Professional Expertise Distilled

SQL Server 2012 Reporting Services Blueprints

Use real-world situations to develop real-world solutions

Marlon Ribunal Mickey Stuewe

[PACKT] PUBLISHING
enterprise professional expertise distilled

SQL Server 2012 Reporting Services Blueprints

Use real-world situations to develop real-world solutions

Marlon Ribunal

Mickey Stuewe

[PACKT] PUBLISHING enterprise professional expertise distilled

BIRMINGHAM - MUMBAI

SQL Server 2012 Reporting Services Blueprints

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 authors, 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: October 2013

Production Reference: 1181013

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

ISBN 978-1-84968-508-5

www.packtpub.com

Cover Image by Artie Ng (artherng@yahoo.com.au)

Credits

Authors
Marlon Ribunal
Mickey Stuewe

Reviewers
Basit A. Masood-Al-Farooq
Varinder Sandhu
Sivakumar Vellingiri
Valentino Vranken

Acquisition Editor
Joanne Fitzpatrick

Lead Technical Editor
Sam Birch

Technical Editor
Pratik More

Project Coordinators
Hardik Patel
Priyanka Goel

Proofreaders
Linda Morris
Bob Phillips

Indexer
Rekha Nair

Graphics
Ronak Dhruv

Production Coordinator
Aparna Bhagat

Cover Work
Aparna Bhagat

About the Authors

Marlon Ribunal is passionate about SQL Server. He loves learning about the technology. Mostly, he is a sponge—the learner—but he loves sharing the knowledge he has accumulated.

He is introvert by nature but loves connecting to people in various avenues, through technical communities' events such as the SQL Saturday, social networks, on his blog, and other forums.

His purpose is to help people in their pursuit of growth and development; and, thereby, help himself realize his full potential as a professional, husband, father, Christian, and human being.

When he's not busy, he blogs at `www.marlonribunal.com`—his technical blog.

You can follow him on Twitter as `@MarlonRibunal`.

And if you're interested in topics related to Productivity and Getting Things Done (GTD), you can follow his other blog at `www.productivitybits.com`.

Mickey Stuewe is a senior SQL database developer with close to twenty years of experience using Microsoft products. She has been working with SQL Server since 1996, and has used all versions of Reporting Services. Her focus is on query optimization, report design, and report creation using SQL Server and Reporting services. She keeps a blog at `http://mickeystuewe.com/` and frequently writes articles for `http://www.mssqltips.com/`. She is a speaker at Dev Connections and SQL in the city, and is a regular speaker at SQL Saturday events and user group meetings throughout the year. She currently lives in Southern California with her two daughters and husband.

You can follow her on Twitter: `@SQLMickey`.

Acknowledgments

This book would not have been possible without the support of, and inspiration from, my family. To Vicky, my loving and beautiful wife, thank you for the unconditional love and affection. I love you so much! To Lawrence and Angelo, my two sons, thank you for your unconditional love. I love you, guys!

I'd also like to thank my father, Mario Ribunal Sr., and my siblings here in the US and in the Philippines for their unwavering support to me and my family. And, to my mother, who will always have a special space in my heart, I hope I made you smile in heaven.

I would also like to thank the Estebat-Javier-Limos-Paredes-Garces Clan of Long Beach, California for their continued support.

I'd also like to acknowledge my high school alma mater, La Salette of Jones in Isabela, Philippines. Thank you for letting me experience the wonder of booting a computer with a floppy disk for the very first time way back when (not too long ago)!

Of course, I would like to say thanks to my SQL Family, Professional Association for SQL Server (PASS), and the SQL Server Community in general. Thank you for inspiring me. Special thanks go to *Donabel Santos* (`@sqlbelle`), *Edwin Sarmiento* (`@EdwinMSarmiento`), *Mark Tabladillo* (`@MarkTabNet`), and *Tom Roush* (`@GEEQL`).

And finally, I'd like to thank my Lord and Savior, Jesus Christ. To God Be the Glory!

Marlon Ribunal

I want to thank my husband, Dan, who supports me in all my endeavors. Without him I would not have the time or energy to work on this book or even participate in the SQL community.

Mickey Stuewe

About the Reviewers

Basit A. Masood-Al-Farooq is an internationally known Lead SQL DBA, Trainer and Technical Author, with twelve years experience of the Microsoft technology stack. He is an accomplished development and production SQL Server DBA, with a proven record of delivering major projects on time and within budget. He is expert at evaluating client's needs against the capabilities of the SQL Server product set, with the objective of minimizing cost and maximizing function through making innovative use of advance capabilities. Basit has authored numerous SQL Server technical articles on various SQL Server topics, for different SQL Server community sites that include `http://sqlmag.com/`, `www.mssqltips.com/`, `www.sqlservercentral.com/`, `www.sswug.org/`, `www.sql-server-performance.com/`, and `http://searchsqlserver.techtarget.com/`. He has also developed and implemented many successful database infrastructure, data warehouse and business intelligence projects. He also has a good understanding of ITIL principals. He holds a Master's degree in Computer Science from London Metropolitan University, and industry standard certifications from Microsoft, Sun, Cisco, Brainbench, Prosoft, and APM, including MCITP Database Administrator 2008, MCITP Database Administrator 2005, MCDBA SQL Server 2000 and MCTS .NET Framework 2.0 Web Applications.

He can be reached via twitter (`@BasitAali`), blog (`http://basitaalishan.com`), or via LinkedIn. His LinkedIn profile address is `http://uk.linkedin.com/in/basitfarooq`.

I have not worked on any other book but I regularly write technical articles on various SQL Server topics, for different SQL Server community sites such as `http://sqlmag.com/`, `www.mssqltips.com/`, `www.sqlservercentral.com/`, `www.sswug.org/`, `www.sql-server-performance.com/`, and `http://searchsqlserver.techtarget.com/`.

I would like to thank my wife Aniqa and my sons Saifaan and Rayyan for coping with my hectic work schedule.

Varinder Sandhu is an IT professional. He is a Microsoft-certified SQL Server Database Administrator and technology specialist in Microsoft .NET Framework—Application Development Foundation. He has a Master of Computer Application (MCA) degree from Sikkim Manipal University and a B Sc. in Computer Science from Guru Nanak Dev University, Amritsar. He has around 6 years of IT experience with a contribution in all aspects of the software engineering process, from architecture and requirements definition, through design, estimating, coding, testing, and maintenance of software applications.

Check out his blog at `www.varindersandhu.in`.

I would like to thank my mom and dad who always inspire me for my work.

SIVAKUMAR VELLINGIRI is a Senior Technical Lead at Prokarma Softech (P) Ltd, India. He has been a part of the IT industry for more than 10 years. During his career, he has worked primarily with SQL Server and Business Intelligence technologies. Siva has worked on many Business Intelligence, Data Warehousing, Database design, performance tuning and optimization for high transactional systems. He is a Microsoft Certified Technology Specialist and is certified in SQL Server and Business Intelligence.

Siva writes frequently on his blog `http://sivasqlbi.blogspot.in` on various subjects concerning SQL Server and Business Intelligence technologies. When he is not in front of a computer, he plays with his 10 month old son, Sachin.

A heartfelt thanks to the editorial and management team at *Packt Publishing* for giving me this wonderful opportunity.

Finally, thanks to my parents, wife Saranya and my son Sachin, who have put up with my long hours and weekend nights spent during the review of this book. I appreciate their love and encouragement and could not have done this without them.

Valentino Vranken is a Senior Microsoft BI Consultant working for Ordina, Belgium (Europe). Ever since the start of his career he has been gathering experience using various programming languages to manipulate data in SQL Server databases. Nowadays, his focus is mainly on Business Intelligence implementations, still using SQL Server.

In terms of certification, Valentino has earned two MCITP (BI and DEV) and three MCTS (BI, DEV, and DBA) certifications on SQL Server 2008, and has also passed two out of three exams to reach MCSA on SQL Server 2012.

In his free time, he is active as a Board Member of the Belgian SQL Server User Group (`www.sqlug.be`), writes articles for his blog (`http://blog.hoegaerden.be`), and helps the community through several technical forums. Many of his articles are republished at well-known specialist sites.

In 2012, he received the MVE (Most Valuable Expert) award in recognition of his outstanding community contributions at Experts Exchange.

You can follow him on Twitter: `@ValentinoV42`.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `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` 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` for more details.

At `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 web browser

Free Access for Packt account holders

If you have an account with Packt at `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` on Twitter, or the *Packt Enterprise* Facebook page.

Table of Contents

Preface 1

Chapter 1: Let's Breakdown the Numbers 7

- Defining a data source for the project 9
- Defining the dataset for the project 11
- Creating the report file 13
- Creating a parameterized report 15
- Creating a radio button parameter 20
- Making a parameter optional 21
- Populating a drop-down parameter with a dataset 24
- Creating a multivalued parameter 28
- Creating a cascading parameter 29
- Creating a cascading multivalued parameter 32
- Sales details report in action! 34
- Summary 35

Chapter 2: Making a Good Impression 37

- Report page setup 39
- Formatting expression 41
- Adding report objects 42
- Formatting literals in an expression 44
- Creating a row group 46
- Implementing Drilldown with row visibility 47
- Adding subtotals for each group 49
- Adding the grand total 50
- Adding a page header 51
- Adding an image in the header 51
- Adding a page footer 52
- Adding a page number 52
- Alternating row colors 53

Creating a report template 54
Using a custom report template 55
Summary 56
Chapter 3: Getting Some Actions 57
Creating the main report 58
Creating a detail report 61
Adding a subreport 62
Creating a drillthrough or linked report 65
The ALL parameter 70
Adding a hyperlink (go to URL) 70
Adding a tool tip 71
Interactive sorting 72
Adding a Document Map 73
Creating a bookmark 74
Summary 77
Chapter 4: Everything Looks Better with Charts 79
Creating a simple chart 80
Adding data to a chart 82
Formatting the chart 83
Adding a tool tip 85
Embedding data bars in a tablix 87
Showing a data label 89
Setting the bar color conditionally 91
Adding sparklines in the tablix 92
Aligning axes in the sparkline 95
Using gauge in a tablix 97
Configuring the pointer of a gauge 98
Adding a scale for the gauge 98
Defining a range of the gauge 99
Adding an Indicator 101
Summary 103
Chapter 5: Location, Location, Locations! 105
Creating a map (polygon) visualization 106
Options for spatial and data views 108
Options for theme and data visualization 112
Formatting the map 113
Changing the color rules 114
Adding center points 116
Customizing distribution options 116
Using a shapefile 119

Removing empty center points **121**
Adding a random point layer **123**
Adding a line layer **128**
Adding a tile layer **130**
Additional Formatting **130**
Summary **132**
Chapter 6: Let's Get Analytical! **133**
Checking the TCP port of Analysis Services **134**
Creating a new inbound rule for Analysis Services **136**
Changing the deployment setting for Analysis Services **138**
Deploying the multidimensional model **142**
Creating the data source for the report **142**
Adding a dataset based on the multidimensional model **144**
Working with the query designer **144**
Building a dataset using the query designer **147**
Using the dataset on the report designer **150**
Adding a filter in the dataset **151**
Adding a parameter in the dataset **152**
Creating a calculated member **154**
Summary **157**
Chapter 7: Sir, We're Ready to Deploy! **159**
Deployment configurations **163**
Development to production migration **165**
User security **168**
Shared schedules **170**
Cached reports **171**
Snapshots **173**
Subscriptions – e-mail delivery / scheduled delivery **174**
Subscriptions – data-driven subscription **176**
Step 1 – specify the type of data-driven subscription 176
Step 2 – select the location of the shared data source 176
Step 3 – set up the SQL statement or stored procedure that will return the data-driven property data. 176
Step 4 – set up the delivery properties 177
Step 5 – set up the parameter values 178
Step 6 – set up the schedule for the subscription 179
Summary **179**

Chapter 8: Beyond the Horizon! 181
Integrating SSRS with SharePoint 182
Creating Reporting Service Application 183
Provisioning subscriptions and alerts 186
Creating a web application 186
Creating a site collection 187
Deploying SSRS projects to SharePoint 189
Getting introduced to Power View 192
Understanding PowerPivot 193
Adding a data source to a model 193
Creating a Power View report in SharePoint 198
Summary 200
Appendix A: SSRS Best Practices 201
Location of SSRS database installation 201
Utilizing the SSRS execution logs 201
Report management 204
Standardizing queries 204
Report standards 205
Testing reports 205
Appendix B: Transactional Replication for Reporting Services 207
Options for report delivery 208
Transactional replication to the rescue 210
Implementing transactional replication 211
Setting up the distributor 212
Configuring the publication 215
Getting the subscription ready 218
Summary 221
Index 223

Preface

SQL Server Reporting Services (SSRS), as a platform, has grown to become a mature reporting solution, beginning with its SQL Server 2008 Reporting Services iteration. Actually SQL Server 2005 Reporting Services got some attention too, but the 2008 version made it a strong contender in the report solution battle which was at that time dominated by Crystal Reports.

More and more companies are implementing SSRS as part of their Business Intelligence implementation. We can only expect this trend to increase as SQL Server is continuing to increase its share of the database market.

SQL Server Integrated BI Solution–Microsoft BI Stack (SQL Server, SSIS, SSAS & SSRS)–is becoming increasingly popular in the BI market. The primary reason for this is that it's quite easy to learn the platform if you have a sufficient amount of passion and the right tools and support.

SQL Server 2012 Reporting Services Blueprints is intended to be just that, a series of Blueprints–things that work straight out-of-the-box and that mirror what a real job requires real report developers to do. You will not be bored with an "Introduction to X" or "What's under the hood of Y". We go straight to the practical development of reporting skills, explaining actions as they are taken. This means working through step-by-step tutorials instead of studying infrastructure and business use explanations.

This book is a step-by-step, task-driven tutorial. It provides enterprise reporting blueprints that address common business report requirements and scenarios. Right from the very start, you'll learn SSRS beyond the basics, giving you the skills to create the best reports for any task. Explore the possibilities, master your options, and choose the best tool for the job.

What this book covers

Chapter 1, *Let's Breakdown the Numbers*, covers the use of parameters in different scenarios. We figured that you already have a basic knowledge of how SSRS works. So, we eliminated the boring stuff. The chapter begins with showing you how to create a data source and dataset with T-SQL and stored procedures.

Chapter 2, *Making a Good Impression*, covers the structure and properties of the report. This is a critical part of the design and presentation of the report.

Chapter 3, *Getting Some Actions*, covers the addition of interactive elements to the report.

Chapter 4, *Everything Looks Better with Charts*, demonstrates the basic and advanced uses of various visualizations in a report.

Chapter 5, *Location, Location, Location!*, covers the use of basic and advance maps, also other geospatial elements in SSRS.

Chapter 6, *Let's Get Analytical!*, covers the use of OLAP cubes in a SSRS report if you are dealing with SQL Server Analytical Services (SSAS).

Chapter 7, *Sir, We're Ready to Deploy!*, gives you options for successfully deploying your SSRS projects.

Chapter 8, *Beyond the Horizon!*, shows you basic information on how to integrate SharePoint 2010/2013, PowerPivot, and Power View with SSRS.

Appendix A, *SSRS Best Practices*, gives you few tips on report management, report standards, execution logs, and so on.

Appendix B, *Transactional Replication for Reporting Services*, demonstrates the step-by-step process of setting up Transactional Replication.

What you need for this book

The authors used SQL Server 2012 SP1 Enterprise Edition on Windows Server 2012 for writing the step-by-step tutorials in this book. In order for you to accomplish all the steps discussed in this book, you need the SQL Server 2012 Enterprise or Business Intelligence Edition. You can still use the Standard Edition but it doesn't support the advanced BI features, such as PowerPivot and Power View. Here's a link to the comparison of the three editions of SQL Server 2012: `https://www.microsoft.com/en-us/sqlserver/editions.aspx/`.

Here are the tools you need to follow the step-by-step tutorials in this book:

- Windows 8 or Windows Server 2012 (check the SQL Server 2012 installation requirements)
- SQL Server 2012 Enterprise or Business Intelligence Edition
- SQL Server Data Tools (SSDT), or SSDT for Visual Studio 2010, or higher (If using Visual Studio)
- SharePoint 2013 (You can use SharePoint 2010 but the steps may look slightly different from this book)

Who this book is for

This book does not require readers to have advance knowledge of T-SQL, but the readers are expected to know how to write basic aggregates in T-SQL. SQL Server Reporting Services architecture and any pertinent information are not included in this book. Knowing how data sources and datasets work in advance will greatly help you to progress through the tutorials quickly. This book is perfect for those new to SQL Server Reporting Services. Other SQL Server users, such as database administrators, who are getting involved in report development can greatly benefit from the fast-paced style of this tutorial book.

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: "Find the location of the `Sales Commissions.rdl` file that we modified."

Any command-line input or output is written as follows:

```
=iif(Parameters!SalesPersonID.Value = -1, Fields!BusinessEntityID.Value,
Parameters!SalesPersonID.Value)
```

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: "Click on the **OK** button to save your changes."

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`, 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`.

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` 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` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Let's Breakdown the Numbers

John Kirkland is an awesome "accidental" SQL Server DBA for Red Speed Bicycle LLC—a growing bicycle startup based in the United States. The company distributes bikes, bicycle parts, and accessories to various distribution points around the world.

To say that they are performing well financially is an understatement. They are booming! They've been expanding their business to Canada, Australia, France, and the United Kingdom in the last three years.

The company has upgraded their SQL Server 2000 database recently to the latest version of SQL Server 2012. Linda, from the Finance Group, asked John if they can migrate their Microsoft Access Reports into the **SQL Server 2012 Reporting Services**.

John installed SSRS 2012 in a native mode. He decided to build the reports from the ground up so that the report development process would not interrupt the operation in the Finance Group.

There is only one caveat; John has never authored any reports in SQL Server Reporting Services (SSRS) before.

Let's give John a hand and help him build his reports from the ground up. Then, we'll see more of his SSRS adventures as we follow his journey throughout this book.

In this chapter, we will learn the following:

- Creating a report from scratch
- Using stored procedure in a report
- Defining data sources and datasets
- Using dataset as a parameter source

- Creating a parameterized report
- Creating multivalued parameters
- Creating cascading parameters
- Creating cascading multivalued parameters

Here's the first report requirement for John: a simple table that shows all the sales transactions in their database. Linda wants to see a report with the following data:

- Date
- Sales Order ID
- Category
- Subcategory
- Product Name
- Unit Price
- Quantity
- Line Total

We will build our report, and all succeeding reports in this book, using the **SQL Server Data Tools** (**SSDT**). SSDT is Visual Studio shell which is an integrated environment used to build SQL Server database objects. You can install SSDT from the SQL Server installation media.

In June 2013, Microsoft released SQL Server Data Tools-Business Intelligence (SSDTBI). SSDTBI is a component that contains templates for **SQL Server Analysis Services** (**SSAS**), **SQL Server Integration Services** (**SSIS**), and **SQL Server Reporting Services** (**SSRS**) for Visual Studio 2012.

SSDTBI replaced Business Intelligence Development Studio (BIDS) from the previous versions of SQL Server.

You have two options in creating your SSRS reports: SSDT or Visual Studio 2012. If you use Visual Studio, you have to install the SSDTBI templates.

Let's create a new solution and name it **SSRS2012Blueprints**.

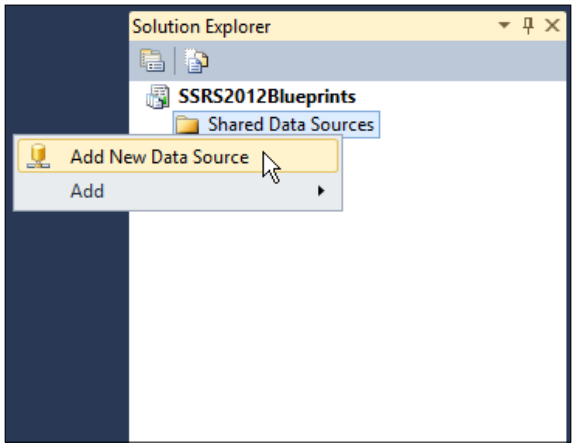
> For the following exercises, we're using SSRS 2012 in native mode. Also, make a note that we're using the `AdventureWorks2012 Sample` database all throughout this book unless otherwise indicated. You can download the sample database from **CodePlex**. Here's the link: `http://msftdbprodsamples.codeplex.com/releases/view/55330`.

Defining a data source for the project

Now, let's define a shared data source and shared dataset for the first report. A shared dataset and data source can be shared among the reports within the project:

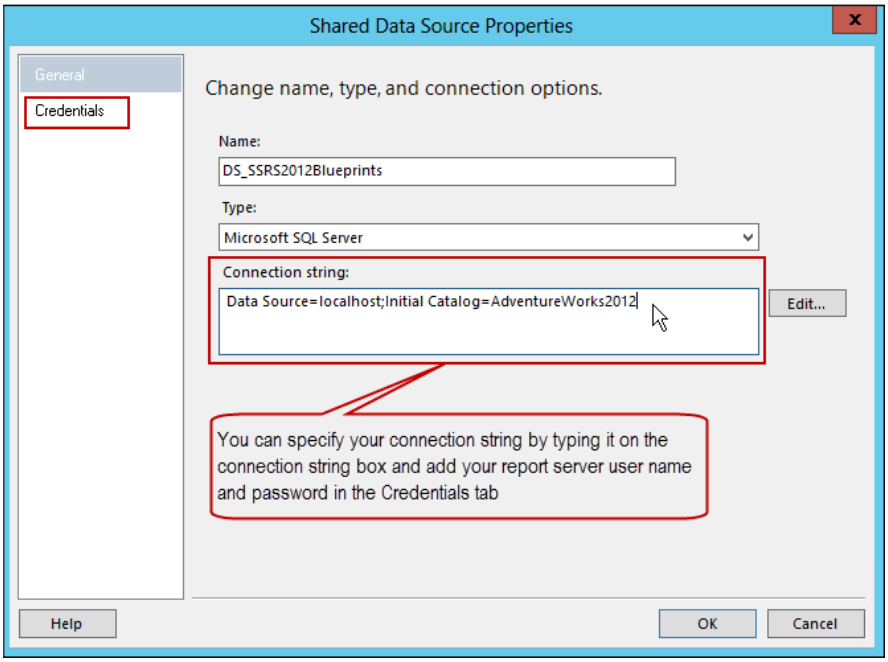
1. Right-click on the `Shared Data Sources` folder under the **SSRS2012Bueprints** solution in the **Solution Explorer** window, as shown in the following illustration.

   If the **Solution Explorer** window is not visible, access it by navigating to **Menu | View | Solution Explorer**, or press *Ctrl* + *Alt* + *L*:



2. Select **Add New Data Source** which displays the **Shared Data Source Properties** window. Let's name our data source `DS_SSRS2012Blueprint`.

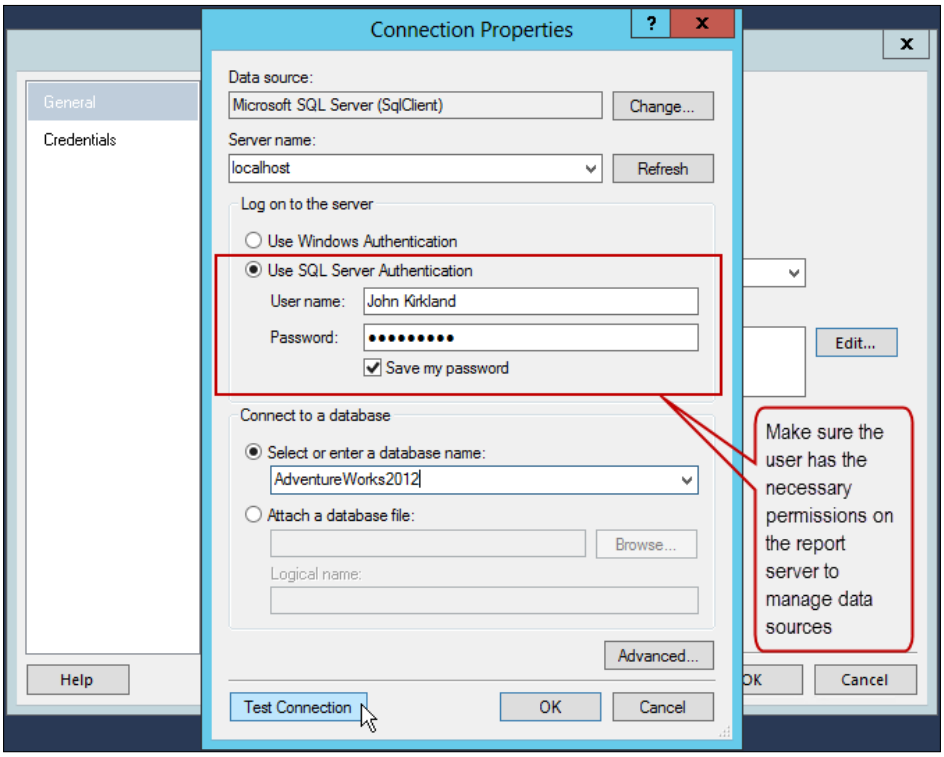
   For this demonstration, let's use the wizard to create the connection string. As a good practice, I use the wizard for setting up connection strings for my data connections. Aside from convenience, I'm quite confident that I'm getting the right connections that I want.



3. Another option for setting the connection is through the **Connection Properties** dialog box, as shown in the next screenshot.

   Clicking on the **Edit** button next to the connection string box displays the **Connection Properties** dialog box:

> Shared versus embedded data sources and datasets: as a good practice, always use shared data sources and shared datasets where appropriate. One characteristic of a productive development project is using reusable objects as much as possible.



For the connection, one option is to manually specify the connection string as shown:

```
Data Source=localhost;Initial Catalog=AdventureWorks2012
```

We may find this option as a convenient way of creating our data connections. But if you're new to the report environment you're currently working on, you may find setting up the connection string manually more cumbersome than setting it up through the wizard.

Always test the connection before saving your data source. After testing, click on the **OK** buttons on both dialog boxes.

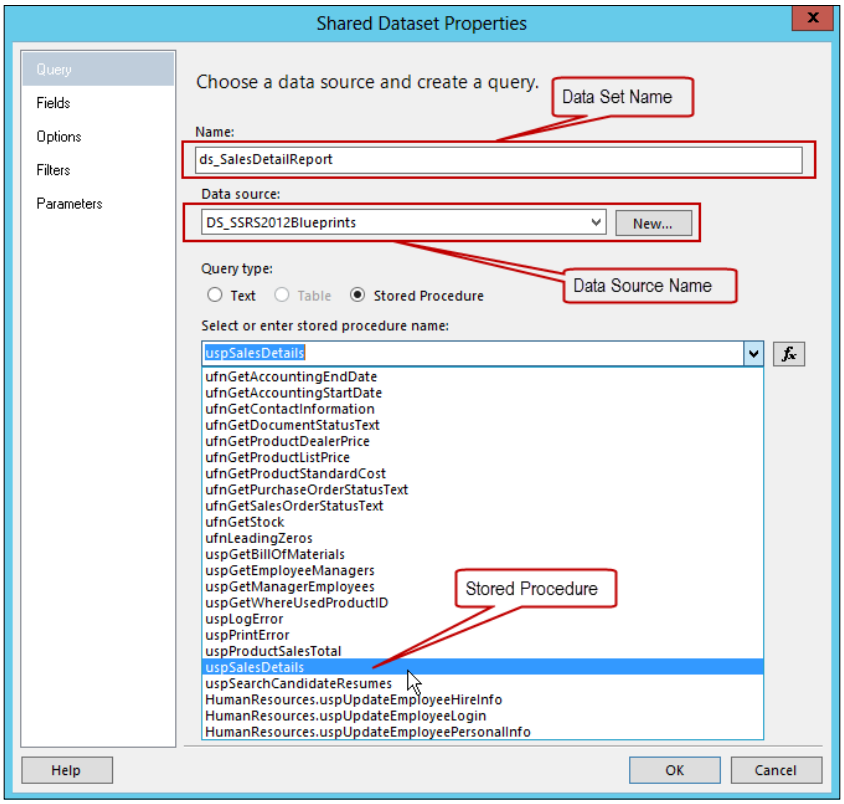
Defining the dataset for the project

Our next step is to create the shared dataset for the project. Before doing that, let's create a stored procedure named `dbo.uspSalesDetails`. This is going to be the query for our dataset.

Download the T-SQL codes included in this book if you haven't done so already. We're going to use the T-SQL file named `uspSalesDetails_Ch01.sql` for this chapter.

We will use the same stored procedure for this whole chapter, unless otherwise indicated.

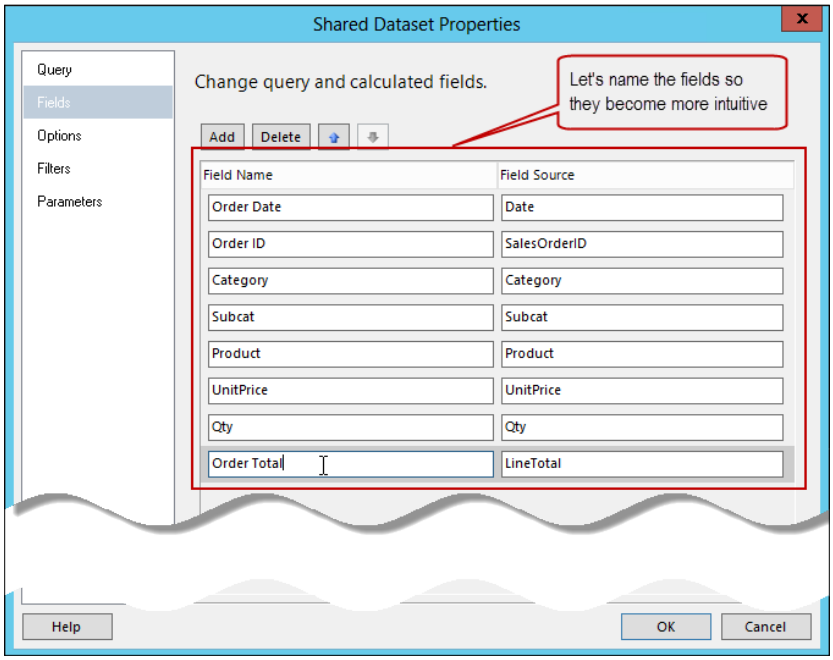
1. Right-click on the `Shared Datasets` folder in **Solution Explorer**, just like we did when we created the data source. That displays the **Shared Datasets Properties** dialog.
2. Let's name our dataset `ds_SalesDetailReport`. We use the query type stored procedure, and select or type `uspSalesDetails` on the **Select or enter stored procedure name** drop-down combo box. Click on **OK** when you're done:



Before we work on the report itself, let's examine our dataset.

In the **Solution Explorer** window, double-click on the dataset `ds_SalesDetailReport.rsd`, which displays the **Shared Dataset Properties** dialog box.

Notice that the fields returned by our stored procedure have been automatically detected by the report designer. You can rename the field as shown:



> Ad-hoc Query (Text Query Type) versus Stored Procedure: as a good practice, always use a stored procedure where a query is used. The primary reason for this is that a stored procedure is compiled into a single execution plan. Using stored procedures will also allow you to modify certain elements of your reports without modifying the actual report.

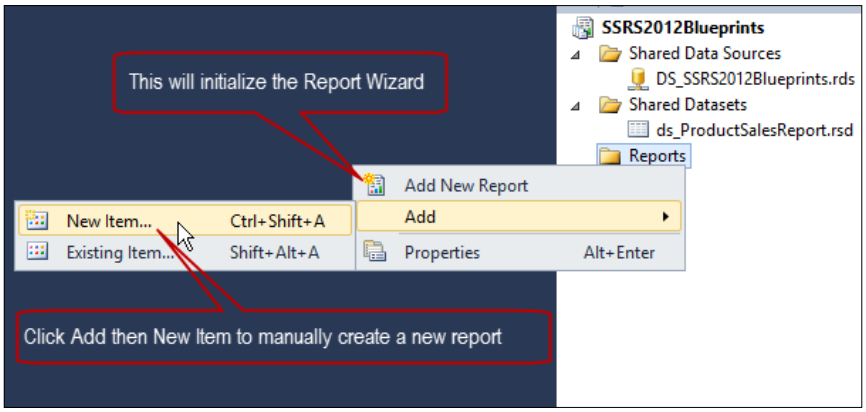
Creating the report file

Now, we're almost ready to build our first report. We will create our report by building it from scratch by performing the following steps:

1. Going back to the **Solution Explorer** window, right-click on the `Reports` folder. Please take note that selecting the **Add New Report** option will initialize **Report Wizard**. Use the wizard to build simple tabular or matrix reports.

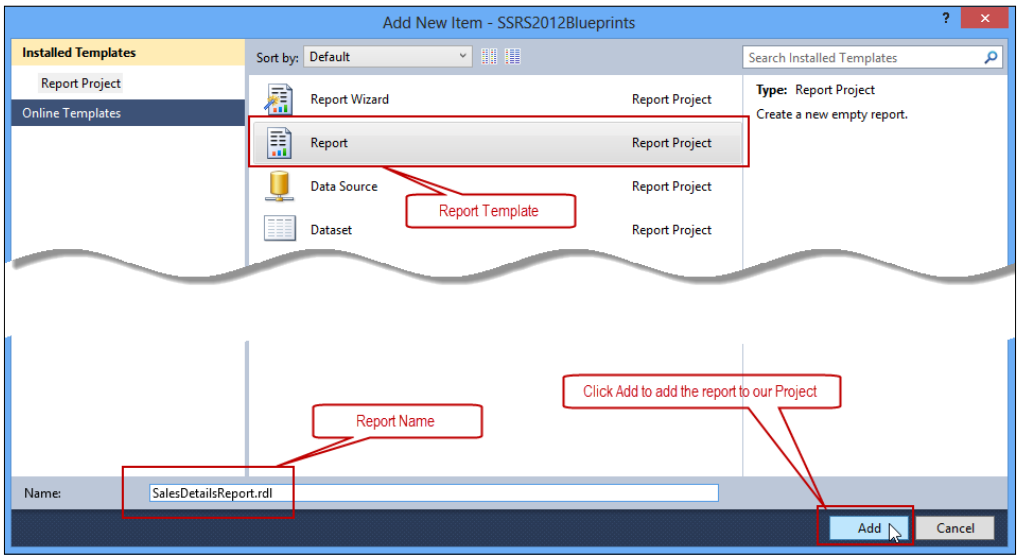
   Go ahead if you want to try the wizard but for the purpose of our demonstration, we'll skip the wizard.

2. Select **Add**, instead of **Add New Report**, then select **New Item**:

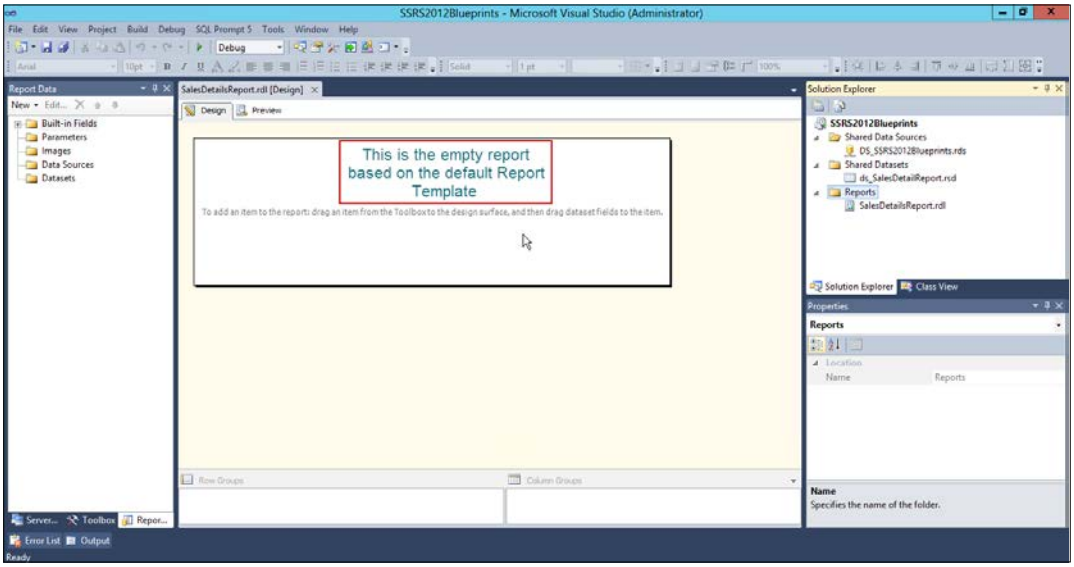


Selecting **New Item** displays the **Add New Item** dialog box as shown in the following screenshot.

3. Choose the **Report** template (default report template) in the template window. Name the report `SalesDetailsReport.rdl`.
4. Click on the **Add** button to add the report to our project:

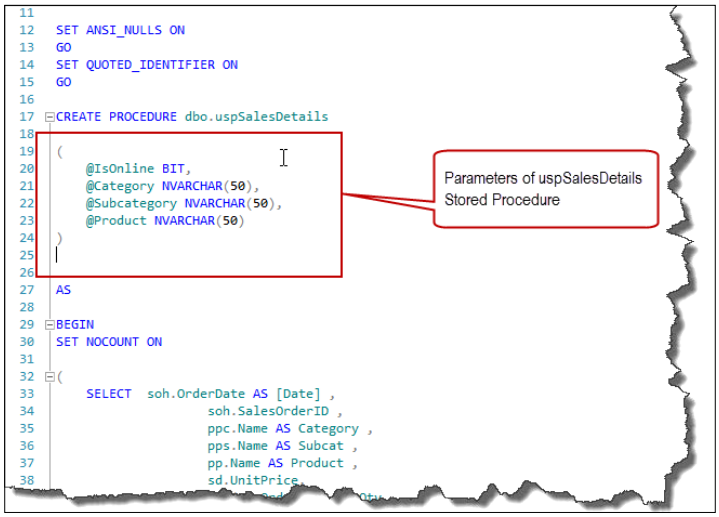


Clicking on the **Add** button displays the empty report in the report designer. It looks similar to the following screenshot:



Creating a parameterized report

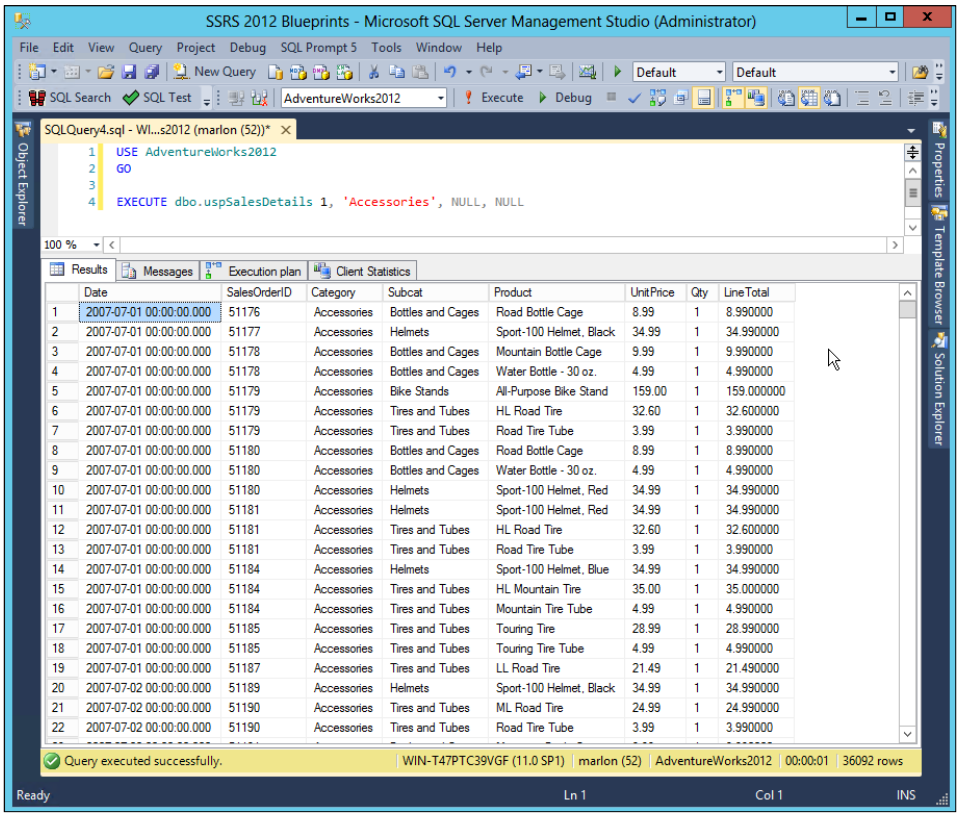
You may have noticed that the stored procedure we created for the shared dataset is parameterized. It has the following parameters:



It's a good practice to test all the queries on the database just to make sure we get the datasets that we need. Doing so will eliminate a lot of data quality issues during report execution.

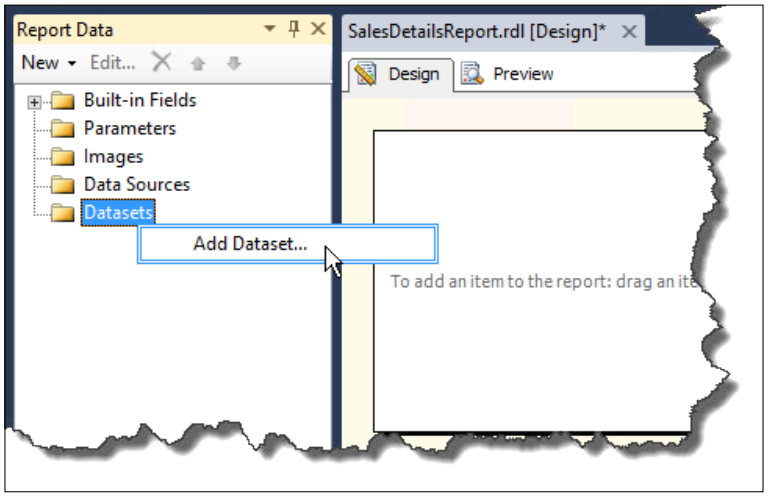
This is also the best time to validate all our data. We want our report consumers to have the correct data that is needed for making critical decisions.

Let's execute the stored procedure in SQL Server Management Studio (SSMS) and take a look at the execution output. We want to make sure that we're getting the results that we want to have on the report.



Now, we add a dataset to our report based on the shared dataset that we had previously created:

1. Right-click on the **Datasets** folder in the **Report Data** window. If it's not open, you can open it by navigating to **Menu** | **View** | **Report Data**, or press *Ctrl* + *Alt* + *D*:



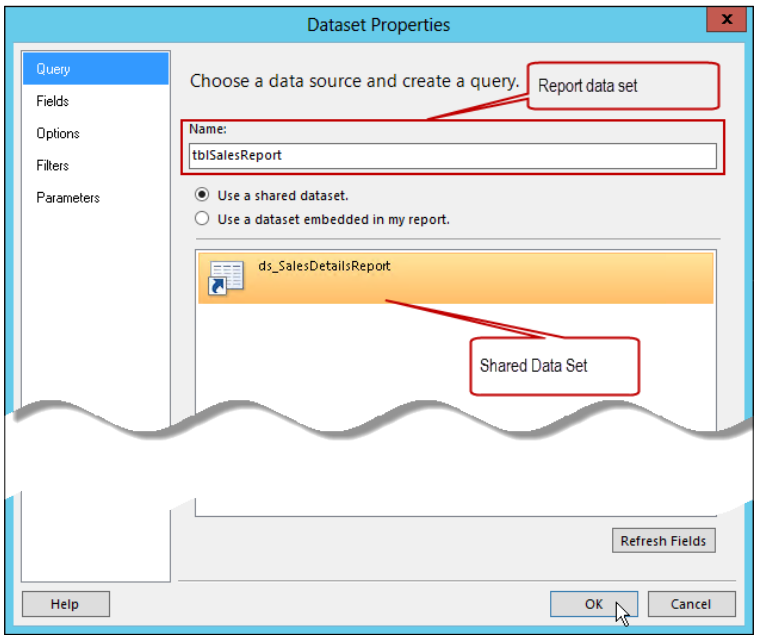
2. Selecting **Add Dataset** displays the **Dataset Properties**. Let's name our report dataset `tblSalesReport`.

   We will use this dataset as the underlying data for the table element that we will create to hold our report data.

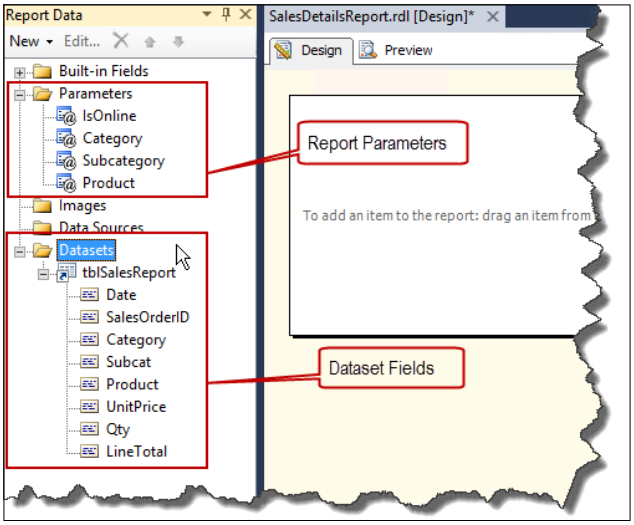
3. Indicate that we want to use a shared dataset.

4. A list of the project shared datasets is displayed. We only have one at this point, which is the `ds_SalesDetailsReport`.

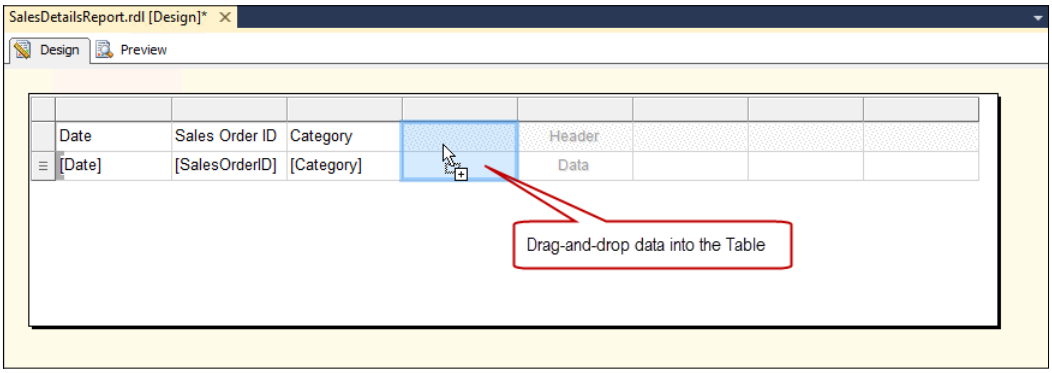
   Let's select that one, then click on **OK**.



Going back to the **Report Data** window, you may notice that we now have more objects under the `Parameters` and `Datasets` folders.



5. Switch to the **Toolbox** window. If you don't see it, then go to **Menu | View | Toolbox**, or press *Ctrl* + *Alt* + *X*. Double-click or drag a table to the empty surface of the designer.
6. Let's add more columns to the table to accommodate all eight dataset fields. Click on the table, then right-click on the bar on the last column and select **Insert Column | Right**.
7. To add data to the report, let's drag each element from the dataset to their own cell at the table data region.



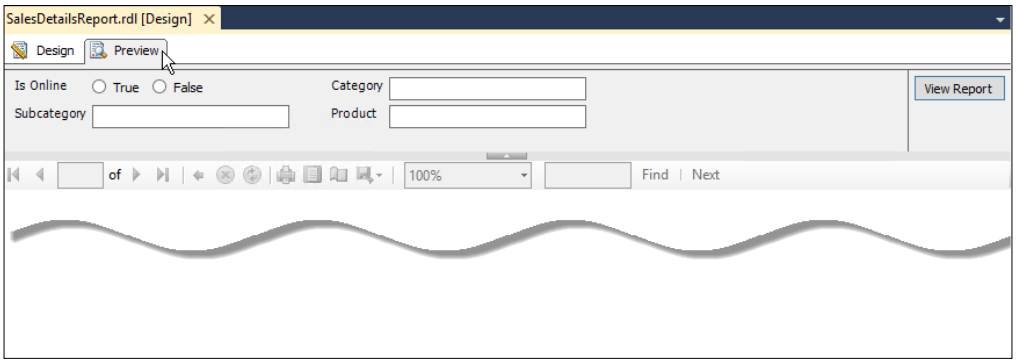
There are three data regions in SSRS: table, matrix, and list. In SSRS 2012, a fourth data region has been added but you can't see that listed anywhere. It's called **tablix**.

Tablix is not shown as an option because it is built into those three data regions. What we're doing in the preceding screenshot is essentially dragging data into the underlying tablix data region.

But how can I add my parameters into the report? you may ask.

Well, let's switch to the **Preview** tab. We should now see our parameters already built into the report because we specified them in our stored procedure.

Our report should look similar to the following screenshot:



Creating a radio button parameter

Like I said, all the parameters shown in the previous screenshot are prebuilt by way of our stored procedure. That's including the radio button parameter at the top of our report.

How to do this? Let's take a look at our stored procedure.

You might have noticed that the first parameter is called `@IsOnline` and is of T-SQL type BIT or Boolean (`TRUE` or `FALSE`).

This parameter is checked against a column in the underlying table named `OnlineOrderFlag`, which holds a value of either `0` or `1` (BIT); `1` returns `TRUE`. We're passing this parameter in the `WHERE` clause as shown in the following expression:

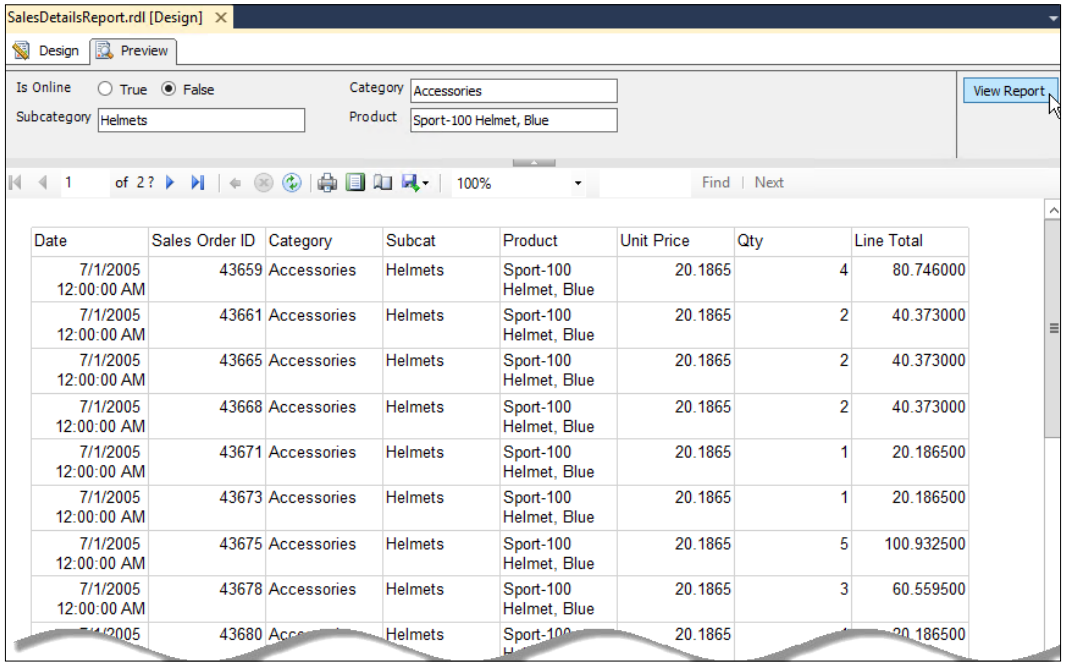
```
Select …
From …
Where OnlineOrderFlag = @IsOnline
```

So, let's run our report with some values. Supply the following data into their respective parameter container:

```
IsOnline: False
Category: Accessories
Subcategory: Helmets
Product: Sport-100 Helmet, Blue
```

Basically, what we want to see in this report execution are all the items sold in-store only, excluding all the online purchases.

We should get something like the following screenshot:



Date	Sales Order ID	Category	Subcat	Product	Unit Price	Qty	Line Total
7/1/2005 12:00:00 AM	43659	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	4	80.746000
7/1/2005 12:00:00 AM	43661	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	2	40.373000
7/1/2005 12:00:00 AM	43665	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	2	40.373000
7/1/2005 12:00:00 AM	43668	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	2	40.373000
7/1/2005 12:00:00 AM	43671	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	1	20.186500
7/1/2005 12:00:00 AM	43673	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	1	20.186500
7/1/2005 12:00:00 AM	43675	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	5	100.932500
7/1/2005 12:00:00 AM	43678	Accessories	Helmets	Sport-100 Helmet, Blue	20.1865	3	60.559500
7/1/2005	43680	Acc	Helmets	Sport-100	20.1865	1	20.186500

Making a parameter optional

Another common requirement on parameterized reports is the ability to exclude one or more parameters at runtime. For example, we want to see all the sold items online under the accessories category without going further down to subcategory and product.

In this example, let's make subcategory and product optional. We need to modify our stored procedure first, as done in the following expression:

```
Select …
From …
Where Subcategory  = COALESCE(@Subcategory, Subcategory)
      AND Product = COALESCE(@Product, Product)
```

Please take a note that the fields may have different names in the code included with the book. The previous code is for clarity purpose only. Modify the codes accordingly.

Basically, what we're trying to do here is we're testing the `@Subcategory` and `@Product` parameters for null values. The `COALESCE()` function returns the first nonnull in the argument.

We can also use the `ISNULL()` function, which works like this: `ISNULL (check_expression, replacement_value)`.

> Don't confuse the `ISNULL()` function with the `IS NULL` operator. They're two different things.

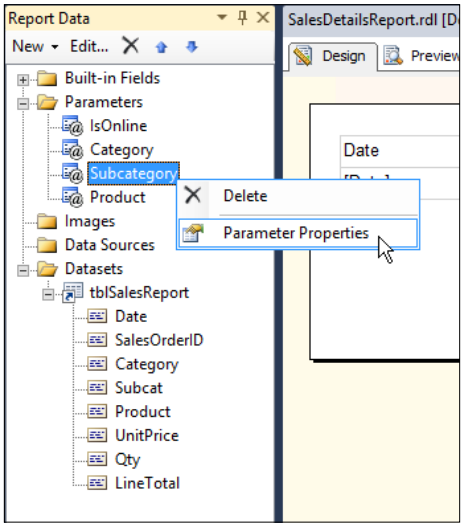
You might be debating which one to use, `COALESCE()` or `ISNULL()`.

The difference between the two functions are widely documented and debated on various forums. Let it suffice to say that `ISNULL()` is not an ANSI standard, and thus, proprietary only to T-SQL.

Let's go back to our report.

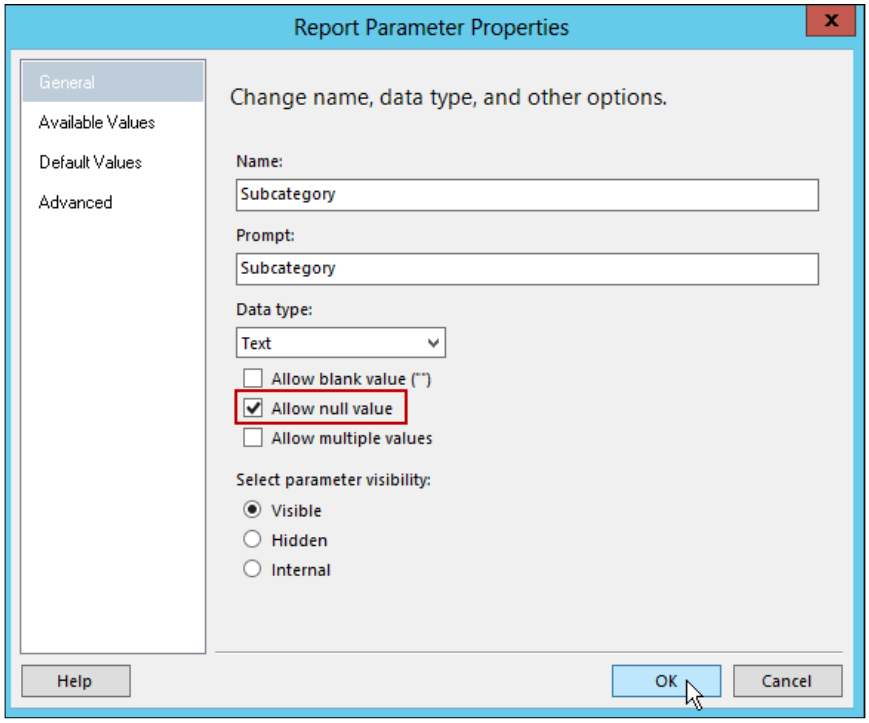
After we modified the stored procedure to test null values, we then modify the subcategory and product parameters in our report:

1. In the **Report Data** window, let's expand the `Parameters` folder and right-click on the **Subcategory** parameter then select **Parameter Properties**:



Clicking on the **Parameter** properties displays, you guessed it right, the **Report Parameter Properties** dialog box.

2. Go to the **General** tab and then select the **Allow null value** checkbox, then click on **OK**.



3. Repeat the same on the **Product** parameter and we're all set. Now, let's run our report with the optional parameters.
4. Let's try the following values:

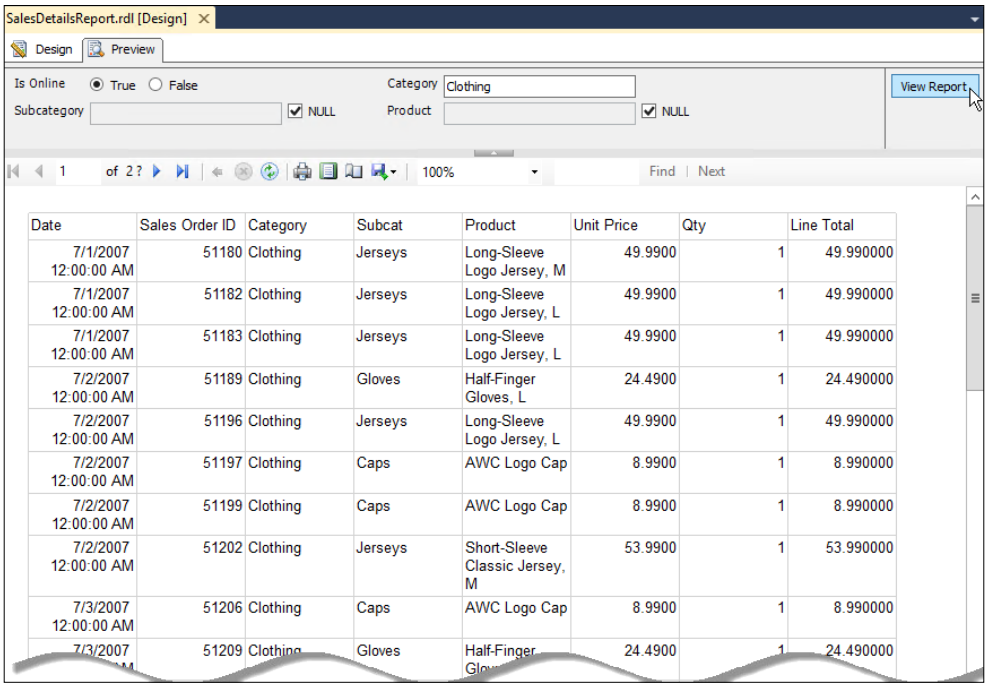
```
IsOnline: True (we want to know what items are sold online only,
not including in-store)

Category: Clothing

Subcategory: NULL

Product: NULL
```

We would come up with something like the following screenshot:



Date	Sales Order ID	Category	Subcat	Product	Unit Price	Qty	Line Total
7/1/2007 12:00:00 AM	51180	Clothing	Jerseys	Long-Sleeve Logo Jersey, M	49.9900	1	49.990000
7/1/2007 12:00:00 AM	51182	Clothing	Jerseys	Long-Sleeve Logo Jersey, L	49.9900	1	49.990000
7/1/2007 12:00:00 AM	51183	Clothing	Jerseys	Long-Sleeve Logo Jersey, L	49.9900	1	49.990000
7/2/2007 12:00:00 AM	51189	Clothing	Gloves	Half-Finger Gloves, L	24.4900	1	24.490000
7/2/2007 12:00:00 AM	51196	Clothing	Jerseys	Long-Sleeve Logo Jersey, L	49.9900	1	49.990000
7/2/2007 12:00:00 AM	51197	Clothing	Caps	AWC Logo Cap	8.9900	1	8.990000
7/2/2007 12:00:00 AM	51199	Clothing	Caps	AWC Logo Cap	8.9900	1	8.990000
7/2/2007 12:00:00 AM	51202	Clothing	Jerseys	Short-Sleeve Classic Jersey, M	53.9900	1	53.990000
7/3/2007 12:00:00 AM	51206	Clothing	Caps	AWC Logo Cap	8.9900	1	8.990000
7/3/2007	51209	Clothing	Gloves	Half-Finger	24.4900	1	24.490000

Populating a drop-down parameter with a dataset

Normally, we don't want to type in values in the parameter field. So, we want to populate that field with the available data from our backend table.

Another reason is we want to give the report consumers only the available values from our dataset.

For example, with the category parameter, we may want to provide all the category values from our dataset. We don't want to leave them guessing what categories are out there to choose from.

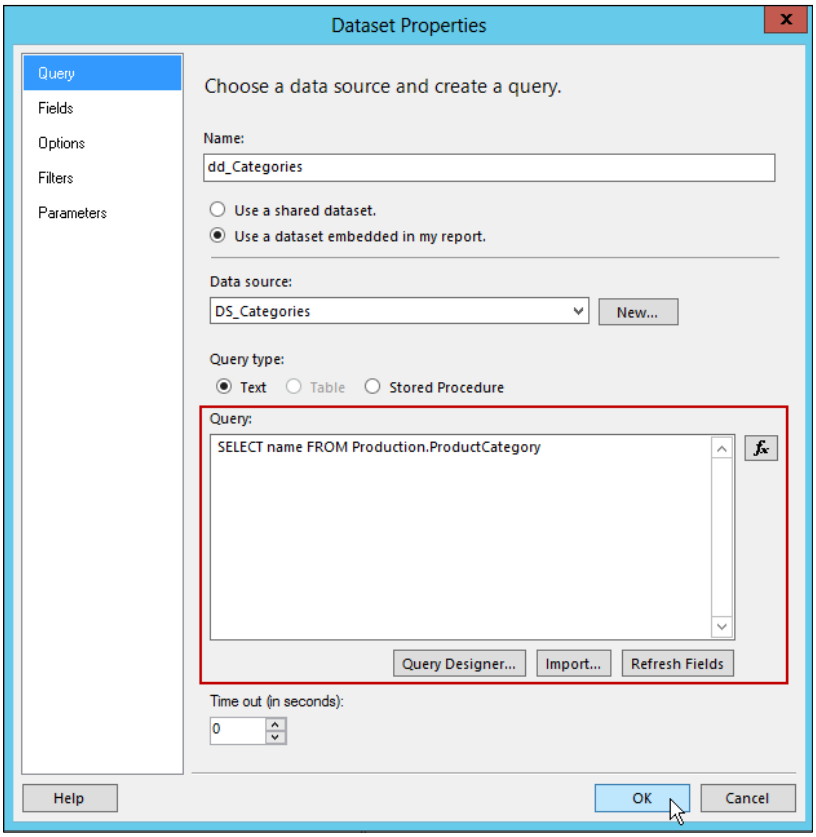
To populate our `@Category` parameter, we need to create a new dataset for our report.

Since we might only need this for this particular report, we'll create a report dataset (in contrast to a shared dataset).

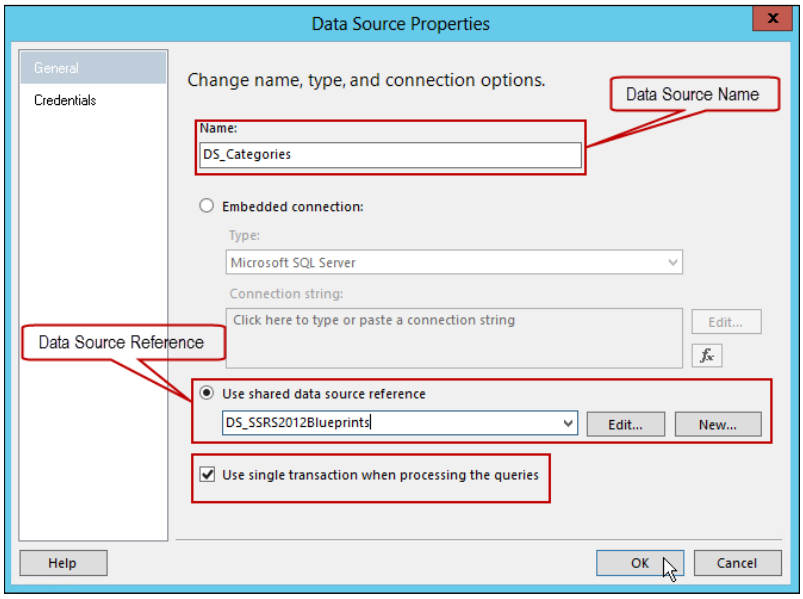
1. Right-click on the `Datasets` folder in the **Report Data** window, and select **Add Dataset**. The **Dataset Properties** dialog box then displays our options to create a dataset.
2. Let's name our dataset `dd_Categories` and select the option **Use a dataset embedded in my report**.
3. Let's skip the data source, we'll come back to that in a second. On the Query type, select Text then type:

   ```
   SELECT Name FROM Production.ProductCategory.
   ```

   Our **Dataset Properties** should look like the following screenshot:



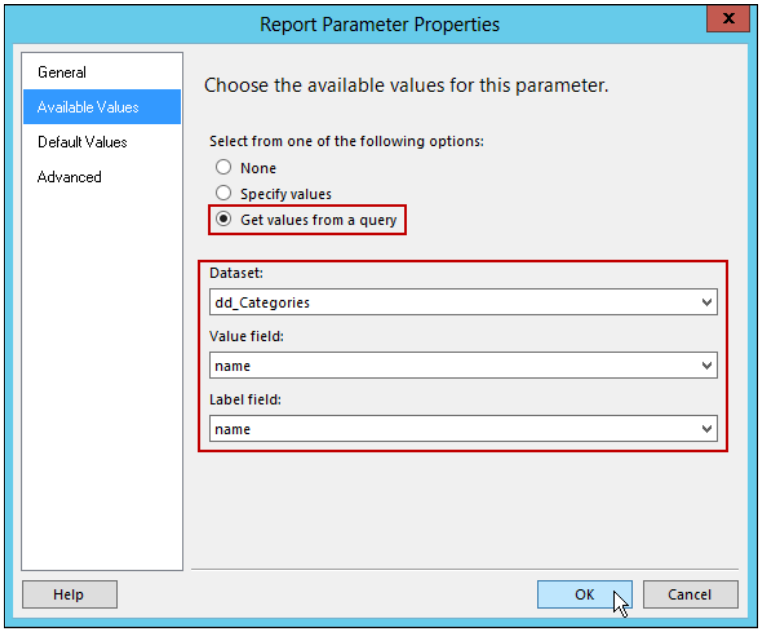
4. Let's go back to the data source. Click on the **New** button next to the data source field, which displays the **Data Source Properties** window. Let's name our source `DS_Categories`, and reference our project's data source as shown in the following screenshot:



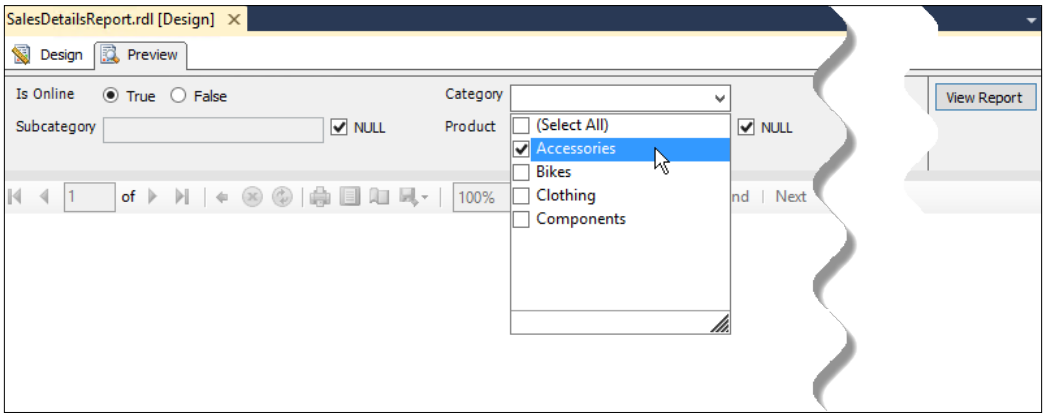
We may also want to select the option to **Use single transaction when processing the queries**. We don't need to specify new credentials for this source because we're inheriting all the credentials from our referenced data source. Click on **OK** on both dialog boxes when done.

5. Now, we right-click on the `@Category` parameter in the **Report Data** window, and select **Parameter Properties**.
6. On the **General** tab, select **Allow multiple values**.

7. Switch to the **Available Values** tab. Select **Get values from a query** and specify the dataset name, value field, and name field according to the dataset we just created. Click on **OK** when done:



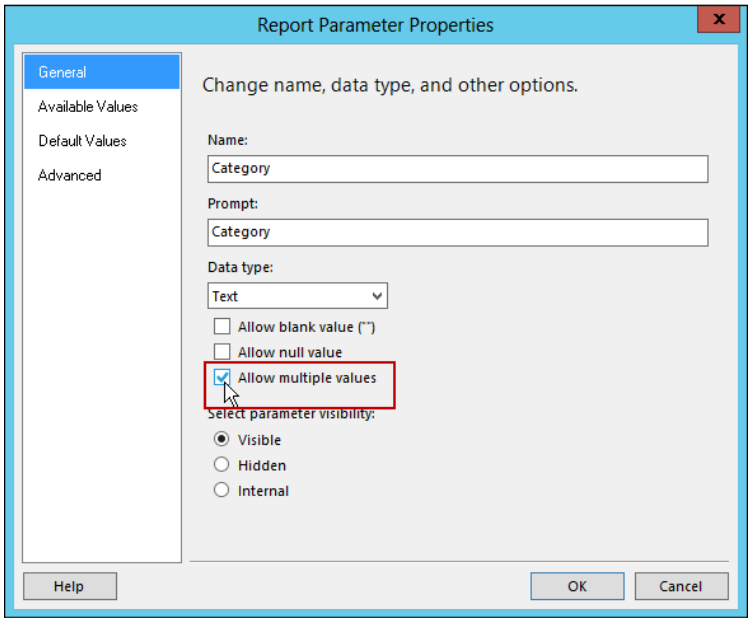
Our report **Category** parameter should now show the drop-down list of categories:



Creating a multivalued parameter

You might have noticed that the category parameter at this point only allows us to select one category at a time. We may want to select two or more Categories.

Let's set the Category parameter to accept multiple values by selecting Allow multiple values in its properties:



The next thing we need to do is create a table-valued function that returns an array of categories. What the function does is simply collect the selected categories from our report and pass it to our stored procedure. We'll do two more things to complete our multivalued parameter.

> Table-valued functions are user-defined functions that returns `Table` datatype. For more information about Table-valued functions, please go to this link `http://technet.microsoft.com/en-us/library/ms191165(v=sql.105).aspx`.

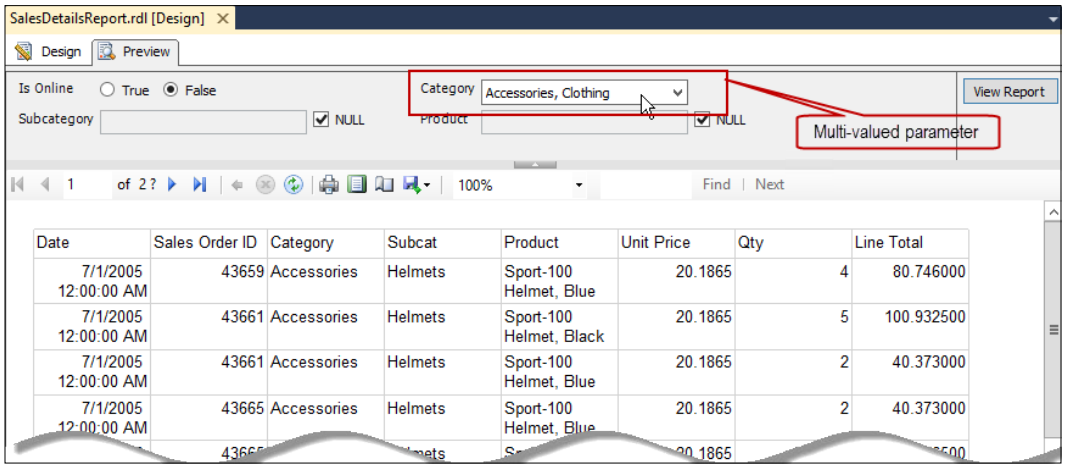
First, let's create the table-valued function and name it `MultiValueParamSplit`. Please find the script named `MultiValueParamSplit_Ch01.sql` in the code folder for this chapter.

Next, let's modify our stored procedure `dbo.uspSalesDetails` to make use of our function like this:

```
Select …
From…
Where Category IN (Select [Category] From dbo.MultiValueParamSplit(@
Category, ','))
```

> **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.

Now, our report should allow us to select multiple categories to pass on our **Category** parameter:

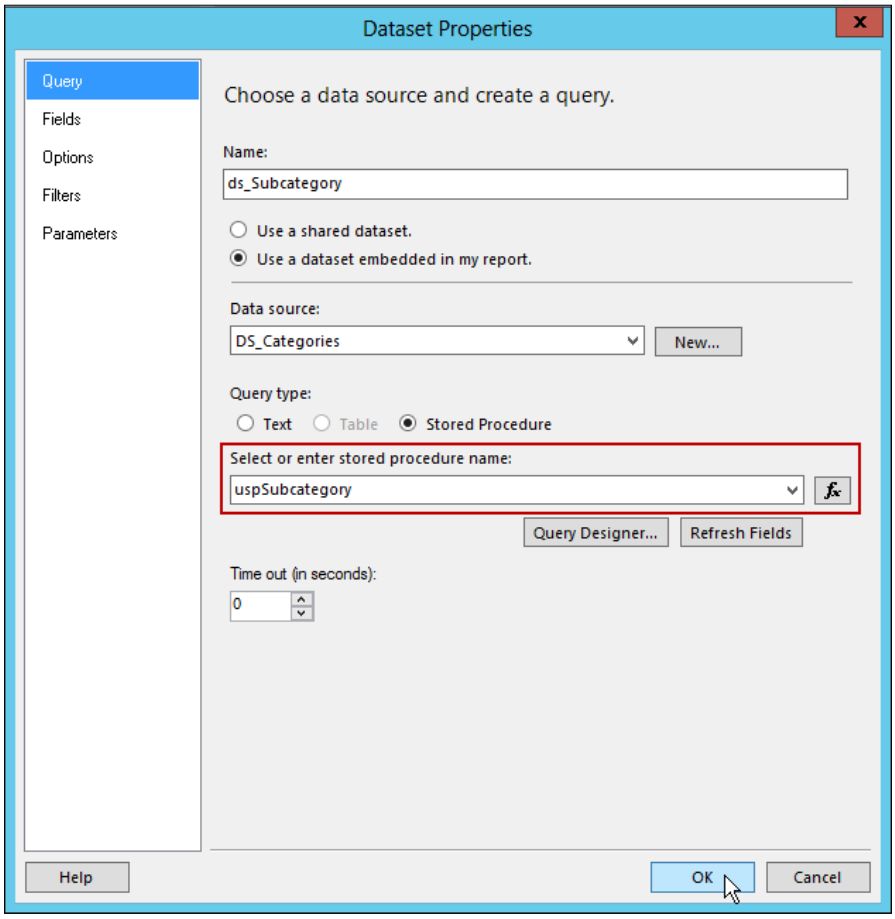


Creating a cascading parameter

Now, let's set up a cascading parameter between **Category** and **Subcategory**. We want to only show subcategories that belong to a particular category. So, if we select **Bikes** on our **Category** parameter, we want to see only the subcategories that belong to bikes, that is, mountain bikes, road bikes, and touring bikes.

Let's create a stored procedure named `dbo.uspSubcategory`. Run the script called `uspSubcategory_Ch01.sql` included in this book.

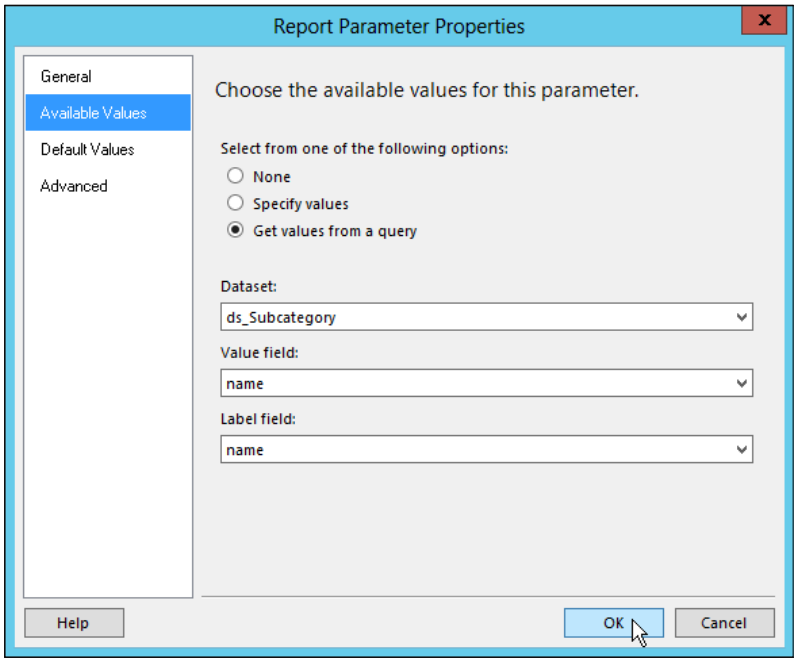
After we create the `dbo.uspSubcategory` stored procedure, let's create a new dataset out of that procedure. Let's name it `ds_Subcategory`:



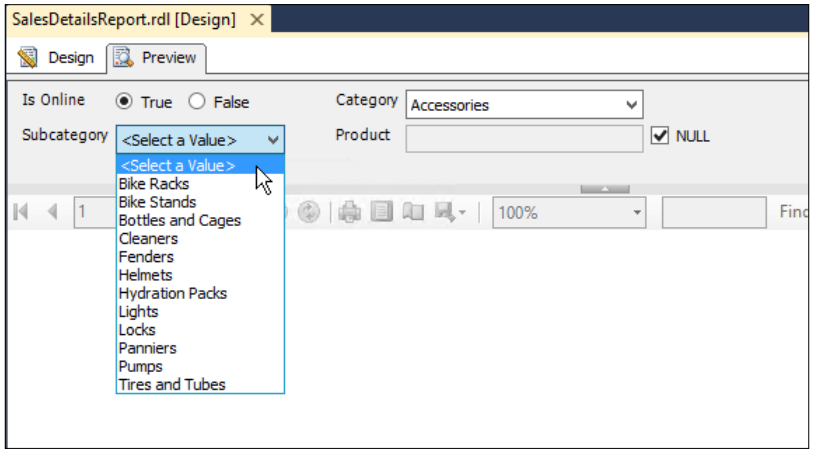
The `uspSubcategory` stored procedure accepts a `Category` variable. So, under the hood, what we're trying to set up is to pass the value of the `category` field to the parameter of the `Subcategory`.

By default, the report designer detects all the parameters in the dataset (in this case, a stored procedure). The **Parameter** tab of the **Dataset Properties** dialog shows all the parameters detected by the report designer.

After setting up the properties for our new dataset, let's set up the `@Subcategory` parameter to use the same dataset:



We should now see that our **Category** parameter cascades to the **Subcategory** parameter:

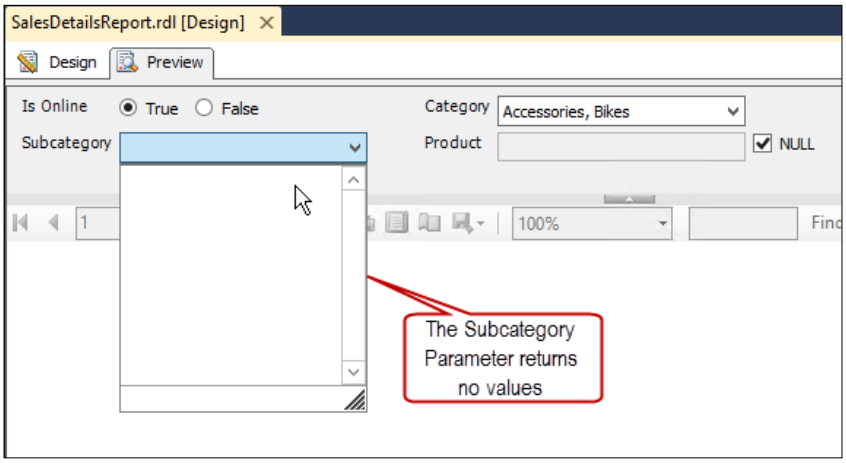


Please make note that we only needed to pass a `@Category` parameter to our **Subcategory** stored procedure without doing any other setup or process on the Report Designer/Manager side, because, both the `@Category` and `@Subcategory` Report Parameters are on the same scope, which is this report (`SalesdetailsReport.rdl`).

Creating a cascading multivalued parameter

We may have noticed that the cascading functionality between the **Category** and **Subcategory** works only when we have a single category. If we select multiple Categories, we want to get all the **Subcategories** (multivalued) under those **Categories**.

We might think that setting up the `@Subcategory` parameter to "allow multiple values" would do the trick. But that won't work:



I've seen people handle this requirement by using custom codes that are processed on the report server.

It's a good practice that anything that can be processed on the SQL Server database should be processed on the database.

With the use of the table-valued function (`MultiValueParamSplit`) we created previously, we can set the `@Subcategory` parameter to accept multiple values without modifying our report. This is the primary reason why we should prefer using stored procedures for our datasets.

Let's modify the dataset behind the Subcategory parameter by modifying the `dbo.uspSubcategory` stored procedure.

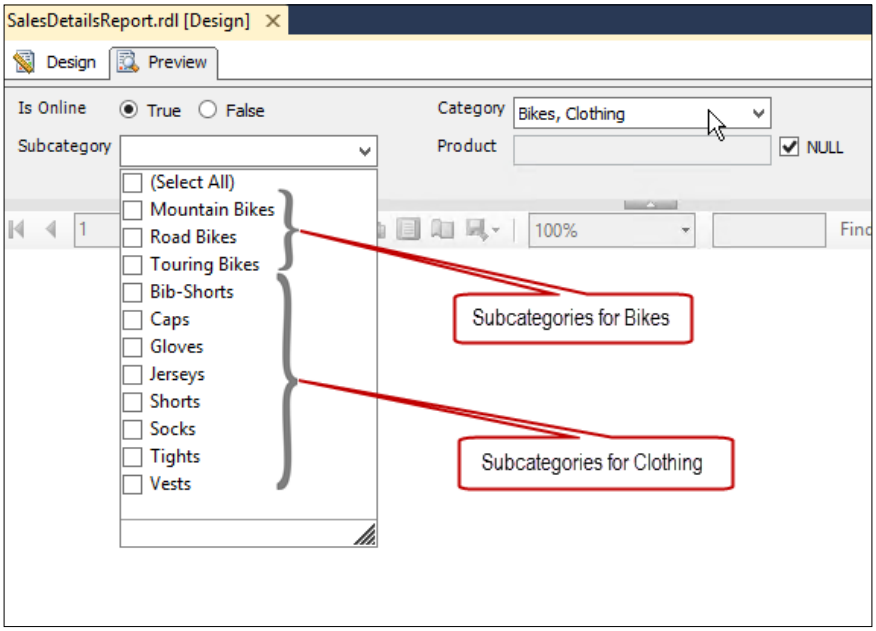
We only need to change the `WHERE` clause from:

```
Select…
From…
WHERE Category = @Category
```

A clause that uses our `MultiValueParameterSplit` function:

```
Select…
From…
WHERE Category IN (SELECT [Category] FROM dbo.MultiValueParamSplit(@
Category, ',')
```

We should now see a cascading effect between the multivalued **Category** and **Subcategory** parameters:



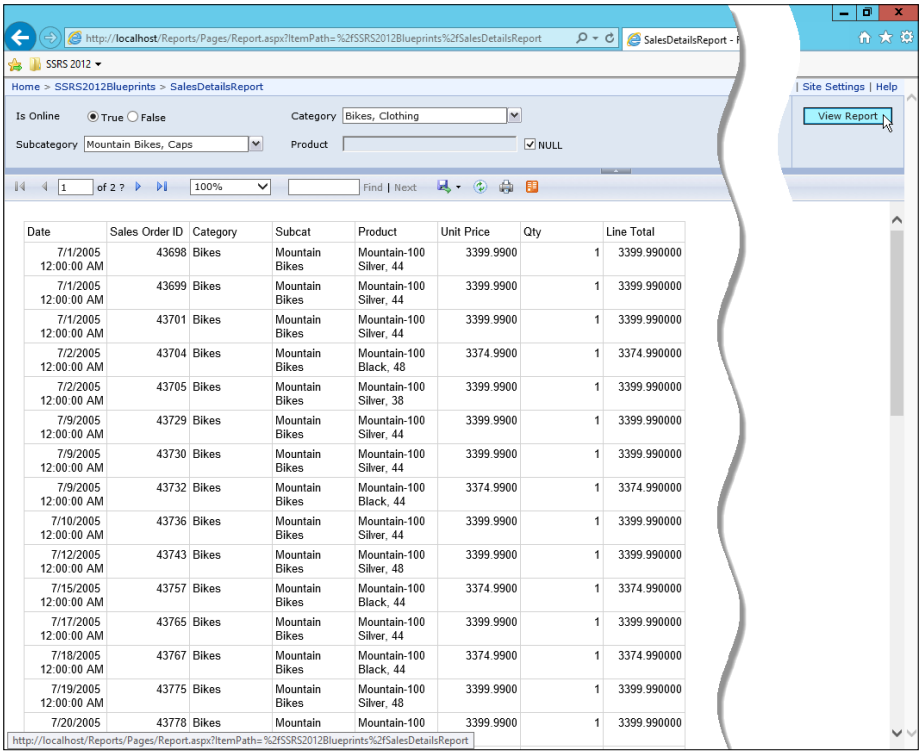
To make use of these cascading subcategories, we still need to modify the underlying dataset of our report which is the `tblSalesReport` to accept the multivalued subcategory parameter. To do that, let's modify the stored procedure `ds_SalesDetailsReport`.

Let's change the `WHERE` clause of the `@Subcategory` parameter from:

```
Select…
From…
Where Subcategory = @Subcategory
to
Select…
From…
Where Subcategory IN (Select [Category] FROM [dbo].[MultiValueParamSplit]
(@Subcategory, '',''))
```

Sales details report in action!

Here's what our report, the sales details report, looks like in action:



Summary

In this chapter, we learned how to build a report from the ground up. We've also learned different tricks on how to implement parameters. If you've followed all the exercises, you should now have a strong grasp on how to use parameters in different scenarios.

2
Making a Good Impression

Let's continue to tag along on John Kirkland's SSRS journey. He was excited that he was able to build a report out of the data he migrated to the company's new SQL Server 2012 instance.

John showed the report to the finance group and they were happy that they now have a more convenient reporting solution with SSRS. This might just be the one reporting solution they've been hoping for, and have finally got. They knew that what John has shown them is just scratching the surface of what they can build for all their reporting requirements.

John wanted to make everything right, so he consulted the Records Manager of the company for any documentation compliance that he might need to be aware of. According to the Records Manager, all documents in any form must adhere to the company's rules of documentation. He wants John to make sure that the structure of all reports be consistent with the company's brand.

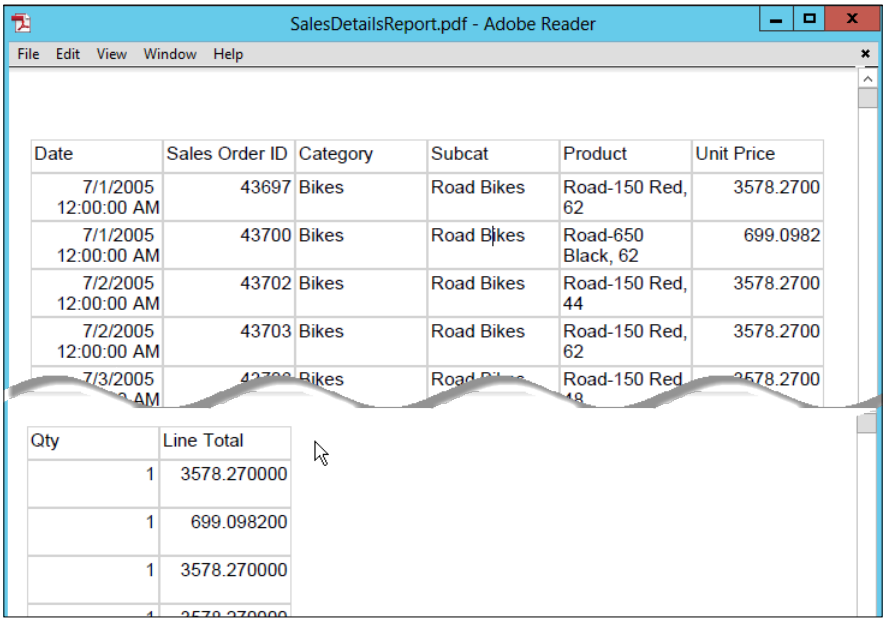
We've learned to build a report from the ground up and used different types of parameters in a SSRS report in *Chapter 1*, *Let's Breakdown the Numbers*. In this chapter, we will polish a slightly modified version of that report to demonstrate some of the settings provided by the SSRS designer.

In this chapter we will learn the following:

- Report formatting
- Grouping
- Simple drilldown
- Grand total and subtotal
- Pagination
- Header and footer

- Report structure
- Adding images
- Report template

Let's render our report from *Chapter 1, Let's Breakdown the Numbers*, into Portable Document Format (PDF). This is what we've got so far:



Obviously, the report looks bland. The width of the table overflows into a second page, and the text in the report is not properly formatted.

Create a new report, or just copy the RDL file from *Chapter 1, Let's Breakdown the Numbers*, into a new report. For the purpose of our demonstration in this chapter, we have removed the `@IsOnline` parameter.

The parameters we are using in this chapter are `@Category`, `@Subcategory`, and `@Product`—all set as cascaded multivalued parameters.

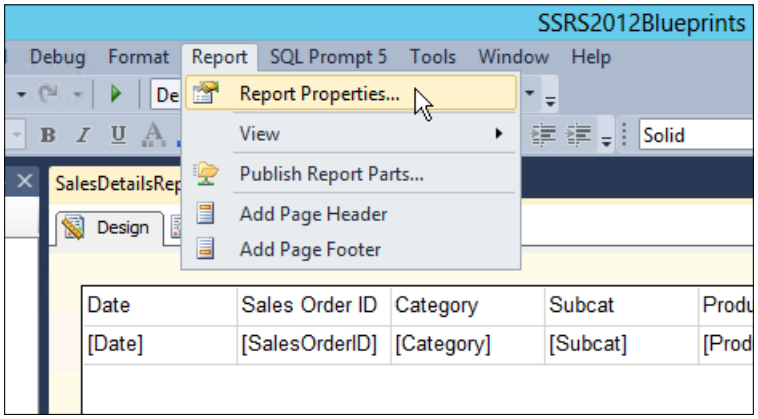
If you want to build a new report from the ground up, run the script called `uspSalesDetails2_Ch02.sql`, which is included in this book, to create a new dataset.

You can also find the script to create the dataset for our `@Product` parameter in the file called `uspProduct_Ch02.sql`, also included in the book.

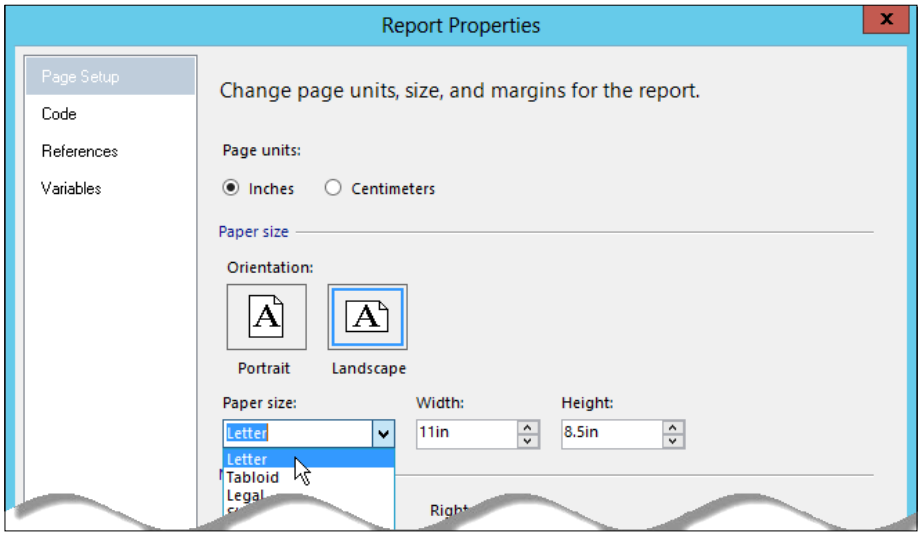
Report page setup

The size of the report table is greater than the page width if the page layout is set to the portrait orientation. To accommodate all the columns of the table in a single page, let's change the orientation of the report to Landscape.

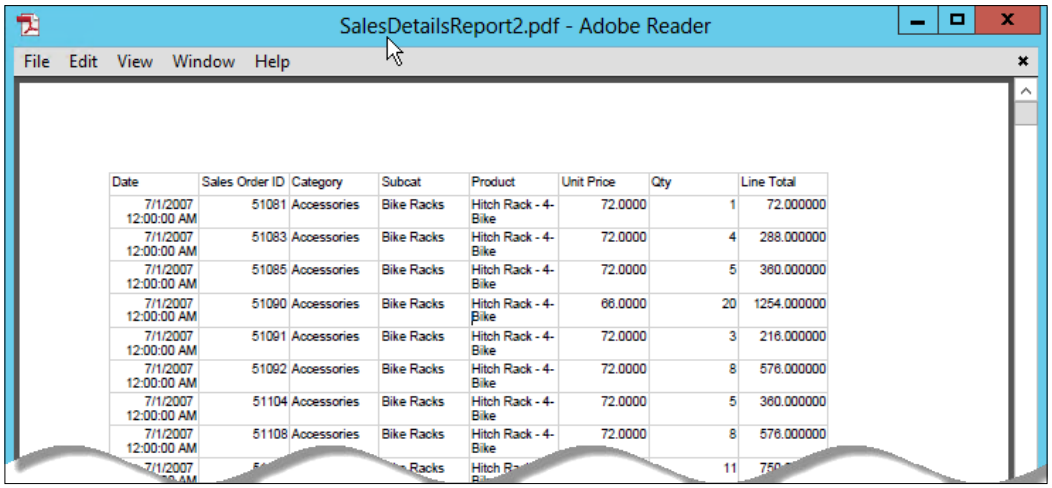
1. Go to the menu bar, and navigate to **Report | Report Properties** as shown in the following screenshot:



2. That displays the **Report Properties** dialog box. Select **Landscape** for **Orientation** and **Letter** from the **Paper size** drop-down list as shown in the following screenshot. Of course, depending on your requirements, you can choose whatever is appropriate.



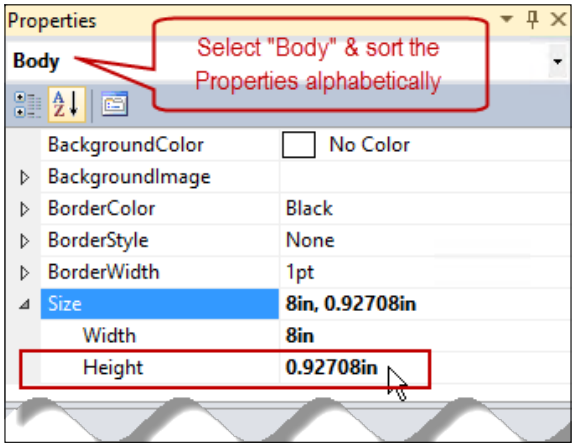
Now let's take a look at the rendered PDF:



Date	Sales Order ID	Category	Subcat	Product	Unit Price	Qty	Line Total
7/1/2007 12:00:00 AM	51081	Accessories	Bike Racks	Hitch Rack - 4-Bike	72.0000	1	72.000000
7/1/2007 12:00:00 AM	51083	Accessories	Bike Racks	Hitch Rack - 4-Bike	72.0000	4	288.000000
7/1/2007 12:00:00 AM	51085	Accessories	Bike Racks	Hitch Rack - 4-Bike	72.0000	5	360.000000
7/1/2007 12:00:00 AM	51090	Accessories	Bike Racks	Hitch Rack - 4-Bike	66.0000	20	1254.000000
7/1/2007 12:00:00 AM	51091	Accessories	Bike Racks	Hitch Rack - 4-Bike	72.0000	3	216.000000
7/1/2007 12:00:00 AM	51092	Accessories	Bike Racks	Hitch Rack - 4-Bike	72.0000	8	576.000000
7/1/2007 12:00:00 AM	51104	Accessories	Bike Racks	Hitch Rack - 4-Bike	72.0000	5	360.000000
7/1/2007 12:00:00 AM	51108	Accessories	Bike Racks	Hitch Rack - 4-Bike	72.0000	8	576.000000
7/1/2007			Racks	Hitch Ra		11	750

If you're getting an empty page at the end of the report, you need to adjust the body size (height) of the report. You usually get blank pages in a rendered PDF because the body height of your report is greater than the page height. You can either drag the body of the report to resize the body height or do it manually by setting the properties of the report at design time.

3. If the **Properties** Window is not already open, go to the **Menu | View | Properties** Window, or simply press *F4*. Set the **Height** of the report **Body** appropriately:

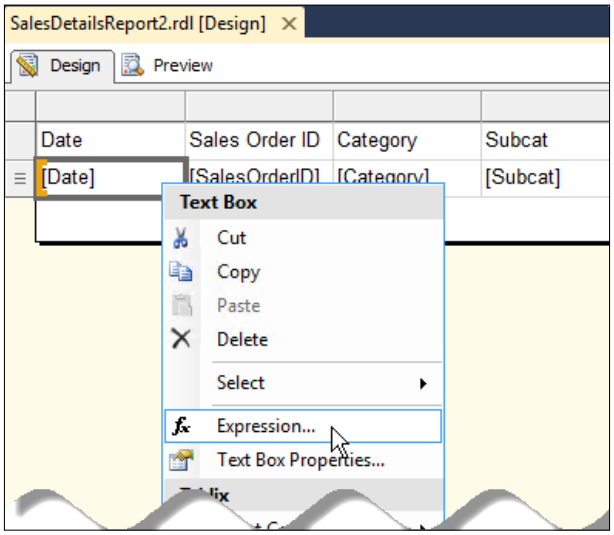


Formatting expression

The format of the data is very important for the design and clarity of the report. All the values must be expressed in their proper format.

The date value in our current report is showing as M/D/YYYY 12:00:00AM. We may want to get rid of the time in our date; unless, of course, time is required in our report.

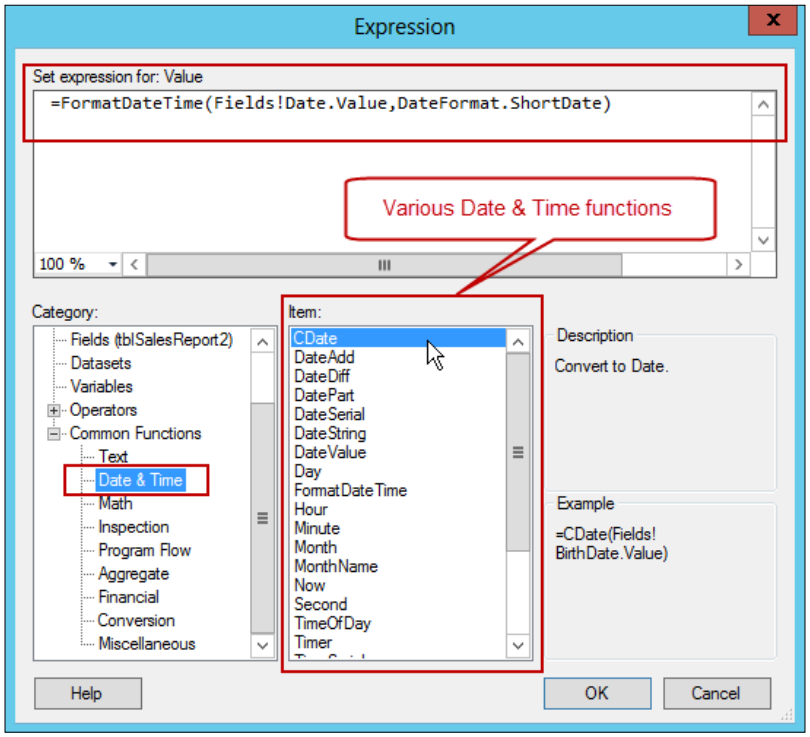
1. Right-click on the **[Date]** cell and select **Expression…** to display the **Expression Builder** dialog box:



2. The default expression for the **Date** is `Fields!Date.Value`. Change that to:

```
=FormatDateTime(Fields!Date.Value,DateFormat.ShortDate)
```

The `FormatDateTime` function, with its argument `DateFormat.ShortDate`, converts our date to short date format.



3. You can also include literals in the expression. Let's add the literal `"SO"` prefix to `SalesOrderID` (that is, `SO51061`) like this:

   ```
   ="SO" & Fields!SalesOrderID.Value
   ```

4. Let's also format the two Dollar amounts in the report; **Unit Price** and **Line Total**.

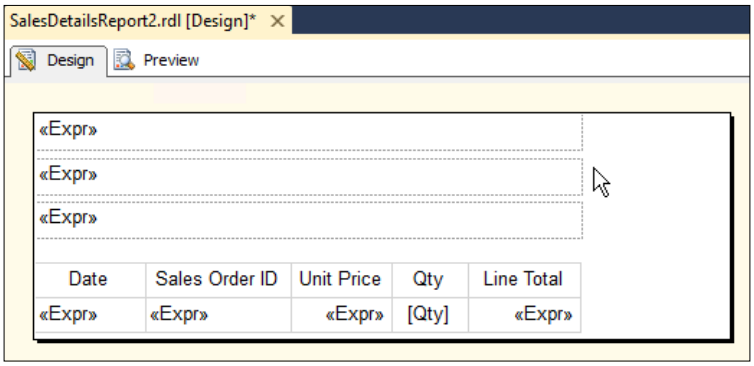
   ```
   =FormatCurrency(Fields!UnitPrice.Value)
   ```

   ```
   =FormatCurrency(Fields!LineTotal.Value)
   ```

Adding report objects

We might have formatted our data, but it doesn't look any better at this point. We still need more elements in the report to get the right look and feel that we want.

Let's delete the `Category`, `Subcategory`, and `Product` columns, and switch the page orientation from landscape to portrait. Make room for three textboxes at the top of the table, and then, drag-and-drop three textboxes as shown in the following screenshot:



1. Add the following expressions in the three textboxes:

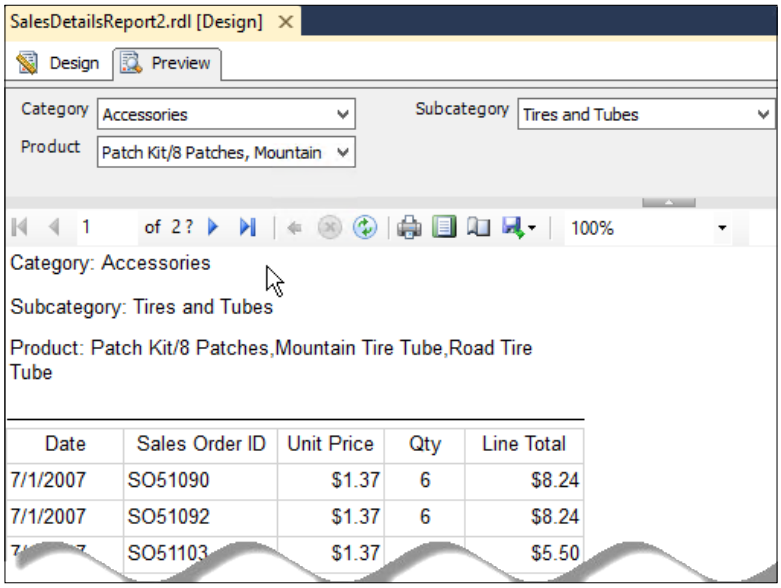
   ```
   ="Category: " & Parameters!Category.Value(0)
   ="Subcategory: " & Parameters!Subcategory.Value(0)
   ="Product: " + Join(Parameters!Product.Label,",")
   ```

2. Let's run our report with the following parameters:

   ```
   Category: Accessories
   Subcategory: Tires and Tubes
   Product: Select the top 3
   ```

   The textboxes will contain the values defined by the report parameters. The `Join` function in the **Product** drop-down list concatenates the multivalue from the `Product` parameter into a comma-separated string.

3. Add a line on top of the table by dragging the Line object from the toolbox. Here's how our report looks like now:



Formatting literals in an expression

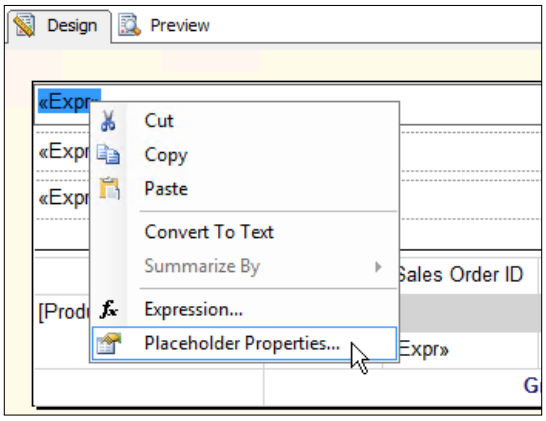
We may want to make label for our textboxes: Category, Subcategory, and Product:

- **Category**: Accessories
- **Subcategory**: Tires and Tubes
- **Product**: Patch kit / 8 patches, mountain tire tube, road tire tube

Prior to SQL Server Reporting Services 2008, you can only do that by creating two separate textboxes, so you can set separate font styles for each textbox.

SSRS 2008 and above have another property, called placeholder, that allows us to set different styles for literals and expressions within a single textbox. Let's do that:

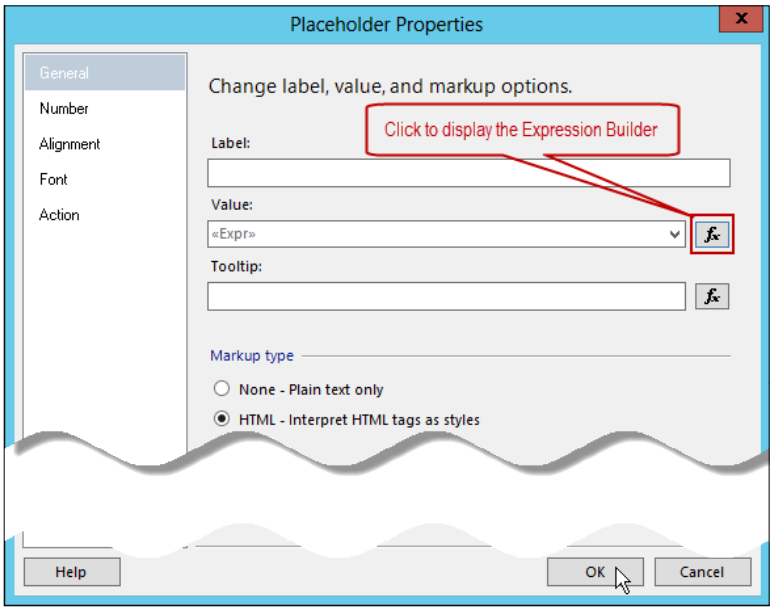
1. Go back to the designer, highlight the **<<Expr>>** in the first textbox and right-click on it, then select **Placeholder Properties**:



2. That displays the **Placeholder Properties** dialog box. Click on the expression builder button next to the **Value** drop-down list (the button labeled **fx**). Change the expression to:

```
="<B>Category:</B> " & Parameters!Category.Value(0)
```

3. Click on **OK** when done.
4. Going back to **Placeholder Properties**, select **HTML-Interpret HTML** tags as styles for the markup type. Then click on **OK**.



5. Do the same for the **Subcategory** and **Product** expressions.

You can also format the value in a textbox as a plain text by setting its format in the **Placeholder Properties** dialog box. In our example, we can make the expression bold by setting its **Font** property to **Bold** in the **Font** tab.

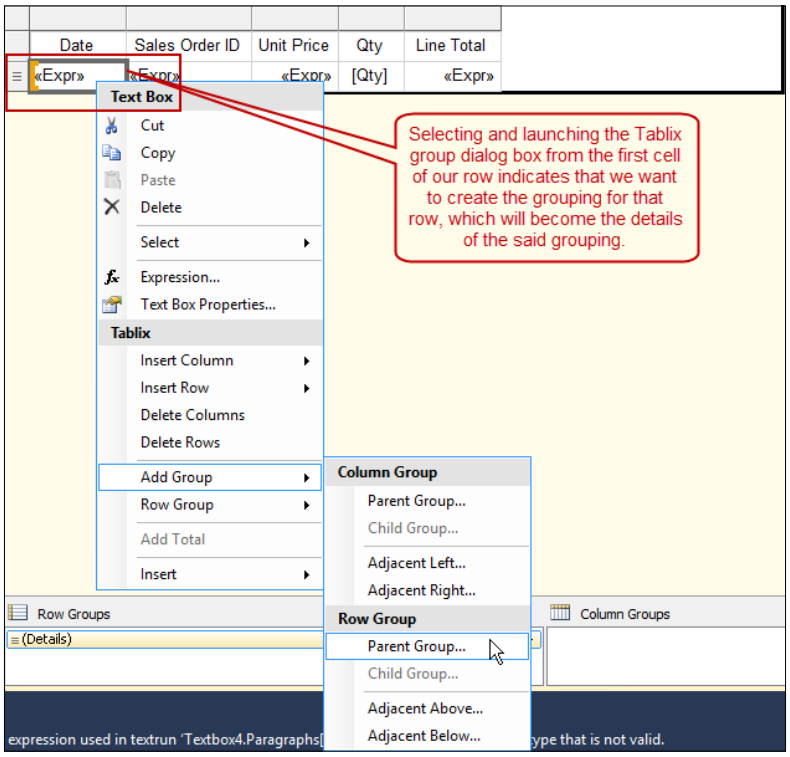
You can have multiple placeholders in a single textbox. All placeholders can be formatted individually.

Run the report to see the changes. We should now have **Category:**, **Subcategory:**, and **Product:** in bold.

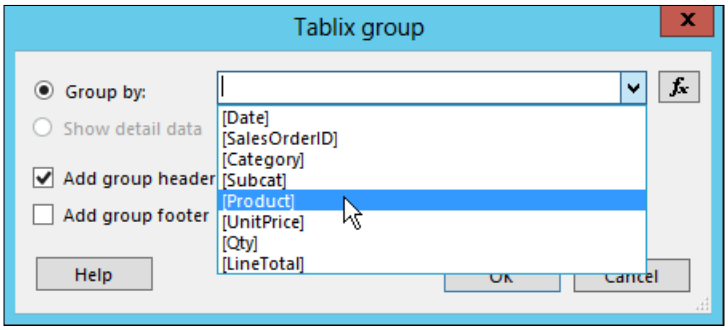
Creating a row group

Grouping data in a report adds readability to your overall design. But, be aware of the negative impact of adding groups in the performance of your report.

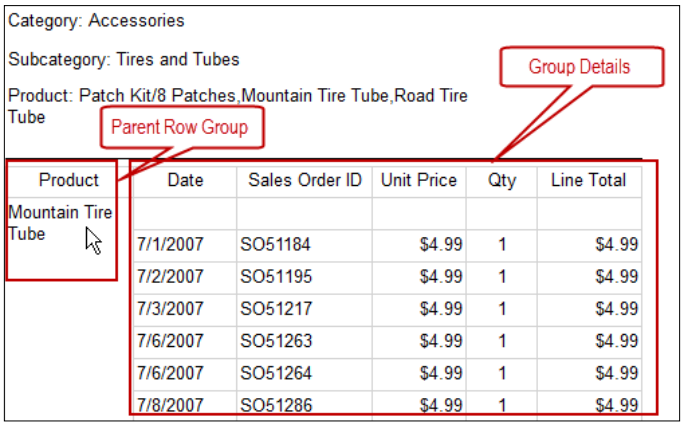
1. Right-click on the first column (**Date**), and navigate to **Add Group | Parent Group** (under **Row Group**) as shown in the following screenshot:



2. That displays the **Tablix group** dialog box. From the **Group by** drop-down list, let's select **[Product]** because that is how we want to group our data—by **Product**. Also, select the **Add group header** checkbox, which will add an extra row that we can use to display aggregates for each **Product** group:



A new column now appears in the first position of our table. A new row is also added on top of our detail data. Let's take a look at the grouping by **Product** that we just created:



Implementing Drilldown with row visibility

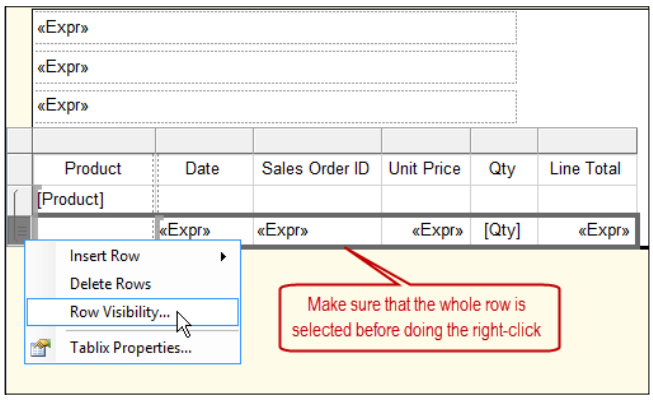
Implementing an expand/collapse control for groups or simple Drilldown in SSRS is relatively easy. It involves two steps:

1. Create group (which we already did recently).
2. Set the visibility property of the details of that group.

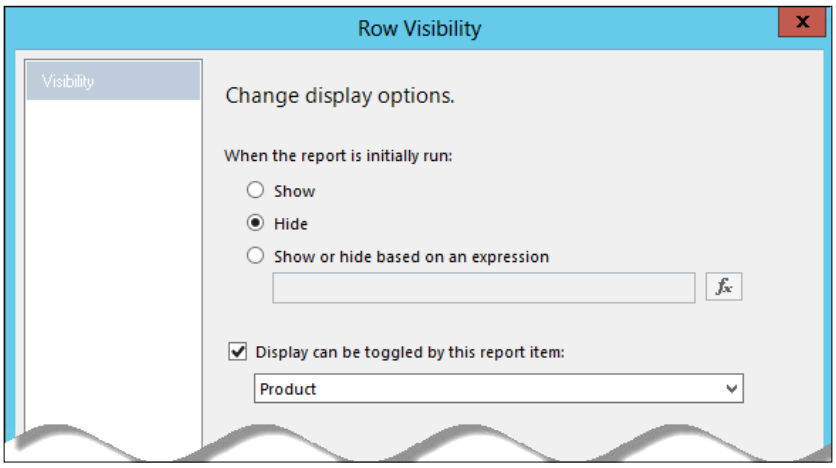
> Adding aggregate and group elements in your report, especially when dealing with huge datasets, may have some negative impact on the rendering performance of your report. If you cannot avoid aggregates and groupings on a huge dataset, consider creating a separate report for that purpose.

Perform the following steps:

1. Select and right-click on the detail row, then select **Row Visibility**:

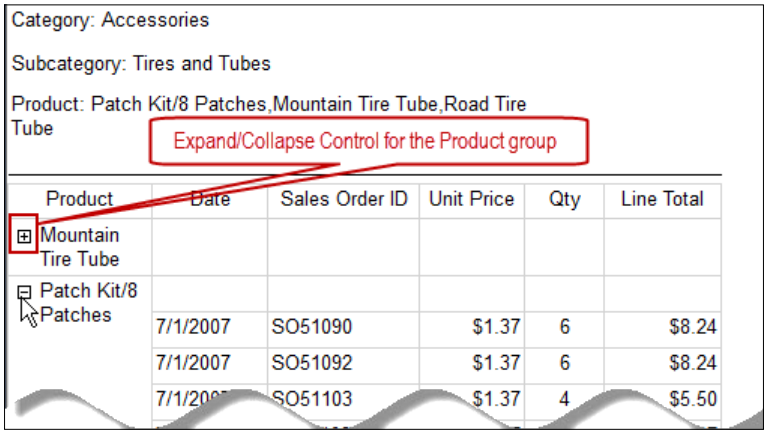


2. Doing so displays **Row Visibility** dialog box. Set the details to **Hide** when the report is initially running (this will give the drillthrough effect in our example).
3. Select **Display can be toggled by this report item**. Select **Product** from the drop-down list as shown in the following screenshot:



We can also use an expression to show or hide the details, depending on the condition we specify. But let's go ahead and just select **Hide**.

Now, let's take a look at our report with the added collapsible grouping:



Adding subtotals for each group

Sometimes we need to show the subtotal of each group. Let's create subtotals for each group in our report:

1. Let's create two new cells on our group header:

   Add the following expression on the cell just right above `Qty`:

   ```
   =Sum(Fields!Qty.Value)
   ```

   Add the following expression on the next cell to the right (`Line Total`):

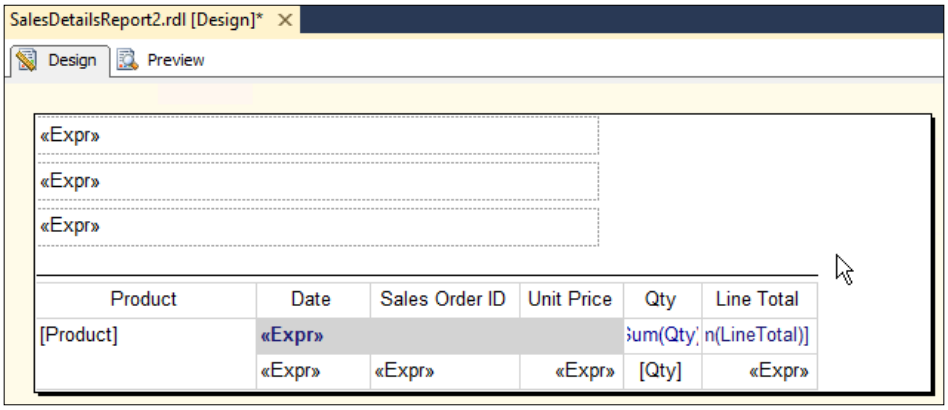
   ```
   =Sum(Fields!LineTotal.Value)
   ```

2. Set the **Text Box Properties** of both cells according to their types: `Number` for `Qty` and `Currency` for the `Line Total`. You can accomplish this by right-clicking on each cell then select **Text Box Properties** (`Number`).
3. Let's also select the three other cells: `Date`, `Sales Order ID`, and `Unit Price`, and right-click on it, then select **Merge Cells**. Add the following expression:

   ```
   ="Subtotal for " & (Fields!Product.Value)
   ```

4. Set the **Text Properties** of the merged cell to the appropriate format. We can be fancy with all the font and cell settings. Let's just keep it simple for the purpose of this demonstration, and make the expression bold.

This is how the designer looks like after we added the new cells and made a few changes:

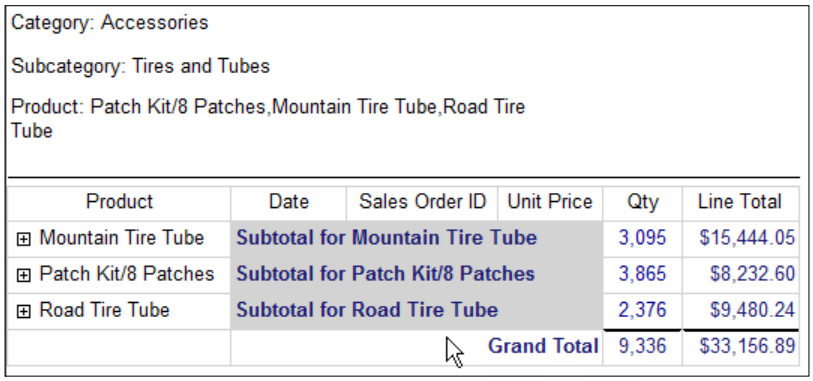


Adding the grand total

Tabular reports like this almost always require the grand total to be shown. Adding a grand total for all the groups is even simpler:

1. In the **Row Groups** pane select the **Product Group** item, then go to **Add Total | After**. Selecting **After** puts the grand total at the end of our report.
2. On the `Grand Total` row, we simply copy the same expressions we have for both `Qty` and `Line Total` from our group header. Let's delete the literal **Total** that was automatically added underneath our product group.

Let's run our report with the added subtotal and grand total:



Adding a page header

All the reports that we develop on SSRS are considered documents, although in digital form, and thereby follow the same rules of whatever record management procedures we have in place at our companies.

We need to add identity to our SSRS reports; hence, we need to put some predefined metadata in a header and footer. Let's do the header first:

1. On the report designer window, right-click on any empty space on the report surface, then go to **Insert | Page Header**. You can simply go to the menu and go to **Report | Add Page Header**.
2. Let's add two rectangles on the header. Place them side-by-side. The rectangle in SSRS can serve as a container for other objects (such as textbox, image, lines, and another rectangle).
3. On the right rectangle, drag a textbox object, which will hold our dummy address:

   ```
   Red Speed Bicycle LLC
   One Microsoft Way
   Redmond, WA 98052-6399
   ```

   Ok, that's not really a dummy address. You may remember that it's Microsoft's mailing address.

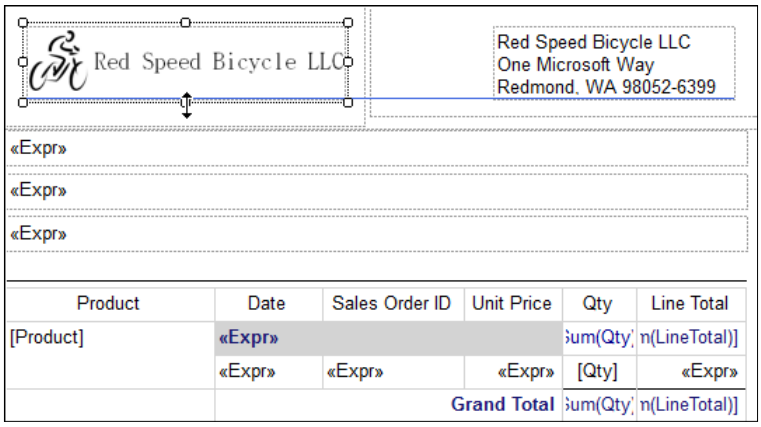
4. Set the textbox height to allow the three lines of our address. The `CanGrow` property of the textbox is set to `True` by default.

Adding an image in the header

A report header usually includes a logo or an image that identifies the brand or company. Let's add a photo to the header:

1. Drag an image object to the left rectangle in the header. That displays the **Image Properties** dialog box.
2. Select **Embedded** for the image source.
3. Click on the **Import** button and find the picture file named `Red_Speed_Bicycle_LLC_Logo.png`, which is included with this book. Click on **OK** when done.

4. Resize the picture appropriately.



You can add any elements you like to make the report look fancier. You can play around with combinations of expressions. We've achieved our purpose here.

Adding a page footer

Page footer is also important in our reports. If your company requires you to have some disclaimer or privacy notice written on the report, the footer is usually the right place to put that.

1. Right-click on any empty space on the report surface, then go to **Insert | Page Footer**. Just like adding the header, you can also add the footer from the **Report** menu item.
2. Let's add a textbox object in the middle of the footer, and on the textbox type our privacy notice: `Company Use Only. Not For Distribution`.

Adding a page number

Pagination is very important, especially, when the report is printed. Let's add a page number in the footer in the form of page x of y, where x is the current page and y is the total pages.

1. Let's add a another textbox in the footer on the left-hand corner and just beneath the first textbox that we created for our privacy notice.
2. Add this expression:

```
=Globals!PageNumber & " of " & Globals!TotalPages
```

Let's take a look at our report with the added page number and privacy notice in the footer:

	7/11/2007	SO51349	$3.99	1	$3.99
	7/12/2007	SO51361	$3.99	1	$3.99
	7/12/2007	SO51362	$3.99	1	$3.99
	7/14/2007	SO51391	$3.99	1	$3.99
	7/15/2007	SO51407	$3.99	1	$3.99
	7/15/2007	SO51419	$3.99	1	$3.99

Company Use Only. Not For Distribution.

1 of 83

Alternating row colors

You've probably seen tables with alternating row colors in some applications. The alternating row colors are normally added for readability purposes. You would notice that wide tables with multiple columns are easier to read when the colors of their rows are alternate.

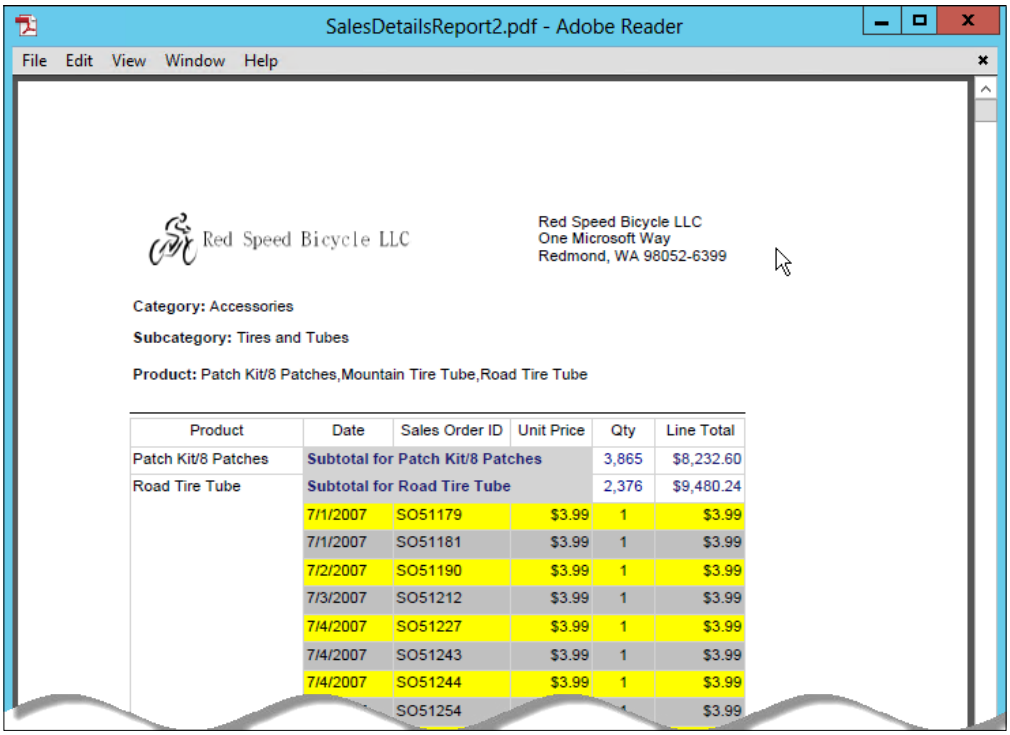
Let's make the row color alternate between two colors: silver and yellow.

1. Select the whole detail row and, on the **Properties** window, look for the `BackgroundColor` property and set it to an expression (from the drop-down list select **Expression**).
2. Add this expression:

```
=IIF(RowNumber(Nothing) Mod 2 = 0, "Silver", "Yellow")
```

> In previous example, the `IIF` function is a decision function that returns a Boolean Value (`True` or `False`) depending on the criteria (whether it's true or False). For more info about the `IIF` function, check this link: `http://msdn.microsoft.com/en-us/library/ms157328.aspx#DecisionFunctions`
>
> Moreover, the `RowNumber` function retrieves the row number of the current row. The `Nothing` keyword indicates that we want to start our row count at the top of our table. For more information about the `RowNumber` function, check this link: `http://msdn.microsoft.com/en-us/library/ms157328.aspx#RowNumber`
>
> The `MOD` is an arithmetic operator that returns the remainder (integer) of a division.

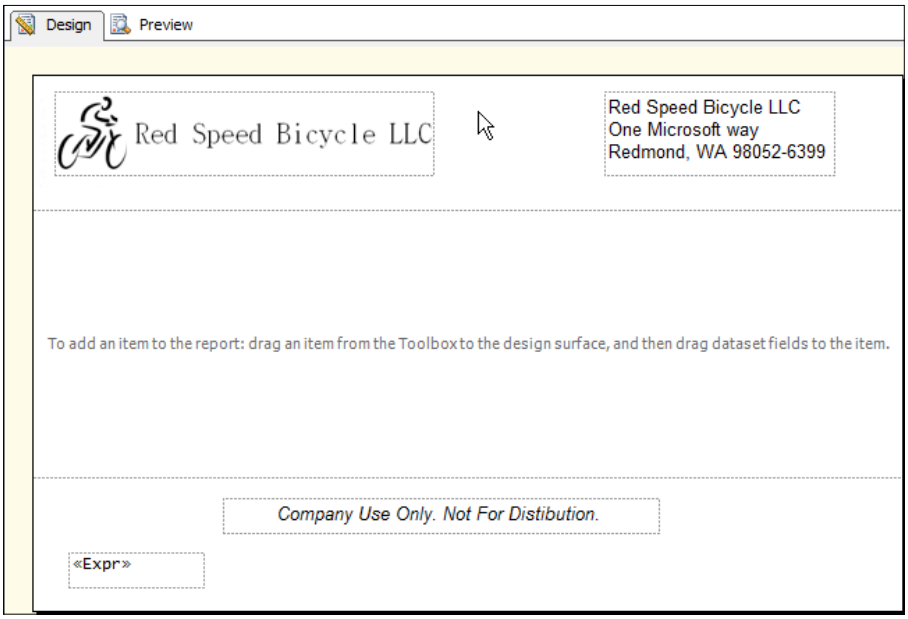
Let's take a look at our report in action:



Creating a report template

Using a report template is a good thing to practice in SSRS development. If we want to have that consistent look, at least in header and footer, we need to use a template.

1. Let's create a new report and name it as `SSRSBlueprintTemplate.rdl`.
2. Now create an empty report with header and footer. Add the same information and expressions we've added in our previous report:



3. Save the report file (`SSRSBlueprintTemplate.rdl`) at the location `C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies\ProjectItems\ReportProject`.

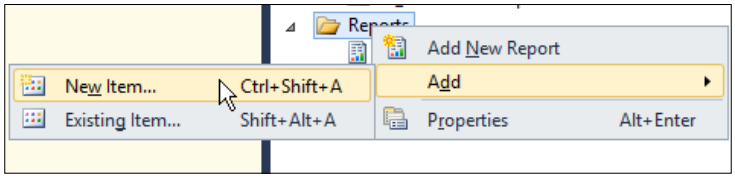
We want to make sure that we save the template to this location so that SSDT can detect and load it as an option in the **Add New Item** dialog box.

Using a custom report template

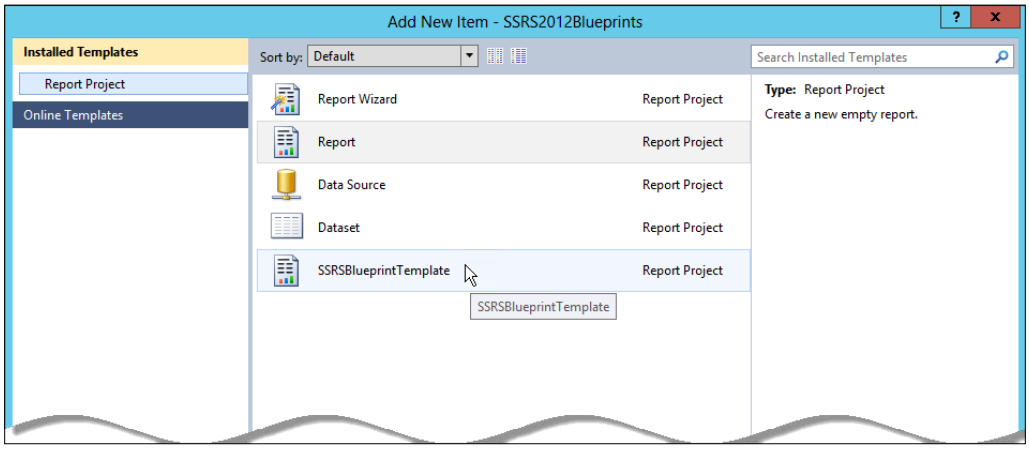
Custom templates save time. In fact, designing your report templates should be part of the preparation of every new SSRS projects.

Let's use the template that we just created:

1. In the **Solution Explorer**, select the `Report` folder and add a new report:



2. That displays the **Add New Item** dialog box. Simply select our custom template from the template option:



That's it. Create your report as usual. If you want your reports to have a consistent branding, using a template is the way to go. Plus, it's a time-saver if you're really concerned about your productivity.

But here's a caveat about templates. Any changes done in an instance of a template are not automatically pushed to the main template. So if you want to make some changes on one report, but ultimately want the same changes on all other reports, apply the changes to the main template first and save them, then use that template to all the reports as directed previously.

Summary

As we demonstrated in this chapter, we have a lot of options at our disposal to make our reports look pretty and consistent. The built-in functions are enough in most cases to build reports that suit our branding and other critical business requirements. Most importantly, we've learned how to customize various parts of our reports for clarity and consistency.

3
Getting Some Actions

We're now on the third leg of John Kirkland's SSRS journey. John's bosses are happy with the progress of the report projects. Now, they are seeing the true value of SQL Server Reporting Services 2012, that is, a solution that enables them to deliver meaningful data in a reasonable amount of time to those who need them.

The higher ups understand very well that John's skills on report development are still in their infancy and they are just so excited about John learning all this stuff quickly. John is just happy that he's contributing more value to the company.

As we move along, we will tackle how to add interactive elements to a report, and how one report can interact with another report.

In this chapter, we will learn the following:

- Subreport
- Drillthrough report
- Actions
- Tool tips
- Document Map
- Sorting

Let's prepare three reports for this chapter: **Sales Order Header report** (main report), **Sales Detail report** (detail report), and **Sales Territory report**.

We will create a fourth report at the end of this chapter to demonstrate additional features.

Creating the main report

Let's create the first report, the Sales Order Header report, using the template we created in the previous chapter.

Before proceeding, find the SQL file named `uspSalesOrderHeader.sql` (included with this book) and execute it to create a stored procedure called `dbo.uspSalesOrderHeader`. We use this to form the dataset for this report.

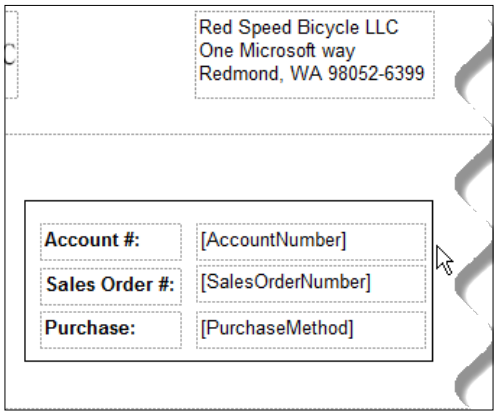
Our stored procedure has one parameter, `@SalesOrderNumber`, which calls for `SalesOrderID`. It's up to you whether or not you want to make your datasets shared or embedded. That's not the focus of our demonstration in this chapter.

1. Drag a list object and place it on the left-hand side of the body of the template. Add a solid border to it.
2. Then, add six textbox objects inside the listbox and arrange them into two columns with three rows in both columns.
3. In the first column, put the following labels: `Order Date`, `Salesperson`, and `Territory`.
4. Add the corresponding data fields in the second column. Format the fields as necessary.

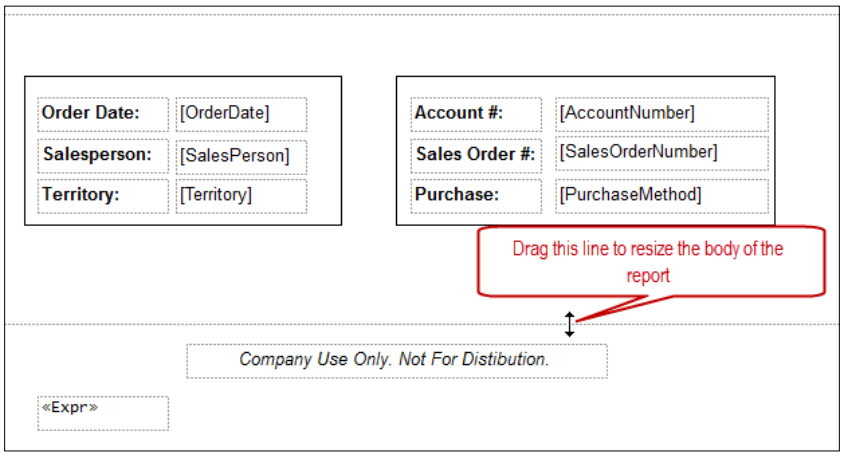
   As an option for the data fields, you can just drag the particular fields from the dataset and drop them next to the label box.

   That should look like this:

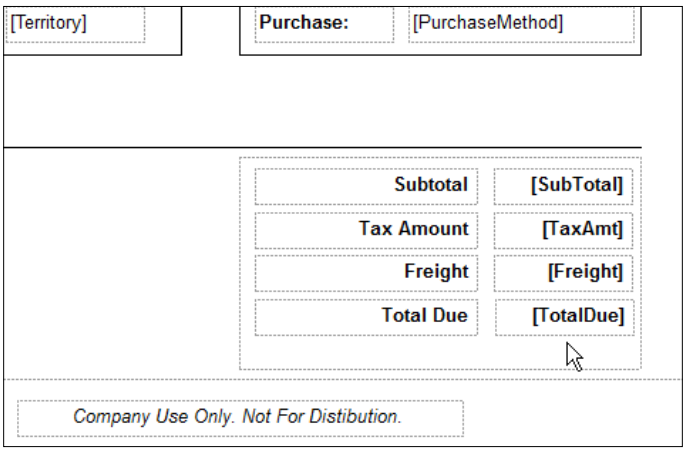
5. Let's add another list object on the right-hand side of the body of our template. Add the following textbox objects in one column and their corresponding data fields next to them: `Account #`, `Sales Order #`, and `Purchase` (method). This looks similar to the following screenshot:



6. Let's make room for another set of data at the bottom part of the body of our template. Resize the body by dragging down the footnote demarcation as shown in the following screenshot:



7. Let's add a third list object to the bottom-right side of the body of the template. Also, add a line, across the body of the report and place in on top of this listbox.
8. Add eight boxes inside the listbox and divide them into two columns with four textboxes in each column.
9. Add the following labels on the first column: `Subtotal`, `Tax Amount`, `Freight`, and `Total Due`. Of course, don't forget to add the corresponding data fields in the next column. That looks something like the following screenshot:



That's basically our main report. It looks like nothing special. You can add more fields and make it look better with a better design.

What we're trying to show here is how it's easy to put together various report objects and data to make a form, such as a sales order form.

Let's take a peek at our form:

Now that we've created our main report, let's move on to the next step—adding details to the sales header.

Creating a detail report

The next report that we're going to create is the Sales Order Details report. Find the SQL file named `uspSalesOrderDetail.sql` available with the code files for this book. Execute the script to create a stored procedure called `dbo.uspSalesOrderDetail`.

This report has the same single parameter as the `uspSalesOrderHeader` stored procedure, `@SalesOrderNumber (SalesOrderID)`. This will be the dataset that we'll use for our next report.

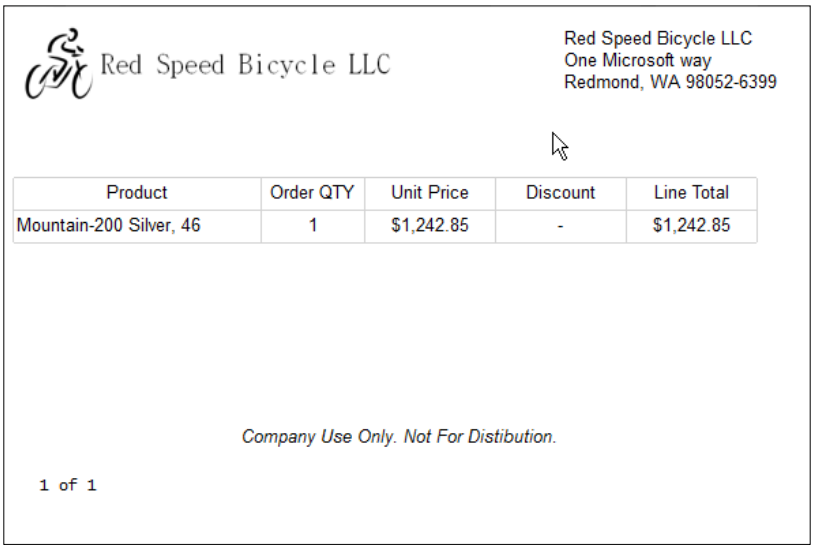
Although we're going to use this as a subreport that we embed into the main report, we're going to treat this just like any other standalone report.

There's a reason for that: **Reusability**.

We're going to keep this report as simple as possible. Let's name it **SalesOrderDetailReport**.

1. Let's add a table with five columns with the following data fields: `Name`, `OrderQty`, `UnitPrice`, `UnitPriceDiscount`, and `LineTotal`.
2. Name the data fields **Product**, **Order QTY**, **Unit Price**, **Discount**, and **Line Total** respectively.

   The report looks similar to report:



Adding a subreport

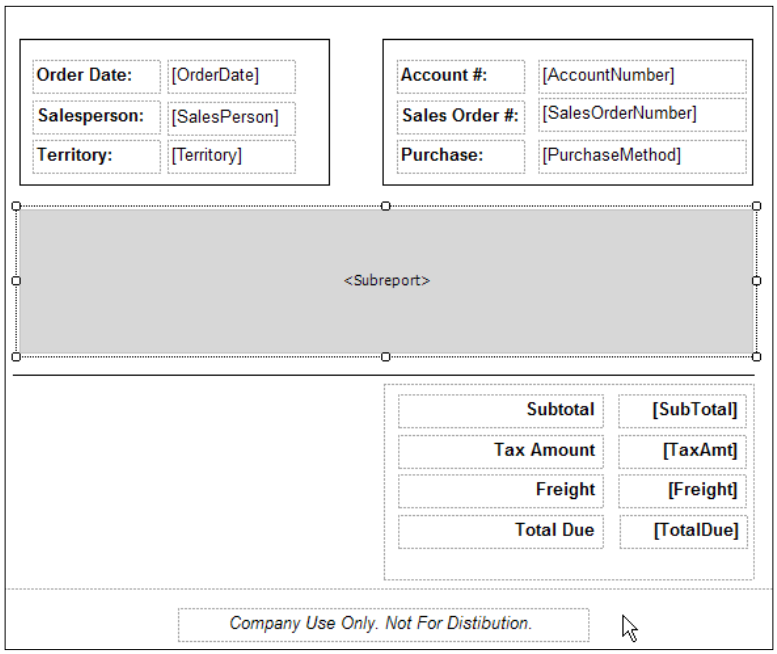
Our next step is to embed the Sales Order Detail report into the Sales Order Header report.

1. Open the first report, **Sales Order Header Report**, if you've already closed it.

> Underlying processes are needed to run a subreport on top of the process needed to run the main report. That usually means additional load on the report server. Additional loads like that impact the performance of your report. Remember that the subreport is run as a separate report.

2. Let's drag a **Subreport** object into the middle portion of the body of our report (on top of the line). Resize accordingly.

   You'll have something like the following:



3. Let's configure the subreport. Right-click on the subreport object and select **Subreport Properties**. Let's name our subreport `SubSalesOrderDetail`.
4. On the drop-down box that says **Use this report as subreport**, let's select **SalesOrderDetailReport**.
5. On the **Parameters** tab, click on the **Add** button. Add the `SalesOrderNumber` as the **Name** and `@SalesOrderNumber` as **Value**, then click on **OK**.

What we're doing here is passing the parameter of our main report, Sales Order Number, to the parameter of our subreport, which is also named **SalesOrderNumber**.

The key here is to make sure we're passing the correct parameter from the parent report to the child report. It's easy to get confused with scenarios where you have two or more subreports with different datasets. Remember the key: Parent-Child relationship.

On the report designer, you might get an error similar to `The subreport 'SubSalesOrderDetails' could not be found at the specified location. Please verify that the subreport has been published and that the name is correct.`

We usually get that error when we're passing the wrong parameter to the subreport (the child report). If you're still getting that said error even after fixing all the parameter issues, there's only one way to find out if the configurations were set correctly, that is, deploy!

This is how it looks like when our report is deployed:

Order Date: 3/1/2007
Salesperson: Tsvi Reiter
Territory: Southeast

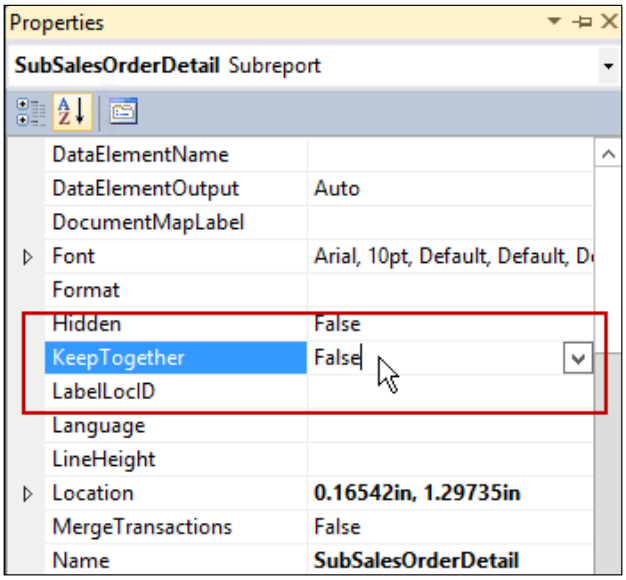
Account #: 10-4020-000368
Sales Order #: SO49524
Purchase: IN-STORE

Product	Order QTY	Unit Price	Discount	Line Total
Full-Finger Gloves, L	2	$22.79	-	$45.59
Mountain-200 Black, 42	2	$1,229.46	-	$2,458.92
Full-Finger Gloves, S	4	$22.79	-	$91.18
Full-Finger Gloves, M	4	$22.79	-	$91.18
Mountain-200 Black, 38	1	$1,229.46	-	$1,229.46

Subtotal	**$3,916.32**
Tax Amount	**$373.84**
Freight	**116.82**
Total Due	**$4,406.98**

Also, set the `KeepTogether` property of the subreport to `False` to avoid getting blank pages on rendered PDF, especially when the rows in the subreport exceed one page.

To do this, select the subreport in the main report; then in the **Properties** window, find the `KeepTogether` property and set it to `False`:



Creating a drillthrough or linked report

Let's create our third report and call it `SalesTerritoryReport`. Use the template we created in the previous chapter.

Find the SQL script called `uspSalesPersonSalesByYear.sql` and execute it to create a stored procedure called `dbo.uspSalesPersonSalesByYear`. This stored procedure has one parameter, that is, `@SalesTerritory`.

As you might have noticed, the query returns sales total for years 2005, 2006, 2007, and 2008 by territory, using the `PIVOT` function in T-SQL.

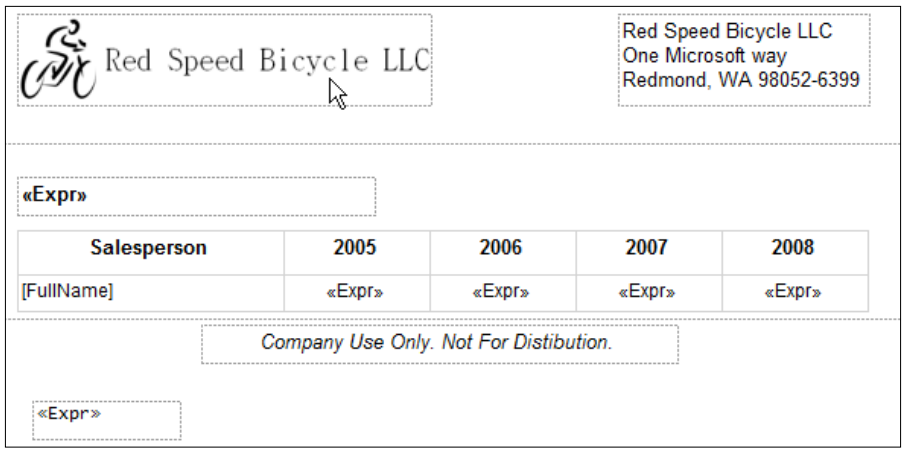
Again, let's keep this simple:

1. Drag a table object into the body of our report. We only need five columns for this report.

> The report designer added an `ID` prefix to our year columns (for example, ID2005) because field names must start with a letter from the alphabet.

2. Just above the left-hand corner of the table, add a textbox to hold the value of the parameter.

   You should have something like the following:



3. Let's create a separate dataset for the `@SalesTerritory` parameter. Name it `ds_rptSalesTerritory` or just anything you want.
4. Let's throw in a bonus exercise here: adding the string `ALL` as an option in the parameter.

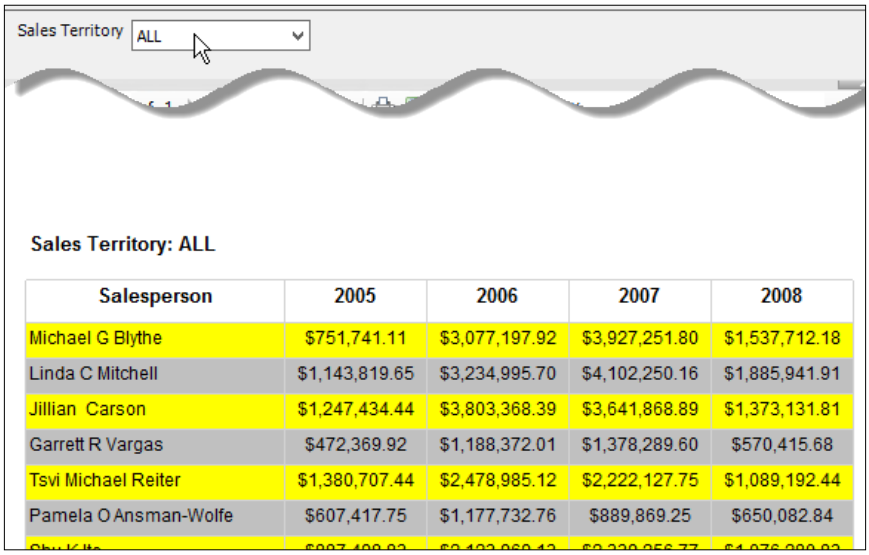
   Let's set the parameter dataset with this query:

```
Select 'ALL' as Name
UNION ALL
Select Name
From Sales.SalesTerritory
```

> As a best practice, always use `UNION ALL` unless you need the `UNION` functionality. `UNION ALL` avoids the overhead of the `SORT` (`Distinct`) operator. Read more about the execution plan in SQL Server.

I'll explain the whole `ALL` parameter thing in a second. For now, let's focus on our drillthrough or linked report.

This report now looks like the following illustration after applying some of the things we learned from the previous chapters:



Salesperson	2005	2006	2007	2008
Michael G Blythe	$751,741.11	$3,077,197.92	$3,927,251.80	$1,537,712.18
Linda C Mitchell	$1,143,819.65	$3,234,995.70	$4,102,250.16	$1,885,941.91
Jillian Carson	$1,247,434.44	$3,803,368.39	$3,641,868.89	$1,373,131.81
Garrett R Vargas	$472,369.92	$1,188,372.01	$1,378,289.60	$570,415.68
Tsvi Michael Reiter	$1,380,707.44	$2,478,985.12	$2,222,127.75	$1,089,192.44
Pamela O Ansman-Wolfe	$607,417.75	$1,177,732.76	$889,869.25	$650,082.84

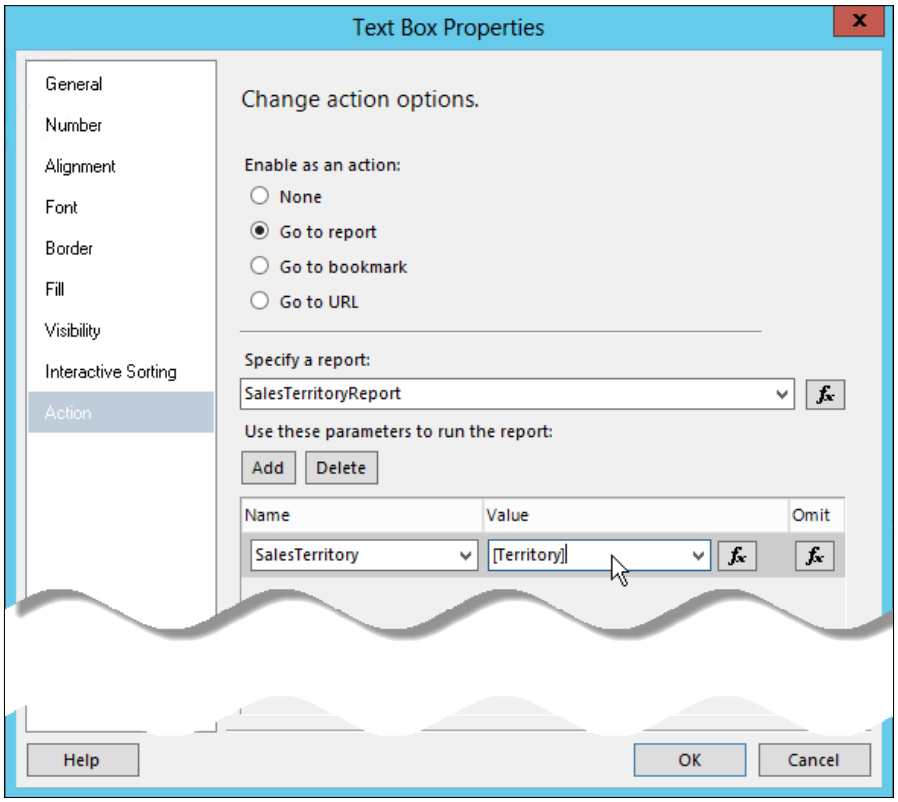
We can now set aside this report.

Now, let's go back to our main report, which is the **SalesOrderHeaderReport**.

5. Right-click on the **[Territory]** data field in the first listbox, then select **Text Box Properties**.
6. On the **Text Box Properties** dialog window, select the **Action** tab and enable the **Go to report** action.
7. Specify that we want the **SalesTerritotyReport** as the report we want to go to.
8. Then click on the **Add** button where it says **Use these parameters to run the report**.

   Our **Go to report** has only one parameter, which is called `SalesTerritory`. Let's put that under the **Name** section. For **Value**, let's select the data field **[Territory]** from our main report.

This is a simple parameter passing; from the main report to the go-to report.

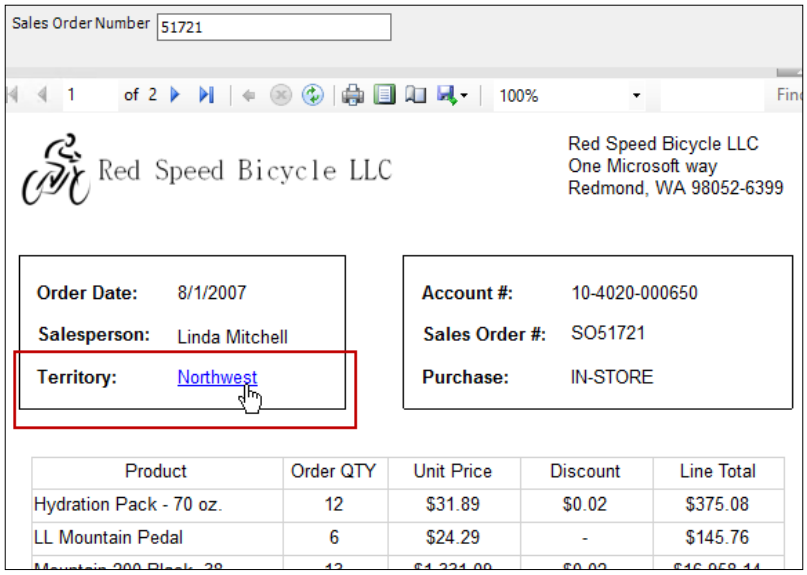


9. We'd also like to change the color of the font to blue and have it underlined, and make it appear as a hyperlink to signal that this data field is clickable.

   Click on **OK** when done.

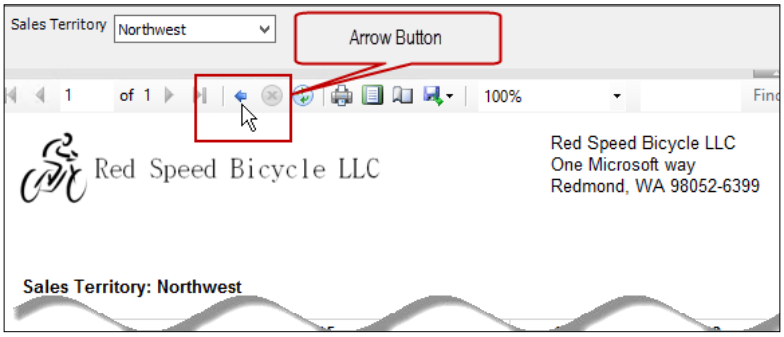
   Now, let's try our report with **Sales Order Number** `51721`.

You will notice that when you hover your cursor over **Territory Northwest**, you now have a hand cursor which indicates an action will occur when clicked:



10. Click on **Northwest** and that will open the **SalesTerritoryReport** and pass the territory value **Northwest** to the **Sales Territory** parameter:

    You can use the little arrow button on top of the window to return to the main report:



You can use the browser back arrow to go back to the main report when it's already deployed on the production.

The ALL parameter

There are other options on how to add `ALL` as a variable to return the whole dataset in a single parameter. Normally, you would do this on the dataset query:

```
SELECT 'ALL' as col1, NULL as col2, NULL, as col3
UNION ALL
SELECT col1, col2, col3
FROM…
```

Then allow the parameter to accept `NULL`.

You can also do what we've done here.

Take a look at the `WHERE` clause of our stored procedure `dbo.uspSalesPersonSalesByYear`.

```
WHERE ('ALL' IN (@SalesTerritory)) OR (pvt.SalesTerritory IN (@
SalesTerritory))
```

That's it.

If you use the `ALL` parameter in combination with a multivalued parameter, you need to use the `MultiValueParamSplit` function as explained in *Chapter 1, Let's Breakdown the Numbers*.

Adding a hyperlink (go to URL)

Adding URL links to a report is easy. Let's link our logo to `http://microsoft.com`.

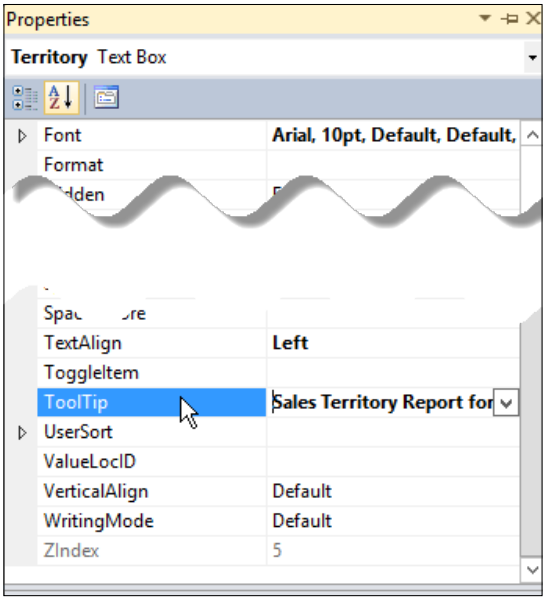
1. Right-click on the logo and select **Image Properties**.
2. In the **Image Properties** dialog window, select the **Action** tab.
3. Let's enable the **Go to URL** action. Then, let's put the URL where indicated.
4. That's it. Run the report and the logo will send you to `www.microsoft.com` when you click on it.

Adding a tool tip

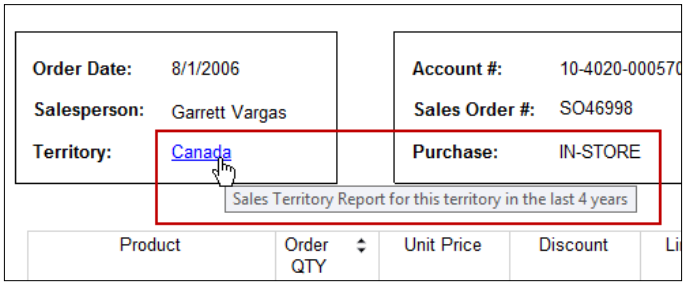
Tool tips help our report consumers navigate our reports. For example, we want to let them know what that `Northwest` call for action is all about, or where is that leading them to.

Let's add a tool tip to the `[Territory]` field.

Click on the `[Territory]` field and on the **Properties** window, look for the **ToolTip** property. Let's write: **Sales Territory Report for this territory in the last 4 years**.



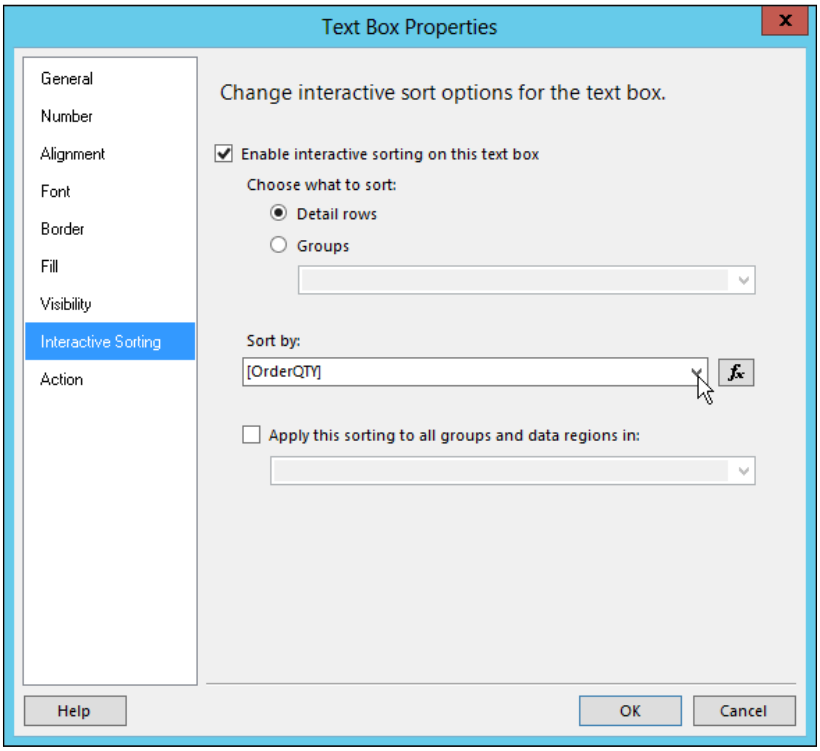
Let's save our report and run it; and just like that, we've got the tool tip!



Interactive sorting

1. Adding interactive sorting is also a breeze. Let's go back to our SalesOrderDetailReport and add interactive sorting on the **Order QTY** column.
2. Right-click on the header of the **Order QTY** column, then select **Text Box Properties**.
3. Select the **Interactive Sorting** tab. Select the checkbox where it says **Enable interactive sorting on this text box**, and select **Detail rows** to sort.
4. On the drop-down list where it says **Sort by**, select **[OrderQTY]**.

   You can also sort by groups as indicated in the other options.



5. Click on **OK** when done.

Now, let's try our interactive sorting. Here's the before and after shot of the pre-sort and after sort:

Product	Order QTY
Hydration Pack - 70 oz.	12
LL Mountain Pedal	6
Mountain-200 Black, 38	13
Long-Sleeve Logo Jersey, XL	3
LL Mountain Frame - Black, 48	1
LL Mountain Seat/Saddle	3
LL Bottom Bracket	7
Women's Mountain Shorts, L	7
Mountain-500 Black, 48	3
Mountain-500 Black, 42	1

Product	Order QTY
Water Bottle - 30 oz.	24
Long-Sleeve Logo Jersey, L	23
AWC Logo Cap	19
Bike Wash - Dissolver	16
LL Mountain Handlebars	16
Sport-100 Helmet, Blue	15
Sport-100 Helmet, Black	14
ML Mountain Handlebars	14
Mountain-200 Black, 38	13
Short-Sleeve Classic Jersey, S	13

Adding a Document Map

Document Map is a neat feature in SSRS. For the demonstration of this feature, let's create a new report. Yes, this is already the fourth report in this chapter.

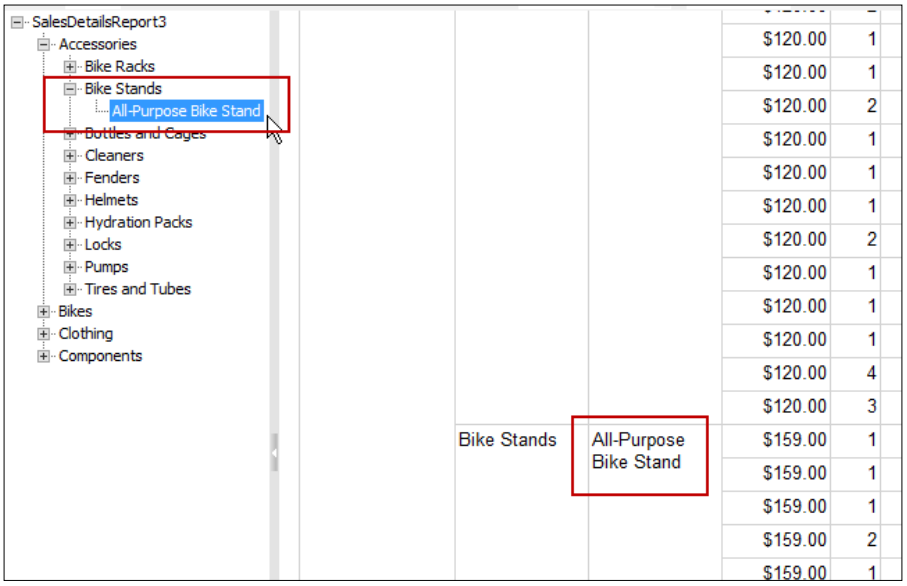
1. Let's name this report SalesDetailsReport3. Create an embedded dataset and use that as our source for this report. Look for the SQL file called `SalesDetailsReport3.sql` that comes with code files for this chapter.
2. Add row grouping on **Category** | **Subcategory** | **Product**. Don't drilldown from **Category** to **Product**. Leave the grouping as is.

   Our report should look similar to this:

Category	Subcat	Product	Unit Price	Qty	Line Total
Accessories	Bike Racks	Hitch Rack - 4-Bike	$72.00	5	$360.00
			$69.60	11	$750.29
			$72.00	9	$648.00
			$66.00	18	$1,128.60
			$69.60	14	$954.91
			$72.00	4	$288.00
			$72.00		$144.00

3. In the **Row Groups** window, right-click on the **Category** group and select **Group Properties**.
4. Select the **Advanced** tab. On **Document Map**, select **[Category]**. Do the same with **[Subcat]** and **[Product]**.

   And, the Document Map looks like this in action:



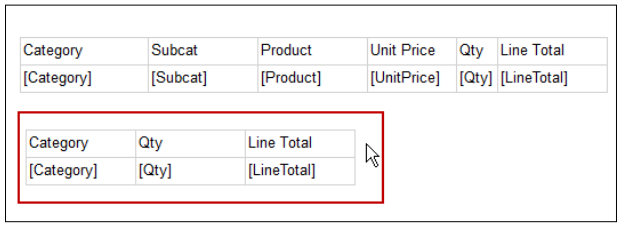
5. Clicking on **All-Purpose Bike Stand** will bring you to where you can find that product in the report.

Creating a bookmark

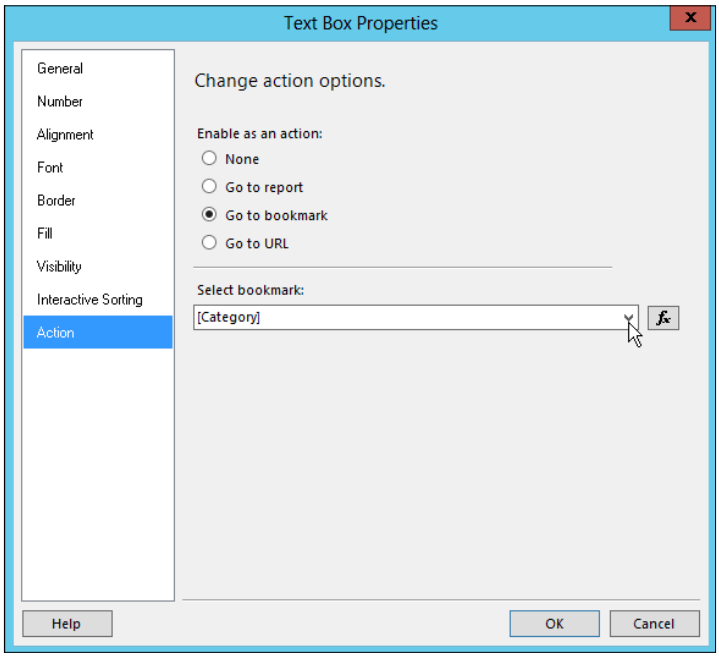
A bookmark in SSRS is another action that helps report consumers to easily navigate a report with a huge dataset. Let's create a bookmark by performing the following steps:

1. Let's add another table to our SalesDetailsReport3. Find the T-SQL script named `BookmarkReport.sql` and create a dataset based on that script on the same report.

Our dataset for this second table has three columns. It looks like the following report:

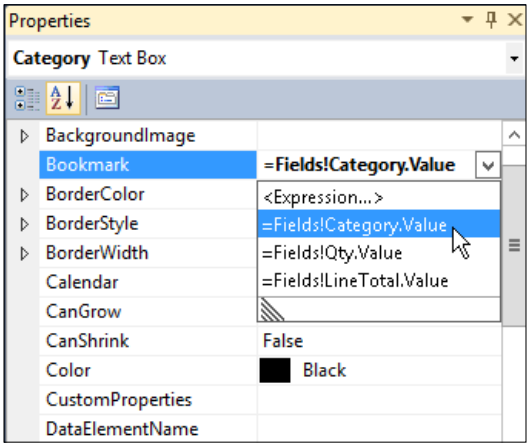


2. Right-click on the **[Category]** data field of the first table, and then select **Text Box Properties**. On the **Text Box Properties** dialog window, select the **Action** tab.
3. Select **Go to bookmark** as the action. Select **[Category]** as bookmark where it says **Select bookmark**.
4. Also, change the color and style of the Category font to make it appear as if it's a hyperlink (color blue, underlined). Again, this is to indicate that we're calling for action.
5. Click on **OK** when done:

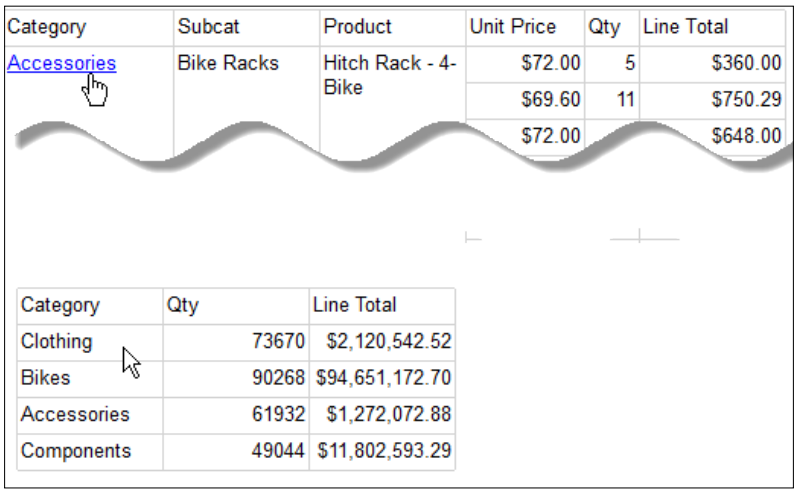


Now, let's move to the second table.

6. Click on the **[Category]** field, and then go to the **Properties** window. Look for the **Bookmark** property. Then select or set the property to `=Fields!Category.Value` as shown in the following screenshot:



We're done. Let's test our bookmark.



Clicking on the hyperlink on the first table moves our screen focus to the second table. Success!

Summary

We've learned how to interact with the report elements in this chapter. We've also learned to make a report interact with another report. There are many options in SSRS that we can avail of to make the experience of the report consumers as convenient as possible. Navigating within a report and across different reports is much easier when the report consumers are equipped with the features we've demonstrated in this chapter.

4
Everything Looks Better with Charts

John Kirkland is now becoming the office's report hero. People at the Sales department love the guy. Why wouldn't they? John solved most of their data crunching problems. He provided them with an easy way to build and author reports that are vital to their daily functions.

Gone are the days when the sales people had to go through different levels of IT functions just to get hold of their data. Sometimes, they had to wait a little bit longer than necessary, because, their IT people have more pressing priorities.

John eliminated most of the daily routine tasks involved in gathering critical data and putting them together to convey meaningful information. The report request-to-delivery turnaround has never been quicker. John is happy. The report consumers are happy. This could only mean one thing—more requests for more advanced reports are going to land on John's lap. And that's what exactly happened.

The Sales Director loved the sales reports that one of his mid-managers showed him in their last monthly meeting. Data summaries, aggregates, and groupings make their sales numbers easier to digest. But they want more.

The purpose of their monthly sales meeting is not really to ruminate on exact numbers or on every dollar down to the cent they earned on every product they sold. Sometimes, they just want to have a 10,000 feet view of their overall sales performance.

John knew exactly how to accomplish this.

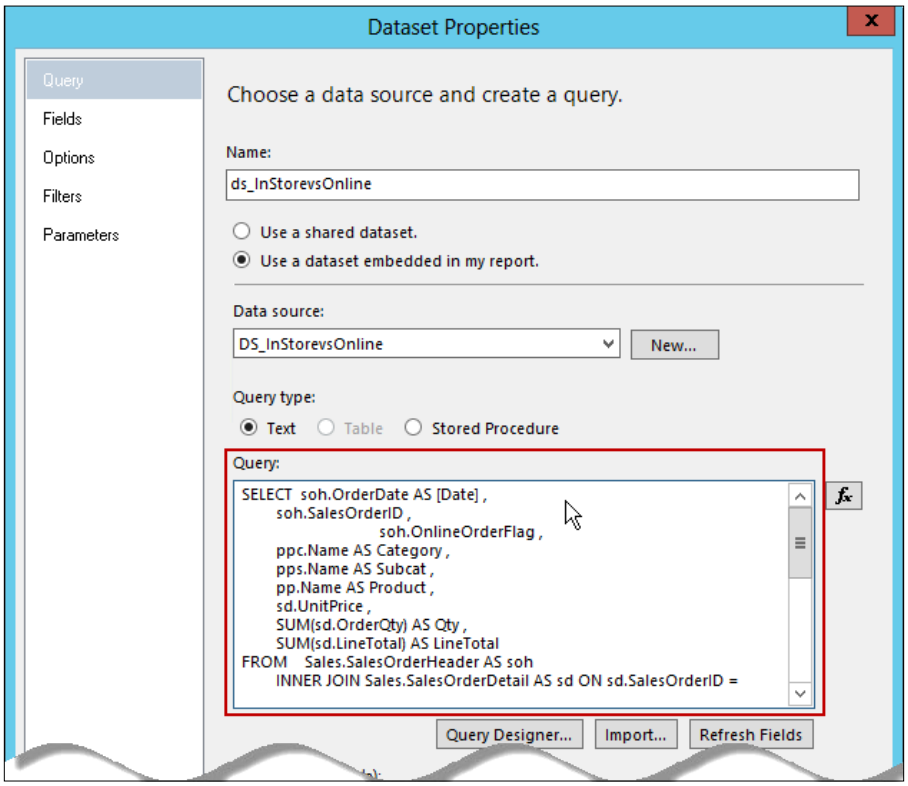
In this chapter, let's tackle the following:

- Chart
- Data Bars
- Sparklines
- Gauges
- Indicators

Creating a simple chart

Find the T-SQL script included in this chapter; and build our dataset based on that script. Look for the `In-StorevsOnlinePurchases.sql` script that comes with the code files for this book. If you haven't changed the project data source from the previous chapter, you can use the same to connect to the `AdventureWorks2012` database.

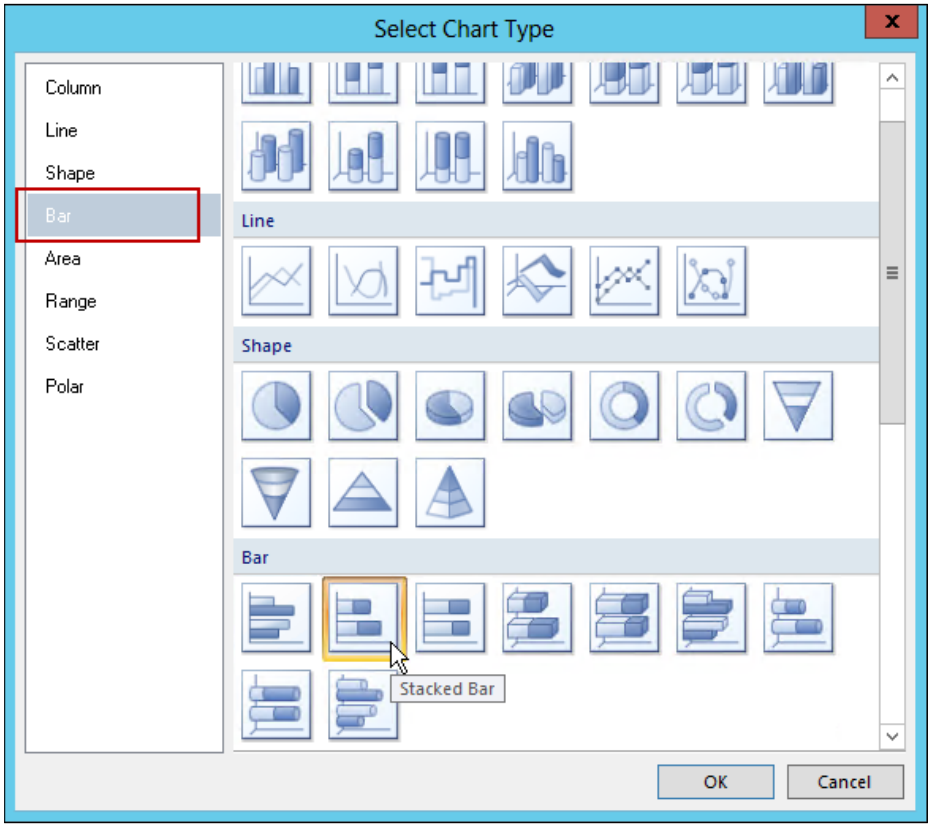
Build an embedded dataset to make it simple. Just copy the script into the **Dataset Properties** query window as shown in the following screenshot:



Now that we have our dataset, create a new report using our template. Let's create a chart report called **In-Store versus Online Sales**.

You probably have an idea of what this report is about based on the title. That's one thing to consider in your design practice. Create report titles that are intuitive as much as possible. You'll wonder how this can really help the report consumers.

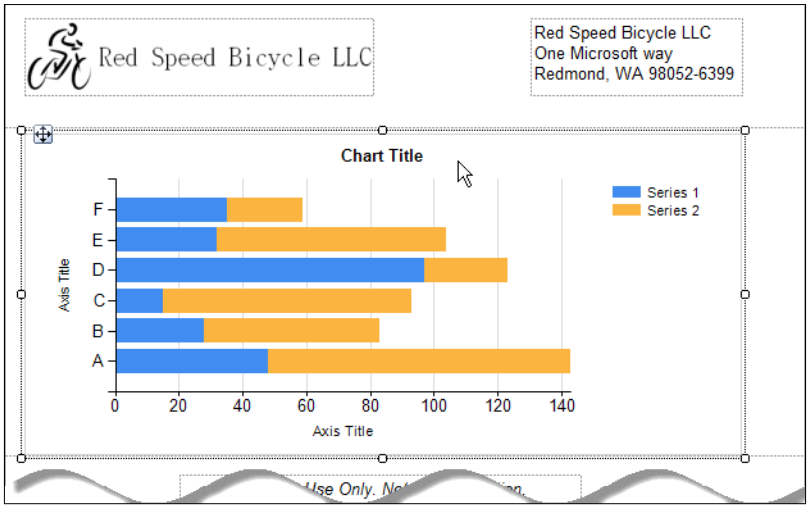
Drag a graph object into the report designer. That displays the **Select Chart Type** dialog box:



Let's select the **Stacked Bar** icon.

Since we're trying to show a couple of series in this report, what's sold In-Store versus Online, we're using the stacked bar.

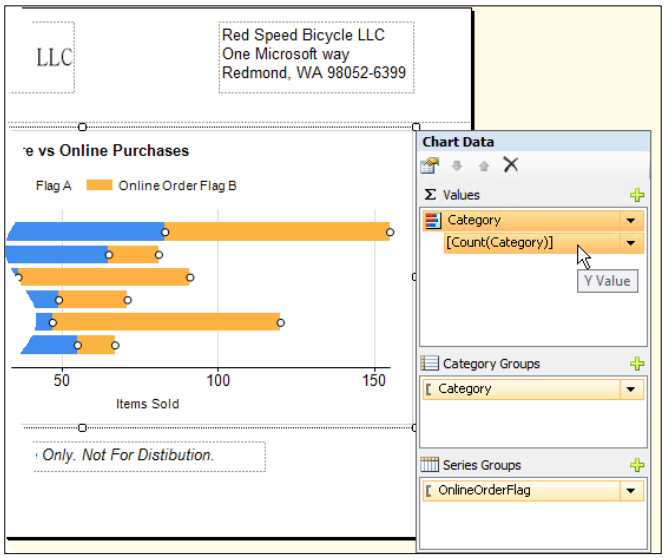
Resize the chart so that it fills the whole body of the report. With the stacked bar chart, our report designer looks like the following illustration:



Adding data to a chart

What we're going to do here is look at the relative percentage of product categories that are sold in store and online. If you noticed we have an `OnlineOrderFlag` column in our dataset. A product is either sold in store or online.

To add data to the chart, click in the chart area to display the **Chart Data** dialog box.

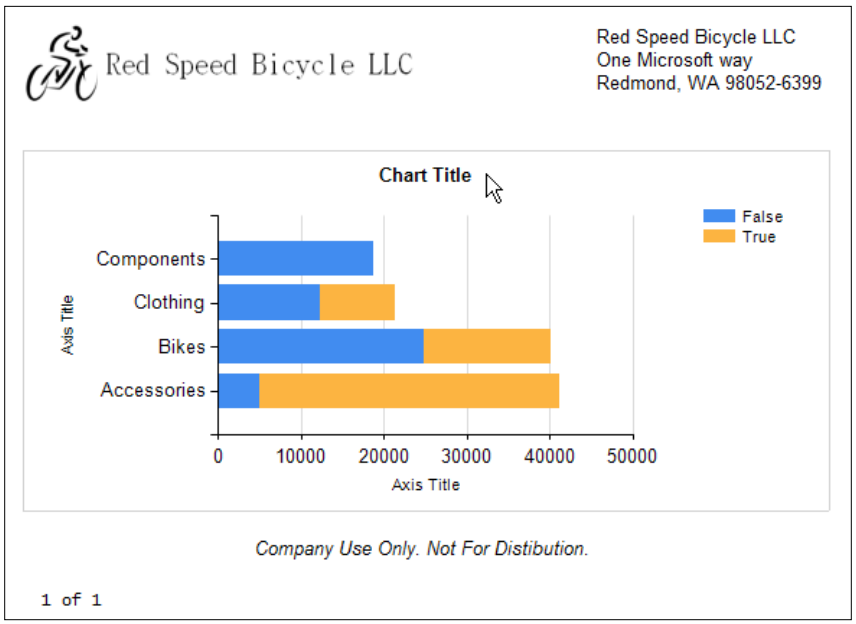


In the previous illustration, we put the `Category` field of our dataset as our chart data value. The value aggregate **[Count(Category)]** just below that indicates that we're doing a count operation on that field.

We can do other aggregate functions such as `Average`, `Sum`, `Max`, `Min`, and other aggregates. Choose any function according to your aggregate requirements.

We're using the `OnlineOrderFlag` as the series for this chart.

Let's preview our chart with the supplied data:



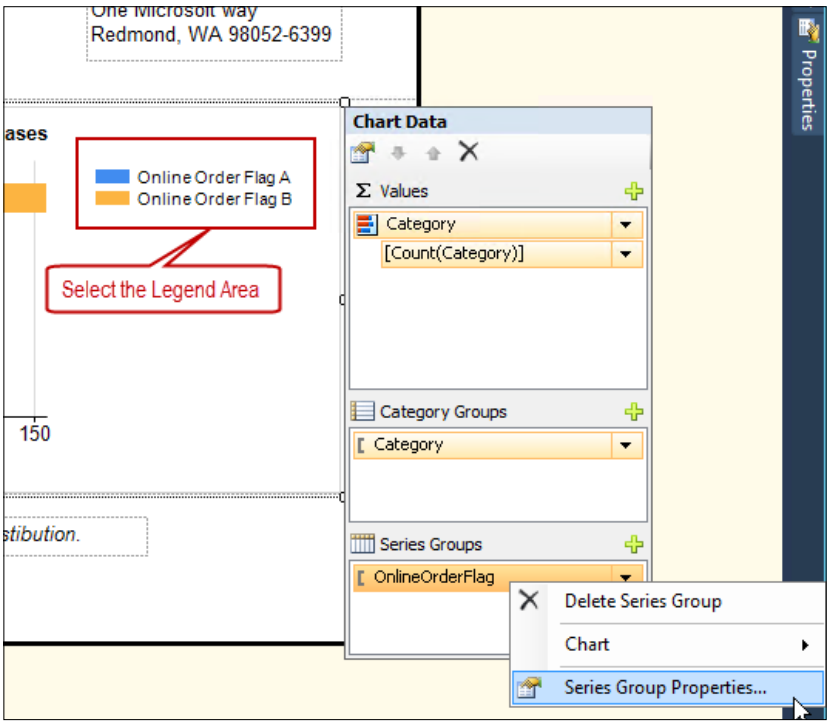
Formatting the chart

If we take a look at the preceding chart, we're missing some things. We do not have a **Chart Title** and **Axis Title**. We're going to fix that.

There are two ways of accomplishing this. Right-clicking on **Chart Title** displays the context menu where you can access **Title Properties**. Here, we can set the title, visibility, font, action, position and others. We can also simply double-click on **Chart Title** to set a title. We can do the same with **Axis Title**.

If you also noticed, the legend, `False` and `True`, is not really intuitive. The legend is showing `False` and `True` values because our `OnlineOrderFlag` field, which our series is based on, is of type BIT, either `0` or `1` (`False` or `True`).

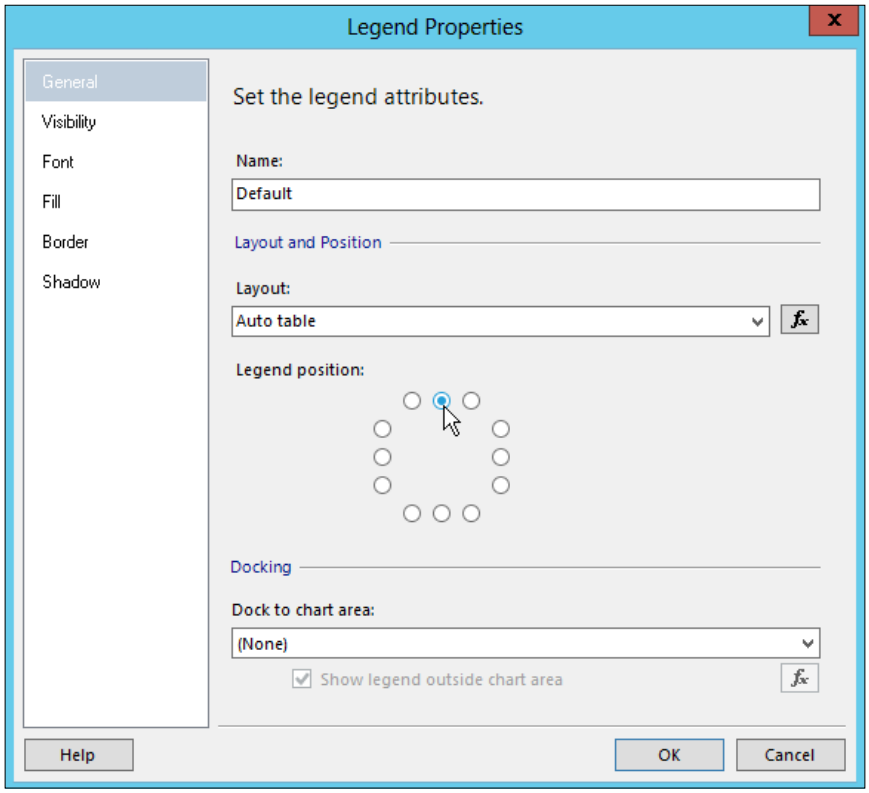
To change the label of the legend, click on the legend to display the **Chart Data** configuration menu. Right-click on the **OnlineOrderFlag** under **Series Group** and select the **Series Group Properties** option, as shown in the following screenshot:



On the **Series Group Properties** dialog box click on the expression button next to the drop-down textbox of the label and enter the following expression:

```
=IIF(Fields!OnlineOrderFlag.Value = False, "In-Store", "Online")
```

To change the position of the legend, click on the legend, right-click on it and select **Legend Properties**. The following dialog box opens:



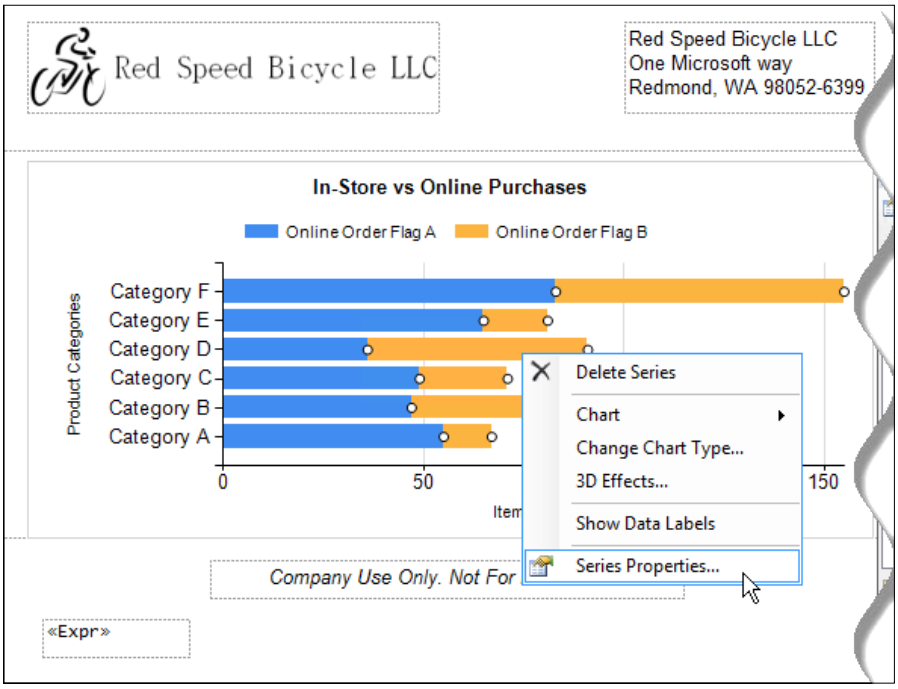
Select the radio button in the middle-top to indicate that we want the legend to display just underneath the chart title.

Adding a tool tip

So far, we got what we want in this report. We have a chart that shows the proportion between In-Store and Online sales. The bars show the comparative percentage between the two purchase methods, visually.

But we still want to get some numbers on the chart, such as the number of products sold and their dollar value, without cluttering the chart. We can achieve this with the use of the tool tip.

Right-click on the chart series and select **Series Properties** as shown in the following screenshot:

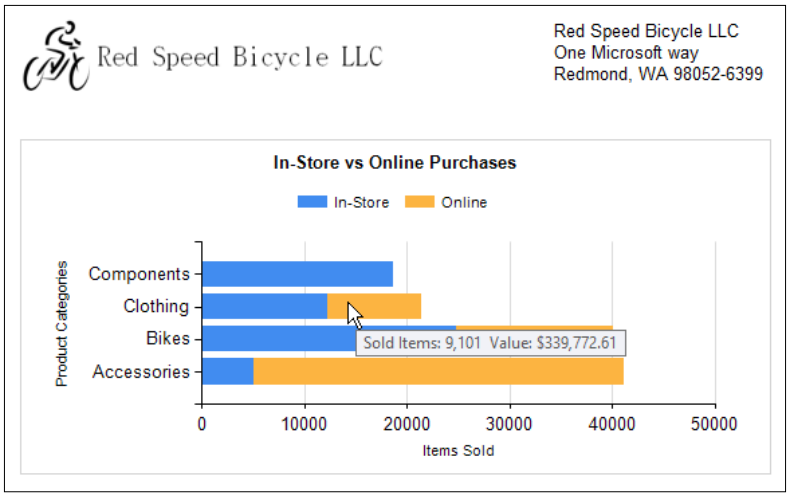


On the **Series Properties** dialog box, click on the expression button next to the drop-down textbox of the tool tip and enter the following expression:

```
=CStr("Sold Items: " + FormatNumber(Count(Fields!Category.Value),0) + "
Value: " + (FormatCurrency(Sum(Fields!LineTotal.Value),2)))
```

The `CStr` function converts the argument to string. We cannot concatenate a string and an integer. So, we need that `CStr` function to convert the category count returned by the `Count` function into string.

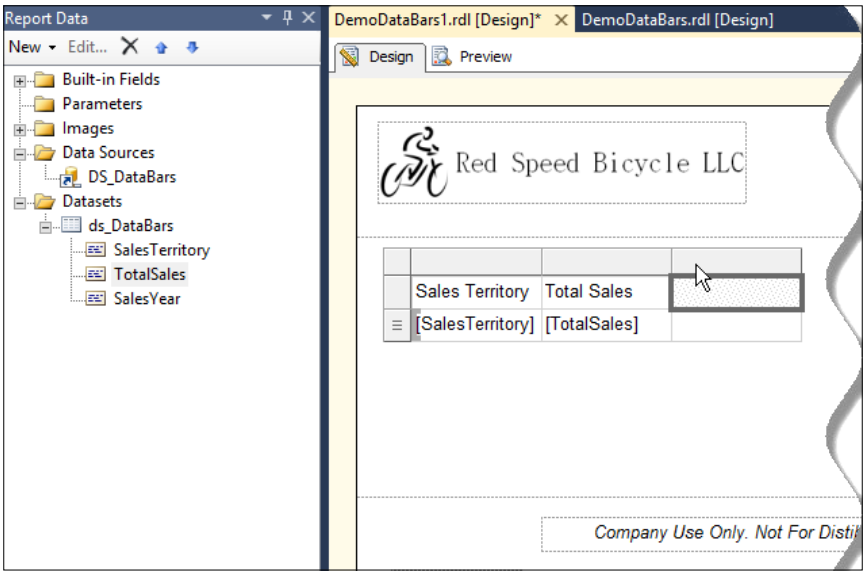
Presto! We've added another layer of data without actually adding any clutter to the chart itself. Now, if we hover our mouse over any of the series or bar, a tool tip shows the number of items sold, and the dollar value for that category:



Embedding data bars in a tablix

Let's create a new report. Name it whatever you want. Find the T-SQL script called `2008SalesTerritoryComparison.sql` and create the dataset based on that. Name the dataset `ds_DataBars`. We're going to use this name in an expression in the next section, so make sure you remember it.

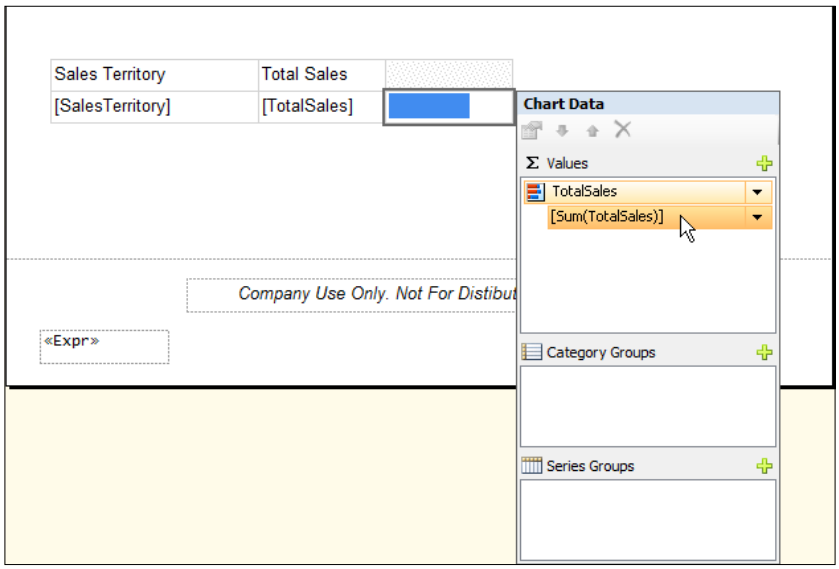
Drag the `SalesTerritory` and `TotalSales` fields into the tablix as shown in the following screenshot:



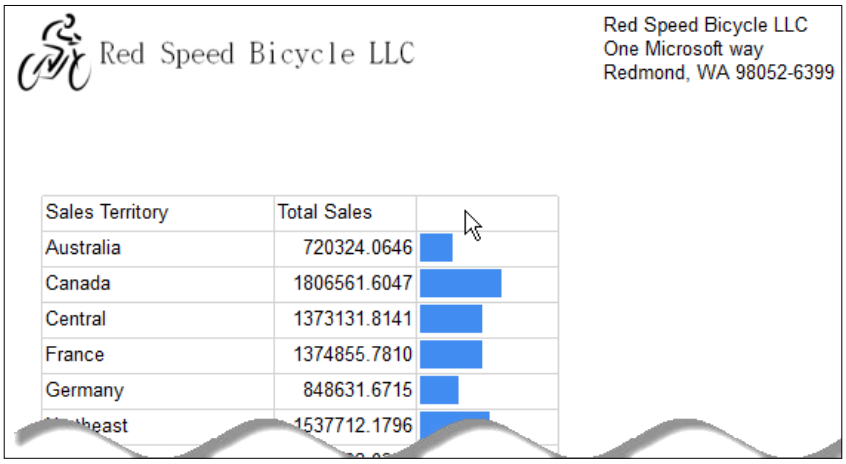
What we're going to look at here is the sales comparison among the territories, and see the distribution of dollars earned company wide.

Let's add a new column, if it's not already added. Drag a data bar object into the detail field of the third column. Select stacked bar on the **Select Data Bar Type** dialog box.

Now, for the value of our chart data, select `TotalSales`. Make sure you aggregate the value by `SUM` if it's not already selected:



Let's preview our report and check if we've achieved what we wanted:

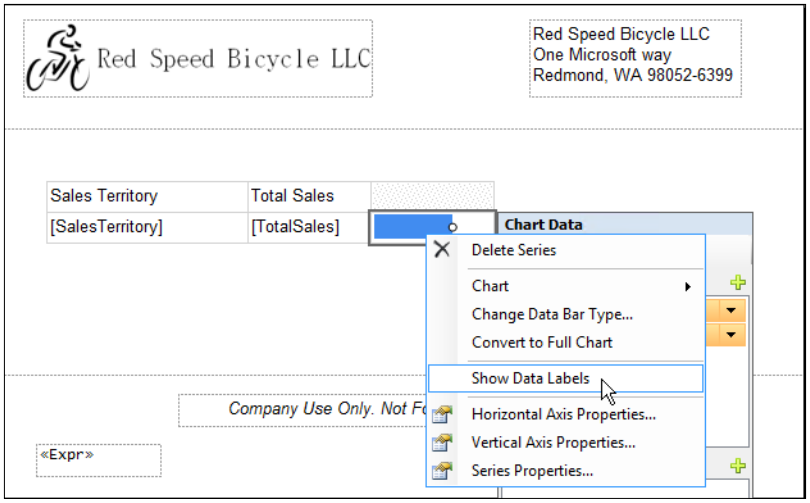


Showing a data label

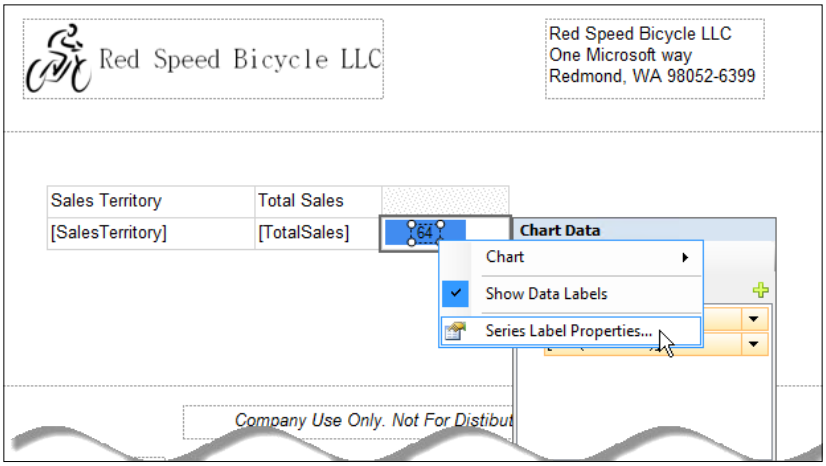
Aside from formatting, our report is still missing something. Remember that what we want to see is a distribution of sales value among the territories. The bars do not really tell that story. We want to see the percentage distribution.

Let's do just that.

Select the bar, and then right-click on it to display the **Series Properties** context menu. Select **Show Data Labels** as shown in the following screenshot:



When the label appears on the bar, select and right-click on it to display its context menu. Select **Series Label Properties** as shown in the following screenshot:

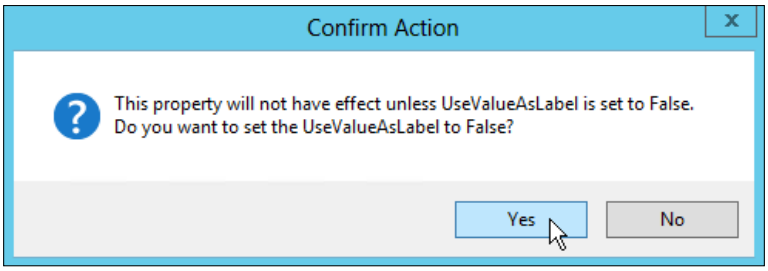


On the **Series Label Properties** dialog box, click on the expression button next to Label data drop-down textbox, and enter the following expression:

```
=Fields!TotalSales.Value/Sum(Fields!TotalSales.Value, "ds_DataBars")
```

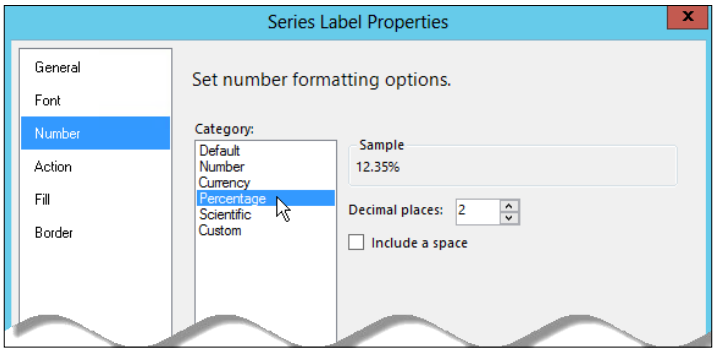
Please take note of the parameter `ds_DataBars` in the expression. That is the name of your report dataset. If you have a different name for the dataset, use that instead.

Now, if you click on **OK**, a **Confirm Action** dialog box is displayed asking you if you want to set the `UseValueAsLabel` property of the `Series Label` to `False`:



By clicking on **Yes**, we're essentially telling the report designer "yes, we want to use this expression instead of the value of the field (`SalesTotal`) as the label".

Format the value of our expression as a percentage by selecting **Percentage** from the **Category** drop-down list:



After a few format settings, the report now looks something like the following illustration, showing the percentage of each sales territory against all other territories:

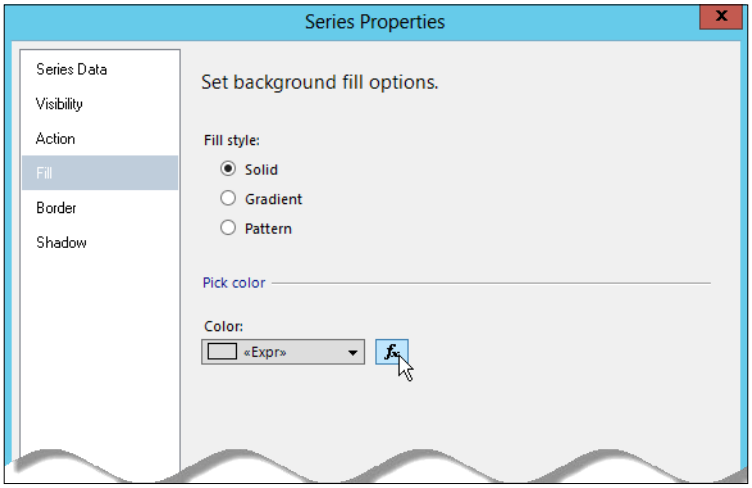
Sales Territory	Total Sales	Approx %
Germany	$848,631.67	5.41%
Australia	$720,324.06	4.59%
United Kingdom	$1,808,043.26	11.53%
Northeast	$1,537,712.18	9.81%
Southwest	$2,962,222.74	18.89%
Canada	$1,806,561.60	11.52%
France	$1,374,855.78	8.77%
Northwest	$2,162,203.03	13.79%
Southeast	$1,089,192.44	6.95%
Central	$1,373,131.81	8.76%

Setting the bar color conditionally

Aside from the common formatting that we've already seen in the previous chapters, you may notice that our report here has different bar colors (blue, red, and green).

- **Red**: It shows the least earning
- **Green**: It shows the biggest earning
- **Blue**: It shows the earning between the minimum and maximum

For accomplishing this, select the bar and right-click on it to display its context menu. From the context menu select **Series Properties**.



When the **Series Properties** dialog box is displayed, select the **Fill** tab. On the **Pick Color** section of the dialog box, click on the expression button next to the **Color** drop-down list, and enter the following expression:

```
=IIF(Fields!TotalSales.Value = Min((Fields!TotalSales.Value), "ds_
DataBars"),"Red", IIF(Fields!TotalSales.Value = Max((Fields!TotalSales.
Value), "ds_DataBars"),"Green","Blue"))
```

What we're doing in this expression is testing if the current `TotalSales` value is the least in the dataset, then if it is, we want to change the color of the bar to red. Green, if it is the highest value. Otherwise, the color bar is blue.

This is a quick way of showing the least and best performing sales territory in a single glance without really digging deeper into the data.

Adding sparklines in the tablix

Find the T-SQL script called `2006SalesPerformanceReport.sql`. Build the dataset using this script.

Drag a table object into the designer. Add the fields `Sales Territory`, `Category`, and `Total Sales` to the table.

Sparklines only display aggregated data. They won't work for a single detail. But just like the data bars, sparklines can display multiple series also; in our case, monthly sales for every category.

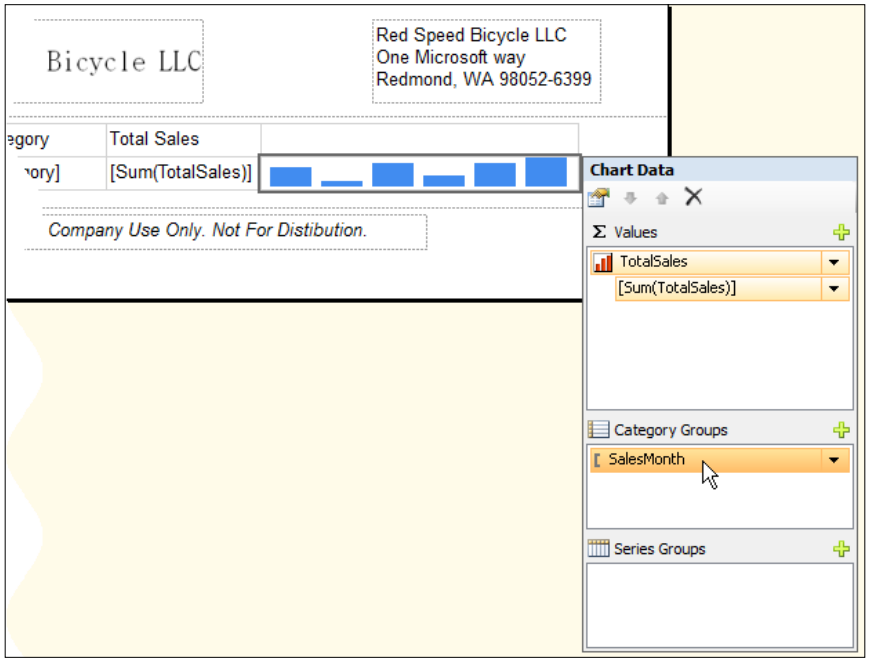
Let's group the `Sales Territory` field into a row group. Then, make the `Category` field as a child group to the `Sales Territory` group. Then, aggregate (`Add Total`) the `Total Sales`.

Format the table to look similar to the following illustration:

Add a new column to the table, and then drag a sparkline object to that column. For the sparkline type, select the column.

For our Chart Data value, let's add `TotalSales` and aggregate it (`Sum`). Let's use `SalesMonth` as our `Category` group.

The reason we have the `SalesMonth` on the `Category` group is that we want the sparkline to use it as its multiple series (sales per month). To illustrate that:

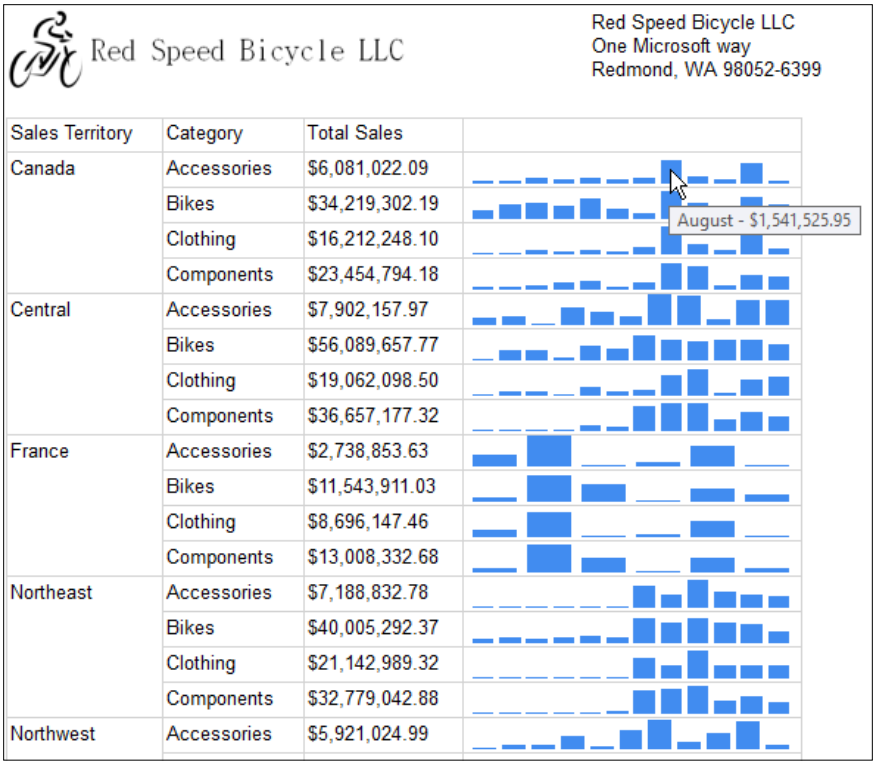


Format the `Total Sales` column as currency. Add a tool tip to our sparkline series. Use the following expression for the tool tip:

```
=Switch(Fields!SalesMonth.Value = 1, "January", Fields!SalesMonth.
Value = 2, "February", Fields!SalesMonth.Value = 3, "March",
Fields!SalesMonth.Value = 4, "April", Fields!SalesMonth.Value = 5, "May",
Fields!SalesMonth.Value = 6, "June", Fields!SalesMonth.Value = 7, "July",
Fields!SalesMonth.Value = 8, "August", Fields!SalesMonth.Value = 9,
"Spetember", Fields!SalesMonth.Value = 10, "October", Fields!SalesMonth.
Value = 11, "November", Fields!SalesMonth.Value = 12, "December") + " - "
+ FormatCurrency(Sum(Fields!TotalSales.Value),2)
```

The `Switch` function, according to MSDN, evaluates a list of expressions and returns an object value corresponding to the first expression in the list that is `True`. In the previous expression, the `SalesMonth` field contains an integer that corresponds to the number of the particular month. For example, `1` is `January`, `2` is `February`, and so on. We use the `Switch` function in this case to switch the number with the month name.

Now, let's take a look at the report:



You see how easy it is to present multiple data series in a single sparkline. Each sparkline in our report shows how the total sales are distributed in a month for that single product category.

What we usually want to see in reports like this is observe how our series fluctuates from one point to another; in this case, month to month sales. And we can easily accomplish that by using sparklines.

Aligning axes in the sparkline

Take another look at our report above. Something is amiss, right?

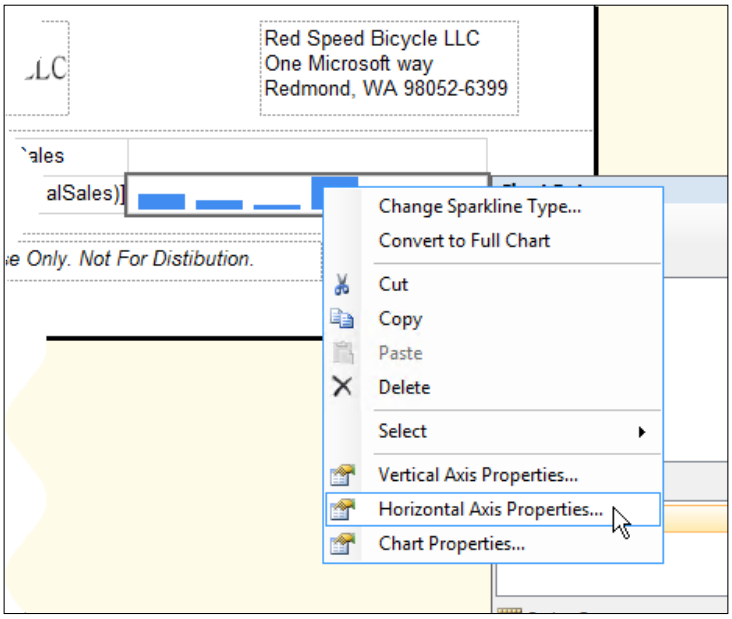
Compare the sparkline series between Canada and France. Canada has twelve (12) bars on its categories compared to France's six (6). What this tells us is that France has sales data only on the half of the year (six months). But which half: first or second?

That's hard to tell the way our Sparklines is presented in the previous report.

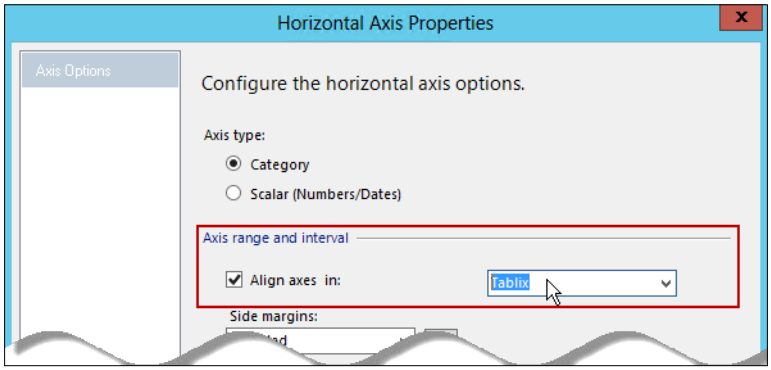
We want to align each and every bar so we can tell which months have no sales data or which month is missing data, whatever the case is.

Let's fix that.

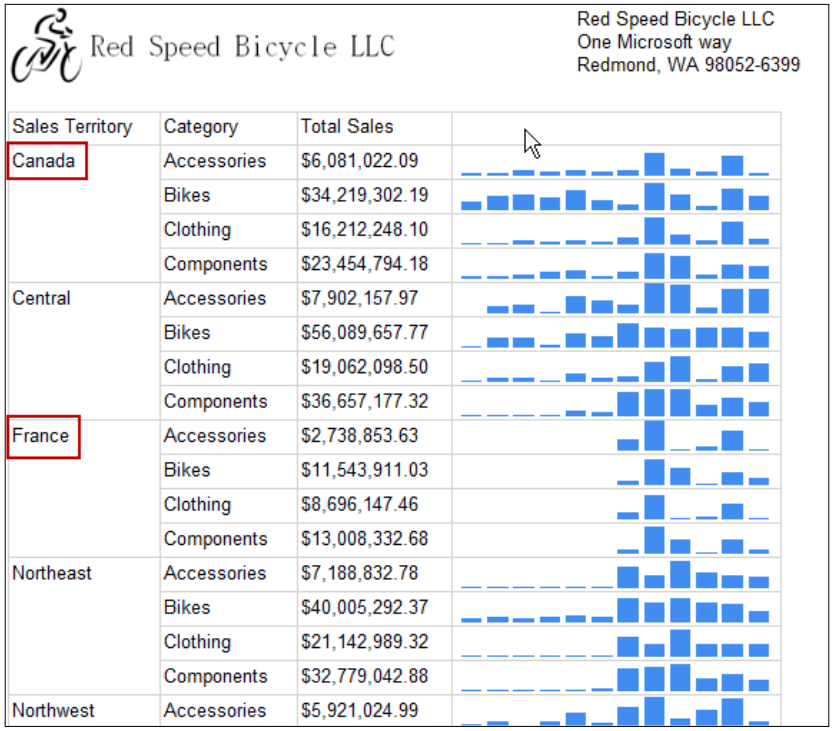
Going back to the report designer, select and right-click on the sparkline, then select the **Horizontal Axis Properties** option from its context menu, as shown in the following screenshot:



On the **Horizontal Axis Properties** window, select the **Align axes in** checkbox under the **Axis range and interval** section of the axis property, and select the tablix name (our tablix is named `Tablix`), as shown in the following screenshot:



With that little change, our report now has aligned bars from month-to-month across the whole table. It should look something like this (pay attention to **Canada** and **France**, highlighted in red box in the following screenshot, and see what the difference is this time):



Sales Territory	Category	Total Sales
Canada	Accessories	$6,081,022.09
	Bikes	$34,219,302.19
	Clothing	$16,212,248.10
	Components	$23,454,794.18
Central	Accessories	$7,902,157.97
	Bikes	$56,089,657.77
	Clothing	$19,062,098.50
	Components	$36,657,177.32
France	Accessories	$2,738,853.63
	Bikes	$11,543,911.03
	Clothing	$8,696,147.46
	Components	$13,008,332.68
Northeast	Accessories	$7,188,832.78
	Bikes	$40,005,292.37
	Clothing	$21,142,989.32
	Components	$32,779,042.88
Northwest	Accessories	$5,921,024.99

Using gauge in a tablix

Key Performance Indicator (KPI) is one of the most import kind of reports, specially to decision makers. There are a few ways of implementing KPIs in SQL Server Reporting Services 2012.

Using graphs with the combination of expressions that manipulate bar, cell, and font colors corresponding to certain conditions is a common method of producing KPI reports. Gauge and indicator objects give better KPI implementation in most cases.

Let's create a table report with an inline gauge in one of its columns.

Before we proceed, execute the T-SQL script called `SalesPersonQuota.sql`, which came with the downloadable code files for this chapter. This script creates a new table called `SalesPersonQuota`, based on the view `Sales.vSalesPerson` in the `AdventureWorks2012` database, and updates the `SalesQuota` column in the new table with new values.

Create a new report using our template. For the dataset use text query type, and type the following:

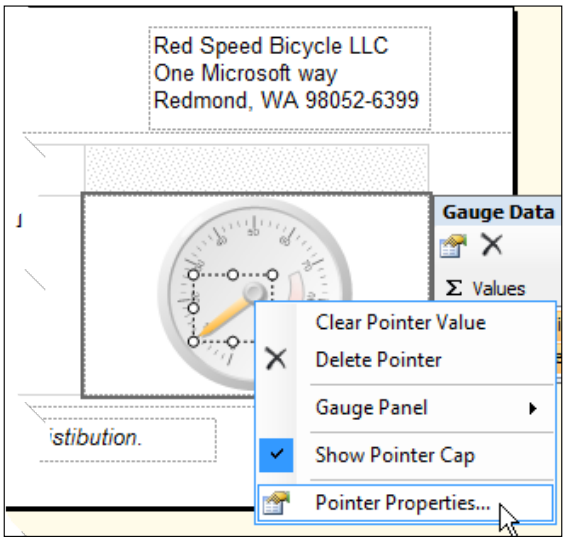
```
SELECT SalesPerson,
       SalesQuota,
       SalesYTD,
       SalesLastYear
FROM   SalesPersonQuota
```

Drop a 4-column table into the report designer. Add the three fields into the table. Drag a gauge object in the fourth column. Select the radial gauge type. Resize the columns accordingly.

There are three elements to set in a gauge: pointer, scale, and range.

Configuring the pointer of a gauge

Let's configure the pointer. Select the gauge pointer, then right-click on it. From its context menu, select **Pointer Properties**:



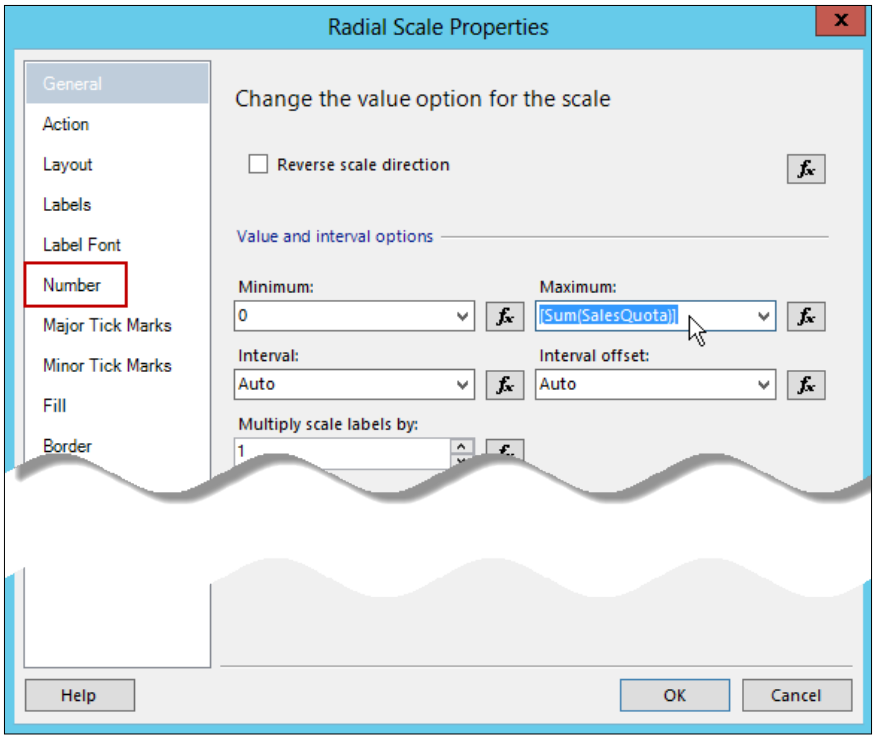
Set its value to `[SUM(SalesYTD)]`.

Let's add another pointer. Select and right-click on the gauge itself, select **Add Pointer** from the context menu to add a secondary pointer. Set the value of the second pointer to `[SUM(SalesLastYear)]`.

Change the color of our primary pointer, fill color to lime green and the secondary orange. What we want to achieve by the two pointers on the same gauge is compare the current sales with the previous sales value at the same point in time.

Adding a scale for the gauge

Now, select the scale (numbers in the gauge) and right-click on it to display its context menu. On the **Radial Scale Properties** dialog box, set its **Maximum** value to that of the `[SUM(SalesQuota)]`. Before you save and close the dialog box, format the number. Don't put decimal and show its value in Millions.



You probably get the idea of what the relationship is between the pointers (Sales YTD / Sales Last Year) and scale (`SalesQuota`). The pointer indicates how far or close the value (SalesYTD / Sales Last Year) is to its target or goal (`SalesQuota`).

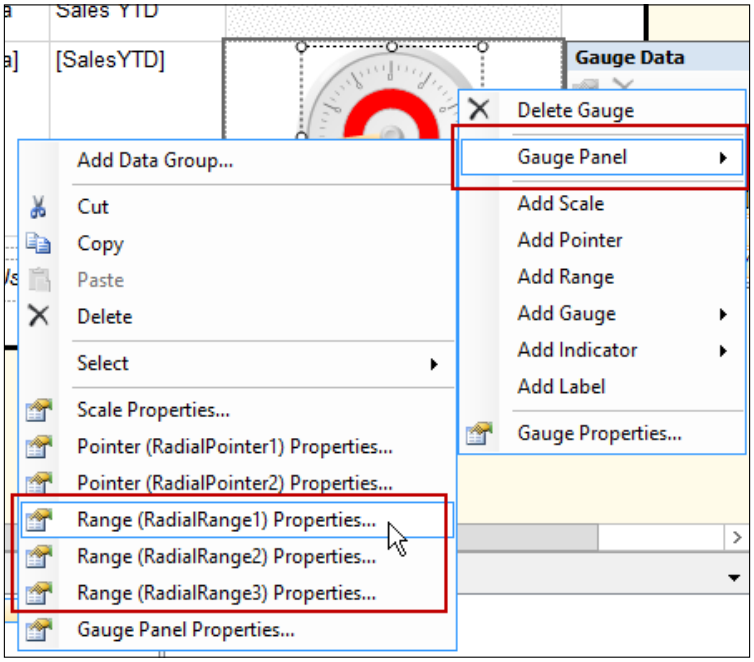
Defining a range of the gauge

The range of predefined milestones is common to KPIs such as quota fulfillment and sales growth. These KPIs speak of how well (or how badly) the sales person performs within a definite period. They also show the sales person's capability to consistently meet quota or sales goals. We usually use red or danger zone to mean something failed at reaching a predefined mark (that is, quota), green zone if something hit the right mark. Our demonstration will make this clear.

We're going to add three ranges (zones) in our gauge.

By selecting the gauge, right-click on it for displaying its context menu. From the menu, select **Add range**. Do this two times more to create the second and third range.

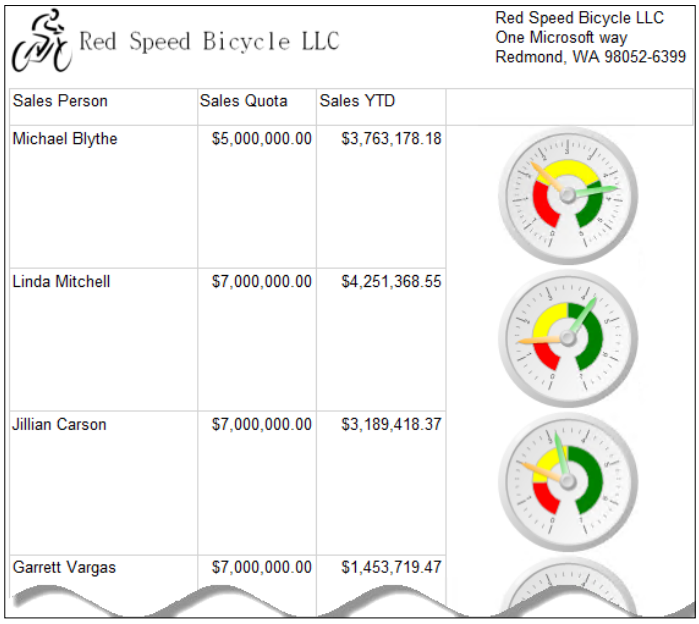
For setting the property of our first range, select and right-click on the gauge. From the menu select **Gauge Panel**, and then select **Range (RadialRange1) Properties** as shown in the following screenshot:



Set the property of each range according to the following table:

Range	Start Range	End Range	Placement	Distance	Start Width	End Witdh	Fill (Solid)
RadialRange1	0	1500000	Inside	30	30	30	Red
RadialRange2	1500000	3500000	Inside	30	30	30	Yellow
RadialRange3	3500000	7000000	Inside	30	30	30	Green

Now, let's take a look at our gauge report:

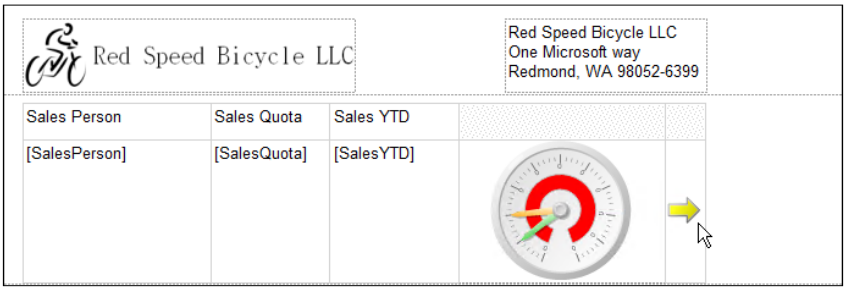


Adding an Indicator

We use an indicator to gauge something against a set standard. Let's say that sales amounting to 75 percent of the target quota at a certain point in time is a good indicator of a good sales performance.

Let's add an indicator to the gauge report we created previously. If you don't want to mess with that, just copy that RDL file into a new report and modify that instead.

Let's add a new column to the right of the gauge column. Adjust all the columns appropriately. When the **Select Indicator Type** dialog box is displayed, select the 3 arrows (colored) directional indicator:



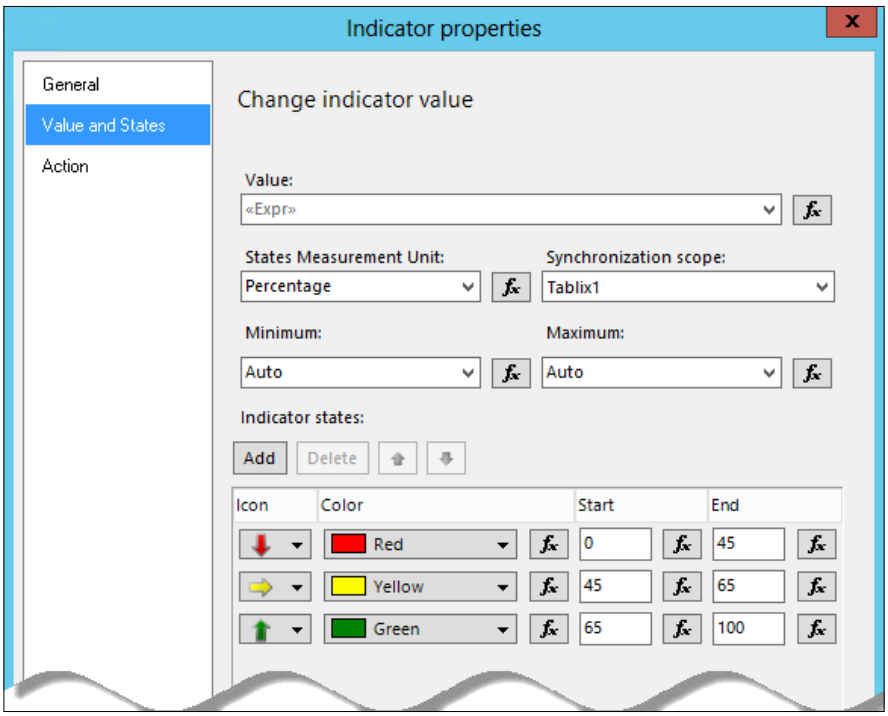
Select and right-click on the Indicator, then select **Indicator Properties**. On the **Value and States** tab, click on the expression button next to the **Value** drop-down list, and enter the following expression:

```
=(Fields!SalesYTD.Value/Fields!SalesQuota.Value)*100
```

What we're trying to do here is find out the percentage of `SalesYTD` against the given `SalesQuota`.

Let's just say that our predefined indicators of sales performance are the following:

- 0-45 is definitely not a good sign (red)
- 45-65 means doing just OK (yellow)
- 65-100 means good

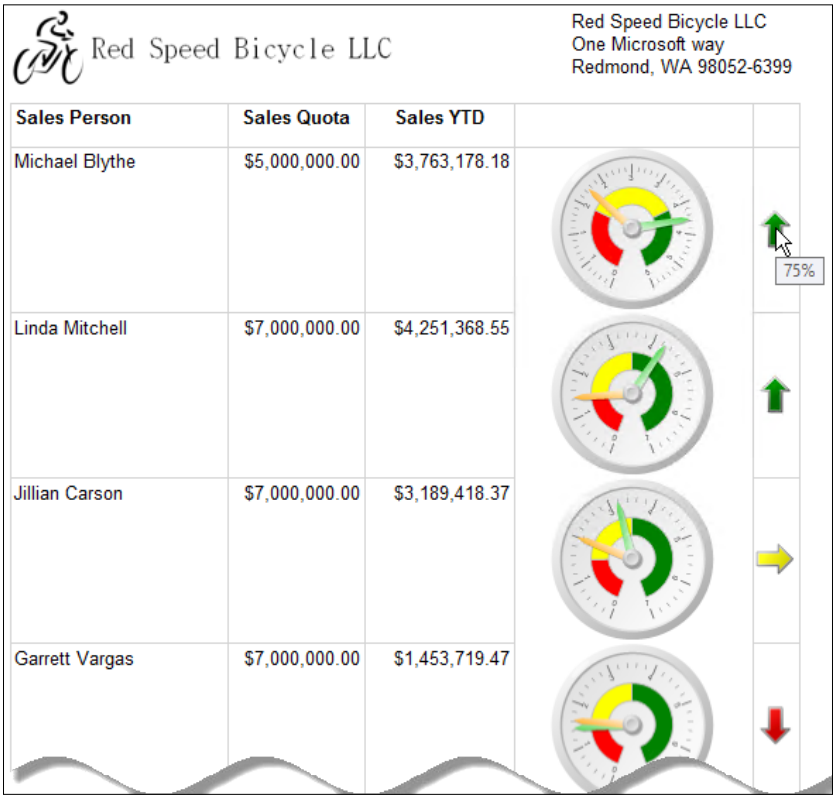


Let's add a tool tip to the indicator. On the **General** tab of the **Indicator Properties** window, enter the following expression for the tool tip value:

```
=FormatPercent(Fields!SalesYTD.Value/Sum(Fields!SalesQuota.Value),0)
```

The `FormatPercent` function converts the argument into percentage value. So, 75 becomes 75 percent.

Let's take a look at our completed report:



We're done!

Summary

We can do a lot of things in our reports using varying data visualization. What we've seen in the demonstrations in this chapter is just scratching the surface of SQL Server Reporting Services 2012.

We can build great reports with colorful graphs, gauges, and indicators without much effort as far as report development is concerned. SSRS 2012 is designed to meet the common report development requirements, so report authors or developers and the business people can minimize the time they spend on the presentation layer, and thereby, maximize their effort on the quality, validity, and accuracy of their reports.

5

Location, Location, Locations!

John Kirkland's chart reports have become a big hit in every department of their company. Their numbers have become alive in vivid colors with the use of charts, bars, sparklines, gauges, and indicators. Now it's easier for the business analysts to visualize (pun intended) their company's financial performance through data visualization.

John is now the most loved IT expert in their company. He has now become the Superfluous go-to guy for anything that has to do with data analytics.

Based on John's SQL Server Reporting Services reports, the company is performing well on all fronts. They were able to pinpoint some data points where they need to make some improvements, and were able to recognize some gaps in their sales strategies that the Sales Director will address in their next quarterly meeting.

John knows for sure he will get busier as the meeting approaches. Requests for reports for so and so will again be the order of his day, and he has heard about the company's plan to expand to more international cities.

One of the finance analysts approached John the next day. The Sales Director wanted some kind of visualization that included overview of their sales territories and the sales numbers associated with each.

John's brain clicked! He has SQL Server Reporting Services spatial visualization features to thank!

Let's us follow John as he learns how to lay down data points over map layers.

In this chapter, you'll learn the following:

- Adding the polygon layer
- Adding the point layer
- Adding the tile layer
- Using a shapefile
- Customizing center point

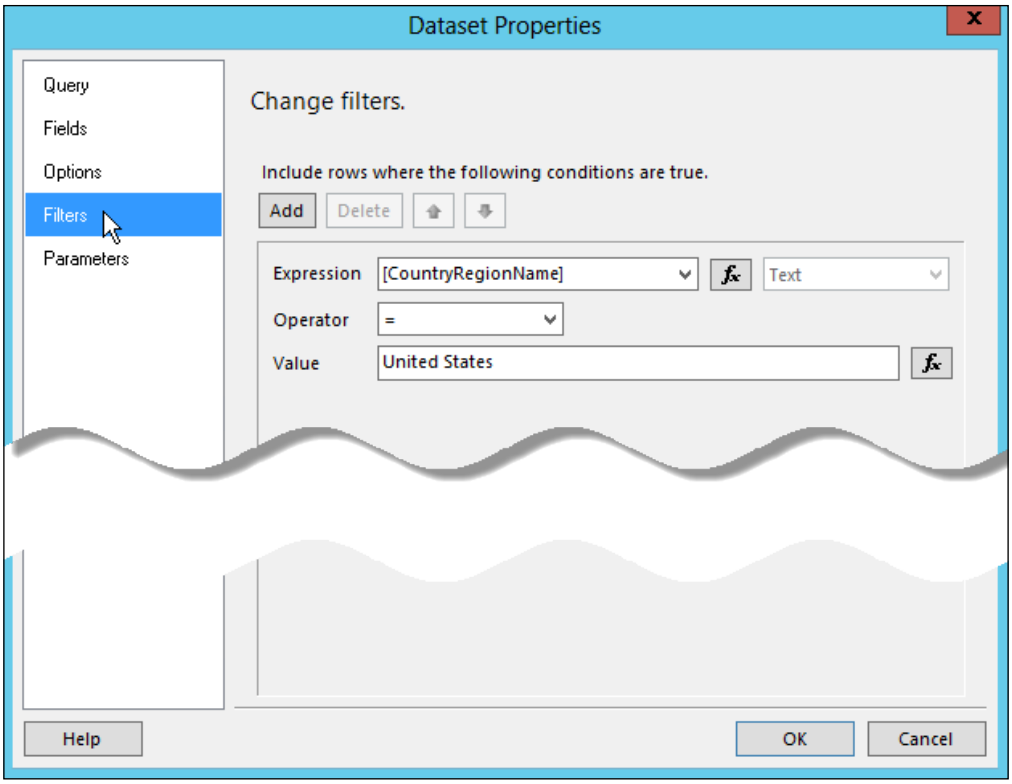
Creating a map (polygon) visualization

Let's create a map visualization using the wizard.

1. First find the T-SQL script named `uspSalesByTerritory.sql`. Execute the script on the AdventureWorks2012 database to create a stored procedure called `uspSalesbyTerritory`.

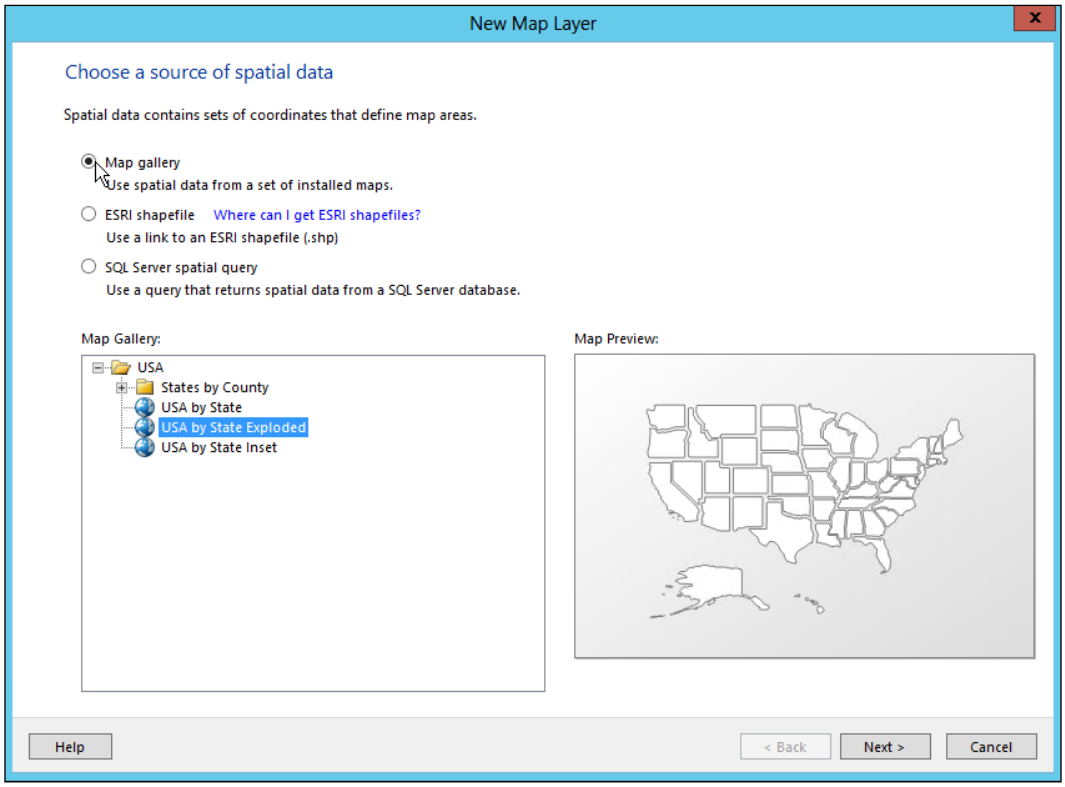
2. In the report designer, create a shared dataset based on the `uspSalesByTerritory` stored procedure. Let's name it `ds_SalesByTerritory`.

3. Create a new report using our report template. Name it whatever you want. Create a dataset based on the shared dataset that we just created. In the **Dataset Properties** window, select the **Filters** tab, and click on the **Add** button to add a filter with the following conditions:
    - **Expression** is set to **[CountryRegionName]**
    - **Operator** is set to **=** (equals sign)
    - **Value** is set to **United States**



4. Click on the **OK** button when done.
5. Drag a map object from the report designer toolbox into the report body. This displays the **New Map Layer** dialog box.

6. On the first dialog box, select **Map gallery** as the source of our spatial data. Select **USA by State Exploded** from **Map Gallery**.



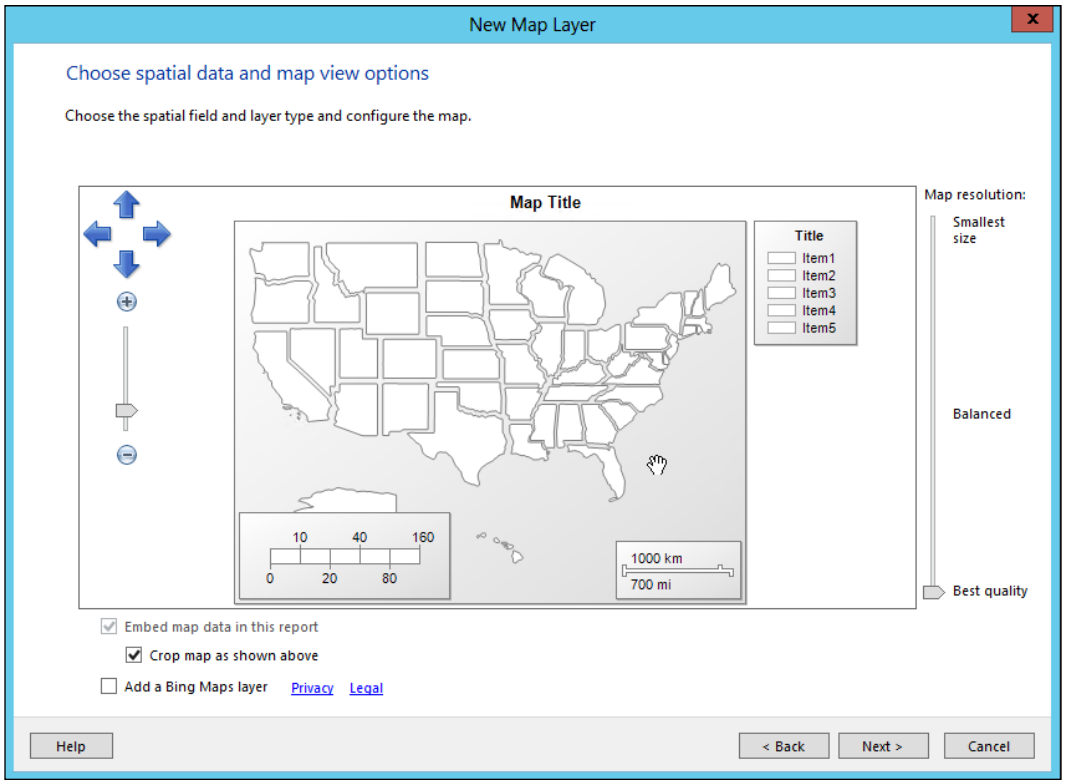
7. Click on the **Next** button when done.

The second dialog box provides options for spatial data and map views.

Options for spatial and data views

In the spatial and data views options, we have the option to crop our map to a particular zoom level that we prefer. We can also set the resolution of our map (visual quality). As far as the map resolution is concerned, note that we're choosing between quality and performance. The higher the resolution, the more it impacts the performance of rendering our report.

1. The arrow keys allow us to pan the map from left or right, and up or down. You can also "*grab*" the map to pan in any direction. You can do this by moving the mouse cursor to the map, and either right-click or left-click on it.
2. Use the arrow slider to zoom in or zoom out the map.
3. Use these keys to center the map, and zoom in to fit in the size of the map border.
4. Select **Crop map as shown above**.
5. You can add a bing layer to the map but we don't really need that in this demonstration. Add it anyway if you like.

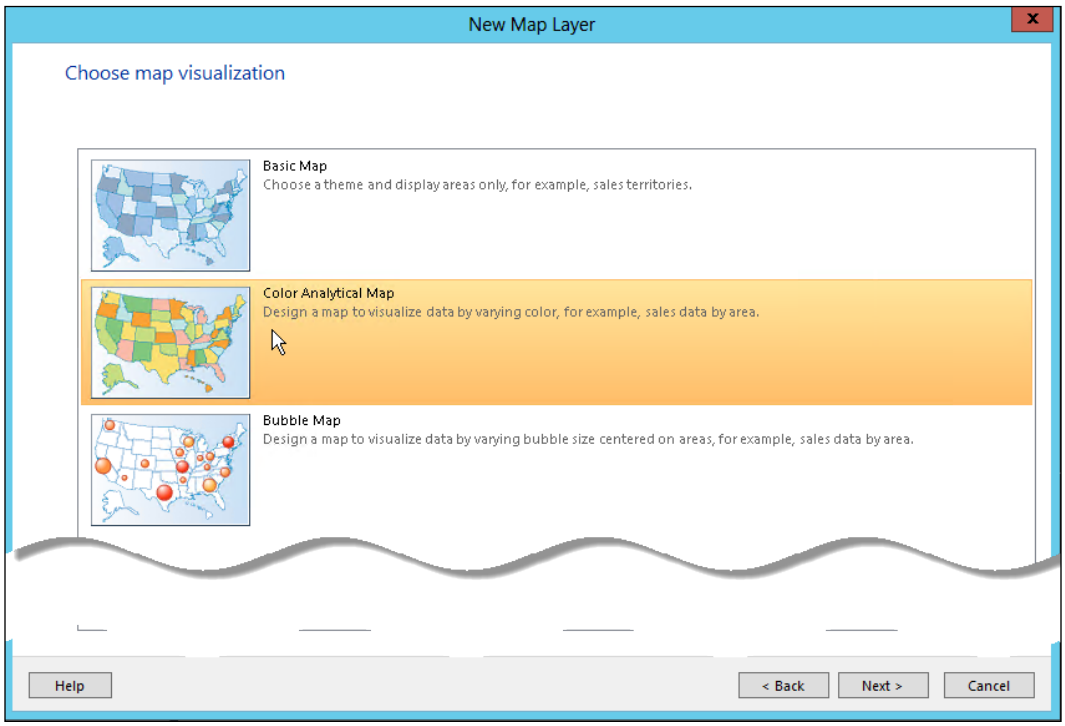


6. Click on the **Next** button when done.

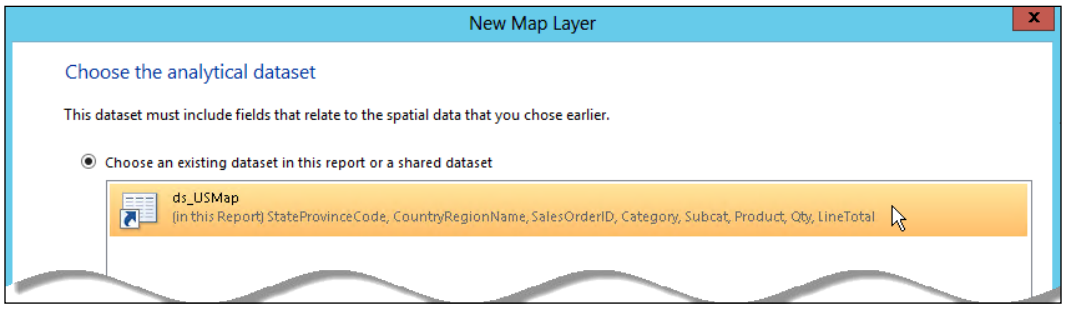
   The next dialog provides options for map visualization.

7. Let's select **Color Analytical Map**.

   You can choose any of the options depending on what you need. In case of our demonstration, we want to visualize data by varying color.



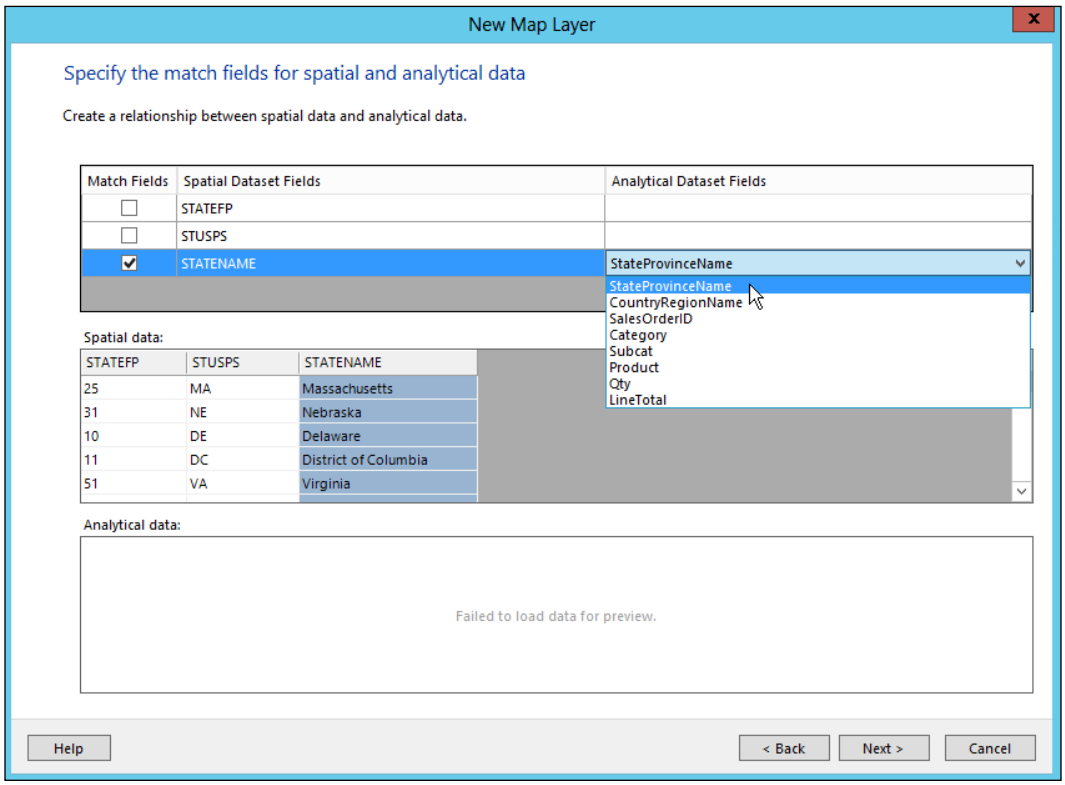
8. Click on the **Next** button when done.
9. On the next dialog, let's choose an existing dataset in this report. We only have one dataset in this report, so let's select that one:



10. Click on **Next** when done.
11. The next dialog provides options that allow us to tie our data with the map. Basically, what we are doing here is matching **Spatial Dataset Fields**—the data associated with the map itself—with **Analytical Dataset Fields**, our defined dataset for this report.
12. Let's match the map's **STATENAME** (State) field with our **StateProviceName** field.

    Select the checkbox next to the **STATENAME** field. This will highlight the data column underneath to show us its associated spatial data. This gives us a quick look of the spatial data value.

13. Let's select our matching data field, in this case, **StateProviceName**.



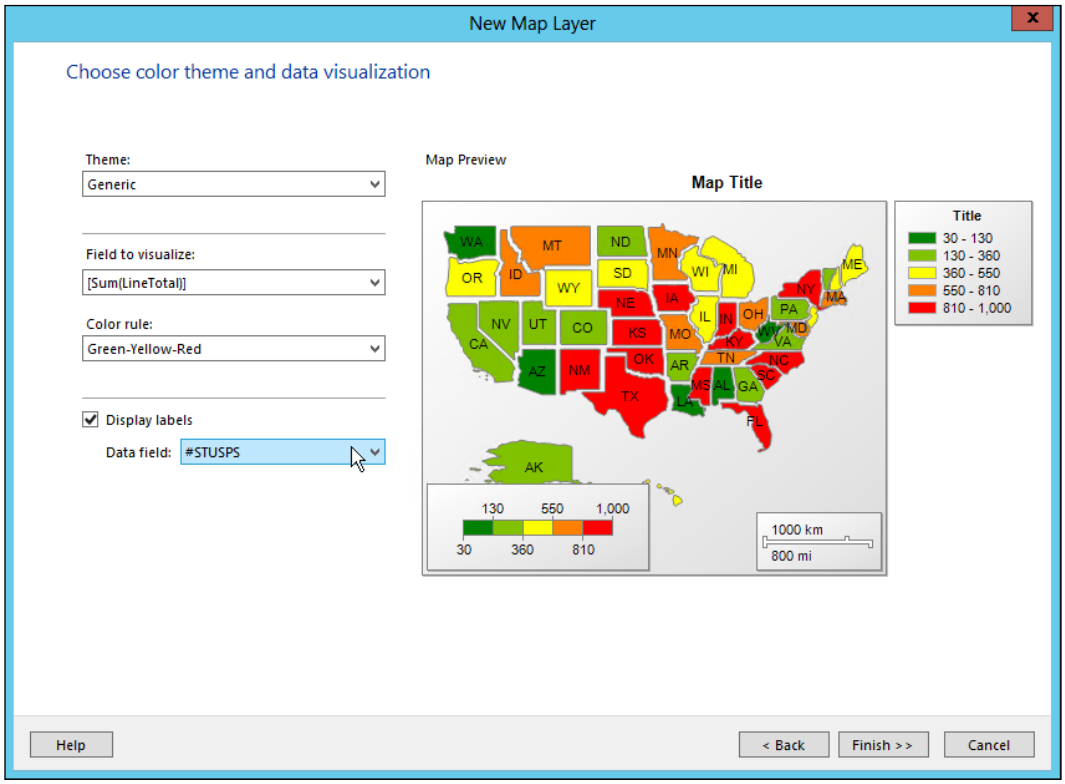
14. Click on **Next** when done.

The next dialog provides options for theme and data visualization.

Options for theme and data visualization

In the theme and data visualization options, we can set the criteria of aggregation we want for our map, and also the color rule. We can change both on the report designer if we decide to change the aggregate or color later.

1. Select the **Generic** theme.
2. For the **Field to visualize** option, select **[Sum(LineTotal)]** from the drop-down textbox.
3. Select **Green-Yellow-Red** for **Color rule**.
4. Select the checkbox next to **Display labels**; and for **Data field** select **#STUSPS**, which adds state code labels on the map as shown in the following screenshot:



5. Click on **Finish** when done.

Formatting the map

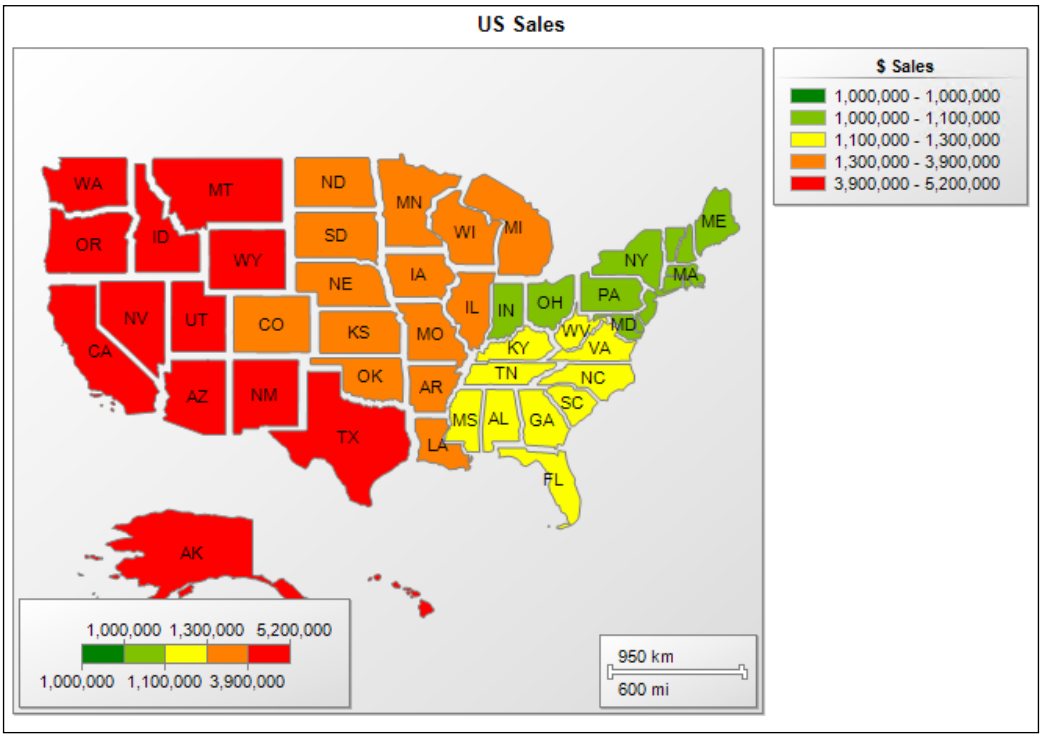
Selecting and right-clicking on any elements in the map gives you access to the element's properties. You can also tweak the map element properties by using the **Properties** window. Again, if the **Properties** window is not displayed, go to the **Menu | View | Properties** Window; or, you can simply press *F4*.

You can simply change the titles of the map and the legend just by double-clicking on the title element and edit it inline. Alternatively, you can just grab any element such as the scale and move it around.

Let's do that. We will also change the legend style to Column (right-click on the legend, then select **Legend properties**).

At this point, that's all we need to do.

Now we can preview our map:



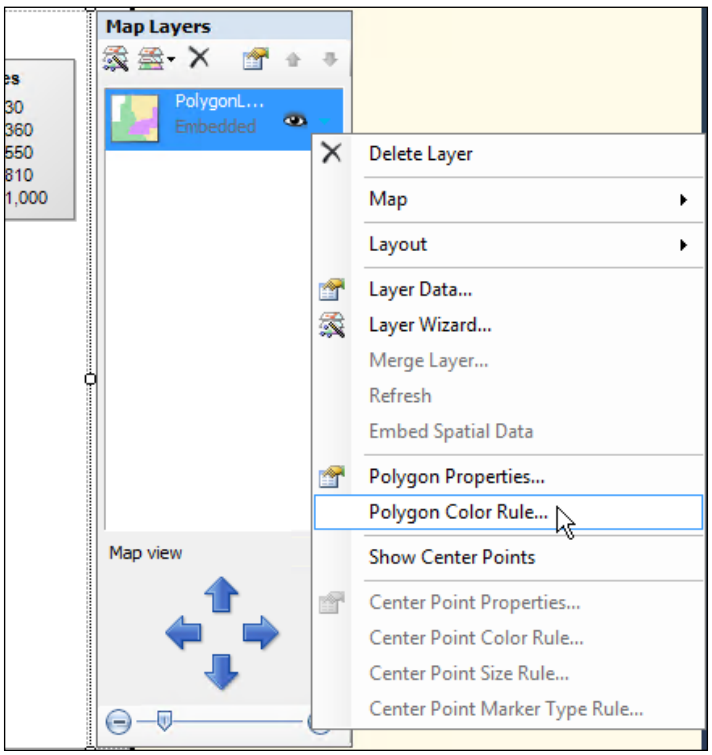
Changing the color rules

You can change the colors of your maps if you don't like the default colors provided by the Color Rule Green-Yellow-Red:

1. Select the Map which displays the **Map Layers** properties pane.

   The map that we just created is a polygon layer object. So, you see a **PolygonLayer1 (Embedded)** in the properties referring to our map (polygon), unless you named the layer with a different name.

2. Select **Polygon Color Rule** as shown in the following screenshot:

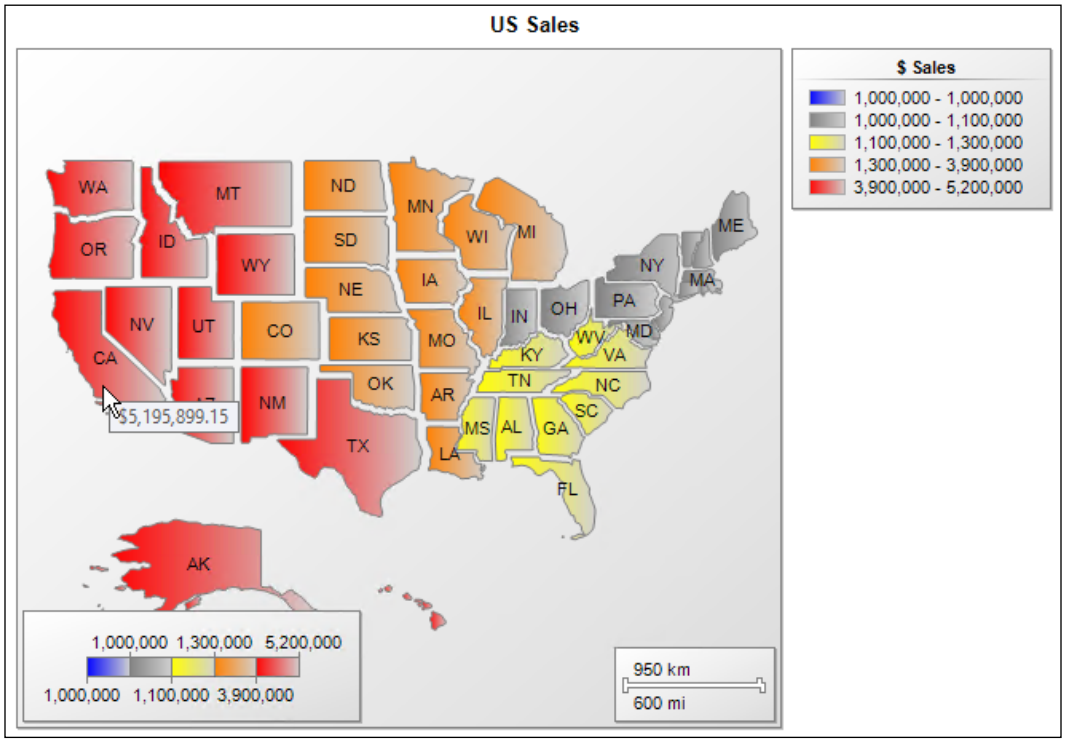


   This displays the **Map Color Rules Properties** Dialog box.

3. On the **General** tab, change the **Start color** to **Blue**. And on the **Distribution** tab, increase the **Number of subranges** from the default **5** to **8**. Play with the properties and see how that affects your map in real time in the report designer.

4. While we're at it, let's also change the map polygon properties. If you noticed from the previous illustration, **Polygon Properties** is one of the options in the menu.
5. In the **Polygon Properties** dialog box, select the **General** tab. Click on the **Expression** button next to drop-down textbox on the tool tip. Enter the expression `=FormatCurrency(Sum(Fields!LineTotal.Value))`.
6. Also change **Fill Style** to **Gradient** from the **Fill** tab.
7. Click on **OK** when done.

We can now preview our map with the new properties setup:



You might not notice the difference in this book, but changing some properties of the map really does make a difference from the design perspective.

Adding center points

If you noticed in the **Map Layer** properties, there's another option in the menu called **Center Point**. We can add a center point in a map layer to add another nuance of the same dataset.

For example, our map shows some kind of heat map of which state is making the most sales in dollar terms. The color red represents the "hottest" state.

Let's introduce a center point on the same layer. For instance, suppose that we want to see which state is selling the most number of products. More products sold don't necessarily mean higher revenue in some cases. One state might be mostly selling accessories while the other is mostly selling the big items. In this case, although the former is selling more items, the latter is selling more in dollar terms.

Let's see how this scenario is demonstrated by adding a center point on the same layer.

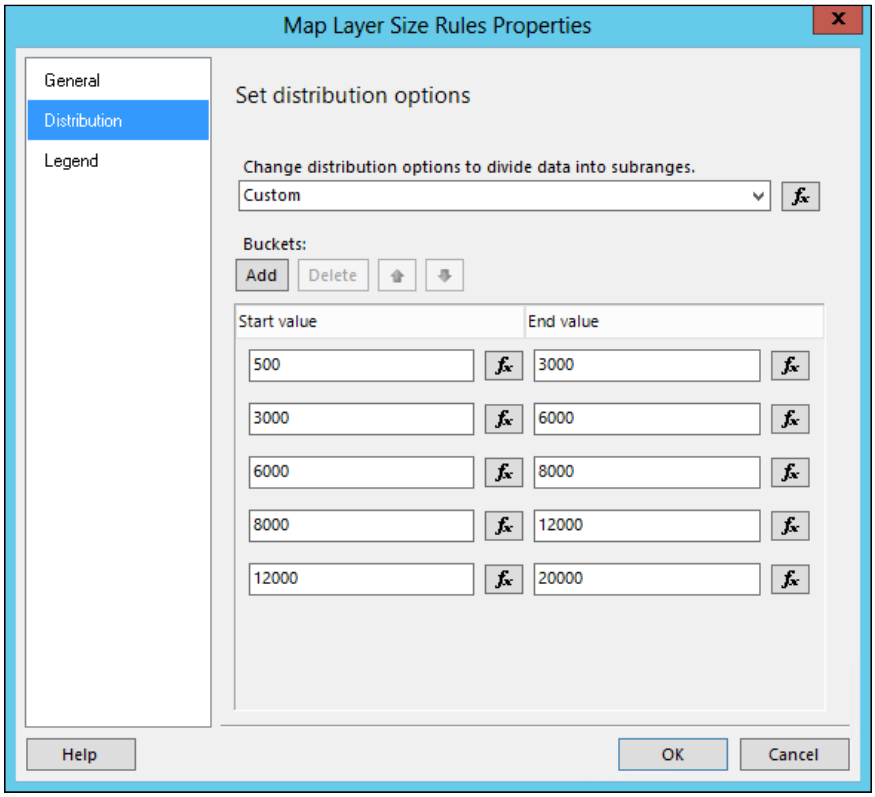
Again, from the **Map Layer** properties pane, select **Show Center Points**. That marks the center point property as checked.

Customizing distribution options

The distribution options enable us to show the variance effect among the ranges defined by the size of our aggregates. We have the option to either use the built-in distributions, or create a customized one.

1. From the **Map Layer** properties pane, select **Center Point Size Rule**. This displays the **Map Layer Size Rules Properties** dialog box (you can also do this with the map color rules).
2. On the **General** tab, select **Visualize data by using size**. Select **[Sum(QTY)]** from the drop-down textbox of the **Data Field** option. Set **Default size options** to zero (0).

3. From the **Distribution** tab, select **Custom** from the drop-down textbox of the **distribution** option. Add **Buckets** as shown in the following screenshot:



4. Click on **OK** when done.

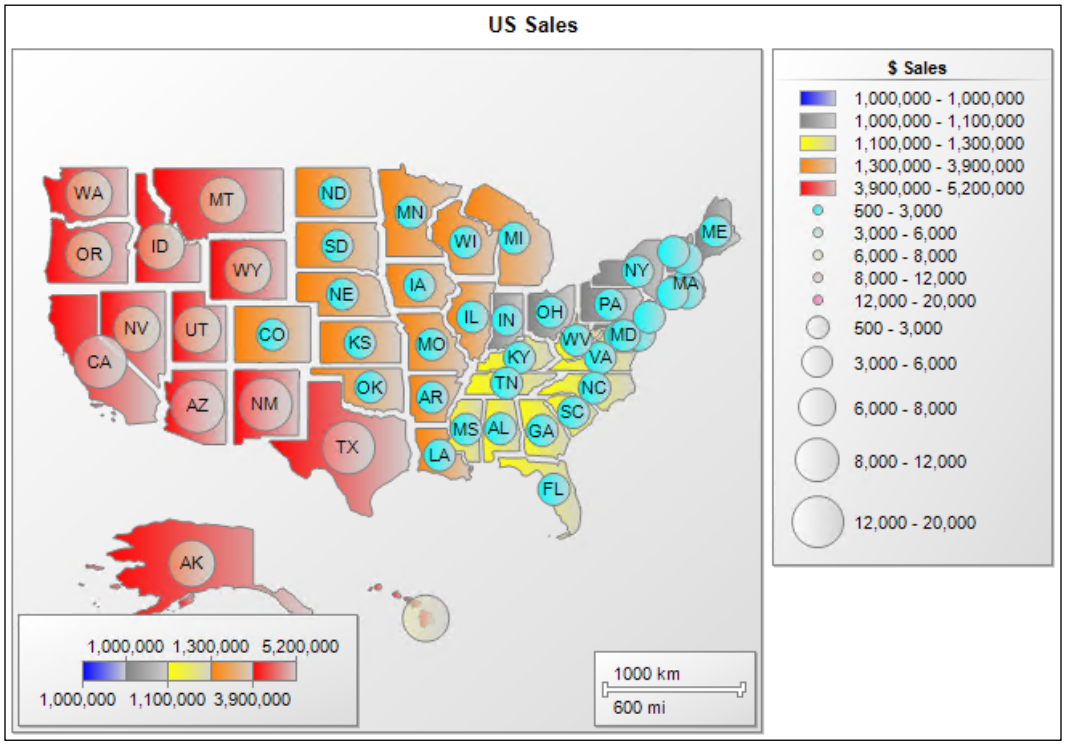
   Let's do the same thing with the color rules for our center points.

5. Select **Center Point Color Rule** from the **Map Layer** properties pane.
6. On the **General** tab, select **Visualize data by using color ranges**. Select **[Count(Qty)]** for **Data Field**. You can actually select any of the fields depending on what data you want to measure.

7. You can select whatever color you prefer for the color ranges but, for this demonstration, select the following:
    - Start color: Aqua
    - Middle color: Khaki
    - End color: Hot Pink
8. We may also want to make the colors transparent so they won't block our state labels.

   Click on the arrow next to the color to show the color selector dialog. Set **Transparency** to **100** for all three colors.
9. On the **Distribution** tab, select **Custom** for the **distribution** option. Let's just copy the same start and end values from our size rule (see previous illustration).

We can now preview our report with the added center point:



Using a shapefile

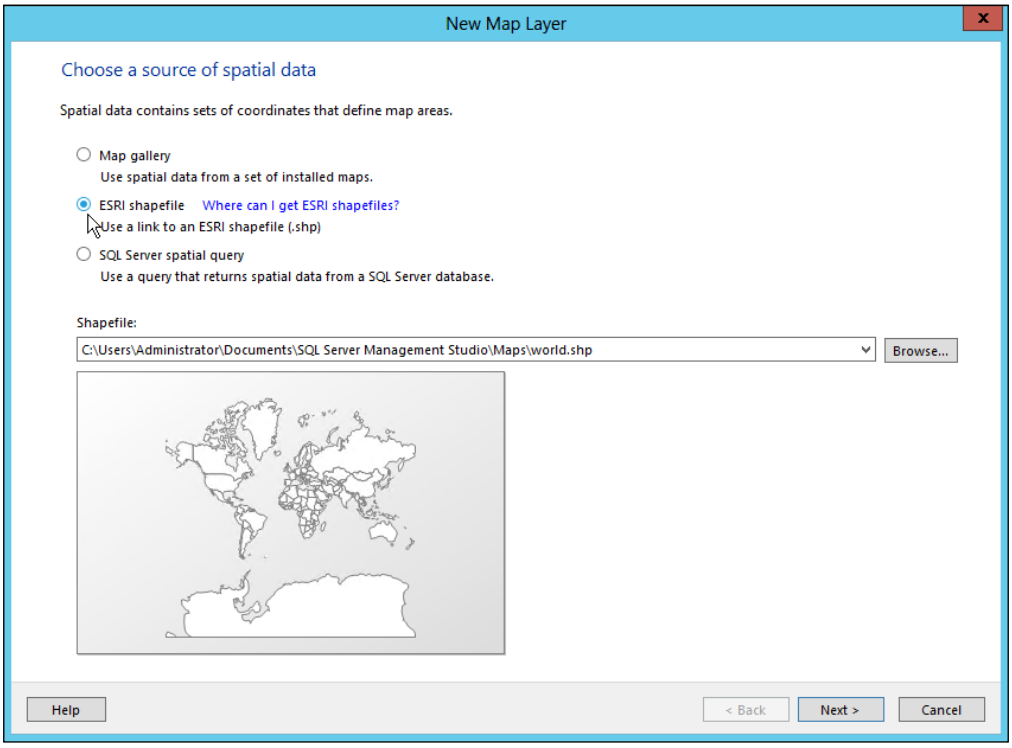
SQL Server 2012 Reporting Services has maps for the United States only. Fortunately, it allows shapefiles (map files) to be imported to reports. You can find plenty of free and commercial shapefiles over the internet.

The shapefile for the world map used in this demonstration is downloaded from VDS Technologies. The downloaded files include `world.dbf`, `world.shp`, and `world.shx`. You need all these files for the map to work in your reports.

1. Make sure that all the third-party shapefiles used in your reports can be accessed by the report server.

> VDS Technologies website is `http://www.vdstech.com`. Please check the site for information on using their maps for your projects.

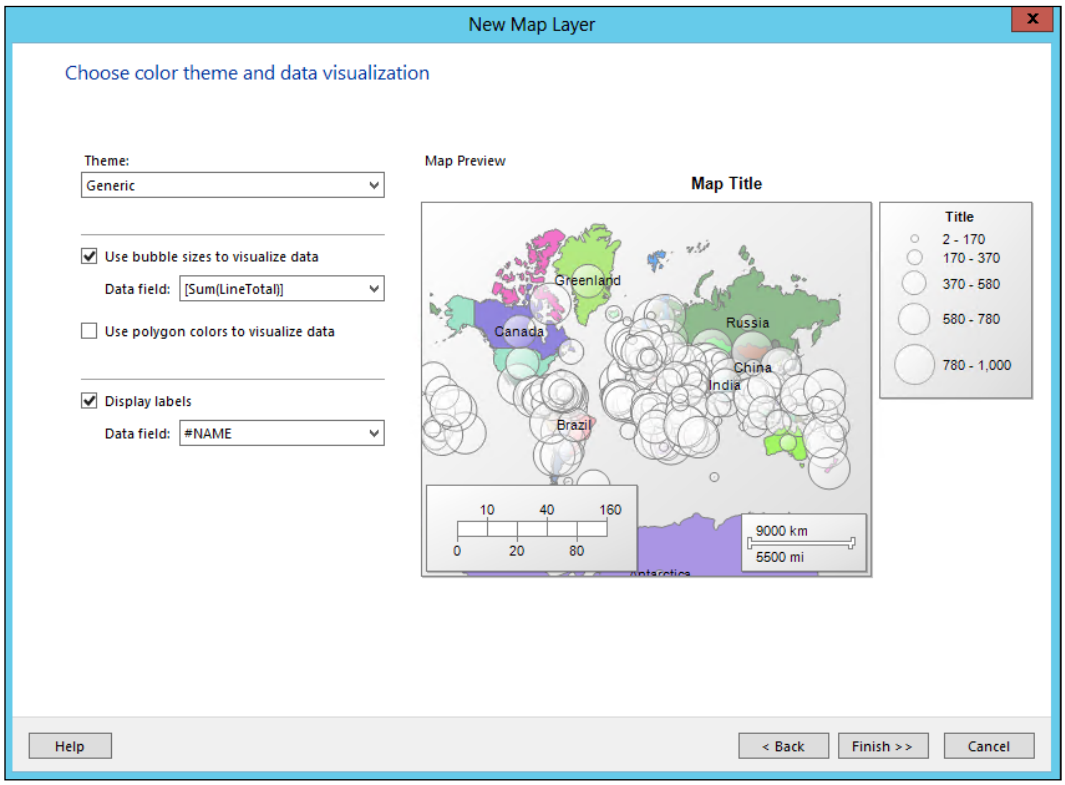
2. Create a new report and use the same shared dataset we used from our previous map report.
3. Now drag a **Map** object to the report designer.
4. Select **ESRI shapefile** as the source of your spatial data, and browse for the `world.shp` file.



5. Select **Bubble Map** for the map visualization.
6. Match **NAME** from the **Spatial Dataset Fields** with **CountryRegionName** in our **Analytical Dataset Fields**.

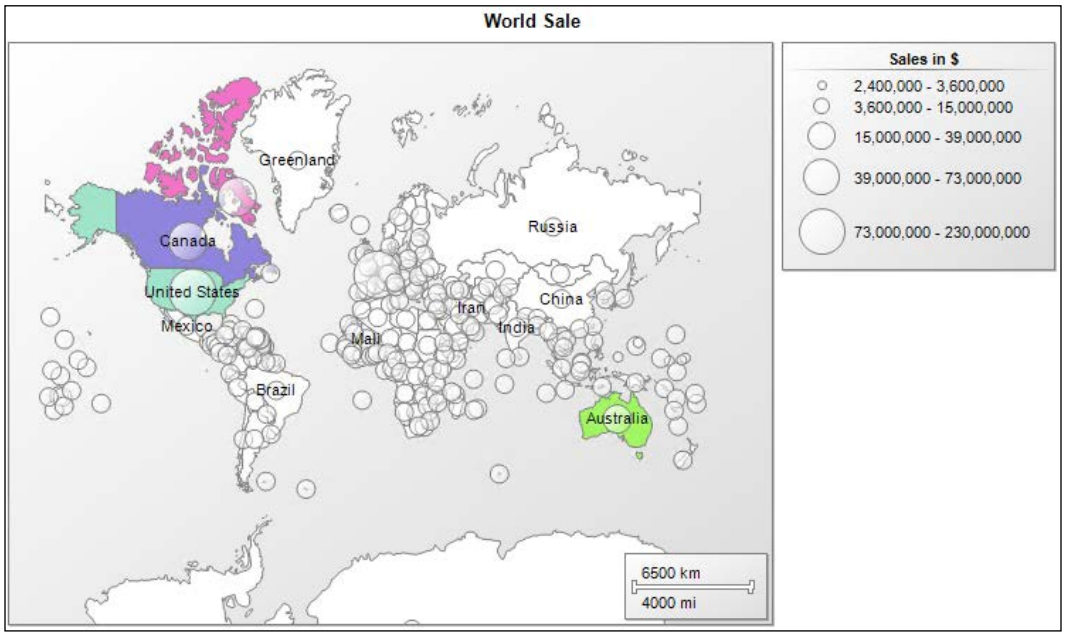
   Let's keep the theme simple by using Generic theme.

7. Select the checkbox next to **Use bubble sizes to visualize data**, then select **[Sum(LineTotal)]** for **Data field**. Also, select the checkbox next to **Display labels**, and select **#NAME** for **Data field**.



8. Click on **Finish** when done.

Let's preview our world map report:

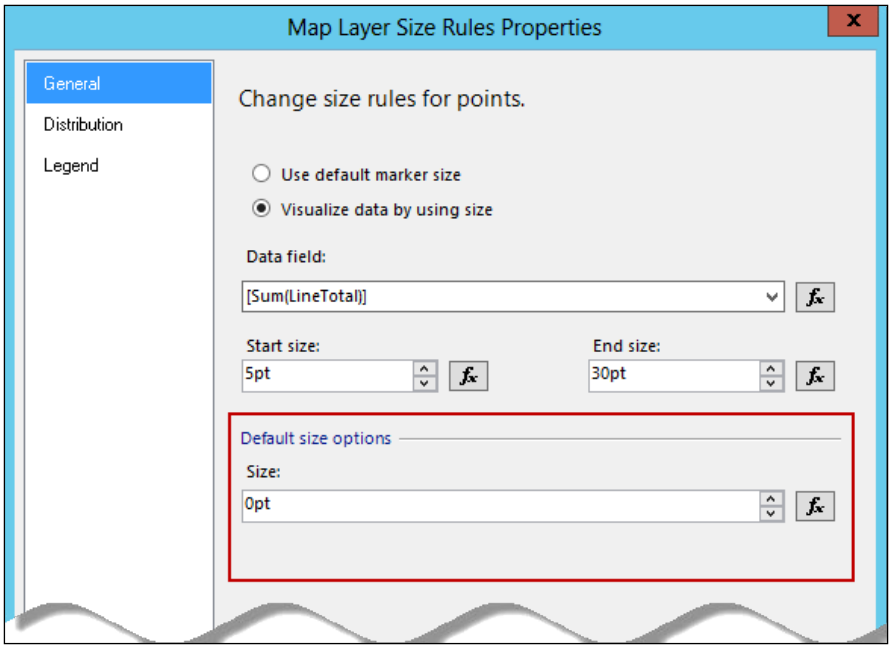


Removing empty center points

You notice that there are points just for every country in our map. We want to clean this up by removing these empty points, leaving only those included in our analytical dataset.

1. Select the map and access its **Map Layers** properties. Refer to our previous map if you forget how to access the properties.

2. Select **Center Point Size Rule**. On the **General** tab, set **size** of the **Default size options** section to **0pt** (zero).

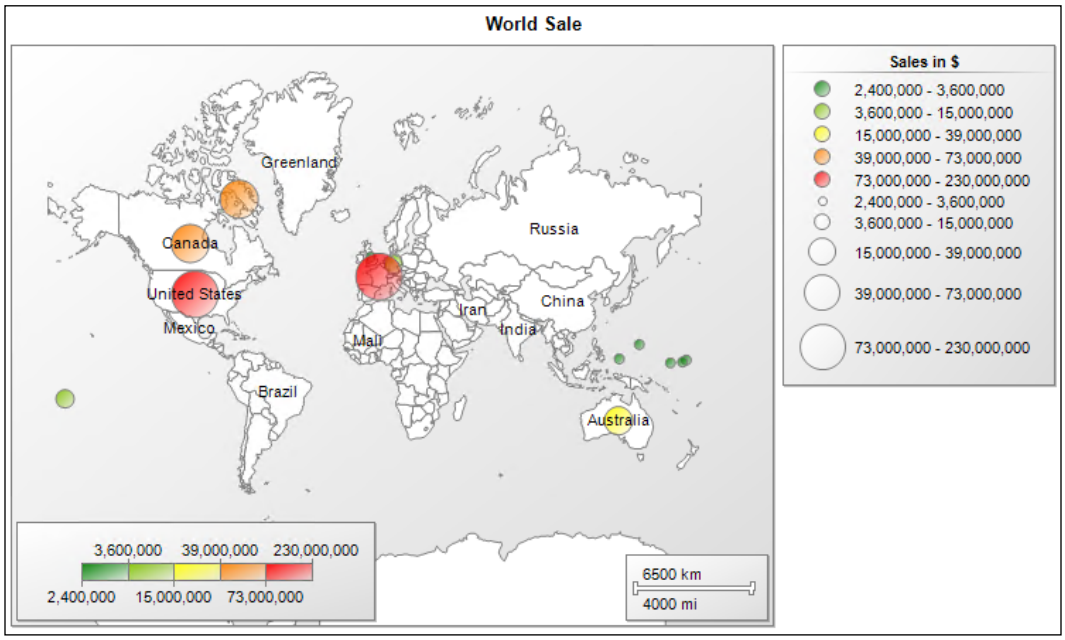


3. Click on **OK** when done.

   We may also want to change the polygon color rules (map color).

4. Select **Apply template style**, which means apply no color; just so we can focus our attention to the bubbles.

Let's preview our map:



Adding a random point layer

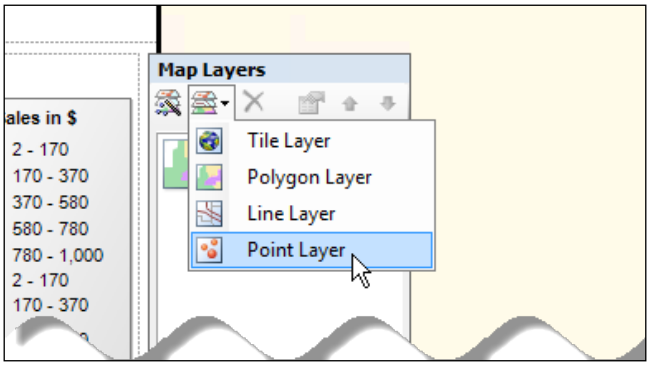
We can add a point layer using analytical dataset from our database, say top 10 sellers of tires. You can do that by simply matching the analytical dataset with the spatial data (map).

Adding a point layer has the same steps as adding a polygon layer.

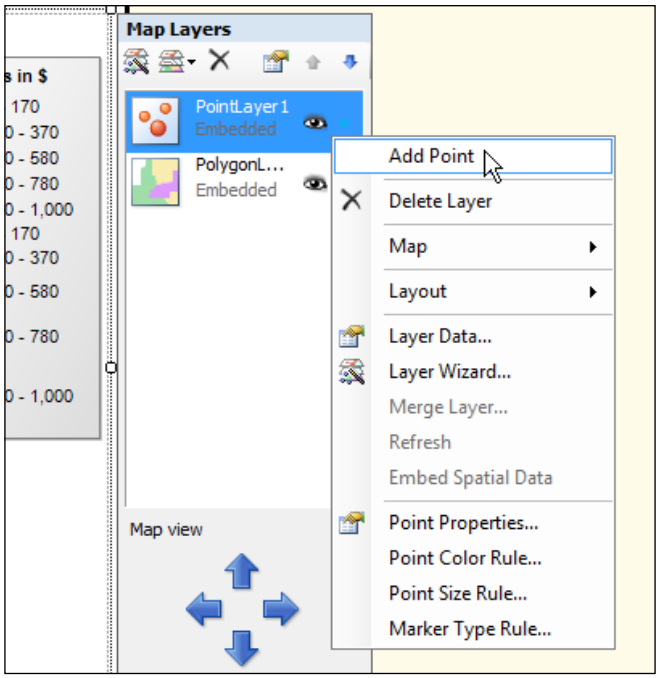
But what if you want to identify a point that is not based on a dataset?

Let's say we want to add a new layer on our previous world map. Let's identify the countries the company have on their expansion list; namely, Russia, China, and Brazil.

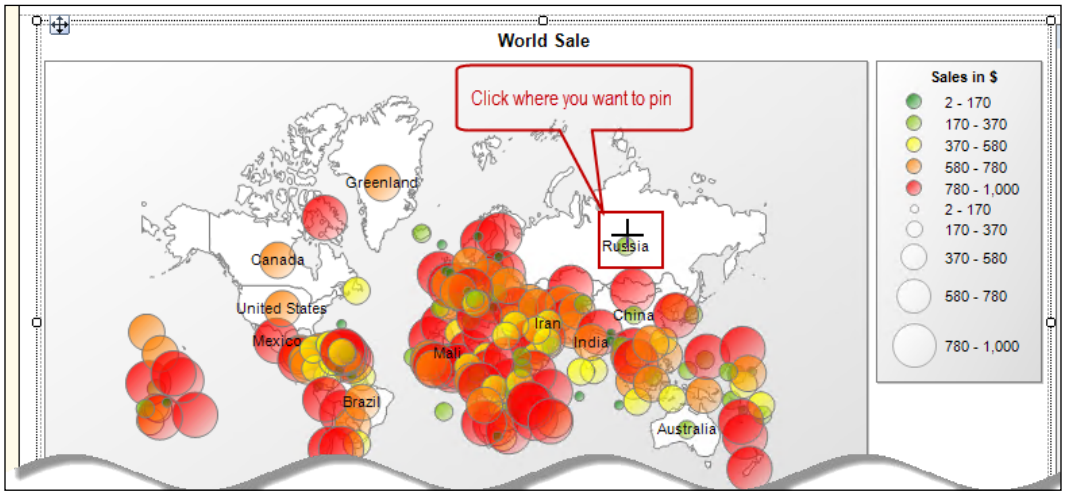
1. In the **Map Layer** properties pane, click on the add layer button and select **Point Layer** from the menu as shown in the following screenshot:



2. Click on the arrow of **PointLayer1** (if you didn't rename it) to access its context menu, and select **Add Point**, as shown in the following screenshot:



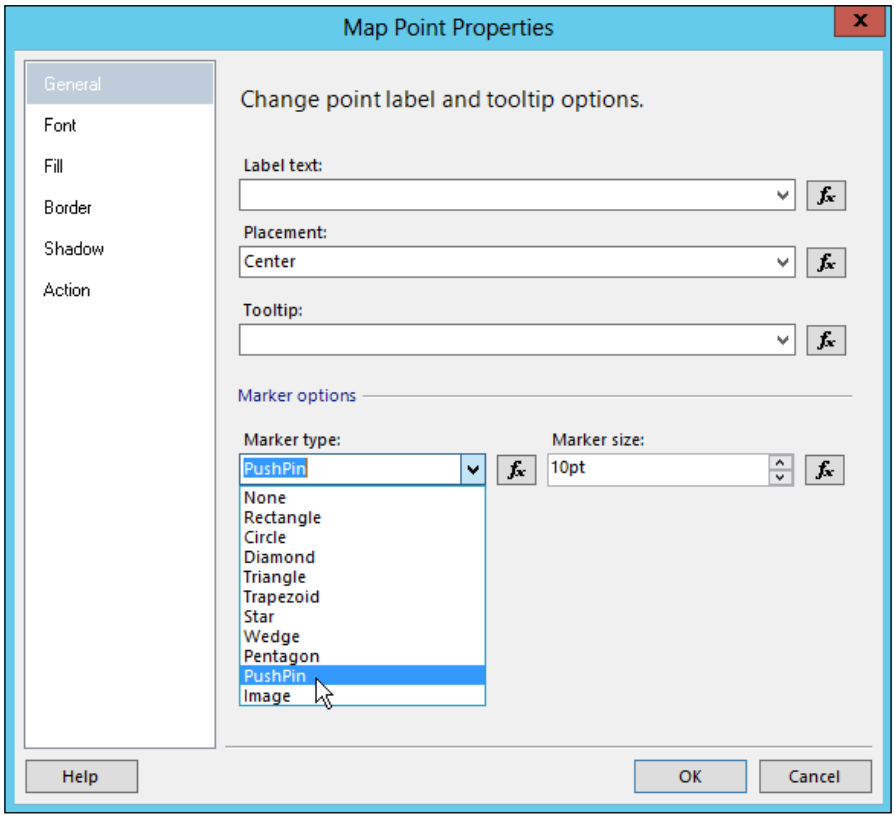
3. That displays a crosshair, which you use that to pinpoint the area in the map that you want to identify, in this case Russia:



4. Repeat the same steps for China and Brazil.
5. Let's change **Marker type** of our point.

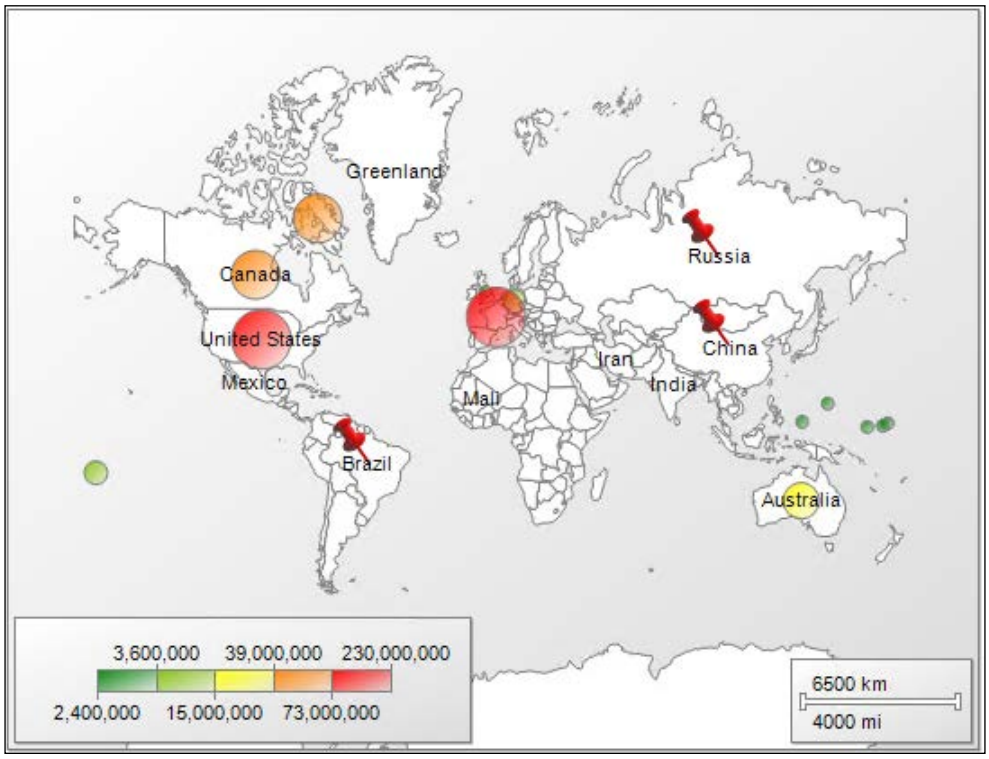
   Click the arrow next to **PointLayer1**, and select **Point Properties**. On the **General** tab, select the **PushPin** marker. Place the marker in the **Center**.

6. Also, change the color of the pushpins to **Red** in the **Fill** tab:



7. Click on **OK** when done.

Let's preview our map with the added pushpins:



Adding a line layer

You're probably familiar with seeing lines on map applications indicating a route or path. A line layer is quite similar to a point layer. They are both defined by their coordinates. The difference is that a line layer requires two sets of coordinates [(x1 y1), (x2 y2)], whereas a point has only an (x y) coordinate.

1. Create a new dataset with the following T-SQL command, and name it `ds_Mapline`:

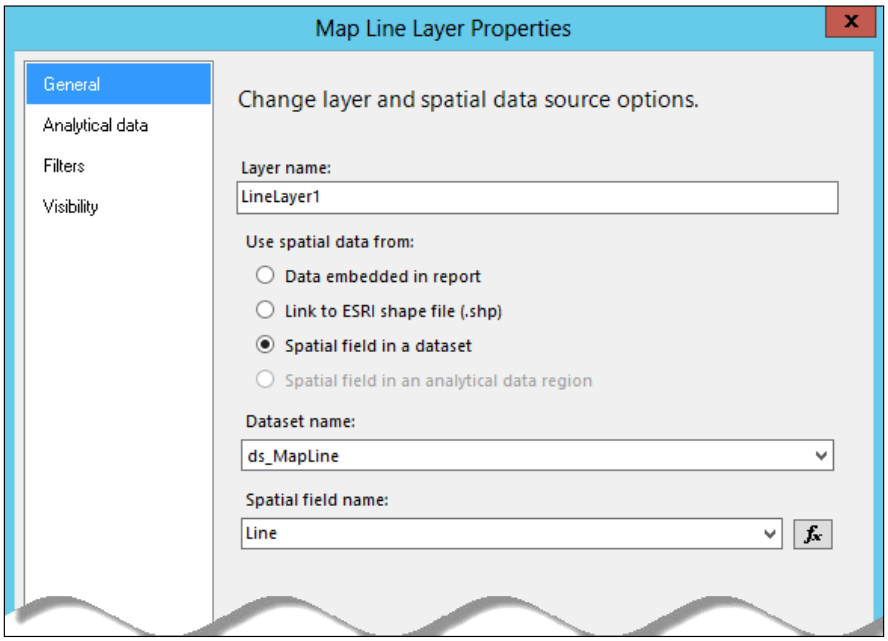
```
SELECT CAST('LINESTRING(
  -122.164644615406 47.7869921906598,
  16.0414569096018 49.6523079296896,
  -122.164644615406 47.7869921906598,
  2.32573072365188 48.8385347036971,
  -122.164644615406 47.7869921906598,
  6.93164071591788 51.5966635525063,
  -122.164644615406 47.7869921906598,
  20.0414569096018 49.6523079296896
)' AS GEOGRAPHY) as Line
```

   Or just find the script named `SpatialLine.sql` included in this chapter, and copy the T-SQL command.

   You may be wondering what these coordinates (spatial data) are; these are randomly selected from the `Person.Address` table in the `AdventureWorks2012` database.

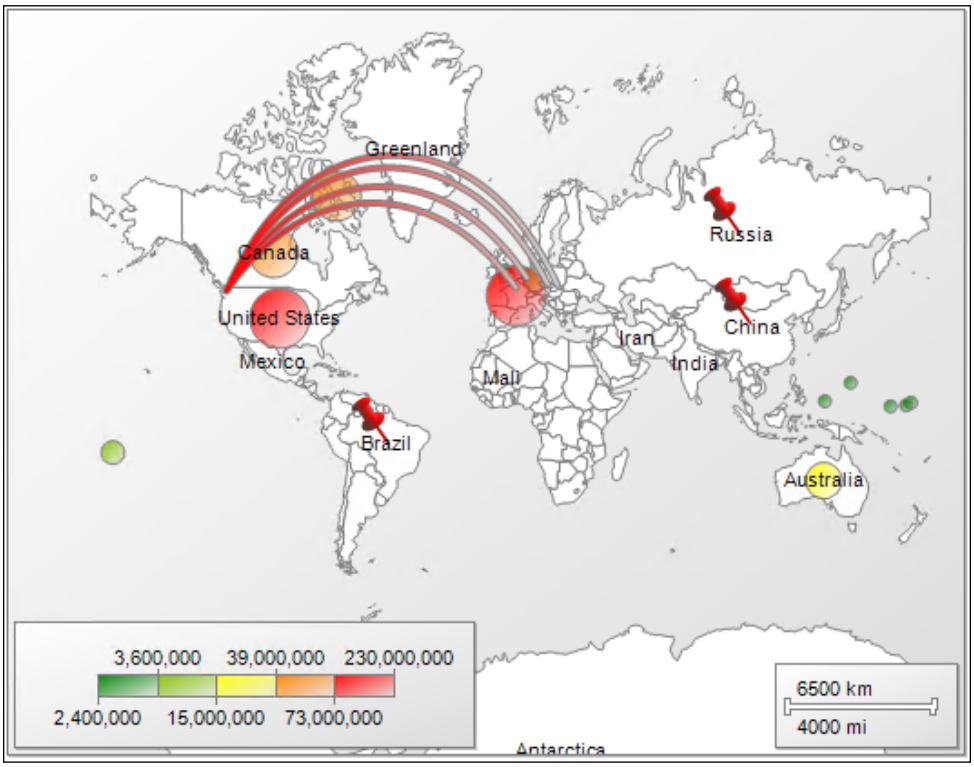
2. In the **Map Layers** pane, add **Line Layer**. Access the layer data property from the contextual menu of **LineLayer1** (of course the name of the layer would be different if you changed it), and select **Layer Data**.

3. In the **General** tab, select **Spatial field in a dataset**. Select **ds_Mapline** from the **Dataset name** drop-down textbox.
4. We only have one column on this dataset, **Line**, so select that for the **Spatial field name**.



5. Click on **OK** when done.
6. Change the line width to 1.25pt and the color to gradient red in the **Line Properties** window.
7. Click on **OK** when done.

Let's preview our map with the added line layer:



Adding a tile layer

Add a tile layer using the add layer button from the **Map Layers** pane. Change the **Tile Type** to **Road**.

That's it. You're done.

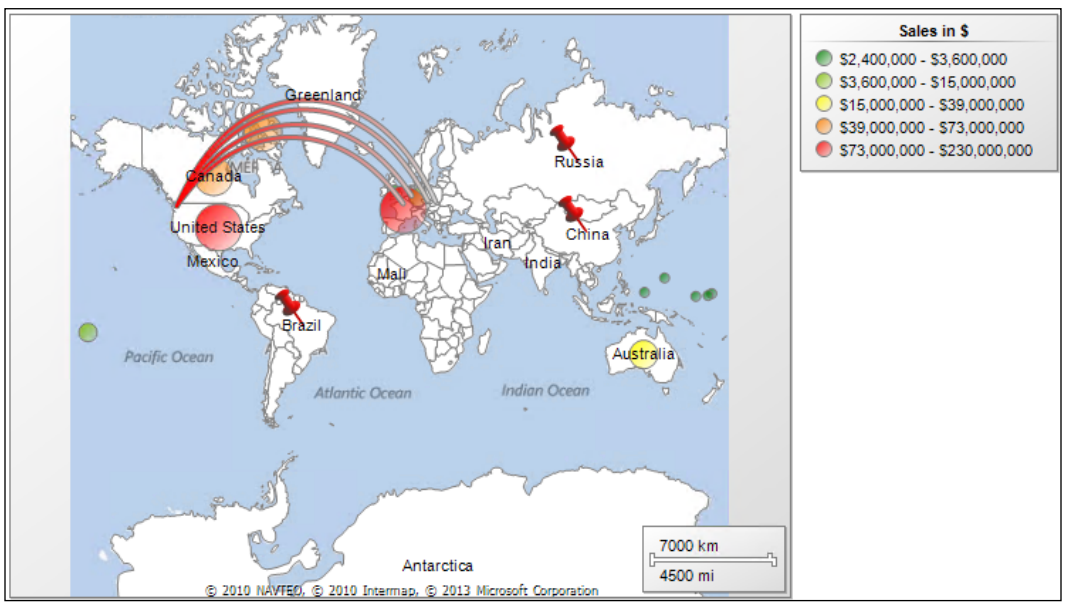
Additional Formatting

There are few more options that we can tweak to improve our map design. We have the options to either move the elements around in the map, or remove them altogether. Let's remove our map scale and format the legend:

1. In our map context, we don't really need **Map Color Scale**. Delete that by selecting the scale, then pressing the *Delete* key. You can also move it somewhere by grabbing-and-dragging it to any location.

2. If you also notice that the size of the bubble in the legend is redundant, because the size of the distribution is already obvious in the map itself. Let's delete that.
3. From the **Map Layers** pane select **PolygonLayer1**, and select **Center Point Size Rule** on its contextual menu.
4. In the **Legend** tab, select **nothing** from the **Show in this legend** drop-down textbox. That removes the bubble size in the legend.
5. Click on **OK** when done.
6. Now, let's go to **Center Point Color Rule**. On the **Legend** tab, change the legend text to **#FROMVALUE{C0} - #TOVALUE{C0}**, which converts the sales number from numeric to currency format. The zero (0) value indicates we don't want to add decimals.

Now, let's preview our new map with the added tile layer and the changes we made on the scale and legend:



Summary

Map reports with their associated spatial data give us a quick view of the meaning hidden behind the data. If you're dealing with demographics, you may want to take advantage of the tools designed to address this kind of data. SQL Server 2012 Reporting Services provides most of the tools you need to create and deploy spatial visual reports.

6
Let's Get Analytical!

Seeing that the reports developed by John Kirkland are greatly helping the company understand its performance, the management decided to explore the use of SQL Server Analysis Services (SSAS). The company has accumulated huge amount of data from various sources, and SSAS is the perfect tool to implement the company's analytics initiative.

The company's senior BI Developer designed an Online Analytical Processing (OLAP) database and it is ready for deployment on the Analysis Services server.

John was tasked to develop reports based on the cubes in the OLAP.

There is a big advantage in using an analytic engine in handling Big Data. The SQL Server database engine is of great use in storing and retrieving relational data. But once you demand complicated calculation and aggregation, the database engine can only do so much.

Now that the company has embraced analytics, John can slice and dice their data and, thus, can have a deeper understanding of what those data mean. John is now more capable of delivering meaningful data to report consumers.

Please take a note that we are not going to step-by-step cover anything on creating dimensions or cubes on Analysis Services, aside from setting up the deployment of Analysis Services project to the server. In this chapter, you'll learn the following:

- Checking port that the Analysis Services listens to
- Setting up a new inbound rule in the firewall for SSAS deployment
- Deploying model for Analysis Services
- Adding SSAS data source
- Working with the query designer
- Building and using a dataset based on a cube
- Adding a filter on the dataset
- Adding a parameter on the dataset
- Creating a calculated member

We need the following files from **CodePlex** for this chapter to build the data source for our report:

- `AdventureWorksDW2012 Data File`
- `AdventureWorks Multidimensional Models SQL Server 2012`

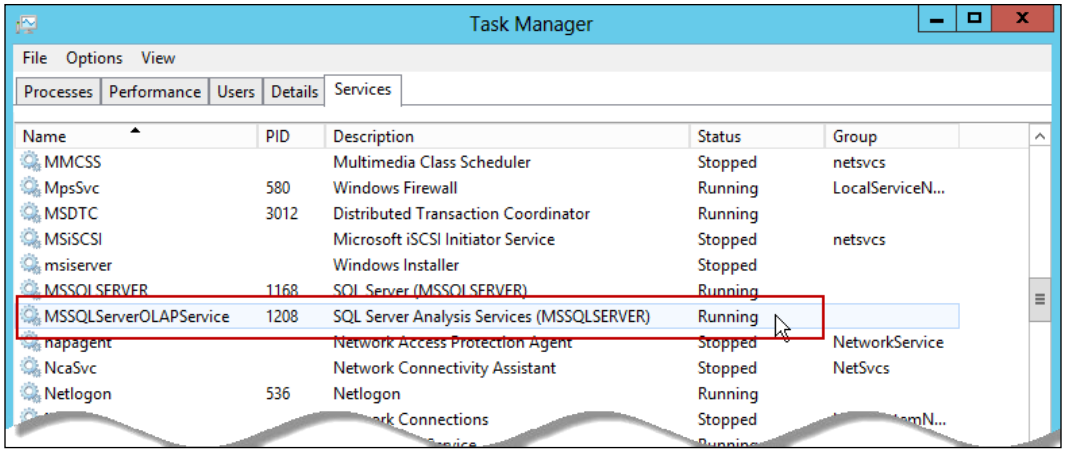
Here is the link to the download page: `http://msftdbprodsamples.codeplex.com/releases/view/55330`

Checking the TCP port of Analysis Services

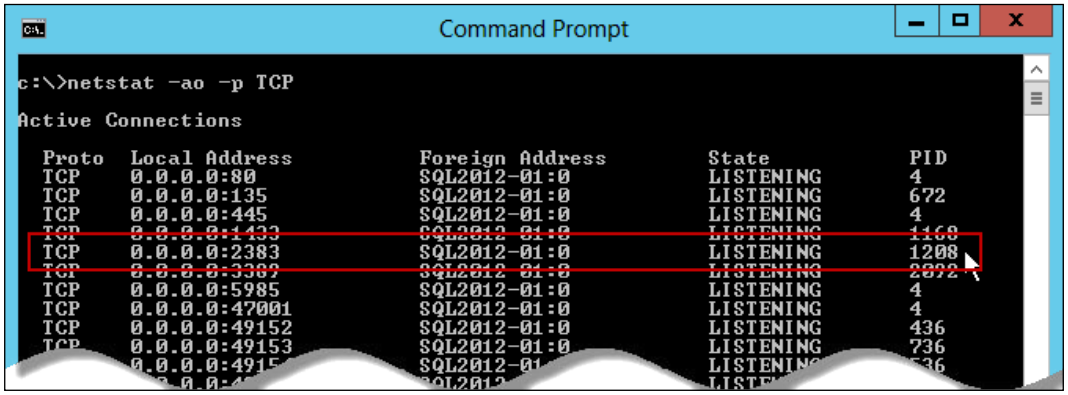
If this is the first time you deploy an Analysis Services model, chances are you need to set up the firewall of the server where the Analysis Services is installed. You may also need these settings if you cannot log in from a remote SQL Server Management Studio session but can log in without any problem locally.

You may not need this firewall setting if you are deploying the model from the same server your Analysis Services is installed.

1. Get the Process ID (PID) of the Analysis Services running in your server, as shown in the following screenshot:



2. To view the TCP Port of PID `1208`, run `netstat -ao -p` TCP from the command line:

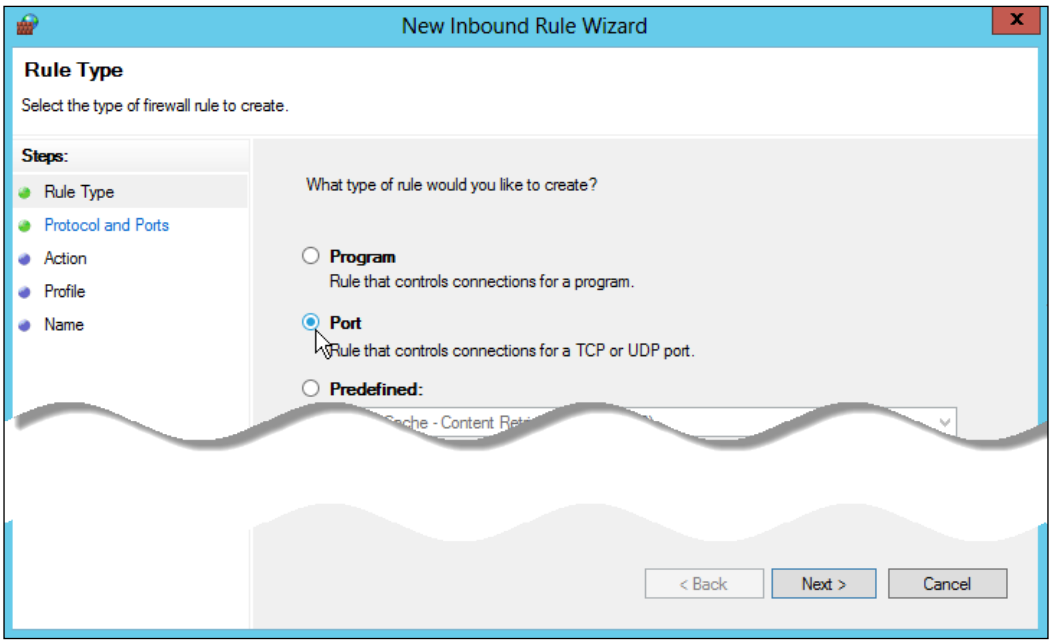


> SQL Server Analysis Services listens to TCP port `2383` by default; but it reserves both ports `2382` and `2383`. Run the `netstat` command to figure out what port Analysis Services is currently listening to.

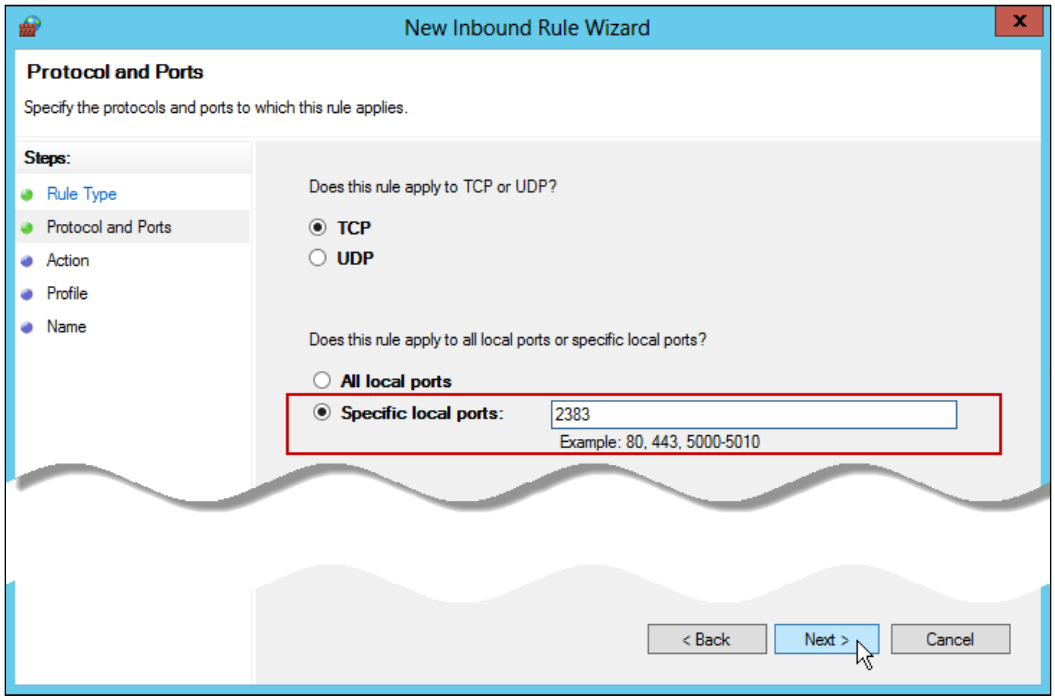
Creating a new inbound rule for Analysis Services

Now that we have the correct information of the TCP port, let's create an inbound rule that would allow connections to Analysis Services.

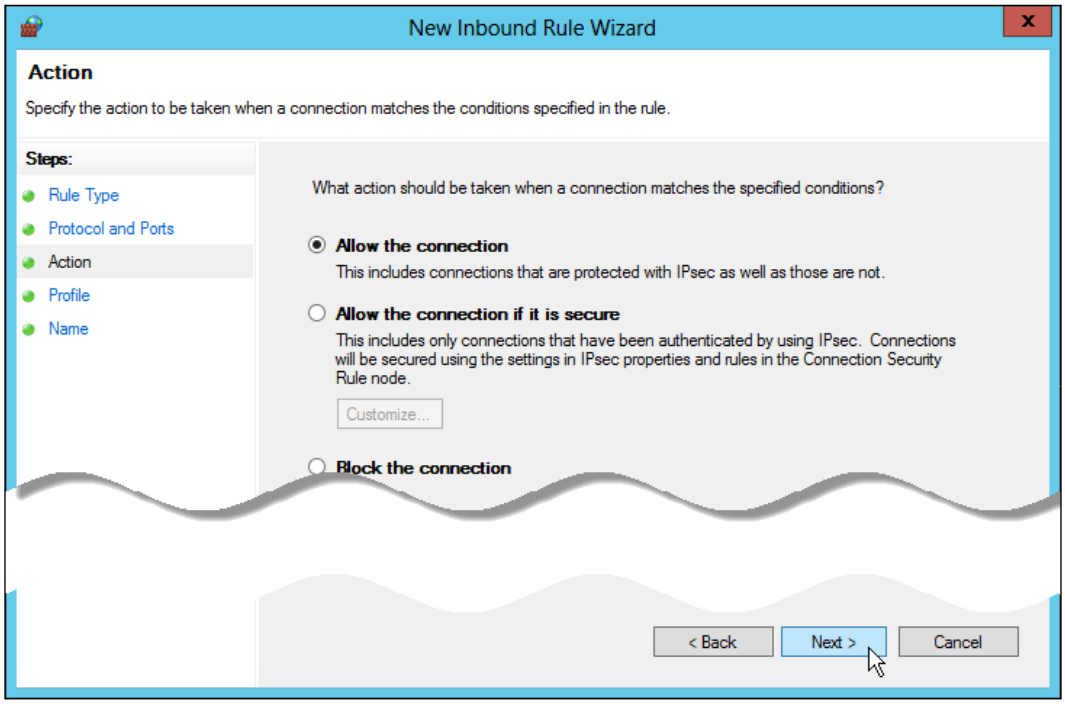
1. From the **Administrative Tools**, open **Windows Firewall with Advanced Security**, and create a new inbound rule (TCP Port). Click on the **Next** button:



2. Create a new inbound rule to allow connections via TCP port `2383` (or `2382`) in the firewall setting of the Analysis Services server. Click on the **Next** button:



3. Select the appropriate **Action** for the inbound rule. In this case, we select **Allow the connection**. Click on the **Next** button:



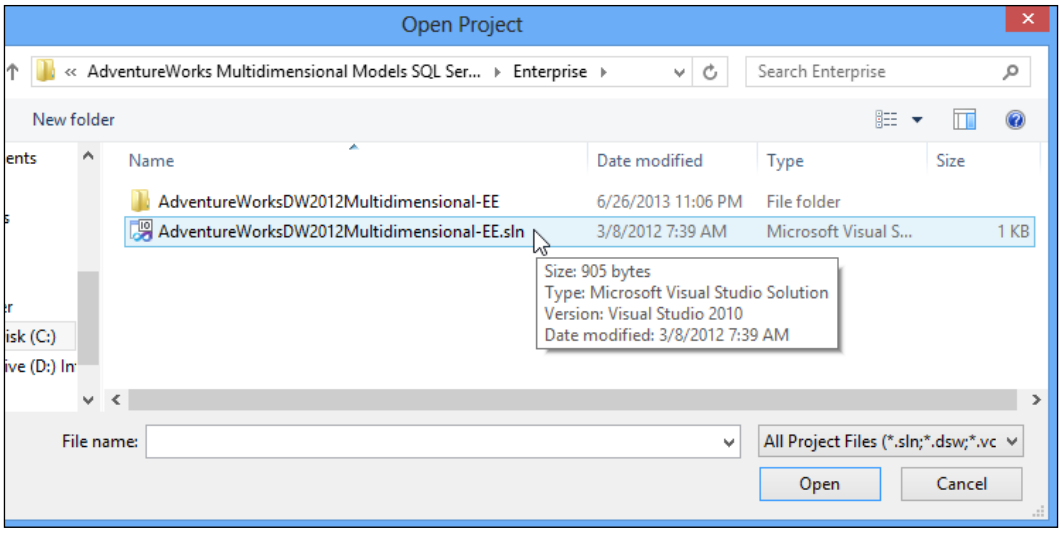
4. Select the appropriate **Profile** (**Domain**, **Private**, or **Public**) when the inbound rule applies. Click on **Next**, and give the new rule a **Name**, then click on **Finish**.

Changing the deployment setting for Analysis Services

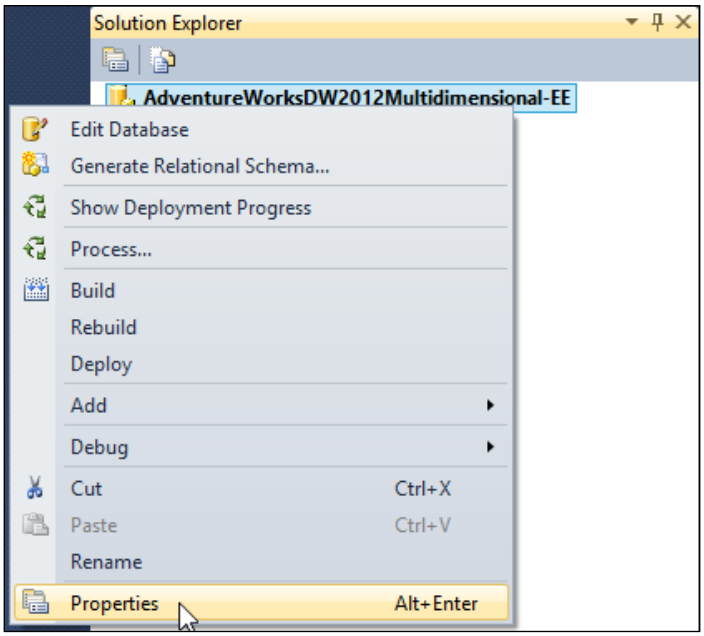
Attach the `AdventureWorksDW2012 Data File` into a new database in a SQL Server 2012 instance. We will use this data warehouse as a source to build our multidimensional model on the Analysis Services.

Let's deploy the SSAS model (in this case, the enterprise version):

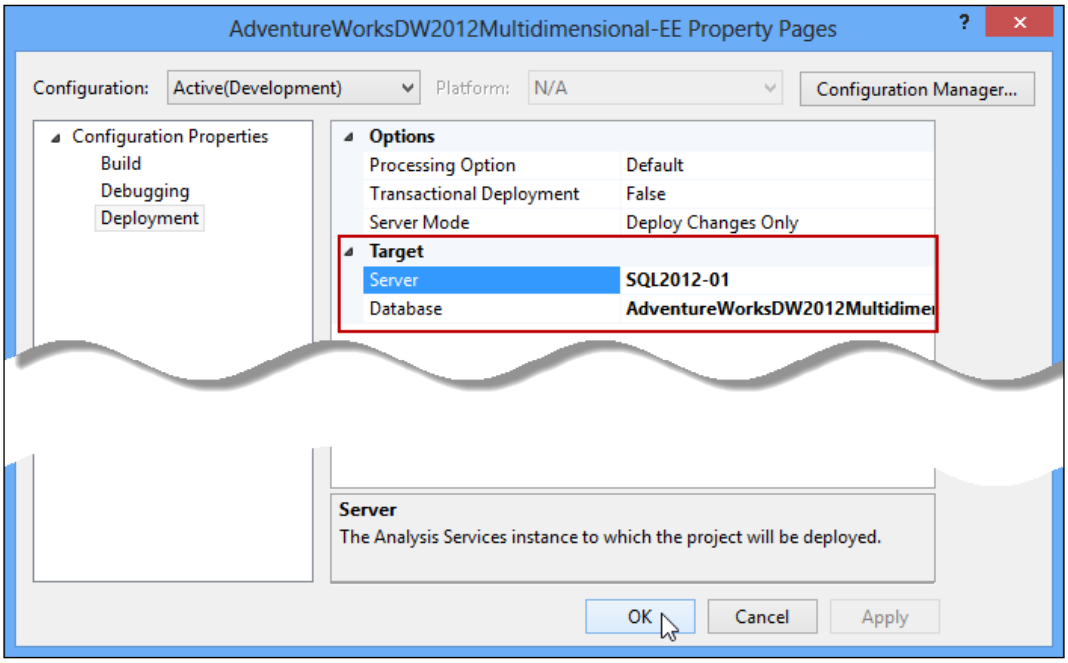
1. Let's open the solution file included in the multidimensional file that we have just downloaded:



2. In the SQL Server data tools designer section, right-click on the `AdventureWorksDW2012Multidimensional-EE` project, and select the **Properties** option from the contextual menu:

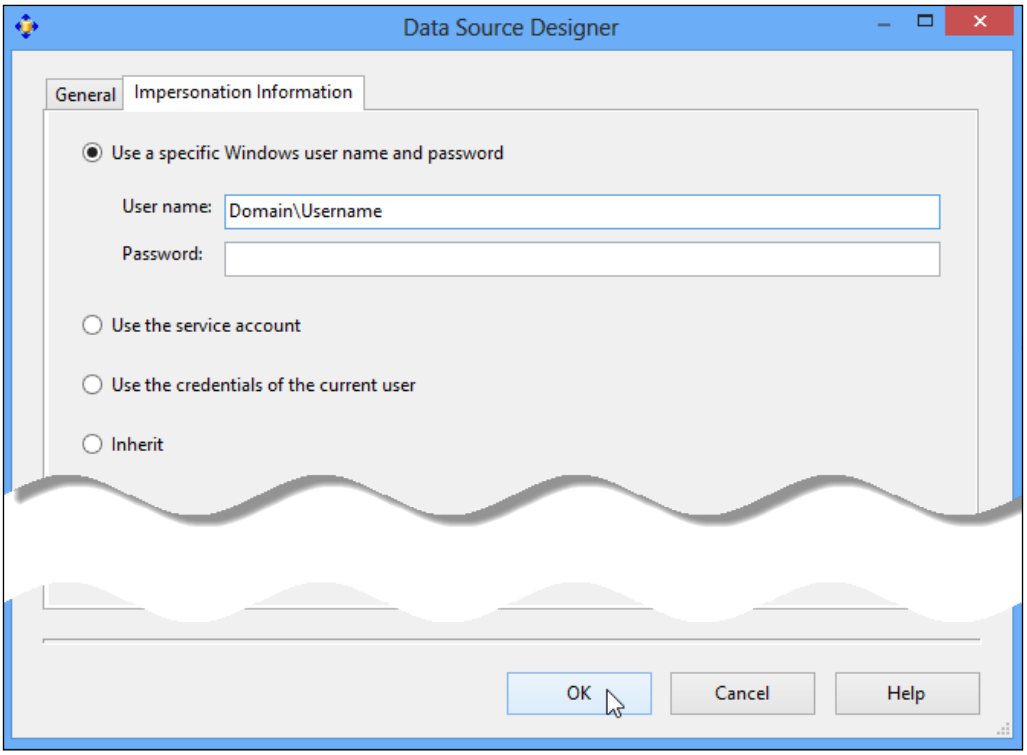


3. Click on **Deployment** and specify the **Target Server**; you may list **localhost** if you're deploying locally. The target server in this case is called `SQL2012-01`.



4. Let's check the connection string of the data source of our project. Make sure that it is connected to the `AdventureWorksDW2012Multidimensional-EE` data warehouse database.

5. Let's also change **Impersonation Information** and select **Use a specific Windows username and password**.



Consult with your network administrator for the proper setting for impersonation information. For more information about user access, check *Grant User Permissions on an Analysis Services Multidimensional Database* from TechNet. The link is `http://technet.microsoft.com/en-us/library/ms175408.aspx`.

Deploying the multidimensional model

Now that we have set up the necessary settings to connect our project to the target Analysis Services server, we are ready to deploy our model:

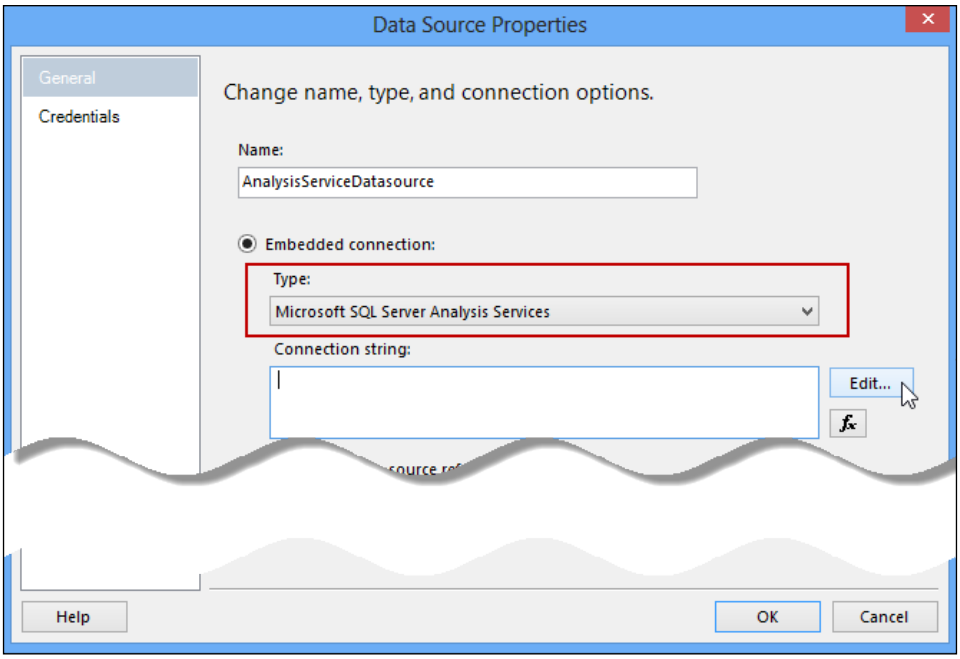
1. From the **Solution Explorer** window, right-click on the project, then select **Deploy**.
2. You should be able to deploy the project to the Analysis Services server properly. If you encounter errors, check the error information, and check your connection and data source information again making sure that your properties are set up correctly.

Creating the data source for the report

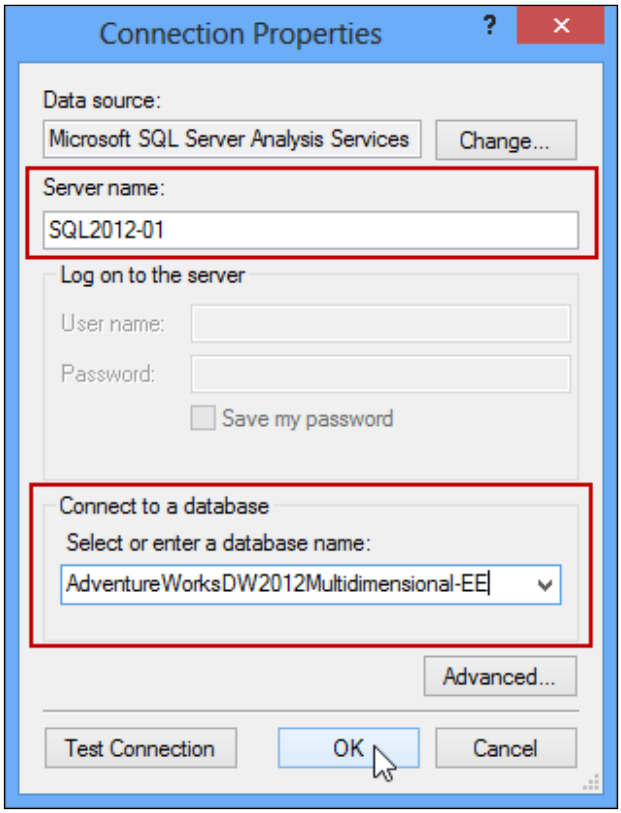
After the successful deployment of the multidimensional model for Analysis Services, close the project and create a new project. Now, create a new report.

Let's define a data source for our report:

1. Add a new report data source in the **Report Data** window. Give your data source a name; in our case we put `AnalysisServiceDatasource`. Select **Microsoft SQL Server Analysis Services** under the connection type:



2. Click on the **Edit** button of the **Connection string** box, and specify the properties for our connection:



3. Test the connection to make sure you have the correct information to connect to the Analysis Services server. Then, click on the **OK** button.

Adding a dataset based on the multidimensional model

Now that we have set up a data source to connect to the Analysis Service database, we need to define a dataset for our report, which is based on a cube in the multidimensional model we deployed in the previous demonstration.

If you've created datasets in the exercises in previous chapters, there shouldn't be any difference up to this point.

1. Right-click on the `Dataset` folder in the **Report Data** window. Select **Add Dataset**, which displays the **Dataset Properties** dialog box.
2. Name your dataset. Then, select **Use a dataset embedded in my report**. Select the data source we created above **AnalysisServiceDatasource**, and select **Text** for the query type.

You can either manually write the query or use the query designer to build your dataset. If you don't know the Multidimensional Expressions language, you'll probably build your first dataset using the query designer.

Working with the query designer

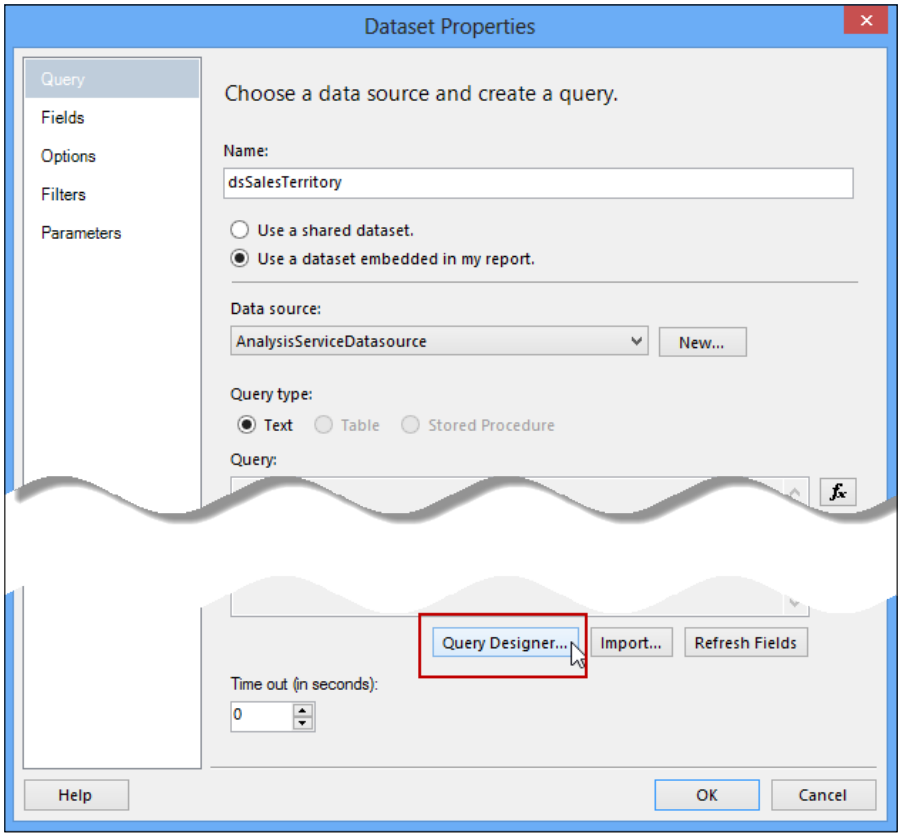
You are probably familiar with the T-SQL language that we use to query data from SQL Server, that is, OLTP. But Analysis Services requires a different language for querying OLAP cubes, which is called **Multidimensional Expressions** or, simply, **MDX**. For more information on *Querying Multidimensional Data with MDX*, check `http://msdn.microsoft.com/en-us/library/bb500184.aspx`.

Another language used in SSAS is the prediction query language called **Data Mining Extensions** or **DMX**. DMX is used in data mining model to predict unknown values in a dataset. You also probably heard about **Data Analysis Expressions** or **DAX**, which is a formula language used in PowerPivot workbooks in Excel.

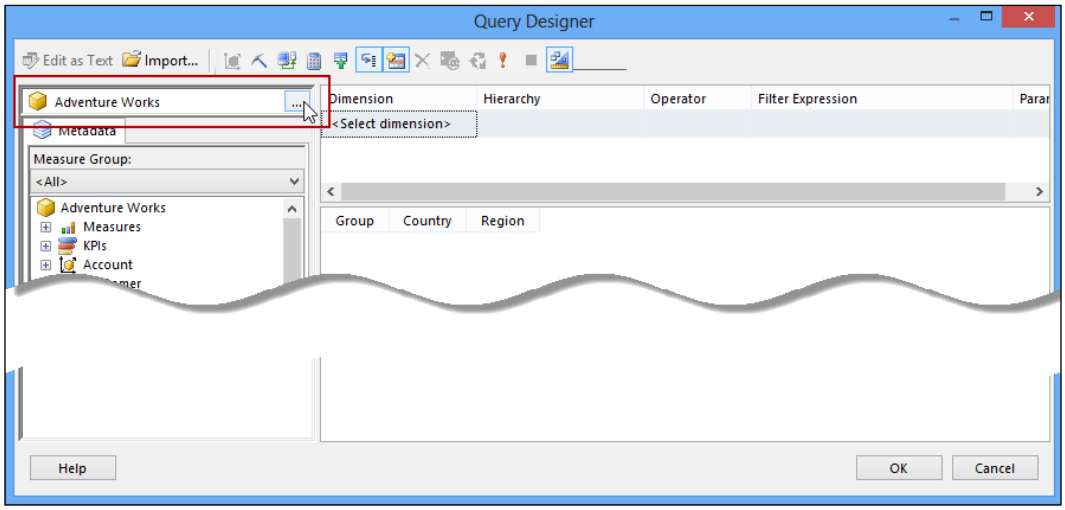
You need to learn and understand MDX if you work with multidimensional data. If you aren't familiar with MDX at the moment, reporting services comes with a query designer for multidimensional data.

Let's create a dataset using the query designer:

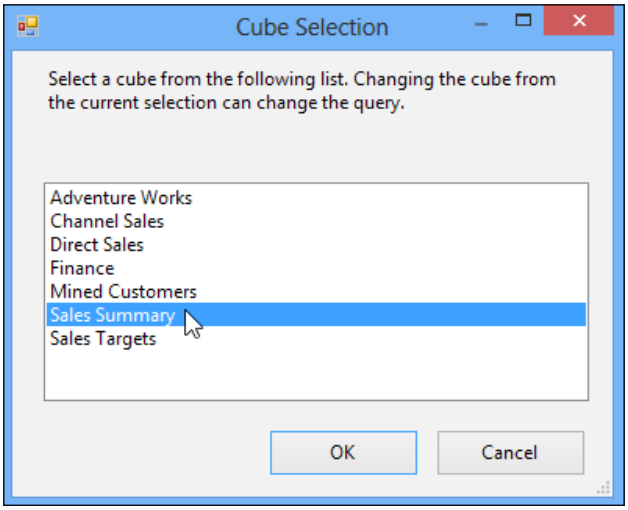
1. From the **Dataset Properties** dialog box, click on the **Query Designer** button underneath the query box.



2. We can change the cube by clicking on the ellipsis button next to the cube textbox on top of the left-hand side of the query designer.



3. That displays the **Cube Selection** dialog box. Select the cube appropriate to your report as shown in the following screenshot:



4. Get yourself familiarized with the buttons by hovering and reading the title of the buttons which indicates the associated actions.

Building a dataset using the query designer

With the query designer, you can build a dataset by dragging-and-dropping metadata from your cube to the designer. There is no easier way to build a dataset based on a cube than by the drag-and-drop method.

> It helps to know the basic data warehouse concepts: schemas, dimensions, facts, and measures.

In a basic cube model, you have facts and dimensions which are attributes that correspond to a particular business process or transaction.

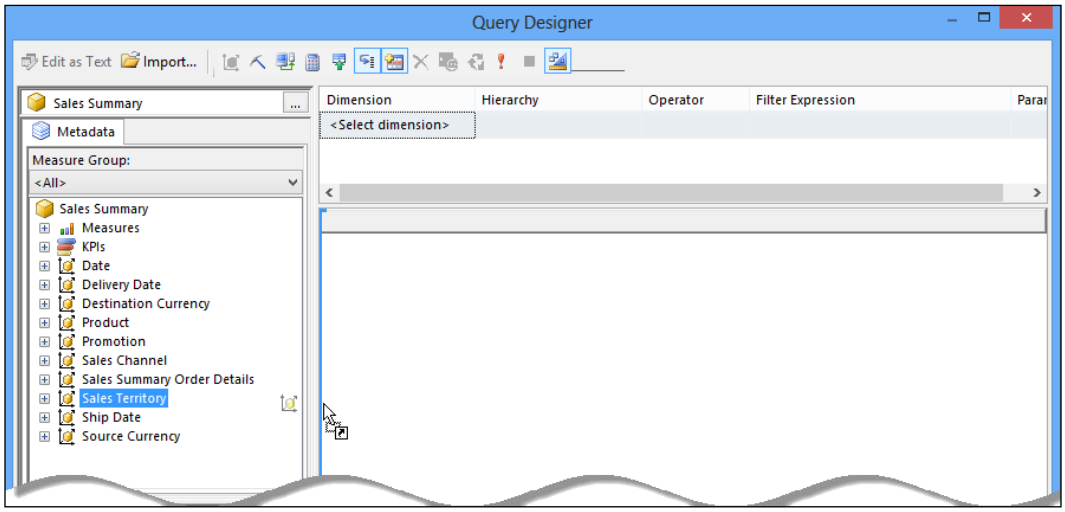
Let's take sales as a business process. The facts in sales are prices and quantities. Facts are usually numeric. On the other hand, dimensions are the context in which the sales transaction happened. In this case, the dimensions are date when the actual sales took place, the product sold, the sales person, and other related attributes.

To fully understand the performance of your sales, you need some kind of metrics; you need to measure the sales by means of aggregation (sum, average, and so on.). These metrics are represented by numeric values called measures.

For our purpose, let's focus on measures and dimensions:

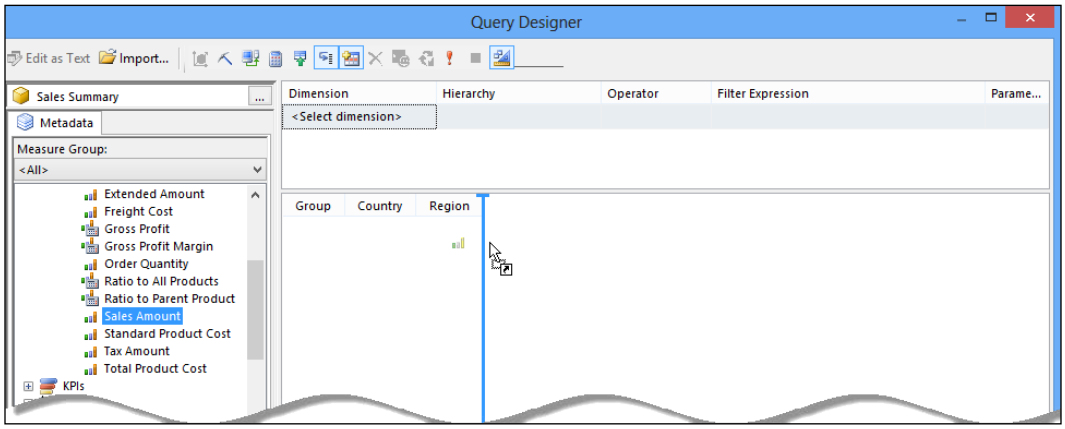
1. Select the **Sales Summary** cube from the **Cube Selection** dialog box. Click on **OK**.

2. Select the **Sales Territory** dimension, and drag it to the designer as shown in the following screenshot:

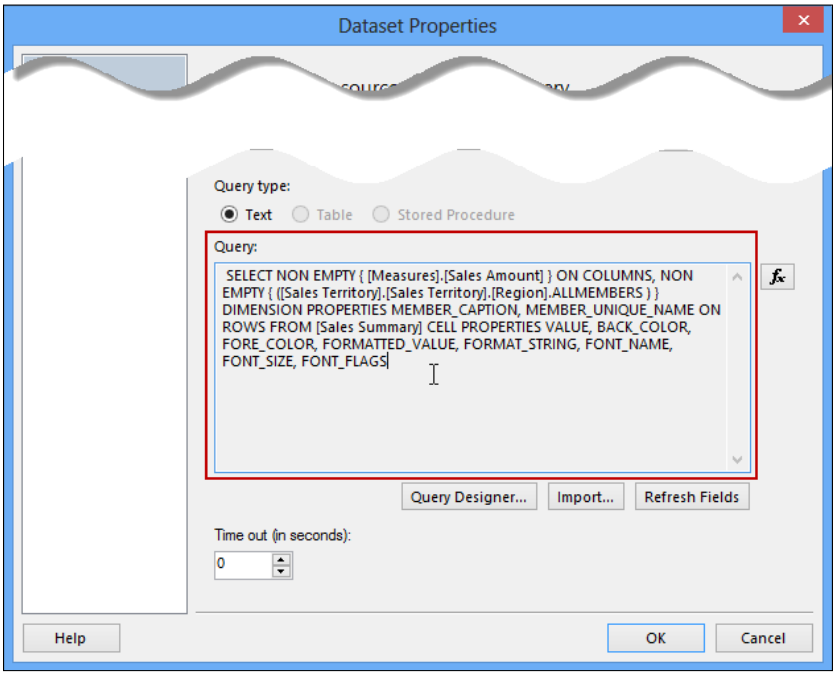


You may notice that it doesn't give you any data except for the three columns: **Group**, **Country**, and **Region**. You may also notice the message in the middle of the designer that says **No rows found. Click to execute the query**. What we have so far is the context of the Sales Territory, where sales occurred. Now we need the measure:

3. Expand the **Measure** metadata. Then expand **Sales Summary**. Select and drag-and-drop the **Sales Amount** measure to the designer as shown in the following screenshot:



4. Now click on **OK** in the designer. Going back to the **Dataset Properties** dialog box, you may notice that the query designer converted the design to a text query. This is the MDX language. Move around metadata in the query designer and pay attention to how that changes the MDX query. This is one way of learning the MDX language.

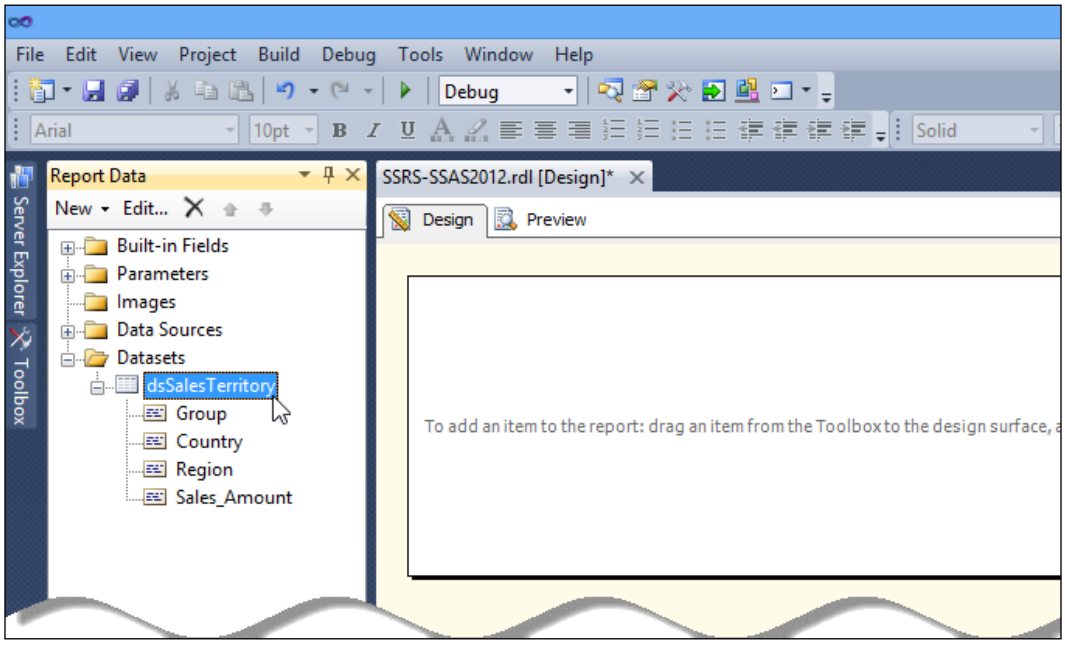


5. Name the dataset `dsSalesTerritory`. Click on **OK** when done to close the **Dataset Properties** dialog box.

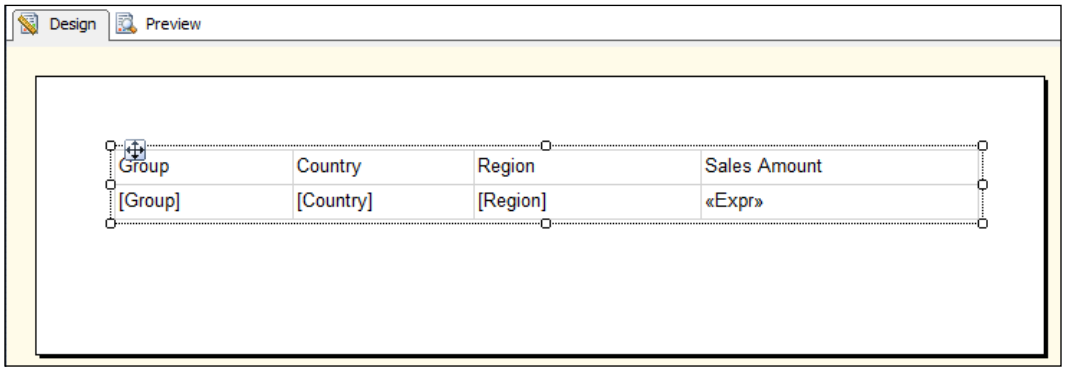
Using the dataset on the report designer

Using a dataset derived from a multidimensional model or cube is no different than using a dataset derived from a T-SQL query or stored procedure. The only difference, a very good difference, is that the Analysis Services is doing the calculations and aggregations on the analytic engine, and not on the database engine.

1. From the report designer, expand the `dsSalesTerritory` dataset. You may notice that there are four fields in this dataset: `Group`, `Country`, `Region`, and `Sales_Amount`.



2. Now, create a table with four columns. Drag-and-drop the four fields of our dataset into the four columns of the table. Format the columns appropriately.



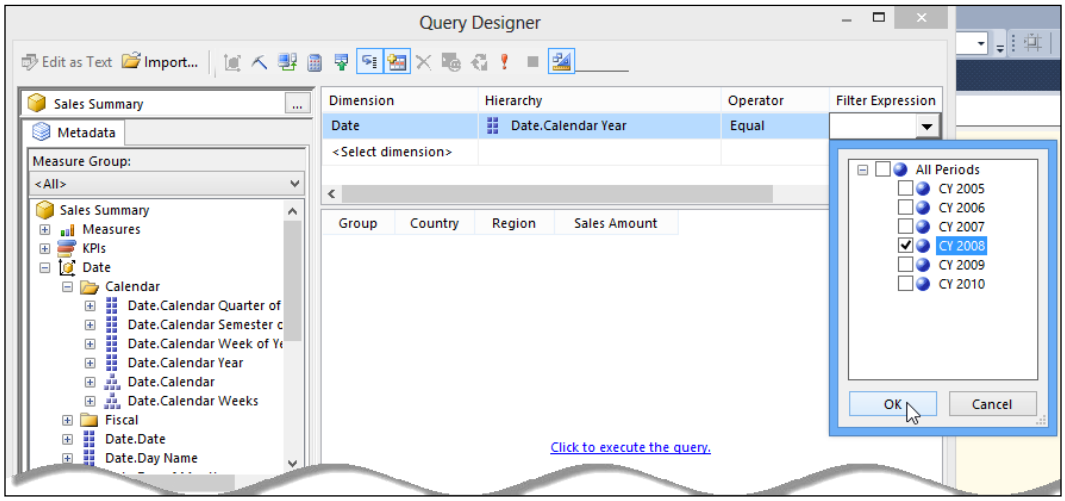
Adding a filter in the dataset

You may want to filter your dataset by an attribute from another dimension, such as the Date dimension. The query designer provides the tools needed to slice and dice the data with multiple dimensions. We may not call the multidimensional model a "cube" if we cannot examine the data in different dimensions.

Let's filter the dataset by the sales amount in the year 2008 only. Normally, we want to test the most recent data but we're ok with the 2008 data for now.

1. From the report designer, double-click on the `dsSalesTerritory` dataset.
2. That displays the **Dataset Properties** dialog box. Click on the **Query Designer** button which will load the query designer dialog box.
3. You can add a filter in two ways:
    - Expand the `Date` dimension. Then, expand the `Calendar` attribute. Select the `Date.Calendar Year` attribute hierarchy. Drag it to the filter pane.
    - On the filter pane, click on the first field under the **Dimension** column. Click on the down-arrow and select `Date` from the options. Select `Date.Calendar Year` on the **Hierarchy** column.
4. Select **Equal** under **Operator** column.

5. In the **Filter Expression** column, click on the down-arrow. Expand **All Periods**, and select **CY 2008**.



6. Click on **OK** when done.

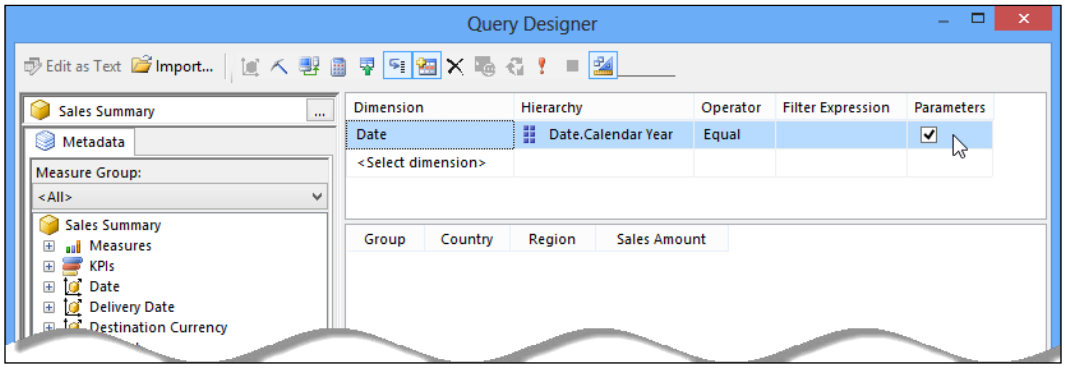
Adding a parameter in the dataset

There are situations where we need to allow our report consumers to decide which attribute they want to filter their data with. We use a runtime parameter in these cases. Adding a parameter in a dataset based on a cube in the query designer is as easy as selecting a checkbox.

Let's convert our `Date` filter into a parameter:

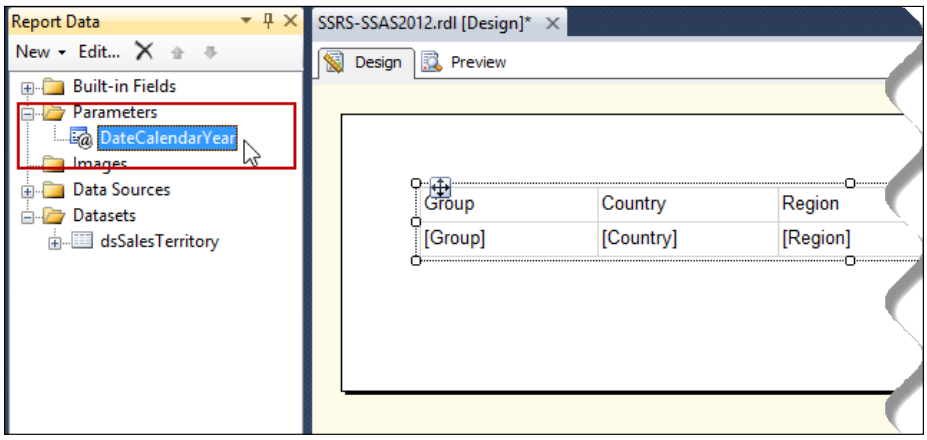
1. Start by modifying our `Date` filter in the filter pane. We will need the same dimension and hierarchy, that is, `[Date]` and `[Calendar Year]` respectively. You may need to delete the existing filter we created previously.
2. Set the **Operator** to **Equal**.
3. Leave the **Filter Expression** empty; or select a value, which assigns a default year value on runtime.

4. Scroll to the far-right if you don't see the **Parameters** column. Select the checkbox to indicate that we want this `Date` filter as a dataset parameter.

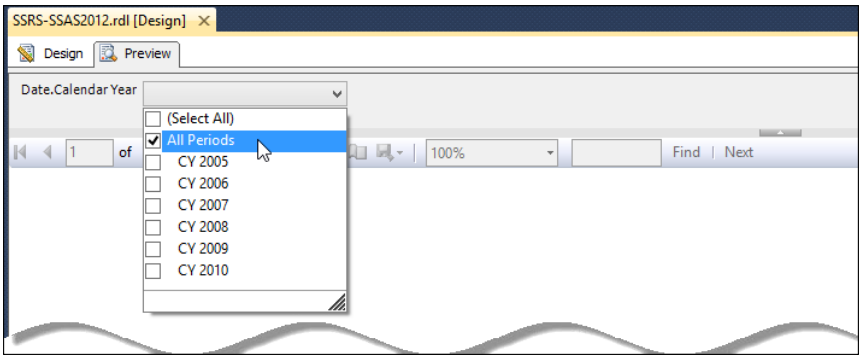


5. Click on **OK** twice to go back to the report designer window.

You may notice that the report designer now includes a report parameter named `@DateCalendarYear` as shown in the following screenshot:



Let's preview the report, and check the added parameter:

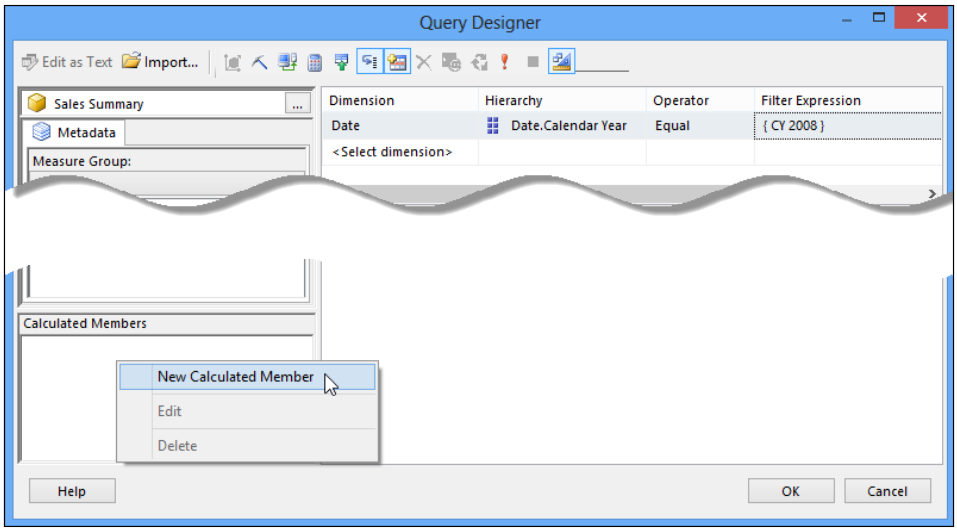


Creating a calculated member

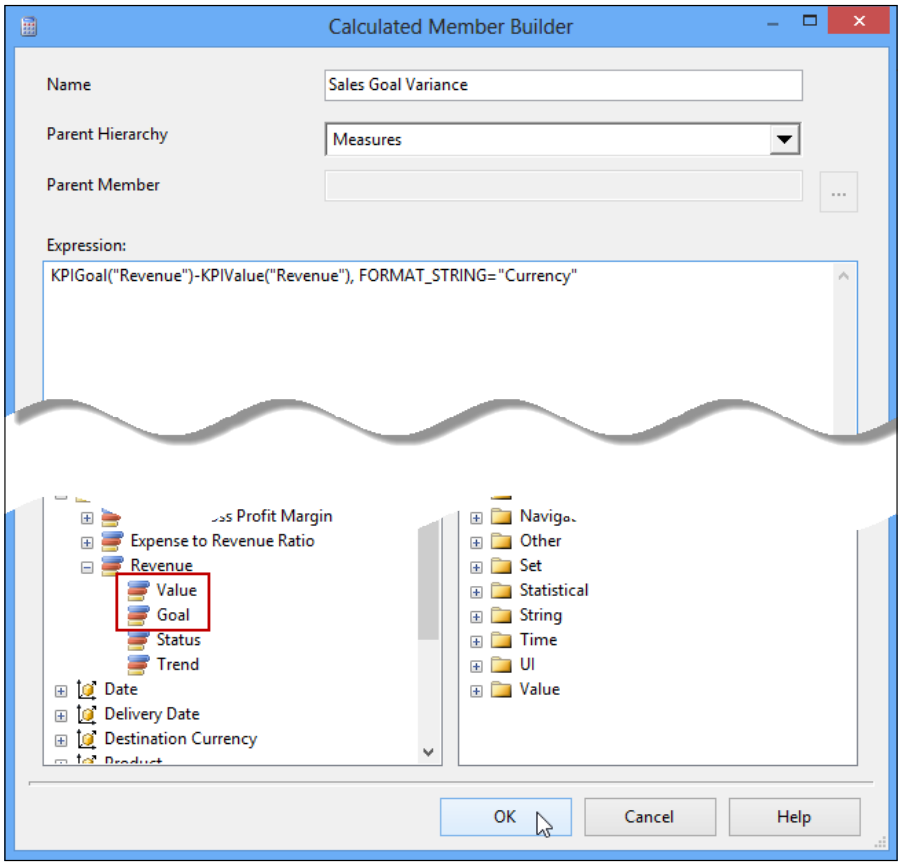
Since we want to take advantage of the power of the analytic engine, we want all of the aggregations in our report to be calculated in the cube rather than in our report server. This is an efficient way of handling the calculations.

Let's say we want to find out the variance between the sales goal and the actual sales of each territory in dollars. We can handle this simple calculation using the query designer:

1. On the query designer, right-click on the empty space in the **Calculated Members** pane on the lower left-hand side of the designer, as shown in the following screenshot:

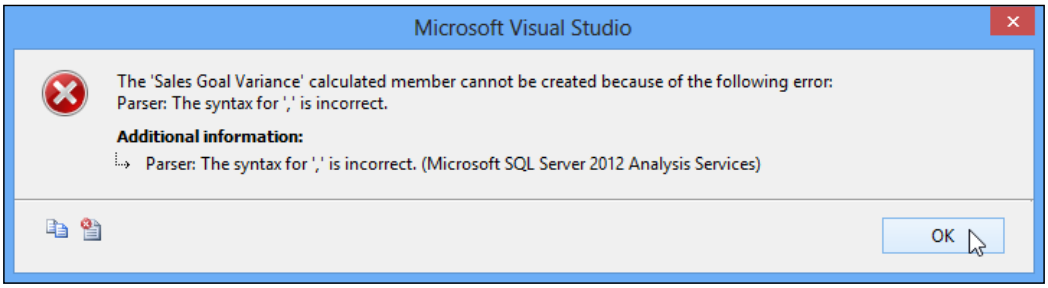


2. That displays the **Calculated Member Builder** dialog box. Let's name this calculation `Sales Goal Variance`.
3. Expand **KPIs** in the **Metadata** pane. Expand **Revenue**.
4. Drag **Value** and **Goal** to the **Expression** pane. Insert a minus sign (-) between the two values to indicate subtraction. You can also write the expression directly to the **Expression** pane.
5. Format the value by adding a format property to the calculated value like this: `Format_String = "Currency"`.

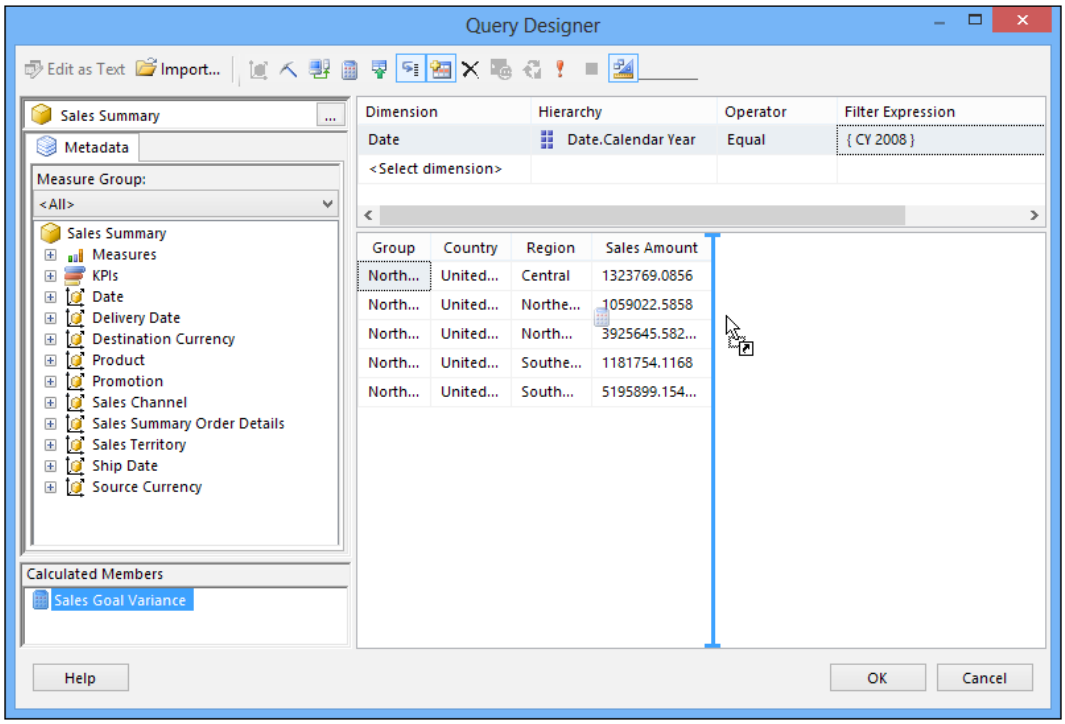


Find more information on string format from TechNet see *FORMAT_STRING Contents (MDX)* at `http://technet.microsoft.com/en-us/library/ms146084.aspx`.

6. Click on **OK** when done. You may encounter a parse error but you can ignore it for now. The syntax passes the syntax check.



7. Click on **OK** in the error box.
8. In the **Query Designer** window, drag the **Sales Goal Variance** to the existing table:



9. Click twice on **OK** to return to the report designer.

Summary

In this chapter, we've seen the basics of integrating SQL Server Analysis Services (SSAS) into SQL Server Reporting Services (SSRS). SSRS provides the tool to quickly facilitate this integration with the use of the query designer for SSAS. Although learning Multidimensional Expressions is essential in implementing a successful SSAS-SSRS project, we can probably survive simple implementation just with the query designer.

7
Sir, We're Ready to Deploy!

John Kirkland just came out of another meeting with the Finance Manager, Linda. She has been thrilled with his work and has asked him if he is up for another challenge. He plans to accomplish her requests by setting up some subscriptions and snapshots for the reports.

She has asked that on the first of the month each sales person should receive a PDF document containing their commission for the previous month. She was very concerned with security for this report, because each sales person should not see anyone else's commission. This won't be a problem for John because he can limit access to the report by setting up security for the report. He can also create the report so that only a single sales person's commission shows in each PDF document. John thinks this is a great opportunity for using caching. He plans to cache the report data so that the SQL Server will only be asked once for the data. Then, he can deliver each individual report to each sales person from the cached data.

Linda has also asked that the same report be saved once a month with everyone's commissions so that it can be helpful for any future auditing needs, and that a copy be sent to her e-mail as well. John knows that this request can be accomplished by creating snapshots of the report and placing them in the `Report History` folder where the Finance Manager can reference them.

On his way back to his desk, John found out some good news from his manager. He has been doing such a great job of creating reports, so his manager wants him to train someone from the operations team on how to deploy the reports to production.

Let's follow John while he accomplishes his latest set of tasks.

In this chapter, you will learn how to:

- Deploy reports from within SQL Server Data Tools
- Deploy reports manually

- Set up user security
- Set up a shared schedule
- Set cache settings on a report
- Set up snapshots
- Set up an e-mail subscription and a data-driven subscription

First, find the T-SQL files named `uspSalesCommissions.sql` and `uspSalesCommissionsSubscription.sql` (included with this book) and execute them in the `Adventureworks2012` database. `uspSalesCommissions.sql` is used for the sales commission report and `uspSalesCommissionsSubscription.sql` will be used for the data-driven subscription later on.

Let's take a look at `uspSalesCommisison.sql`. In *Chapter 3*, *Getting Some Actions*, we learned about using parameterized queries. For this report, we will need two parameters. One to pass in today's date to calculate last month's beginning and end date, and one to narrow the data to just one sales person. Since we are going to use caching for this report, we don't want to pass the stored procedure, the sales person's `SalesPersonID`. We want to make that parameter an internal parameter, which we will do in a minute.

For calculating the beginning and ending dates of last month, we are going to leverage a new function that SQL Server 2012 has called `EOMONTH`. This function has two parameters. The first parameter is the date that will be used to find the last day of the month. The second parameter is used to move to a different month. `EOMONTH` can be used in combination with the `DATEADD` function to calculate the first day of a specific month.

```
CREATE PROCEDURE [dbo].[uspSalesCommissions]
(
    @RequestedDate AS datetime
)
AS
    SET NOCOUNT ON;

    DECLARE
        @StartDate AS datetime
       ,@EndDate AS datetime

    SELECT
         @StartDate = DATEADD(day, 1, EOMONTH(@RequestedDate, -2))
        ,@EndDate = EOMONTH(@RequestedDate, -1)
```

We need the first day and the last day of the month prior to the `@RequestedDate` parameter. For the first day, we can go back two months then add one day. This pushes the date to the first of the previous month. The end of the previous month is calculated by using `-1` for the second parameter of the function.

Find the report file called `Sales Commissions.rdl` (included with this book) and add it to your project. It uses the same data source as the previous chapters.

Let's open up the `Sales Commissions.rdl` report and make a few adjustments to it so that it will satisfy all of Linda's requirements.

First, let's add the internal parameter, `SalesPersonID`, which we mentioned previously. It will have a default value of `-1`. This value will tell the report that it should generate a report with all the sales people's IDs. We will also make the parameter hidden so that no one will see it. This parameter will also be hidden to the users when they run the report, but we'll be able to leverage this parameter in the subscriptions which we will create later in the chapter. Let's perform the following steps:

1. Right-click on **Parameters** in the **Report Data** window, and select **Add Parameter**.
2. Name the parameter `SalesPersonID`.
3. Set the **Prompt** property to `Sales Person ID`.
4. Set the **Data Type** property to **Integer**.
5. Set the **Parameter Visibility** property to **Hidden**.
6. Click on the **Default Values** menu item on the left-hand side.
7. Click on the **Specify Values** option button.
8. Click on the **Add** button.
9. Set the **Value** property to `-1`.
10. Click on the **OK** button.

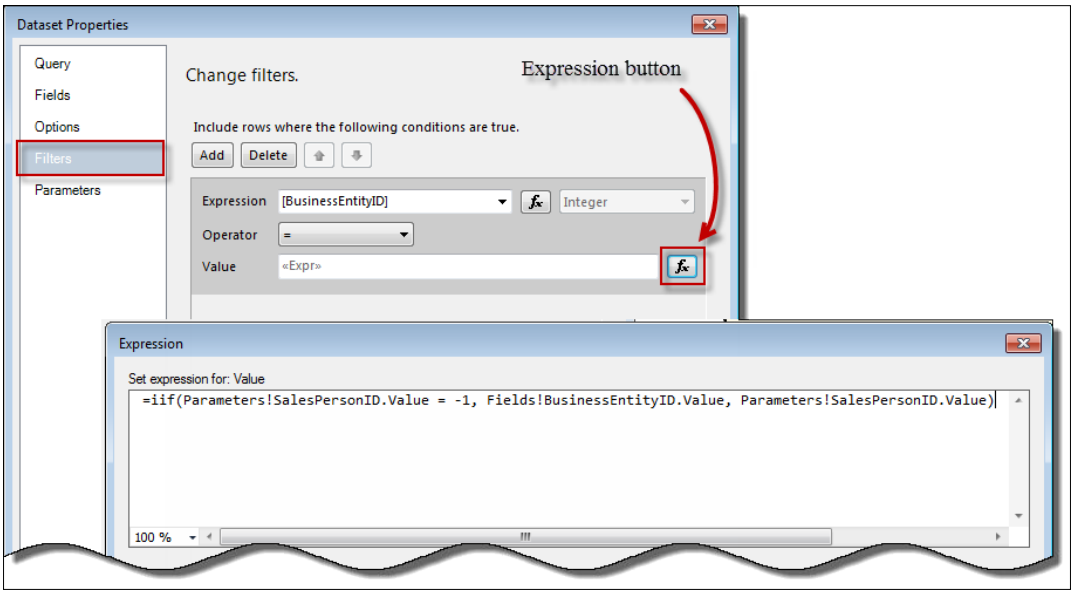
For unit testing you want to have the **Visibility** property of the `SalespersonID` parameter set to **Visible**, but remember to set it back to **Hidden** before deploying the report.

Now, let's set up a filter using the `SalesPersonID` parameter we just created. Since the data for the entire report will be filtered, we are going to set up the filter on the dataset. This filter will be leveraged when we set up the Data Driven Subscription later on.

1. Right-click on the dataset `uspSalescommissions` in the **Report Data** window, and select **Dataset Properties**.
2. Click on the **Filters** menu item on the left-hand side.
3. Click on the **Add** button.
4. Set the **Expression** property to **[BusinessEntityID]**.
5. Click on the Expression button for the **Value** property.
6. Create the following expression.
7. Click on **OK** on the **Expression** window and the **Dataset Properties** window.

```
=iif(Parameters!SalesPersonID.Value = -1, Fields!BusinessEntityID.
Value, Parameters!SalesPersonID.Value)
```

When the `SalesPersonID` parameter is `-1`, the report will show all the sales people. Otherwise, it will only show a single sales person based on the ID in `SalesPersonID`.



Deployment configurations

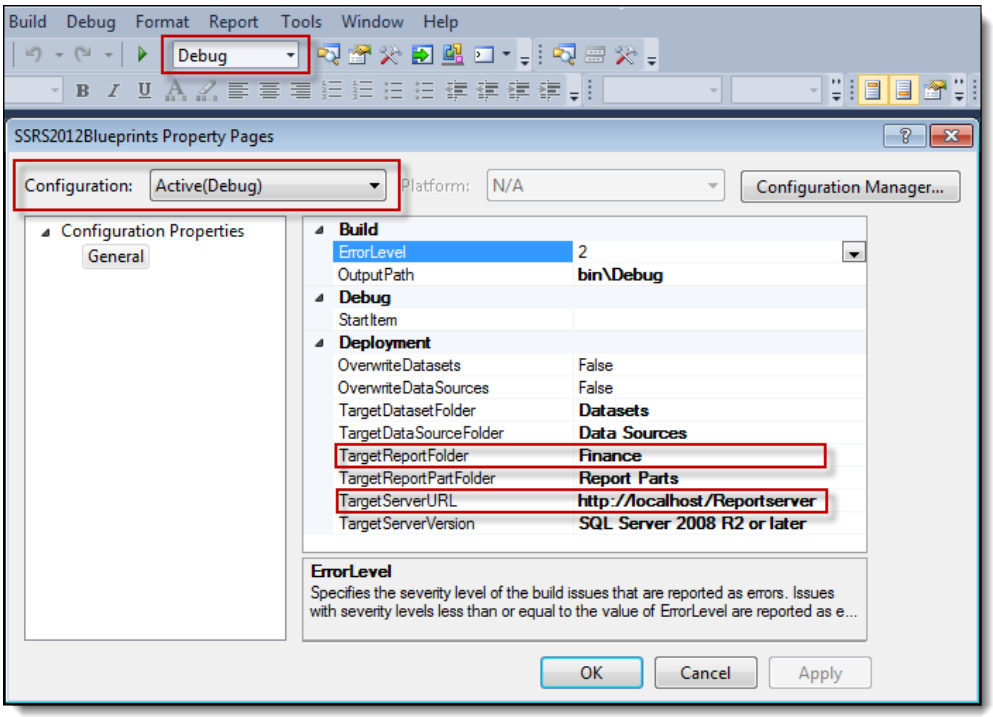
There are some settings that can be set in the project so that the reports, data sources, and shared datasets can be deployed from the project. Multiple configurations can be set up, so that a project can be deployed to multiple environments easily.

> While deploying directly from the project is easiest, I don't recommend this for deploying to production. It is too easy to accidently deploy an object to the wrong environment. I recommend deploying to production manually, which I'll show you later in the chapter.

Now that the sales commissions report is set up, let's set some properties to make deploying easier:

1. Right-click on the **Project** menu.
2. Select the **Properties** menu item.

> If SQL Server Data Tools had the focus on a report, then the menu item will include the name of the project.



The **Properties** page for the project includes settings for deploying the project. These settings are related to a particular configuration. A configuration is a group of settings that are used together. Multiple configurations can be set up to represent multiple deployment locations. The default configuration is `Debug` and is always shown in the standard toolbar.

You can choose to have the datasets and data sources always overwritten when the entire project is deployed. If you choose to not use the overwritten option, then they will only be deployed when they do not exist on the Report server.

> If you choose to have the datasets and/or data sources always overwritten, make sure that the project has the correct settings for the location you are deploying to, otherwise, they will be overwritten each time the project is published.

The following properties allow you to specify where on the Report server the various objects will be deployed to.

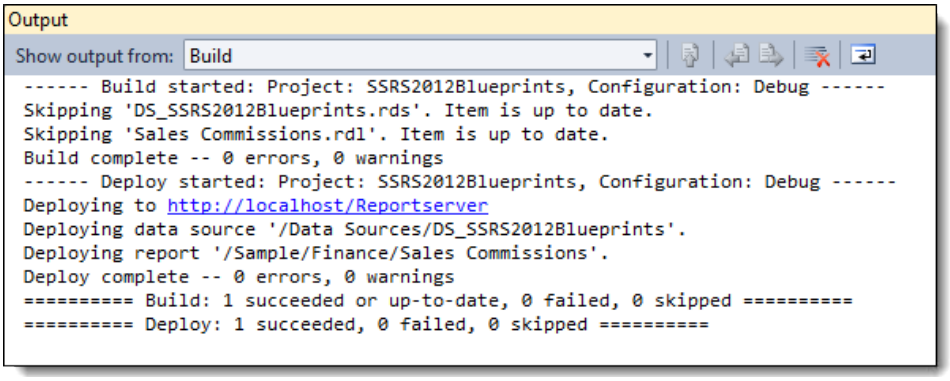
- `TargetDatasetFolder`
- `TargetDataSourceFolder`
- `TargetReportFolder`
- `TargetReportPartFolder`
- `TargetServerURL`
- `TargetServerVersion`

> Report parts are only used by the Report Builder web application.

Let's set some of the properties for the default configuration, then we can deploy our report. We will be deploying to a subfolder called `Finance` in a folder called `Sample` on our local Report server.

1. Set `TargetReportFolder` to `/Sample/Finance`.
2. Set `TargertServerURL` to `http://localhost/ReportServer`.
3. Click on the **OK** button.
4. Right-click on the project name, `SSRS2012Blueprints` in the **Solution Explorer**.
5. Select **Deploy** in the menu.

All the reports, data sources, and datasets in the project will be deployed to the Report server. If the data sources and datasets existed already in the specified folders, then they will not be redeployed. Also, all the reports will now be in a folder called `Finance`. The output window will give us a report of what was deployed and what was not deployed. Any errors will be noted in the output window as well.



Development to production migration

You can also deploy reports and datasets manually. Data sources need to be created manually. A manual deployment to production will protect your SQL Server Data Tools environment from accidently deploying to the production environment.

Let's follow John as he trains someone on the operations team on how to deploy the sales commission report and the shared data source to production.

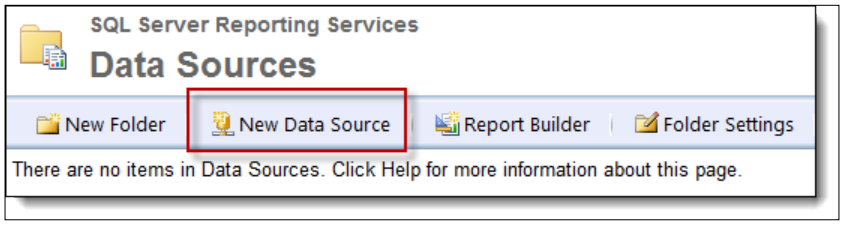
Let's create a new folder for our manual deployment, called `Finance`. We will use this folder for the rest of the chapter.

1. Open up the **Report Manager**, `http://localhost/Reports`.
2. Click on the toolbar item called **New Folder**.
3. Name the folder `Finance`.
4. Click on the **OK** button.
5. Open the new folder you just created.
6. Click on the toolbar item called **New Folder**.
7. Name the folder `Data Sources`.
8. Click on the **OK** button.

We are going to manually create a data source using specific login credentials. The process is similar to what was discussed in *Chapter 1, Let's Breakdown the Numbers*. We can't use Window's Integrated Security due to the fact that we will be creating snapshots and subscriptions with this data source. Windows Integrated Security requires someone to be physically logged in. The login account will need to have permissions to execute all the stored procedures that are used by the reports. The login can be a Windows account or a SQL login.

> In a production environment, a generic login, created specifically for reporting, is a better approach than using a person's login credentials. That way when the person leaves the company, all of the reports will still work.

1. Open the `Data Sources` folder that we just created.
2. Click on the toolbar item called **New Data Source**:



3. Set the **Name** field to `DS_SSRS2012Blueprints`.
4. Set the **Data source type** property to `Microsoft SQL Server`.
5. Set the **Connection String** property to `Data Source=(local);initial Catalog=AdventureWorks2012`.
6. Click on the **Credentials stored securely in the report server** option button.
7. Set the username to `John Kirkland`. Also set his password.
8. Click on the **Test Connection** button to make sure the connection works.
9. Click on the **OK** button to save your changes.

> If John Kirkland is a Windows account, then also check the **Use as Windows credentials when connecting to the data source** checkbox.

Let's manually upload the sales commissions report to the new folder we created:

1. Open the folder named `Finance`.
2. Click on the toolbar item called **Upload File**.
3. Click on the **Browse** button.
4. Find the location of the `Sales Commissions.rdl` file that we modified.
5. Click on the **OK** button.

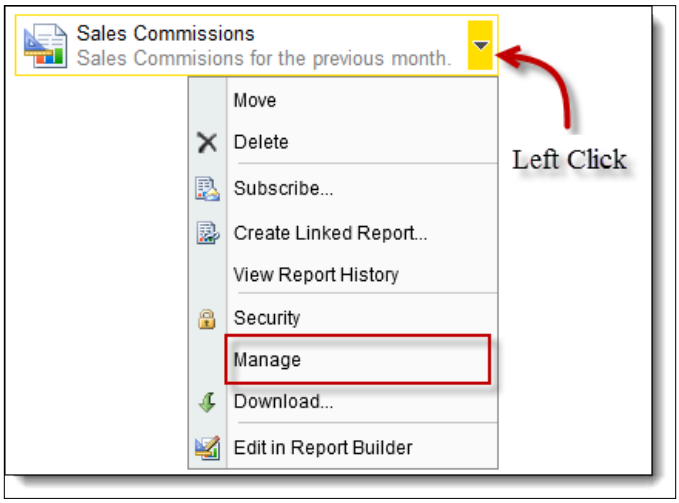
If you are overwriting an existing instance of a report file, then it is important to select the **Overwrite item** checkbox. This will reset the parameters to what is contained in the report file. If the **Overwrite item** checkbox is not selected, then the report file will be placed with the new report file, but the default parameter values will not be updated.

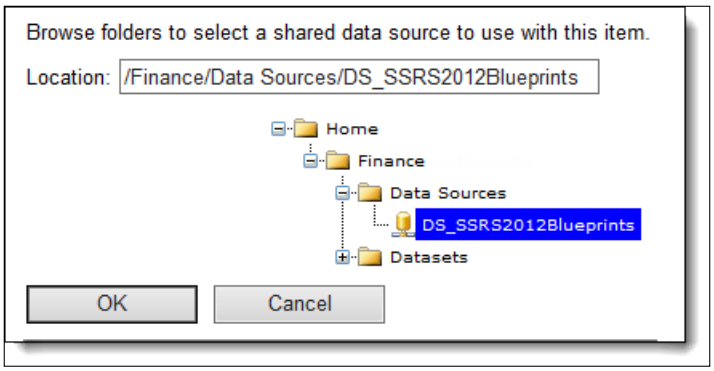
We now have the report on the server, but it will not work yet. This is due to the report not knowing where we created the shared data source. Shared datasets are reattached the same way. Let's fix the data source location:

1. Hover to the right-hand side of the report name until an upside-down arrow appears, then click on the upside-down arrow.
2. Select the **Manage** menu item.
3. Select the **Data Sources** menu item on the left-hand side.
4. Click on the **Browse** button.
5. Click on the tree item named **Finance**.
6. Click on the tree item named **Data Sources**.
7. Select the data source we created.
8. Click on the **OK** button.

9. Click on the **Apply** button.

It's very important to click on the **Apply** button. If you skip that step, the data source location will not be saved even though you clicked on the **OK** button in the prior screen.

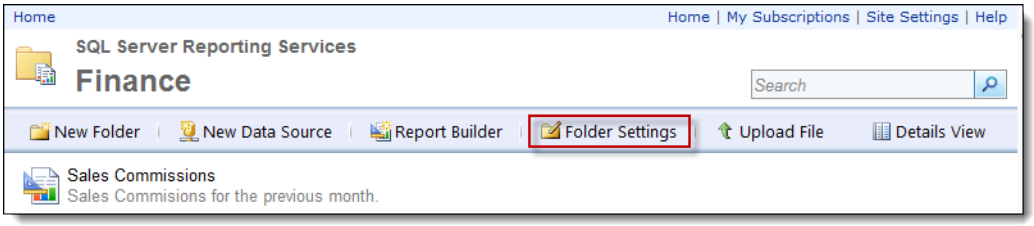




User security

Before we can let Linda use the report, we need to give her permission to run the report. Security can be set at any level, including the report itself. The security will be inherited from the previous level unless it is overwritten. There are various levels of security that can be given to an individual or a group.

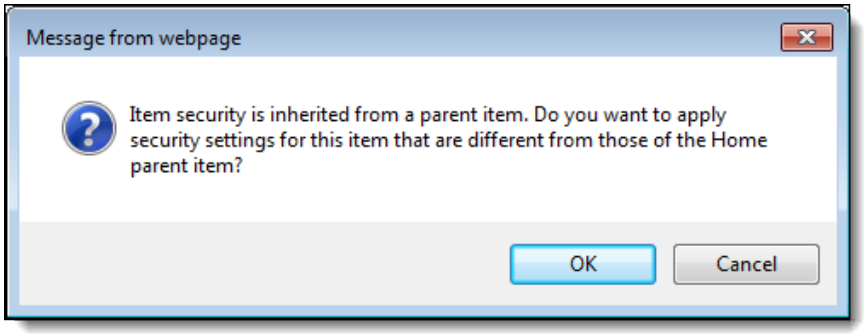
Here's how we'll set Linda's security; we are going to give her permission to access the `Finance` folder only, and all the reports within.



To do so, perform the following steps:

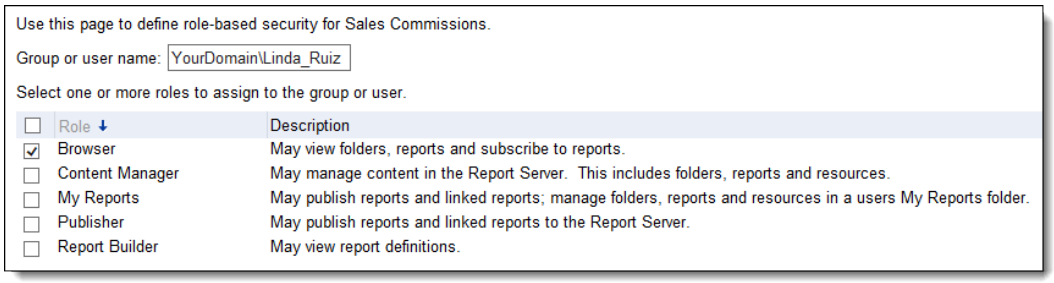
1. Open up the `Finance` folder.
2. Select the toolbar item called **Folder Settings**.
3. Select the **Security** menu item on the left-hand side.
4. Click on the toolbar item called **Edit Item Security**.

You'll be shown a message box asking if this is what you really want to do. It is asking if you want the security for this folder to be different from the security of the parent folder. You can revert to the parent folder security at any time. To do so, perform the following steps:



1. Click on the **OK** button.
2. Click on the toolbar item called **New Role Assignment**.
3. Set the **Group or user name** property to `YourDomain\Linda_Diaz`. Replace `YourDomain` with the `domain` that you are working in.
4. Select the **Browser** checkbox. This will allow her to see the folder and reports, but she will not be able to modify anything.

5. Click on the **OK** button.

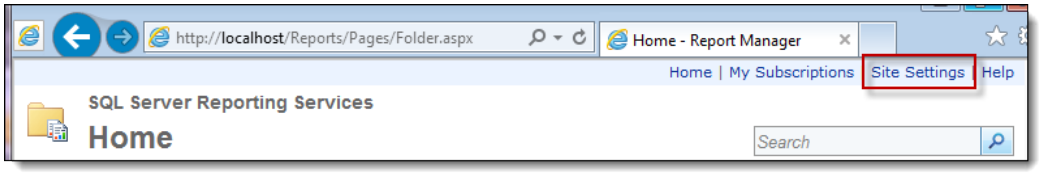


Linda can now use the link `http://localhost/Reports/Pages/Folder.aspx?ItemPath=%2fFinance&ViewMode=List` to access the `Finance` report folder.

Shared schedules

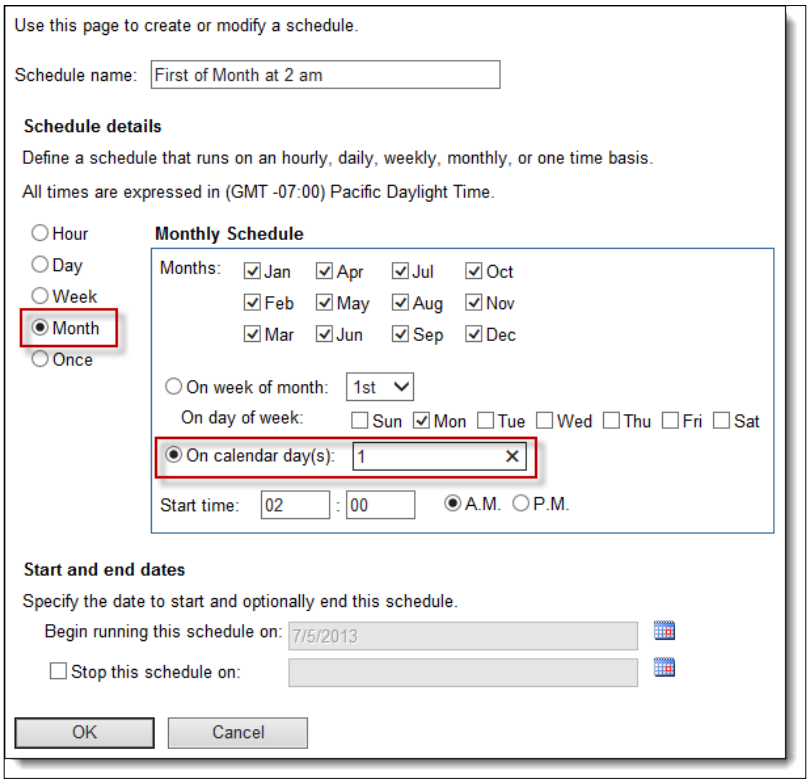
The subscriptions and snapshots that Linda has requested need to be created on the first of the month. Report Manager allows us to create shared schedules. These are predefined time frames that can be referenced at various places. If the schedule needs to be modified, then there is only place that the schedule needs to be changed. All the items that are using that shared schedule will be updated to the new time frame.

Let's set up a shared schedule for the subscriptions and snapshots to use. It will be for the first of the month and will run at 2 am.



1. Click on **Site Settings** at the right-hand side at the top of the screen.
2. Click on the **Schedules** menu item on the left-hand side.
3. Click on the toolbar item called **New Schedule**.
4. Set the **Schedule name** property to `First of Month at 2 am`.
5. Select the **Month** option button.
6. Select the **On calendar day(s)** option button and set the value to **1**.

7. Click on the **OK** button.



There are many different combinations that can be created in this screen. Schedules can even be created to start on a particular date.

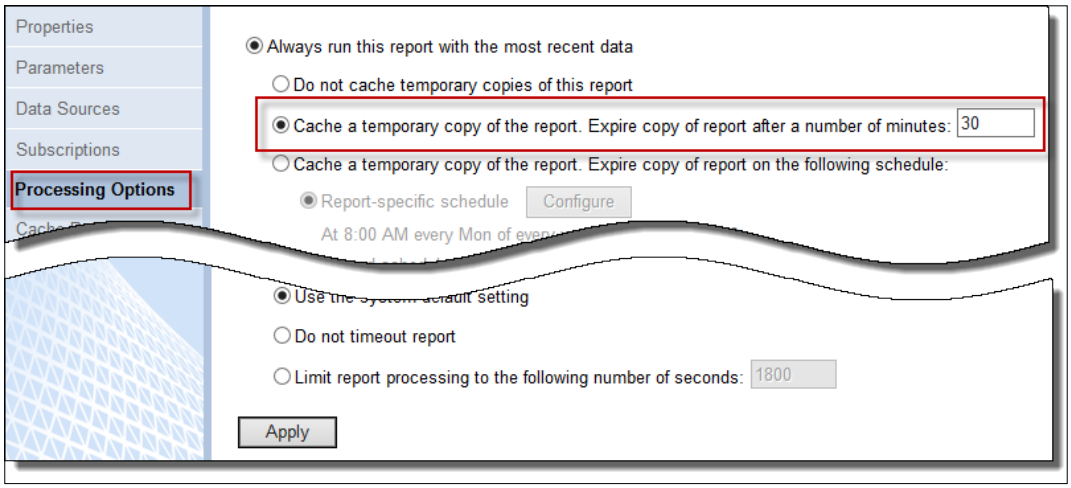
> The option called **Once** is very handy for unit testing schedules since it will only execute once.

Cached reports

Let's set up caching for the sales commission report. What caching does, it allows Report Manager to run the SQL query one time and store the data for subsequent use. The length of time the data is stored is based on the length of time set in the properties. Parameter changes will also affect the length of time the query remains cached. This is why we made the `SalesPersonID` a report parameter and not a parameter in the stored procedure.

Let's set the cache properties so that the data will be available for half an hour after the data is requested. This will provide enough time for all the reports to be generated for our data driven subscription that we will set up later. First, we need to specify that we want to use caching:

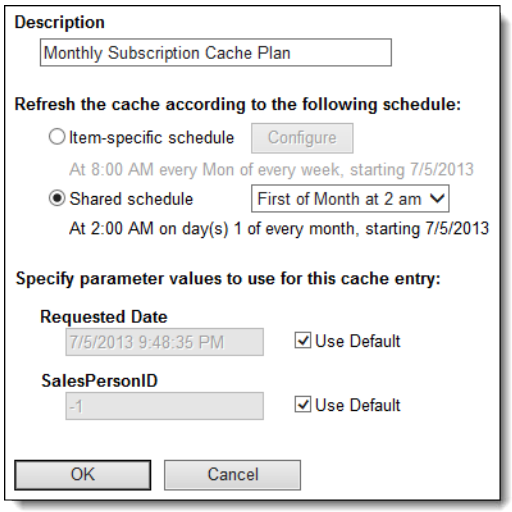
1. Hover to the right of the report name until an upside-down arrow appears, then click on the upside-down arrow.
2. Select the **Manage** menu item.
3. Click on the **Processing Options** menu item on the left-hand side.
4. Select the second option button labeled: **Cache a temporary copy of the report. Expire copy of report after a number of minutes**.
5. We will use the default value of `30`.
6. Click on the **Apply** button at the bottom of the screen:



Now, we can set up a cache plan for the first of each month:

1. Click on the **Cache Refresh Options** menu item on the left.
2. Click on the toolbar item called **new Cache Refresh Plan**.
3. Set the **Description** property to `Monthly Subscription Cache Plan`.
4. Select the **Shared schedule** option button.
5. Select the shared schedule we created, called **First of Month at 2 am**.
6. Select the **Use Default** checkboxes for both parameters.

7. Click on the **OK** button:



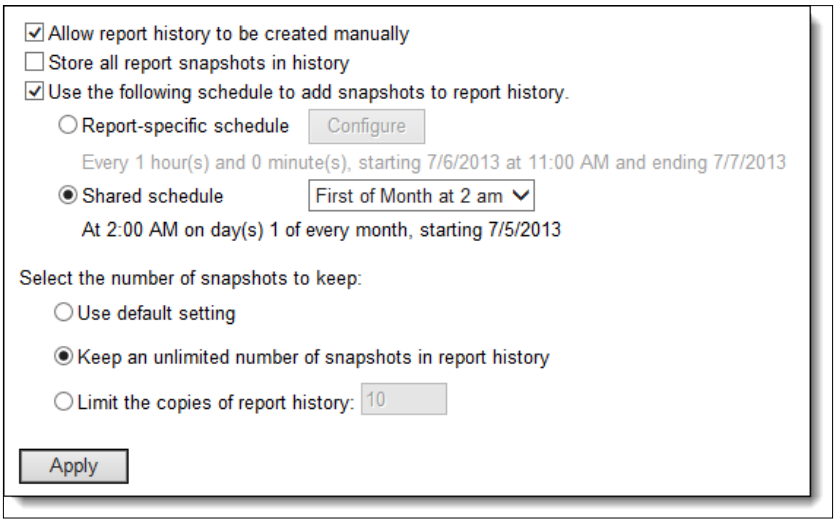
Snapshots

A snapshot saves the report on a predefined schedule so that the report can be looked at again in the future with the same data as when the report was run. The data won't change for these reports. This is very useful for many reasons, including having an audit trail and running expensive reports only once for a given period.

Let's set up a monthly snapshot for Linda:

1. Click on the **Snapshot Options** menu item on the left.
2. Select the third checkbox called: **Use the following schedule to add snapshots to report history**.
3. Select the **Shared schedule** option button.
4. Select the shared schedule we created, called **First of Month at 2 am**.
5. Select the second option button called: **Keep an unlimited number of snapshots in report history**.

6. Click on the **Apply** button:



Linda will now be able to go to the `Report History` folder and see the past snapshots.

Subscriptions – e-mail delivery / scheduled delivery

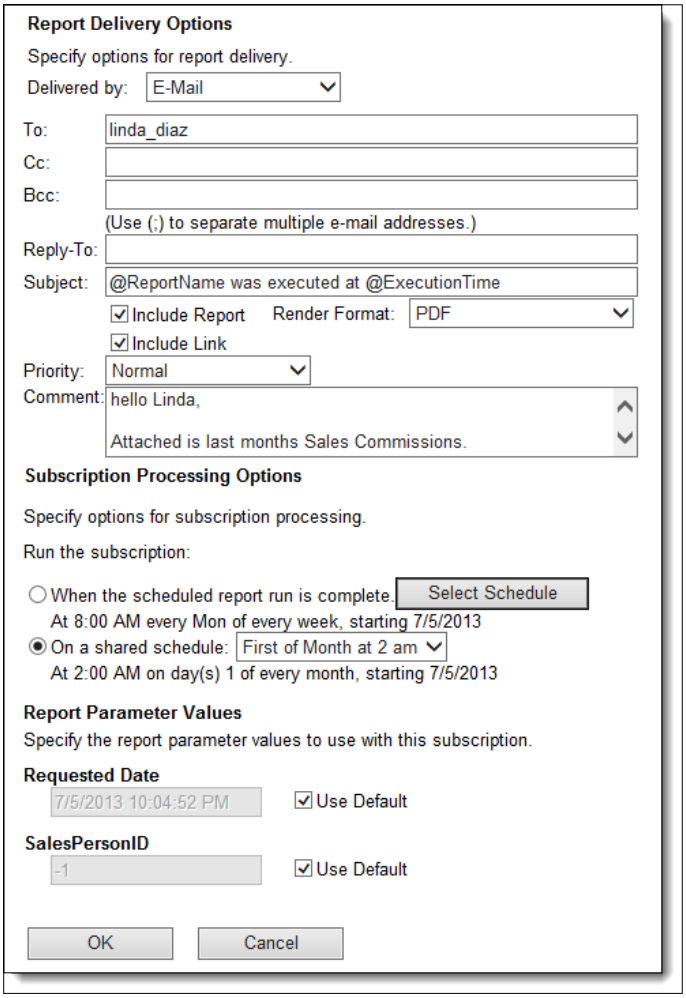
Report Manager provides the option to have reports automatically generated and delivered based on a schedule. The report can be delivered via an e-mail or file sharing. The file sharing method allows the report to be saved in a folder that can be accessed by others when they choose.

Let's set up a subscription for the sales commissions report so that it can be automatically e-mailed to Linda on the first of the month.

> Make sure that your Network Administrator or DBA has set up e-mail in the Reporting Services Configuration Manager so that the subscription will work with e-mail.

1. Click on the **Subscription** menu item on the left-hand side.
2. Click on the toolbar item called **New Subscription**.
3. Set the **To** property to `linda_diaz` (or your own e-mail so that you will get the e-mail).

4. Select **PDF** for the **Render Format** drop-down list.
5. Add a comment to the **Comment** property.
6. Select **On a shared schedule** option button.
7. Select the shared schedule we created, called **First of Month at 2 am**.
8. Select the **Use Default** checkbox to the left of the **Requested Date** and **SalesPersonID** parameters.
9. Click on the **OK** button.



Subscriptions – data-driven subscription

Another type of subscription is the data-driven subscription. This is a powerful way of creating subscriptions since you can use the database to set most of the properties of the subscription. The data-driven subscription is generated through a wizard. There are three different methods of delivery: e-mail, file share, and null. The null delivery does not generate a report. It is used to preload the cache for a report that is set up for using cached data.

Let's create a subscription to be sent out on the first of each month for each of the sales people:

Step 1 – specify the type of data-driven subscription

1. Select the toolbar item called **New Data-driven Subscription**.
2. Set the **Description** property to `Last Month's Commissions for Each Sales Person`.
3. Set the **Specify how recipients are notified** property to `E-mail`.
4. Select the **Specify a shared data source** option button.
5. Click on the **Next** button.

Step 2 – select the location of the shared data source

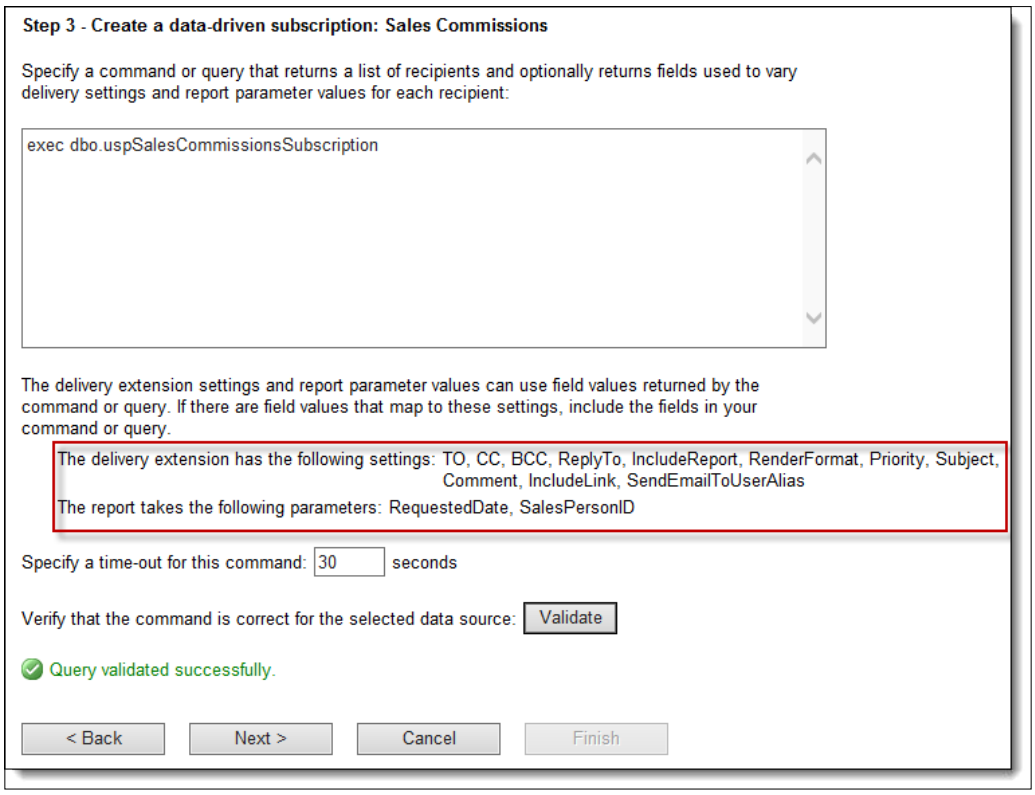
1. Set the **Location** property to `/Data Source/DS_SSRS2012Blueprints`.
2. Click on the **Next** button.

Step 3 – set up the SQL statement or stored procedure that will return the data-driven property data.

1. Set the query to the following code: `exec dbo.uspSalesCommissionsSubscription`.
2. Click on the **Validate** button to verify if the query works.

3. Click on the **Next** button.

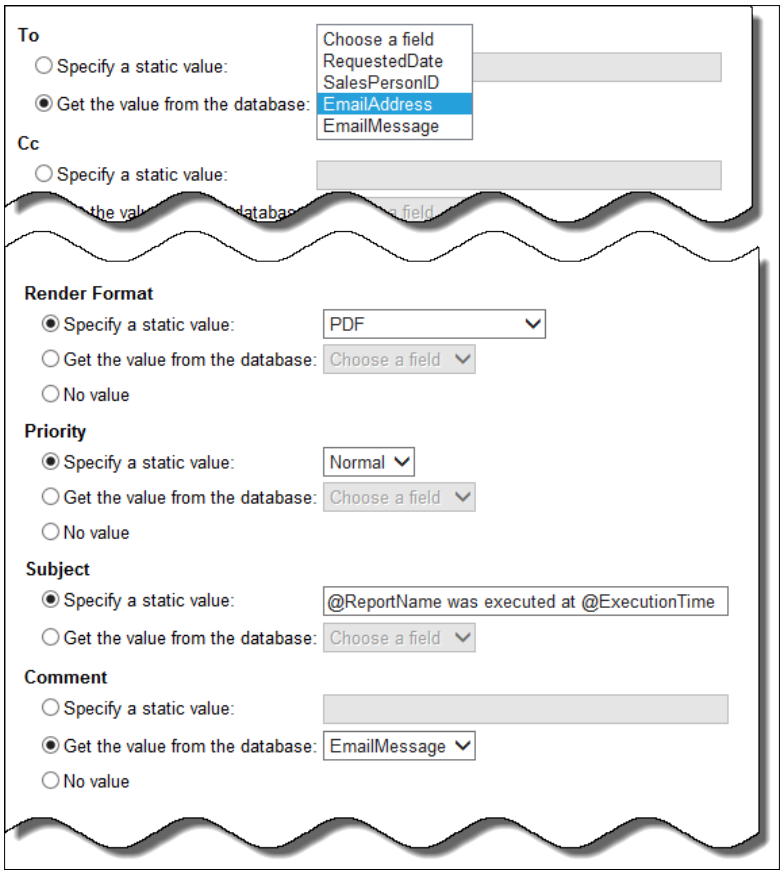
This page has some handy information. It tells you exactly what fields you can have the query return for the data-driven subscription.



Step 4 – set up the delivery properties

1. Select the **Get the value from the database** option button for the **To** property.
2. Select **EmailAddress** from the drop-down list.
3. Select **PDF** from the drop-down list for the **Render Format** property.
4. Select the **Get the value from the database** option button for the **Comment** property.
5. Select **EmailMessage** from the drop-down list.

6. Click on the **Next** button:

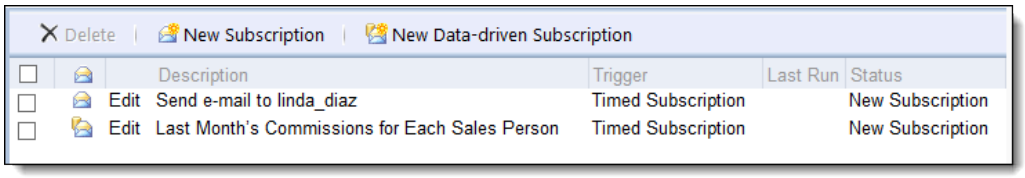


Step 5 – set up the parameter values

1. Select the **Get the value from the database** option button for the **Requested Date** property.
2. Select **RequestedDate** from the drop-down list.
3. Select the **Get the value from the database** option button for the **SalesPersonID** property.
4. Select **SalesPersonID** from the drop-down list.
5. Click on the **Next** button.

Step 6 – set up the schedule for the subscription

1. Select the **On a shared schedule** option.
2. Select **First of the Month as 2 am** from the drop-down list.
3. Click on the **Finish** button.



Subscriptions should be tested just like reports to make sure the parameters work correctly, and that the e-mail settings were accurately created. One way to test is to have the report sent to you a few minutes into the future. This will allow you to test the subscription without having to wait days or even months.

Summary

In this chapter, we learned two different ways to deploy to the Report Manager. One way was to use SQL Server Data Tools and the other way was to upload the files manually. We also learned how to cache report data and create snapshots of reports on a schedule. Finally, we learned how to deliver reports through two different kinds of subscriptions.

8
Beyond the Horizon!

We have come to the last leg of John Kirkland's journey to learning SQL Server Reporting Services 2012. One thing he loves about this journey is that he didn't only gain a significant amount of technical skills but also gained an advanced understanding of their company's business in general.

And you, the reader, can very well identify with John's experience all throughout the book as we went from developing simple reports to the advanced ones. John is you. And you should congratulate yourself for a job well done.

Being at the center of any technical project that involves data analysis and reporting always imparts the insider knowledge about the business that is often accessible only to the management.

Thus a report developer, or any technical developer for that matter, holds an important role in any organization. People like you have become linchpins to many companies across multiple disciplines and industries.

As business requirements increase in complexity, Microsoft's offerings on their premier reporting platform will also advance to adapt to accommodate the many challenges of corporate reporting. More and more advanced tools and features are being added into the platform with every iteration.

This chapter provides basic information on **SharePoint**, **PowerPivot**, and **Power View**.

We'll lightly tackle the following:

- SharePoint Integration
- Power View SharePoint add-in
- PowerPivot Excel add-in

Integrating SSRS with SharePoint

SharePoint integration with SQL Server Reporting Services was introduced in SQL Server 2005 SP2 (Windows SharePoint Services 3.0 or Microsoft Office 2007 SharePoint Server). One of the obvious benefits of this integration is the convenience of access to the reports.

Some companies have a SharePoint site as their main intranet for internal communication. It's a good idea to integrate reporting with the SharePoint site as a way of consolidating different information in a one-stop site.

If you are interested in the SQL Server Reporting Services 2012 and SharePoint endpoint architecture, read the MSDN blog titled *SQL Server Reporting Services 2012 SharePoint integrated mode Endpoints architecture*. Here's the shortened link to the article: `http://bit.ly/SQLSP`.

> In SQL Server 2005 and 2008, the Reporting Services Configuration Manager is used for configuring the SharePoint integration. That changes in SQL Server 2012. The SharePoint mode configuration is now handled in SharePoint Central Administration.

Let's integrate SQL Server 2012 and SharePoint 2010 or 2013 (SharePoint 2013 is used in this setup).

Before we proceed, please make sure that you've already done the following:

1. Install SQL Server 2012 in SharePoint Mode. Here's a quick step-by-step instruction on how to *Install Reporting Services SharePoint Mode as a Single Server Farm*: `http://bit.ly/SQL2012SPMode`.
2. Install SharePoint 2010 or 2013. SharePoint 2013 is used in this demonstration. The SharePoint and SQL Server are installed on separated boxes in my setup.
3. If you are installing SharePoint in an offline Windows 2012 server (not connected to the Internet), read and follow the instructions in *SharePoint 2013: Install Prerequisites Offline or Manually on Windows Server 2012 - A Comprehensive Guide*, which is found at `http://bit.ly/SP2013PreReqfiles`.
4. Install the Reporting Services add-in for SharePoint 2010 or 2013. The appropriate add-in for your setup can be found at `http://bit.ly/SPAddIn`.

Please note that SQL Server 2012 Reporting Services, Reporting Services add-in, and SharePoint follow a strict rule on component combination.

Please refer to the following table for your correct combination:

Report Server	Add-in	SharePoint Version
SQL 2012 SP1	SQL 2012 SP1	SharePoint 2013
SQL 2012 SP1	SQL 2012 SP1	SharePoint 2010
SQL 2012	SQL 2012	SharePoint 2010

For a complete combination list on all other SQL Server and SharePoint versions, please check the *Supported Combinations of SharePoint and Reporting Services Components* section. Here's the link: `http://bit.ly/SupportedCombi`.

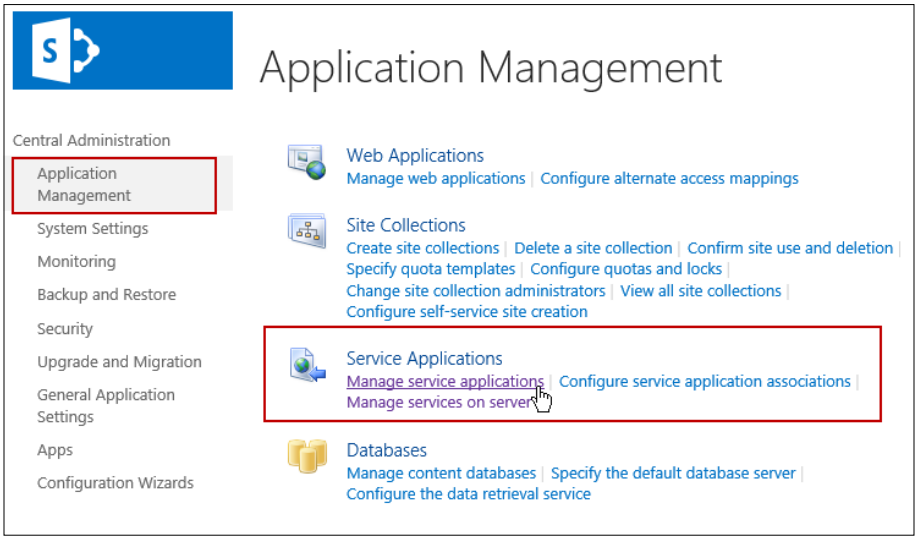
Once you've completed the setup, you're now ready to configure the integration between the components.

Creating Reporting Service Application

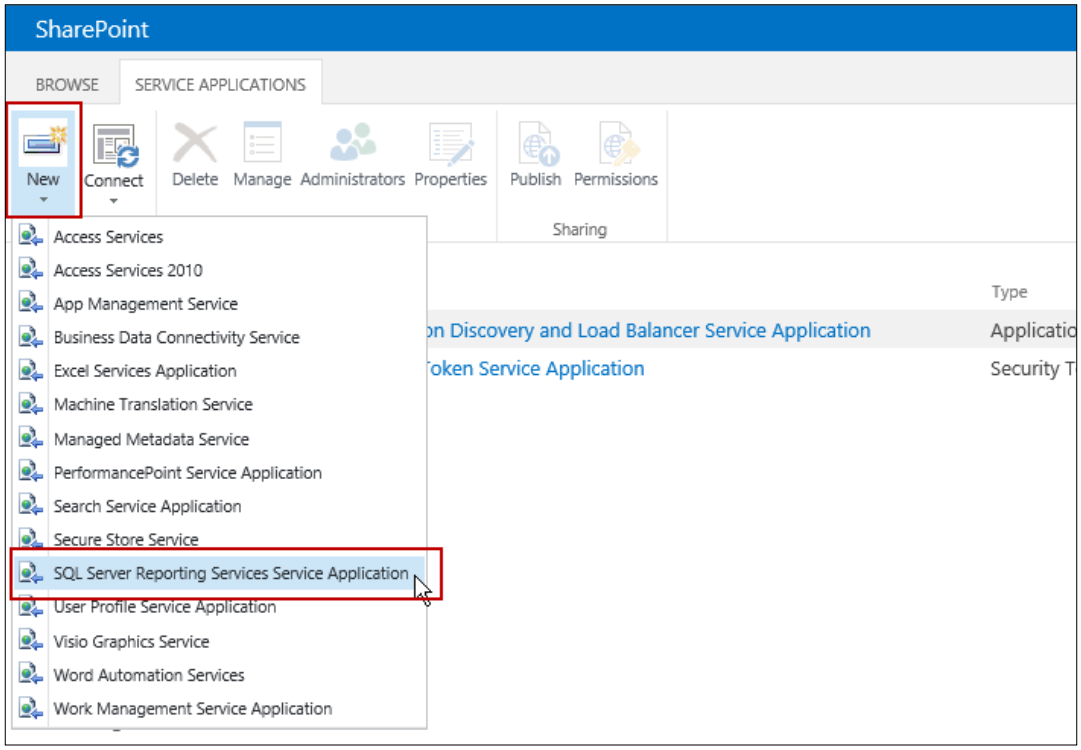
After the set up for SQL Server Reporting Services and SharePoint are complete, the next step is to create a SQL Server Reporting Services Service Application.

Let's create the application:

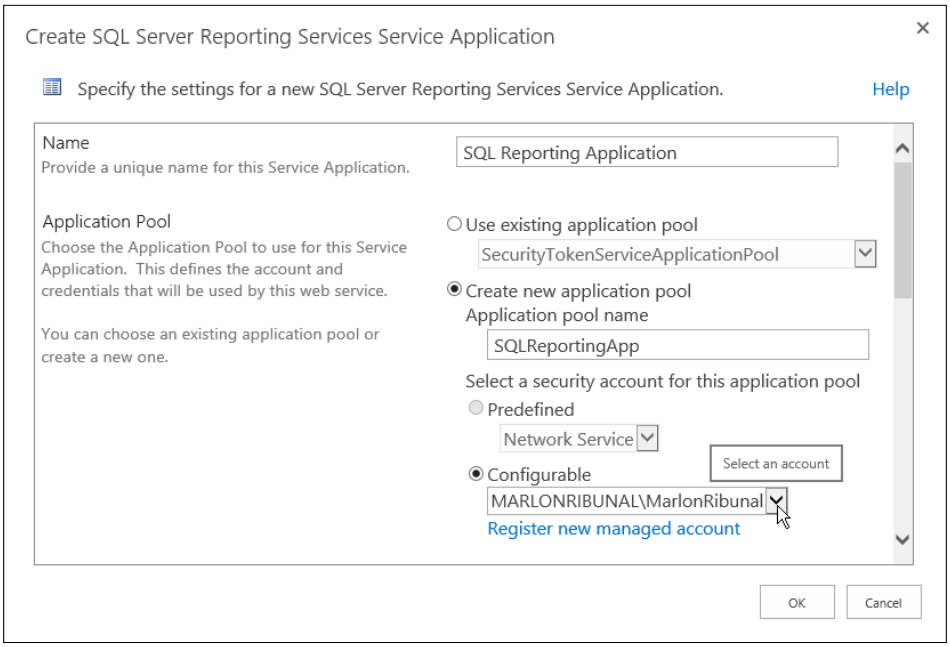
1. Open SharePoint 2013 **Central Administration**.
2. Select the **Application Management** tab, and select **Manage service applications** under the **Service Applications** heading as shown in the following screenshot:



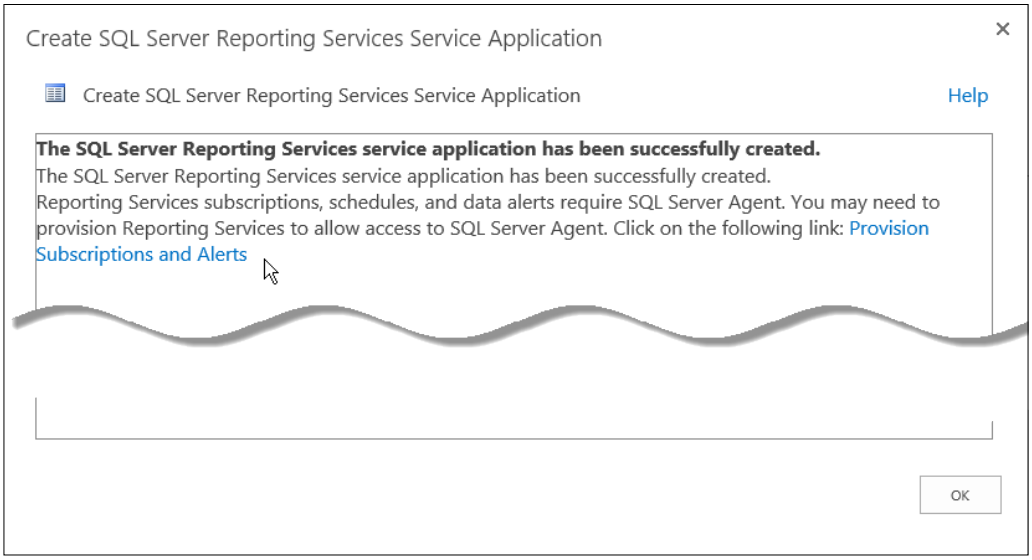
3. Click on **New** on the ribbon panel of **Service Applications** tab, and select **SQL Server Reporting Services Service Application** as shown in the following screenshot:



4. Specify the settings for a new **SQL Server Reporting Services Service Application**. Type the appropriate information in the boxes. Click on **OK** when done. Make sure you put correct and appropriate information in the **Application Pool** setting. Once you click on **OK**, the setup cannot be cancelled.



5. You'll get a message that confirms the successful creation of the service application. A link to provision subscriptions and alerts is also provided in the message:



Provisioning subscriptions and alerts

Another useful feature of the Reporting Services and SharePoint integration is the ability of the system to handle subscriptions and alerts. This is not really new, because, we have the same features on the SSRS Report Manager. But this is handy since you don't have to configure things like this somewhere else.

Let's configure the SQL Server Agent:

1. Click on the **Provision Subscription and Alerts** link from the message we got when we created the service application.
2. That displays the Provision **Subscription and Alerts – SQL Reporting Application** dialog box.
3. Click on **Download Script** to download the SQL Reporting Application, `GrantRights.sql` file from your SharePoint server (SP2013 in my case). Save the file in an accessible location.
4. Log in to your SQL Server 2012 Server, and enable the SQL Server Agent. Execute the script downloaded from the SharePoint server.

Let's set this aside for now. We may not cover the use of subscriptions and alerts in this chapter, but I included this step-by-step nonetheless as a reference.

Creating a web application

Now that we have created the SQL Reporting Services Service and have it running on our SharePoint 2013 Server, we're ready to create our website that will host our reports. All these setup steps might be confusing, and they really are if you're trying to do them at once in one sitting.

To put it simply, the SharePoint is our platform. The application service that we created, SQL Reporting Application, is the infrastructure upon which we are going to run our web application/website.

Let's create our website:

1. Open SharePoint 2013 **Central Administration**. Select the **Application Management** tab. Select **Manage web applications** under the **Web Applications** heading.
2. Click on **New** on the ribbon panel on the **Web Application** tab.
3. That displays the **Create New Web Application** dialog box. Select **Create a new IIS web site**.

4. Name the new website **SQL Reporting Website**.
5. Use the default port `80`.
6. Leave the **Security Configuration**, **Claims Authentication Types**, **Sign In Page**, and **Public URL** settings with their default values.
7. On **Application Pool Name**, type `SQL Reporting Website Pool`.
8. On the **Database Name and Authentication**, type your SQL Server name, in my case `SQL2012-02`. I leave the default database name, WSS_Content. This database will be created by the setup.
9. Select the recommended database authentication, which is **Windows authentication**.
10. On **Service Application Connections**, select **SQL Reporting Application**. That might already be selected by default if that's the only service application on the list. Click on **OK** when done.
11. If the setup does not encounter any error, you'll receive a confirmation message that the web application has been created. Don't click on **OK** on the message just yet.

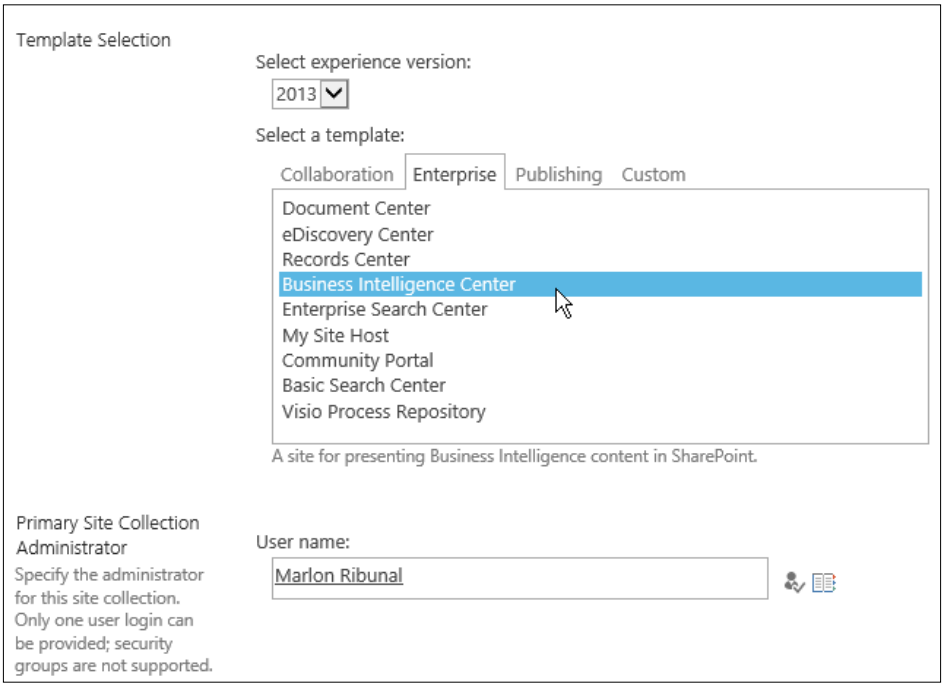
Creating a site collection

The web application that we just created serves as the frame or scaffolding of our site collection. It is only the bare bones. Now, we need to give it some flesh so it will take some form and functionalities. To do that, we need to create a site collection.

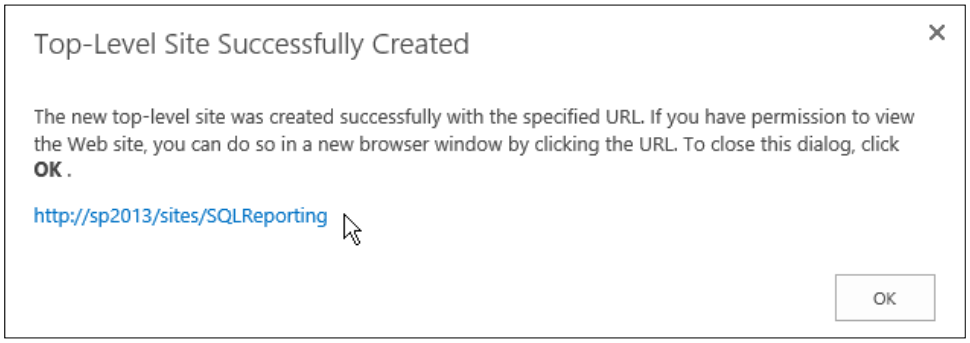
Perform the following steps for creating site collection:

1. Before you click on the **OK** button on the confirmation message after creating the web application, you might have noticed the link that says **Create Site Collection**. Click on that link.
2. That displays the **Create Site Collection** dialog box. Type **SQL Reporting Collection** on the **Title** field.
3. On **Web Site Address**, select **/sites/** for the site prefix. And type **SQLReporting** on the **Site Name** field.

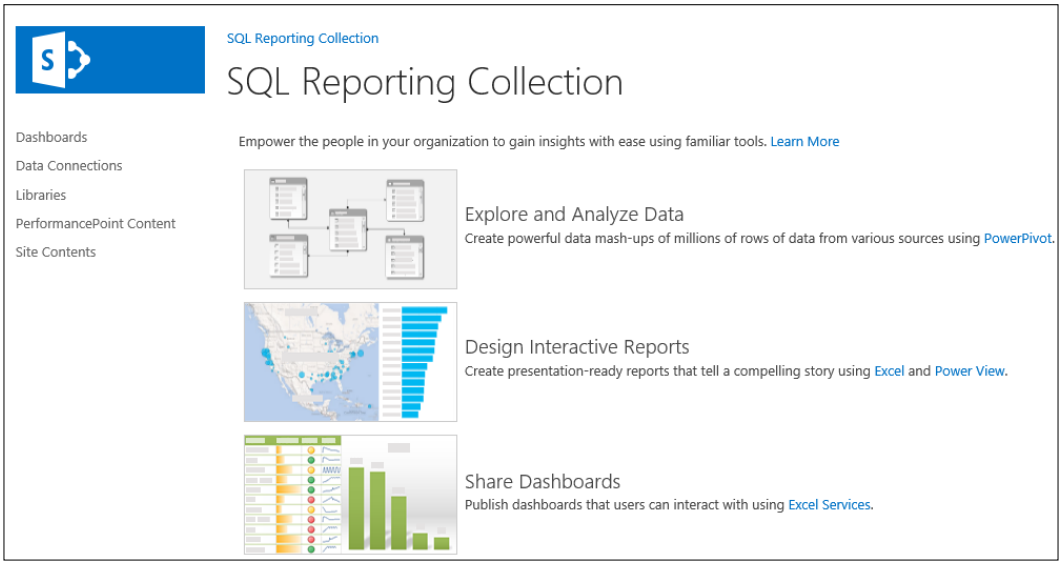
4. On the **Template Selection** window, go to **Enterprise | Business Intelligence Center**. This gives us the components we need for our reporting purposes.



5. On the **Primary Site Collection Administrator** and **Secondary Site Collection Administrator** fields, provide the usernames that will have administrator privileges to the site (domain account). Click on **OK** when done.
6. You should get a confirmation message if the site is successfully created. Take note of the URL of your new site.



7. Go to the new site by clicking on the link from the confirmation message or type it directly in your URL bar. This is what the new website looks like using the `Business Intelligence Center` template:



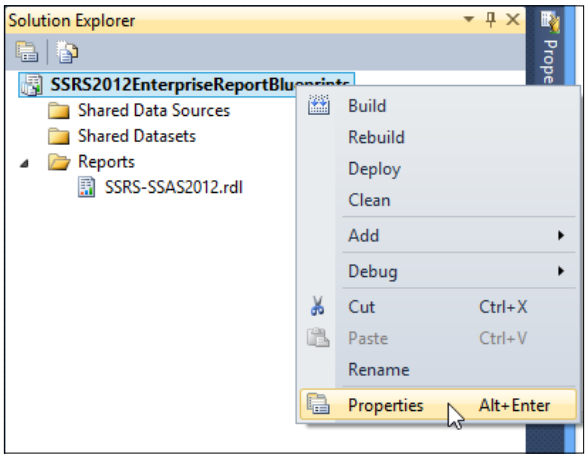
Now that your site is set up, you're ready to deploy SSRS projects. Look for SharePoint 2010/2013 resources on how to manage and customize your site. Unfortunately, we do not cover that topic in this book.

Deploying SSRS projects to SharePoint

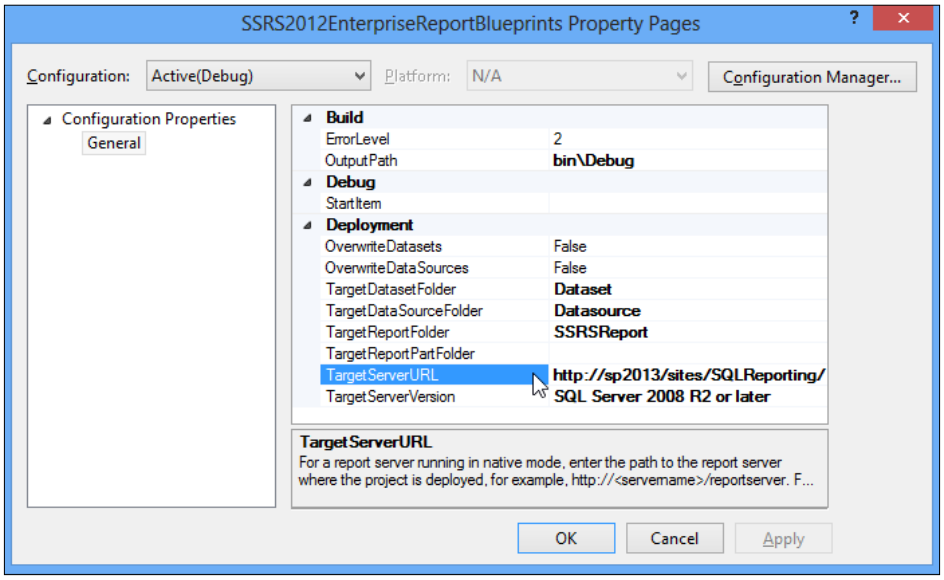
The preferred tool in report development is the SQL Server Data Tools (SSDT), which we've been using in this book. If you setup the SharePoint 2010/2013 and SSRS 2012 in SharePoint mode correctly, deploying your SSRS projects to your SharePoint site is a breeze.

Let's configure SSDT for deployment to SharePoint:

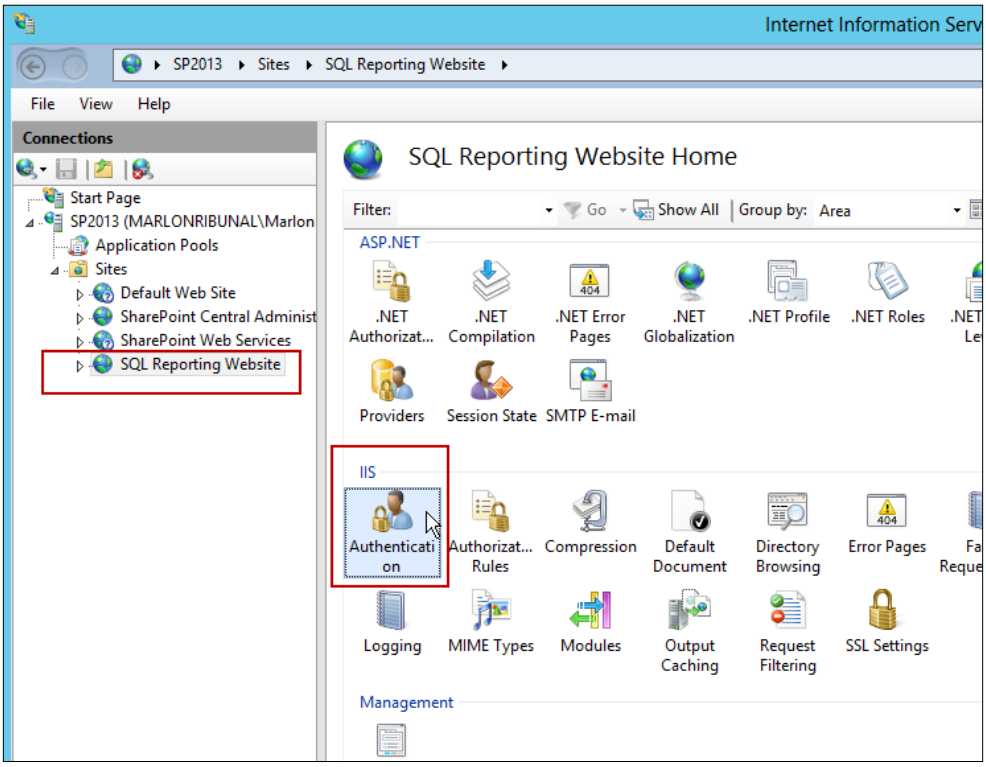
1. Open your SSRS project in SSDT. On the **Solution Explorer** window, right-click on your project and select **Properties** from the context menu as shown in the following screenshot:



2. That displays the project **Property Pages** dialog box. Under the **Deployment** section, enter `http://<servername>/<site>/<subsite>` in the `TargetServerURL` field. In this case, our subsite is our SQL Reporting Collection. So, our target server would be: `http://sp2013/sites/SQLReporting`.



3. If you encounter authentication issues, review your installation and make sure that you have the correct access setup to both of the SharePoint and SQL Server. Review the installation process at `http://bit.ly/SPSSRSInstall`.
4. Another common cause of the authentication error has something to do with the anonymous authentication being enabled in your site. Check with your web administrator before doing this. To disable the anonymous authentication, go to your SharePoint server. Press the Windows key + *R* on your keyboard, and then type `inetmgr`.
5. That displays the **Internet Information Services (IIS) Manager** dialog box. Expand your server on the tree, and then expand the **Sites** tree. Select your website, in our case **SQL Reporting Website**. In the **IIS** section, double-click on **Authentication**.



6. Right-click on **Anonymous Authentication** and select **Disable**.

Getting introduced to Power View

Power View is one of the new tools in Microsoft Business Intelligence's offering. It is a simple and yet powerful presentation tool that will get your reports to the next level. Power View might just be the biggest feature that came with SQL Server Reporting services 2012.

Authoring highly visualized reports has never been easier before this browser-based interactive tool. Power View is an integrated tool that you can use with SSRS 2012 and SharePoint 2010 and 2013.

Power View is actually a feature of Microsoft Excel 2013. As far as SQL Server 2012 Reporting Services is concerned, Power View is an SSRS 2012 add-in for SharePoint 2010 and 2013 Enterprise editions.

Technically, we have two versions of Power View:

- Power View for Excel 2013
- Power View for SharePoint 2010/2013

You can, therefore, create Power View reports in two ways: through Excel 2013 and SharePoint 2010/2013 with the SQL Server 2012 Reporting Services Power View add-in.

Get yourself familiarized with the two BI Semantic Models: Multidimensional BI Semantic Model and Tabular BI Semantic Model. Power View manages these two models in different ways, and both are hosted by SQL Server Analysis Services.

You're probably familiar with the concepts of the two models already. Multidimensional models are also known as cubes. The tabular, based on table, model is more of a spreadsheet in format. Those are not really complete descriptions, but you get the idea.

Before you can use Power View with SharePoint 2010 and 2013 Enterprise Edition and SQL Server 2012 Reporting Services, you must have the following components already installed (see previous section for related links):

- SharePoint 2010/2013 Farm
- SQL Server 2012 SP1
- SQL Server Reporting Services (SharePoint Mode)
- SQL Server 2012 PowerPivot for SharePoint (found in SQL Server 2012 setup)

Understanding PowerPivot

PowerPivot is a free extension for Microsoft Excel 2010 and 2013. PowerPivot is a steroid for Excel. It enables Excel to process a massive amount of data.

Excel with PowerPivot enabled can hold millions of rows in a single sheet and that doesn't have a substantial impact on performance. With PowerPivot, Excel becomes a legit Business Intelligence tool.

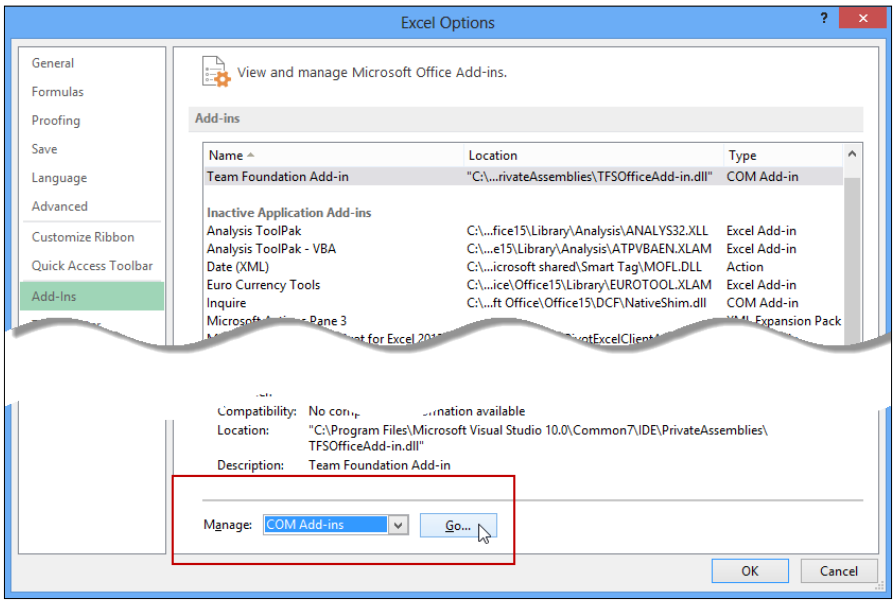
Power View and PowerPivot are closely intertwined. If you want to create a Power View report in SharePoint, you have to create the model first in Excel. This model is created with PowerPivot.

Adding a data source to a model

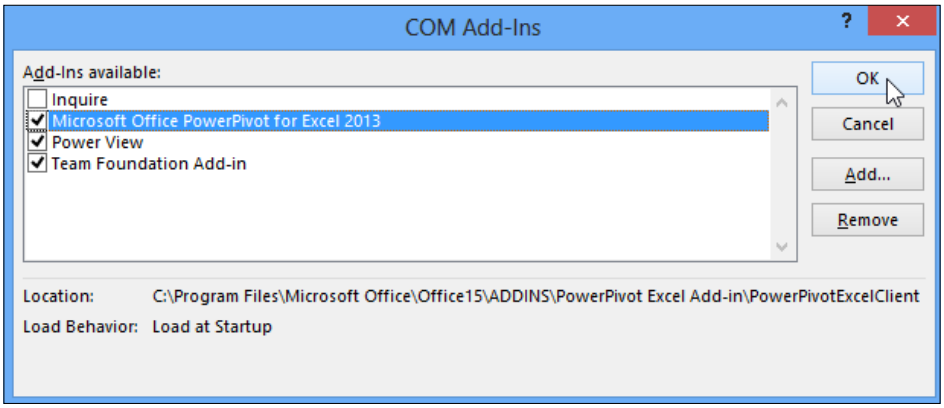
A data model is the basis for any Power View and PowerPivot.

Let's create that. In this demonstration, I'm using Excel 2013. The difference in Excel 2010 may not be significant. You may want to double check that.

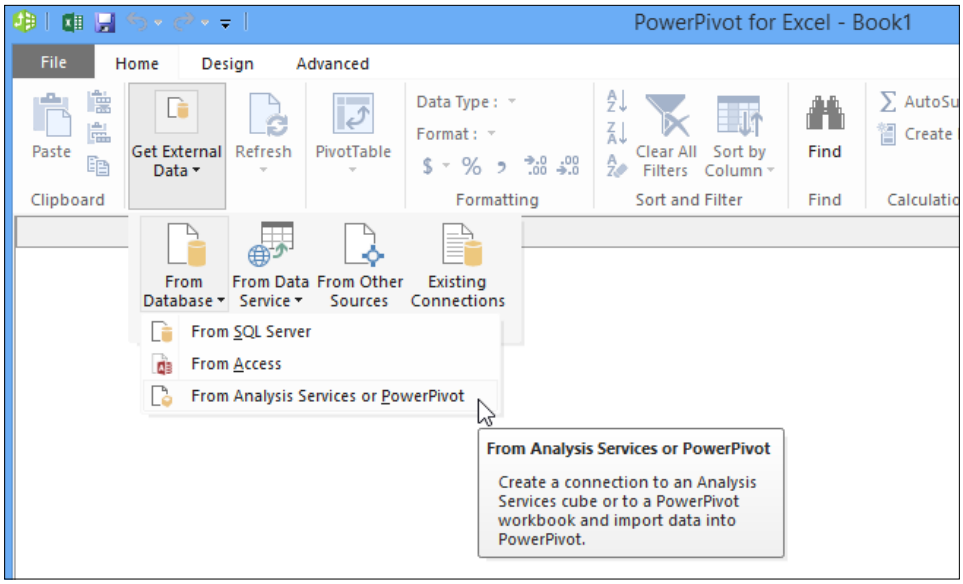
1. Open an empty Excel workbook. Check if the PowerPivot add-in is enabled. If you don't see PowerPivot in the ribbon panel, it's not enabled. To enable, go to **File** | **Options** | **Add-Ins**. Under the **Add-ins** list, select **COM Add-ins** from the **Manage** drop-down box, then click on **Go**, as shown in the following screenshot:



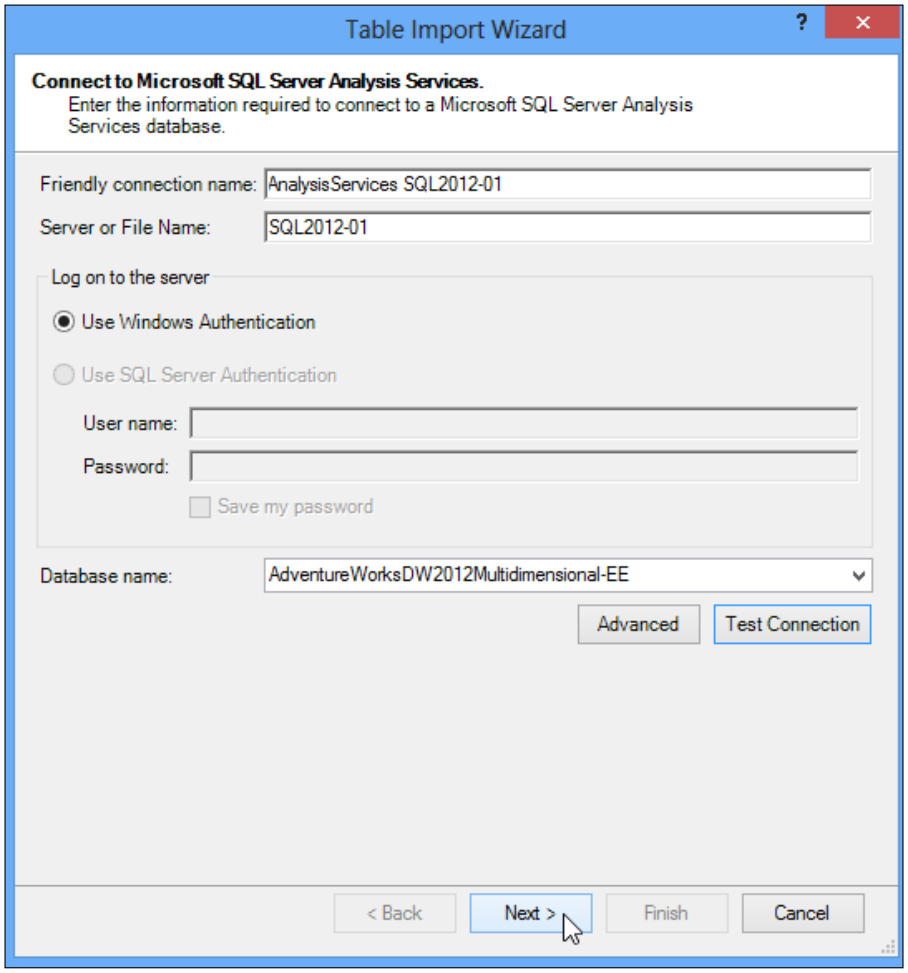
2. In the **COM Add-Ins** dialog box, select **Microsoft Office PowerPivot for Excel 2013** and **Power View**. Click on **OK** when done.



3. You may notice that there's now a new tab created in your workbook named **PowerPivot**. Click on that. Select **Manage** from the ribbon panel.
4. That displays the **PowerPivot for Excel** pane. Click on **Get External Data** and go to **From Database | From Analysis Services or PowerPivot** as shown in the following screenshot:

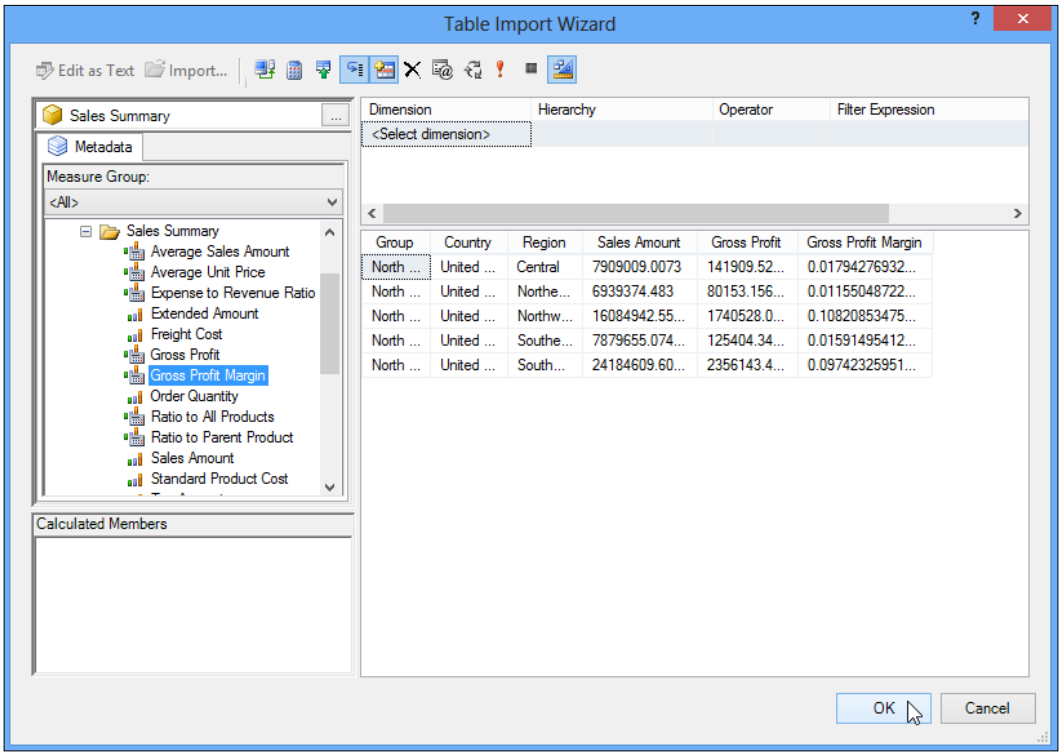


5. That displays the **Table Import Wizard** window. In **Connect to Microsoft SQL Server Analysis Services**, connect to the SQL Server 2012 Analysis Services we created in *Chapter 6*, *Let's Get Analytical!*, `SQL2012-01` in my case. In the **Database name** drop-down list select `AdventureWorksDW2012Multidimensional-EE`. Test your connection. Click on **Next** when done.



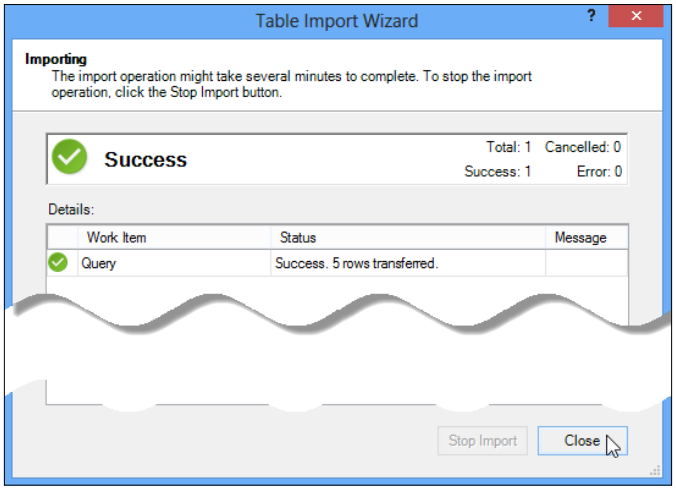
6. Click on the **Design** button in **Specify a MDX Query**.

7. That displays the MDX query designer. Select **Sales Summary** from the cube selector. Then, drag the Sales Territory dimension to the designer. For the measure, expand **Sales Summary** and drag **Sales Amount**, **Gross Profit**, and **Gross Profit Margin** next to the dimensions in the designer:



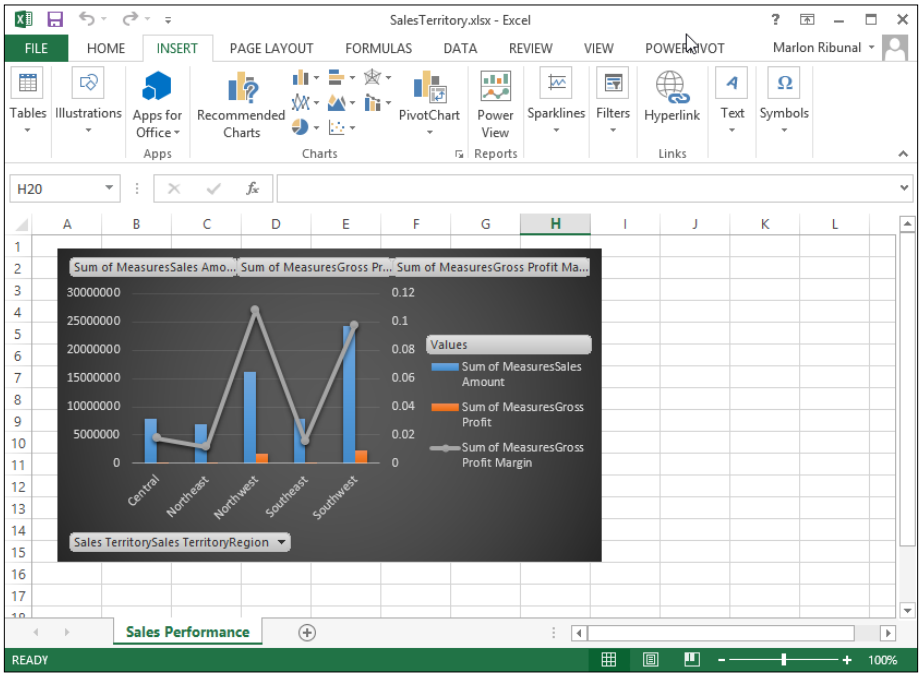
8. Validate the MDX query. Then, click on **Finish**.

9. You'll receive a confirmation when the data import is completed successfully. Click on **Close**.



10. Save the Excel workbook. Name it `SalesTerritory.xlsx`. We're going to need that in the next exercises.

Here's a simple chart that was created by this model:

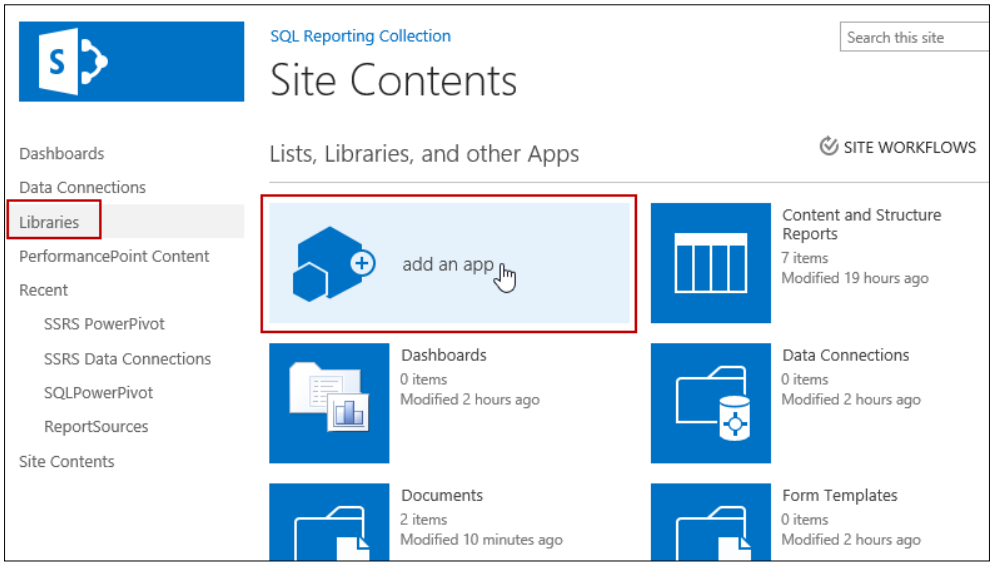


You can do a lot in Excel with data like this, especially with the Power View and PowerPivot add-ins. A lot of books cover that topic.

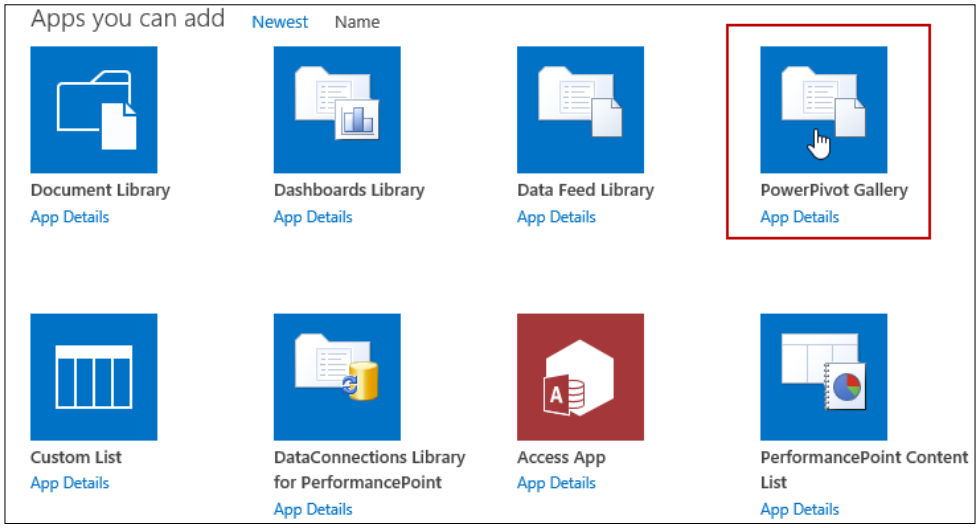
Creating a Power View report in SharePoint

We will see how PowerPivot, Power View, SharePoint, and SQL Server 2012 Reporting Services come together. This is where everything gets exciting. You'll be impressed how Microsoft Business Intelligence offering has come a long way.

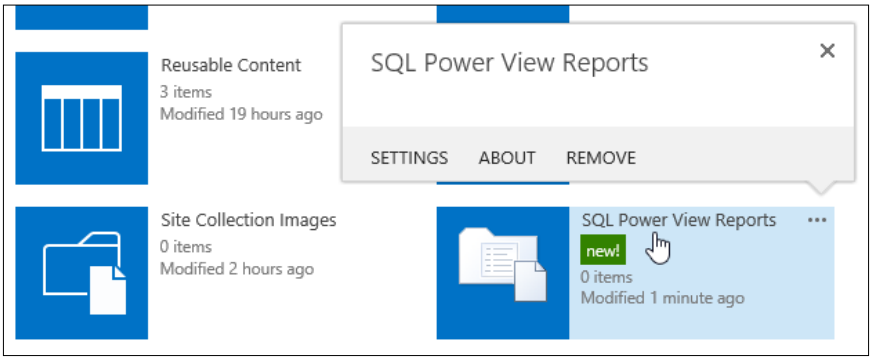
1. Open Internet Explorer and go to the **SQL Reporting Collection** website that we've created previously. If you've followed the previous instructions, the URL is `http://sp2013/sites/SQLReporting`.
2. Create a new library using the `PowerPivot Gallery` template in SharePoint. To do that, from the site's menu, select **Libraries**. Click on **add an app** from the **Site Contents** section:



3. In the apps page, select **PowerPivot Gallery** under **Apps you can add**, as shown in the following screenshot. Microsoft Silverlight is required to correctly render the gallery in your browser:

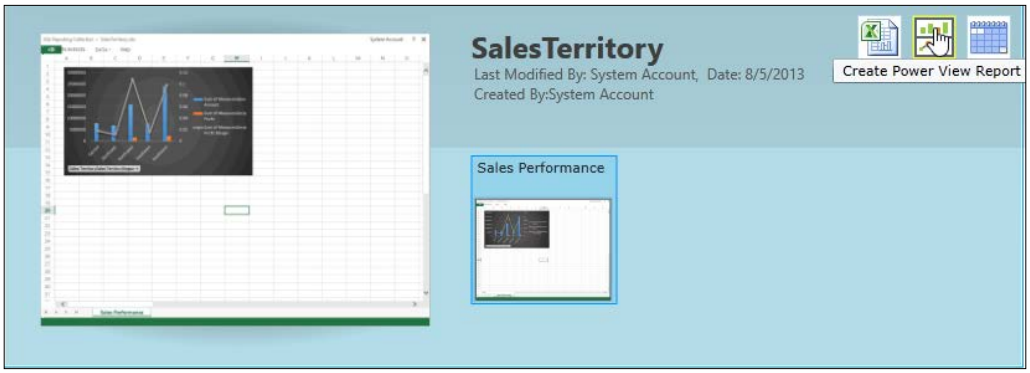


4. Name the new PowerPivot Gallery **SQL Power View Reports**. Click on **Create** to create the gallery.
5. Look for, and open, the new gallery in the **Libraries** section.



6. Go to the **File** tab and select **Upload Document**. Browse for the `SalesTerritory.xlsx` file. Then, click on **OK** to initiate the upload.

7. When the upload is completed, an interface with your document information displays in the design pane. You also have three icons to open a new Excel Workbook, create a Power View Report, and manage data refresh. Select **Create Power View Report**, as shown in the following screenshot:



Now, you can design your Power View report. There are great resources on learning Power View development. Pick a great book on the topic.

Summary

Microsoft has done a great job on the latest iteration of SQL Server Reporting Services. We have various options outside of the traditional report development with Report Server and SQL Server Data Tools. The combination of SQL Server 2012 Reporting Services, Power View, PowerPivot, and SharePoint has definitely raised the bars in Business Intelligence and Reporting.

A

SSRS Best Practices

Location of SSRS database installation

For better performance it is recommended to install the Reporting Services Database on a separate server than the Reporting Services installation. This allows the Reporting Services Server to not have to share resources. This can be critical when larger reports are being generated with charts or complicated tables. The default location is to install Reporting Services on the same server as the SQL Server engine.

Utilizing the SSRS execution logs

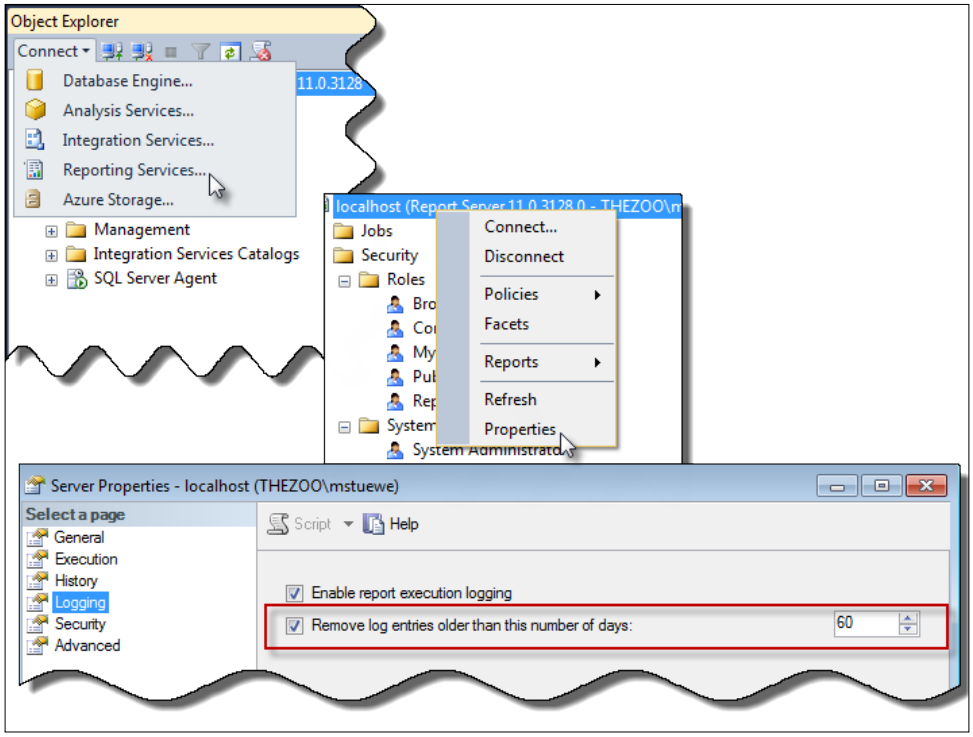
The `ExecutionLogStorage` table in the `ReportServer` database contains a wealth of information to help manage reports. This table contains who, what, when, and how long a report was used. Even reports that were used in subscriptions are listed in this table, along with failed reports.

The `ExecutionLogStorage` table contains only 60 days' worth of data. This is not enough to determine trends of report use. There may be end of quarter reports that are heavily used only a couple of days a quarter.

There are two approaches for increasing how many days of data are stored in this table. One is to increase the setting for how many days are maintained in the `ExecutionLogStorage` table. If this technique is used, then make sure to keep an eye on the growth of the table. To increase the number of days that are saved, follow these steps:

1. Open up **Microsoft SQL Server Management Studio**.
2. Connect to the instance of SQL Server Reporting Services you wish to change.

3. Right-click on the SQL Server Reporting Services name, and select **Properties**.
4. Click on **Logging** in the menu on the left-hand side.
5. Change the number of days to that which should be maintained in the log.



The second approach would be to have the data in the `ExecutionLogStorage` table, summarized and placed in a data warehouse. This will allow you to see trends over time, without causing the `ExecutionLogStorage` table to grow too quickly.

Instead of querying the `ExecutionLogStorage` table directly, you can use a view that is provided with Reporting Services called `ExecutionLog3`. This is a great view to use, since numeric flags have been changed to text for you. The following query summarizes how many reports were run each month by user:

```
SELECT
  ItemPath
   ,CONVERT(varchar(2), MONTH(TimeStart)) + '/' + CONVERT(varchar(4),
YEAR(TimeStart)) AS ExecutionMonth
   ,UserName
```

```
   ,COUNT(ExecutionId) AS CountExecutions
   ,COUNT(DISTINCT ExecutionId) AS CountDistinctExecutions
   ,AVG(TimeDataRetrieval) AVG_TimeDataRetrieval
   ,MAX(TimeDataRetrieval) AS MAX_TimeDataRetrieval
   ,AVG(TimeProcessing) AS AVG_TimeProcessing
   ,MAX(TimeProcessing) AS MAX_TimeProcessing
   ,AVG(TimeRendering) AS AVG_TimeRendering
   ,MAX(TimeRendering) AS MAX_TimeRendering
   ,Status
FROM
  ReportServer.dbo.ExecutionLog3
WHERE
  RequestType = 'Interactive'
  AND [format] = 'RPL'
  AND ItemPath <> 'Unknown'
GROUP BY
  Itempath
   ,CONVERT(varchar(2), MONTH(TimeStart))
   ,CONVERT(varchar(4), YEAR(TimeStart))
   ,UserName
   ,Status
ORDER BY
  Itempath
   ,UserName
```

The `TimeDataRetrieval`, `TimeProcessing`, and `TimeRendering` fields are given in milliseconds. These three fields breakdown how long it takes to create a rendered report and can be very useful to pinpoint which part of the process is causing a slow running report.

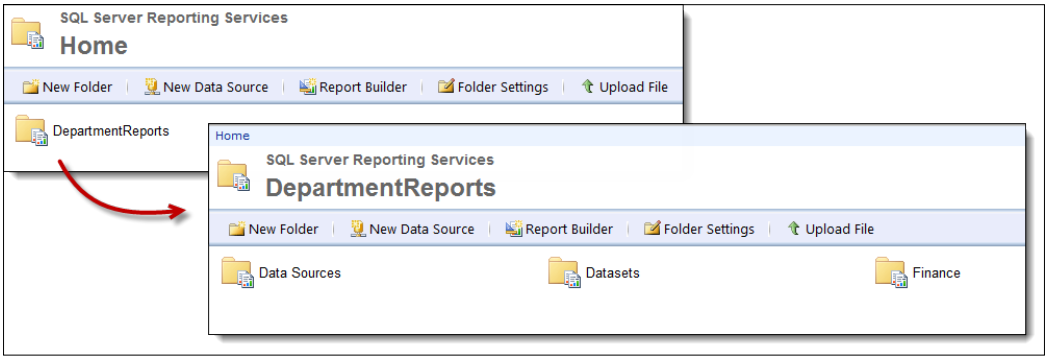
`COUNT(DISTINCT ExecutionId)` was used, since the log contains each time the user clicked on the **View Report** button. By counting the distinct values, you will get how many times they opened the report as opposed to how many times they clicked on the button.

By viewing these metrics, you can determine the usage of the reports over time, which will help you determine reports that need improvement and the reports that should be archived because of disuse.

For more information on the `ExecutionLogStorage` view, please go to `http://technet.microsoft.com/en-us/library/ms159110.aspx#bkmk_executionlog3`.

Report management

One of the easiest ways to manage a large quantity of reports is to organize the reports using folders. The same folder structure should be used for development, and QA servers for ease of deployment and maintenance. The folder structure should be one of the first things architected, since it can be cumbersome, if not impossible, to change later. Since there are third party products that utilize SQL Server Reporting Services, creating a base folder to contain all the reports created by your team is a good first step. An example would look like the following screenshot:



Standardizing queries

Using stored procedures instead of inline SQL statements is a cleaner approach for accessing data. This also allows the query to be tuned without the report needing to be redeployed.

Using standard parameter names and listing similar parameters in the same order for related reports will also streamline the process of creating reports and provide a sense of familiarity to users.

Report standards

Each report should have a few standard pieces of information to help the end user tell one printed copy from another, or for the person at the Help Desk to know which version of the report is being discussed. Here are a few suggestions:

- Keep the report title displayed in the report as close to the same as the report name displayed in the Reporting Services Manager. This helps tying a paper copy to the electronic version.
- Include as many parameters on the report on the first page or in the header as possible, especially date ranges. This helps the end user to remember which parameters they chose when comparing paper copies and provides the Help Desk with vital information to troubleshoot any issues.
- Include a tracking/version number in the footer of the report. The tracking/version number should change each time the report is changed and redeployed to production. This will allow you to determine if a report in question is the latest report or not.
- Include the date and time the report was printed, as well as page numbers.
- Some companies require a statement of confidentiality to be included, as well as the company name and address.

Testing reports

There are several common areas that should be tested and checked for each report. Here is a list of common items to test:

- Date ranges should be tested for one day, one month, several months, one year, and sometimes multiple years. The user may say they only want a small date range, but after they get to like the report, they will start to use larger date ranges. Test the performance of the report as the date range increases. You may find that some reports can't be generated in a reasonable amount of time for large date ranges and a different solution may be needed.
- Make sure a descriptive label is used for the date range parameter and is also placed on the report. This helps later down the road when the end user wants to compare data across several reports. If the date fields are not the same, then the data returned will be different and may not be comparable.
- Test all Expand/Collapse controls that are used in the report. Verify that the correct icon for the plus/minus sign is being displayed. If it is backwards, the `InitialToggleState` property of the textbox may need to be changed.

- Export the report to all supported export types. Verify that the field names are legible for the end user. For example, the `csv` export type uses the textbox name for the column heading. The ultimate test is displaying the report in a **Print Preview** screen of the exported type. This will display headers and footers that may not otherwise be visible.
- If the user's credentials modify how the report looks, or how much data is displayed, then similar credentials should be used to test the report.
- Verify that the parameters are displaying in the correct order, as well as displaying the correct lists and options to the end user.
- Verify the initial sort order of the data. If the data is not explicitly sorted in a particular way, then data may display in a different order on a different server due to the indexes or lack of indexes in the database, as well as how Reporting Services processes the data.

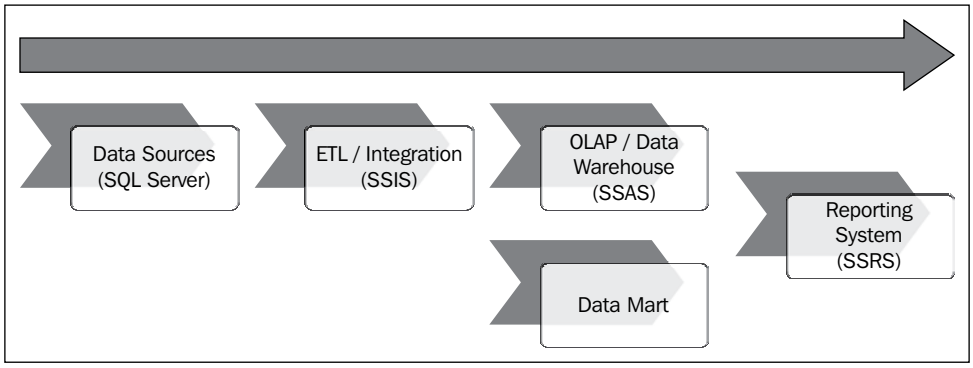
B

Transactional Replication for Reporting Services

The first concern of any report developer—as far as report development is concerned—is how to implement a reporting solution without negatively impacting the existing data infrastructure. Reporting requires significant resources, process and infrastructure-wise. You cannot just build a SQL Server Reporting Services (SSRS) system and slap it in your primary production SQL Server OLTP. It just doesn't work that way.

As a developer, your highest concern is to steer clear of any possible conflict with the performance of any production databases. You should be wary of the implications of building a reporting system in your environment.

An ideal reporting delivery project usually involves the whole array of Microsoft Business Intelligence (BI) stack. The process would look something like this:



An infrastructure, as illustrated, previously would be too expensive to implement. Data cleansing and integration in the **Extract**, **Transform**, and **Load** (**ETL**) phase can take a huge amount of time and resources to complete. Implementing an **Online Analytical Processing** (**OLAP**) system is rigorous and expensive as well, probably more expensive than ETL/Integration. Plus, the hardware requirements for either of the two (ETL and OLAP) can blow your whole department budget.

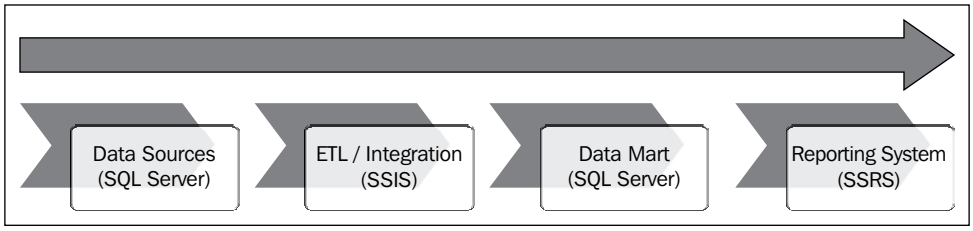
In an economy where businesses are cautious with their finances, it would be next to impossible to convince the management to embark on such project.

It's really hard to justify the cost of implementing a project of this magnitude to the stakeholders. And there is a reason for that: A reporting system is seen by most as a matter only of housekeeping—data tracking, bookkeeping, and analysis—and that it is not worth the overhead it incurs on the finance aspect of the business.

Another concern that a developer should consider, therefore, is the political aspect of the reporting project. But, that is a topic better addressed in some other books.

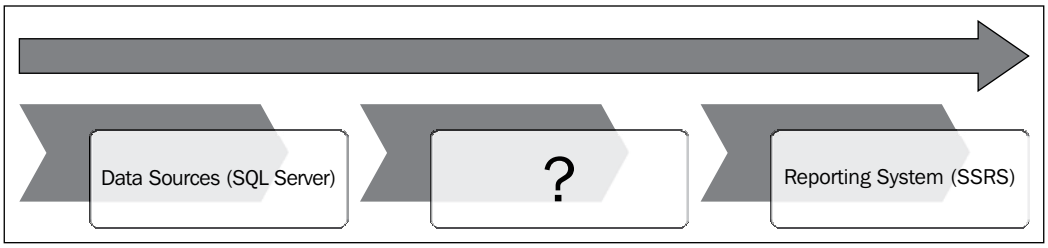
Options for report delivery

Let's just say that a full stack BI solution is not an option in your **SSRS 2012** project implementation. We agree that it's too expensive for most reporting system requirements. We could simplify the structure into something like the following diagram:



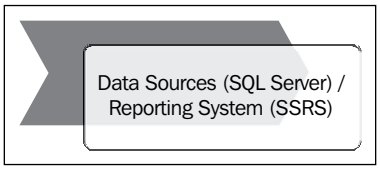
In the preceding illustration, we eliminate the most expensive part of the infrastructure, the OLAP. We cut a great portion of the project cost but we probably eliminate the most important segment of our BI. But, that's just fine for the most part because our data marts might just be adequate for what we need.

But still, a reporting system that requires data cleansing and integration is not cheap either. Let's just say that the management still couldn't afford such a system. So, we might end up with something like the following diagram:



Now that we eliminate the ETL integration portion, we are left with a big gap between our data store and reporting system. We have a backend (data store) and frontend (reporting system) but we don't have a middle process that would bridge the two.

The worst possible thing that you would resort to is eliminating the in-between process:

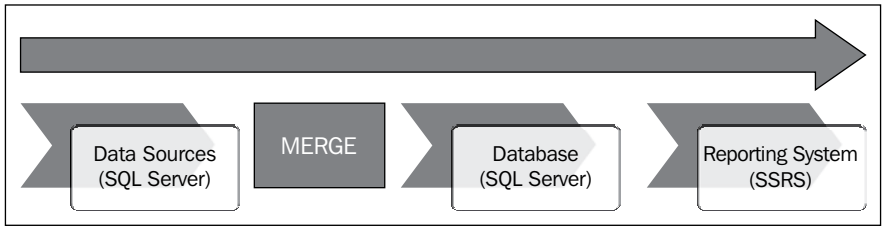


You cannot afford to put your SQL Server Reporting Services inside your primary SQL Server boxes. You could probably save some money but it could cost you more in the long run.

Our solution then is to put some process between our data stores and reporting system.

One option that could close that gap is to build a separate SQL Server box, say SQL Box 2, with similar database structures as our original data source, say SQL Box 1, and use SQL Box 2 as the data source for our reporting system. We then implement some merge mechanism to transfer data from SQL Box 1 to SQL Box 2. In this scenario, we could use T-SQL scripts with a joining clause such as `MERGE`.

The whole process could look something like the following diagram:



Again, this setup looks decent. It's practical and economical to implement. You could probably run the scripts through a SQL Server Agent Job and run them at schedule. But, there's a little problem with this.

The performance of the merging process still depends on so many things such as hardware, joins, object dependencies, indexes, and many others. There are just too many points of failure in this setup.

Performance is not just the main concern in the merge setup. Performance is just one side of the coin. The other side is time. There is high latency between when the merge occurs and the completion of the merge. Although you can automate the merge process, it could still take some time to complete the merging.

Transactional replication to the rescue

Now, we can eliminate the need for merging by implementing **transactional replication**. What we can accomplish with transactional replication is to propagate the changes in the source database (publisher) to the target database (subscriber) in near real time.

To understand how replication works in SQL Server, think about how a newspaper publication works. Replication actually uses the metaphor in publishing.

Here are the main components of replication:

- **Publisher**: This is the source database. It is where we need to replicate the data from. Your other data store is not SQL Server? No problem! You can use an Oracle database as a source.
- **Subscriber**: This is the recipient or the target database. A subscriber can turn to be a publisher as well.
- **Distributor**: In newspaper publishing, the publisher needs a distributor to distribute its publication to its subscriber. In replication, a distributor handles the command that is executed on the subscriber.

- **Publications**: In the newspaper publishing metaphor, this is the actual newspaper. Publications in replication contain common objects that are to be propagated to the subscriber.
- **Articles**: Just like a newspaper has articles in it, news, ads, editorial, essays, and many others, replication articles are objects that can be replicated from the source database to the target database such as functions, stored procedures, tables, views, and index views.

There are actually other types of replication, aside from transactional replication, but we're not going to discuss all those types here. The other types of replications are:

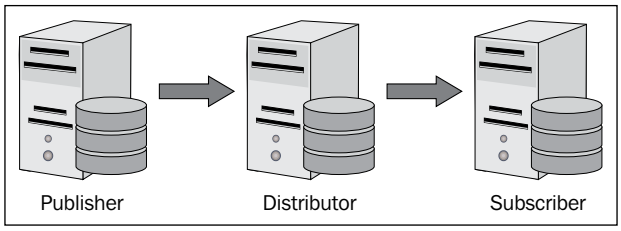
- Snapshot replication
- Peer-to-peer transactional replication
- Merge replication

Implementing transactional replication

Implementing the basic transactional replication topology is not really a complicated process. The **SQL Server Management Studio** (**SSMS**) provides a wizard to set up the replication.

There are few topologies that you can implement, depending on the complexity of your replication requirements.

In this step-by-step process, we are going to implement only the basic topology, which looks like the following diagram:



There are three general steps in implementing transactional replication. These are setting up the following (in order):

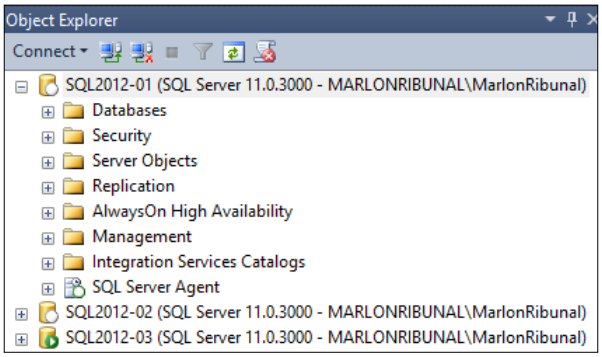
- **Distributor**
- **Publisher**
- **Subscriber**

Before we get started, make sure that a copy of the `AdventureWorks2012` database is installed on your publisher.

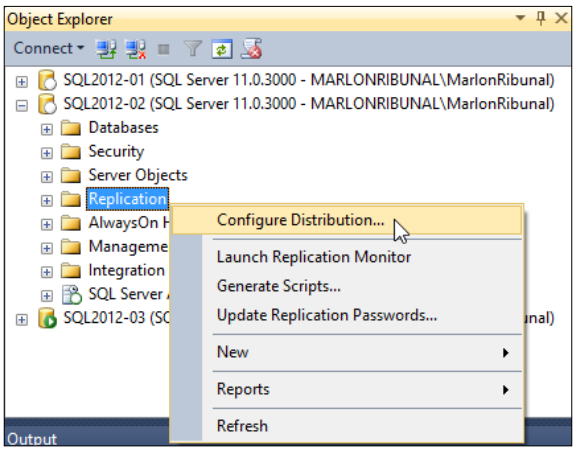
Setting up the distributor

The first step in implementing a transactional replication is setting up the distribution. Actually, the general steps are similar in all other types of replication, although there are additional steps to take in the other types:

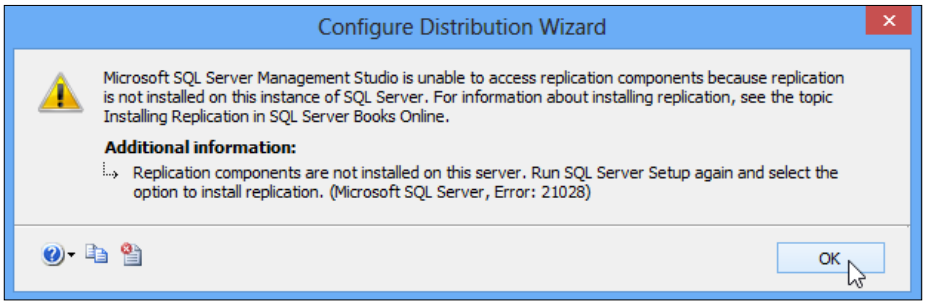
1. Open the SQL Server Management Studio (SSMS). Typically, you have three separate servers setup for the publisher, distributor, and subscriber. In these demonstrations, these roles are assigned to `SQL2012-01`, `SQL2012-02`, and `SQL2012-03` respectively.



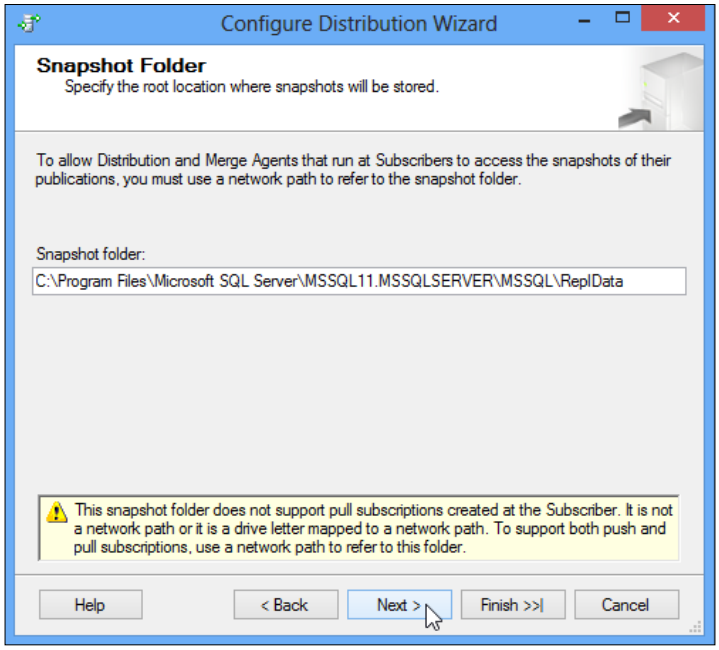
2. Expand the `SQL2012-02` server object in the **Object Explorer** window. Select and right-click on the `Replication` folder, and select **Configure Distribution** as shown in the following screenshot:



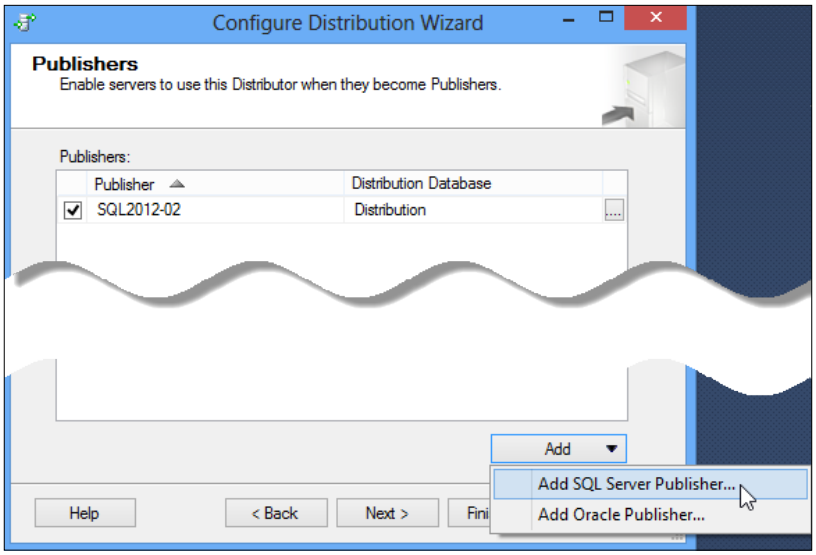
3. If you get an error message like the one in the following screenshot, you need to run the SQL Server setup on the server and install the `Replication` component. If you haven't done that already, go and install `Replication` in all three SQL Server instances:



4. If the `Replication` component is properly installed in your server, selecting the **Configure Distribution** initiates **Configure Distribution Wizard**. Click on **Next**.
5. For the **Distributor** server, select the **default** distributor, which is the current server. Click on **Next**.
6. In the next step, type in the folder location for **Snapshot folder**. Read the important information at the bottom of the dialog box. Click on **Next**:



7. On the next step, type a **Distribution Database** name. For this demonstration, **Distribution** for a name is just fine. Accept the default file location of the database file and log file. Click on **Next**.
8. The next step will ask you to specify a **Publisher** server. At the bottom of the dialog box, click on the **Add** button, and select **Add SQL Server Publisher**, which is `SQL2012-01` in our case. You may notice that `SQL2012-02` is also listed as **Publisher** which means that I will also use it as a publisher at some point (`SQL2012-02` will be both distributor and publisher). Click on **Next**.

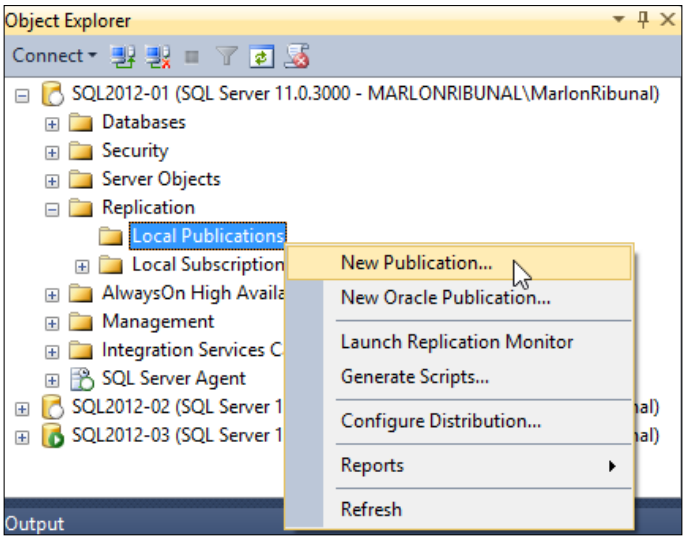


9. That displays the **Connect to Server** prompt of SQL Server. Log in to `SQL2012-01`. That adds `SQL2012-01` to the list of **Publishers**. Click on **Next**.
10. At the next step, specify a **Distributor Password**. Make sure that the password you specify complies with the existing password policy of your network. No validation is done this time. You'll only know whether or not the password creation is successful at the end of the wizard. Click on **Next**.
11. At the end of the wizard, you have the option to generate a script file to configure the distribution. You can take a look at the generated scripts. I recommend you don't configure replication with the script unless you really know what you are doing. Make sure **Configure distribution** is selected, and then click on **Next**. Review the configure list, and then click on **Finish** to complete the wizard.
12. Double check if the distribution is configured successfully. Click on **Close** when done.

Configuring the publication

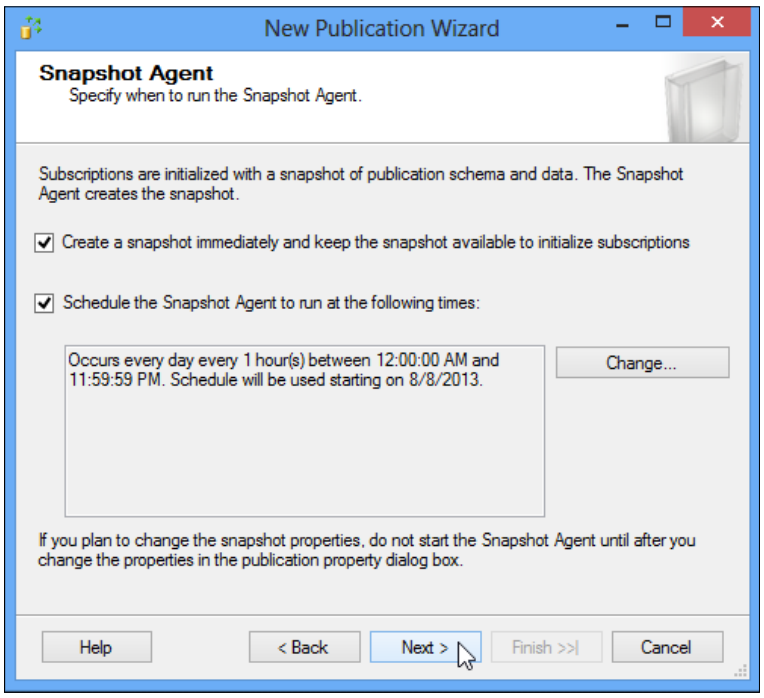
The next step, after setting up distribution, is publication. This is the part where we select the objects that we want to propagate to the subscriber. In the distribution, we set up `SQL2012-01` as the publisher. So, let's go to that server and configure the publication:

1. In the **Object Explorer** window, expand the `SQL2012-01` object. Then, expand the `Replication` folder. Select and right-click on the `Local Publications` folder, and select **New Publication** as shown in the following screenshot:



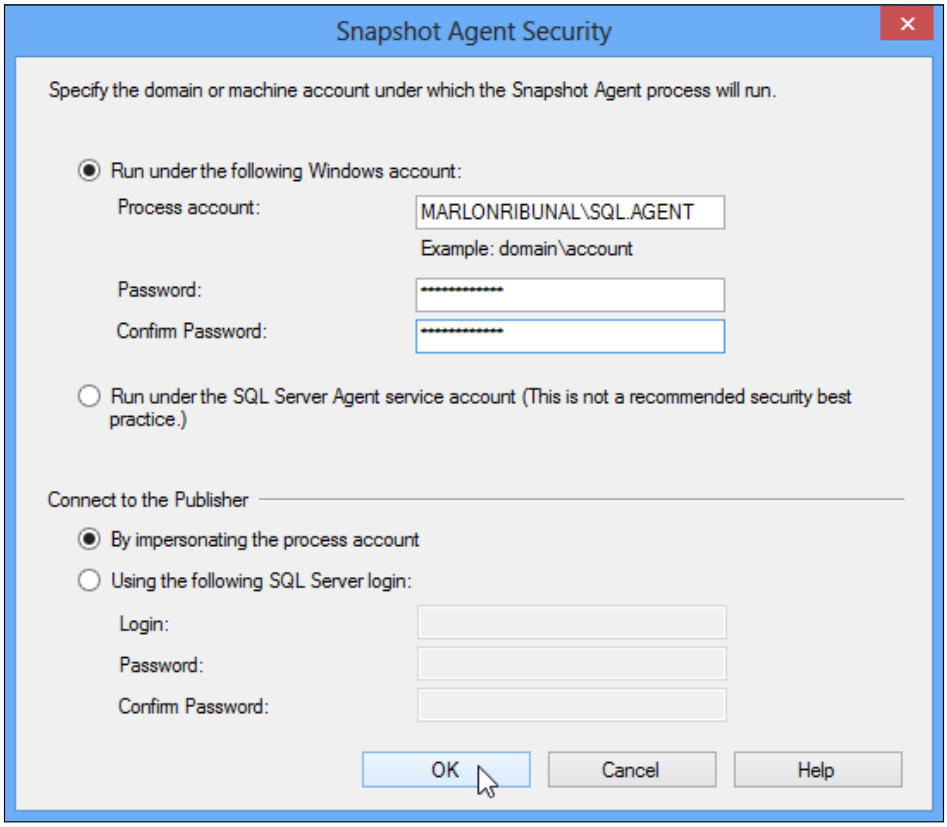
2. That initializes the **New Publication Wizard**. Read the information on the dialog. Click on **Next**.
3. On the select **Distributor** step, select the second radio button to specify we want to use a different distributor. Click on the **Add** button, which displays the **Connect to Server** prompt. Log in to `SQL2012-02` which we designated as our distributor. Click on **Next**.
4. At the **Administrative Password** dialog, specify the password that we entered when we set up the distributor. This allows the publisher to contact the distributor. Click on **Next**.
5. The next step allows us to choose the database the data or objects that we want to publish. In the **Databases** list, select `AdventureWorks2012`. Click on **Next**.
6. In the **Publication** type, select **Transactional publication**. Click on **Next**.

7. Now, we get to select the database objects, or the articles that we want to publish. Expand the tables and select `SalesOrderHeader` and `SalesOrderDetail`. You may notice you can select an individual column too. This is huge because you may not want to propagate sensitive data that has security risks, such as social security numbers, credit card numbers.
8. Use the **Article Properties** drop-down list to set properties of the table articles. We'll just click on **Next** now.
9. The **Filter Table Rows** step gives you another opportunity to filter your data. Just click on **Next** for now.
10. On the next step, **Snapshot Agent**, specify that we want to create a snapshot immediately. And also, select to run the Snapshot agent at a particular schedule.



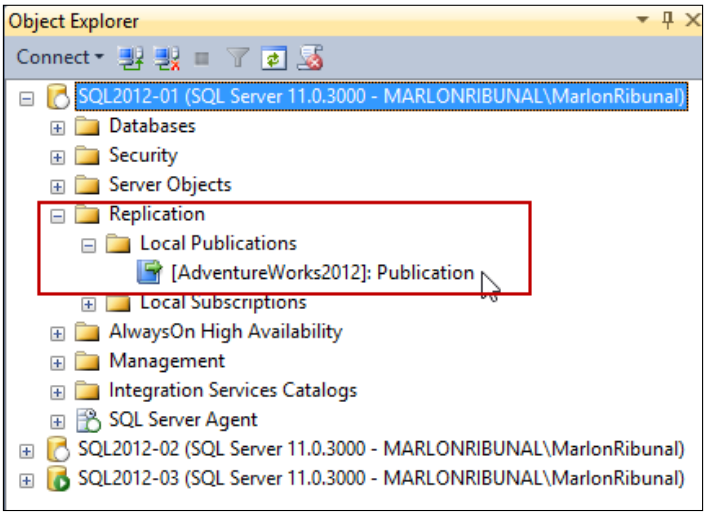
11. Click on **Next** when done.

12. On the **Agent Security** step, specify a domain account that is either the SQL Server Agent account's security context or a Windows account with low privilege. Click on the **Security Settings** button to configure. Click on **Next**.



13. On the next two steps, select the option to create this publication and give it a name. Click on **Finish** to complete the setup.

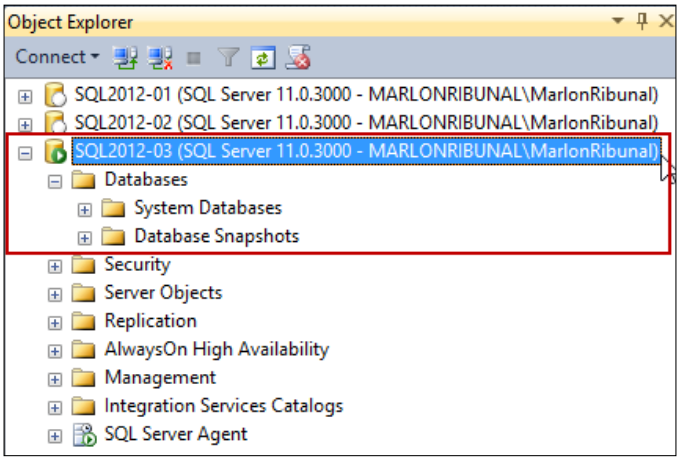
14. Verify that the publication is successful. Check the **Object Explorer** window if the **Publication** object now exists:



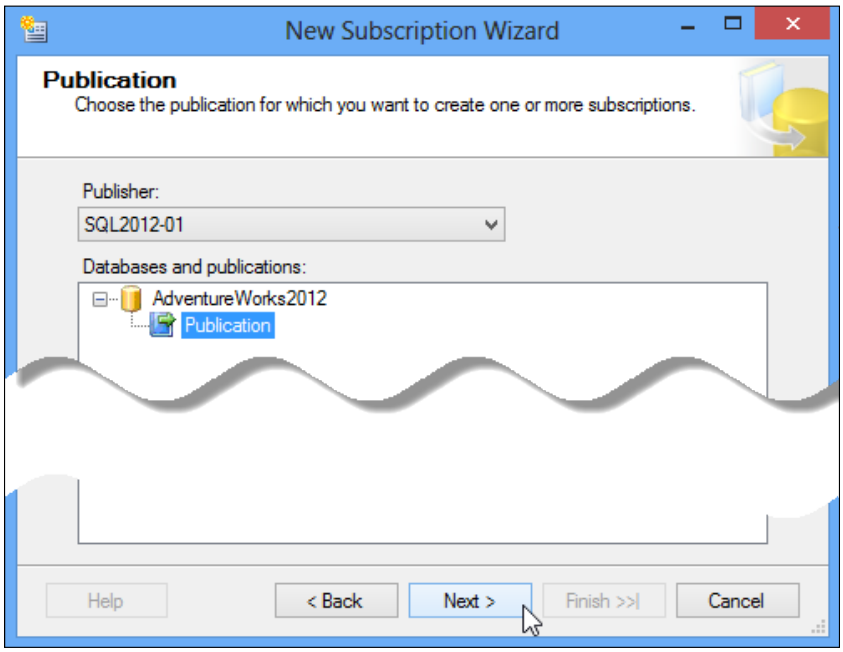
Getting the subscription ready

Our transactional replication is almost complete. We've already set up the distribution. We've also configured the publication. Articles are now ready to be propagated and distribution just needs to identify the subscribers.

We'll set up a third SQL Server box to act as subscriber, in this case `SQL2012-03`. As you can see, at this point, our subscriber server is empty:

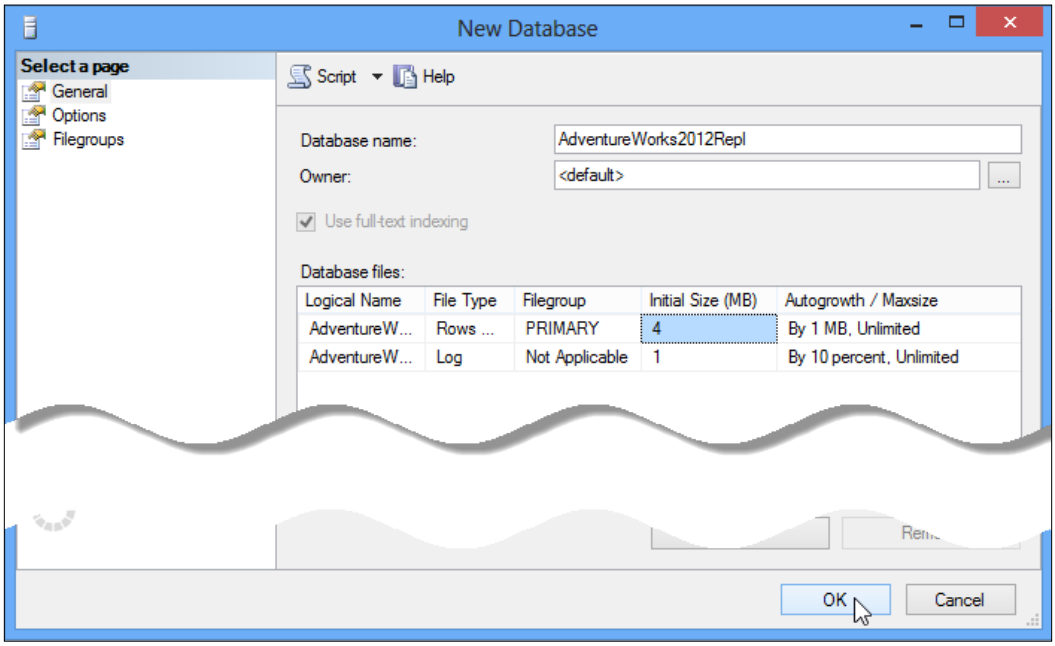


1. In the **Object Explorer** window, expand the **SQL2012-03** object. Then, expand the `Replication` folder. Select and right-click on the `Local Subscriptions` folder, and select **New Subscription**.
2. That initializes the **New Subscription Wizard**. Read the information on the dialog. Click on **Next**.
3. On the next step, the **Publication**, click on the **Publisher** drop-down textbox. If the publisher server is not listed, in this case `SQL2012-01`, select **<find SQL Server Publisher>** which prompts you to login to the publisher, `SQL2012-01`. When the databases and publications appear in the listbox, select the publication and click on **Next**.



4. The next step gives you two options: push or pull subscriptions. The default is pull subscription. Read the description under each option. For now, let's accept the default (pull). Click on **Next**.

5. On the next step, specify the subscriber server, `SQL2012-03`, on the **Subscription Database** column, click on the drop-down and select **<new database>**. That displays the **New Database** dialog. Give the new database a name, like `AdventureWorks2012Repl`. Accept the **default** for now. Click on **OK** to create the database.



6. Click on **Next**.
7. On the **Distribution Agent Security**, enter an appropriate distributor and subscriber account by clicking on the ellipsis, which displays the **Distribution Agent Security** dialog. Click on **OK**. Then, click on **Next**.
8. On the **Synchronization Schedule**, select **Run Continuously**. Click on **Next**.
9. On the **Initialize Subscription**, make sure that **Initialized** is selected and **Initialize When** is set to **Immediately**. Click on **Next**.
10. Select **Create the subscription(s)** in the **Wizard Actions**. Click on **Next**. Then, click on **Finish** to complete the subscription.
11. Verify the subscription.

Summary

The ideal backend infrastructure of a reporting system is one that is supported by the full stack of Microsoft Business Intelligence—SQL Server, SQL Server Integration Services, and SQL Server Analysis Services. Not everybody can afford such an expensive system. But that should not be enough to deter an organization from pursuing a decent reporting project. Many options such as merging and replication can be implemented to put together a seamless flow of data from their sources to the stakeholders.

Index

Symbols

@SalesTerritory parameter 66
@Subcategory parameter 34

A

Add New Item dialog box 56
alerts
  handling 186
ALL parameter 70
Analysis Services
  deployment setting, changing 138-141
  new inbound rule, creating 136-138
  TCP port, checking 134, 135
Articles 211

B

bar color
  blue 91
  green 91
  red 91
  setting, conditionally 91, 92
bookmark
  creating 74-76

C

cached reports
  creating 171, 172
calculated member
  creating 154-156
Calculated Member Builder dialog box 155
cascading multivalued parameter
  creating 32, 33
cascading parameter
  creating 29-32
center points
  adding 116
chart
  data, adding to 82, 83
  formatting 83, 85
CodePlex 9, 134
Color Analytical Map 110
color rules
  changing 114
COM Add-Ins dialog box 194
Confirm Action dialog box 90
custom report template
  about 55
  using 55, 56

D

data
  adding, to chart 82, 83
Data Analysis Expressions (DAX) 144
data bars
  embedding, in tablix 87, 88
data-driven subscription
  about 176
  delivery properties, setting up 177
  parameter values, setting up 178
  shared data source location, selecting 176
  SQL statement, setting up 176, 177
  subscription schedule, setting up 179
  type, specifying 176
data label
  showing 89, 90
Data Mining Extensions (DMX) 144

dataset
defining, for project 11-13
drop-down parameter, populating with 24-27
filter, adding in 151
parameters, adding 152, 154
using, on query designer 147-149
using, on report designer 150
dataset based multidimensional model
adding 144
Dataset Properties query window 80
data source
adding, to model 193-198
defining, for project 9-11
data source, for report
creating 142, 143
deployment configurations 163, 164
detail report
creating 61, 62
distribution options
customizing 116-118
distributor
about 210
setting up 212-214
Document Map
adding 73, 74
Drilldown
implementing, with row visibility 47, 48
drillthrough
creating 65-69
drop-down parameter
populating, with dataset 24-27

E

e-mail delivery / scheduled delivery subscription 174
empty center points
removing 121, 122
expression
formatting 41, 42
literals, formatting in 44-46
Extract, Transform, and Load (ETL) 208

F

filter
dataset, adding in 151
FormatPercent function 103

G

gauge
pointer, configuring 98
range, defining 99-101
scale, adding 98, 99
using, in tablix 97
gauge pointer
configuring 98
gauge range
defining 99-101
Get the value from the database option 177
grand total
adding 50

H

hyperlink
adding 70

I

image
adding, in header 51, 52
indicator
adding 101-103
InitialToggleState property 205
In-Store versus Online Sales 81
interactive sorting 72, 73

K

KeepTogether property 65

L

line layer
adding 128, 129
linked report
creating 65-69
literals
formatting, in expression 44-46

M

main report
creating 58-61
manual deployment
migrating, to production 165-168
map
formatting 113
map design
formatting options 130, 131
Map Layers pane 131
map (polygon) visualization
creating 106-108
model
data source, adding to 193-198
Multidimensional Expressions (MDX) 144
multidimensional model
deploying 142
multivalued parameter
creating 28, 29
MultiValueParameterSplit function 33

N

New Database dialog 220

O

Object Explorer window 212, 219
Online Analytical Processing (OLAP) 208

P

page footer
adding 52
page header
adding 51
page number
adding 52
parameter
making optional 21-24
parameterized report
creating 15-19
PowerPivot 193
Power View
about 192
versions 192
Power View report
creating, in SharePoint 198-200
Print Preview screen 206
project
data source, defining 9-11
publication
about 211
configuring 215-218
publisher 210

Q

queries
standardizing 204
query designer
used, for dataset building 147, 149
working with 144, 146
Query Designer window 156

R

radio button parameter
creating 20, 21
random point layer
adding 123-126
report
managing 204
page setup 39, 40
standards, maintaining 205
testing 205, 206
report delivery
options 208-210
report designer
dataset, using on 150
report file
creating 13, 15
Reporting Service Application
about 183
creating 183-185
report management
about 204
example screenshot 204
report objects
adding 42, 44
report standards
maintaining, suggestions 205

report template
creating 54, 55
row colors
alternating 53
row group
creating 46, 47
RowNumber function 53

S

sales details report 34
Select Indicator Type dialog box 101
shapefile
about 119
using 119, 120
shared schedules
setting up 170, 171
SharePoint
Power View report, creating in 198-200
SSRS, integrating with 182, 183
SSRS projects, deploying 190, 191
simple chart
about 80
creating 80-82
site collection
creating 187-189
snapshot
about 173
setting up 173
Snapshot folder 213
sparklines
adding, in tablix 92-94
axes, aligning in 95, 96
spatial and data views
about 108
options 108-111
SQL Server Reporting Service (SSRS)
integrating, with SharePoint 182, 183
SQL Server Analysis Services (SSAS) 8
SQL Server Data Tools (SSDT) 8
SQL Server Integration Services (SSIS) 8
SQL Server Management Studio (SSMS) 211
SSRS2012Blueprints 9
SSRS database installation
location 201
SSRS execution logs
utilizing 201-203
SSRS projects
deploying, to SharePoint 189, 191
subreport
adding 62-64
subscriber
about 210
identifying 218-220
subscriptions
handling 186
testing 179
subtotals
adding, for each group 49
Switch function 94

T

Table Import Wizard window 195
tablix
about 19
data bars, embedding in 87, 88
gauge, using 97
sparklines, adding in 92, 94
TargetDatasetFolder property 164
TargetDataSourceFolder property 164
TargetReportFolder property 164
TargetReportPartFolder property 164
TargetServerURL property 164
TargetServerVersion property 164
TCP port, Analysis Services
checking 134, 135
theme and data visualization 112
tile layer
adding 130
tool tip
adding 71, 85, 86
transactional replication
components, articles 211
components, distributer 210
components, publication 211
components, publisher 210
components, subscriber 210
diagram 211

implementing 210-212
implementing, setting up 211
setting up 211
types 211

U

user security 168, 169

V

View Report button 203

W

web application
creating 186

[PACKT] PUBLISHING enterprise professional expertise distilled

Thank you for buying

SQL Server 2012 Reporting Services Blueprints

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`.

About Packt Enterprise

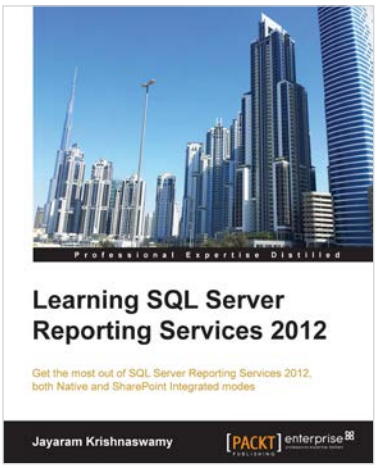
In 2010, Packt launched two new brands, Packt Enterprise and Packt Open Source, in order to continue its focus on specialization. This book is part of the Packt Enterprise brand, home to books published on enterprise software – software created by major vendors, including (but not limited to) IBM, Microsoft 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. 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.

[PACKT] PUBLISHING enterprise professional expertise distilled



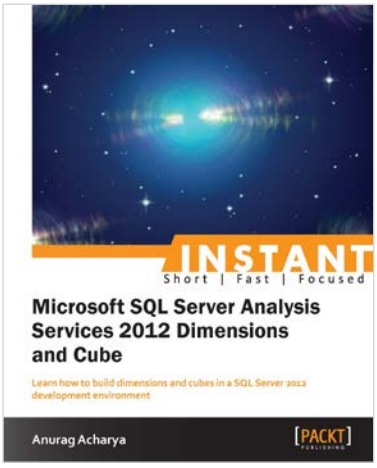
Learning SQL Server Reporting Services 2012

ISBN: 978-1-84968-992-2 Paperback: 566 pages

Get the most out of SQL Server Reporting Services 2012, both Native and SharePoint Integrated modes

1. Build applications using the latest Microsoft technologies: SSIS 2012, SSDT, WPF, and SharePoint 2010
2. Reach out to the cloud and master Windows Azure Reporting Services
3. Learn the ins and outs of SQL Server Reporting Services 2012 for Native and SharePoint Integrated modes

---



Instant Microsoft SQL Server Analysis Services 2012 Dimensions and Cube

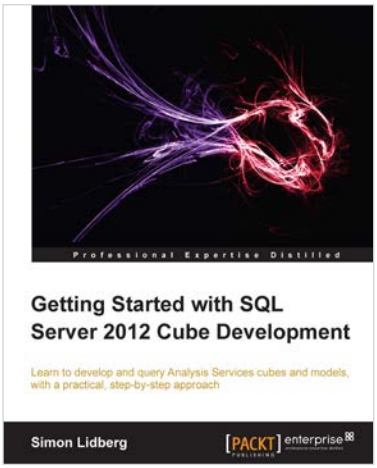
ISBN: 978-1-84968-872-7 Paperback: 72 pages

Learn how to build dimensions and cubes in a SQL Server 2012 development environment

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results.
2. Create your own SQL Server development environment
3. Full of practical tutorials on cube design and development

Please check **www.PacktPub.com** for information on our titles

[PACKT] PUBLISHING enterprise professional expertise distilled



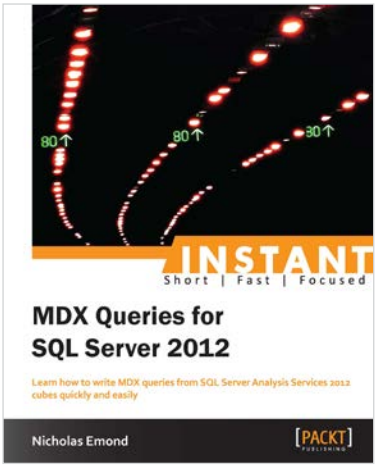
Getting Started with SQL Server 2012 Cube Development

ISBN: 978-1-84968-950-2 Paperback: 288 pages

Learn to develop and query Analysis Services cubes and models, with a practical, step-by-step approach

1. Learn how to develop a complete business intelligence solution using SQL Server 2012
2. Understand the difference between tabular in-memory models and OLAP cubes, and which to use where and when
3. Add advanced features such as key performance indicators (KPIs) and calculated measures to your business intelligence model

---



Instant MDX Queries for SQL Server 2012

ISBN: 978-1-78217-806-4 Paperback: 54 pages

Learn how to write MDX queries from SQL Server Analysis Services 2012 cubes quickly and easily

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results
2. Write queries, use functions, and utilize what you write in cubes and reports
3. Write Calculated Members and Named Sets

Please check **www.PacktPub.com** for information on our titles