWs

A useful tip when dealing with large data sets and QVDs supplying the QlikView application is to open a QVW without data. From the QlikView start page (if it isn't open, go to the **Start Page** window and navigate to **Help | Show Start Page**), select **Recently Opened Documents**, then right-click on the desired QVW document. A shortcut menu appears, allowing you to open the document without data; so open the document and reload the data. The shortcut menu also allows you to browse the documents in the folder, add the QVW document to **Favorites**, or remove the QVW document from the **Recently Opened Documents** list.

The feature that does not display data upon opening is a valuable tool in avoiding time-consuming data loads when opening QlikView applications.

If the file is no longer in the **Recently Opened Documents** list, you can add a `/nodata` switch to the QlikView program, and this will open your QVW files with no data.



Note that this feature may get you locked out of any section access in the application, so test on a copy of the application first.

## Summary

In this chapter, we have reviewed the usefulness of QVD files, how to create and update the QVD files in both full load and incremental loads, how to create QVDs with QlikView Publisher, how to load data from QVDs, how to view the contents of QVDs, and how to open QlikView documents without a data loading overhead. In the next chapter, we will take a look at debugging QlikView applications and scripts.



# 8

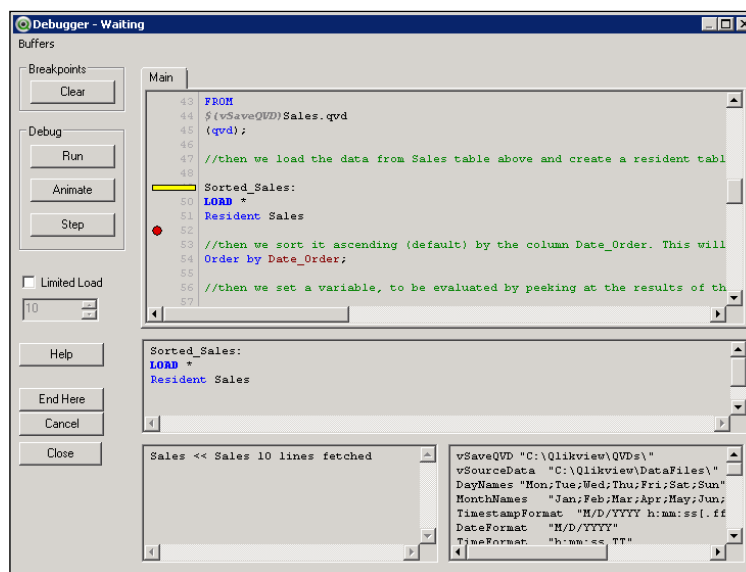
## Debugging

This chapter describes how to use the QlikView tools and functions to help you identify the bugs and flaws in your QlikView scripts using syntax checker, the QlikView script debugger tool, `EXIT SCRIPT`, `TRACE`, partial loads, and logfiles.

### QlikView script debugger

The QlikView debugger is a tool used to help you identify the problems in the script that cause errors or other issues. The debugger allows close monitoring of the script and also allows you to view the script actions and variable settings. The debugger window, as shown in the following screenshot, is accessed by clicking on the **Debug** button in the main toolbar in Script Editor.

The following illustration is a screenshot of the QlikView Debugger dialog (debugger):



The features of the QlikView debugger include:

- **Clear:** In the **Breakpoints** section, this button clears all the established breakpoints. You can set a breakpoint (a pause in the script created by clicking a line number). Breakpoints are represented in the debugger main window by a red circle near the code line numbers. You can set as many breakpoints as you want in the code. Not all lines can support a breakpoint, so it may sometimes be helpful to set breakpoints on multiple lines near where the breakpoint should occur.
- **Run:** In the **Debug** section, this button starts and runs the script execution. The script will run until it hits an error, breakpoint, the `EXIT SCRIPT` command, or the end of the script. It starts the script from the beginning or restarts the script after a breakpoint. It also restarts the script running at normal speed after **Animate** or **Step**.
- **Animate:** This button executes the script slowly and allows you to more easily track the progress and outputs of the script and set breakpoints ahead of the code execution. The script will run until stopped as explained in the previous feature.
- **Step:** This button executes the script one line at a time only, and stops. The user must click on **Step** again to move one line forward, or click on **Run** or **Animate** to start the script execution again. The comment lines will be ignored.
- **Limited Load:** This is a useful debugging feature that saves the developer's time when running scripts against large data sets. The maximum number of rows to be loaded per table is selected in the **Limited Load** box (after enabling the check box). If the maximum number of rows is set too low, note that it is possible that no data associates across tables.
- **Help:** This button displays QlikView Help.
- **End Here:** This button ends the debugger run of the script and closes the debugger window. All the data loaded until that point in the last debugging script execution remains loaded in the current QlikView application.
- **Cancel:** This button stops the script execution in the debugger, closes the debugger, and discards any data loaded so far in the session.
- **Close:** This button closes the debugger immediately (usually with an **Execution of Script Failed** error dialog). If you click on **Yes** to reload old data, the previous data set will be loaded, and the application will remain open. If you click on **No**, the application will be closed with no data loading.



After a successful run of a script in the debug mode, you will need to close the window, usually by clicking on the **X** icon on the debug dialog window.

The QlikView debugger has four windows, as follows:

- The main large window is the script window, which displays the script portions as it executes them. You can set breakpoints by clicking on any code line. Clicking once again on the breakpoint removes it (as does clicking on the **Clear** button). The current script line that is being executed is annotated with a horizontal yellow bar. The **Include files** execution will appear on separate tabs of the main window.
- The lower-middle section of the screen is the script execution window, which shows the current script statement that is being executed.
- The bottom-left section of the screen displays the status codes and script errors, if any.
- The bottom-right section of the screen displays all the variables and current values. If a variable has changed in value, it is displayed in red.

## Using the Exit Script function

It is a good practice to use the `Exit Script` function where needed in the script, instead of a debugger breakpoint. Breakpoints only work while in the debugging mode and will be lost upon the closing of the debugging window. When a script execution encounters the `Exit Script` command, it stops running the script immediately. All the data loaded so far in the script execution remains.

The `Exit Script` command is a useful tool to use when incrementally debugging portions of the script. You can insert `Exit Script` at some point after a `Load` statement, such as before the `Store` statement, then preview the table in Table Viewer to see if the data conformed the way you intended. If all is well, you can remove the `Exit Script` command or comment it out (note that the `Exit Script` command has the same effect as commenting out the remainder of the script as it stops any script execution from that point forward). Typically, `Exit Script` is placed on a separate tab and promoted and demoted to test various sections of the script.

## Using syntax checker

The first line of defence in QlikView debugging is the built-in syntax checker. QlikView automatically color codes the code statements for you (the color and font can be customized by the developer in the Script Editor dialog). You can also explicitly call the syntax checker by selecting the command, **Tools | Syntax Check**, from the menu bar in Script Editor. Paying careful attention to what QlikView is telling you about your code should be of primary importance before running the reload script. The common errors are lack of parentheses (shown in red and bold) in nested statements, missing semicolons and colons, and lack of commas. In the case illustrated in the following screenshot, there is a comma missing after `Sales`. Any code underlined with squiggly red lines is code in error. In the following case, the error starts at the point of `Sales` because there is no comma at the end of that line (comments, in green text, are ignored by the syntax checker). The syntax checker is a useful tool, but it is not perfect. It can miss errors, particularly when encountering the dollar sign expansion of variables.

```

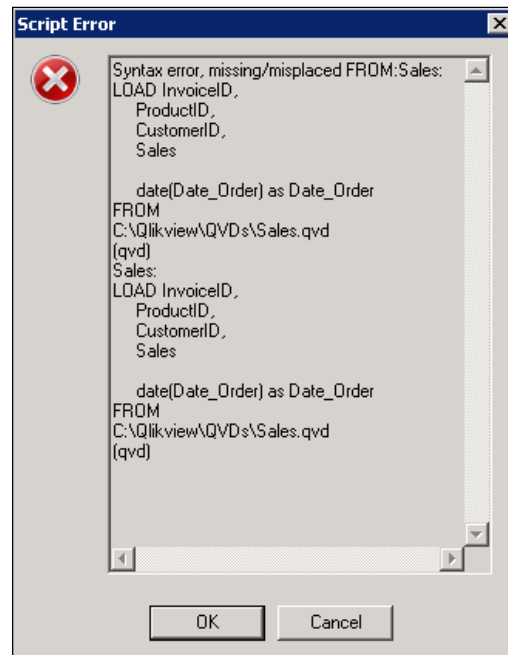
2
3 Sales:
4 LOAD InvoiceID,
5       ProductID,
6       CustomerID,
7       Sales
8
9 //using the "date" function below sets the data in
10 date(Date Order) as Date Order
11 FROM $(vSaveQVD)Sales.qvd
12 (qvd);
13
14 //then we load the data from Sales table above and

```

## Common QlikView script errors

QlikView error messages displayed during the running of the script, during reload, or just after the script is run are key to understanding what errors are contained in your code. After an error is detected and the error dialog appears, review the error, and click on **OK** or **Cancel** on the **Script Error** dialog box. If you have the debugger open, click on **Close**, then click on **Cancel** on the **Sheet Properties** dialog. Re-enter the Script Editor and examine your script to fix the error. Errors can come up as a result of syntax, formula or expression errors, join errors, circular logic, or any number of issues in your script.

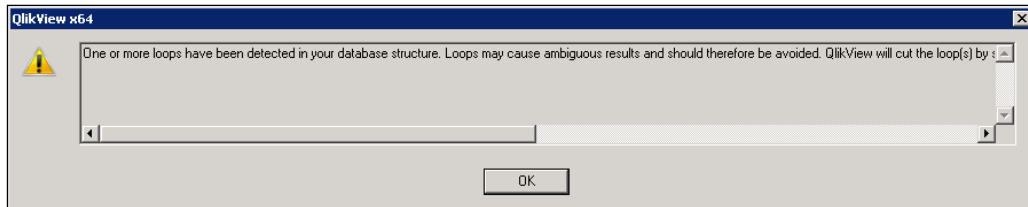
The following are a few common error messages you will encounter when developing your QlikView script. The first one, illustrated in the following screenshot, is the syntax error we received when running the code that missed a comma after `Sales`, from the previous section of this book. This is a common syntax error. It's a little bit cryptic, but the error is contained in the code snippet that is displayed. The error dialog does not exactly tell you that it expected a comma in a certain place, but with practice, you will realize the error quickly.



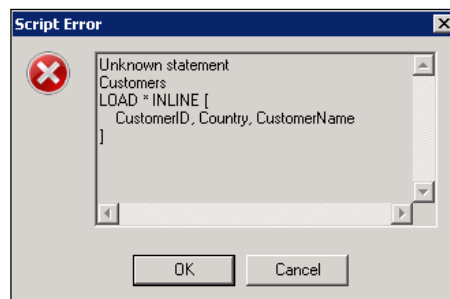
The next error is a circular reference error. This error has been discussed previously in the book and will be handled automatically by QlikView. You can choose to accept QlikView's fix of loosening one of the tables in the circular reference (view the data model in Table Viewer for more information on which table is loosened, or view the **Document Properties** dialog, **Tables** tab to find out which table is marked **Loosely Coupled**). Alternatively, you can choose another table to be loosely coupled in the **Document Properties**, **Tables** tab, or you can go back into the script and fix the circular reference with one of the methods described in *Chapter 6, Advanced Scripting and Data Model Optimization*.



The following screenshot is a warning/error dialog displayed when you have a circular reference in a script:



Another common issue is an unknown statement error that can be caused by an error in writing your script—missed commas, colons, semicolons, brackets, quotation marks, or an improperly written formula. In the case illustrated in the following screenshot, the error has encountered an unknown statement—namely, the Customers line that QlikView is attempting to interpret as Customers Load \*.... The fix for this error is to add a colon after Customers in the following way: Customers:

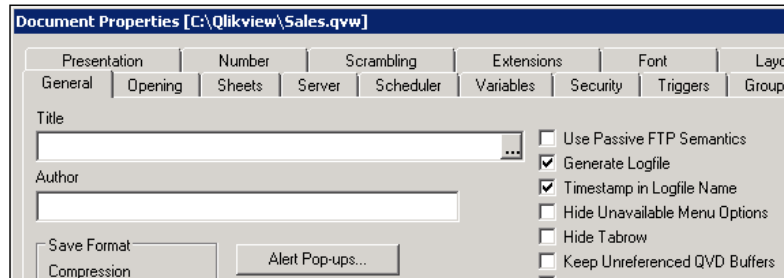


There are instances when a load script will fail silently. Attempting to store a QVD or CSV to a file that is locked by another user viewing it is one such error. Another example is when you have two fields with the same name in your load statement. The debugger can help you find the script lines in which the silent error is present.

## Debugging using logfiles

Script logging for the current QlikView application can be activated by opening the **Document Properties** dialog, **General** tab. If the **Generate Logfile** checkbox is selected (as in the following screenshot), QlikView will create a logfile of the script execution (the output will be as seen in the **Script Execution Progress** dialog when you run the script). This file, with the default name `qv.log`, is stored in the same folder in which the current QlikView application is. If you activate the **Timestamp in Logfile Name** checkbox, the file name is saved with the current script run time stamp in the logfile name.

The logfile is usually the first place to look for errors if a load fails or displays errors and is a key tool in your debugging. Find the errors in the logfile, and work through your script again to spot the errors using the debugger and `Exit Script` function to help you isolate the errors.

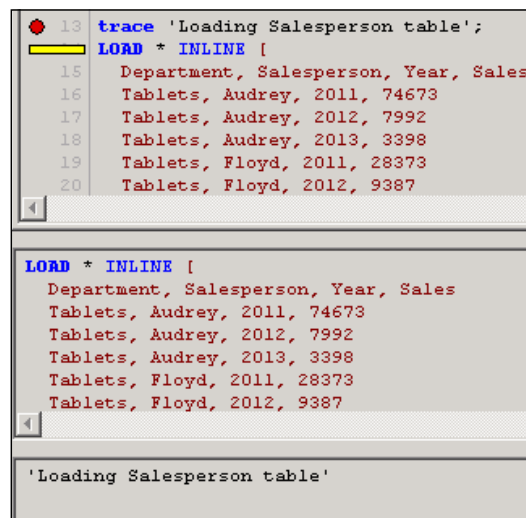


You can place a custom text that is displayed in the Script Execution dialog and appears in the logfile using the `TRACE` command. Place your custom text (which can also be a variable) to flag the area of the script that you want to examine upon script runtime and also in the logfile.

The format is:

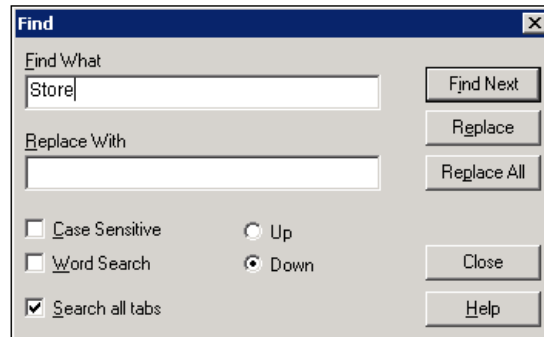
```
TRACE 'string';
```

Here's a screenshot of the **Debugger** window with a custom trace message. Note that we could have used a custom message from a variable we set, instead of this string text.



## Using find/replace in debugging

Though not a debugging tool, the find/replace across tabs is a useful tool to mention. In the Script Editor window, using the find feature by navigating to **Edit | Find/Replace** in the command menu or pressing *Ctrl + F* on the keyboard allows you to search (and/or replace text) across all the script tabs by selecting the **Search all Tabs** check box.



## Removing or partially loading data

There is some value in removing data that is loaded into the QlikView application. You can do this in a number of ways, including opening the **Debugger** window and immediately closing it and then choosing to select old data. No data will be loaded to the QlikView application. You can also open the QlikView application with the recent documents (on the QlikView start page) list and right-click and choose to load without any data. Lastly, you can open a loaded document and choose **File | Reduce Data | Remove All Values**.

Alternatively, you could choose a partial load of one or more tables (either adding a new table with data or replacing an existing table with data). The syntax for the partial load is to add either **Add** or **Replace** just before the **Load** statement, then use the **File | Partial Load** menu command or the *Ctrl + Shift + R* keyboard command to reload only those tables marked with **Add Load** or **Replace Load**. Only is an optional qualifier to **Add** and **Replace**, instructing QlikView that the statement should be ignored during normal (nonpartial) reloads, but during a partial reload it will be run and ignore other **Load** statements. Some examples:

```
Table1:
LOAD Product, ID FROM Product.csv;
ADD LOAD Product, ID FROM NewProduct.csv;
```

During a normal reload, data is loaded from `Product.csv` and stored in the QlikView table, `Table1`. Data from `NewProduct.csv` is then concatenated to `Table1`.

During a partial reload, data is loaded from `NewProduct.csv` and added to the QlikView table, `Table1`. No check for duplicates is made because a `where not exists` statement is not included.

```
Table2:
SELECT Product, ID FROM Product.csv;
ADD LOAD Product, ID FROM NewProduct.csv where not exists(Product);
```

A check for duplicates is made by means of seeing if `Product` exists in the previously loaded table data (using the `where not exists` clause).

During a normal reload, data is loaded from `Product.csv` and stored in the QlikView table, `Table2`. Data from `NewProduct.csv` is then concatenated to `Table2`.

During a partial reload, data is loaded from `NewProduct.csv`, which is appended to the QlikView table, `Table2`.

```
Table3:
LOAD Product, ID FROM Product.csv;
ADD ONLY LOAD Product, ID FROM NewProduct.csv Where not
exists(Product);
```

A check for duplicates is made by means of seeing if `Product` exists in the previously loaded table data (using the `where not exists` clause).

During a normal reload, data is loaded from `Product.csv` and stored in the QlikView table, `Table3`. The statement (`Add Only`) loading `NewProduct.csv` is disregarded.

During a partial reload, data is loaded from `NewProduct.csv`, which is appended to the QlikView table, `Table3`.

## Summary

By the end of this chapter, you should understand how to use the QlikView tools and functions to identify the problem areas and bugs in your QlikView script. In the next chapter, we will review some helpful layout tips for the user interface of your QlikView application.



# 9

## Layout Tips for Developers

This chapter describes basic QlikView object layout tips and designs, themes, dimension limits, containers, calculated colors, as well as a couple of useful advanced object usage settings (alternate states and set analysis).

### Your KPI story

It may not be the QlikView developer's job to interview business users of a proposed application or solution, but it's helpful to know how dimensions and measures are derived from the business side. Your QlikView efforts will develop an application that takes the results of those business user interviews and delivers actionable intelligence or allows business discovery for a particular issue or question.

There are two main sets of questions that will help reveal what the real **key performance indicators (KPIs)** are. These KPIs are what's really important to track – the KPI story. The rest of the data is supporting data. The questions that usually reveal KPIs are:

- What business issues concern you? What answers do you need? How often will you need them? This set of questions will help you understand what the KPIs are and how often they should be reported. This will also help you understand what other supporting information is needed and how often, as well as any target value you will need to set the KPIs. For instance, if a sales executive wants to know if his or her team is on track to make the sales quota, he or she might need to have targets for the week, month, quarter, and year.
- When you get the answer of what concerns them, what other questions will you have? What actions will you take based on this answer? Again, this diving deep into the questions will help you understand the supporting issues for the KPI, and help you understand what actions are taken at what target values.

Your QlikView applications should track KPIs that are useful to an organization. The old adage instructs us that everything of value should be measured. While these measurements (or metrics) can be valuable, they are not, in themselves, KPIs.

In order to be useful to business users, KPIs should be important, show the past, present, and future, and also contain a target (or planned) value. For instance, an online retailer may track the number of visitors to a website, but it is really sales that should be tracked. Past sales, current sales, and a trend showing where the sales are going all are included in a sales KPI. While site visitors are important, and add color and supporting arguments to the KPIs, site visitor values by themselves are not KPIs.

Dashboards and reports contain metrics (less often, KPIs); metrics are numeric values (sales amounts, number of people, and number of orders), examples may be sales orders (say dollars) by a time dimension (quarters, years). Time dimensions come with hierarchies (year, quarter, month, day, for instance), and when speaking of time and other dimensions, one refers to the granularity of a metric (is it by week, month or year? is it by country, region, or city?). In QlikView, we can define that drill-down hierarchy in the chart itself by forming a drill-down group.

Regarding the time dimensions, one of the most powerful features of QlikView is the ability to create multidimensional charts, such as sales by region by time period. It's natural for QlikView applications to have this feature, due to the associative nature of QlikView, as a timestamp is usually associated with any sales.

KPIs are values showing how well a metric is performing compared to a target. The KPI is often in the form of a ratio, index, or percentage. Are we 30 percent above target for this year? What about for the quarter, month, week, and day? KPIs imply that there are defined target numbers across all grains of time or other dimensions.

QlikView markets itself as a business discovery tool (formerly called a decision support tool), but often is used for interactive reporting purposes. This form of QlikView application doesn't totally harness its business discovery aspects, but is also very useful to business users. The following descriptions of scorecards, dashboards, and reports may be useful in your discussions of business intelligence projects:

- **Scorecard:** This is a very focused but high-level report, usually containing a few KPIs and supporting metrics, devoted to one specific strategic goal. The scorecard may also consolidate all the KPIs and operational metrics, perhaps based on a weighted scale for each KPI or metric, to derive one total score for the scorecard. An example of a scorecard may be **employee satisfaction**, where employee survey scores and numbers for employee separation and retention are considered. The scorecard is typically used by the highest level of the business.

- **Dashboards:** Often, dashboards are less strategic and more operationally focused. Such dashboards may focus on specific areas of the business, such as call centers, sales, or human resources. Dashboards are usually larger than a typical scorecard, and contain more data than is contained in charts and tables. Operational KPIs and metrics may be contained in dashboards, so that users can act upon that data. Dashboards are used by everyone from line managers to executives.
- **Reports:** Reports are the most common business intelligence tool in use today. Reports are often static, containing charts and tables. They can be complex as well, with drill-down and linked subreports available.

## Layout consistency

Best practices in any user interface (UI) also apply to QlikView applications, and one of the most important design considerations is **consistency**. It allows users to learn how to navigate the application faster, be more efficient, and focus more on the data rather than the interface. Once consistency has been established, users can readily transfer that knowledge of application navigation and data interpretation to other applications and contexts.

Some key points of consistency are:

1. Design a user application with all areas of good user interface design in mind. Research thoroughly, get user feedback, offer layout options to users, and solicit feedback with experienced UI designers and developers.
2. Once you have a good design developed that meets the best practices of UI theory, try to use it consistently on all your QlikView applications.
3. Keep the layout for all the objects the same on all tabs of a QlikView application, with similar alignment and arrangement on the sheet and in charts, with the same sheet dimensions.
4. Maintain consistent color in sheets and chart objects. Assign meanings to colors rather than arbitrarily assigning colors to chart elements. For example, red could mean losses, and blue could mean gains (be careful of using red and green in the same chart or sheet, for red-green color blindness is common). Also, colors could be lighter or darker based on how old or recent the underlying data is.
5. Use the same font colors and sizes in similar areas of the application.
6. In charts, keep the title bar in the same format and look, and keep the legend or axis title in the same area on all charts, if possible.



7. Align text in tables in the same way (right or left justified).
8. Maintain the same sort order in all chart objects, if possible (lowest to highest, highest to lowest, axis sort, and sort by values or text).

## Layout best practices

If you're a developer, the odds are that you are more concerned with a working, functional application than the application's look and feel. But knowing some key points about user interface layout will make your product more usable.

Some key points in layout best practices, in no particular order, are:

- Avoid crowded objects and charts. Some UI experts suggest no more than 6 to 8 charts for a QlikView desktop application (per sheet), and no more than 4 charts per sheet for a mobile- or tablet-formatted application.
- Colors are a big source of design mistakes. Don't use red and green on the same chart or sheet, to avoid the red-green color blindness issue. This is a big topic, and usability authors like *Stephen Few* have some great suggestions on the topic of color. Some of his tips are to avoid darker colors in bar charts, but use the darker colors in thin lines and scatter charts (the greater volume of ink, the less dark colors are needed). Avoid using too many colors, and avoid randomly assigned colors. Colors should have meaning. Use the **Persistent Colors** feature in the chart's **Colors** tab, in the **Chart Properties** dialog. This will ensure dimensions retain the same specific color in the charts and in the application.
- Choose the best chart for the job. For example, use line charts or bar charts for comparing a dimension over time, and use stacked bar charts and pie charts for comparing parts to a whole. Study excerpts written by *Stephen Few* for chart type tips and standards. Note that pie charts (and perhaps gauge charts) should be used sparingly, and keep data as the foremost concern over style. This is another large topic best researched elsewhere. Spend time on learning about data visualization techniques.
- Some experts advise not to use more than 10 sheets in your QlikView application, and less in mobile applications.
- Consider using a **Getting Started** tab to help users learn to navigate, link to other tabs or QlikView applications, or link to QlikView help.
- Use descriptive names for sheets/tabs and objects.
- Understand the screen resolution needed for end users, for desktop, and for mobile.

- Selected fields should always be visible, and the most selected fields should be located at the top of the sheet.
- When the charts are finished or you have a draft version, uncheck **Allow Move/Size** in the **Layout** tab of the **Chart Properties** dialog to limit inadvertent movement of charts in the application.

## Optimizing your QlikView application user interface

The design of a QlikView application can have significant impacts on performance, even when comparing two applications that show the same data. The following are some of the design tips recommended by QlikTech:

- The use of macros in a QlikView application can slow down performance.
- Note that all visible charts and objects on a graph take up RAM for the active sheet, and even objects not visible take up some amount of RAM.
- Limit the number of list boxes, table boxes, and pivot tables in your applications (particularly on the same sheet, but overall as well).
- Limit the number of records displayed in a table by imposing a calculation condition, if possible. This will narrow down the number of records returned and reduce the calculation time.
- Use charts instead of text objects when you want to display the result of complex expression calculations.
- The `date (Now)` function requires much more calculation overhead (it calculates every second of the day) than the data function `today ()`. Use `today ()` if you need to calculate the current date.
- Repeated use of large or complex expressions within a chart can cause performance problems.
- Limit the number of distinct text values in a field if you can. For example, break up telephone numbers and addresses into separate component fields rather than using one long string. String subparts that are repeated are candidates for breaking apart, and can be used as selectable fields for user selections and filtering. For example, street address, city, state, and ZIP code can be broken down into parts rather than using the string 123 Main St., New York, NY 10001.
- Reduce the number of fields in memory by limiting use of `SQL select all (*)` in scripts. Choose the fields you need rather than all of them. Also, drop tables in the script when they are not needed.

- Consider using a document reduction strategy (such as loop and reduce / loop and distribute) if you can distribute QlikView applications that are subsets of the entire application. This is an option if you want to send out QV applications pertinent to specific departments, regions, or job roles.

## Using themes for a consistent look

QlikView developers will often re-use a QlikView application on subsequent projects, deleting the objects they do not want, and modifying or copying objects they want to include in the new application. This approach is perfectly valid, but the use of QlikView themes is useful in enforcing formatting properties and corporate styles in applications, and saves time in application development.

QlikView themes can be created and edited for the document, sheet, or objects. The theme itself is stored in an XML file (with a .qvt extension), such as a Windows Application Data (AppData) folder for the active user, as shown in the following path (Windows 7). Note that the AppData folder may be a hidden folder and may need to be revealed in Windows Explorer, in **Organize | Folders and Search Options | View | Files and Folders | Show hidden files....**

```
C:\Users\<user>\AppData\Roaming\QlikTech\QlikView\Themes
```

A theme is used to store or apply layout properties for the document, sheet, and objects. Each of these comprises a section in the template file. For each of these sections in the file, there are subsections that describe object properties for specific object types, a subsection on border and caption properties, and a subsection on printer settings (where applicable).

The theme file is generated by **Theme Maker Wizard**, which is accessible by clicking on the **Theme Maker** button in **Document Properties | Layout and Sheet Properties | General** and **Chart Properties | Layout**. The **Theme Maker Wizard** allows you to create a new theme or modify an existing theme.

## Creating a new QlikView theme

To create a new theme, you must first format the document, or the sheet, or an object to fit your expectations and standards. When you are satisfied, you can access **Theme Maker Wizard** from any of the previously described locations using the **Theme Maker** button. Perform the following steps to create a new theme:

1. When the **Theme Maker Wizard** screen opens, click on **Next**.

2. In the **Step 1 - Select theme file** screen, select **New Theme**, leave the **Template** name as **<None>** and click on **Next**. Enter a template filename and click on **Save**. Note that if you are adding to (or editing) an existing template, you should choose **Modify Existing Theme**.
3. Select the source from the drop-down list in the **Step 2 - Source selection** screen. The source defaults according to where you opened **Theme Maker Wizard** (current object, sheet, or document). You can change the source as desired. Select one or more sources if the properties you wish to import to the template are specific for the chosen object type, and if the template should import the caption border and print settings (if applicable, these will likely not apply to the sheet object or document). Click on **Next**.
4. In the **Step 3a - Object type specific properties** screen, select the object type properties you want to include in the template. Remember that these properties are imported from the source object(s) chosen in the preceding screen. Default selected choices are usually appropriate, but you can select other properties in addition. Note that you can set as many or as few properties as you wish. In fact, creating task-specific templates can be useful (for example, setting up a template to set the title bar properties).
5. Review or change the properties matrix in **Step 4**. When creating a new template, this screen is less important, but when editing a template this screen will determine the overwriting of existing template properties.
6. In **Step 5**, you can save the theme by clicking on **Finish**. You can also set this template as the default for the current QlikView document, or set this template to be used as the default for all new documents.
7. If desired, repeat steps 1 to 6 above for each chart type, sheet, and document, but this time select **Modify Existing Theme** in the **Step 1 - Select theme file** screen to add to the theme created in this screen.

## Applying themes

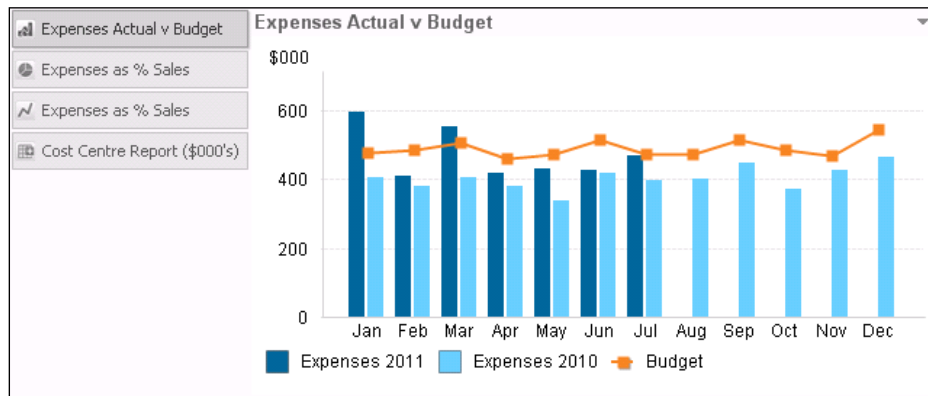
You can set up templates to be the default theme, or you can apply themes manually. Themes can be applied to objects, sheets, or the document, by accessing the appropriate **Apply Theme** button from the object, sheet, or document properties dialog box. You can also apply the theme to multiple objects by using the lasso tool by clicking and dragging around objects. To apply a theme, follow these steps:

1. If you want to apply a theme to a sheet or object, click on the sheet or object (or use the lasso tool) and access the **Properties** dialog box for that object. If the template should be applied to the whole document, access the **Document Properties** dialog box.

2. Click on the **Apply Theme** button. In the document and object properties dialog, the **Apply Theme** button is in the **Layout** tab. In the **Sheet Properties** dialog, the **Apply Theme** button is in the **General** tab.
3. Select a theme from the stored themes and click on **Open**. The theme is applied to the active selection (object, sheet, or document).

## Containers

The container object is a useful sheet object that allows developers to add multiple sheet objects of any kind. The container is a way to save screen space by dedicating just one area that can show multiple charts or tables in the same space. The charts and objects are available for selection in the opened container by clicking on buttons that are placed in the container automatically when you add a chart/object to the container.



To add a container object with multiple sheet objects, follow these steps:

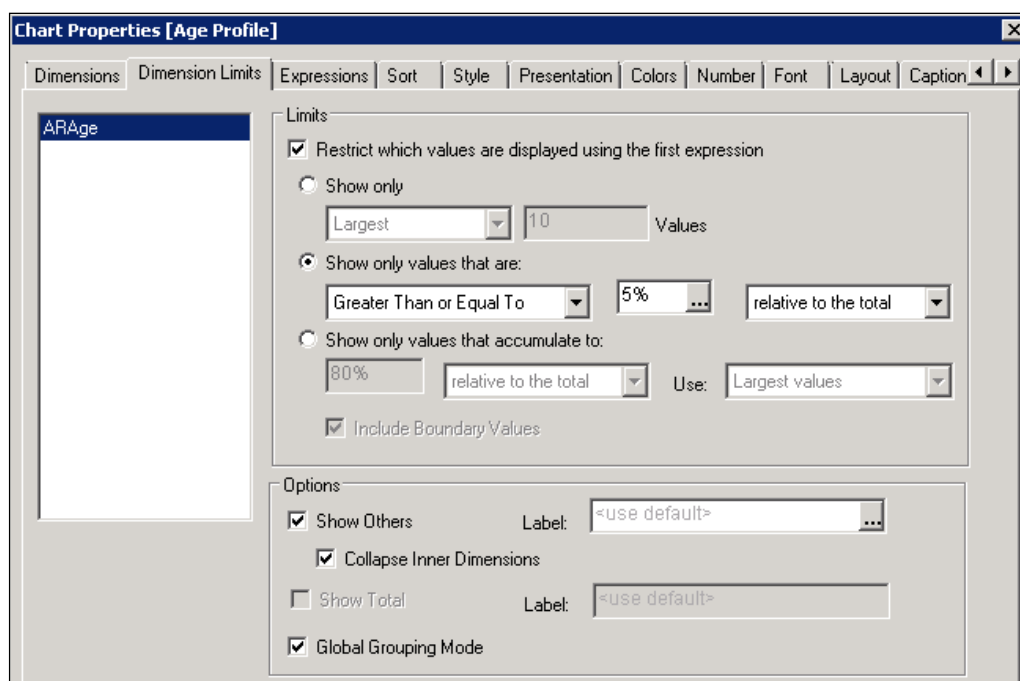
1. In your QlikView application, decide which objects you would like in the container. These charts or other objects should already be constructed. Open the object properties, and in the **General** tab note the object ID.
2. On the sheet, right-click to access the context menu, and select **New Sheet Object | Container**.
3. Find the object IDs for the desired charts and objects you want to have in the container, in the **Existing Objects** pane, and add them to the **Objects Displayed in Container** pane.

4. In the **Presentation** tab, choose where the buttons/tabs for the container should display. Choices are top, left, bottom, right, and drop-down list at the top. Click on **OK** when finished with other formatting, including setting the font and active caption background in the **Caption** tab.
5. In the **Caption** tab, select the checkbox for **Allow Minimize**. This will allow users to minimize the container to free up the screen space for more objects. See the following illustration for a depiction of a minimized container (in this case, a custom color has been defined for this object).



## Dimension limits

Dimension limits are a feature that can be applied to any chart object, except the pivot table and the gauge chart. As a developer, limiting the dimensions that appear in the chart is helpful for allowing users to focus more on meaningful data. Setting dimension limits can be done on the corresponding chart properties dialog box (the **Dimension Limits** tab), as seen in the following screenshot:



Follow these steps to set up dimension limits:

1. Enable dimension limits by selecting the **Restrict which values are displayed using the first expression** checkbox.
2. Determine which show option is desired:
  - **Show only:** Select **First**, **Largest**, or **Smallest** from this dropdown, and set the number of **Values** you wish to show in the chart. Note that **First** will display the data as set in the **Sort** tab of the chart properties dialog. **Largest** and **Smallest** return data in descending (largest to smallest) and ascending (smallest to largest) order based on the expression.
  - **Show only values that are:** Select **Greater Than or Equal To**, **Less Than or Equal To**, **Greater Than**, or **Less Than** in the drop-down list. This filter applies to the next field, which allows you to set a value (either a percentage, exact value, or calculated value). This value can be compared as relative to the total or an exact amount. For example, you could set up the dimension limit to show as **greater than or equal to 5% relative to the total** for representing the top 20 percent of the dimension. All other dimensions can be included in the **Others** data field if the **Show Others** checkbox in **Options** is selected. Note that if the **Show Others** checkbox is selected, it will decrease the number of named dimensions by 1 (so if you set 10 dimensions to show, only 9 dimensions would display, plus one labeled as **Others**).
  - **Show only values that accumulate to:** Choose the value percentage or actual value, in exact amounts or relative to the total, of the first (as defined in the **Sort** tab), smallest, or largest values returned by the dimension. This is useful if you want to display the top percent of sales reps, for example.
  - The **Global Grouping Mode** and **Collapse Inner Dimensions** options only apply if you have subsequent or inner dimensions, and determine whether you want the restrictions set in this tab to apply to the selected dimension, or if you do not want the inner dimension to display.

## Color alerts and calculated colors

Color alerts are also referred to as calculated colors, and they are useful for business users. Color alerts can be coded in the chart (or other) object and are commonly used to change the fonts or background colors of table cells, lines, and bars in charts. Calculated colors can also be used for navigation, to show users what the current tab is:

```
if(GetActiveSheetId()='Document\SH01',$vColor1,$vColor2))
```

The preceding code sets colors as follows: if the sheet (tab) is named SH01, the color of the text or object is the color as set in the `vColor1` variable, and if the active sheet is not SH01, the text or object is the color as set in the variable `vColor2`.

```
if(GetActiveSheetId()='Document\SH01',Color (1), Color (2))
```

The preceding code does the same as the previous one, but sets the colors to either color 1 or color 2 on the document's color palette (as seen in the **Colors** tab in the chart object, for instance). Calculated colors can be useful in identifying top-selling products, above or below average sales, or other metrics.

There are two main ways to set up color alerts: in the **Expressions** tab of the chart dialog box, or in the **Visual Cues** tab in the straight table or pivot table properties dialog box.

Let's try adding a color alert a couple of different ways, by following these steps:

1. Load the following data into your script, save it, and reload the script.

```
CaseCount:
LOAD * INLINE[
    Specialty, Cases
    Thoracic, 400
    Orthopedic, 575
    Endoscopy, 945
    Trauma, 1132
    Neurological, 95
    Sports, 1256
    Gastroenterology, 222
    Cystoscopy, 448
];
```

2. Create a new bar chart by right-clicking on the context menu: **New Sheet Object | Chart**, then selecting the bar chart icon in the properties dialog.



3. In the **Dimensions** tab, add **Specialty** to the **Used Dimensions** section. In the **Expressions** tab, add **Cases** as an expression, then in the **Definition** field, enter `sum(Cases)`. Expand the **Cases** expression by clicking on the plus sign, and click on **Background Color**. In the **Definition** field, enter:

```
=If (sum(Cases) >= avg(distinct total Cases), Color(1), Color(2))
```

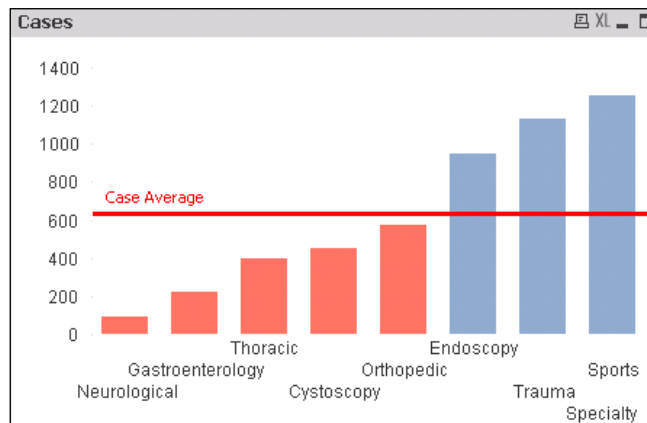
Or:

```
=If (Cases < 600, Color(1), Color(2))
```

The first line of code tells QlikView to evaluate if the sum of cases is greater than or equal to the total of all cases (distinct here is optional). If it is, choose the first color, **Color(1)**, from the color palette in the **Colors** tab in this object's properties dialog box. If it is not greater than or equal to the average number of all cases, color the bars as the second color, **Color(2)**, in the **Colors** tab. We could have also used a variable we set up for the color or the exact RGB values.

Another option, the second line of code, gives a fixed value for the average we want to set as a benchmark (600 cases). If the sum of cases is below 600, use the first color in the **Colors** tab, and if it is larger, choose the second color in the **Colors** tab.

4. In the **Sort** tab, set the data to sort by **Y-value, Ascending**.
5. For this example, let's set a reference line that shows the average number of cases performed per specialty. In the **Presentation** tab, click on **Add** in the **Reference Lines** area. In the **Expression** field, enter `Avg (Cases)`, set the line formatting to 2 point, set it to the color red, enter a label name such as **Case Average** and show the label in the chart, and click on **OK**. Click on **Apply** and then on **OK** once again to show your new chart.



- The work we've done here will also be applied if you transform the chart to a straight table. In this chart's object properties, in the **General** tab choose **Straight Table**. In the **Sort** tab, select to sort by **Numeric Value, Ascending**. If you do not want the totals row to show in the table, go to the **Expressions** tab in the **Total Mode** area and select **No Totals**. Click on **Apply** and then on **OK**.

Cases	
Specialty	Cases
Neurological	95
Gastroenterology	222
Thoracic	400
Cystoscopy	448
Orthopedic	575
Endoscopy	945
Trauma	1132
Sports	1256

## Using visual cues

Visual cues are a handy (and perhaps simpler) way to create visual color alerts in tables. To create a visual cue, follow these steps:

- Keeping *Ctrl* pressed, click-and-drag the chart you just made to another area of the sheet. This will duplicate the object (and is a very useful trick). In the **Dimensions** tab, delete the expression we used to set the background color.
- In the **Upper** field of the **Visual Cues** tab, enter `avg(total Cases)`. Set the text to black and background to light blue.
- In the **Lower** field, set the text to black and the background as light red. Check the **Bold** box to change the font to bold.
- In the **Presentation** tab, set the totals to appear on the last row, and in the **Expressions** tab, set the totals to appear as **Expression Total** on the last row. Select the **Use Label** checkbox and enter `Total Cases`. Click on **Apply** and then on **OK** to view your chart. It appears very much like the previous chart, except for the font and totals row.

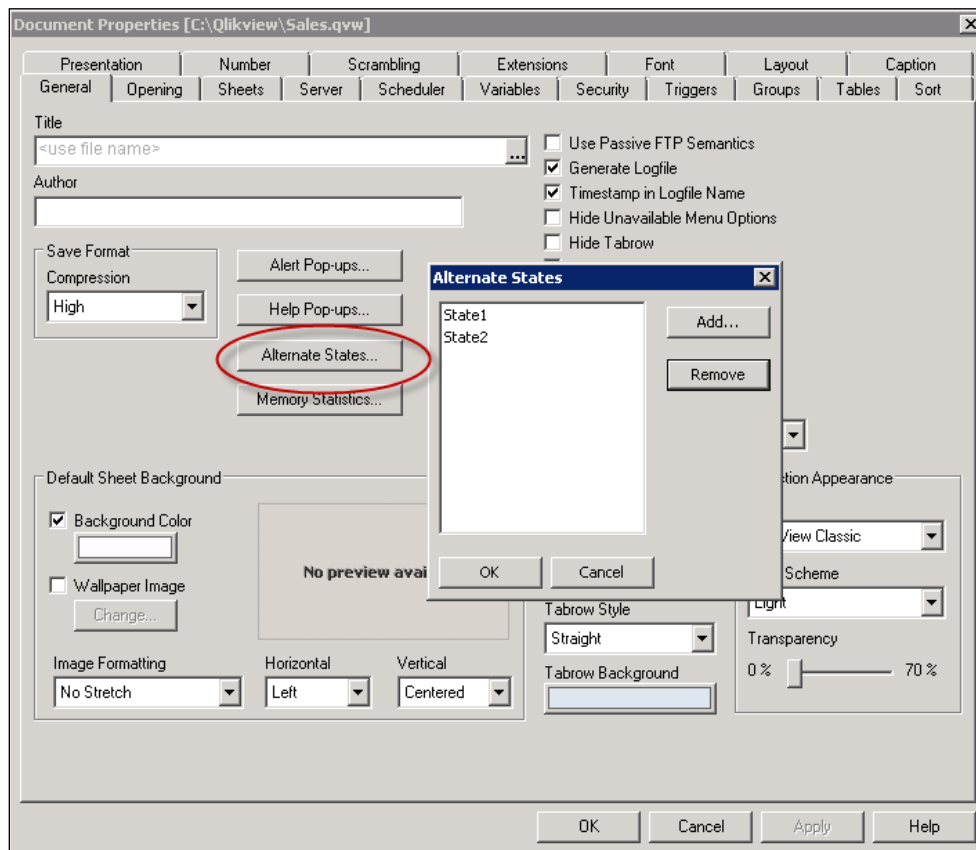
Cases	
Specialty	Cases
Neurological	95
Gastroenterology	222
Thoracic	400
Cystoscopy	448
Orthopedic	575
Endoscopy	945
Trauma	1132
Sports	1256
<b>Total Cases</b>	<b>5073</b>

## Alternate states

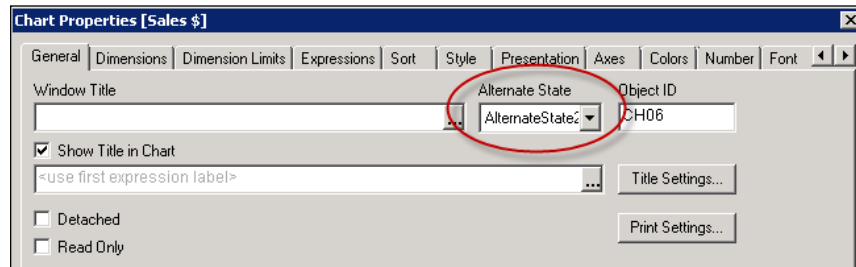
Alternate states can help business users compare two different dimensions (whether they are related or unrelated). Analysis like this can be quite useful, particularly when combined with set analysis (described in the next section), because state settings can be used in an expression in conjunction with a set modifier. Alternate states refers to setting a state of a sheet or object, and sheets and objects with that assigned state will respond to each other. Objects of different assigned states will be disconnected from one another.

Follow these steps to set up alternate states:

1. Enable the alternate states feature for the application in the **Document Properties** dialog, via the **Alternate States** button in the **General** tab. Click on the **Add** button to add the name of an alternate state, such as **AlternateState1**, **State1**, or any description, as shown in the following screenshot:



- Now that alternate states have been added to the document properties, the alternate states feature is activated. This adds a new **Alternate State** drop-down list to the object properties tab, as shown in the following screenshot:



- You can use alternate states to assign an alternate state to a chart and listbox. Any objects assigned to the same alternate state will be linked and associated. This is usually used with a listbox that controls the function of a chart—in this case, both the listbox and the chart will have the same assigned alternate state name.
- You may wish to have two identical listboxes and two identical charts, with each listbox and chart pairing having its own alternate state assignment. This allows the business users the ability to dynamically compare two separate dimensions (two different products, sales persons, regions) against some measure.

Key points to remember about alternate states:

- Alternate state sheets and objects are disconnected from other selections that are made elsewhere, unless those selections were made in an object with the same alternate state assignment.
- This disconnection from the entire document does limit alternate state functionality. It is more powerful when combined with set analysis.
- The document, and any unassigned sheets and objects, always have an alternate state assignment of default.
- Objects can also inherit an alternate state from a parent object. For a sheet object, this can be the alternate state setting of a container, or from a sheet.

## Set analysis

Set analysis is an extremely powerful feature of QlikView, and it's important that QlikView developers have some exposure to it, and eventually a thorough understanding of set analysis. This section serves as an overview of set analysis and a suggestion to explore this feature in greater detail.

Set analysis allows users to compare different sets of data (comparison analysis), without the use of complicated scripting or creating very complex expressions in the chart objects. The data returned by a set analysis expression can be independent of any user selections, and opens up the ability to select all data loaded in QlikView, regardless of current selections. The set is defined by the developer, and may be a bookmark or function of a user selection (such as the inverse of current selections, full dataset, and previous set). Some examples of this in use are:

- Products purchased this year versus last year
- Sales of a selected sales representative versus the top sales representative

## Set analysis syntax

Set expressions are always contained in curly brackets:

```
{ }
```

A set expression can contain the following elements:

- Set identifier
- Set operator
- Set modifier

## Set identifier

The basic element in a set analysis statement is a set identifier. The set identifier is the state of the dataset, and is the first element in the set statement (after the curly brackets). The syntax usage is as follows, for example, the sum of sales:

```
sum( {set identifier} Sales)
```

There are four common set identifier options:

- `{ $ }`: This option designates the current selection set (the default set). An example that shows the sum of sales for the current selection:

```
sum({ $ }Sales)
```

- `{1}`: This option indicates the full set, ignores the current selection, and considers all data. An example that shows the sum of sales for all data:  
`Sum({1} Sales)`
- `{AnyBookMark}`: This option indicates a bookmark (named anything) selection set. An example that shows the sum of sales for the selections saved by the bookmark named `2006Sales`:  
`Sum({2006Sales} Sales)`
- `{$1}`: This option designates the previous selection set. Note that the expression `{$0}` is the current selection and `{$_1}` is the forward selection (if a user clicked on the back button). An example that shows the sum of sales for the previous selections:  
`Sum({$1} Sales)`

## Set operators

Set operators perform functions on the selected selection and any set identifier. Operators are evaluated from left to right. In the absence of any standard brackets to control how the expression is evaluated, intersection and symmetric are evaluated first, followed by union and exclusion. Note that QlikTech advises that the use of operators used with basic aggregation functions (such as sum) on data fields from different tables may have unpredictable results and should be avoided.

There are four common set operators:

- The `+` operator indicates the union between two sets (returns all data from both sets), such as an example of the sum of sales where the sales are in the union of the current selection (indicated by `$`) and the bookmark called `AnyBookMark`:  
`Sum({$ + AnyBookMark} Sales)`
- The `-` operator indicates the exclusion of the selection, such as an example where the sum of sales is calculated for everything (indicated by `1`) except for the current selection (indicated by `$`):  
`sum({1 - $}Sales)`
- The `*` operator indicates the intersection between the two sets (returns data in common to both sets), such as an example where the sum of sales is calculated for any data in common between the current selection (indicated by `$`) and a bookmark named `AnyBookMark`:  
`Sum( {$ * AnyBookMark} Sales)`

- The / operator indicates a symmetric difference between the two sets (returns data from either set but not data in common between the two sets), such as an example where the sum of sales is calculated for any data contained (but not shared) in the current selection (indicated by \$) and a bookmark named AnyBookMark:

```
Sum ({$ / AnyBookMark} Sales)
```

## Set modifiers

Set modifiers are always enclosed by angle brackets, <>. Set modifiers modify the existing selection, and are equivalent to a WHERE clause in SQL, which serves to isolate or narrow down a dataset; using a set modifier is like making a selection in QlikView.

The best way to illustrate the concept of set modifiers is by working through examples. Note that in set modifiers, you can also use search syntax such as >, <, and >=. Some examples of set modifiers are:

- `sum({$ < Year={2012} >} Sales)`: The sum of sales for the current selections for the year 2012.
- `sum( { AlternateState1 < Year= {2012} >} Sales)`: The sum of sales for the year 2012 and the selections made in listboxes designated as an alternate state named AlternateState1.
- `sum( { $ < Year = Year -{2011} > } Sales)`: The sum of the sales for the selected year(s) (noted by \$), but excluding 2011.
- `sum( { $ < Product = {"*"} > } Sales)`: The sum of the sales for all products (wildcard character denoting all) for the selections.
- `sum( { $ <Product = > } Sales)`: The sum of sales for the selected fields, but ignoring the Product field selection.
- `sum( { $ < Product = {}> } Sales)`: The sum of sales for the selected fields, but for anything not associated with the selection in the Product field.
- `sum( {$< Year = {$ (= $(vYear))}> } Sales)`: Using the dollar sign expansion for the vYear variable, this is the sum of sales for the year specified in the variable. An example of this is if the vYear variable was set to the expression = Year(Today()).
- `sum( { 1<Year = {"20*", 1999} - {2001}> } Sales)`: The sum of sales for all years in the full dataset (1) for the years starting with 20 (wildcard \*), and 1999, but excluding 2001 (minus exclusion operator before 2001).

- `sum( { $< Product = {"*laptop*"}> } Sales)`: The sum of sales for all selections for products containing the string `laptop` (by using the wildcard `*` around the string `laptop`).
- `sum( { $< Year = Year + {2012, 2013}> } Sales)`: The sum of sales for the selected year, plus the years 2012 and 2013 (this is a union operation).
- `sum( { $< Year={2011, 2012}, Zone={'Europe'}> } Sales)`: The sum of sales for the years 2011 and 2012, in the Europe zone.
- `sum({$<Customer = (P({1<ProductCategory = {'Instruments'}>}))>}Sales)`: The `P` element here indicates all possible data. The other option for an element is `E`, which is excluding and the opposite effect of possible. This is the sum of sales for all customers who ordered any products of the instrument product category.

## Summary

In this chapter, we have seen how good user interface design can lead to useful applications for business users. Several features of creating and coding chart objects, as well as powerful features such as alternate states and set analysis were discussed.





# Index

## Symbols

{1} option 113  
{1\$} option 113  
{2\$} option 112  
{AnyBookMark} option 113  
- operator 113  
\* operator 113  
/ operator 114  
+ operator 113  
.qvd 77

## A

Add Tab command 15  
Administrative Tools window 6  
Alias statement 43  
alternate states  
  about 110  
  key points 111  
  setting up 110, 111  
Apply Theme button 104  
AS specifier 42  
Auto-Layout menu command 52, 57  
Autonumber statement 59

## C

calculated colors. *See* color alerts  
Capitalize command 13  
centipede schema. *See* snowflake schema  
circular references  
  about 63, 64  
  fixing 64, 65  
Clear Entire Script 13  
color alerts  
  about 107

  adding, ways 107, 108  
  visual cues, using 109

## columns

  manipulating 29, 30

Comment command 13

Connect Statement command 14, 40

Connect to Data Source dialog 7

container object 104

Copy Image menu command 57

cross tables 32

## D

dashboards 99

## data

  loading, partially 94

  removing 94, 95

  unwrapping, from repeating columns 28,  
  29

## database

  connecting to, from QlikView 7

  supported 8

data island 72

## data model

  about 9

  best practices 56

Data tab 16

data transformation 23

date function 82, 83

Debug button 87

debugger. *See* QlikView debugger

Demote command 15

dimensional modeling 10

## dimension limits

  about 105

  setting up 106

**Disconnect Statement command** 14  
**Domain SID command** 14

## **E**

**Easyqlik**  
    URL 84  
**Edit menu**  
    about 13  
    Capitalize command 13  
    Clear Entire Script 13  
    Comment command 13  
    Invert Case command 13  
    Lower Case command 13  
    Uncomment command 13  
    Upper Case command 13  
**Editor Preferences command** 15  
**Edit Script dialog.** *See* Script Editor  
**Environment Variables command** 14  
**Exit Script function**  
    using 89  
**Export Image menu command** 57  
**Export Structure menu command** 57

## **F**

**field comments** 60  
**fields**  
    about 48  
    Alias statement 43  
    AS specifier 42  
    delinking 42  
    linking 42  
    Qualify statements 44, 45  
    Rename Field statement 43, 44  
    renaming, mapping tables used 67  
**field tags** 60  
**File menu**  
    about 12  
    Open External Script File command 12  
    Reload command 12  
    Save Entire Document As... command 12  
    Save Entire Document command 12  
    Save External Script File command 12  
    Table Viewer 13  
**File Wizard: Transform dialogs.** *See*  
    Transformation Wizard  
**Fill function**

    using 27  
**find/replace**  
    used, for debugging 94  
**flat files** 23  
**forced concatenation** 70  
**full loads**  
    of QVDs 79  
**Functions tab** 16

## **G**

**garbage**  
    removing, from data files 24-26  
**generic tables** 33, 34

## **H**

**Hash128 function** 59  
**Help menu** 16  
**Hidden Script** 11  
**hierarchies** 35-37

## **I**

**images**  
    copying 57  
    exporting 57  
**Include Statement command** 14  
**incremental loads**  
    of QVDs 80  
**information density** 60  
**Insert menu**  
    about 13  
    Connect Statement command 14  
    Disconnect Statement command 14  
    Domain SID command 14  
    Environment Variables command 14  
    Include Statement command 14  
    Load Statement command 14  
    Script Files command 14  
    Section Access command 14  
    Set Statement command 13  
    Test Script command 14  
**Insert Tab at Cursor command** 15  
**internal tables**  
    viewing 57  
**IntervalMatch function** 70, 71  
**Invert Case command** 13

## K

key performance indicators (KPIs) 97, 98  
key table 64

## L

layout  
    best practices 100, 101  
    consistency 99, 100  
link table. *See* key table  
Load statement 41, 71, 84  
Load Statement command 14  
logfiles  
    used, for debugging 92, 93  
loosely coupled table 63  
Lower Case command 13

## M

mapping tables  
    using, to rename fields 67, 68  
master calendars 50  
Merge with Previous command 15  
metadata  
    using 73, 74

## N

NoConcatenate 81  
number interpretation variables 39, 40

## O

Object Linking and Embedding Database.  
    *See* OLE DB  
ODBC  
    about 5  
    configuring 7  
ODBC Administrator 32 bit command 15  
ODBC Data Source Administrator dialog 7  
OLE DB 5, 6  
Online Analytical Processing (OLAP) 54  
Online Transactional Processing (OLTP)  
    about 54  
Open Database Connectivity. *See* ODBC  
Open External Script File command 12

## P

print menu command 57  
Promote command 15

## Q

Q-Eye  
    URL 84  
QlikMarket  
    URL 74  
QlikTech 8  
QlikView  
    database, connecting to 7  
    databases, supported 8  
    Hidden Script 11  
    script 9  
    script debugger 87, 88  
    script errors 90, 92  
    Table Viewer 51  
    user interface, optimizing 101  
QlikView Data Files. *See* QVD  
QlikView debugger  
    about 87  
    data, loading partially 94, 95  
    data, removing 94, 95  
    dialog (debugger) 87  
    Exit Script function, using 89  
    features 88  
    find/replace, using 94  
    logfiles used 92, 93  
    script errors 90-92  
    syntax checker, using 90  
    windows 89  
QlikView Governance Dashboard applica-  
    tion 74  
QlikView Publisher  
    QVDs, creating with 83  
QlikView themes. *See* themes  
QlikView Worksheet. *See* QVW  
QUALIFY statement 58  
Qualify statements 44, 46  
quotation marks 48, 49  
QVD  
    about 77  
    advantages 77, 78  
    content, viewing 84

- creating 78
- creating, with QlikView Publisher 83
- full loads 79
- incremental loads 80, 81, 83
- loading from 84
- QVDs.** *See* **QVD**
- QVDCreateDate** 82
- QVDCreateTime** 80
- QVW** 5
- QVWs**
  - no data opens, managing 85
  - reload, managing 85

## R

- records**
  - about 48
  - in tables, previewing 59, 60
- regular script statement** 47
- Relational Database Management Systems (RDBMS)** 6
- Reload command** 12
- Remove command** 15
- Rename command** 15
- Rename Field statement** 43, 44
- reports** 99
- rotate function** 30

## S

- Save Entire Document As... command** 12
- Save Entire Document command** 12
- Save External Script File command** 12
- scorecard** 98
- script**
  - about 9
  - expressions 48
  - number interpretation variables 39, 40
  - segmentation, via tabs 41
  - tabs, organizing 20
- Script commenting** 17
- script control statement**
  - about 47
  - examples 47, 48
- Script Editor, commands**
  - Edit menu 13
  - File menu 12
  - Help menu 16

- Insert menu 13, 14
- Tab menu 15
- Tools menu 15
- Tools pane 16
- script errors** 90, 91, 92
- Script Execution Progress dialog** 20
- Script Files command** 14
- scripts**
  - tabs, organizing 20
- Section Access command** 14
- Select data source wizard** 5
- select statement**
  - building 18, 19
  - creating 17
  - database, connecting to 18
  - results, viewing 20
  - script, running 20
- Set analysis**
  - set identifier 112
  - set modifiers 114, 115
  - set operators 113, 114
  - syntax 112
- set identifier, Set analysis**
  - {1} option 113
  - {\$1} option 113
  - {\$} option 112
  - {AnyBookMark} option 113
  - key points 112
- set modifiers, Set analysis**
  - about 114
  - examples 114
- set operators, Set analysis**
  - operator 113
  - \* operator 113
  - / operator 114
  - + operator 113
- Set Statement command** 13
- Settings tab** 16
- snowflake schema**
  - about 55
  - data model, example 55
- source tables**
  - viewing 57
- SQL Select statement** 41
- standard tables.** *See* **generic tables**
- star schema**
  - about 54

data model, example 54

**STORE statement 78**

**structure**

exporting 57

**subset ratio 60**

**Syntax Check command**

about 15

using 90

**synthetic keys**

about 58

eliminating 58

**synthetic table**

about 58

eliminating 58

removing 64-67

## T

**tables**

combining 69

concatenating 69

**Table Viewer**

about 13, 51-53

images, exporting 56, 57

internal tables, viewing 57

source tables, viewing 58

structures, exporting 56, 57

**Tab menu**

about 15

Add Tab command 15

Demote command 15

Insert Tab at Cursor command 15

Merge with Previous command 15

Promote command 15

Remove command 15

Rename command 15

**tabs**

in script, organizing 20

**Test Script command 14**

**themes**

applying 103

creating 102, 103

using 102

**Tools menu**

Editor Preferences command 15

ODBC Administrator 32 bit command 15

ODBC Administrator 64 bit command 15

Syntax Check command 15

**Tools pane**

Data tab 16

Functions tab 16

Settings tab 16

Variables tab 16

**Transformation Wizard**

about 24

column, manipulating 29, 30

data, unwrapping from repeating columns  
28

Fill function, using 27

garbage, removing from data files 24-26

script, results 30

tables, rotating 30, 31

## U

**Uncomment command 13**

**Upper Case command 13**

## V

**Variables tab 16**

**visual cues**

using 109

**vSaveQVD variable 78, 79**

## W

**where not exists clause 83**

**wildcard character 46**

## Z

**Zoom menu command 57**





## **Thank you for buying QlikView Scripting**

### **About Packt Publishing**

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: [www.packtpub.com](http://www.packtpub.com).

### **About Packt Enterprise**

In 2010, Packt launched two new brands, Packt Enterprise and Packt Open Source, in order to continue its focus on specialization. This book is part of the Packt Enterprise brand, home to books published on enterprise software – software created by major vendors, including (but not limited to) IBM, Microsoft and Oracle, often for use in other corporations. Its titles will offer information relevant to a range of users of this software, including administrators, developers, architects, and end users.

### **Writing for Packt**

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to [author@packtpub.com](mailto:author@packtpub.com). If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.





## QlikView 11 for Developers

ISBN: 978-1-84968-606-8

Paperback: 534 pages

Develop Business Intelligence applications with QlikView 11

1. Learn to build applications for Business Intelligence while following a practical case -- HighCloud Airlines. Each chapter develops parts of the application and it evolves throughout the book along with your own QlikView skills.
2. The code bundle for each chapter can be accessed on your local machine without having to purchase a QlikView license.



## QlikView for Developers Cookbook

ISBN: 978-1-78217-973-3

Paperback: 290 pages

Discover the strategies needed to tackle the most challenging tasks facing the QlikView developer

1. Learn beyond QlikView training
2. Discover QlikView Advanced GUI development, advanced scripting, complex data modelling issues, and much more
3. Accelerate the growth of your QlikView developer ability
4. Based on over 7 years' experience of QlikView development

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles



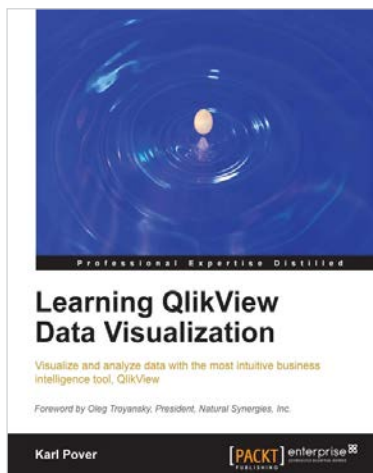
## Instant QlikView 11 Application Development

ISBN: 978-1-84968-964-9

Paperback: 60 pages

An intuitive guide to building and customizing a business intelligence application for your data

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results
2. Learn how to analyze data for business discovery with QlikView 11 with automatic data linking and wizards
3. Create your own analysis interfaces using tables, lists, and charts



## Learning QlikView Data Visualization

ISBN: 978-1-78217-989-4

Paperback: 156 pages

Visualize and analyze data with the most intuitive business intelligence tool, QlikView

1. Explore the basics of data discovery with QlikView
2. Perform rank, trend, multivariate, distribution, correlation, geographical, and what-if analysis
3. Deploy data visualization best practices for bar, line, scatterplot, heat map, tables, histogram, box plot, and geographical charts

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles