****

Analysis and Reporting Services, Design and Implementation: A SQL Server 2008 Technical Case Study

SQL Server Technical Article

**Writers:** Nicholas Dritsas, Nopparat Leelavanich

**Contributors:** Artich Asawachaiyawong, Sairung Inlert

**Technical** **Reviewers**: Carl Rabeler, Robert Bruckner

**Published:** June 2010

**Applies to:** SQL Server 2008, SQL Server 2008 Analysis Services, and SQL Server 2008 Reporting Services

**Summary:**

Providing strong customer service has helped this customer become the largest commercial bank in Thailand. Always looking for new ways to enhance the services it provides, the bank wanted to study its customer data to help it better determine customer needs, but a reporting system based on manual report development in spreadsheets by using mainframe data (provided in text files) proved cumbersome. The company created a business intelligence (BI) data warehouse using the Microsoft Application Platform, including Microsoft SQL Server 2008 Enterprise (64-bit) database software. Reports are now generated automatically using SQL Server 2008 Reporting Services. Multidimensional data cubes for support of BI analytics were created using SQL Server 2008 Analysis Services. The customer has gained a better view into its operations and deeper customer knowledge since deploying its BI solution.

**Table of Contents**

[1 Introduction 5](#_Toc264310779)

[2 Design Summary 6](#_Toc264310780)

[2.1 Overall Requirements 6](#_Toc264310781)

[2.2 Overall Solutions 6](#_Toc264310782)

[3 Integrating Reporting Services Reports and Analysis Services Data 8](#_Toc264310783)

[3.1 Server Aggregate Functions 8](#_Toc264310784)

[3.2 MDX Query and Tablix Behavior for the Main Dataset 11](#_Toc264310785)

[3.3 Utilizing Report Parameters and the Dimension Caption Property 15](#_Toc264310786)

[3.4 Example Case Study: Sales Summary Reports 16](#_Toc264310787)

[3.4.1 Business Requirement 17](#_Toc264310788)

[3.4.2 Report Design 17](#_Toc264310789)

[4 Implementing Dynamic Balance Scorecards with Analysis Services and Reporting Services 22](#_Toc264310790)

[4.1 Business Requirement 22](#_Toc264310791)

[4.2 Solution Design 24](#_Toc264310792)

[4.3 OLAP Design 24](#_Toc264310793)

[4.4 Report Design 27](#_Toc264310794)

[4.4.1 Report Parameters 27](#_Toc264310795)

[4.4.2 Dataset 27](#_Toc264310796)

[4.4.3 Tablix 28](#_Toc264310797)

[4.4.4 Result 29](#_Toc264310798)

[4.5 Additional Tips 30](#_Toc264310799)

[4.5.1 Improvement Areas 30](#_Toc264310800)

[4.5.2 Ranking 30](#_Toc264310801)

[4.5.3 Gauges 32](#_Toc264310802)

[5 Implementing Dynamic Reports 33](#_Toc264310803)

[5.1 OLAP Design 35](#_Toc264310804)

[5.1.1 Time Intelligence 35](#_Toc264310805)

[5.2 Report Design 36](#_Toc264310806)

[5.2.1 Report Parameters 36](#_Toc264310807)

[5.2.2 Dataset 37](#_Toc264310808)

[5.2.3 Tablix 39](#_Toc264310809)

[5.2.4 Graph 39](#_Toc264310810)

[5.3 Additional Tips 40](#_Toc264310811)

[5.3.1 Improvement Areas 40](#_Toc264310812)

[6 Implementing Dynamic Security 41](#_Toc264310813)

[6.1 Business Requirement 41](#_Toc264310814)

[6.2 Dynamic Security Solution 42](#_Toc264310815)

[6.2.1 User Security Table 42](#_Toc264310816)

[6.2.2 Stored Procedure 43](#_Toc264310817)

[6.2.3 File.dll (.NET Code) 43](#_Toc264310818)

[6.2.4 OLAP Database Role 44](#_Toc264310819)

[7 REFERENCES 45](#_Toc264310820)

[8 CONCLUSION 46](#_Toc264310821)

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, Excel, SharePoint, and SQL Server are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

1. Introduction

One of the largest companies in Thailand came to us wanting to implement a business intelligence (BI) solution to assist their operations unit with information so that they could to perform branch analysis, customer analysis, and customer segmentation. In addition to serving the business needs of the in-scope business unit, Microsoft wanted to provide them with a strategic and tactical solution that would give them a strong edge over their competitors.

In this engagement, the customer wanted to implement a business intelligence solution on the Microsoft® SQL Server® 2008 BI suite, including SQL Server Analysis Services and SQL Server Reporting Services, to ensure the proposed solution could meet their customer requirements of the Balance Scorecard (a dynamically updated dashboard of Key Performance Indicators (KPIs)), dynamic reports, and dynamic security.

This document focuses on the technical concepts and techniques we used to implement reports with SQL Server 2008 Reporting Services, using an Analysis Services cube as the data source. It also covers the techniques we used within Analysis Services to meet the business requirements of the customer.

1. Design Summary

A key part of any design phase is reviewing your customer’s requirements and talking with them about possible solutions you can provide. This section lists the particulars of this customer’s organization, from user types to security goals. It also describes the customized solution we were able to offer using SQL Server 2008.

* 1. Overall Requirements

The following is a list of business requirements that we used in designing the business intelligence solution:

* The customer needs a business intelligence solution that provides information for every business unit.
* There are two types of users: analysts and general users. Analysts need a tool that enables them to analyze data in their organization using ad-hoc analysis reports. General users need general reports.
* Two types of reports are required: the daily updated dashboard and the Balance Scorecard, which provides up-to-date data for most users based on their role and responsibility.
* There are more than a thousand general users in this solution. Each user can access only reports to which he or she is granted access, depending on factors such as organizational level and department.
* Although all users access the same report template, each user can view only data for which he or she has permissions. For example, users from Branch A can see only Branch A data in the sale summary report, while users from Region 1 can view data from all branches in Region 1.
* There are more than a thousand data access permission groups.
* To support the number of general users from thousands of offices all over the country, the front-end tool must be web-based.
* The solution should enable all users to create and adjust the reports by themselves with minimal input from IT.
* Reports should be grouped to reduce both the number of reports and the effort required to create and maintain them.
  1. Overall Solutions

Microsoft designed and implemented a business intelligence solution using features of SQL Server 2008, including:

* A data mart built on SQL Server 2008.
* A cube built on top of the data mart with SQL Server 2008 Analysis Services. The cube is the main data source for this solution.
* The use of two front-end tools, Microsoft Office Excel® 2007 and SQL Server 2008 Reporting Services, to address the needs of the two different types of users. Analysts use Excel to browse Analysis Services cube, and general users access web-based reports via Report Viewer in SQL Server 2008 Reporting Services.
* An extraction, transformation, and loading (ETL) framework, including ETL packages in SQL Server 2008 Integration Services, to integrate data from transactional databases. To enable selected business users to adjust and maintain data by themselves, we developed an Excel template that could be used as one of the ETL sources.
* Data retrieval from either Analysis Services or the relational database. Most of the Reporting Services reports retrieve data from SQL Server 2008 Analysis Services. Others retrieve data from a relational database in SQL Server 2008.
* Dynamic security for the Analysis Services cube, to support over a thousand data access groups. We applied this solution to reports that retrieved data from the relational database.

1. Integrating Reporting Services Reports and Analysis Services Data

Microsoft SQL Server 2008 Reporting Services provides capabilities to help developers implement rich and complex reports from Microsoft SQL Server Analysis Services. For this customer, we used powerful features such as the server aggregation function, the Multidimensional Expressions (MDX) query language, the Utilize Report parameter, and the Dimension Caption property to develop a solution that met our customer’s need to retrieve Analysis Services data through Reporting Services reports. This section discusses the techniques we used and provides some background about how the techniques work.

This section also includes information about steps you can take to ensure that your Analysis Services data is displayed correctly in Reporting Services.

Finally, this section offers a case study that shows, step by step, how we implemented these features for this customer.

* 1. Server Aggregate Functions

Reporting Services provides most of the basic report aggregation used to calculate or summarize data on the retrieved data sets. If a relational database is used as a data source, you query only detail rows from the data source into report data sets. If you want to display summary data in row groups or column groups, you select report aggregate functions to calculate them. The report aggregate functions include **Avg**, **Count**, **CountDistinct**, **CountRows**, **First**, **Last**, **Max**, **Min**, **STDev**, **STDevP**, **Sum**, **Var**, **VarP**, and **RunningValue**. In this case, Reporting Services is responsible for summarizing the data. This action is performed during report processing time.

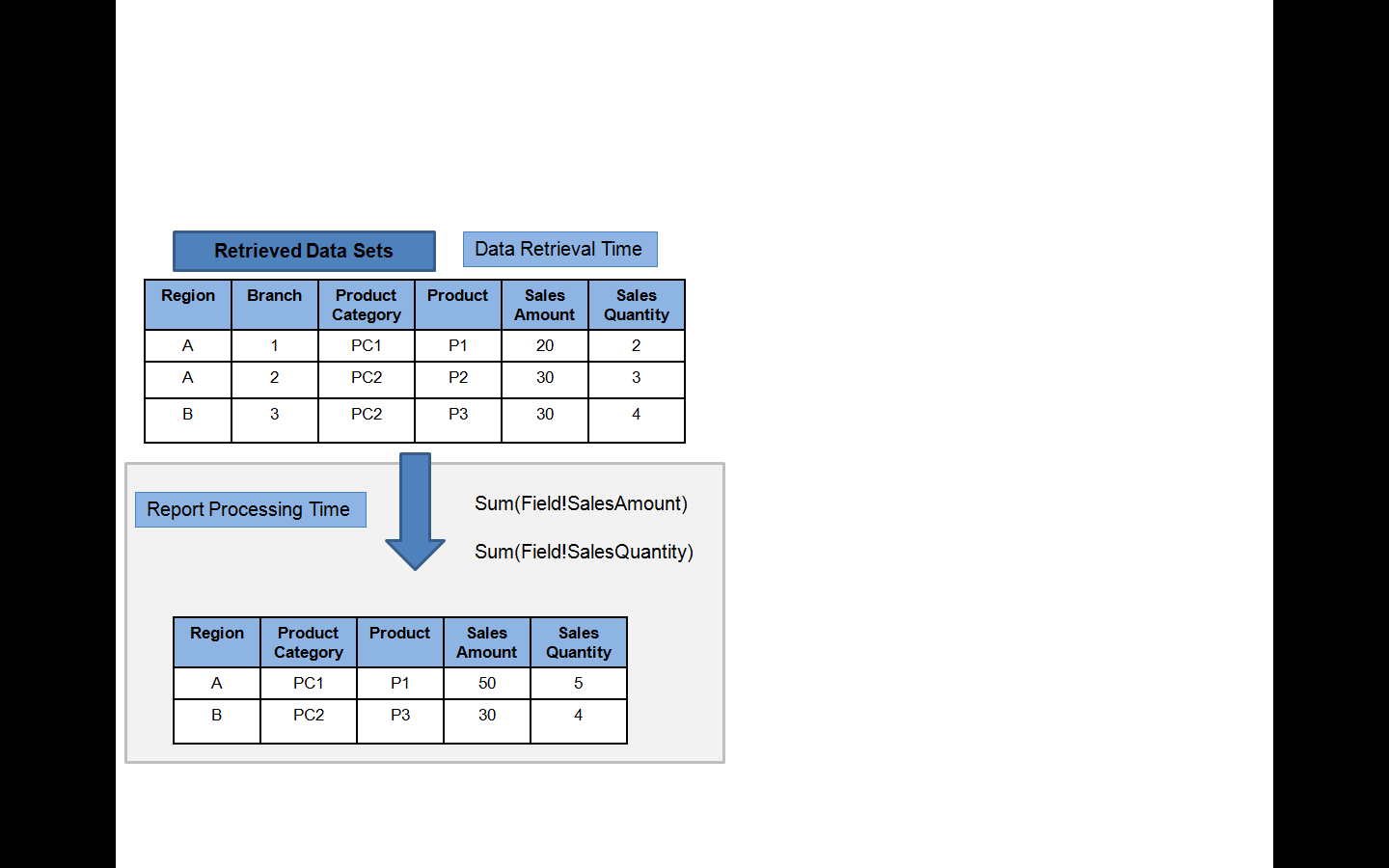


Figure 1

This behavior does not match most scenarios that retrieve data from Analysis Services. In most cases, a cube has already calculated and most of the summarized data stored, with the aggregation function defined in the cube structure. Some Analysis Services aggregate functions do not exist in Reporting Services, especially semiadditive measures.

Accordingly, Reporting Services provides the **Aggregate** function to support this scenario.

If you use the **Aggregate** function (Aggregate(field!xxx)), Reporting Services uses the custom aggregations provided through the data provider to calculate the expression.

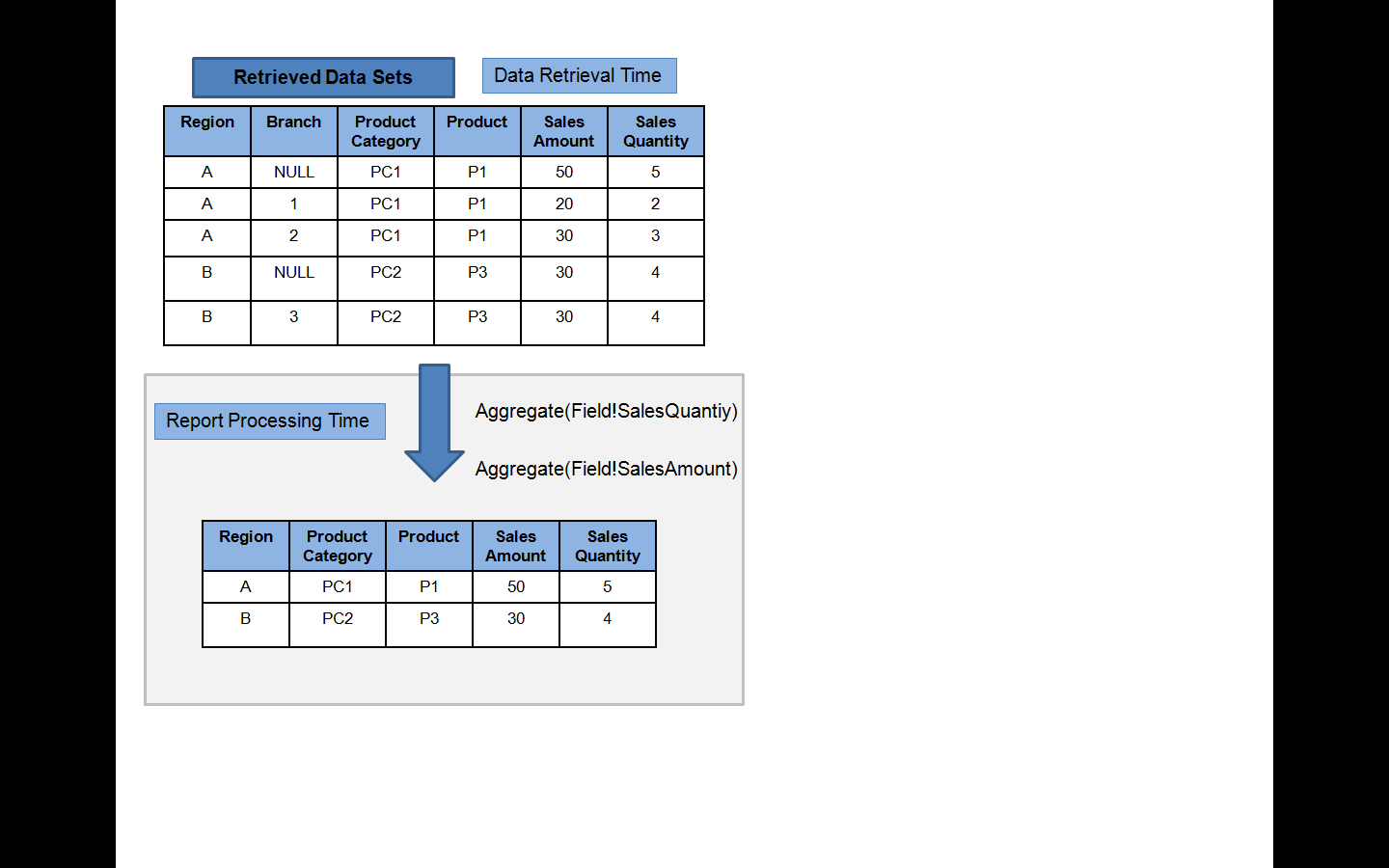


Figure 2

If you compare the retrieved data sets in Figure 2 and Figure 1, you will see that there are two new rows that contain a value of NULL in the Branch column in Figure 2. The Aggregate(Field!SalesAmount) expression, which is grouped by Region in row and ProductCategory and Product in column, finds the Sales Amount value of rows where Branch = NULL in the dataset. If it cannot find the row, empty data is assigned to it.

This behavior is one of the most important issues to watch when you are using the **Aggregate** function in the Tablix data region. This behavior affects the MDX query statement in the dataset, something that will be discussed in the next section.

Figure 3 illustrates how rows in the retrieved dataset map to columns and rows in the Tablix data region if the **Aggregate** function is used.

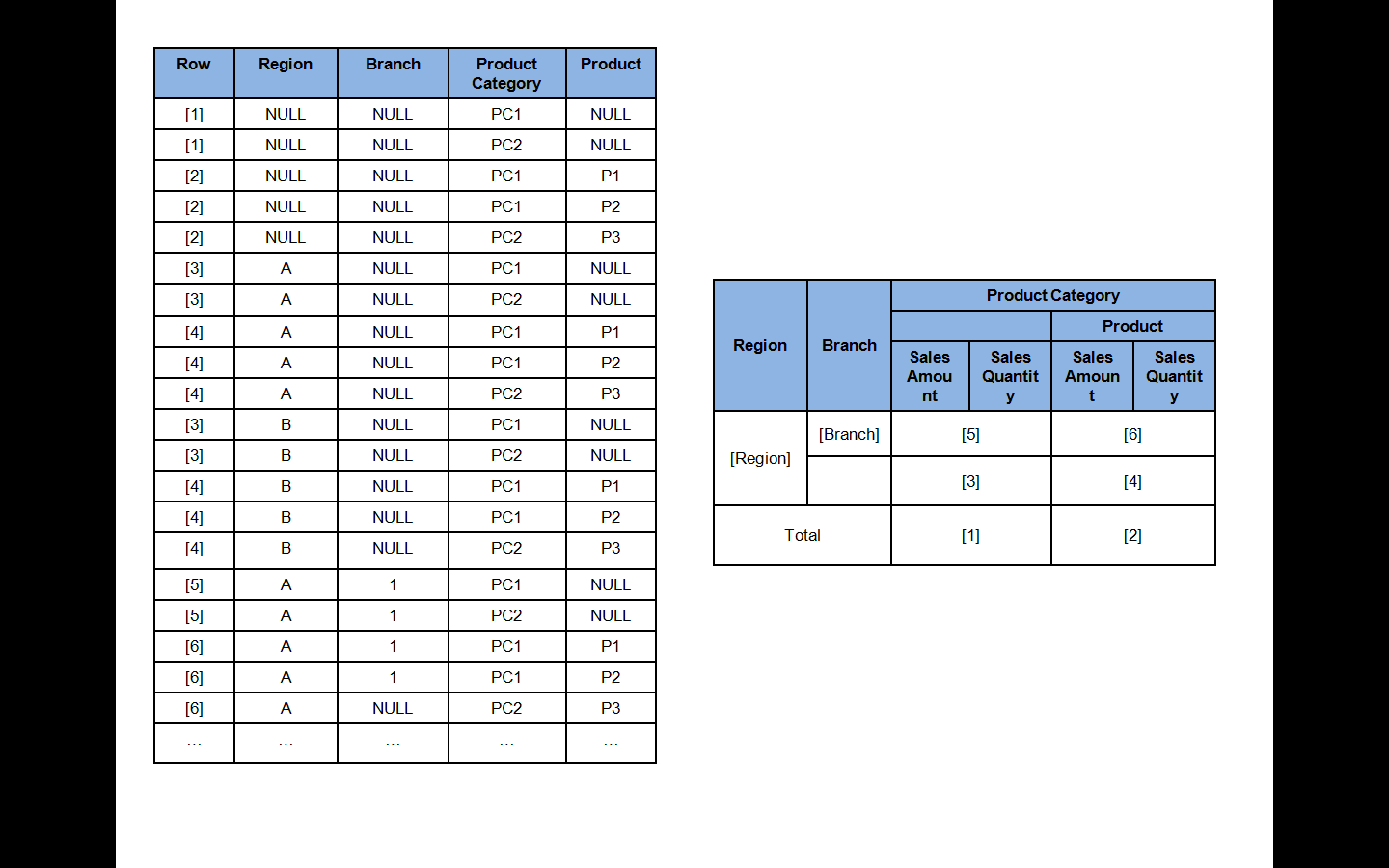


Figure 3

On the other hand, if you delete the Region column group from the grouping structure, values [5] and [6] will also be empty because there is no row of data where Region = NULL and Branch = 1/2/3.

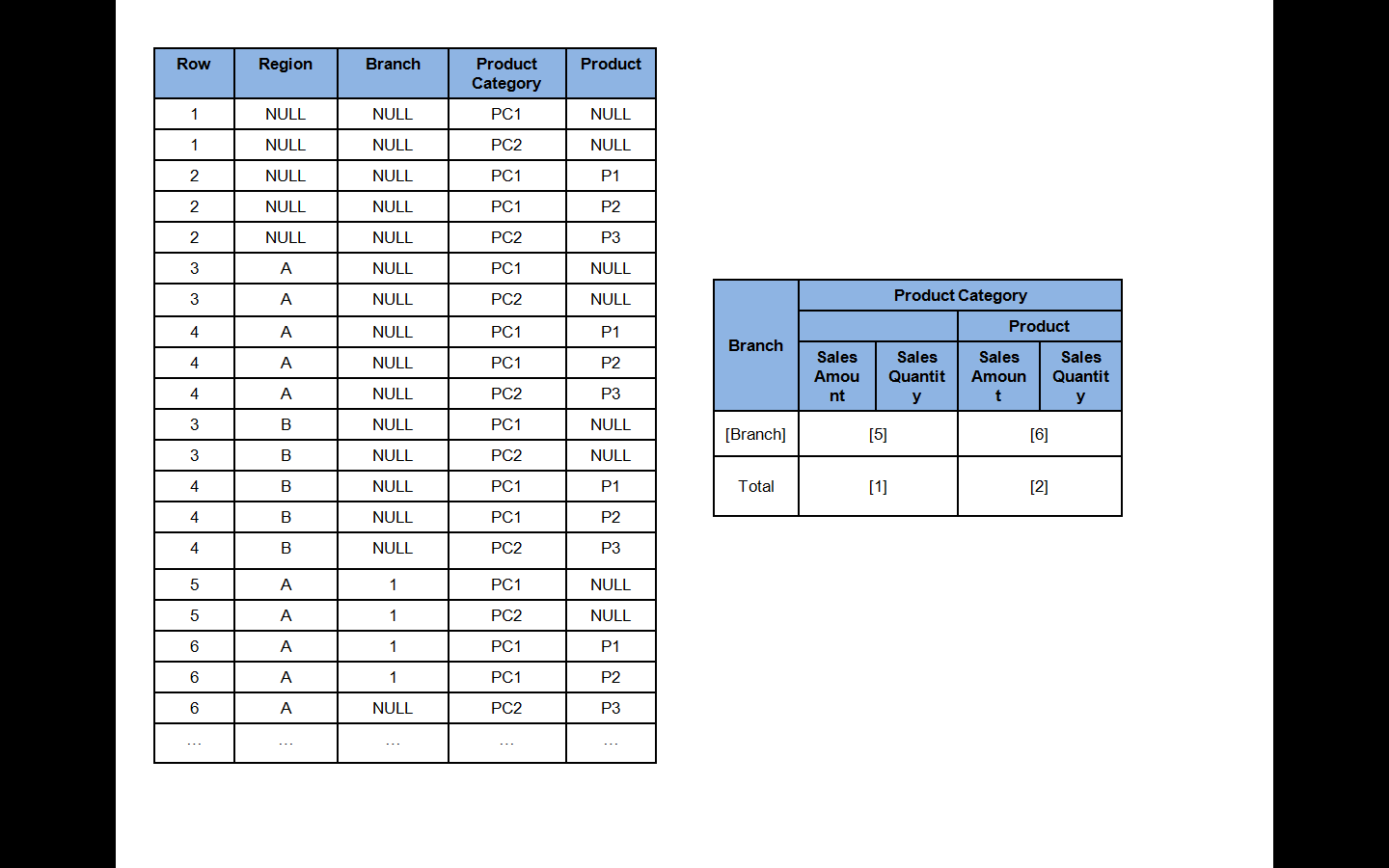


Figure 4

* 1. MDX Query and Tablix Behavior for the Main Dataset

As mentioned in the previous section, MDX statements in each dataset depend on the grouping structure of the Tablix data region, which is designed when the report is developed. This section provides step-by-step instructions for how to write an MDX query to include additional server aggregates so that the report can retrieve them and show various levels of aggregations in the cube.

To help demonstrate the behavior of MDX statements and the Tablix data region, we’ve created a sample cube called Sales. Figure 5 shows the structure of the Sales cube.





Figure 5

To create the sales summary report, group the Product hierarchy in rows and the Month hierarchy in columns. Figure 6 illustrates this report outline.

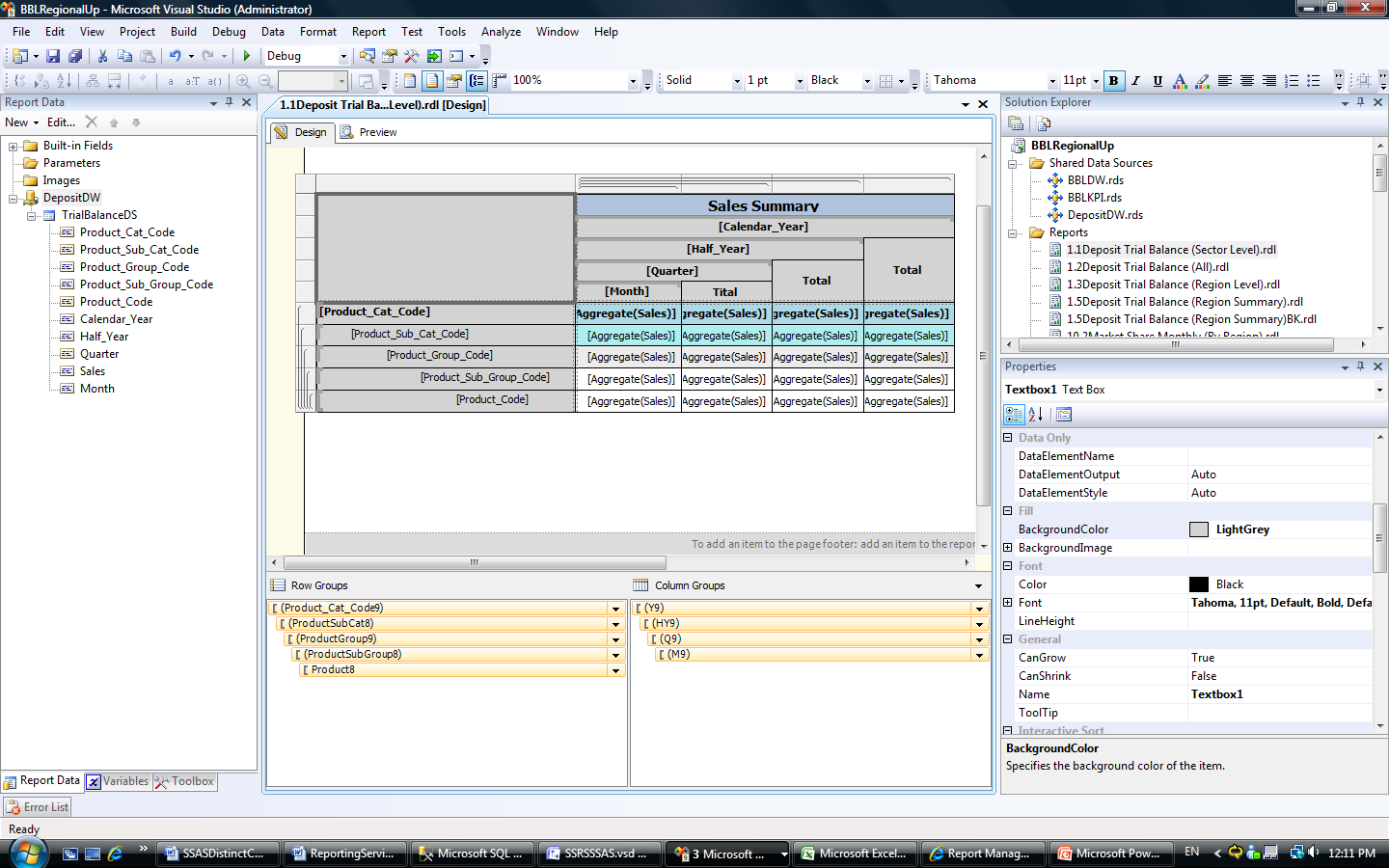
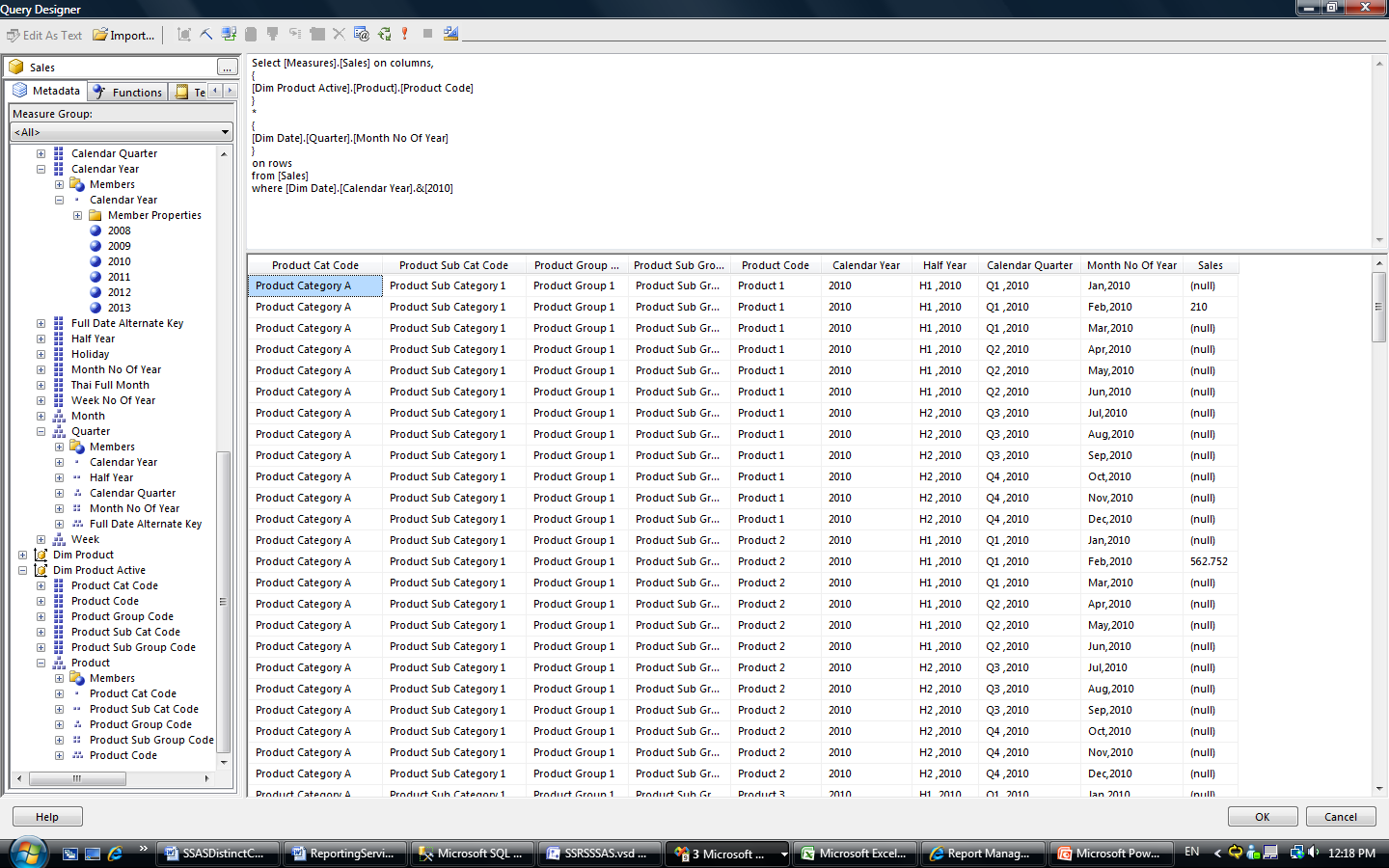


Figure 6

To test the aggregation function and the behavior of the Tablix data region, create this MDX query for the Tablix dataset.

|  |
| --- |
| Select [Measures].[Sales] on columns,  {  [Dim Product Active].[Product].[Product Code]  }  \*  {  [Dim Date].[Quarter].[Month No Of Year]  }  on rows  from [Sales] |

Figure 7 shows the query results and report.



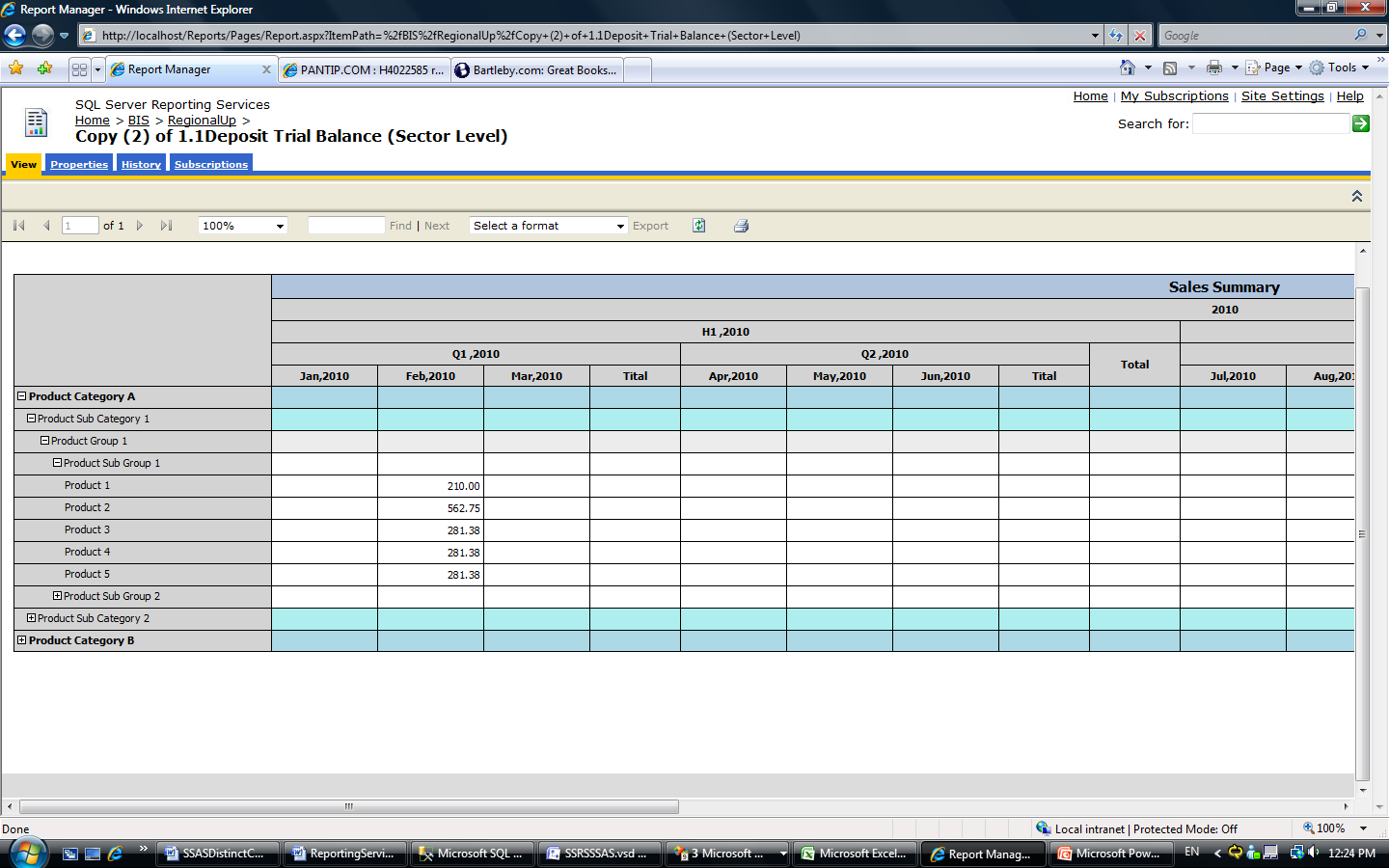
****

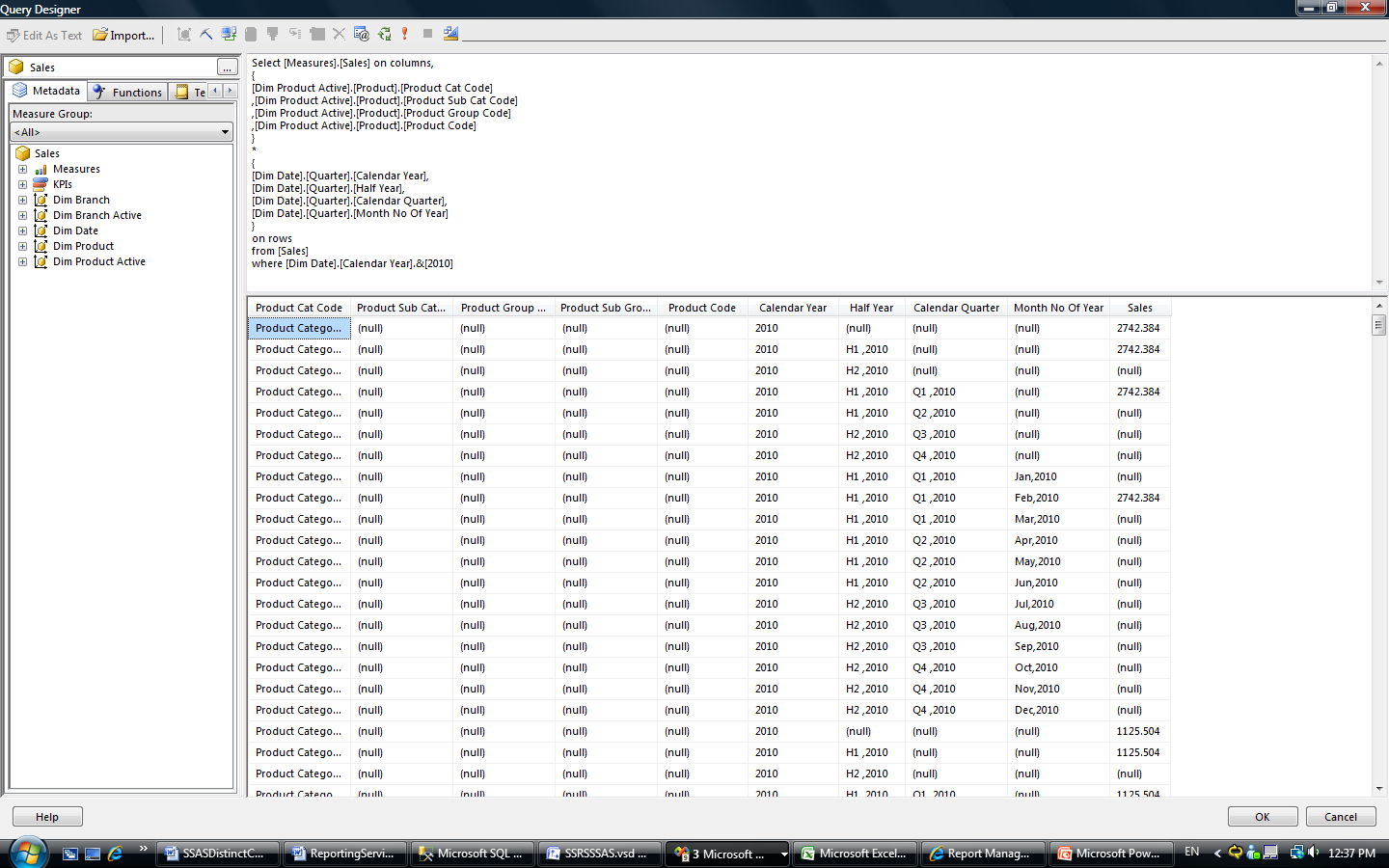
Figure 7

In the report, the sales data appears only at the [product code] and [Month no Of Year] level, which is incorrect.

To make the report display correctly, modify the MDX statement as shown here to query every aggregation level that is displayed in the report layout.

|  |
| --- |
| Select [Measures].[Sales] on columns,  {  [Dim Product Active].[Product].[Product Cat Code]  ,[Dim Product Active].[Product].[Product Sub Cat Code]  ,[Dim Product Active].[Product].[Product Sub Group Code]  ,[Dim Product Active].[Product].[Product Group Code]  ,[Dim Product Active].[Product].[Product Code]  }  \*  {  [Dim Date].[Quarter].[Calendar Year],  [Dim Date].[Quarter].[Half Year],  [Dim Date].[Quarter].[Calendar Quarter],  [Dim Date].[Quarter].[Month No Of Year]  }  on rows  from [Sales] |

Figure 8 illustrates the query results and report, which are now correct.



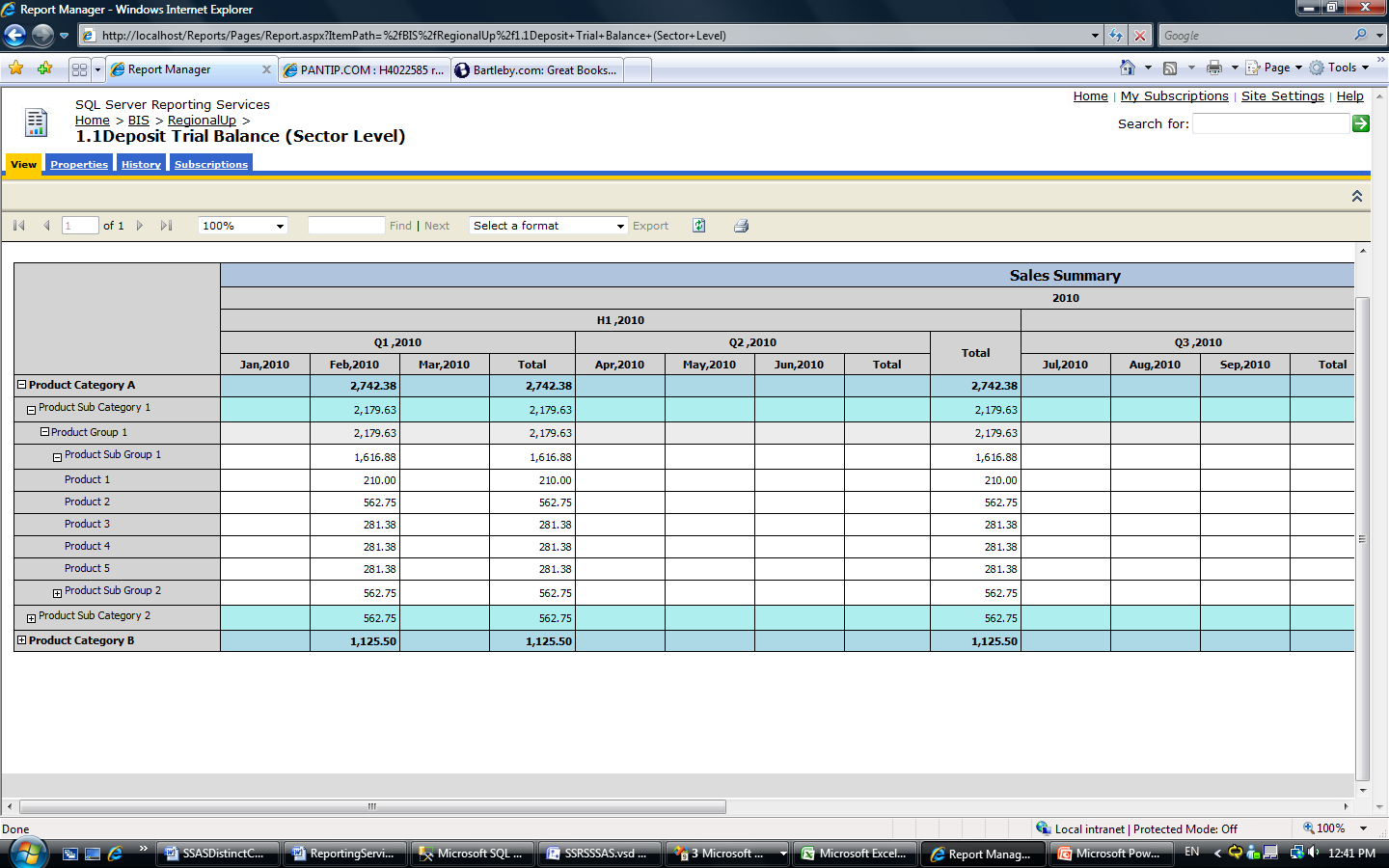
****

Figure 8

Another important concept to understand when you are working with Analysis Services data in Reporting Services is the way aggregations work with grouping. If you select the dimension hierarchy from the cube, you must group the data from the top level to the level for which you want to display the Aggregate() value, even if you do not need to display the actual data in the upper level. Otherwise, you may not get the right results.

For example, in this case, the customer wanted to include data about products in the report, so we selected the Product hierarchy from the Product dimension. Rows that contain the aggregation value of Product Group Code ([Product Sub Group Code] and [Product Code] = NULL) contain the information from [Product Cat Code] and [Product Sub Cat Code], which are the upper levels in the Product hierarchy. If these upper levels are not included, their data isn’t included in the aggregations. This is why you must group by all upper-level attributes in the hierarchy to display the correct server aggregation value in the Reporting Services report.

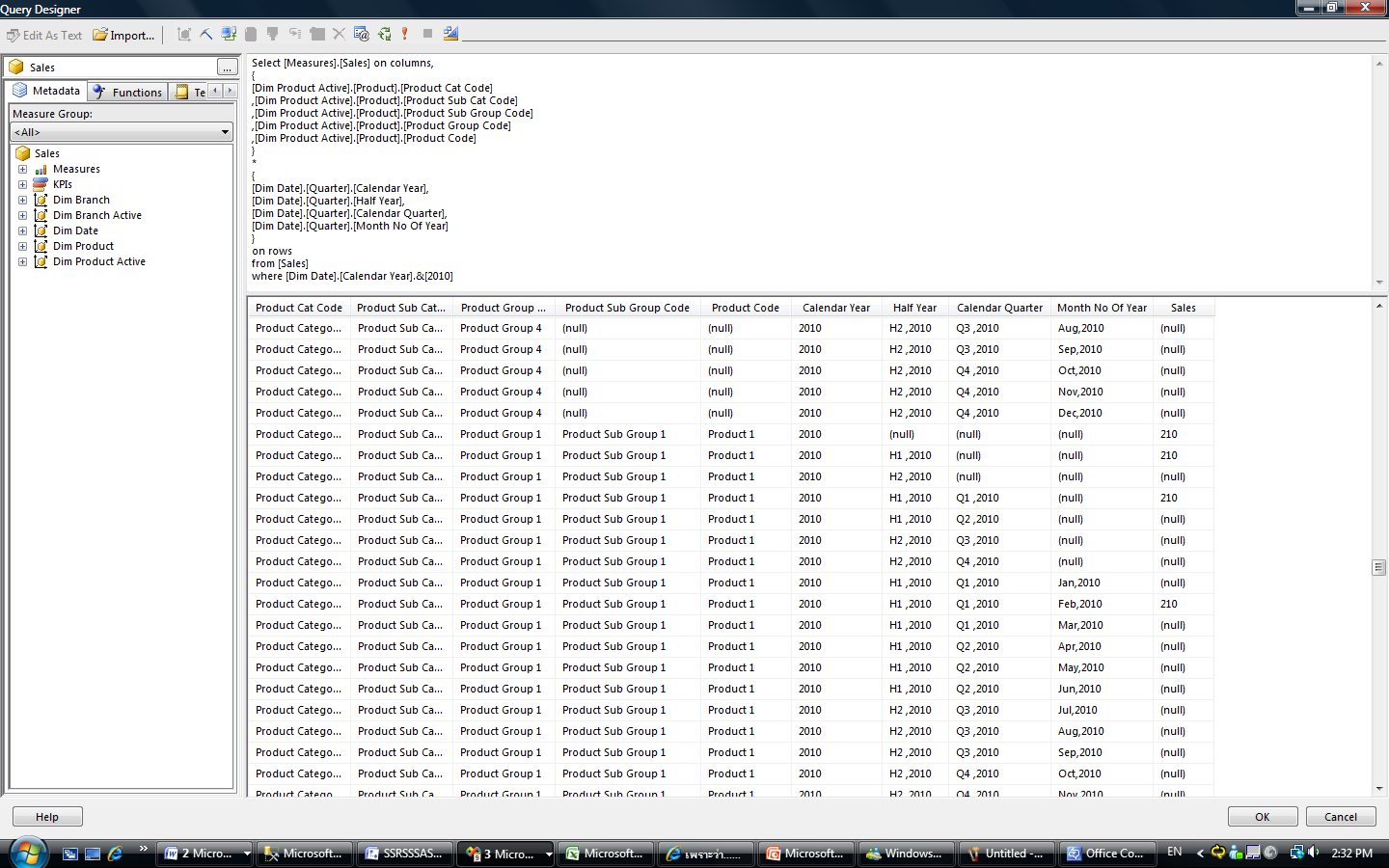
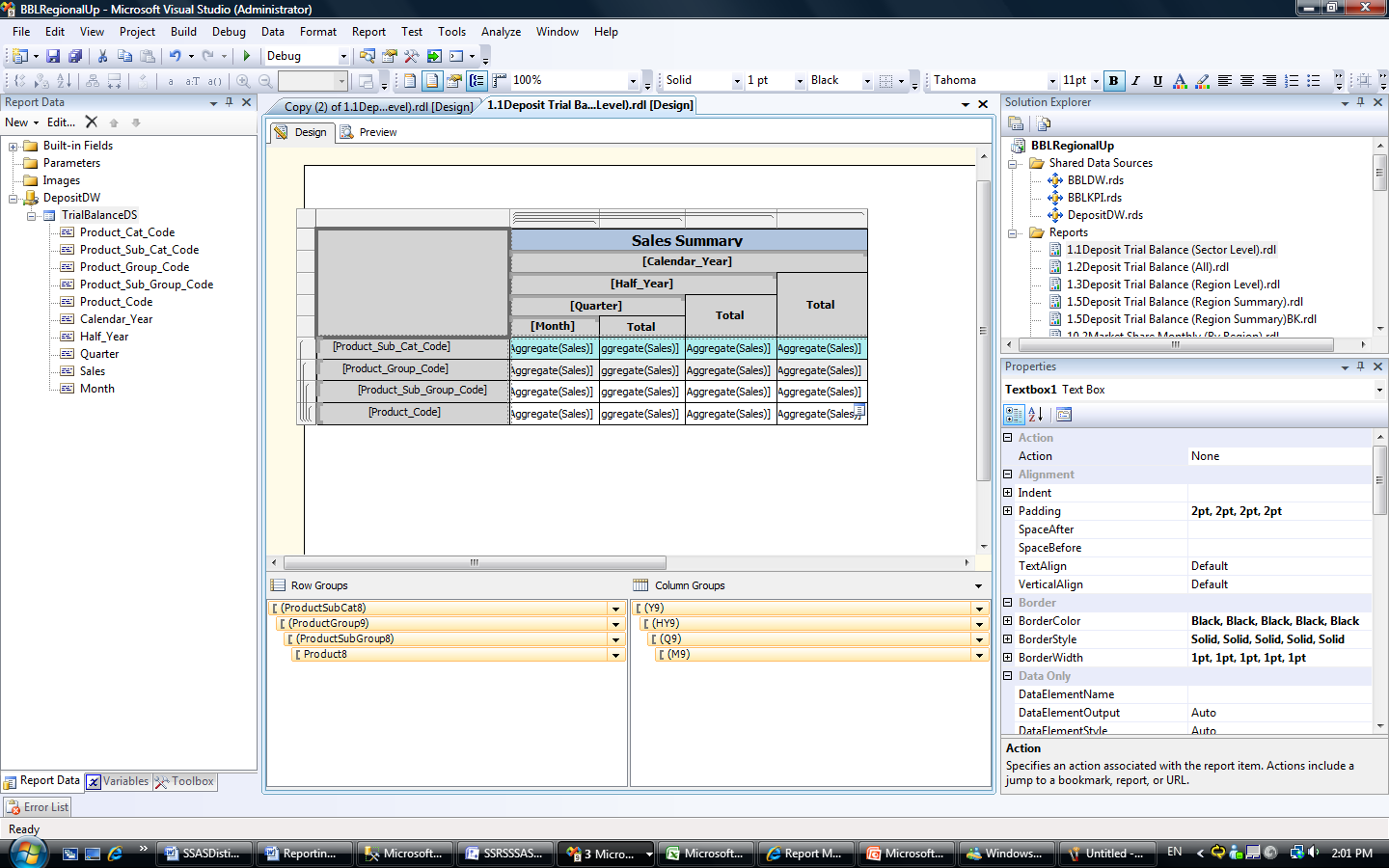


Figure 9

Figure 10 shows the result when the row group that is grouped by [Product\_Cat\_Code] is deleted from the report layout. All aggregation values in the report disappear.



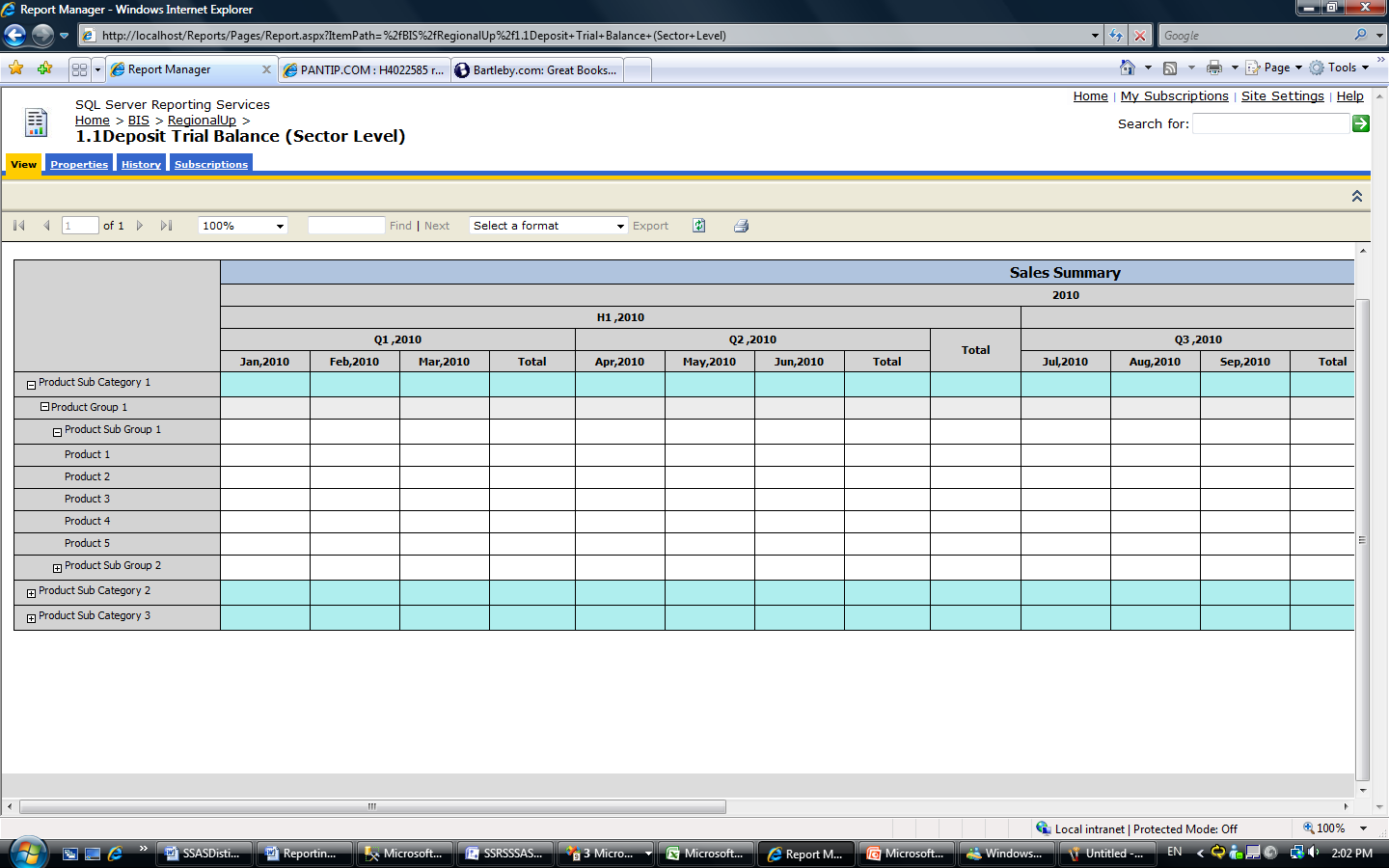


Figure 10

**Tip**

|  |
| --- |
| SQL Server 2008 Reporting Services provides a graphical query designer that you can use as an MDX editor. You can drag and drop functions to generate MDX code. Moreover, the query designer provides a Show Aggregations button, which automatically adjusts the MDX code of the dataset when you use the set the aggregation function of the field in the Tablix data region that binds with this dataset to the Aggregate function. |

* 1. Utilizing Report Parameters and the Dimension Caption Property

You can use the query designer to generate data based on dimension attributes you select. The **Dimension Caption** property enables you to display the name of the dimension in text to the user of the report. This section discusses the use of report parameters and the **Dimension Caption** property.

Report Parameter and Member Unique Name

Most of the reports we designed for this customer use dimension members as report parameter values to slice or select data from the cube.

The query designer in SQL Server 2008 Reporting Services can automatically generate report parameters and their associated datasets when you select dimension attributes.

Here is an example of a dataset generated by the query designer.

|  |
| --- |
| WITH  MEMBER [Measures].[ParameterCaption] AS  [Dim Branch Active].[BranchStructure].CURRENTMEMBER.MEMBER\_CAPTION  MEMBER [Measures].[ParameterValue] AS  [Dim Branch Active].[BranchStructure].CURRENTMEMBER.UNIQUENAME  MEMBER [Measures].[ParameterLevel] AS  [Dim Branch Active].[BranchStructure].CURRENTMEMBER.LEVEL.ORDINAL  SELECT {[Measures].[ParameterCaption],  [Measures].[ParameterValue],  [Measures].[ParameterLevel]}  ON COLUMNS  , [Dim Branch Active].[BranchStructure].ALLMEMBERS  ON ROWS  FROM [Sales] |

In the dataset:

* Member\_Caption provides the name of each member.
* Uniquename provides the full reference name of member such as [Dim Branch Active].[Branch Structure].[Branch Code].[A].
* Level.ordinal provides the level of the member in the hierarchy.

You should set [Measures].[ParameterCaption] as the label and [Measures].[ParameterValue] as the value of the report parameter. Then you can use the query editor to apply this information to your report.

Dimension Caption Property

The dimension properties code enables you to access member properties in the report. This part of MDX code enables you to more easily reference the **MemberUniqueName**, **memberlevel**, or **ParentMemberUniquename** properties. For example, you can reference the **MemberUniqueName** of [Branch Code] field by including “[Branch Code].UniqueName” rather than “[Branch Code].Value”, as in the following example.

|  |
| --- |
| DIMENSION PROPERTIES  MEMBER\_CAPTION  ,MEMBER\_UNIQUE\_NAME  ,PARENT\_UNIQUE\_NAME  ,LEVEL\_NUMBER  ON ROWS |

We used **MemberUniqueName** in the reports for many reasons. We frequently used this property as parameters in Action links of a text box, text, and so on. Because the reports retrieve data from Analysis Services, they always receive cube dimension members as the report parameters. **ParentUniqueName** is always used in parent-child grouping by Analysis Services. For more information, see “Implementing Dynamic Balance Scorecard with Analysis Services and Reporting Services” later in this paper.

* 1. Example Case Study: Sales Summary Reports

This section walks you through the creation of a report called Sales Summary. This is one of the custom reports we created for our Thai customer as part of the SQL Server 2008 solution we provided. We created the Sales Summary report to display the summary sales amount and its movement by month and by day from the Sales cube, which was described in section 3.2. Figure 11 shows the report.

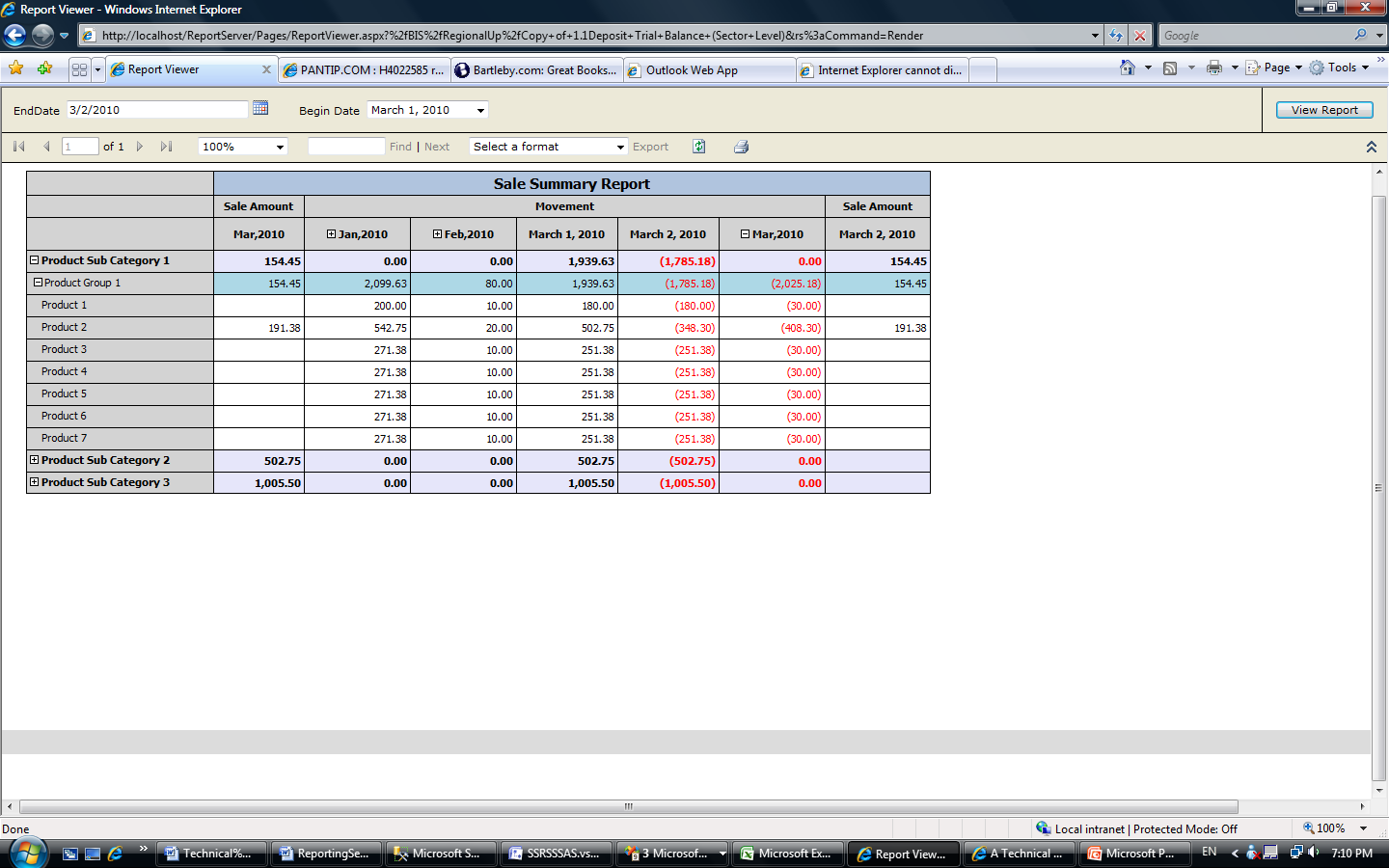


Figure 11

* + 1. Business Requirement

The customer wanted this report to display the following information:

* Amount of sales and movement displayed by product, month, and day.
* Reports filtered by begin date and end date.
* Sales amount movement from January within the year of the selected end date to the month of the selected end date.Dates only from the selected begin date to the selected end date.
* Sales amounts for today’s date.

In addition, the customer wanted users to be able to:

* Drill down to daily sales amount movement in the month of the selected end date.
* Define periods of data ending on days other than the end of the month. If users select an end date that is not the last day of the month, the aggregation value of the month of the EndDate report parameter should be aggregated from the first day of the month to the end date selected, not the last day of the month. For example, for Figure 11, Sale Amount for the Mar, 2010 column should sum from only March 1 and 2, although today is March 20th.
  + 1. Report Design

We used two report parameters, which enabled us to get data dynamically, and generated a dataset to design the report. We also added a Tablix data region to provide a dynamic visual display of the data.

* + - 1. Report Parameters

We created two report parameters:

* EndDate. We used an out-of-the-box date/time date picker.
* BeginDate. We created the dataset for this parameter. We listed the first day of the month and year of the selected value for EndDate to the previous day of the selected value for EndDate.

Here is the MDX code for the BeginDate dataset.

|  |
| --- |
| WITH  Member FullDateAlternateMember  As [Dim Date].[Quarter].CurrentMember.UniqueName  SELECT  {  FullDateAlternateMember  } ON COLUMNS,  {  (STRTOMEMBER(@BeginDate):  STRTOMEMBER(@EndDate))  }  DIMENSION PROPERTIES  MEMBER\_CAPTION  ,MEMBER\_UNIQUE\_NAME  ON ROWS  FROM [Sales] |

These are the query parameters we used:

* @BeginDate query parameter. This Report Definition Language (RDL) subtracts days so that the resulting date represents the first day of the month.

|  |
| --- |
| "[Dim Date].[Quarter].[Full Date Alternate Key].&[" & Format(CDate(Parameters!RptEndDate.Value).AddDays((CDate(Parameters!RptEndDate.Value).Day()\*-1)+1),"yyyy-MM-ddT00:00:00") & "]" |

* @EndDate query parameter.

|  |
| --- |
| "[Dim Date].[Quarter].[Full Date Alternate Key].&[" & Format(Parameters!RptEndDate.Value, "yyyy-MM-ddT00:00:00") & "]" |

* + - 1. Dataset

We generated the dataset using the following steps:

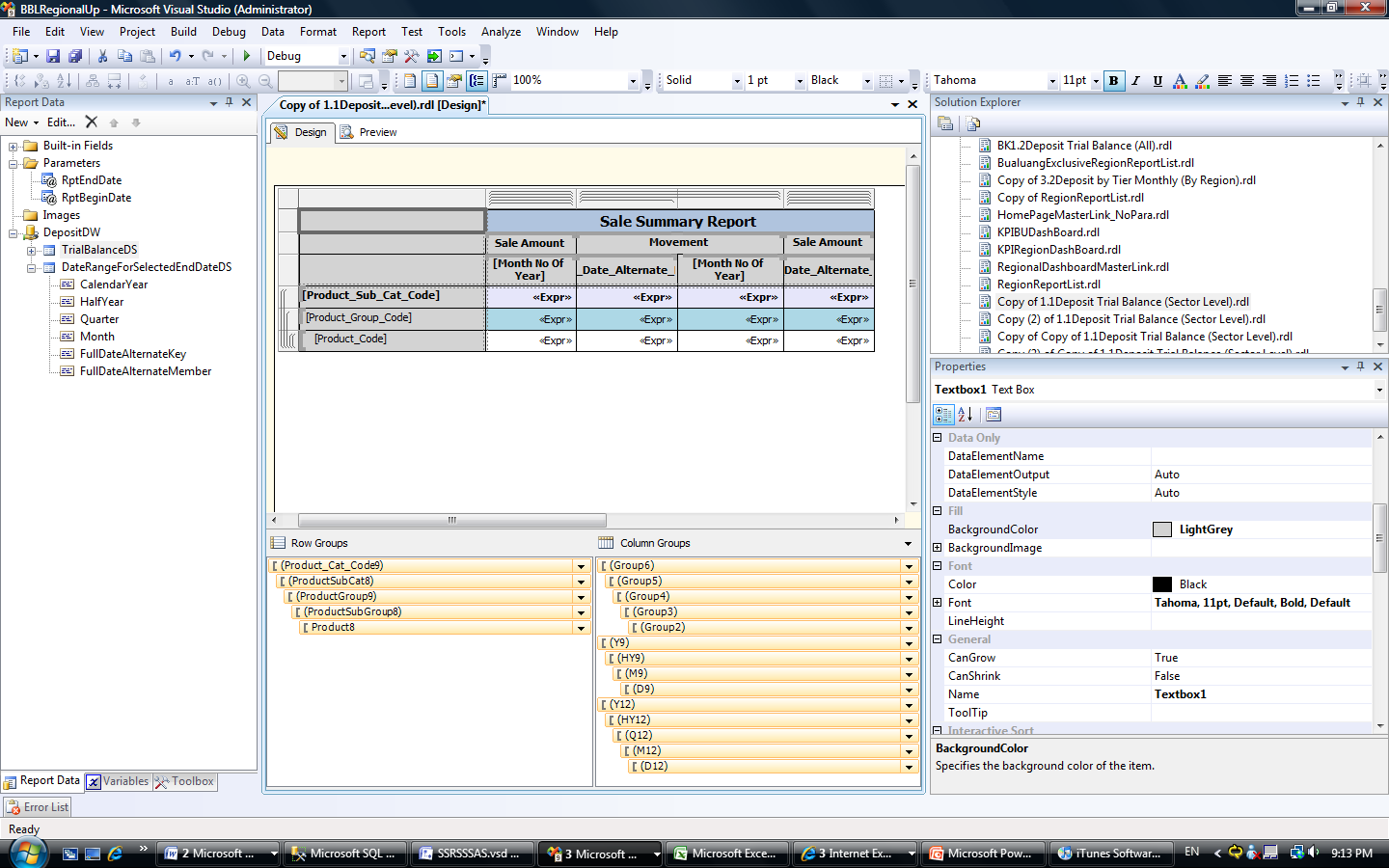
1. Select the Sales and SalesVarianceToLastPeriod measures on columns.
2. Select all members of the Product Sub Cat Code, Product Group Code, and Product Code attributes in the Product hierarchy. These fields are used for row grouping.
3. Select members of [Dim Date].[Quarter].[Month No Of Year] with @DimBeginMonth to @DimEndMonth.
4. Select members of [Dim Date].[Quarter].[FullDateAlternateKey] with @DimBeginDate to @DimEndDate.
5. Use the subcube defined by all of the selected criteria: In the subcube, select members of [Dim Date].[Quarter].[Month No Of Year] only from @DimBeginMonth to @DimBeforeEndMonth. This enables @DimEndMonth to aggregate the measure value from only @DimBeginDate to @DimBeginDate. (As mentioned earlier, the customer required some flexibility with end dates. This was the issue: If users selected the a value for the EndDate report parameter that was not the last day of the month, the data returned by @DimEndMonth could be wrong, because the measure value from every date is aggregated. We used the subcube technique to address this issue.)

This is the MDX code.

|  |
| --- |
| SELECT  Non Empty  {  [Measures].[Sales]  ,[Measures].[SalesVarianceToLastPeriod]  } ON COLUMNS  ,NON EMPTY  {  {  [Dim Product Active].[Product].[Product Sub Cat Code].ALLMEMBERS  ,[Dim Product Active].[Product].[Product Group Code].ALLMEMBERS  ,[Dim Product Active].[Product].[Product Code].ALLMEMBERS  }  \*  {  STRTOMEMBER(@DimBeginMonth):STRTOMEMBER(@DimEndMonth),  STRTOMEMBER(@DimBeginDate):STRTOMEMBER(@DimBeginDate)  }  }  DIMENSION PROPERTIES  MEMBER\_CAPTION  ,MEMBER\_UNIQUE\_NAME  ON ROWS  FROM  (  SELECT  (  {  STRTOMEMBER(@DimBeginMonth):STRTOMEMBER(@DimBeforeEndMonth),  STRTOMEMBER(@DimBeginDate):STRTOMEMBER(@DimEndDate),  STRTOMEMBER(@DimLastYear)  }  )  ON COLUMNS  FROM [Sales]  )  These query parameters contain the unique name of members from the Dim Date dimension. [Quarter hierarchy]They are defined with the expression in the same way the query parameters of Begin Date Dataset are defined (that is, concat string [Dim Date].[Quarter].[… with a date time function to format the report parameters – BeginDate and EndDate).  @DimBeginMonth = January in the year of the selected end date  @DimEndMonth = The month of selected end date  @DimBeforeEndMonth = The month before @DimEndMonth  @DimBeginDate = The selected begin date  @DimEndDate = The selected end date  @DimLastYear = The year before the year of the selected end date |

* + - 1. Tablix

We added a Tablix data region showing the information from the dataset we generated.

****

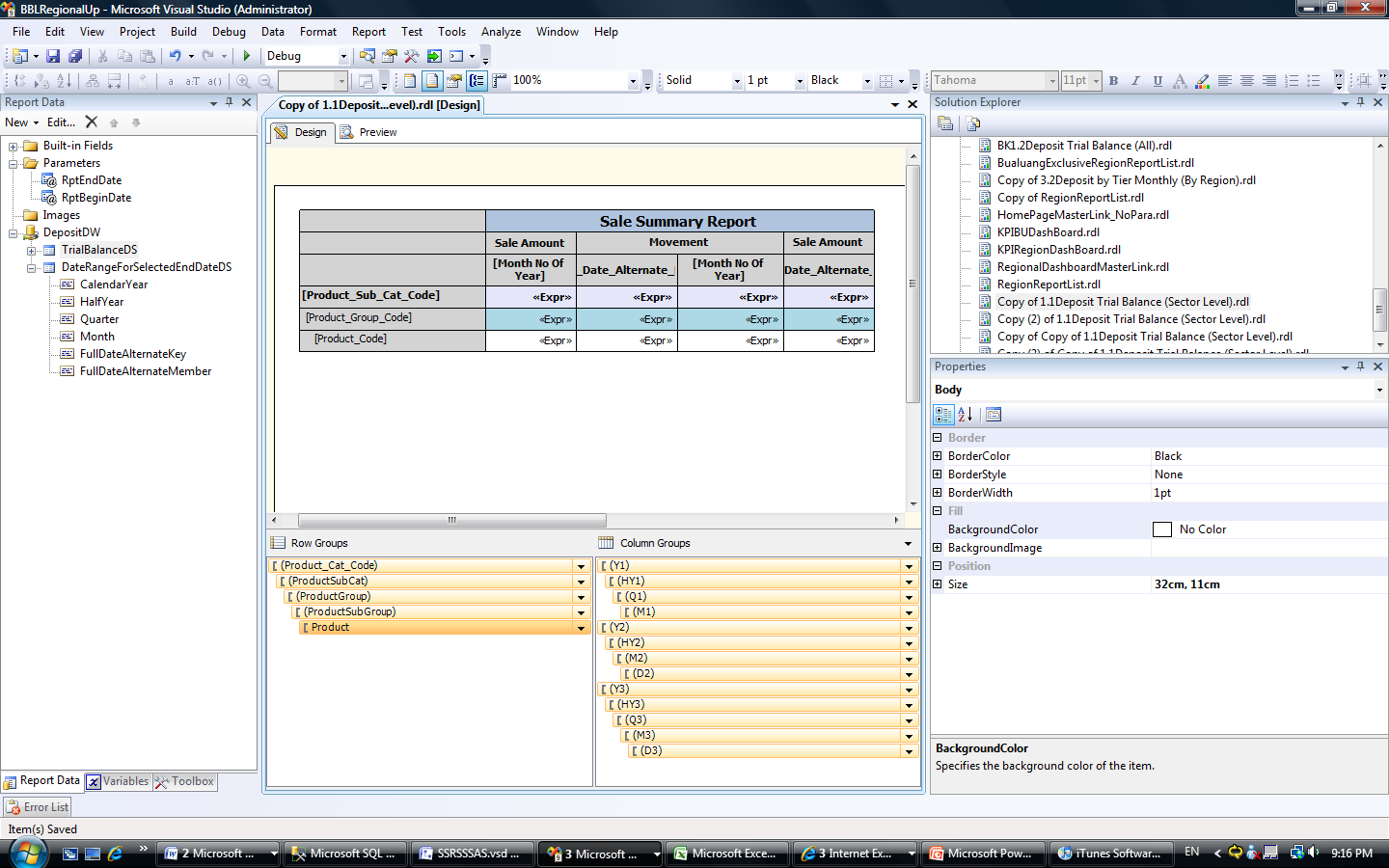


Figure 12

In the Tablix data region, you can see that:

* Attributes are grouped in each level of the Product hierarchy as row groups.
* Three adjacent groups are used to create the column group hierarchy. Together, they represent the grouping of attributes in each level of the Quarter hierarchy.

The first column in the Tablix data region is grouped by Year (Y1), Half Year (HY1), Quarter (Q1), and Month (M1). The expression for the **Hide** property of the group is set to M1 to hide this group whenever the value of [Month No of Year] is not a month in the EndDate report parameter.

The second column is grouped by Year (Y2), Half Year (HY2), Quarter (Q2), Month (M2), and FullDateAlternateKey (D2). The purpose of this column group hierarchy is to show the movement between the first and third column group hierarchies.

The third column is grouped by Year (Y2), Half Year (HY2), Quarter (Q2), and Month (M2). It is the total column of the second column.

The fourth column is grouped by Year (Y3), Half Year (HY3), Quarter (Q3), Month (M3), FullDateAlternateKey (D3). The expression for the **Hide** property of the group is set to D3 to hide this group whenever [FullDateAlternateKey] value is not the EndDate report parameter.

**Tip**

|  |
| --- |
| Be careful when using adjacent groups in the Tablix data region because they may not align properly. For example, we moved the Sales Amount by Month column to the last column, as in Figure 13.    Figure 13  When we executed this report, we got incorrect data in the Sales Amount by month column. It displayed the same data as the Sales Amount by day column, which was grouped by [Full Date Alternate Key], even though we grouped it by [Month No Of Year].    Figure 14  The root cause of this problem is the column group in the first column. The lowest level of this group is [Full Date Alternate Key], which limits the column group to its right: If a column group groups the data to level #n of the hierarchy, its right adjacent group cannot aggregate data in the upper level of the same hierarchy. |

1. Implementing Dynamic Balance Scorecards with Analysis Services and Reporting Services

One of the most important parts of this solution is the Balance Scorecard. The company wanted a daily-updated dashboard that provided up-to-date data of important KPIs. We created a Balance Scorecard for them. This solution provides the company with the capability to manage performance better.

* 1. Business Requirement

This section lists the business requirements that we used to design the company Balance Scorecard.

First, they wanted a Balance Scorecard to be implemented for every business unit with the same mechanism but a different structure. Figure 15 shows the company’s organizational structure.



Figure 15

They wanted the Balance Scorecard of each business unit to consist of different KPI groupings in different structures.

They wanted the Balance Scorecard to include the following measures: Objectives, group of KPIs, KPI, Weight, Actual, Target, Actual on target, and Actual on Target after Weight. They also wanted it to be able to calculate the summary score by using user-allocated weight. They wanted it to have an unbalanced structure without a level limit. Table 1 shows the measures.

|  | **Weight** | **Actual** | **Target** | **Actual on Target** | **Score(Actual on Target after Weight)** |
| --- | --- | --- | --- | --- | --- |
| Total | 100% | - | - | - | 53% |
| Perspective #1 | 50% | - | - | - | 23.0% |
| Objective #1 | 50% |  |  |  | 23.0% |
| KP I #1 | 30% | 30 | 100 | 30% | 9.0% |
| KPI #2 | 20% | 70 | 100 | 70% | 14.0% |
| Perspective #2 | 50% | - | - | - | 30% |
| KPI #3 | 50% | 60 | 100 | 60% | 30% |

Table 1

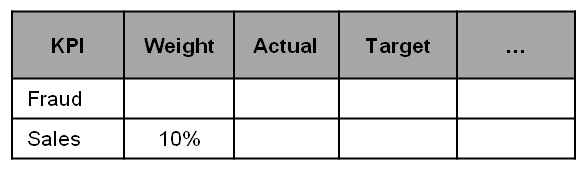
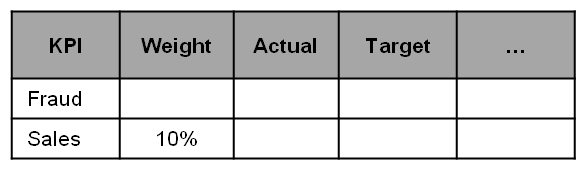
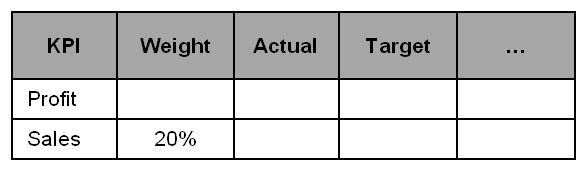
Some organization hierarchies, such as Region and Branch, needed have more than one scorecard instance. For example, there are seven regions under Business Unit #1. All of them share the same Balance Scorecard structure but have different target and weight. There are more than a thousand Balance Scorecard instances in total. Although each business unit uses a different Balance Scorecard structure, the customer wanted them to be able to share some KPIs and aggregate the actual and target value from the lower level to the upper level.



Figure 16

Though they expected the Balance Scorecard structure to change every year, they needed to preserve history that could be accessed when users select scorecards from previous years.

Although different business units use different balance scorecard structures, they needed to be able to share some KPIs and aggregate the actual and target values from the lower levels to the upper levels.



**Region**

**Branch**

“Aggregate: Sum”

Figure 17

Finally, the customer wanted users to be able to adjust or create new Balance Scorecards by themselves. They wanted the IT team involved only if a new KPI was introduced.

* 1. Solution Design

Our solution design for the Balance Scorecard that we provided included the following:

* A Balance Scorecard cube, which contains all necessary data for the report. All cube data is generated by users via an ETL process. It includes Actual, Target, and Weight measures.
* A KPI implementation dimension. This dimension enables end users to adjust and create their own custom Balance Scorecard structures. It contains the Balance Scorecard structure for each business unit in each calendar year. To support unbalanced-structure scorecards, this dimension is a parent-child dimension.
* A Balance Scorecard report for Reporting Services. This report retrieves data from the Balance Scorecard cube. The report parameter is Organization Unit (Scorecard Structure), Organization, Date.
  1. OLAP Design

The Balance Scorecard cube includes three dimensions: Dim Date, Dim Organization, and Dim KPI Implementation.



Figure 18

Figure 18 shows these dimensions.

Dim Date is a time dimension.

Dim Organization contains the organizational structure. The dimension members are Balance Scorecard instances. Figure 19 shows an example of dimension members.

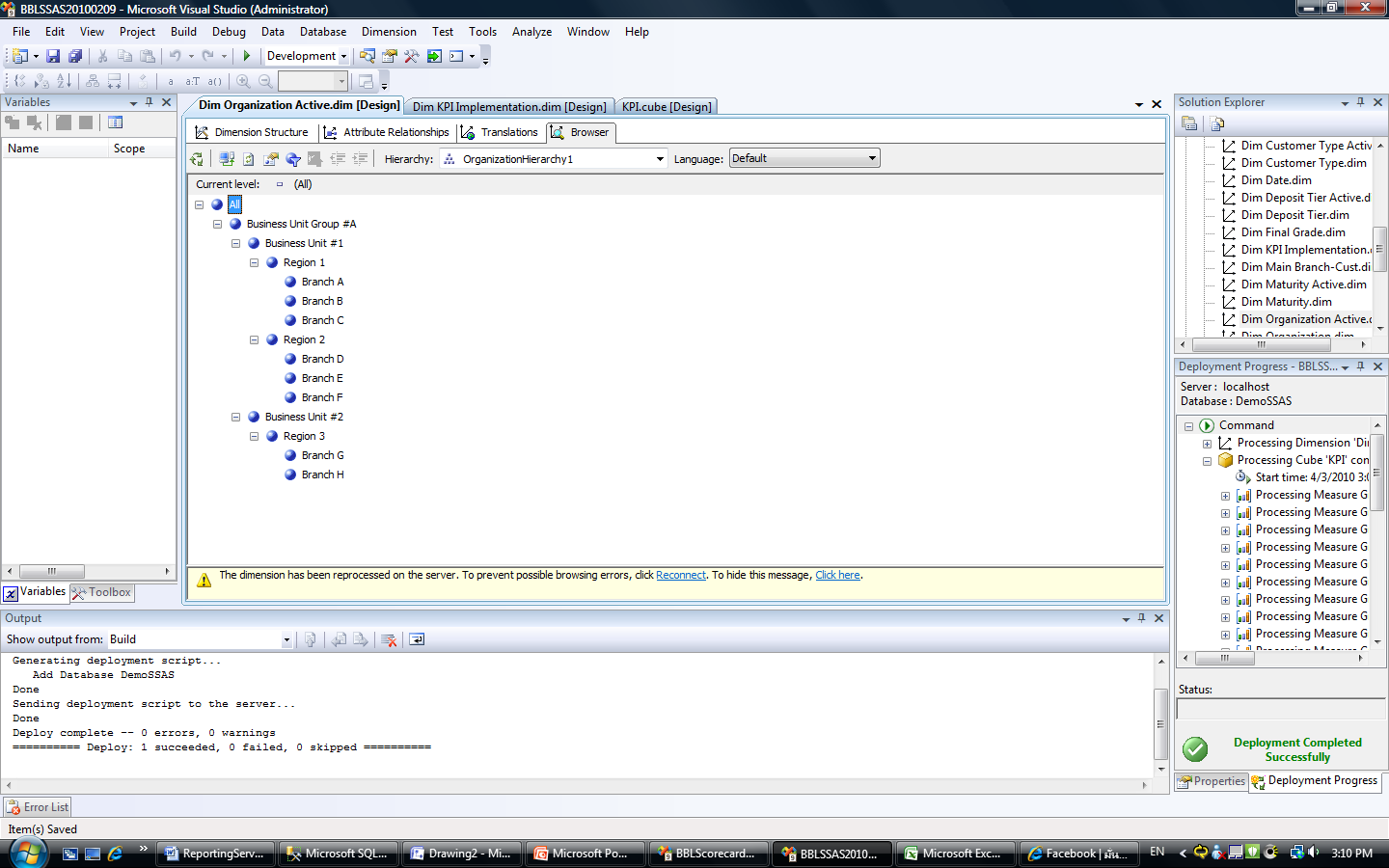


Figure 19

Dim KPI Implementation is a parent-child dimension that contains the definition of the Balance Scorecard structure. This dimension consists of four required attributes, as shown in Figure 20.



Figure 20

KPI Master is the definition of the KPI, such as Sales amount, Sales quantity, Growth, Cost. It also contains the definition of aggregation type of this KPI such as Sum, Average, and Last-Non Empty.

Scorecard Master is the definition of the scorecard.

Scorecard Year lists the year which the KPI was implemented.

KPI Master, Scorecard Master, and Scorecard Year are master tables that have a one-to–many relationship with the Dim KPI Implementation table. Dim KPI Implementation is implemented as a snowflake dimension. It enables the design to use the same KPI for each scorecard for each year. Table 2 is an example of the data used to create the scorecard.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **KPI Implementation ID** | **Parent KPI ID**  **( Parent\_Imp\_Key )** | **KPI** | **Scorecard** | **Year** |
| 1 | NULL | Summary | Company | 2010 |
| 2 | 1 | Financial | Company | 2010 |
| 3 | 2 | Sales Amount | Company | 2010 |
| 4 | 1 | Sales Quantity | Company | 2010 |
| 5 | NULL | Summary | Business Unit #1 | 2010 |
| 6 | 5 | Financial | Business Unit #1 | 2010 |
| 7 | 6 | Sales Amount | Business Unit #1 | 2010 |

When users browse the Balance Scorecard cube via Excel, the Dim KPI Implementation dimension structure appears in the format shown in Figure 21.

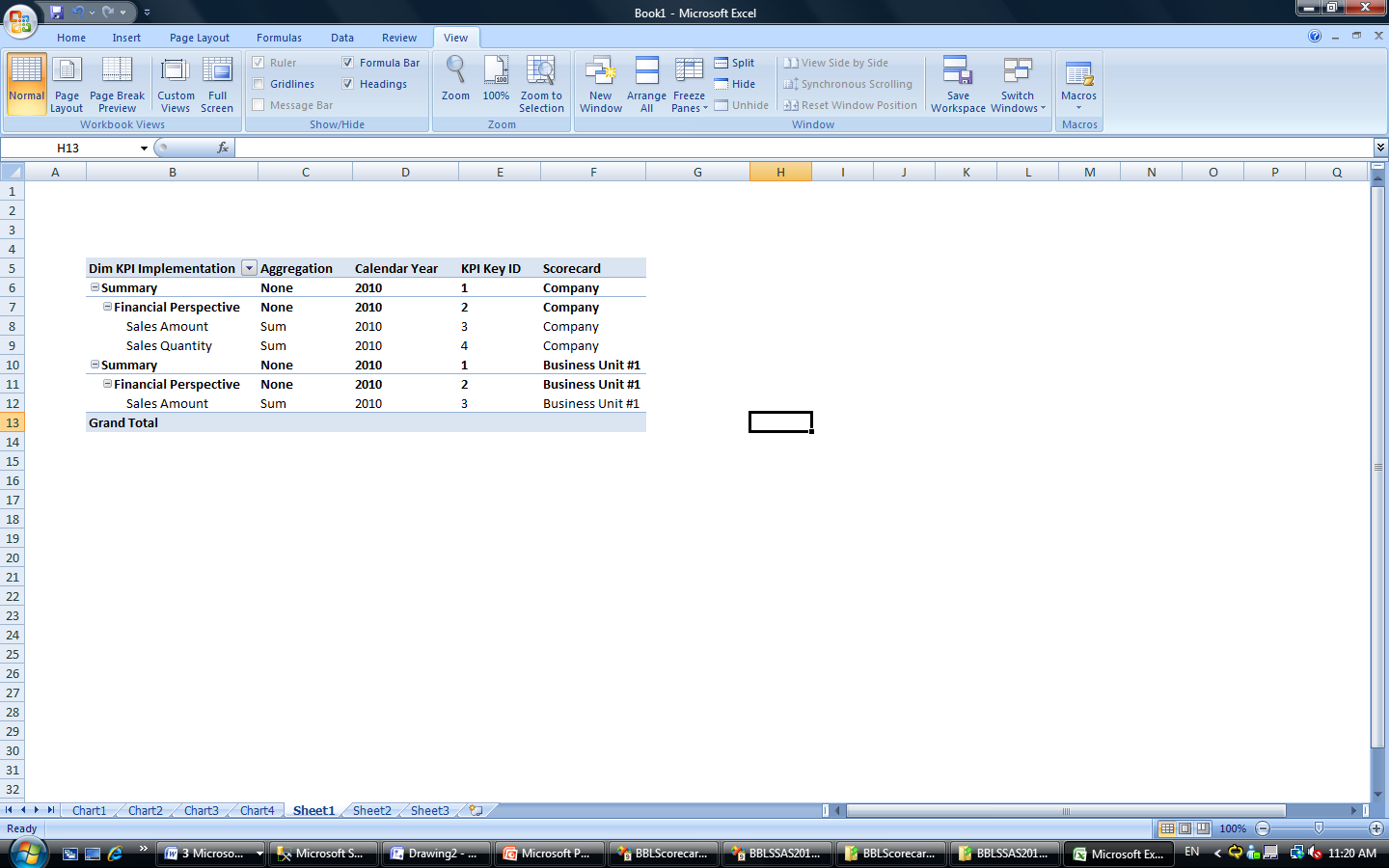


Figure 21

The cube enables users to slice the cube on Scorecard Calendar Year and Scorecard Name.

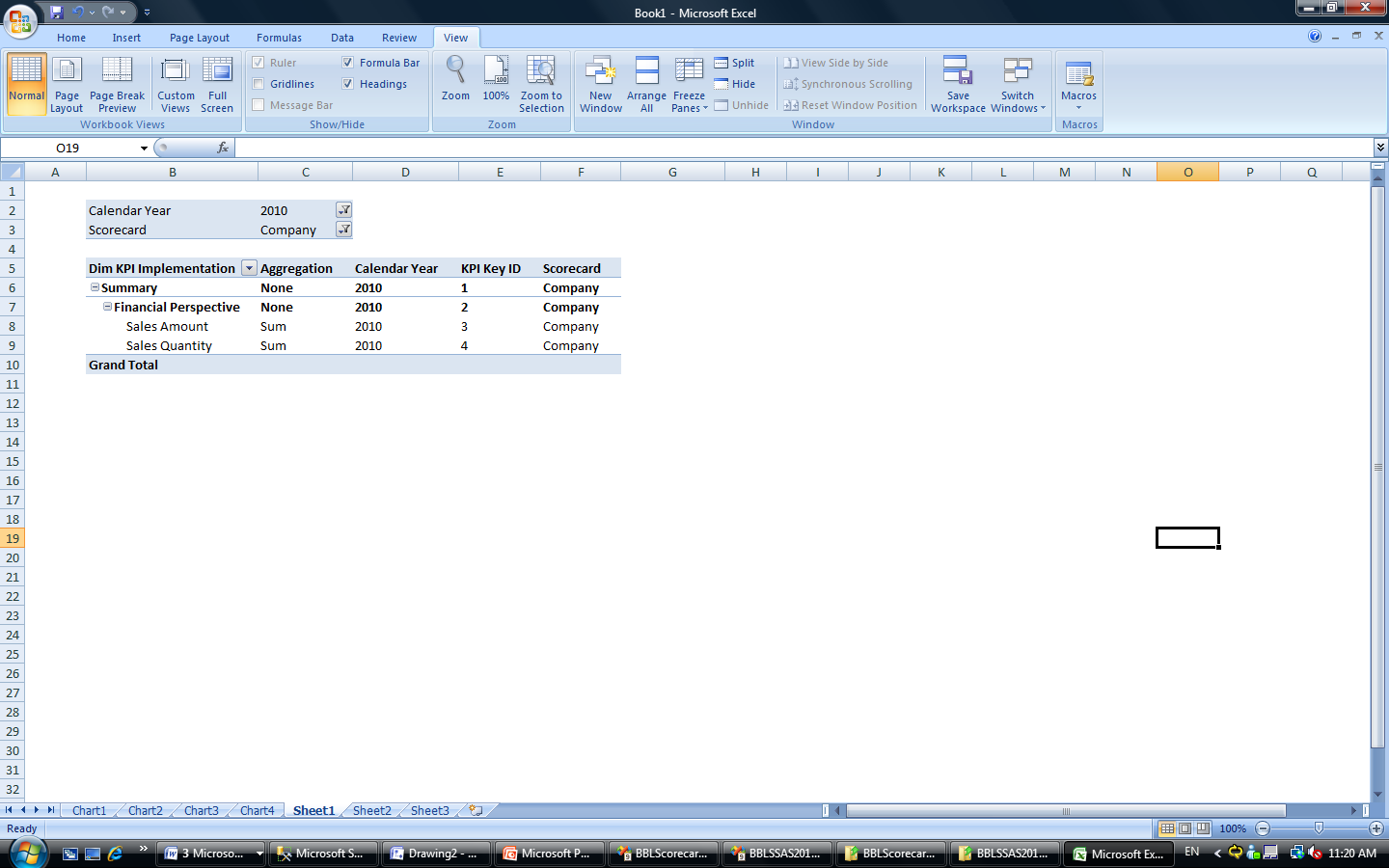


Figure 22

The Actual and Target measures relate to Dim KPI Implementation, Dim Date, and Dim Organization; the Weight measure relates only to Dim KPI Implementation. This design supports requirements of business units that need more than one scorecard instance, such as Region or Branch.

|  |  |
| --- | --- |
| **Scorecard (Scorecard Structure)** | **Dim Organization(Scorecard Instance)** |
| All Company | All Company |
| Business Unit #1 | Business Unit #1 |
| Region | Region #1 |
|  | Region #2 |
|  | Region #3 |

* 1. Report Design

The customer wanted to visualize the Balance Scorecard for every business unit so that they could see their performance against the company plan. Using this information, they would be able to adjust their tactics or strategies if the actual results did not meet the target. They wanted this information to be available through Reporting Services.

More than a thousand users were expected to access this report. Most look at different scorecard structures and different sets of data. The Balance Scorecard cube we generated should help reduce the number of report implementations to one.

* + 1. Report Parameters

The Balance Scorecard report includes five parameters:

* Organization – lists all members of the [Dim Organization] dimension.
* Scorecard Year – lists all [Scorecard Year] members from the [Dim KPI Implementation] dimension.
* Scorecard – lists all [Scorecard Year] scorecard members for the [Dim KPI Implementation] dimension.
* Data Year – lists all [Calendar Year] members from the [Dim Date] dimension.
* Data Month – queries all [Calendar Month] members from the [Dim Date] dimension sliced by Data Year parameter.
  + 1. Dataset

To create the dataset, we selected the Weight, Actual, Target, Score (Actual / Target) measures on columns, and then we selected all members of the parent-child hierarchy of the KPI Implementation dimension on row slices by all report parameters.

To ensure access to the MEMBER\_UNIQUE\_NAME and PARENT\_UNIQUE\_NAME property if parent-child grouping is implemented in the report, we added the following code to the MDX statement.

DIMENSION PROPERTIES

MEMBER\_CAPTION

,MEMBER\_UNIQUE\_NAME

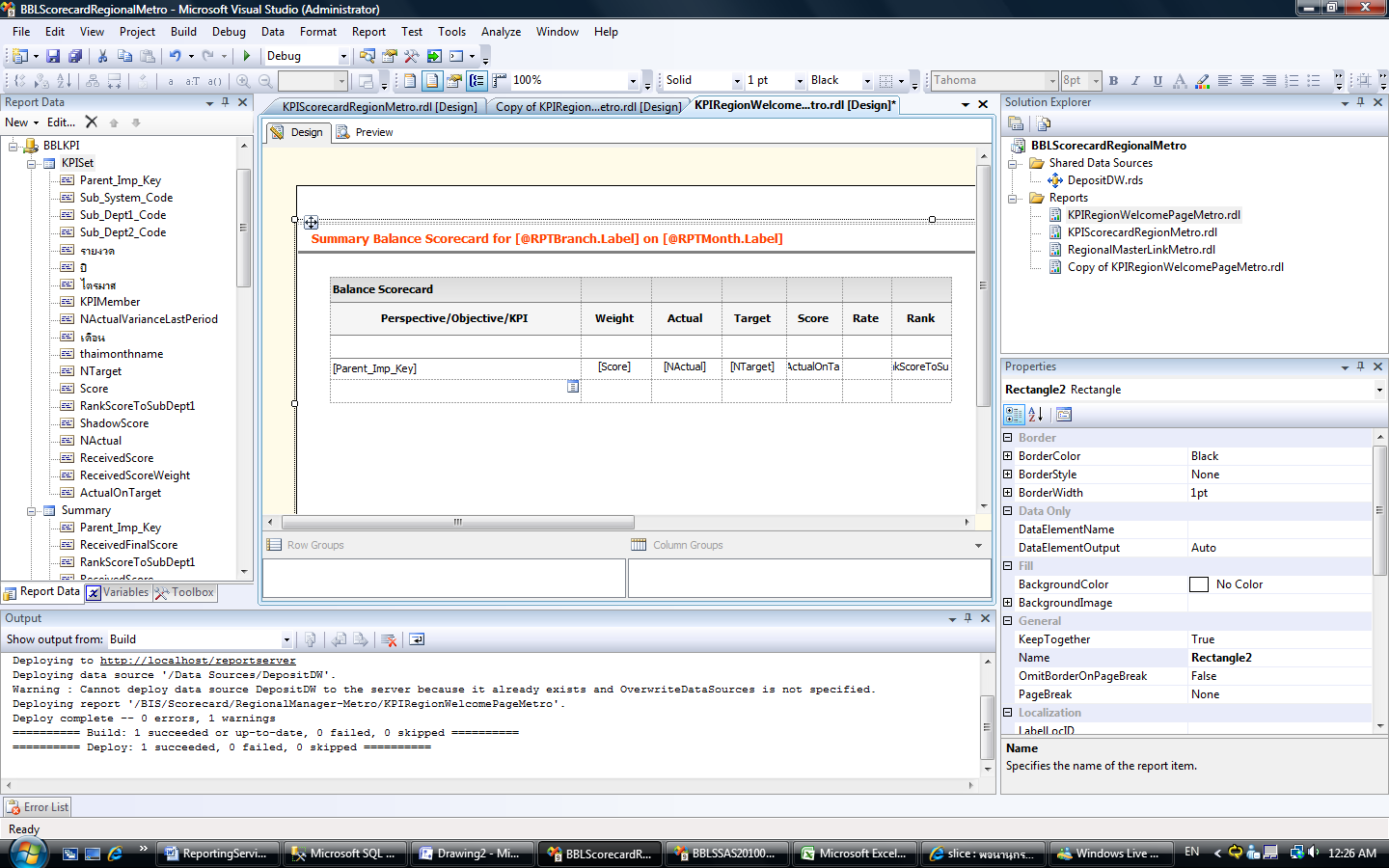
,PARENT\_UNIQUE\_NAME

,LEVEL\_NUMBER

ON ROWS

* + 1. Tablix

In this case, configuring the display in the Tablix data region was very simple. There is only one row group, which groups data by [Parent\_Imp\_Key.Uniquename] attribute in the [Dim KPI Implementation] dimension.



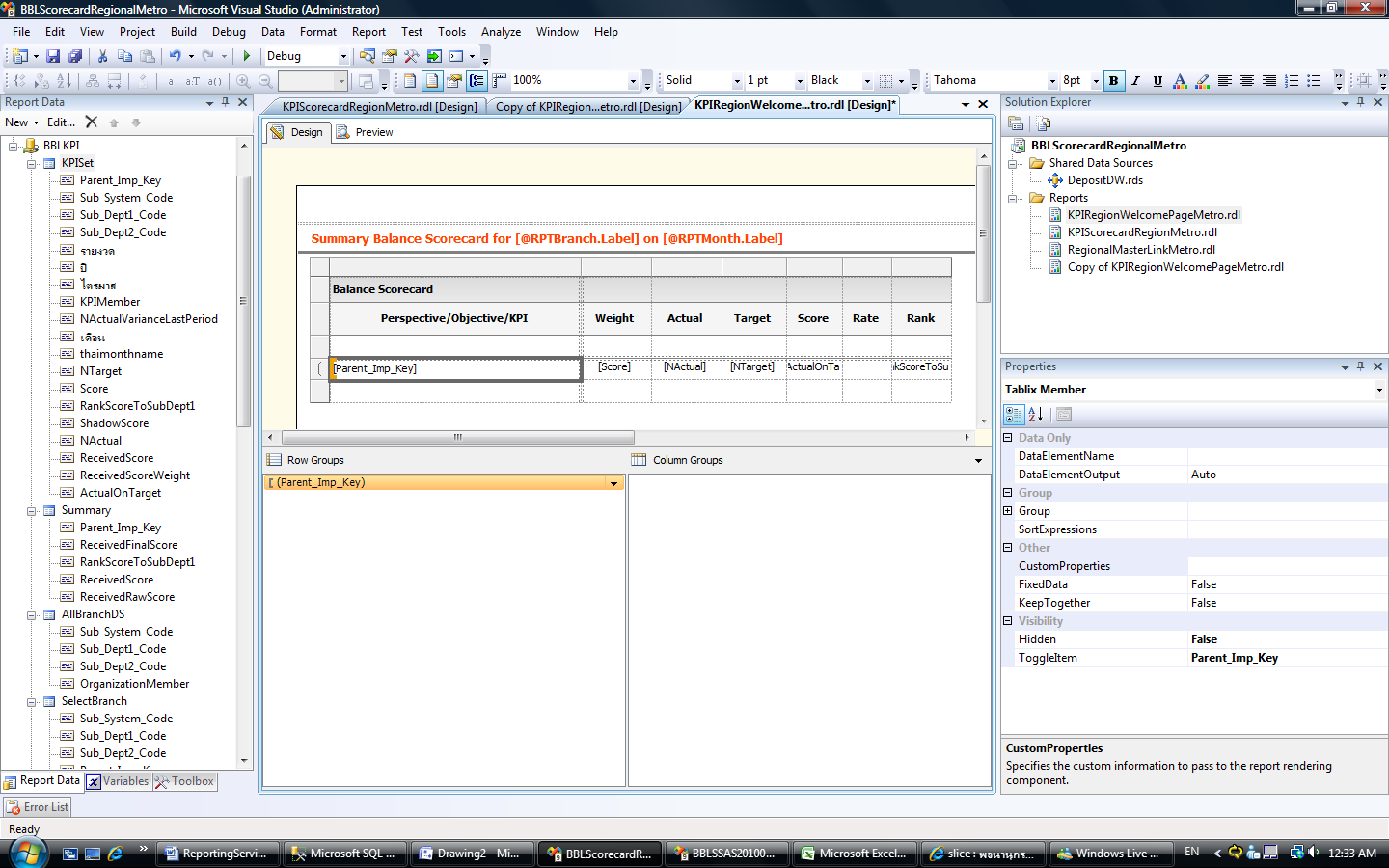


Figure 23

The **Recursive parent** property of the Parent\_Imp\_Key group must be set to [Parent\_Imp\_Key.ParentUniqueName]. This setting is on the **Advanced** page of the **Group Properties** dialog box. Furthermore, you should set this property should be set only if you have a recursive hierarchy in the data. If you do not, it will just slow down the report and potentially surface runtime errors due to data type mismatch and failed comparisons between group expressions and their recursive parent expression.

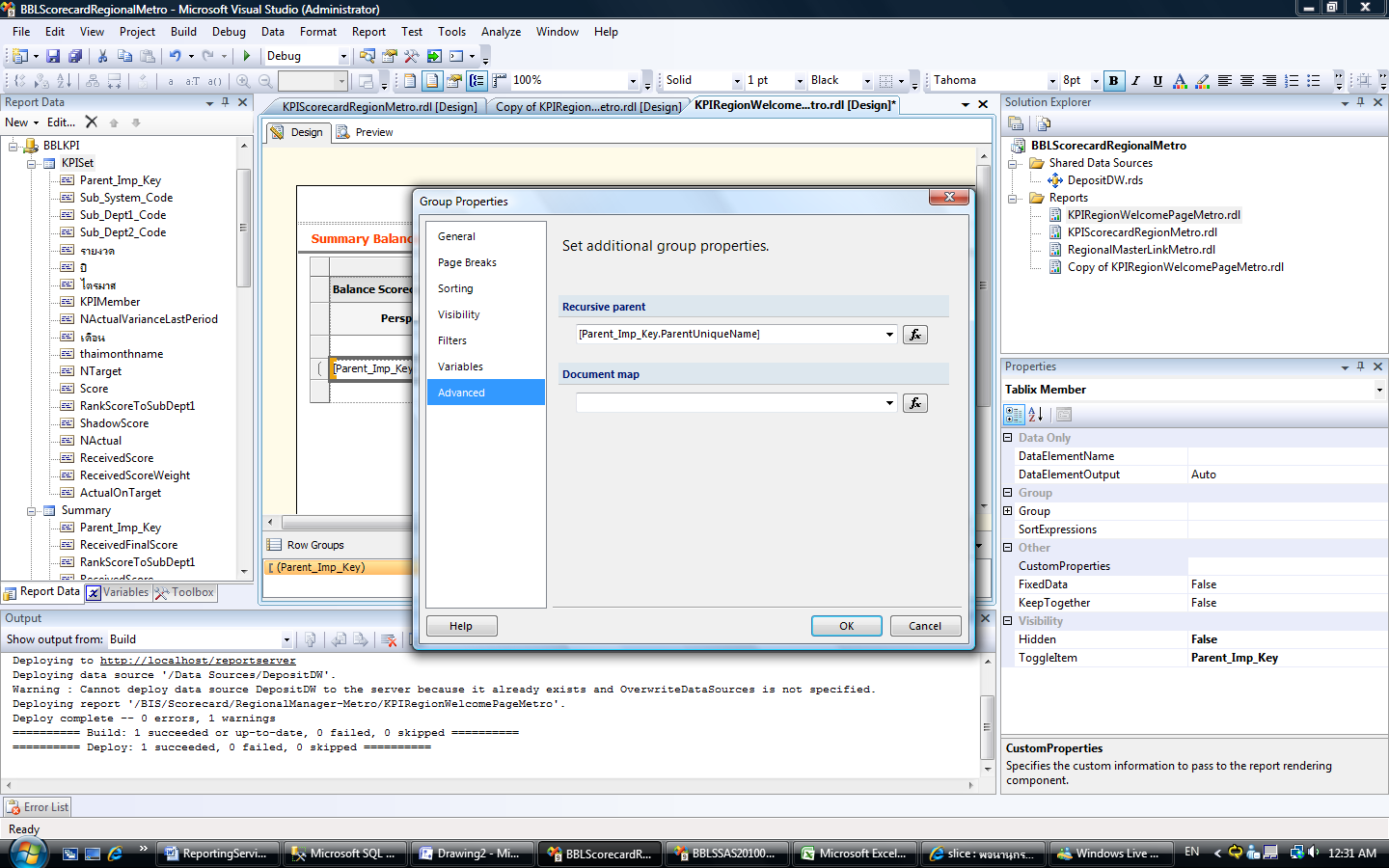


Figure 24

The **Rate** column in the Tablix data region displays different colors that indicate the status of each KPI. To set up this behavior, add an expression into the background image property of the text box. The expression depends on the business requirements of the company.

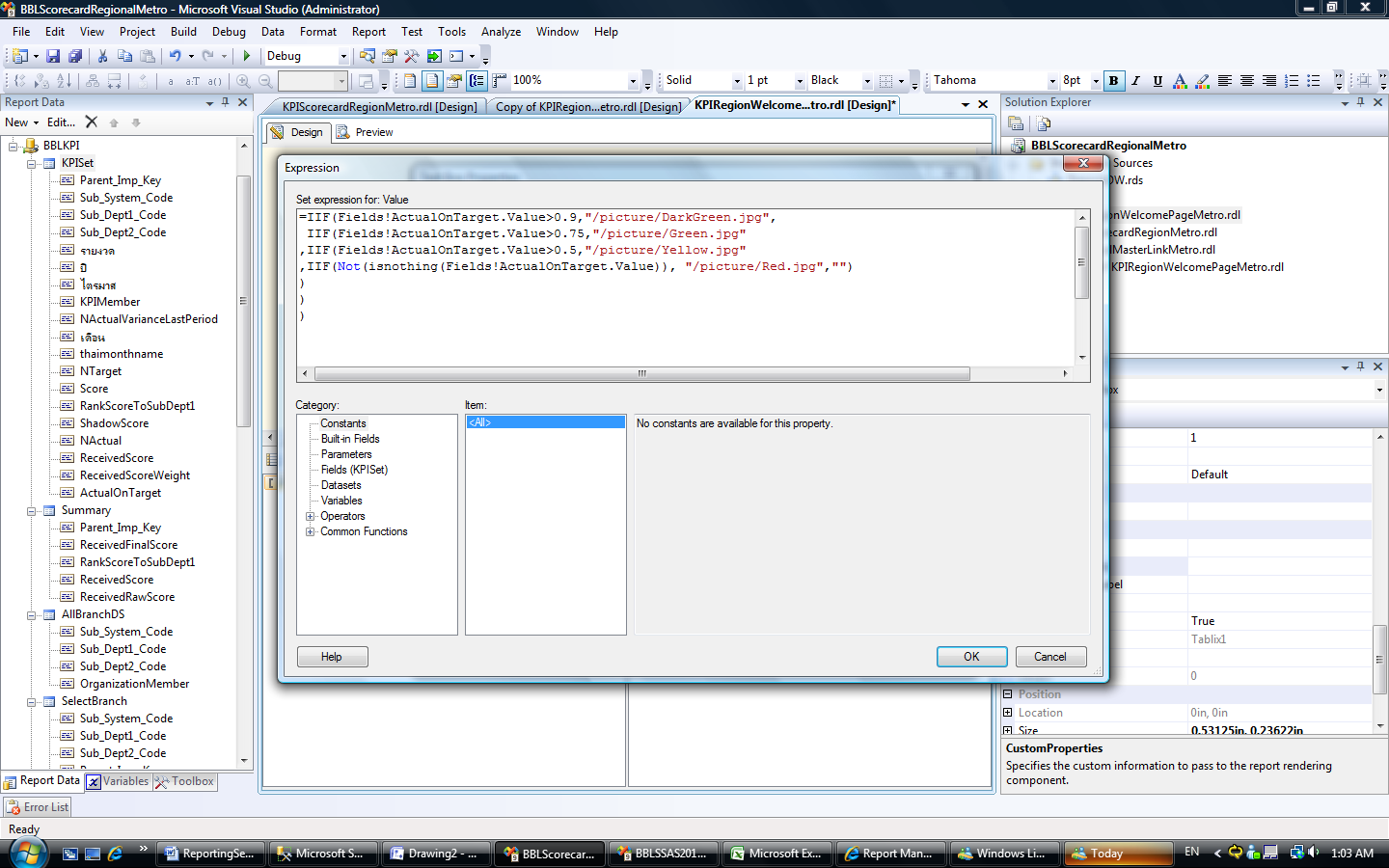


Figure 25

* + 1. Result

Figure 26 shows an example of the results.

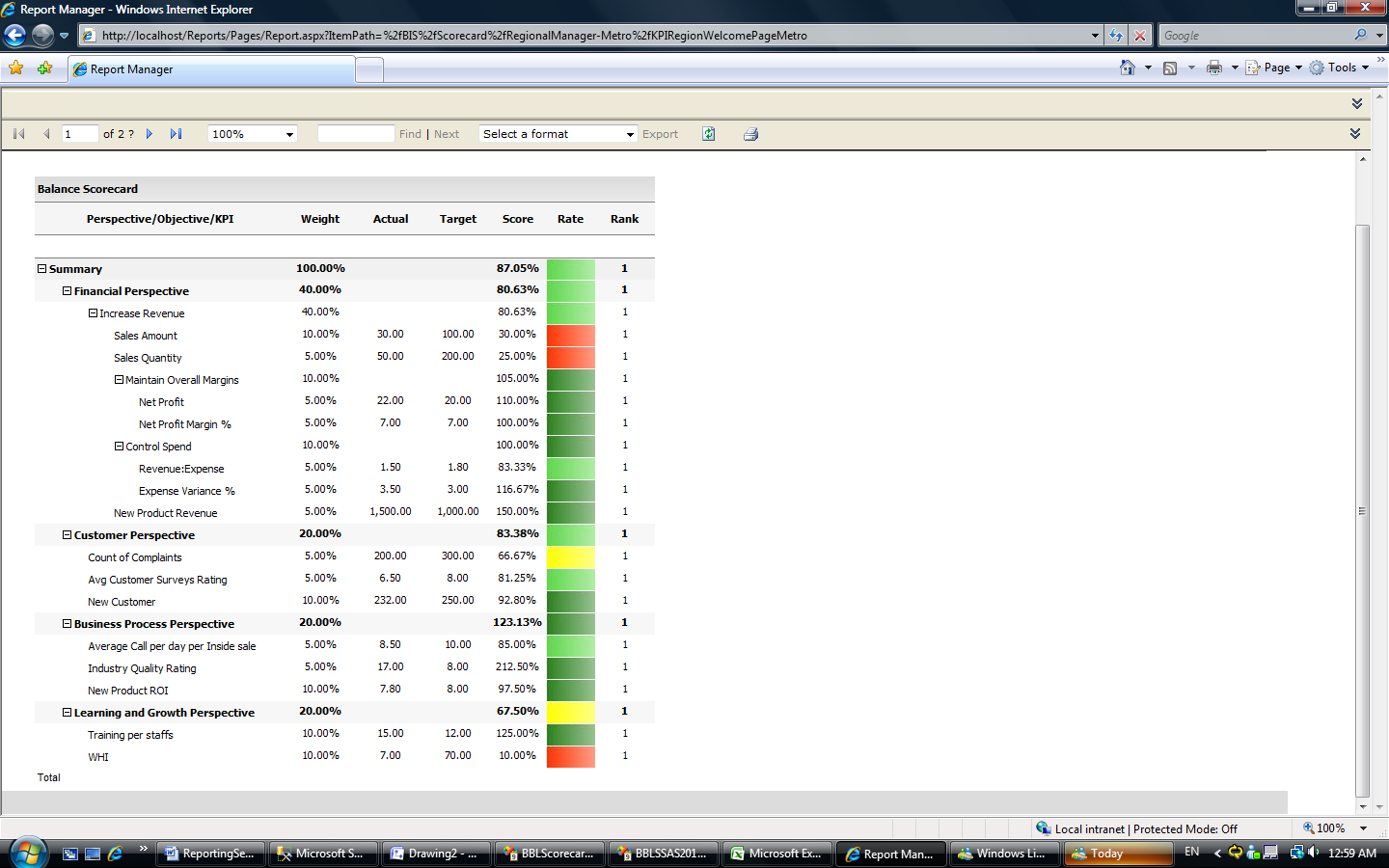


Figure 26

The scorecard information can be filtered by report parameters. These parameters enable all users to generate the Balance Scorecard report for their business unit and organization in each year without implementing new report.

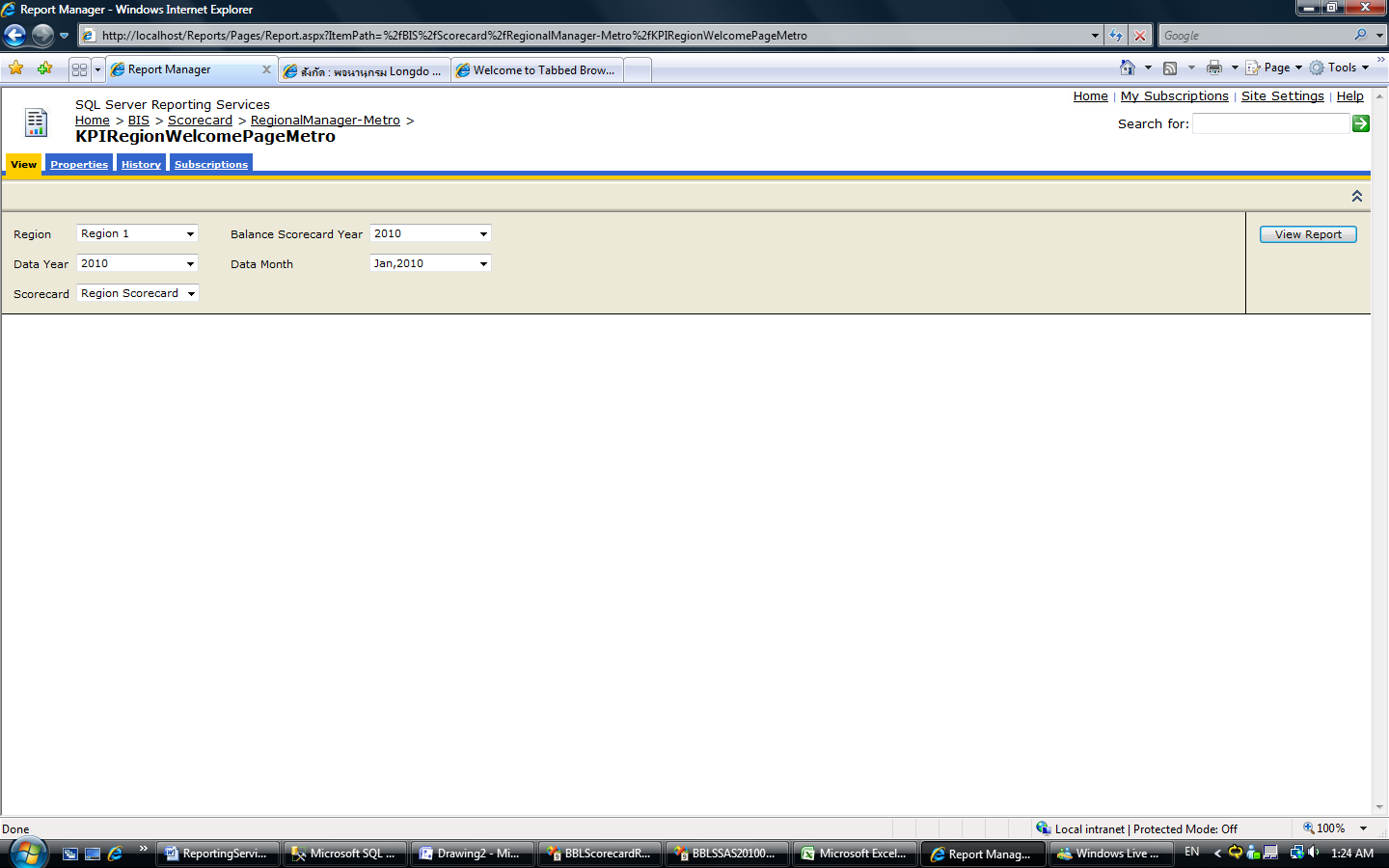


Figure 27

* 1. Additional Tips

This section contains other tips about working with the Business Scorecard. These are tips we learned from our work on this business case.

* + 1. Improvement Areas

We also learned the following while working on this customer case:

* Implement a multigranularity mechanism for both Actual and Target measures on the Dim Organization and Dim Date dimension. This technique enables users to view assigned values into Actual and Target measures in any dimension level, because some KPIs need to be aggregated values from the lower level and some need to assign the value directly. For example, the sales amount for the region level sums up from the branch level, but the electrical/water charge saving in each offices takes its value directly from the regional office (regional level). Another example is that sales amount is be a daily-assigned value, but industry quality rating is a yearly-assigned value.
* Tailor the complexity of the score calculation formula to the needs of the business. This scorecard example is implemented with a simple score calculation formula (Score = % of Actual divided by Target). In the real solution, the score calculation formula we implemented was much more complex. The level of complexity depends on the customer business requirements: Some KPIs are defined with the rule that “bigger is better,” and others are defined with the rule that “smaller is better”. For example, Sale Amount falls into the “bigger is better” category (we want to earn more money), and Turnover Rate falls into “smaller is better” (we want to lose fewer customers). This formula should be one of the properties of members in Dim KPI Implementation dimension; it enables users to set this property via an ETL cycle.
* The KPI unit should be one of the member properties of the Dim KPI Implementation members for proper rendering.
  + 1. Ranking

One of the strongest points of this Balance Scorecard solution is the presence of all of the Balance Scorecard information in the Balance Scorecard cube. This enables developers to easily improve the Balance Scorecard cube by creating new calculated measures on top of the existing structure, thereby enabling users to analyze data in the Balance Scorecard cube.

This section shows how to create a calculated measure to track and display the rank of all branches based on the Summary score. It also provides examples of ways you can include this new measure in your reports. These reports can help drive the competitive environment in the organization.

Here is the calculated measure creation code. This measure contains the ranking data of the Sub Department 3 member between all Sub Department 3 members in the in the same Sub Department 2 member. (The branch business unit is in the Sub Department 3 attribute.)

|  |
| --- |
| CREATE MEMBER CURRENTCUBE.[Measures].RankScoreToSubDept2  AS IIF([Measures].[ActualOnTarget] <> 0,  Rank (  [Dim Organization].[OrganizationHierarchy1].CurrentMember  ,NonEmpty(  Descendants(  Ancestor([Dim Organization].[OrganizationHierarchy1].CurrentMember  ,[Dim Organization].[OrganizationHierarchy1].[Sub Dept2 Code])  ,[Dim Organization].[OrganizationHierarchy1].CurrentMember.Level)  ,[Measures].[ActualOnTarget])  ,[Measures].[ActualOnTarget])  ,NULL),  FORMAT\_STRING = "Standard",  VISIBLE = 1 , DISPLAY\_FOLDER = 'Calculated Measure' , ASSOCIATED\_MEASURE\_GROUP = 'Fact Score Weight'; |

After you implement the code, add the new calculated measure into the Balance Scorecard report layout. Figure 28 shows a report that contains the new measure, Rank.

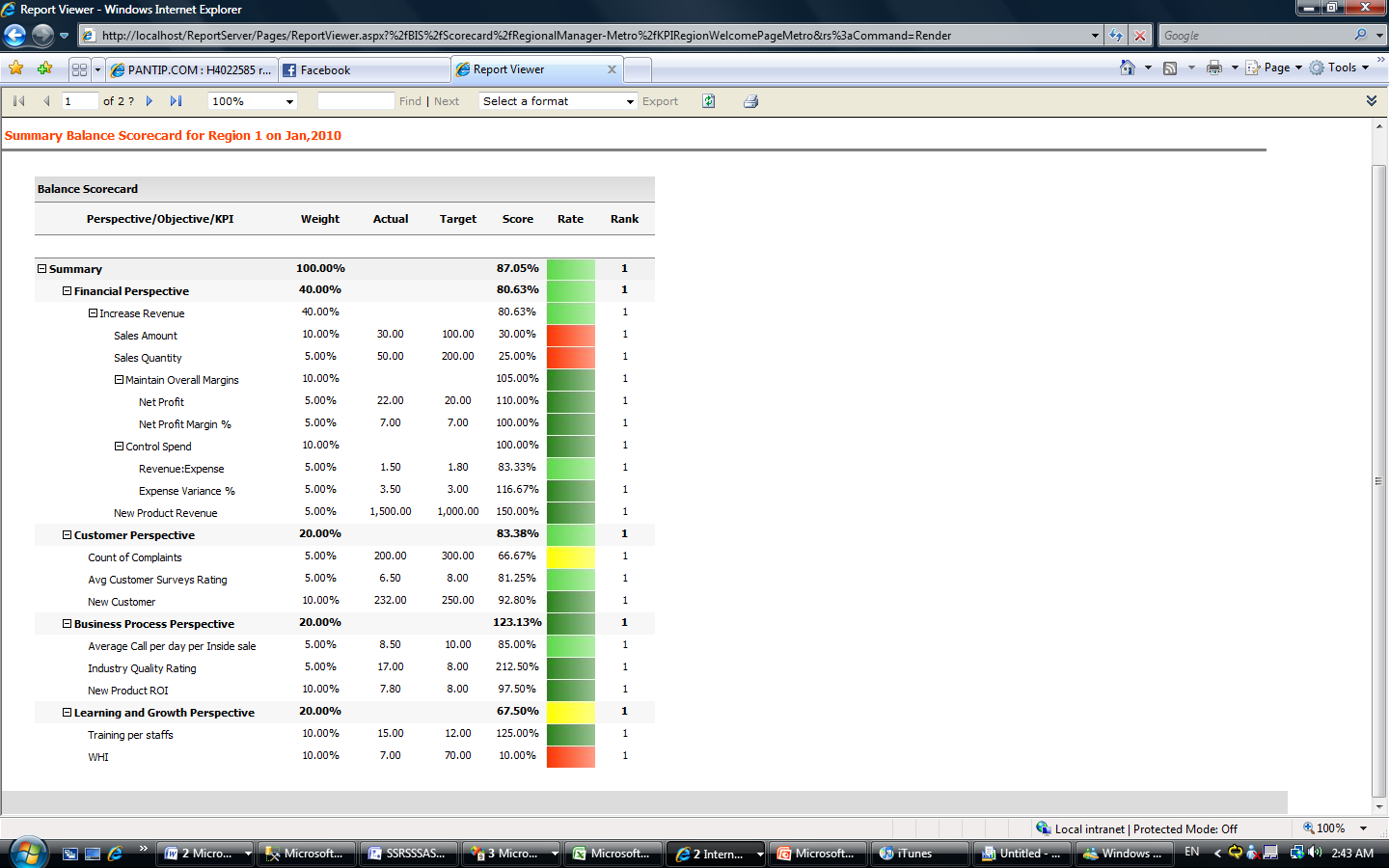


Figure 28

You can also create a new display in the Tablix data region that displays the lists of all branches ranked by Summary score.

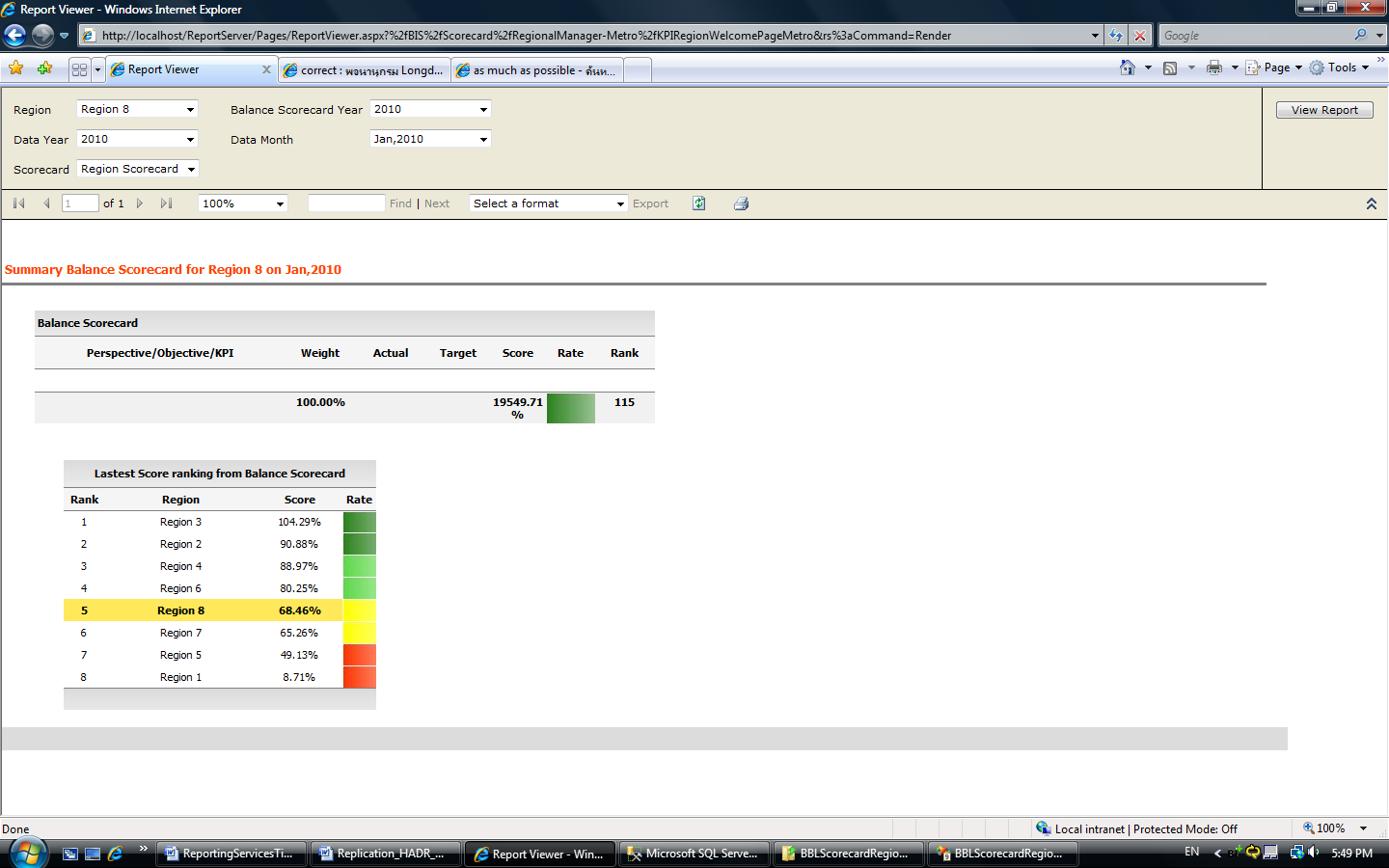


Figure 29

* + 1. Gauges

One of the new features in SQL Server 2008 Reporting Services is the Gauge data region. Adding Gauge data regions can improve the look and feel of your Balance Scorecard report. There are two types of gauges: radial and linear. Figure 30 shows a radial gauge using the Receive Score measure.

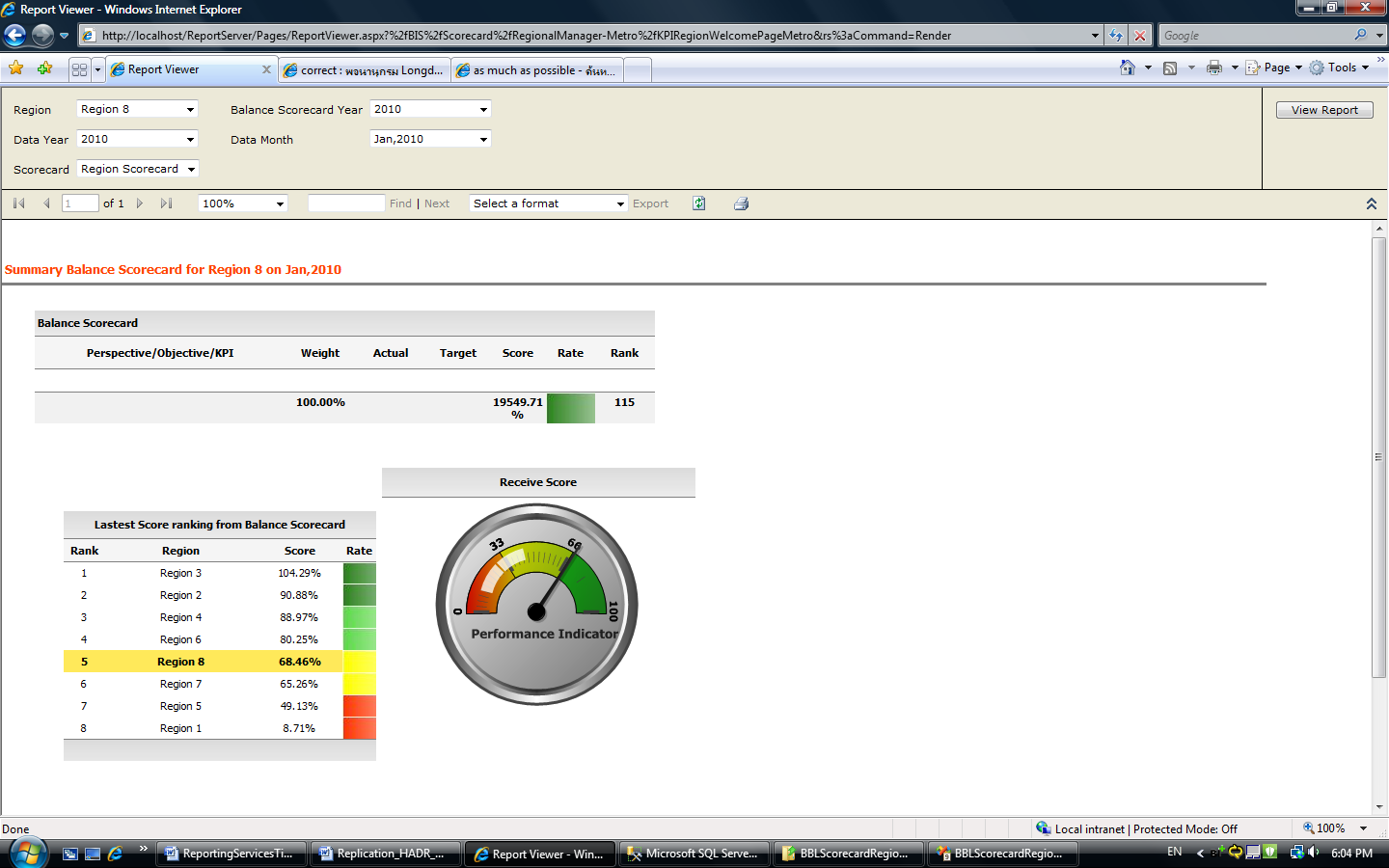
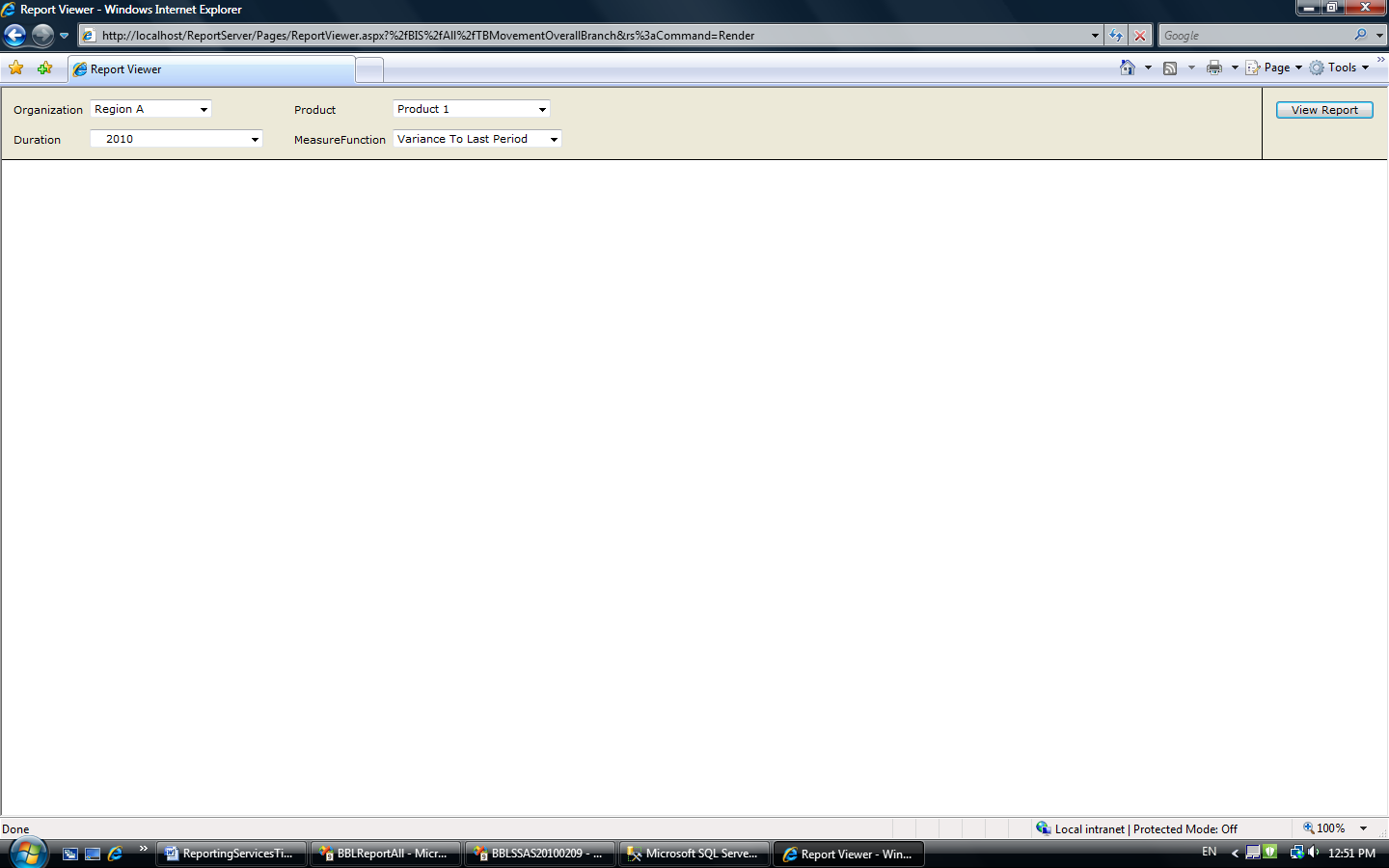


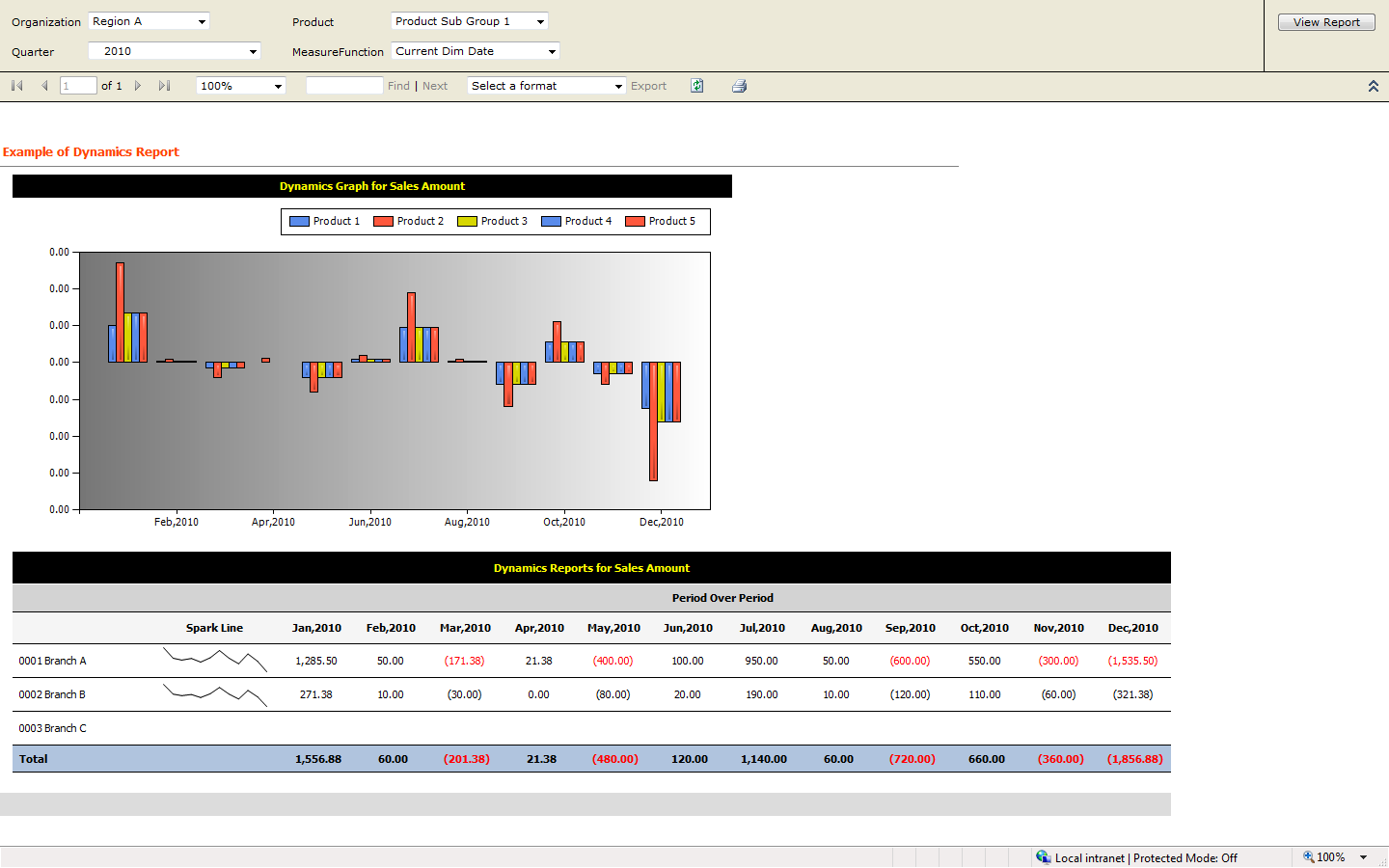
Figure 30

1. Implementing Dynamic Reports

During the requirement gathering phase, Microsoft collected different report requirements from different business units. When we analyzed the report requirements, we found a number of reports that shared the same report layout, differing only in row grouping or measures. We were able to use Reporting Services to create a dynamic report that consolidated the number of reports to a single report layout that many groups could share, even though their requirements were not identical.

Figure 31 shows an example of a dynamic report. This section shows how to implement it.

****



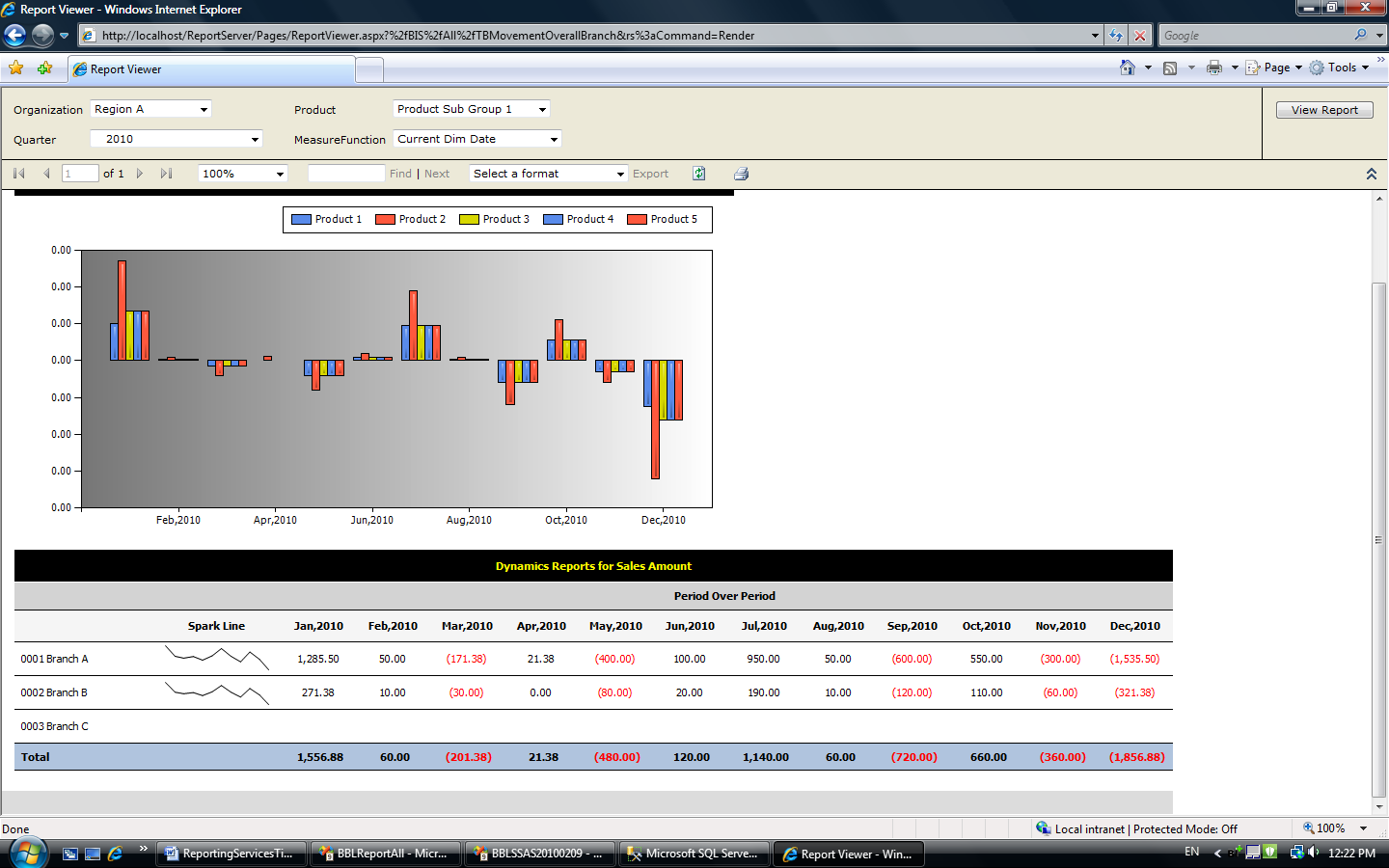


Figure 31

You can display the Sales Amount measure in two formats: the Tablix data region and a graph.

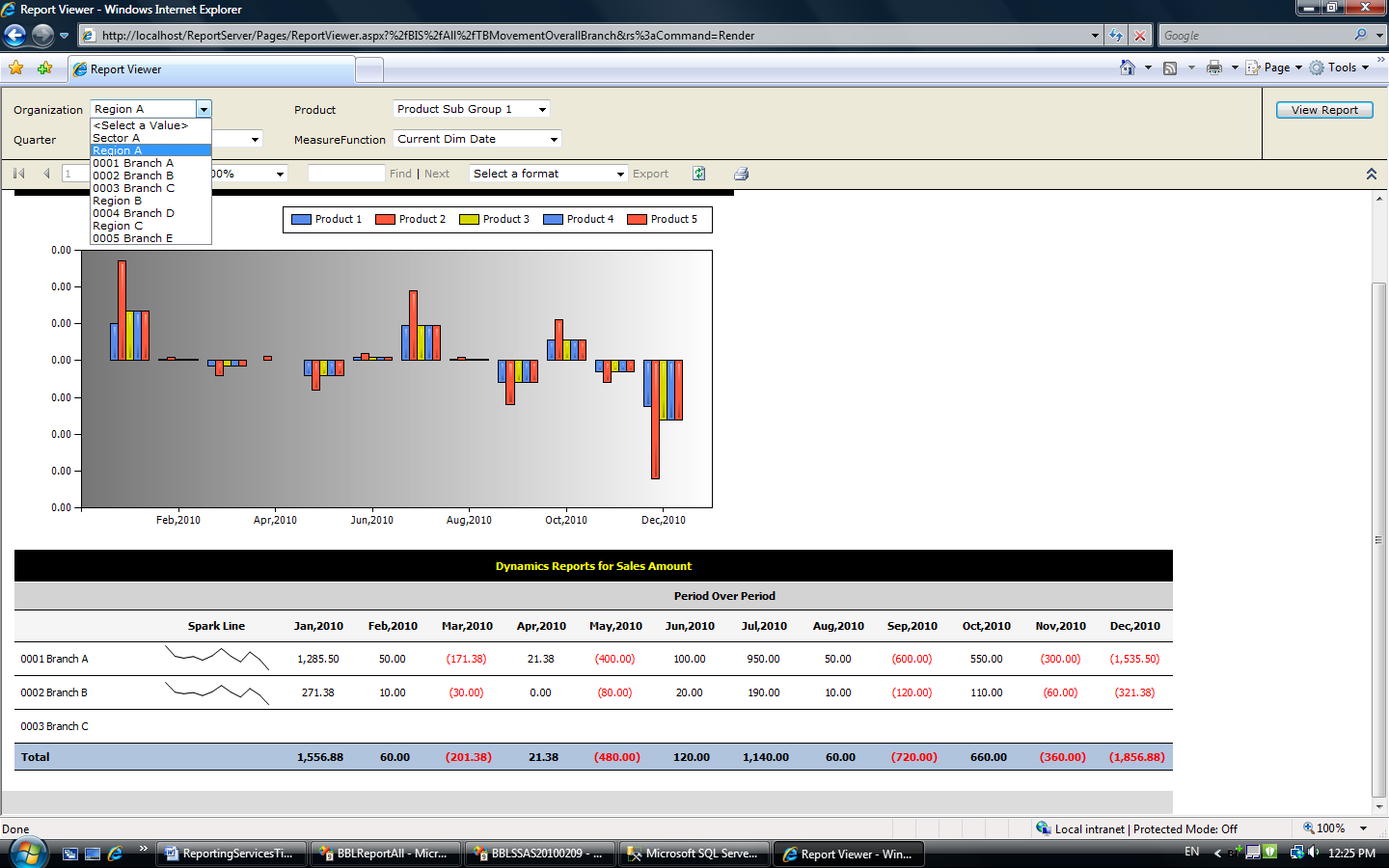


Figure 32

The Organization parameter lists all members under the Branch Structure hierarchy. The Tablix data region in this report is grouped by all children of selected member as a row group. For example, when users select Region A, the Tablix data region lists all branches under Region A as a row group.

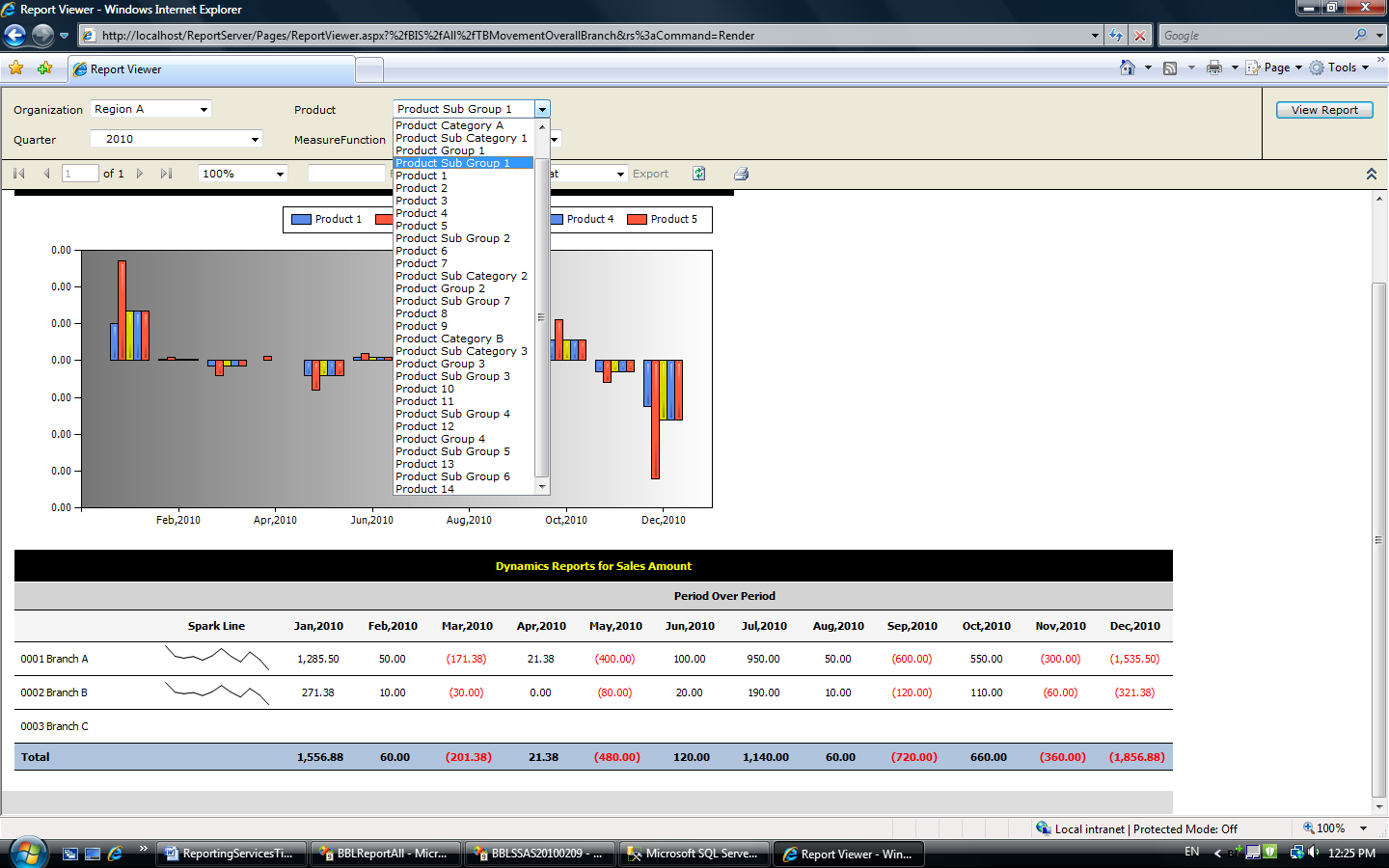


Figure 33

The Product parameter lists all members under the Product hierarchy. A graph in this report is grouped by all products under the selected member as a graph series. For example, when users select Product Sub Group A, a graph displays all products under Product Sub Group 1 as a graph series.

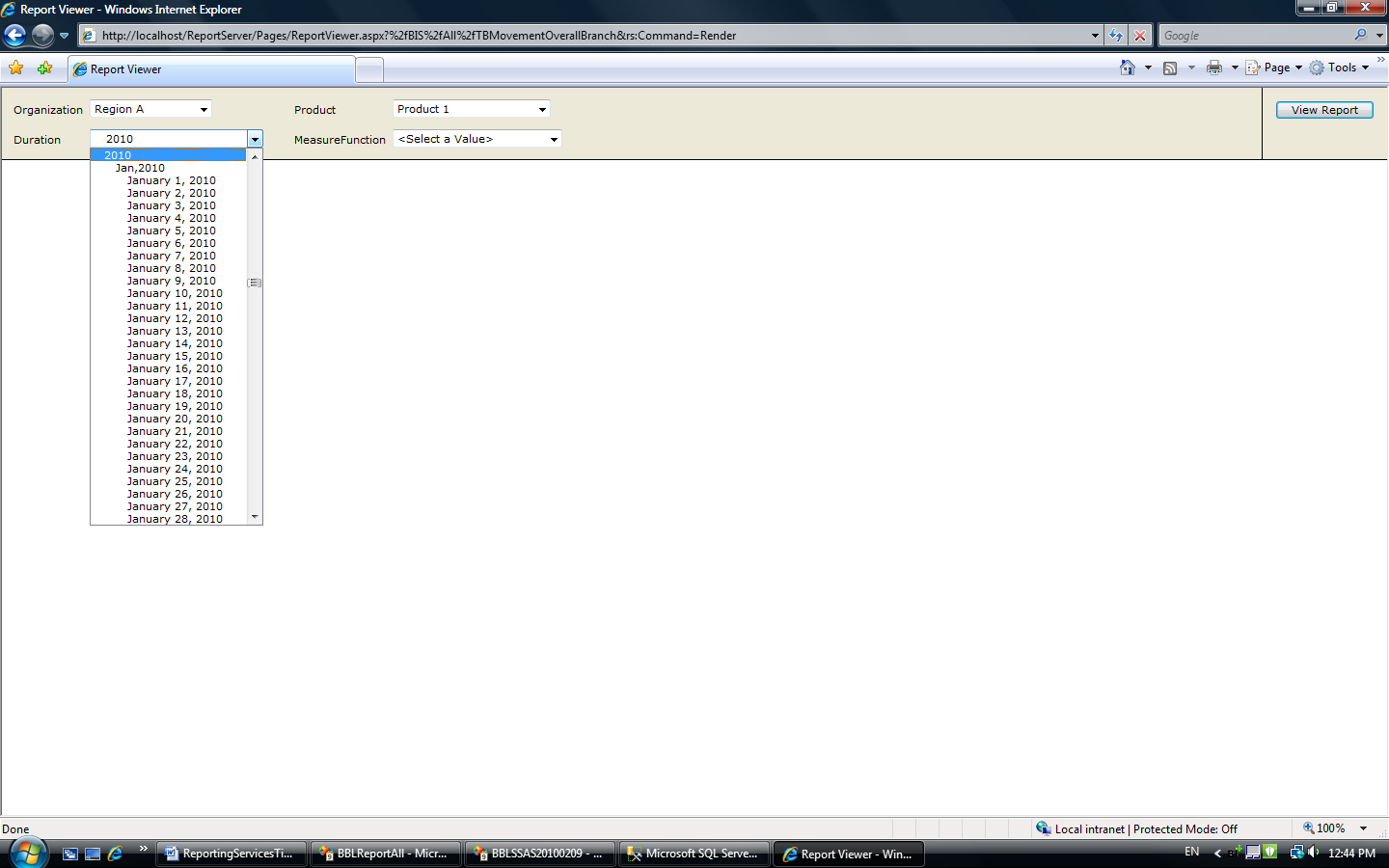


Figure 34

The Duration parameter lists all members under the Month hierarchy. A Tablix data region in this report is grouped by all children of selected member as a column group, and a graph is grouped by all children of the selected member as graph category. For example, when users select 2010, the Tablix data region lists all months in 2010 as its column group and a graph displays all months in 2010 as a graph category.

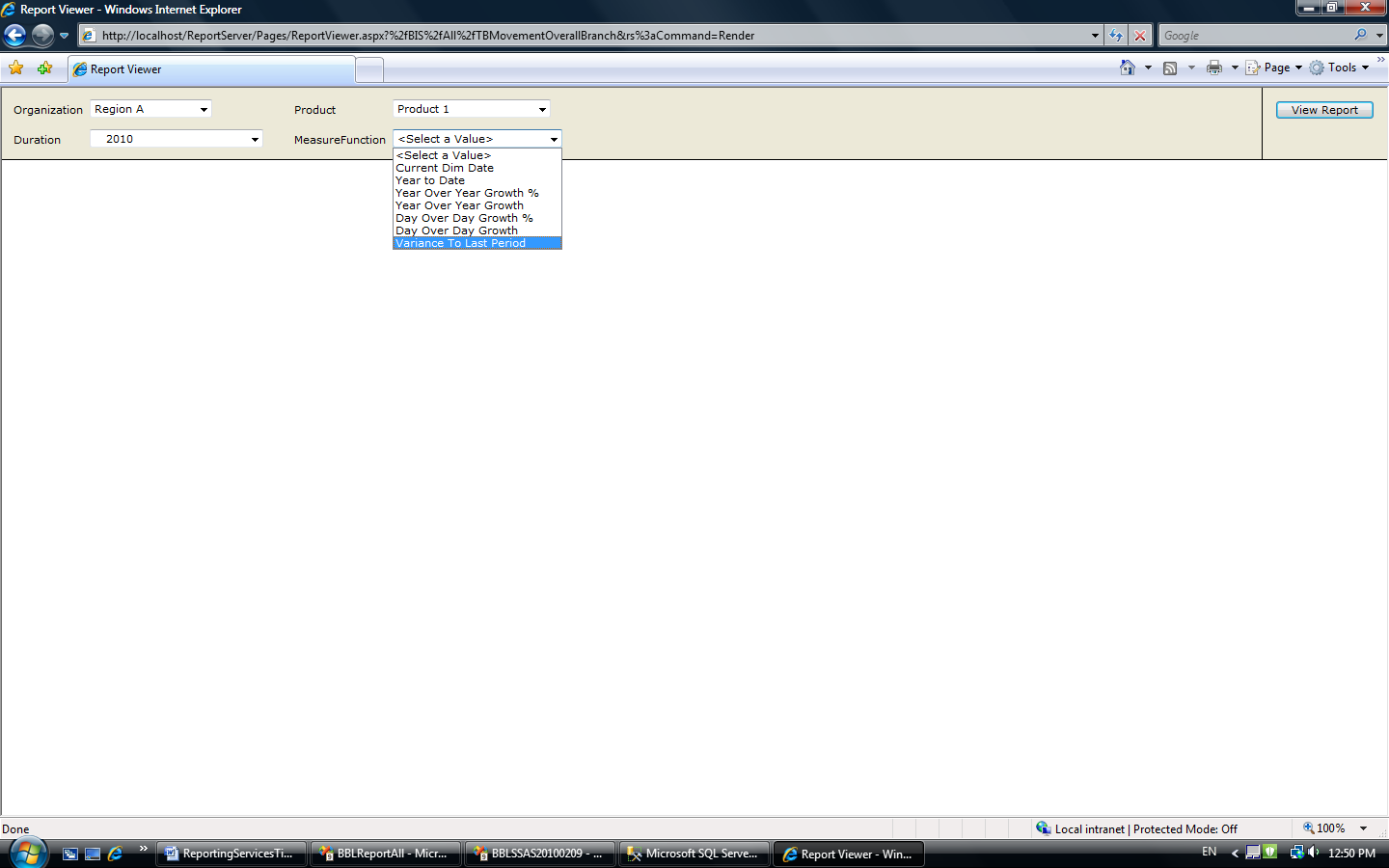


Figure 35

The Measure function parameter lists the functions that users can use for their Sales Amount calculations. For example, if a user selects Variance To Last Period, a report and the Tablix data region display Variance To Last Period of Sales Amount as a value. (Current Dim Date is Sales Amount.)

Each row in the Tablix data region should display a small summary graph (sparkline) before it displays a raw value.

* 1. OLAP Design

This section uses the Sales cube illustrated in Figure 5 to implement the dynamic report. However, the Month hierarchy of Dim Date is used, rather than the Quarter Hierarchy. Figure 36 shows the structure of Month hierarchy.



Figure 36

* + 1. Time Intelligence

To support the measure function business requirement, we used time intelligence in SQL Server 2008 Analysis Services (for more information, see [Defining Time Calculations using the Business Intelligence Wizard](http://msdn.microsoft.com/en-us/library/ms175440.aspx) (http://msdn.microsoft.com/en-us/library/ms175440.aspx) in SQL Server 2008 R2 Books Online). We created time intelligence on the Month Dim Date Calculations hierarchy. We included the Sales Amount measure in this time intelligence.

Figure 37 illustrates a time intelligence hierarchy structure named Month Dim Date Calculation.

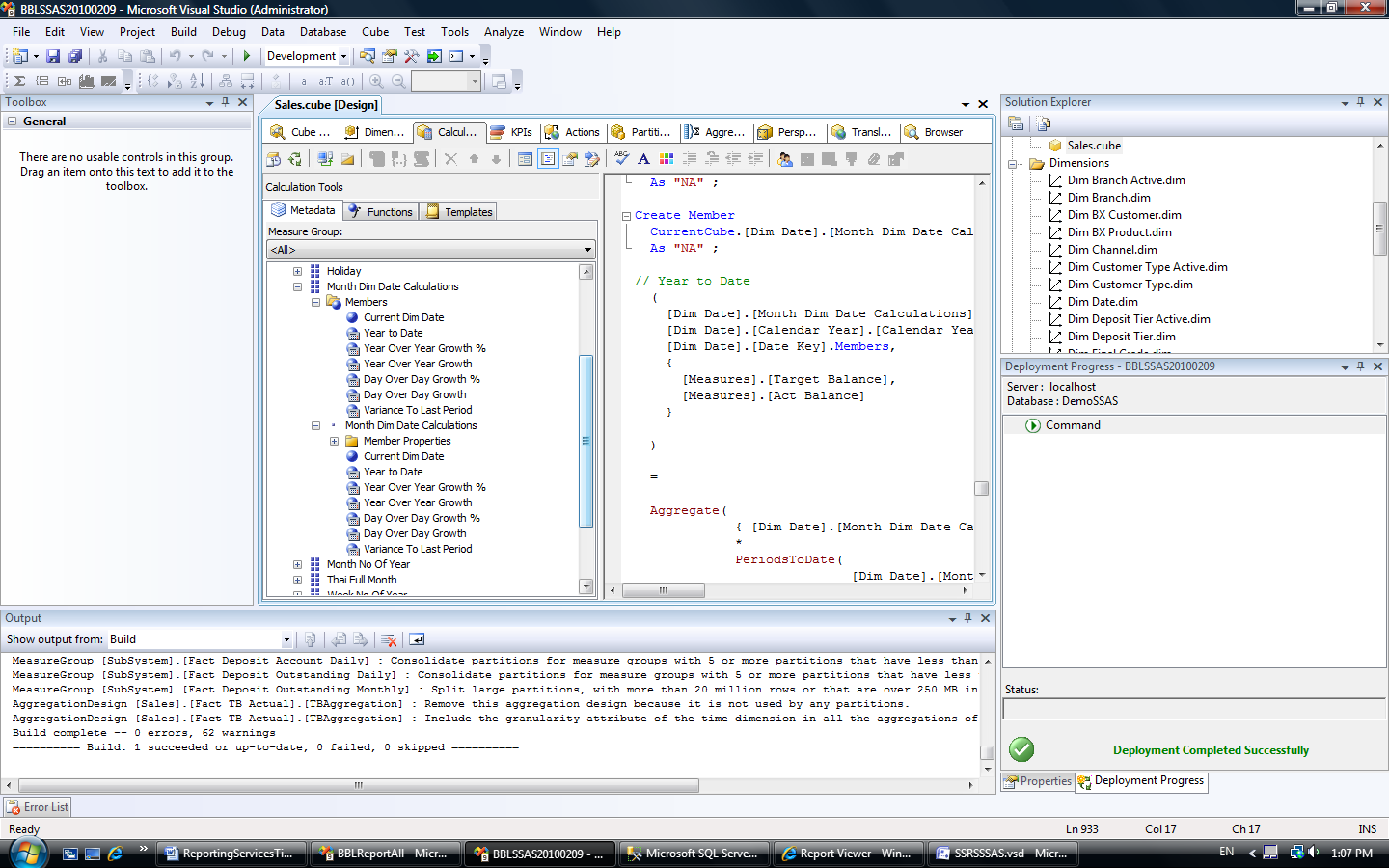


Figure 37

* 1. Report Design

Our report design contained four report parameters. This section lists the parameters and datasets for the Tablix data region and the graph.

* + 1. Report Parameters

To create the report parameters, we created the following report parameters with the same characteristics as described above:

* Organization from Branch Structure Hierarchy
* Product from Product Hierarchy.
* Duration from Month Hierarchy.
* Measure function from Month Dim Date Calculations hierarchy.

All members should be selected and listed under the retrieved hierarchy.

The following code shows the data source of the report parameters.

|  |
| --- |
| (Example – data source of Product parameter)  WITH  MEMBER [Measures].[ParameterCaption]  AS [Dim Product Active].[Product].CURRENTMEMBER.MEMBER\_CAPTION  MEMBER [Measures].[ParameterValue]  AS [Dim Product Active].[Product].CURRENTMEMBER.UNIQUENAME  SELECT  {  [Measures].[ParameterCaption]  , [Measures].[ParameterValue]  }  ON COLUMNS  , [Dim Product Active].[Product].ALLMEMBERS  ON ROWS  FROM [Sales] |

* + 1. Dataset

To meet the customer’s reporting requirements, we created two datasets: one for the Tablix data region and another for the graph. Although the Tablix data region and the graph can share the same dataset, we encountered an issue when we created a Tablix data region and a graph from a single dataset and tried to do grouping.

Datasets for the Tablix Data Region

To create the datasets for the Tablix data region:

* Select all children of the selected value of the organization and duration parameters.
* Slice the retrieved data set on the selected value of the measure function parameter and the selected value of the Product parameter.
* Create the measure BranchMember, BranchLabel, DurationMember, DurationLabel to use as Grouping and display field.
* Select [Measures].[Sale Amount].

|  |
| --- |
| (  Query parameter – Report parameter mapping  @DimBranch - Organization  @DimProduct - Product  @DatePeriod - Duration  @DimMeasureFunction - Measure Function  )  WITH  MEMBER BranchMember AS [Dim Branch Active].[Cluster].currentmember.uniquename  MEMBER BranchLabel AS [Dim Branch Active].[Cluster].currentmember.member\_caption  MEMBER DurationMember AS [Dim Date].[Month].currentmember.uniquename  MEMBER DurationLabel AS [Dim Date].[Month].currentmember.member\_caption  SELECT  NON EMPTY  {  BranchLabel  ,BranchMember  ,DurationLabel  ,DurationMembel  ,[Measures].[Sale Amount]  } ON COLUMNS  ,NON EMPTY  {  STRTOMEMBER(@DatePeriod).children  \*  STRTOMEMBER(@DimBranch).children  }  DIMENSION PROPERTIES  MEMBER\_CAPTION  ,MEMBER\_UNIQUE\_NAME  ON ROWS  FROM [Sales]  WHERE (STRTOMEMBER(@DimMeasureFunction), STRTOMEMBER(@DimProduct)) |

Datasets for the Graph

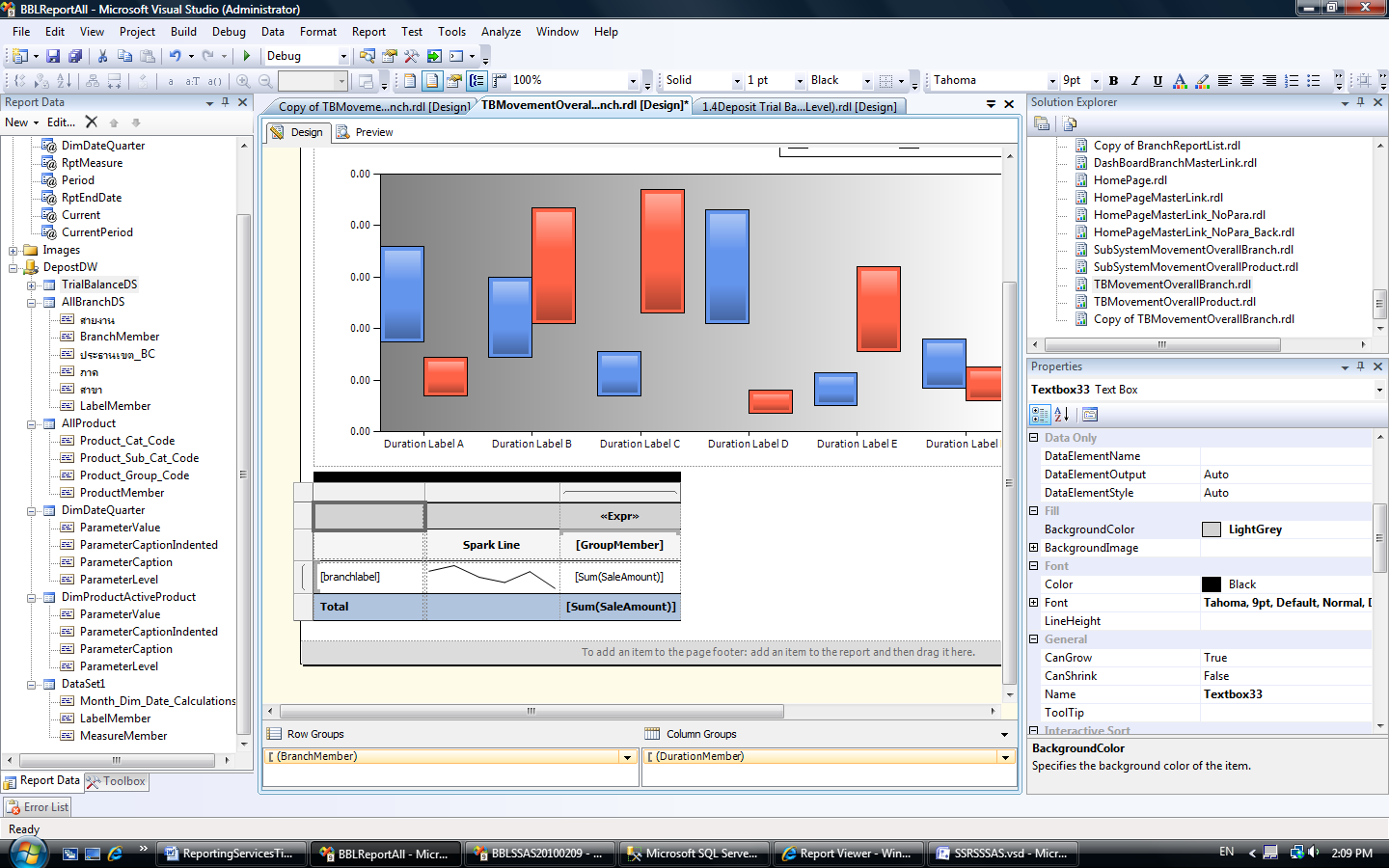
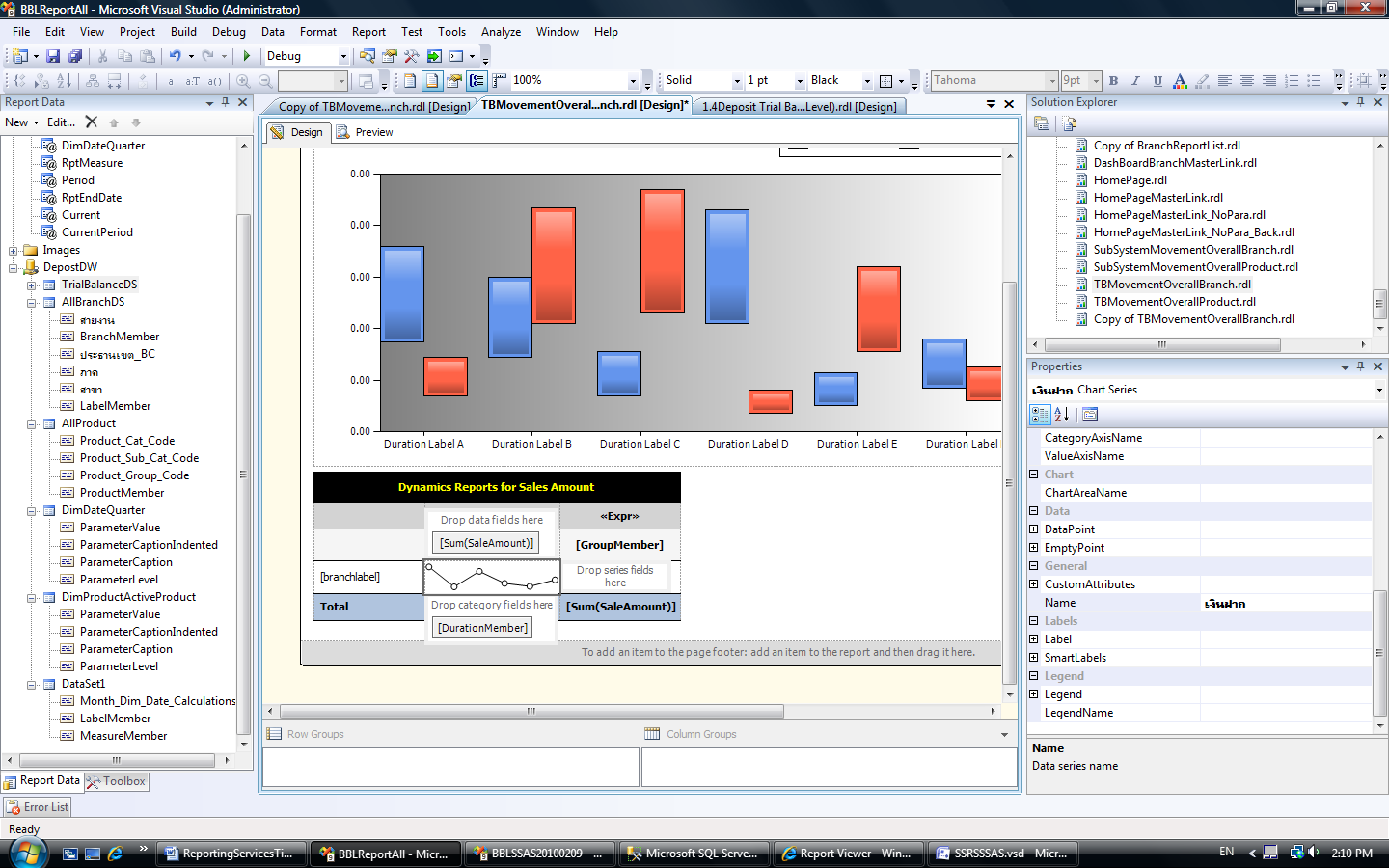
To create the datasets for the graph:

* Select all children of the selected value of the duration parameter.
* Select all products that are descendants of the selected value of the product parameter by using the NonEmpty function.
* Slice retrieved data set on the selected value of the measure function parameter and the selected value of the Organization parameter.
* Create the measure DurationMember, DurationLabel to use as a grouping and display field. (You do not need to create productmember and productlabel from the Product hierarchy because only the Product Code level is selected, meaning that Product is not a dynamic dimension.)
* Select [Measures].[Sale Amount].

|  |
| --- |
| (  @DimBranch - Organization  @DimProduct - Product  @DatePeriod - Duration  @DimMeasureFunction - Measure Function  )  WITH  MEMBER DurationMember AS [Dim Date].[Month].currentmember.uniquename  MEMBER DurationLabel AS [Dim Date].[Month].currentmember.member\_caption  SELECT  NON EMPTY  {  ,DurationLabel  ,DurationMembel  ,[Measures].[Sale Amount]  } ON COLUMNS  ,NON EMPTY  {  {  NonEmpty( [Dim Product Active].[Product].[Product Code].allmembers  ,STRTOSET(@DimProduct))  }  \*  STRTOMEMBER(@DatePeriod).children  \*  STRTOMEMBER(@DimBranch).children  }  DIMENSION PROPERTIES  MEMBER\_CAPTION  ,MEMBER\_UNIQUE\_NAME  ON ROWS  FROM [Sales]  WHERE (STRTOMEMBER(@DimMeasureFunction), STRTOMEMBER(@DimBranch)) |

* + 1. Tablix

This section discusses the Tablix data region shown in Figure 38. It contains a sparkline showing sales amounts for each branch.

* *

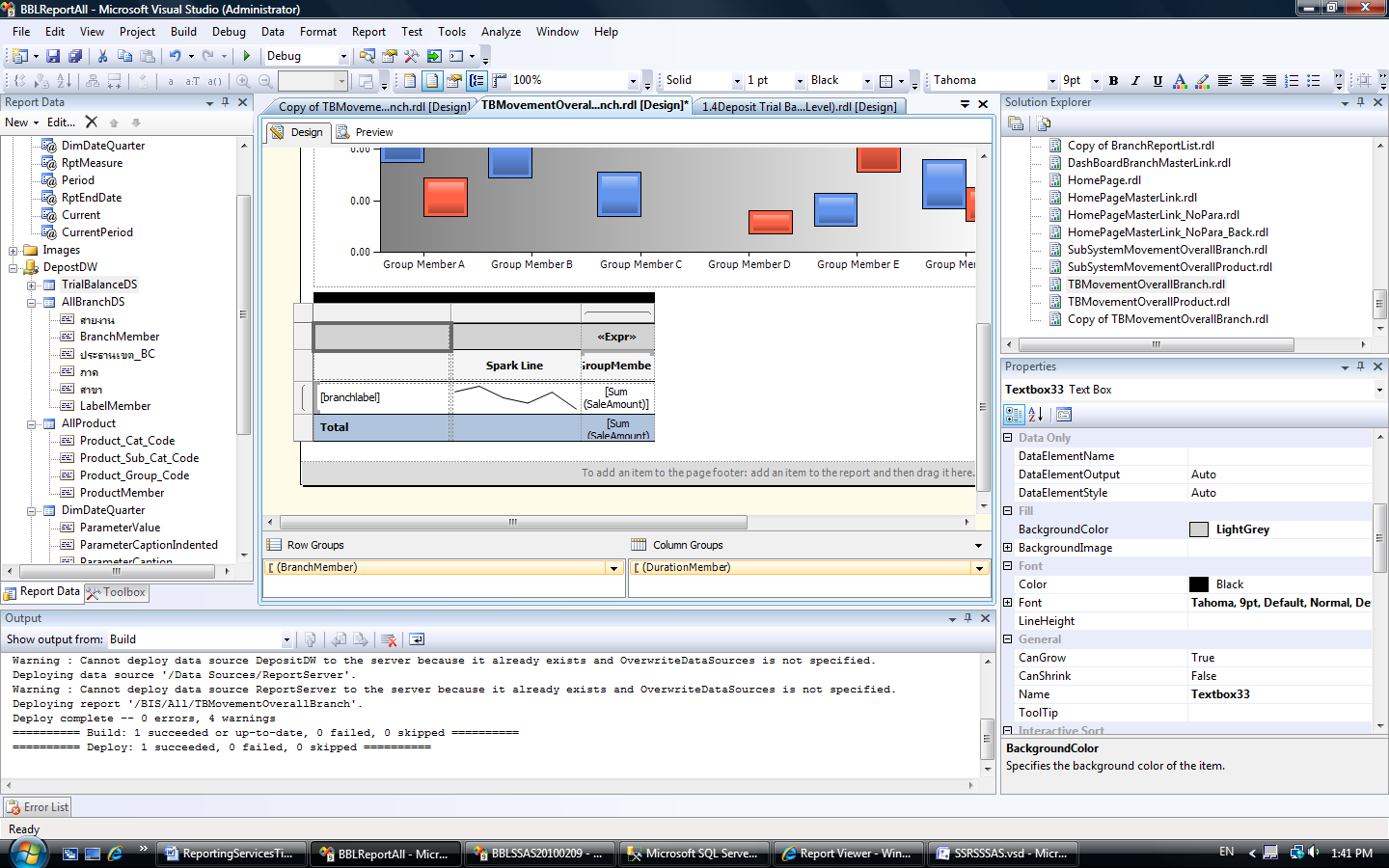
**

Figure 38

To create this Tablix data region:

* Group Row Group with Branch Member. Display label by BranchLabel.
* Group Column Group with DurationMember. Display label by DurationLabel.
* Add Sale Amount field to the value column. Set the Aggregation type to Sum. In this case, the Aggregate() function cannot be used because the Tablix data region does not group from the top level of hierarchy to the level at which data should be displayed. If you use the Aggregate() function, the value will be empty. However, the data will be always correct because the values in this text box come from detail rows that are in the dataset already. This is why the Tablix data region and the graph require two separate datasets. If the same dataset is used, this text box sums the value from each product under the selected value of the product parameter. If users select the measure function, which cannot sum between products such as Year over Year Growth, the result is incorrect.
* Group Category by DurationMember in the **Spark Line** column.
  + 1. Graph

We created the graph shown in Figure 39 by grouping Series by Product\_Code.

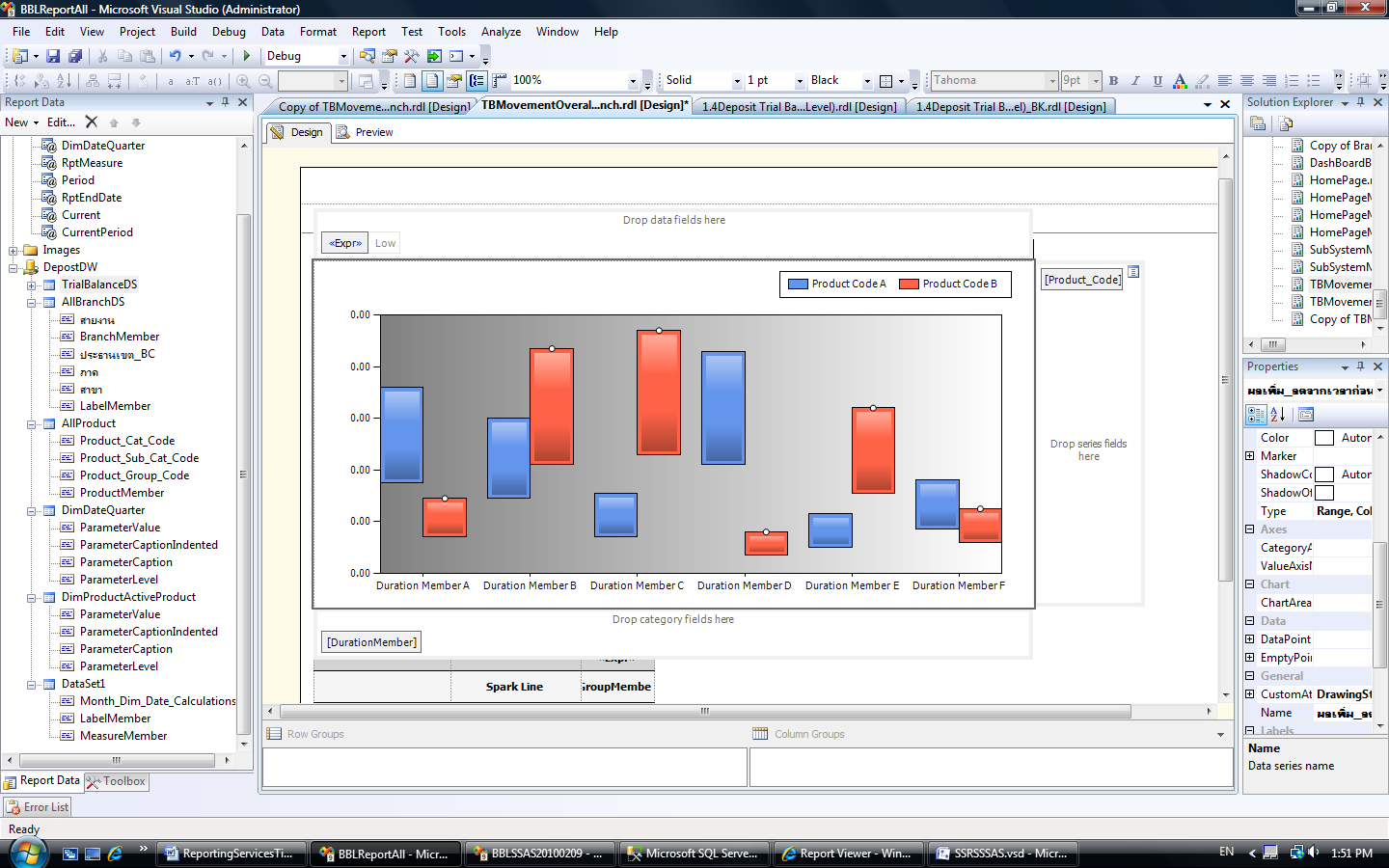


Figure 39

* 1. Additional Tips

This section contains other tips about working with dynamic reports. These are tips we learned from our work on this business case.

* + 1. Improvement Areas

In line with the business requirements for our customer, this report requires only one Sales Amount measure. So, this is not a dynamic measure report. To make this report a dynamic measure report, follow these steps:

1. Add a Measure parameter into the report. This parameter will list all of the measures that this report will use. The value of this parameter should be [Measures].[xxx].
2. Ensure that all measures in the Measure parameter are included in the time intelligence feature. (Include these measures in your MDX code.)
3. Adjust the MDX code of the main dataset by creating a new calculated measure called DynamicsMember.

|  |
| --- |
| WITH MEMBER DynamicsMember AS STRTOMEMBER(@DimMeasure)  /\*\*\*\* where query parameter @DimMeasure = Measure (report parameter)\*\*\*/ |

1. Select DynamicsMember on the column instead of [Measures].[Sale Amount].
2. Replace the Sale Amount field in the report with a DynamicMember field.
3. Implementing Dynamic Security

Last but certainly not least, we implemented a security plan. The customer had over a thousand user groups with different permission levels linked to level and department, as well as region. We needed to find a way to provide the correct level of access for each user or group without using Active Directory® groups. We also wanted to create a system that would be easy for the customer to maintain.

This section describes how we used database roles, a custom mapping table, stored procedures, cube roles, and a custom DLL to dynamically determine access levels for users as they requested data.

* 1. Business Requirement

To meet the customer’s data security requirement for all reports, we needed to design the database roles in Analysis Services cubes to filter query results with the permission rights of the users. However, there were many other complicatedrequirements to be considered.

Users’ permission rights depended on their level and department in the organization. The organization structure was defined by the organization dimension. In total, there were more than a thousand user groups in the organization. Figure 40 illustrates the organization.



Figure 40

Some of the defined business rules were complex. For example, users of Region 1 can access data in their region and all branches under the Region 1, but they can also access data in other regions of the same business unit (Business Unit #1). However, they cannot access data of branches under other regions in the same business unit.

Due to an internal policy, the customer did not have Active Directory® groups defined for the organization structure.

* 1. Dynamic Security Solution

We decided to implement a dynamic security technique to decrease the number of database roles defined in the Analysis Services cube. This technique makes it easier to implement and maintain user roles in the system because it is code-driven, instead of relying on static user IDs.

Four custom components are needed for the solution. Figure 41 illustrates the components and their relationships.

GetAllowedSet (Username)

**Stored Procedure**

**File.DLL**

**(.NET Code)**

**OLAP**

Execute (Username)

MDX FilteredSet

MDX FilteredSet

Group

GroupLookup

(Username)

Result Set

Query with

MDX

**User Security Table**

**๖๗**

**Excel Sheets**

**Reporting Services Reports**

Figure 41

The following sections describe each component in detail.

* + 1. User Security Table

Because the customer did not use Active Directory groups for the organization structure, we created a custom database table to maintain the user list and associated security groups. The table mapped domain usernames to group codes, which we defined. Figure 42 illustrates the group code structure.



Figure 42

* + 1. Stored Procedure

We created a stored procedure to retrieve information from the table. This stored procedure uses the UserName parameter to look up the User Group Code from the user security table. Next, it creates an MDX filter set, which is the set of members of organization dimension that relates with the user group code, and then it sends this MDX filter set to File.dll and the OLAP server.

User Name: Example\Test A

Example\Test A

01010001

Example\Test A

01010002

**Username**

**Group Code**

**{[Dim Organization].[Branch Code].&[0001], [Dim Organization].[Branch Code].&[0002]}**

Figure 43

* + 1. File.dll (.NET Code)

We created aFile.dll assembly to work with role security in the Analysis Services cube. This file is the .NET program that receives UserName for the current user as a parameter in “Domain\Account” format from the OLAP server. The file calls the stored procedure and parses the UserName value to it. Then it returns the MDX filtered set to the caller. We compiled this .NET programming code to a .dll file and registered it as extension assembly file in the Analysis Services server.

* + 1. OLAP Database Role

Because we used our dynamic security technique, we created only one database role in Analysis Services. Then, we wrote MDX code to define the Dimension data permission. This MDX code included a user-defined function that was called with Current User Name as a parameter.

This section shows how to create additional database roles. The MDX code you use to define these roles depends on the business requirements. This is the business requirement that this document uses as the example:

*“Users in Region 1 can access the data in their region and all branches under Region 1; they can also access also other regions in the same business unit (Business Unit #1). However, they cannot access data of branches under other regions in the same business unit.”*

In this case, the MDX code should be defined for both the branch code and the region code attribute in the Organization dimension.

Region Attribute

Add this code in the denied set section of the Roles definition.

|  |
| --- |
| /\*\*\*\* FILE.GetAllowedSet(Username) is the .NET method \*\*\*/  STRTOSET(  ‘  [Dim Organization].[Region Code].[Region Code].members-  NonEmpty(  [Dim Organization].[Region Code].[Region Code].members,  NonEmpty(  [Dim Organization].[Business Unit Code].[Business Unit Code].members,  FILE.GetAllowedSet(Username)  )  )  ’) |

Branch Attribute

Add this code in the denied set.

|  |
| --- |
| /\*\*\*\* FILE.GetAllowedSet(Username) is the .NET method \*\*\*/  STRTOSET(  ‘  [Dim Organization].[Branch Code].[Branch Code].members -  NonEmpty  (  [Dim Organization].[Branch Code].[Branch Code].members,  FILE.GetAllowedSet(Username)  )  ’) |

1. REFERENCES

The following links contain more information about topics covered in this white paper:

“Reporting Recipes” book and downloads, which cover step-by-step instructions for creating sparklines: <http://blogs.msdn.com/robertbruckner/archive/2010/04/11/reporting-services-recipes-book-downloads.aspx>

* Sean Boon’s blog post on sparklines: How To: Build Sparkline Reports in SQL Server Reporting Services,<http://blogs.msdn.com/seanboon/archive/2008/10/10/how-to-build-sparkline-reports-in-sql-server-reporting-services.aspx>

Building parent-child reports in Reporting Services:  
<http://blogs.msdn.com/robertbruckner/archive/2008/10/14/using-analysis-services-parent-child-hierarchies-in-reports.aspx>

* Tuning SSRS-Generated MDX Parameter Queries: <http://cwebbbi.spaces.live.com/Blog/cns!7B84B0F2C239489A!6848.entry>

1. CONCLUSION

Analysis Services and Reporting Services are the core components of the Microsoft BI solution. They can provide fundamental capabilities of business intelligence, including cubes and analytic reports, to your organization with low investment.

This paper discussed a case in which Microsoft worked with a customer to design a custom BI solution that used SQL Server. At the end of the engagement, Microsoft was able to offer a solution that covered most of the customer requirements by using the techniques described in this white paper. Using this solution, the information support department can analyze large amounts of data, and users can browse this data using a familiar tool, Microsoft Office Excel. The solution can deliver up-to-date information to thousands of branches via web-based reports.

Because most of requirements in this customer case are common to many organizations, these techniques may be applicable to your business. If you want, you can also add Microsoft Office SharePoint® Server into the solution. It offers features such as a wider range of user interfaces, collaborative tools, and easier security management.