



[TechNet Home](#) > [Product & Technologies](#) > [SQL Server TechCenter Home](#) > [SQL Server 2005](#)

Reporting Services Solutions, Patterns and Recipes

Published: April 14, 2006

Writer: Paul Turley, Hitachi Consulting

Excerpts taken from Chapter 7 of [Professional SQL Server 2005 Reporting Services](#), second edition, from WROX Press.

The book is written by Paul Turley, Dave DuVarney, James Counihan and Todd Bryant.

This chapter, entitled Report Solution Patterns and Recipes, contains a deep analysis of successful report projects, including project profiles, success factors, user discussions and scope management. Templates are provided to assist requirement gathering and management. The Report Recipes section of this chapter includes 19 specific examples of advanced report designs that implement custom coding and other techniques taught in previous chapters. I've selected 8 of these examples to demonstrate in this condensed version. Note that these are selections taken from throughout the chapter as evidenced by the non-contiguous figure numbers.

Report Recipes

As we have endeavored to solve various business problems, we've learned to do some interesting things with Reporting Services. On consulting engagements, I often find myself in front of a client who is asking questions like "can you do this or that?" Almost inevitably, the answer is "yes," but the question becomes what the best method would be to meet the requirement. With a little outside-the-box thinking, a lot of interesting things are possible. This may involve some custom programming, embedding report items or using customer application components in concert with Reporting Services.

In the following section, I've compiled a description of reporting challenges and solutions we've encountered, developing reports for our clients. For each "solution recipe," I provide a brief list of skills, techniques, and resources needed to apply the report feature. This should give you a good idea about how prepared you may be to use the techniques based on your skill set and the level of complexity. Some of these are easy to duplicate and others require more advanced skills, which may include Transact-SQL and Visual Basic programming. These are not intended to be exercises or step-by-step instructions. I have made a point to provide enough information to demonstrate the concepts and techniques. However, to implement these solutions you will need to apply the skills you learned in the previous chapters.

Greenbar Reports

Once-upon-a-time, most reports were printed on special continuous-feed paper. This paper is fan folded, with a perforation between each page, making it stackable in the input and output printer bins. The long scroll of pages has pin-feed holes on each side to feed it through and align each row with the mechanical print head. One of the common characteristics of this paper is that it has pre-printed green bars for every-other row data. In more modern reports, this format remains popular to help readers visually separate each row of printed information. This typically involves using a light pastel background color for alternating table rows.

What you'll need:

- A Visual Basic function
- Expressions used to call the function on the BackgroundColor property of row items

I've seen a few different techniques used to implement this feature and they all require complex expressions or some use of Visual Basic programming. Fortunately, this isn't hard to do, even if you're new to VB programming. This technique involves using a VB function to return a different color for odd and even rows.

Report items are rendered from top to bottom and then from left to right—like the carriage of a typewriter (for the younger generation, a typewriter is sort of like a computer with moving parts.) This means that custom code procedures and expressions associated with report item properties will always be executed in this order. In our solution, the start of a new row is indicated by passing a toggle flag when the function is called in the left-most column textbox. The following Visual Basic code begins with a class module-level variable used to hold the odd-or-even row indicator between calls. As you can see, the `bOddRow` variable value is toggled between `True` and `False` on each call.

```
Private bOddRow As Boolean
'*****
' -- Display green-bar type color banding in detail rows
' -- Call from BackGroundColor property of all detail row textboxes
' -- Set Toggle True for first item, False for others.
'*****
Function AlternateColor(ByVal OddColor As String, _
    ByVal EvenColor As String, ByVal Toggle As Boolean) As String
    If Toggle Then bOddRow = Not bOddRow
    If bOddRow Then
        Return OddColor
    Else
        Return EvenColor
    End If
End Function
```

The program code is entered on the Code tab of the Report Properties dialog. To access this window, choose Report Properties from the Report menu while using the report designer's Layout tab. This is shown in Figure 7-1. After entering or making modifications to code, click the OK button to update the report definition.

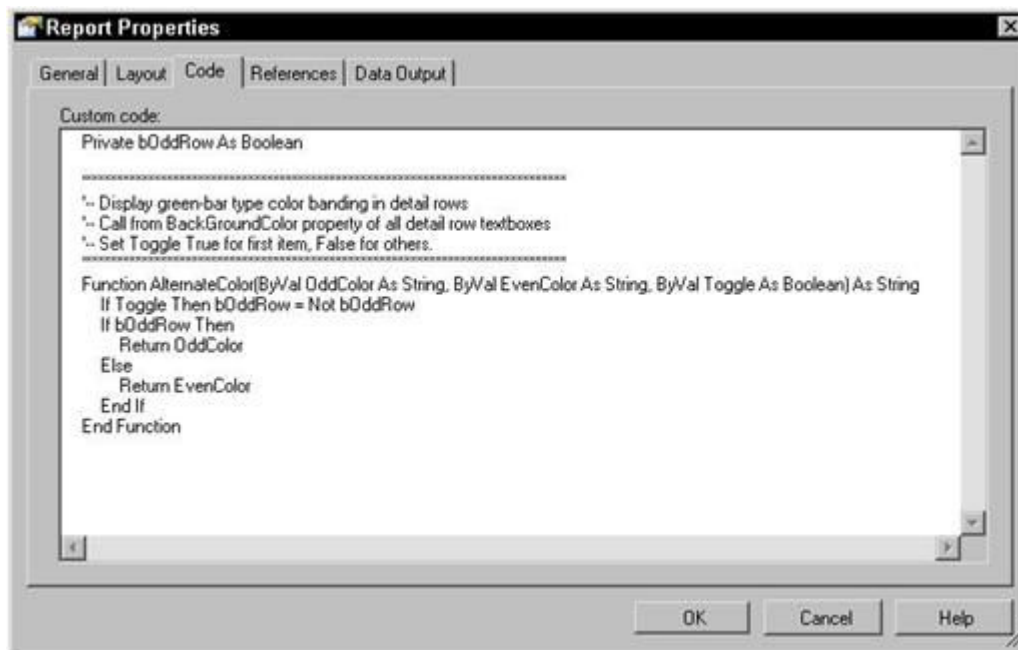


Figure 7-1

[See full-sized image](#)

For the `BackgroundColor` property of the first textbox (in the left-most column of the row,) enter the following expression to call the custom code function:

```
=Code.AlternateColor("AliceBlue", "White", True)
```

Creating a Greenbar Table

Figure 7-3 shows the report in preview. The AliceBlue color I chose for odd rows is subtle. Any combination of standard color names or hexadecimal values can be used. For example, the hex value #FF0000 is equivalent to the color Red.

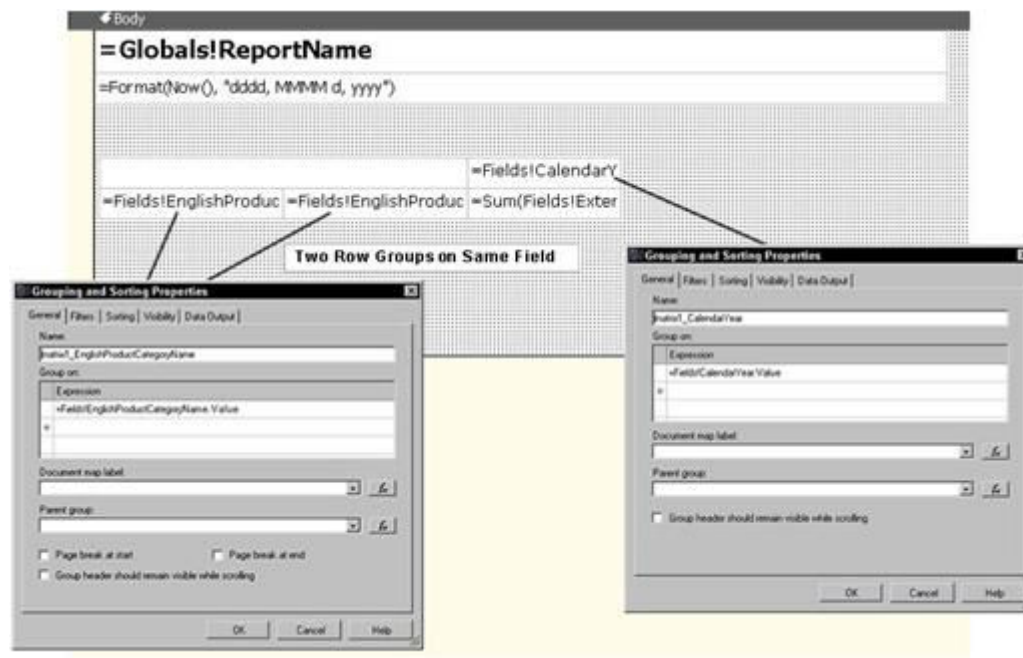
Green Bar - Product Sales by Year by Category List
Sunday, August 21, 2005

Year	Category	Product	Order Quantity	Extended Amount
2001	Accessories	Sport-100 Helmet, Black	331	\$6,681.73
2001	Accessories	Sport-100 Helmet, Blue	353	\$7,118.43
2001	Accessories	Sport-100 Helmet, Red	319	\$6,439.49
2001	Bikes	Mountain-100 Black, 38	312	\$630,178.13
2001	Bikes	Mountain-100 Black, 42	297	\$600,613.22
2001	Bikes	Mountain-100 Black, 44	315	\$636,320.61
2001	Bikes	Mountain-100 Black, 48	273	\$552,823.36
2001	Bikes	Mountain-100 Silver, 38	289	\$589,558.27
2001	Bikes	Mountain-100 Silver, 42	263	\$535,566.42
2001	Bikes	Mountain-100 Silver, 44	268	\$545,086.40
2001	Bikes	Mountain-100 Silver, 48	225	\$458,998.65
2001	Bikes	Road-150 Red, 44	52	\$111,642.02
2001	Bikes	Road-150 Red, 48	52	\$111,642.02
2001	Bikes	Road-150 Red, 52	52	\$111,642.02
2001	Bikes	Road-150 Red, 56	178	\$382,159.24
2001	Bikes	Road-150 Red, 62	109	\$234,018.86
2001	Bikes	Road-450 Red, 44	157	\$137,342.66
2001	Bikes	Road-450 Red, 48	52	\$45,489.29
2001	Bikes	Road-450 Red, 52	352	\$307,927.49
2001	Bikes	Road-450 Red, 58	281	\$245,438.04
2001	Bikes	Road-450 Red, 60	161	\$140,841.83
2001	Bikes	Road-650 Black, 44	136	\$57,046.41
2001	Bikes	Road-650 Black, 48	63	\$26,425.91
2001	Bikes	Road-650 Black, 52	346	\$144,783.23
2001	Bikes	Road-650 Black, 58	263	\$110,317.69
2001	Bikes	Road-650 Black, 60	138	\$57,885.33
2001	Bikes	Road-650 Black, 62	64	\$26,845.37

Figure 7-3[See full-sized image](#)

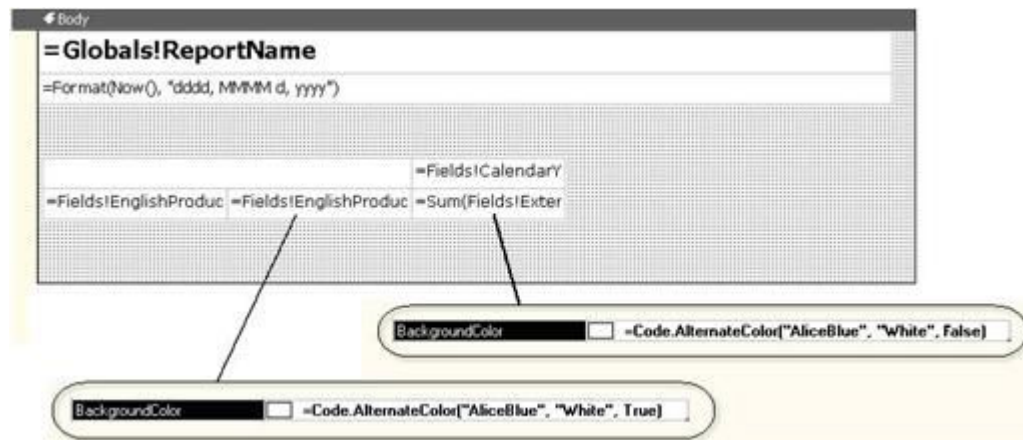
Creating a Greenbar Matrix

The pattern used for a matrix is very similar to a table. Since the matrix generates column cells dynamically, there is no way to specify a different expression for each column. If I wanted the row header to have an alternating background color, I could use the same technique as the table; toggling the odd/even flag explicitly on the row header textbox. But, if the pivot cell is to be the left-most item with an alternate background color on each row, this becomes more challenging. To work around this limitation, I define an extra row group on the same field expression as the previous group in the row hierarchy. Figure 7-4 shows the group definitions. The two row groups both use the same expression. This will cause the second group header textbox to be repeated with each row. I'm going to hide this cell when I'm done.

**Figure 7-4**

[See full-sized image](#)

Next, I set the BackgroundColor property using the same expression that I used in the table example. The second row header textbox sets the AlternateColor function to toggle the odd and even rows. Since the pivot cell (the textbox at the intersection of the row groups and column group) is repeated with the same background color for every column in a row, the second function argument is set to False.

**Figure 7-5**

[See full-sized image](#)

In Figure 7-6, I've reduced the width of this cell and I've also hidden it by setting the Visibility/Hidden property to True. You actually can't completely eliminate all evidence of a cell but you can make it very narrow. I've set the GridSpacing property to .03125 (1/32 of an inch) so I could make this column as narrow as possible.

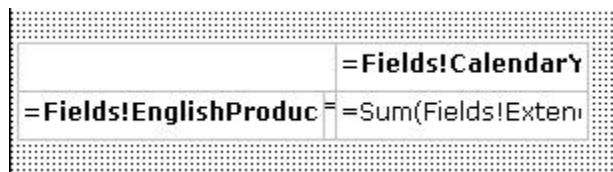


Figure 7-6

Figure 7-7 shows the end result. The utility cell causes a small gap between the row header and the remaining columns, alternate row colors are applied to the aggregate data cells.

Product Sales by Year by Category Matrix				
Sunday, August 21, 2005				
	2001	2002	2003	2004
Accessories	\$20,239.66	\$93,796.84	\$301,289.53	\$162,659.93
Bikes	\$7,399,579.79	\$20,134,190.51	\$25,827,841.05	\$13,435,410.84
Clothing	\$34,467.29	\$489,820.19	\$884,353.56	\$390,164.28
Components	\$615,474.98	\$3,611,041.24	\$5,486,723.33	\$2,091,051.85

Figure 7-7

[See full-sized image](#)

Multiple Criterion Report Filtering

Report design requirements may call for complex combinations of parameter values used to filter report data. Using Transact-SQL, you should be able to handle practically any advanced filtering criteria and filter the data before it reaches the report server. However, if you need to use report filtering to provide the same kinds of filtering support against data already cached by the dataset query, the report designer has some significant limitations in this area. For example, let's say that my report has two parameters for filtering product records; ProductCategory and PriceRange. In this simplified example, the parameter values for both of my parameter lists are the same as the parameter label values.

The ProductCategory parameter list values are as follows:

Parameter Value	Product Category Field Match
Bikes	Bikes
Components	Components
Clothing	Clothing
Accessories	Accessories
All Bike Related	Bikes and Components
All	All Categories

The PriceRange parameter list values are as follows:

Parameter Value	Price Range Field Match
Less than 50	< 50
50 to 100	>= 50 AND < 100
100 to 500	>= 100 AND < 500

500 and Higher	>= 500
All	All Prices

Contending with the various combinations of these and other parameter values in the confines of the report designer's filtering user interface would be very difficult to do. The most flexible method is to write a separate Visual Basic function to handle the matching logic for each parameter and field combination. This code is called for each row. The function returns a value to be matched with a field in the row. If the values match, the row is returned. The following custom code is added to the report on the Code tab of the Report Properties dialog:

```
Function MatchProductCategory(ParamValue As String, FieldValue As String)
    As String
        Select Case ParamValue
            Case "Bikes", "Components", "Clothing", "Accessories"
                Return ParamValue
            Case "All Bike Related"
                If FieldValue = "Bikes" Or FieldValue = "Components" Then
                    Return FieldValue
                End If
            Case "All"
                Return FieldValue
        End Select
    End Function
Function MatchPriceRange(ParamValue As String, FieldValue As String) As
    Decimal
        Select Case ParamValue
            Case "Less than 50"
                If FieldValue < 50 Then Return FieldValue
            Case "50 to 100"
                If FieldValue >= 50 And FieldValue < 100 Then Return FieldValue
            Case "100 to 500"
                If FieldValue >= 100 And FieldValue < 500 Then Return FieldValue
            Case "500 and Higher"
                If FieldValue >= 500 Then Return FieldValue
            Case "All"
                Return FieldValue
        End Select
    End Function
```

Using the Filters tab on the Dataset properties dialog, execute each of the functions, matching its return value to the corresponding field. Figure 7-8 shows this dialog.

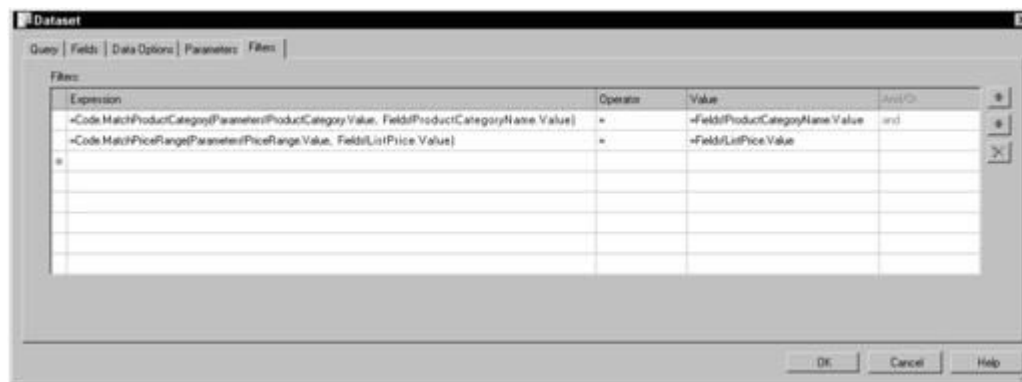


Figure 7-8

[See full-sized image](#)

This technique takes all of the complexity out of this simple dialog and puts it where it belongs: in program

code. That environment gives you the control needed to contend with practically any set of business rules.

TOP X and 'Other' Chart

Many standard chart formats have a limited capacity to effectively display more than a few series values. This can be remedied by capping the number of grouped values to a specific number of top values. However, a top 10 report may not accurately represent the entire data population. In addition to the top ten slices, an additional slice represents everything else. This can be effective then the top ten represent a significant portion of all values. Since each of the standard color palettes include 16 colors, at most make sense to present the top 15 + 1 "other" slice aggregating the remaining rows.

What you'll need:

- A custom query expression that combines two related result sets:
 - A specified number of top-rated values
 - An aggregate row consisting of all remaining rows
- A chart item used to present the combined result set

Figure 7-19 shows the finished report so you can see the concept before we take a look at the design technique. Note that there are eleven slices, ten for the top 10 selling products and one for the sum of all remaining products.

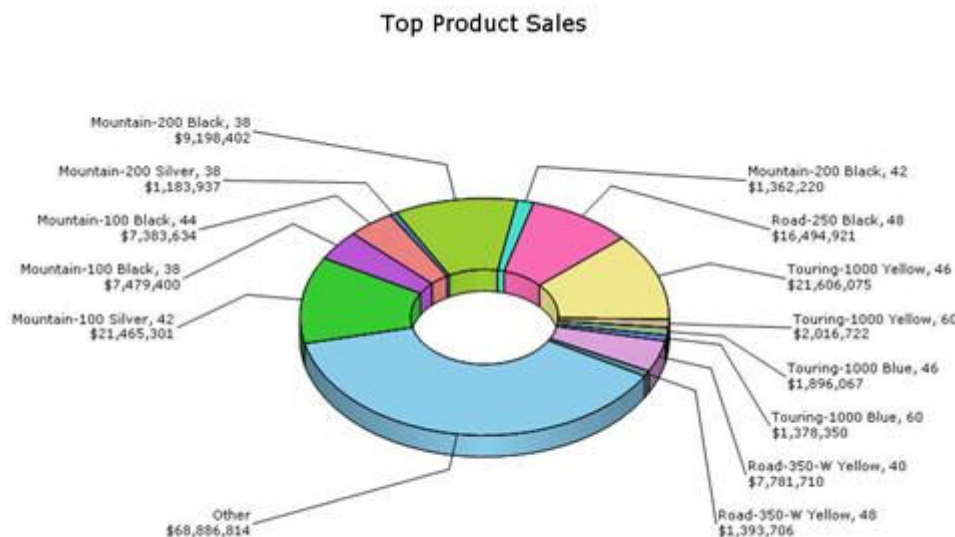


Figure 7-19

[See full-sized image](#)

This technique doesn't require any special settings in the chart itself. The magic is all in the dataset. This requires a little advanced SQL work. The following Transact-SQL query defines two different result sets, one for the top ten rows and then other for everything else. To return the non-top ten values, the top ten query is using as a subquery to exclude this selection for the entire result. The two results are combined using the SQL UNION statement.

```
-- Top 10 and Others:
SELECT Top10.ProductKey, Top10.ProductName
      , Top10.AmountSum
FROM (
  -- Top 10:
```



```

        SELECT TOP 10
            DimProduct.ProductKey
            , DimProduct.EnglishProductName AS ProductName
            , SUM(FactResellerSales.ExtendedAmount) AS AmountSum
FROM    DimProduct INNER JOIN FactResellerSales
        ON DimProduct.ProductKey = FactResellerSales.ProductKey
        INNER JOIN DimTime
        ON FactResellerSales.OrderDateKey = DimTime.TimeKey
        GROUP BY DimProduct.ProductKey, DimProduct.EnglishProductName
        ORDER BY SUM(FactResellerSales.ExtendedAmount) DESC
    ) AS Top10
UNION
SELECT Other.ProductKey, Other.ProductName, Other.AmountSum
FROM (
    -- Others excluding the top 10:
    SELECT TOP 100 PERCENT
        -1 AS ProductKey, 'Other' AS ProductName
        , SUM(FactResellerSales.ExtendedAmount) AS AmountSum
    FROM    DimProduct INNER JOIN FactResellerSales
        ON DimProduct.ProductKey = FactResellerSales.ProductKey
        INNER JOIN DimTime
        ON FactResellerSales.OrderDateKey = DimTime.TimeKey
    WHERE   DimProduct.ProductKey NOT IN
        (
            SELECT TOP 10 DimProduct.ProductKey
            FROM    DimProduct INNER JOIN FactResellerSales
                ON DimProduct.ProductKey =
                    FactResellerSales.ProductKey
                INNER JOIN DimTime ON
                    FactResellerSales.OrderDateKey = DimTime.TimeKey
            GROUP BY DimProduct.ProductKey
            ORDER BY SUM(FactResellerSales.ExtendedAmount) DESC
        )
    ORDER BY SUM(FactResellerSales.ExtendedAmount) DESC
) AS Other

```

You should be mindful that this query must reselect data in the same tables multiple times and may not perform well with large tables. This solution can also be achieved using an upgraded version of the chart item available from Dundas Software. Dundas Charts for Reporting Services includes a similar feature that doesn't require complex query expressions.

Dynamic Images: Scales and Gauges

Images can be made to display different content under different conditions. Using expressions, the image report item may be used with a series of images to show progress gauges or indicators.

What you'll need:

- A series of gauge or scale images representing progressive values
- An image item used to display one of the images
- An expression used to translate integer values to the name of a corresponding image

In a later example, I'll show you how to use the Dundas Gauges component to embed dynamically-rendered graphic content into a report. Using that technique, gauge pointers will show you an exact value, rather than one-in-ten shown here.

The Chart item outputs a static image that is embedded in the rendered report. In cases where you would like to use graphical output that a chart doesn't provide, you may use predefined graphics. The technique I'll demonstrate uses eleven image files which represent a gauge with different values. Based on a data value, the image is replaced with a corresponding graphic. Images may be obtained from a database, from external files or may be embedded within the report definition. Embedded images are convenient because they don't require special security or deployment considerations. However, embedded images should be used with small image files to keep the report definition file size manageable.

The sales quota data in the AdventureWorksDW database didn't match-up very well to the reseller sales totals so I had to modify the quota values to get this demonstration to work. If you want to duplicate this query, I suggest that you backup the database and then modify some of the SalesAmountQuota values in the FactSalesQuota table so the results are more realistic.

Figure 7-20 shows eleven gauge graphic files I've created using Dundas Guages. Each is saved in a folder as a PNG file. The file names also contain their value. This makes it easier to use and expression to derive the correct file name.

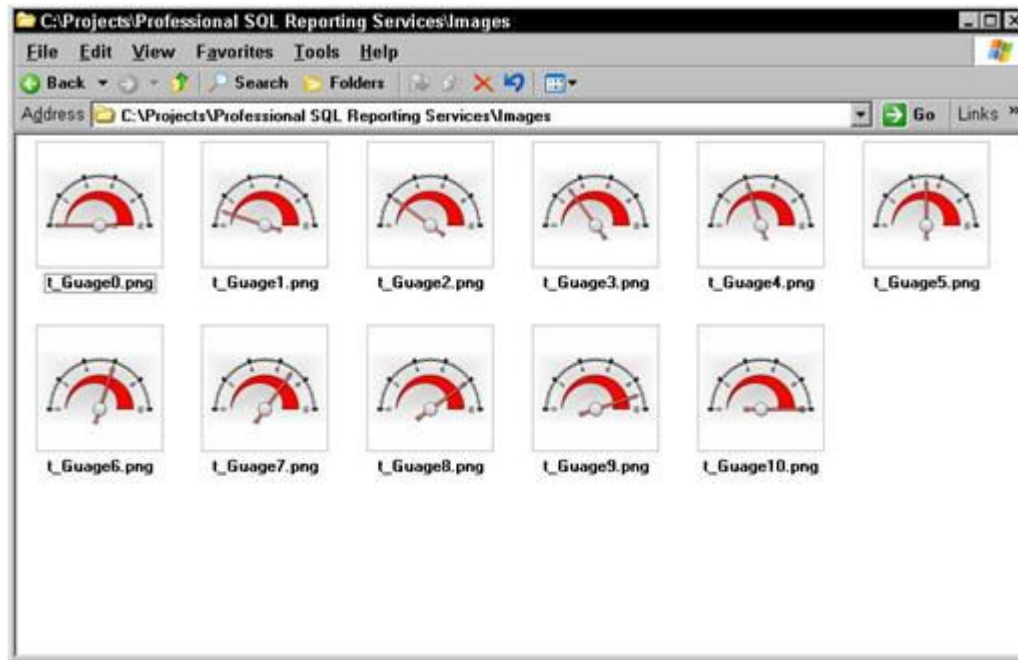


Figure 7-20

[See full-sized image](#)

My dataset query, shown in Figure 7-21, returns comparative quota and actual sales values grouped by year, quarter, and employee.

Sales Quota Gauges.rdl [Design]

Dataset: DataSet1 Command type: Text

```

SELECT FactSalesQuota.CalendarYear, FactSalesQuota.CalendarQuarter, DimEmployee.LastName, DimEmployee.FirstName,
(MIN(FactSalesQuota.SalesAmountQuota) AS SalesAmountQuota, SUM(FactResellerSales.ExtendedAmount) AS ExtendedAmountSum,
SUM(FactResellerSales.ExtendedAmount)/(MIN(FactSalesQuota.SalesAmountQuota) AS PercentOfQuota)
FROM FactSalesQuota INNER JOIN
DimEmployee ON FactSalesQuota.EmployeeKey = DimEmployee.EmployeeKey INNER JOIN
FactResellerSales ON DimEmployee.EmployeeKey = FactResellerSales.EmployeeKey INNER JOIN
DimTime ON FactSalesQuota.TimeKey = DimTime.TimeKey AND FactResellerSales.OrderDateKey = DimTime.TimeKey
GROUP BY FactSalesQuota.CalendarYear, FactSalesQuota.CalendarQuarter, DimEmployee.LastName, DimEmployee.FirstName
ORDER BY FactSalesQuota.CalendarYear, FactSalesQuota.CalendarQuarter, DimEmployee.LastName, DimEmployee.FirstName

```

CalendarYear	CalendarQuarter	LastName	FirstName	SalesAmountQuota	ExtendedAmountSum	PercentOfQuota
2001	3	Ansman-Wolfe	Pamela	165000.0000	127489.8043	0.7726
2001	3	Blythe	Michael	367000.0000	166820.1183	0.4545
2001	3	Campbell	David	226000.0000	257579.9398	1.1397
2001	3	Carson	Jillian	565000.0000	57326.7749	0.1014
2001	3	Ito	Shu	460000.0000	19708.3208	0.1298
2001	3	Mitchell	Linda	637000.0000	5475.9485	0.0085
2001	3	Peiter	Tavi	669000.0000	321164.9386	0.4800
2001	3	Saraiva	Jose	525000.0000	159407.8202	0.3036
2001	3	Vargas	Garrett	244000.0000	9109.1683	0.0373
2001	4	Ansman-Wolfe	Pamela	469000.0000	267976.6887	0.5713
2001	4	Blythe	Michael	556000.0000	221504.2517	0.3983

Figure 7-21

[See full-sized image](#)

For simplicity, and because the image files are small, I'm adding them to the report definition. By default, each image is assigned the same name as the source file. These can be modified if necessary. Figure 7-22 shows the Embedded Images dialog, opened from the Report menu.

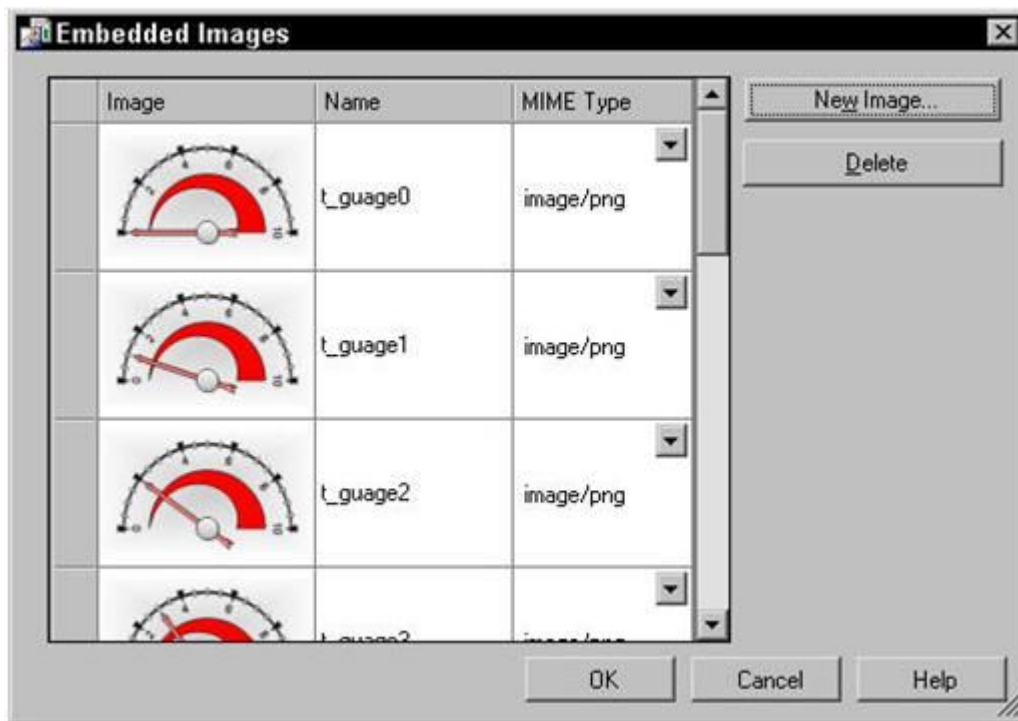


Figure 7-22

[See full-sized image](#)

The report contains a table item with an image in one of the detail cells. When the image item is dropped onto the report design surface, the Image Wizard is launched. The Wizard selections aren't important and will be replaced with properties I'll set manually.



Figure 7-23

[See full-sized image](#)

The Image items properties are set to use embedded images. I've set the image to FitProportional but this may be changed depending on the image size and characteristics. I typically use the Padding properties to provide margin space around the image cell. The image item's Value property will tie each row's sales quota value to a

corresponding gauge.

A calculated column called `PercentOfQuota` returns a float value as a percentage (0.0 to 1.0). By multiplying this value and converting it to an integer, using the Visual Basic `CInt()` function, these values now correspond to my image file names (0 to 10). This expression concatenates this integer with the rest of the file name. To prevent an overflow condition, If the percentage value is greater than 1 (100%), the expression always returns the highest value, 10.

```
= "t_guage" & IIF(Fields!PercentOfQuota.Value > 1, "10",  
CStr(CInt(Fields!PercentOfQuota.Value * 10)))
```

Here's the final result, shown in Figure 7-24 .

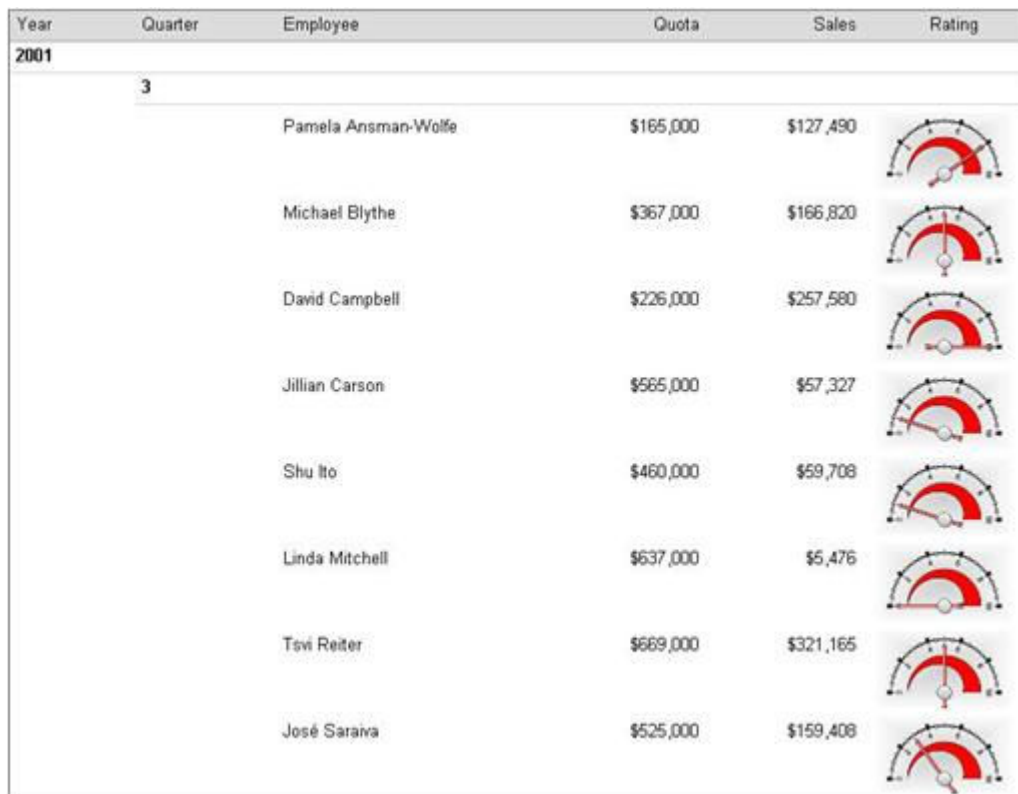


Figure 7-24

[See full-sized image](#)

I'll use the same basic principle in the next example to produce a business scorecard.

Creating a Business Scorecard

[Microsoft Business Scorecard Manager 2005](#) is a comprehensive scorecard and dashboard application that provides knowledge workers with deep contextual insight into business drivers and information is delivered through collaborative environments such as SharePoint Portal Server. BSM also plugs into features of several other Microsoft BI products such as SQL Server Analysis Services, Excel, Visio and Office Web Components. BSM is primarily designed for the non-programmer, power-user and requires little technical expertise to create dashboard-style reports.

For those with a little programming savvy, Reporting Services can also be used for creating simple dashboards and business scorecards.

This type of reporting scenario has quickly become a mainstay in enterprise business applications. Also known

as executive dashboards, business scorecards provide summary level progress and success status information for business leaders.

What you'll need:

- A query expression with data-based or calculated target, budget, variance and actual values
- A multi-group table with drill-down features
- Small images for use as progress indicators
- An expression used to translate KPI and target values to indicator images

Executive Dashboards

To understand and appreciate the value of this type of reporting interface, you need to walk in the shoes of corporate business leaders. A typical corporate officer deals with a lot of people and a variety of information in a day, and often needs to make immediate decisions based on this information. Moving from meeting to meeting, transaction-level details are too granular for most decisions. Business leaders need to know how the business is performing overall and whether there are areas of concern or notable success. I've sat in a lot of meetings with a General Manager or Director sitting on one side of the table and subject experts on another. The officer begins by saying "So, how are we doing?" The subject expert gives a lengthy presentation, stepping through PowerPoint slides, charts, graphs, and diagrams that depict trends and variances based on mountains of data. After the presentation, the officer concludes with the question; "So, how are we doing?" Scorecards and dashboards answer this all important question using succinct summary values and simple graphical, symbolic progress indicators.

Although simplification is a key concept, scorecards go beyond just keeping reports simple. Trends and success indicators should be clear and easy to understand but should provide an avenue to discover more detail and to view related trends and summaries. These objectives are easily achieved using drill-down and drill-through report features.

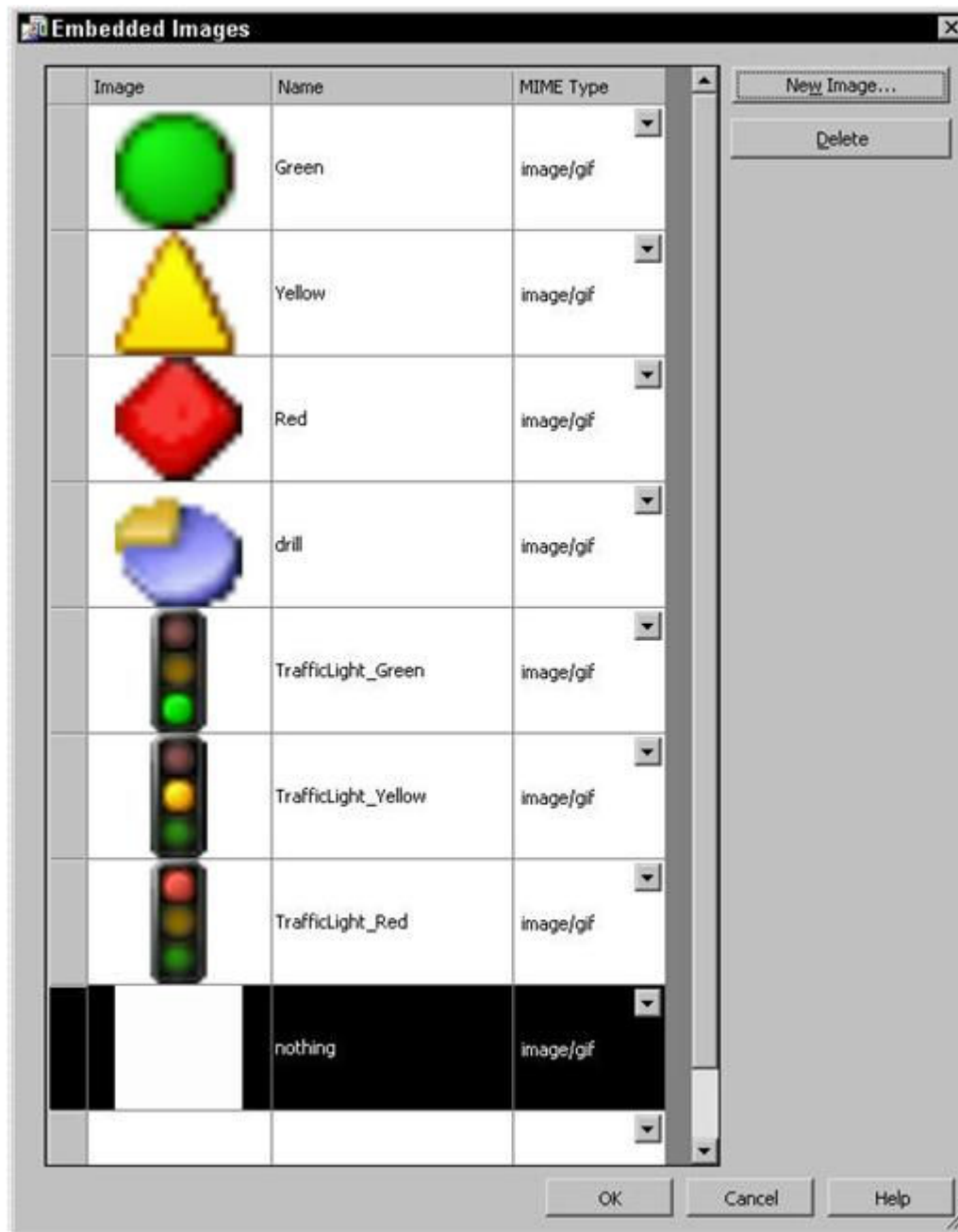
Targets and KPIs

These are the fundamental concepts behind business scorecards. For any given measurement, a target is simply an objective value. Targets are often data-driven values like a Budget, Quota, Baseline, or Goal. A KPI, or Key Progress Indicator, is a set of thresholds used to measure actual values with the target. KPIs may define banding indicators that to signify a range of variances like poor, acceptable and exceptional performance. KPI thresholds may be a single point, corresponding to the target, percentage, or fixed increment offsets with any number of indicator bands.

When considering longer-term trends, you may want to recognize the difference between a positive trend using a KPI and whether or not a value represents a successful outcome, as a KSI (Key Success Indicator.) For example, sales for a particular product may have been unprofitable since it went on the market. If sales are rising, a KPI would show positive sales growth while a KSI would indicate that the company is still in the red. We might simply define two targets, one to measure short-term progress and the other to measure overall profitability.

Indicators

Indicators are graphical icons, representing the state of an actual value with respect to a KPI band. On the scorecard, corresponding indicator icons might be red, yellow and green symbols. Indicators are typically common symbolic metaphors like traffic lights, colored shapes, progress bars, gauges and directional arrows. Figure 7-25 shows some common indicator graphics embedded in a sample report.

**Figure 7-25**

[See full-sized image](#)

Calculating Variance

Variance is the difference between an actual and target value. If time variances will be used extensively, the queries used to make these calculations can be very intensive. Aggregating and calculating sales totals, for example, for a given month over last month, quarter, or year can require some heavy-duty query processing (even with a modest number of detail rows.) Ideally this type of data should be stored in a data mart or data warehouse with pre-calculated variance values stored in the database. The AdventureworksDW database contains some pre-aggregated summary values but as you can see, even for this simple report with only year-over-year variances, the query is fairly complex.

```
SELECT
```

```

        ThisYearSales.SalesTerritoryRegion
    , ThisYearSales.SalesTerritoryKey
    , ThisYearSales.CalendarYear
    , ThisYearSales.LastName
    , ThisYearSales.FirstName
    , ThisYearSales.EmployeeName
    , SUM(ThisYearSales.ExtendedAmount) AS ExtendedAmountSum
    , SUM(ThisYearSales.SalesAmountQuota) AS SalesAmountQuotaSum
    , SUM(LastYearSales.ExtendedAmountSum) AS ExtendedAmountSumLastYear
FROM (
    SELECT
        DimSalesTerritory.SalesTerritoryRegion
    , DimSalesTerritory.SalesTerritoryKey
    , DimTime.CalendarYear
    , DimEmployee.LastName
    , DimEmployee.FirstName
    , DimEmployee.EmployeeKey
    , DimEmployee.FirstName + ' ' + DimEmployee.LastName AS EmployeeName
    , FactResellerSales.ExtendedAmount
    , FactSalesQuota.SalesAmountQuota
FROM DimEmployee INNER JOIN FactSalesQuota
    ON DimEmployee.EmployeeKey = FactSalesQuota.EmployeeKey
    INNER JOIN DimTime ON FactSalesQuota.TimeKey = DimTime.TimeKey
    INNER JOIN FactResellerSales
    ON DimEmployee.EmployeeKey = FactResellerSales.EmployeeKey
    AND DimTime.TimeKey = FactResellerSales.OrderDateKey
    INNER JOIN DimSalesTerritory
    ON DimSalesTerritory.SalesTerritoryKey =
        FactResellerSales.SalesTerritoryKey) AS ThisYearSales
    INNER JOIN
        ( SELECT
            FactResellerSales.EmployeeKey
        , DimTime.CalendarYear
        , DimSalesTerritory.SalesTerritoryKey
        , DimSalesTerritory.SalesTerritoryRegion
        , FactResellerSales.ExtendedAmount AS ExtendedAmountSum
        FROM FactResellerSales
        INNER JOIN DimTime
        ON FactResellerSales.OrderDateKey = DimTime.TimeKey
        INNER JOIN DimSalesTerritory
        ON DimSalesTerritory.SalesTerritoryKey =
            FactResellerSales.SalesTerritoryKey
        ) AS LastYearSales
    ON LastYearSales.CalendarYear = ThisYearSales.CalendarYear - 1
    AND ThisYearSales.EmployeeKey = LastYearSales.EmployeeKey
    AND ThisYearSales.SalesTerritoryKey = LastYearSales.SalesTerritoryKey
GROUP BY ThisYearSales.SalesTerritoryRegion, ThisYearSales.SalesTerritoryKey
    , ThisYearSales.CalendarYear, ThisYearSales.LastName, ThisYearSales.FirstName
    , ThisYearSales.EmployeeName
ORDER BY ThisYearSales.SalesTerritoryRegion, ThisYearSales.CalendarYear
    , ThisYearSales.LastName, ThisYearSales.FirstName

```

When running complex queries like this one, you may need to increase the default connection timeout setting on the data source. The default setting is 15 seconds, which may not be sufficient for this query on all hardware. In a production application with data volumes greater than the sample database, I would recommend testing query performance and possibly using an Analysis Services database with cubes and pre-calculated aggregates. To populate the data warehouse, you will use queries similar to this one and store the results for later retrieval.

Figure 7-26 shows a simple table with two groups; on the SalesTerritory and CalendarYear fields. This table is much like several previous examples. The detail row is hidden by default allowing for drill-down using the SalesTerritoryRegion textbox. Two more images will serve as indicators. These are based on expressions used to change the indicator image.

You will notice that the images have a white background even though I've used background colors to separate the rows. I've only done this to simplify this example. I have simply added the images to the cells in the table header. If you want to use transparent images over a colored or shaded background, you will need to add

rectangles to the header cells and then place images in the rectangles. This way, you can set the `BackColor` property for each rectangle and take advantage of the image transparency. The final example, shown in Figure 7-30, uses this technique to fill the background color behind the scorecard indicator images.




Territory Region	Employee	Quota	Tot. Sales	Yr Variance
Year				
 =Fields!SalesTerritoryRegion.Value		=Sum(Fields!SalesAmountQuota)	=Sum(Fields!ExtendedAmount)	 =1 - (Sum(Fields!ExtendedAmount) / Sum(Fields!ExtendedAmountSumLastYear.Value))
=Fields!CalendarYear	=Fields!EmployeeName.Value	=Fields!SalesAmountQuota	=Fields!ExtendedAmount	 =1 - (Fields!ExtendedAmount / Fields!ExtendedAmountSumLastYear.Value)
			Footer	

Figure 7-26

[See full-sized image](#)

Looking at the columns with text headers, the first column contains the `SalesTerritoryRegion` field in the first group header and the `CalendarYear` field in the detail row.

The second column contains the `EmployeeName` in the detail row.

The third text column is for the `SalesAmountQuota` field. The header uses the `SUM()` function to aggregate the details for the sales territory.

The fourth text column contains total sales values, using the `ExtendedAmount` field.

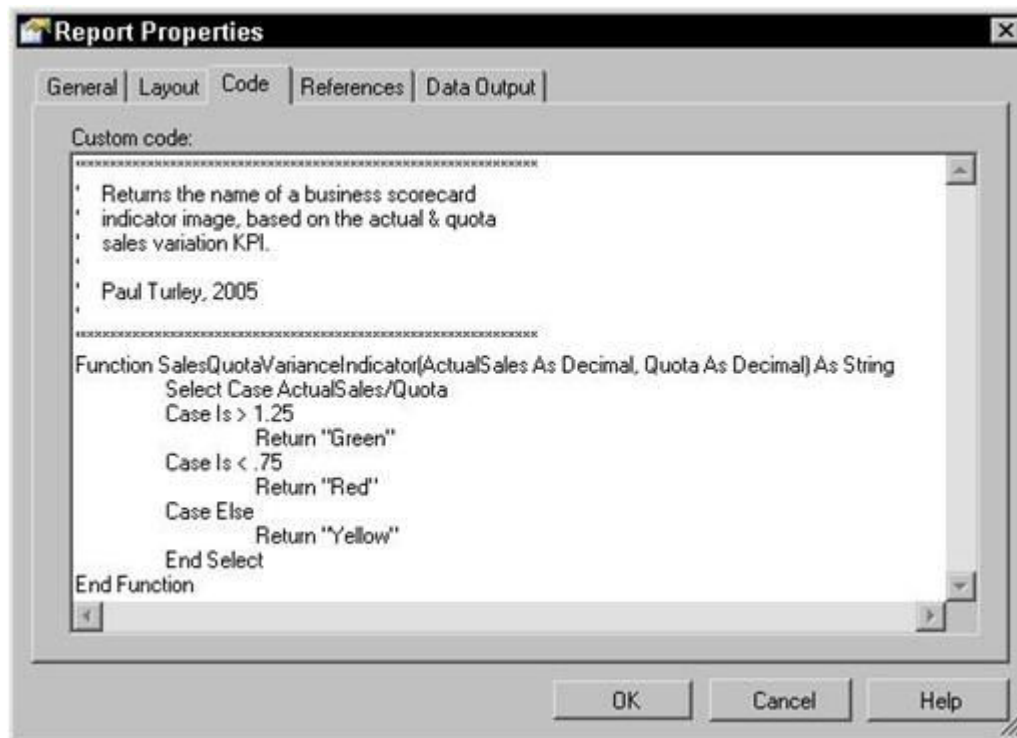
The last column of textboxes, labeled *Yr. Variance*, calculates the total sales amount annual variance. In the header row, the expression uses the `SUM()` function. In the detail row, the `SUM()` function is omitted.

Please ensure that the following code fits on one line, thanks

Note: The line has been split into multiple lines for readability. However, while trying it out on a system you must enter it as one line without breaks.

```
=1-(Sum(Fields!ExtendedAmountSumLastYear.Value)/
Sum(Fields!ExtendedAmountSum.Value))
```

The expression for the sales first set of indicators (the images column after total sales column) calls a Visual Basic function to apply the KPI threshold banding. Figure 7-27 shows this custom code.

**Figure 7-27**

[See full-sized image](#)

Since the image names for the green, yellow, and red indicators are Green, Yellow, and Red, these values are simply returned in the Value property of the image item using the following expression:

Note: The line has been split into multiple lines for readability. However, while trying it out on a system you must enter it as one line without breaks.

```
=Code.SalesQuotaVarianceIndicator(Sum(Fields!ExtendedAmountSum.Value),  
Sum(Fields!SalesAmountQuotaSum.Value))
```

For variety, I've resolved the second indicator column images using only an in-line expression, rather than using a custom function. This is the expression for the header row. The detail row expression is the same but excluding the SUM() function. As a rule, once I've decided to use custom code, I'll typically continue to use custom functions for all but the simplest expressions so I can keep business logic in one place.

Note: The line has been split into multiple lines for readability. However, while trying it out on a system you must enter it as one line without breaks.

```
=IIF(Sum(Fields!ExtendedAmountSum.Value) /  
Sum(Fields!ExtendedAmountSumLastYear.Value) < .8,  
"exclamation_small", "nothing")
```

This expression returns the name of an exclamation mark icon image when this year's sales amount is less than 80% of last year's. I created an image file called "nothing" which is a blank icon with a white background. Using this image effectively displays nothing in the image cell.

Synchronizing Charts and Related Report Items

One of the great advantages to the scorecard approach is that all the information is presented in a concise page. In order to make the best use of screen space, I can use a separate report item to show content related

to the item selected in the scorecard.

Figure 7-28 shows the table and chart items. When I select a sales territory, by clicking on the small pie chart icon in the first column, I want to see sales trend information in a column chart. I've placed a chart to the right of the scorecard and have configured it as a column chart. I've also simplified the chart by removing the legend.

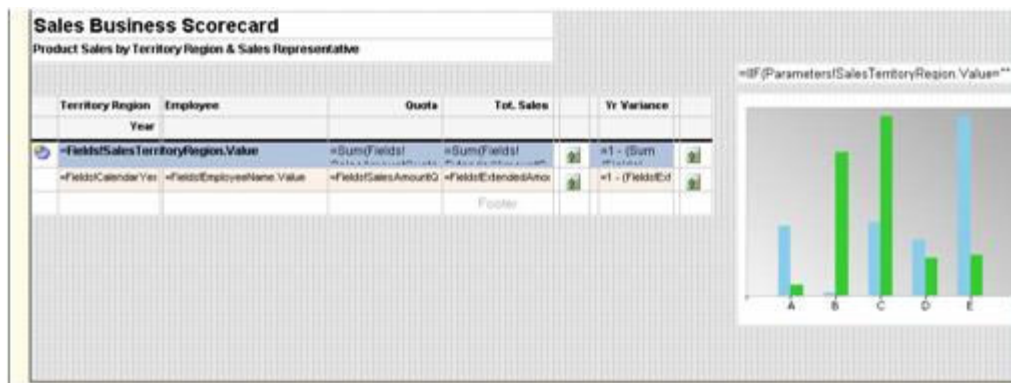


Figure 7-28

[See full-sized image](#)

The chart content is synchronized using a report parameter. The SalesTerritoryKey parameter is used to filter the dataset providing data to the chart. The SalesTerritoryregion parameter is used to provide a title value for the textbox above the chart. Figure 7-29 shows the Navigation properties for the pie icon used to synchronize the chart. Note that the Jump to report property is set to navigate back to this report, re-rendering the same report with the new parameter values.

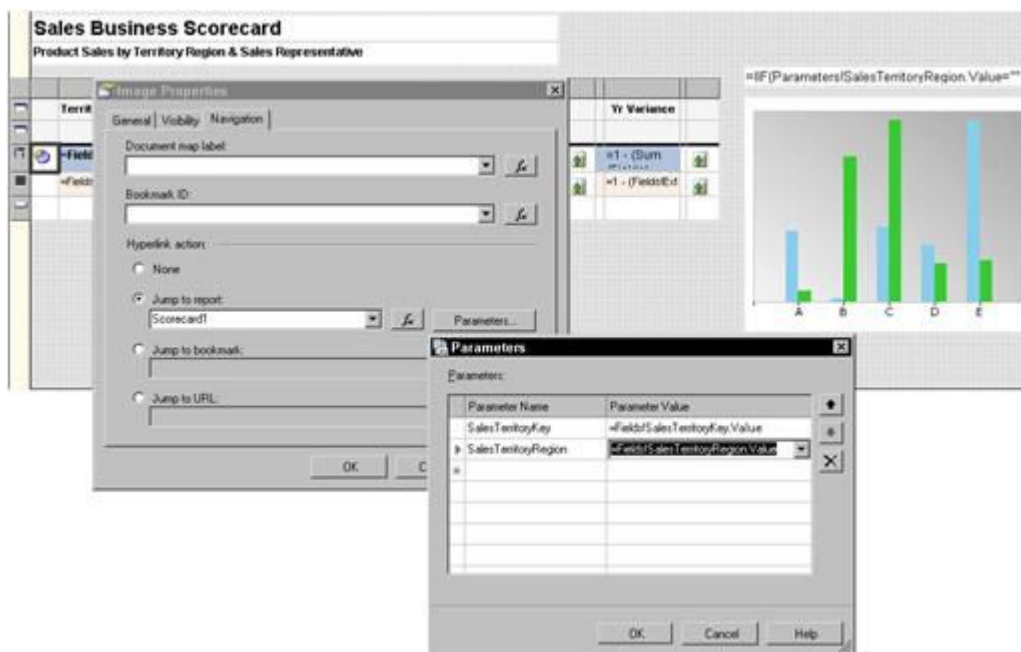


Figure 7-29

[See full-sized image](#)

Figure 7-30 shows the rendered report with some region sections drilled open. I've clicked the pie icon next to the Southwest region to synchronize the chart and view sales trend details for the southwest region. Again, note the background color fill behind the scorecard indicator images using the technique I mentioned earlier.

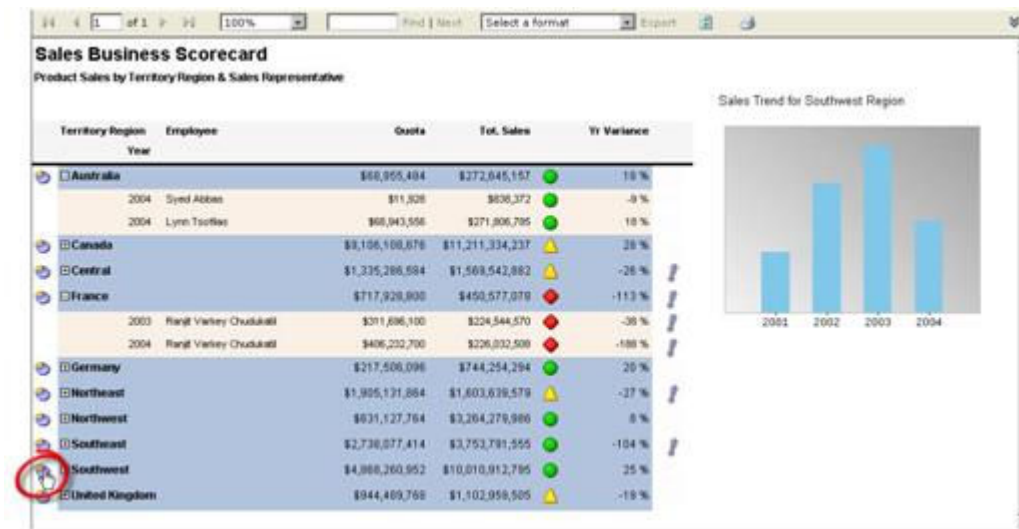


Figure 7-30

[See full-sized image](#)

One of the first things you'll probably notice is that the quota and total sales values are not very close at all. These values have actually changed in different editions of the AdventureWorksDW sample database as it progressed through beta testing cycles. For this report example to be more realistic, you can manually change some of the SalesAmountQuota values in the FactSalesQuota table. The completed sample code on the book's support site, at P2P.WROX.COM, contains a script to update all of the quota values.

Using Charts & Data Ranges in Detail Rows

Reporting Services doesn't support the placement of charts and some other embedded report items in table detail rows. Working around this limitation is very easy and simply requires that a group be defined at the detail row level. You can either delete the detail row or resize it. Detail-level items are placed in the header row rather than the detail row. The following two sparkline chart examples will demonstrate this technique.

What you'll need:

- A table presenting a columnar set of values
- A chart item embedded into a table group header cell

Creating Sparklines

Edward Tufte, one of the most recognized experts on the subject of data visualization presents the idea of sparklines. These are simple, word-sized graphics that are an alternative to large, busy charts used to communicate a simple trend or series of measurements. In order to be meaningful, sometimes charts need to have annotated gridlines, point labels and legends. However, some charts can effectively serve their purpose without the use of supporting text labels. To illustrate observations like "sales are improving," "a product is profitable," or that a trend is cyclical, a simple trend chart needs little or no labeling. Sparklines are best used when embedded in text or other report formats.

What you'll need:

- A query expression used to return trend data
- A small, simplified chart item
- A table item to display master rows

Column and line charts are best suited for this type of presentation. In the first of two examples, I'll use a column chart to show sports games scores for a team throughout the season. The first examples uses data I had on-hand from a project. The second example will use sample data from the AdventureWorks database.

Team Standings

The purpose of the chart, shown in figure 7-31, is to quantify the team's relative position and win/loss trend, rather than to show specific scores. For this I use a no-frills column chart. The dataset returns a team name, game number and score for each team. The column value will represent the number of points that won or lost the game. For example, a team that wins with a score of 5 to 3 would have a winning score of 2. If the team losses 3 to 4, their score would be -1.

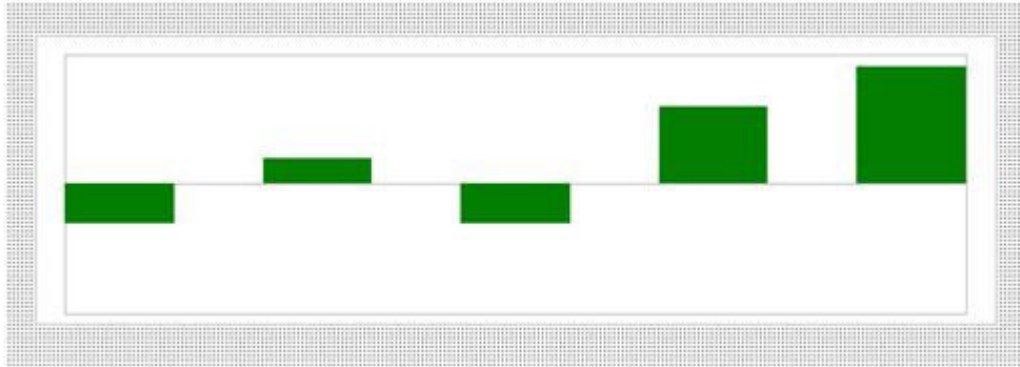


Figure 7-31

[See full-sized image](#)

Figure 7-32 shows the category group. This plots columns along the X axis, one for each game.



Expression
=Fields!Game.Value
*

Figure 7-32

The value group uses the calculated score to plot the column above or below the line to indicate a win or loss.

Figure 7-33 shows this calculation in the Edit Chart Value dialog.

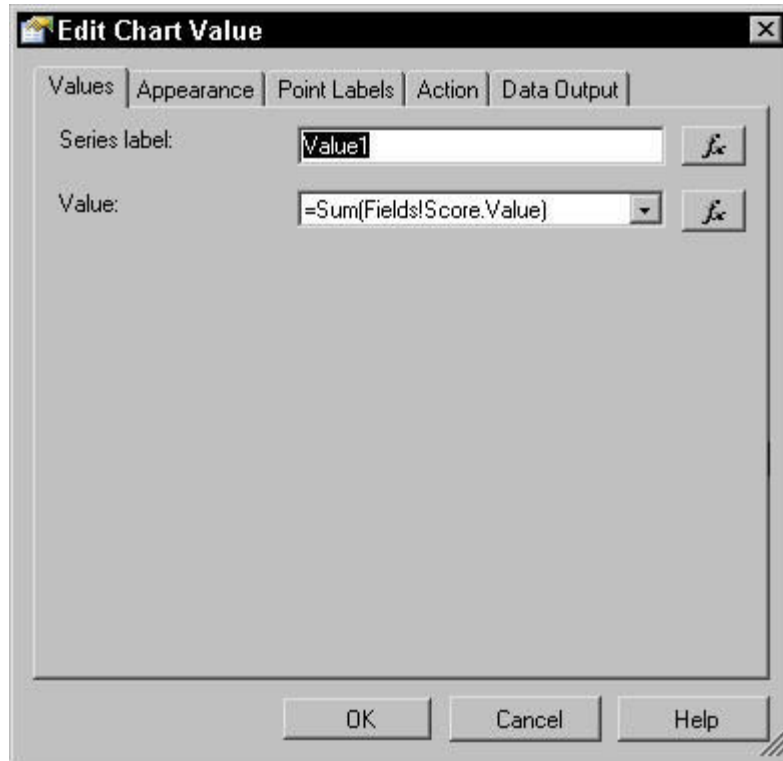


Figure 7-33

Figures 7-34 and 7-35 show the X axis and Y axis configuration. The X axis grid lines, tick marks and labels are removed to keep the chart simple.

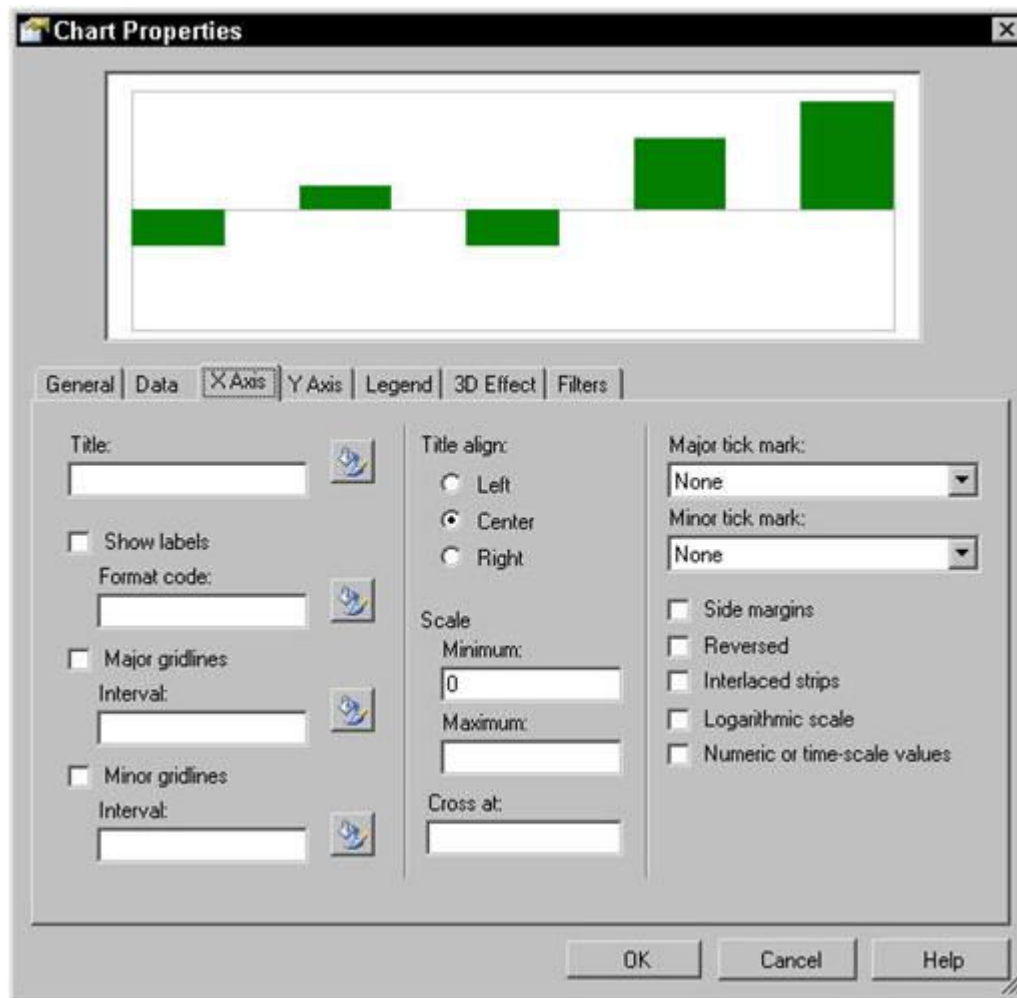
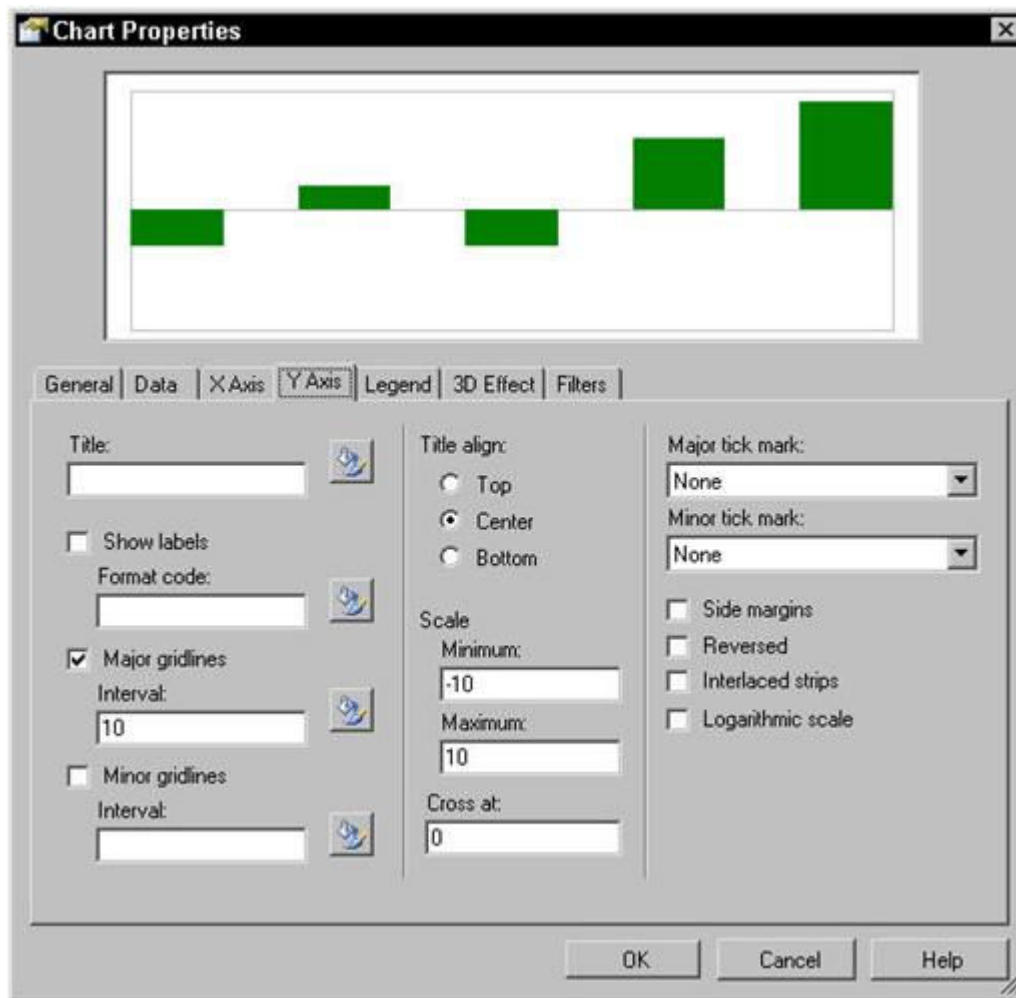


Figure 7-34
[See full-sized image](#)

On the Y axis, lines are displayed at the zero cross point, dividing wins and losses at 10 and -10 to render all chart instances at the same scale.

**Figure 7-35**

[See full-sized image](#)

Since the chart will be quite small, border lines should be thin and subtle. Click the corresponding Style button to show the Style Properties dialog, shown in figure 7-36. I'm using Silver color (50% gray) and .5 point lines. These lines may not be displayed in the designer but should render correctly when the report is previewed or deployed.

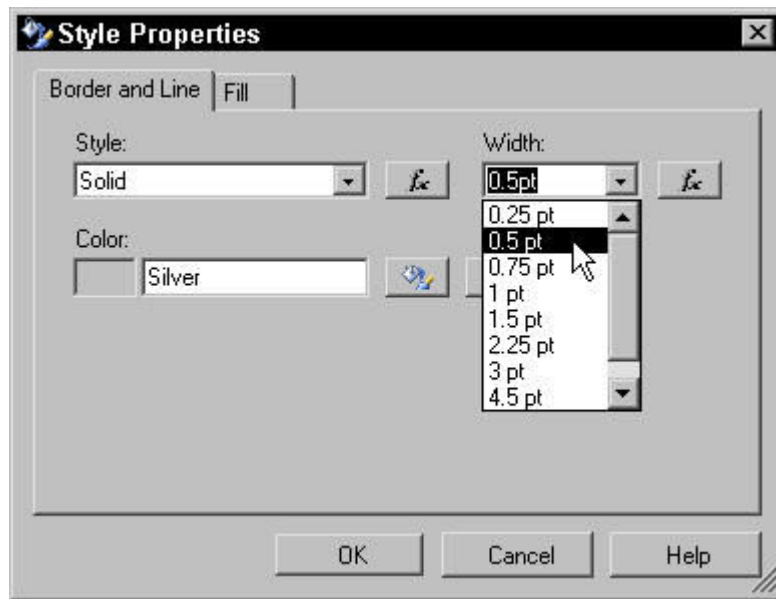


Figure 7-36

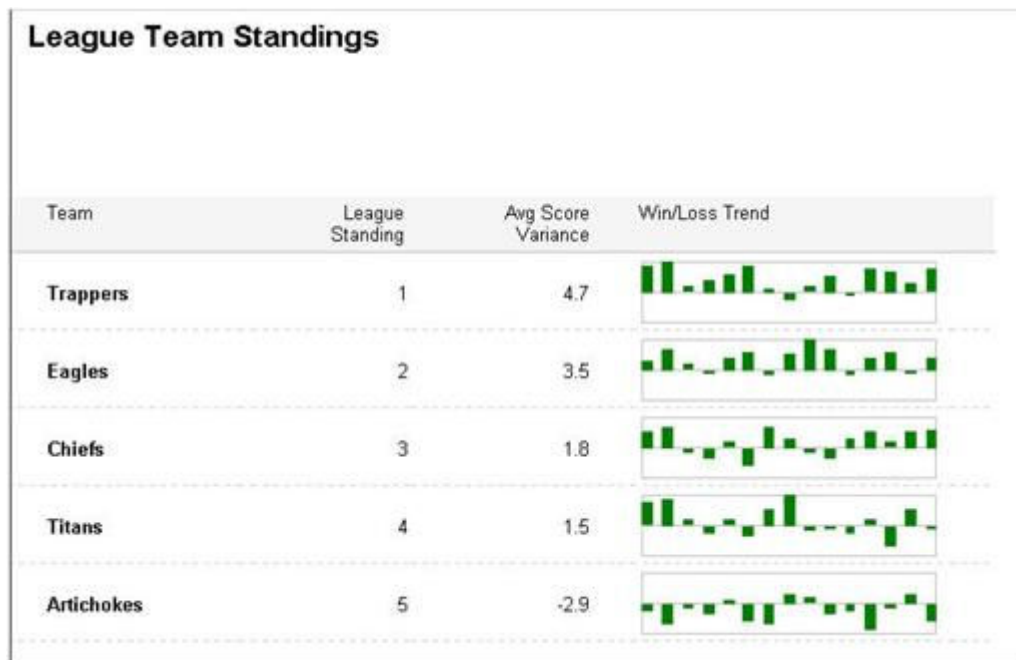
The chart is placed in a group header row within a table grouped on the team field. This will serve as the detail row since a chart can't reside in a detail row. Figure 7-37 shows the finished report in design view.



Figure 7-37

[See full-sized image](#)

The finished report is shown in Figure 7-38 . For each team, their calculated league standing, average score variance, and the win/loss trend sparkline chart is displayed in table rows.

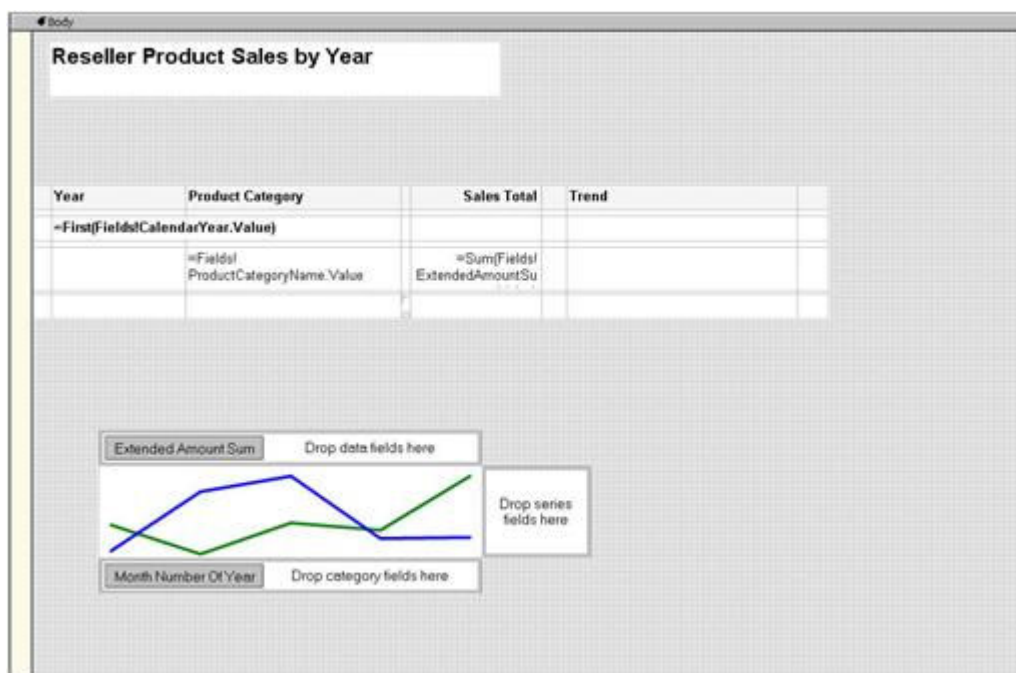
**Figure 7-38**

[See full-sized image](#)

Sales Trends

This example shows product category sales on each row and sales by year in an associated line chart, plotting sales totals by month. This report's dataset is based on a simple query that returns aggregated sales by year, month, and then by product category.

In Figure 7-39, I've added and setup the table and chart in separate areas of the report body. They're both bound to the same dataset. After the table is configured, I'll add it to the table. Like the previous example, a group header row is used in-place of the detail row.

**Figure 7-39**

[See full-sized image](#)

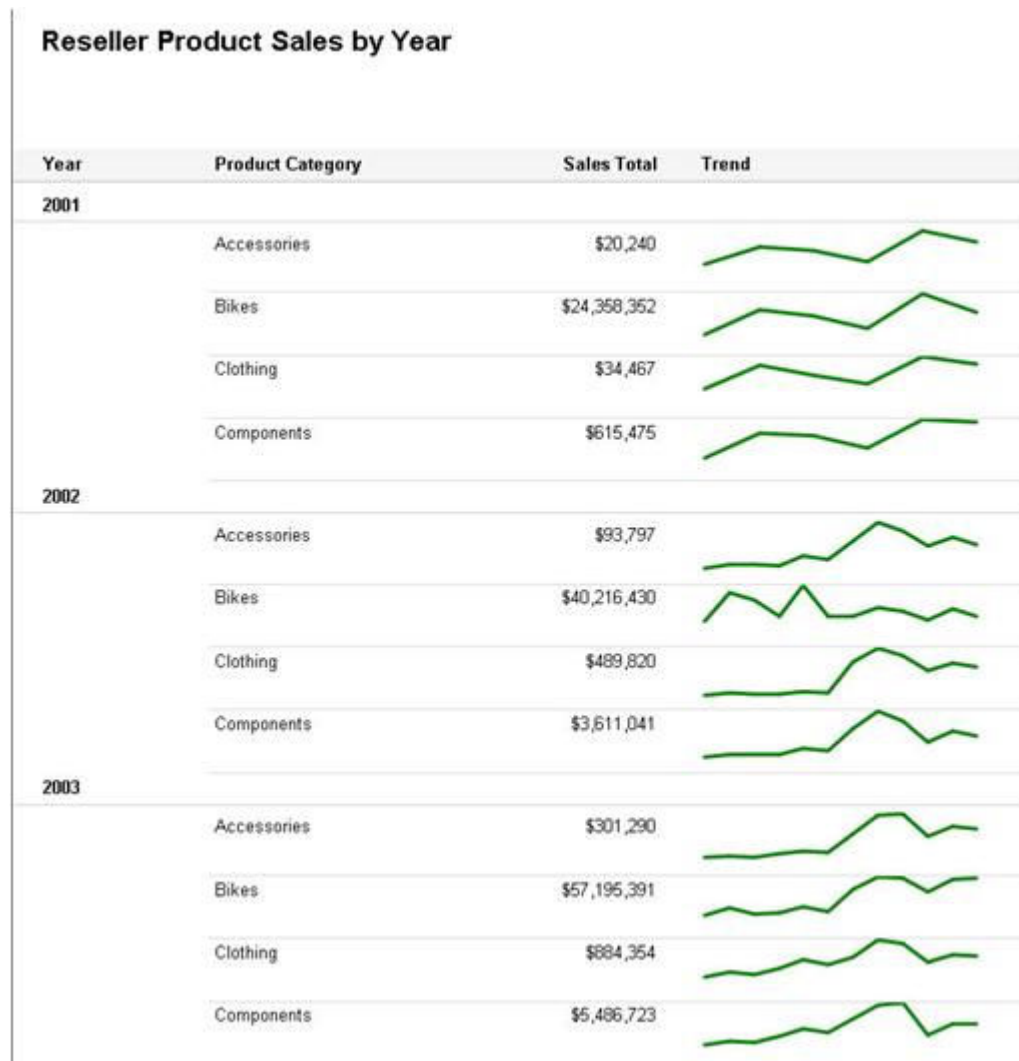
I've configured the chart with no gridlines or labels at all. Its purpose is to show relative sales trends, not specific values. In a production reporting solution, I might create separate chart report, similar to the sparkline chart but with more detail. Figure 7-40 shows this report in design view.



Figure 7-40

[See full-sized image](#)

Finally, Figure 7-41 shows the finished report. The trend line shows sales total over the course of the year. Whether data point represented days, weeks or months, the effect would be the same.

**Figure 7-41**[See full-sized image](#)

Hitachi Consulting has business and IT consulting offices throughout the world. Paul works for Hitachi's Pacific Northwest Business Intelligence practice in Seattle and may be contacted at pturley@hitachiconsulting.com.

[↑ Top of page](#)[Manage Your Profile](#)

© 2008 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#) | [Contact Us](#)

Microsoft