MSSQLTips.com
brought to you by Edgewood Solutions

Your daily source for SQL Server Tips

Dimension Security Stored Procedures in SQL Server Analysis Services SSAS 2005

Written By: Ray Barley -- 9/24/2009

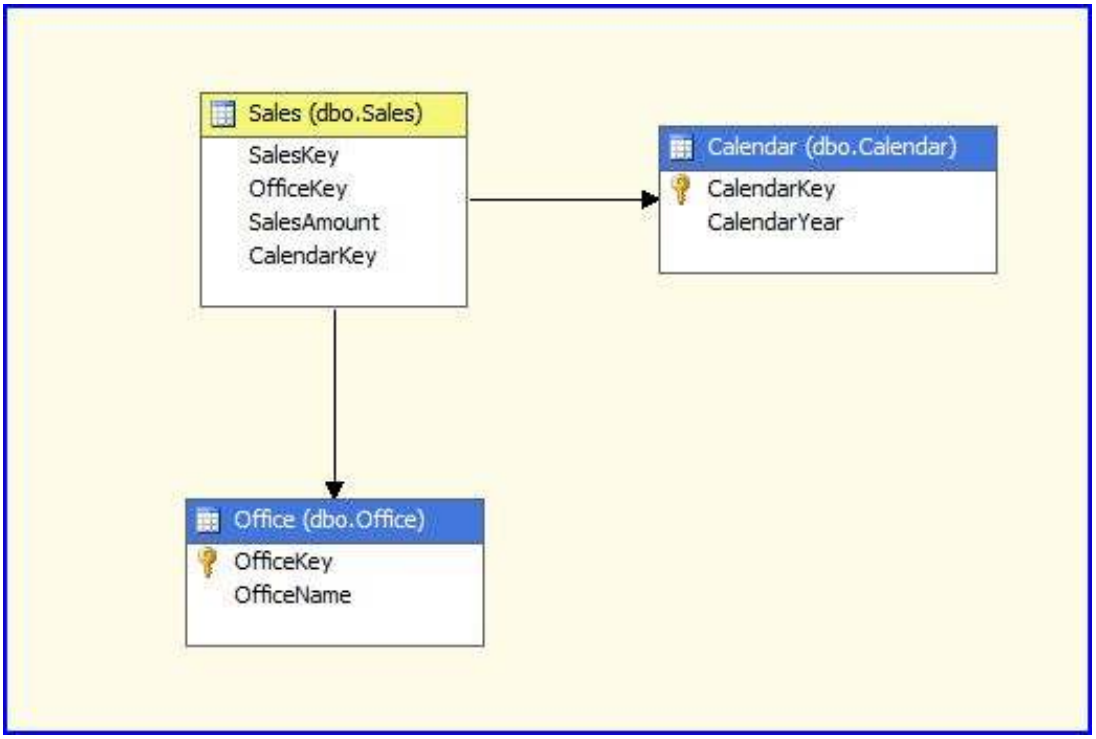
Problem

I have seen the tip Introduction to Dimension Security in SQL Server Analysis Services SSAS 2005 which showed how to define roles in a cube to limit the members of a dimension that are available to the members (Windows users and/or groups) of the role. In that tip the dimension members were specified by simply selecting them via clicking a checkbox. In my case I have a security implementation in the data source to my cube where there are a set of constantly changing rules that determine who can see what. What I need is to be able to leverage the existing security implementation in the data source from the cube. How can I do that?

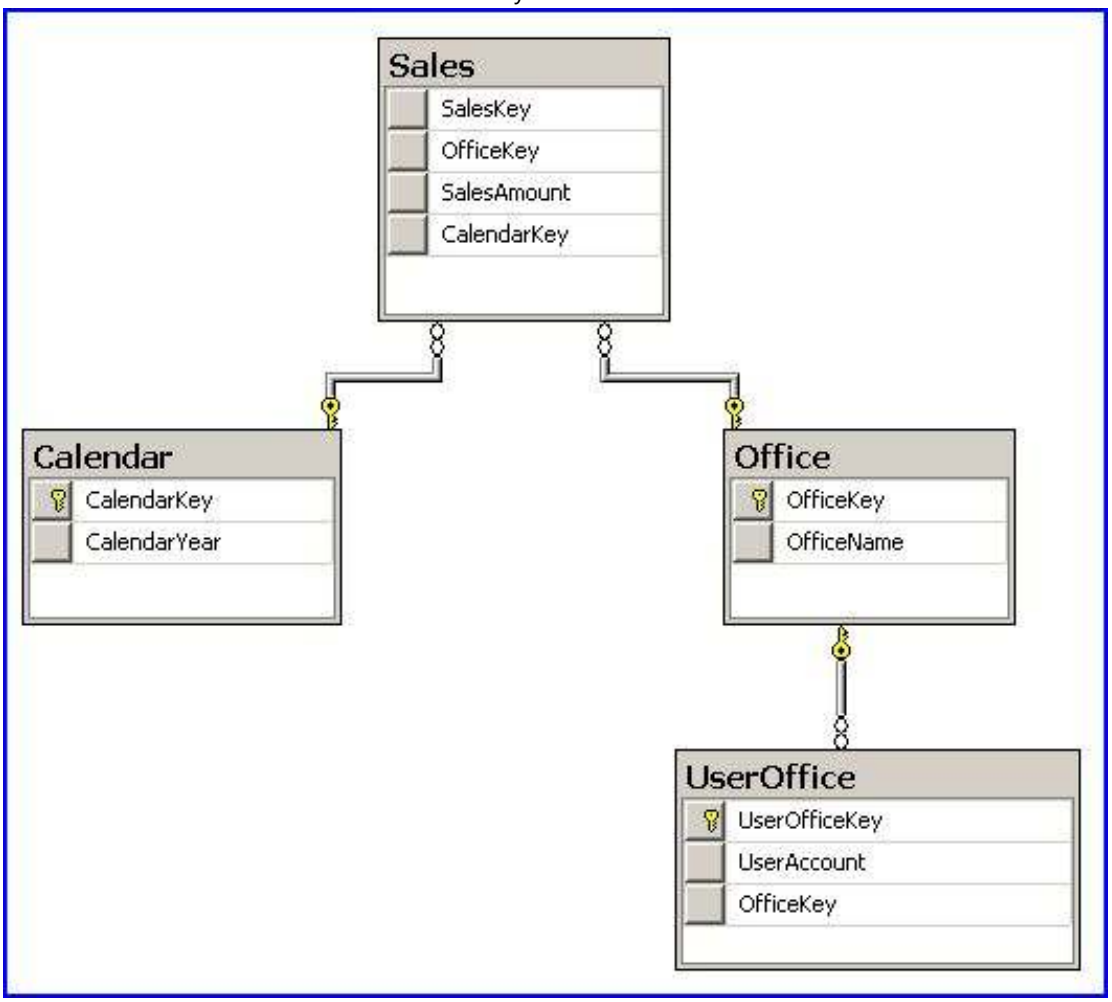
Solution

The role-based security model in SSAS provides three ways to specify dimension security; i.e. what members of a dimension are available to members of a particular role. The tip that you mentioned demonstrated the basic capability where you can specify an allowed set or a denied set of dimension members. You do this by simply selecting or unselecting the dimension members. The other two options allow you to define the allowed or denied set via an MDX expression or call out to a stored procedure. I'll cover the MDX expression option in a future tip; in this one we'll look at how to leverage a stored procedure to dynamically generate the allowed set of dimension members for a role.

As an example I have created the same SSAS cube used in the tip mentioned above in the problem statement. The cube structure is as follows:



The data source for this cube is as follows:



This is the same data source that was used in the earlier tip Dynamically Control Data Filtering in SQL Server Reporting Services Reports which showed how to apply the same type of filtering in an SSRS report based on a SQL Server database. In the above schema the UserOffice table maintains the list of offices that a user can access.

Using the cube structure and data source as shown above, I'll walk through the following steps to demonstrate how to setup dimension security in the cube:

- Create a simple T-SQL stored procedure in the data source that will return the list of offices based on the current user
- Create the SSAS stored procedure code that invokes our T-SQL stored procedure to return the list of offices based on the current user and compile it into a class library DLL; i.e. an assembly
- Deploy the assembly to our SSAS cube
- Setup a role in the cube that will dynamically filter the Office dimension by executing the SSAS stored procedure
- Test the sample

The key point about this solution is that the role-based security that we define in the cube is enforced in the cube no matter what tool we use to query the cube. In other words whether we query the cube from Excel, SQL Server Reporting Services, Report Builder, PerformancePoint, etc. the security will be applied and the user will restricted to accessing the dimension members allowed by the role-based security.

Creating a Stored Procedure in the Data Source

The following T-SQL stored procedure will return the list of offices that the user account is allowed to access:

```
create procedure dbo.FilterOfficeByUser
  @UserAccount nvarchar(50)
as
begin
 set nocount on;
 select o.OfficeKey, o.OfficeName
 from dbo.Office o
 join dbo.UserOffice uo on uo.OfficeKey = o.OfficeKey
 where uo.UserAccount = @UserAccount
end
```

In the above stored procedure we filter the UserOffice table based on the UserAccount then join to the Office table based on the OfficeKey. Our result set will be the list of offices that the user can access.

Creating the SSAS Stored Procedure

The connotation of a stored procedure is a database object that contains T-SQL code. An SSAS stored procedure is actually a method in a .NET class. You write the code in the .NET language of your choice, compile it into a class library DLL, then deploy the DLL to SSAS. The method could execute a T-SQL stored procedure or a T-SQL command, call a web service, etc. You have the entire .NET framework at your disposal. In this example we are going to execute the FilterOfficeByUser stored procedure in the cube data source. We reviewed that stored procedure earlier in this tip. The C# code for our .NET method is shown below:

```
public static Set FilterOfficeByUser(string UserAccount)
{
  Expression expr = new Expression();
  SetBuilder sb = new SetBuilder();
  using (SqlConnection cn = new SqlConnection())
  {
    // prepare connection and command
    cn.ConnectionString =
      string.Format(
        "data source={0};initial catalog={1};integrated security={2};",
        "localhost",
        "mssqltips_dim_security",
        "sspi"
    );
    cn.Open();
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = cn;
    cmd.CommandText = "dbo.FilterOfficeByUser";
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.AddWithValue("@UserAccount", UserAccount);
    // execute stored procedure
    SqlDataReader r = cmd.ExecuteReader(CommandBehavior.SingleResult);
    bool hasRow = r.Read();
    // iterate through the result set
    while (true == hasRow)
    {
      // add each member key to MDX set
      expr.ExpressionText = string.Format(
        "[Office].[Office].&[{0}]", r[0].ToString()
      );
      Member m = expr.CalculateMdxObject(null).ToMember();
```

```
        TupleBuilder tb = new TupleBuilder();
        tb.Add(m);
        sb.Add(tb.ToTuple());
        hasRow = r.Read();
      }
      // return MDX set
      return sb.ToSet();
}
```

The main points about the above method are:

- It uses standard ADO.NET classes such as SqlConnection, SqlCommand, and SqlDataReader to connect to the mssqltips_dim_security database and execute a stored procedure
- The connection string is hard-coded; in a production solution you would retrieve it from some sort of configuration
- Each row returned from the stored procedure contains the OfficeKey column which is added to an MDX set
- The syntax for specifying the dimension member in the SSAS cube is the string [Office].[Office].&[OffficeKey] where OfficeKey is the member key and is the OfficeKey column in the Office table of the mssqltips_dim_security database
- In order to add the dimension member to the MDX set, you convert the string to a Member (the CalculateMdxObject method on the Expression class), add the Member to a Tuple (the Add method on the TupleBuilder class), then add the Tuple to the Set (the Add method on the SetBuilder class)
- The Member, TupleBuilder, Tuple, SetBuilder and Set classes are provided by ADOMD.NET which is essentially ADO.NET for SSAS; you add a reference to the Microsoft.AnalysisServices.AdomdServer assembly to your project in order to use these classes
- The MDX set is the return value of the method

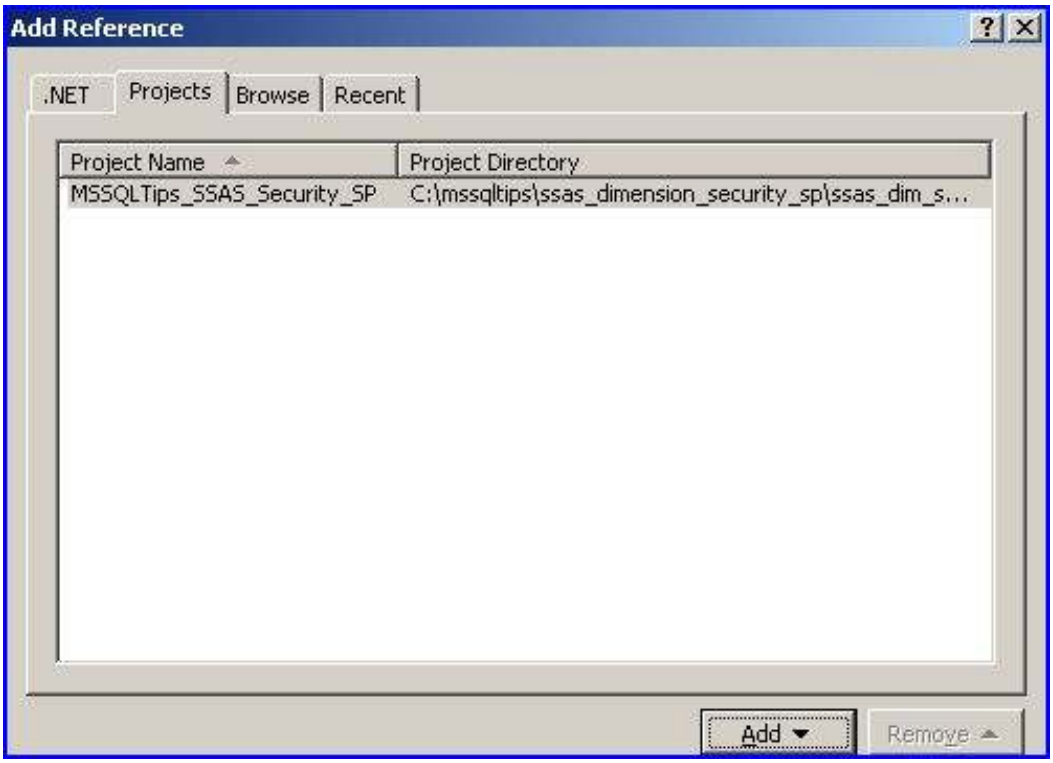
To sum it up we call a stored procedure in our cube's data source that gives us the list of offices that a user is allowed to access then we return that list as an MDX set. The MDX set represents the allowed list of members of the Office dimension for a particular user.

Deploying the Assembly to SSAS

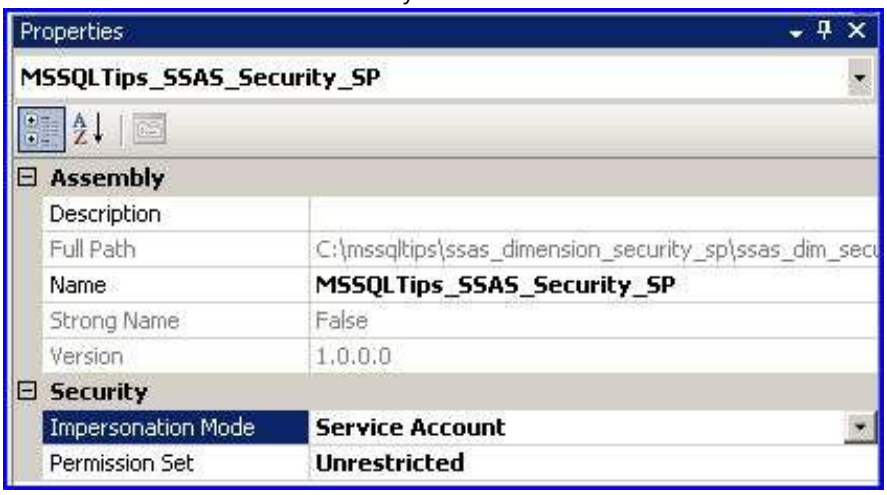
After coding the .NET class method as shown above, you compile it into a class library DLL then deploy that DLL to SSAS. In general .NET terminology a DLL is referred to as an assembly. You can either use SQL Server Management Studio or an Analysis Services project to deploy the assembly to SSAS. We'll use the SSAS project in this example.

You will see an Assemblies node in the Solution Explorer of the SSAS project as shown below:

Right click on the Assemblies node and select New Assembly Reference. Select the assembly to add to the SSAS project from the dialog. In my case the SSAS stored procedure code that we just reviewed above is in a class library project in the same solution as the SSAS project. I click on the Projects tab to add that assembly and the dialog looks like this:

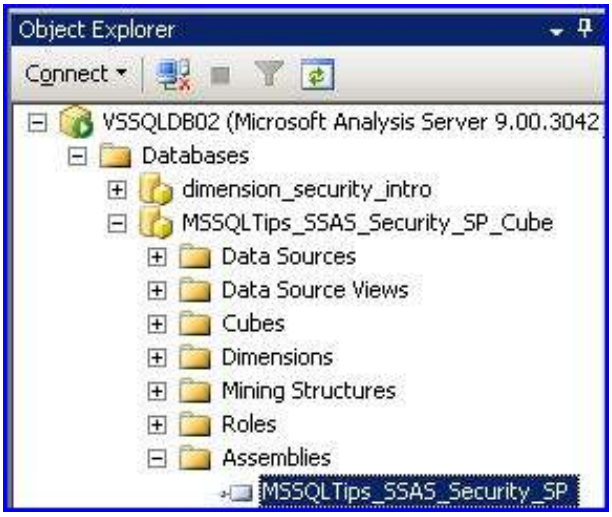


After adding the assembly to the SSAS project, right click on it in the Solution Explorer and edit the properties as shown below:



I set the impersonation mode to Service Account; this is the SSAS service account and those credentials will be used to execute the .NET code in the assembly. I set the permission set to Unrestricted which places no restrictions on the .NET code. Make sure that the account used to execute the .NET code has at least read permission on the data source; i.e. the SQL Server database. For additional information on SSAS stored procedures and the properties noted here, refer to the Books on Line topic Working with Stored Procedures (Analysis Services).

To deploy the SSAS project including the SSAS stored procedure assembly, right click on the project in the Solution Explorer and select Deploy from the menu. After completing this step you can verify that the assembly has been deployed to the cube by opening SQL Server Management Studio (SSMS), connecting to the cube, and drilling down to the Assemblies node as shown below:

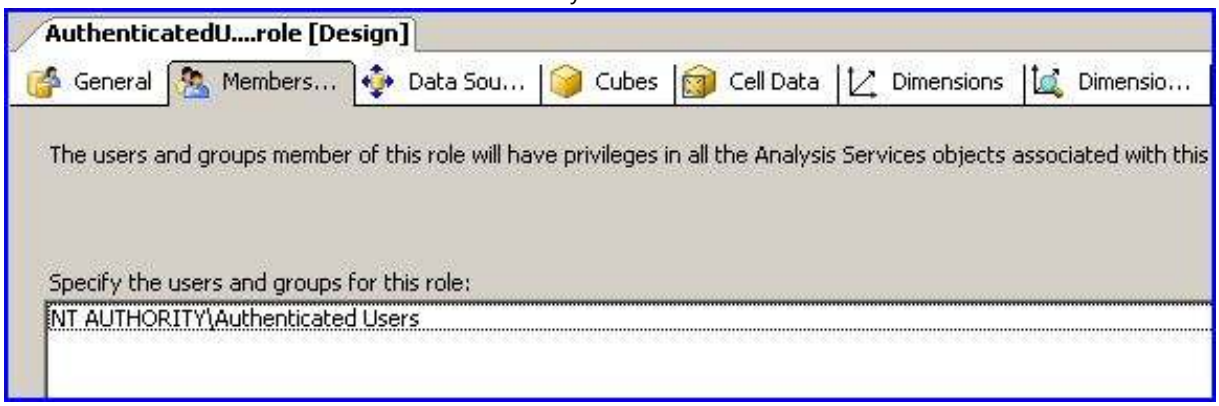


You can also deploy an assembly to the cube using SSMS by right clicking the Assemblies node then selecting New Assembly from the menu. I like associating the assembly with the SSAS project as I have shown.

Setting Up Role-Based Security in the Cube

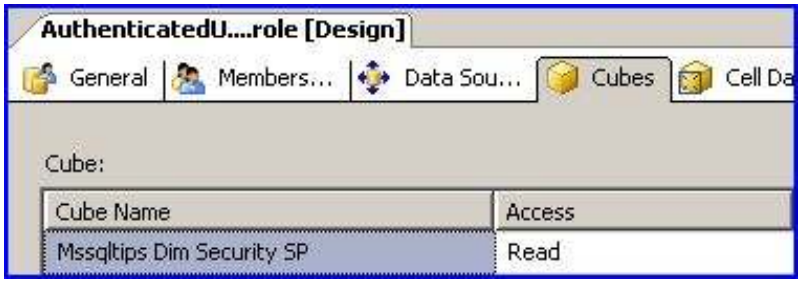
In the previous tip Introduction to Dimension Security in SQL Server Analysis Services SSAS 2005 I showed how to setup role-based security in a cube using SSMS. In this example I'm going to do it in the SSAS project. It is quite possible that while you are working on building the dimensions and measure groups in the SSAS project you really don't have a handle on the role-based security requirements so you can use SSMS to set it up after you've deployed the cube.

To begin, right click on Roles in the Solution Explorer of the SSAS project and select New Role from the menu. Click on the membership tab and add the Authenticated users group as shown below:

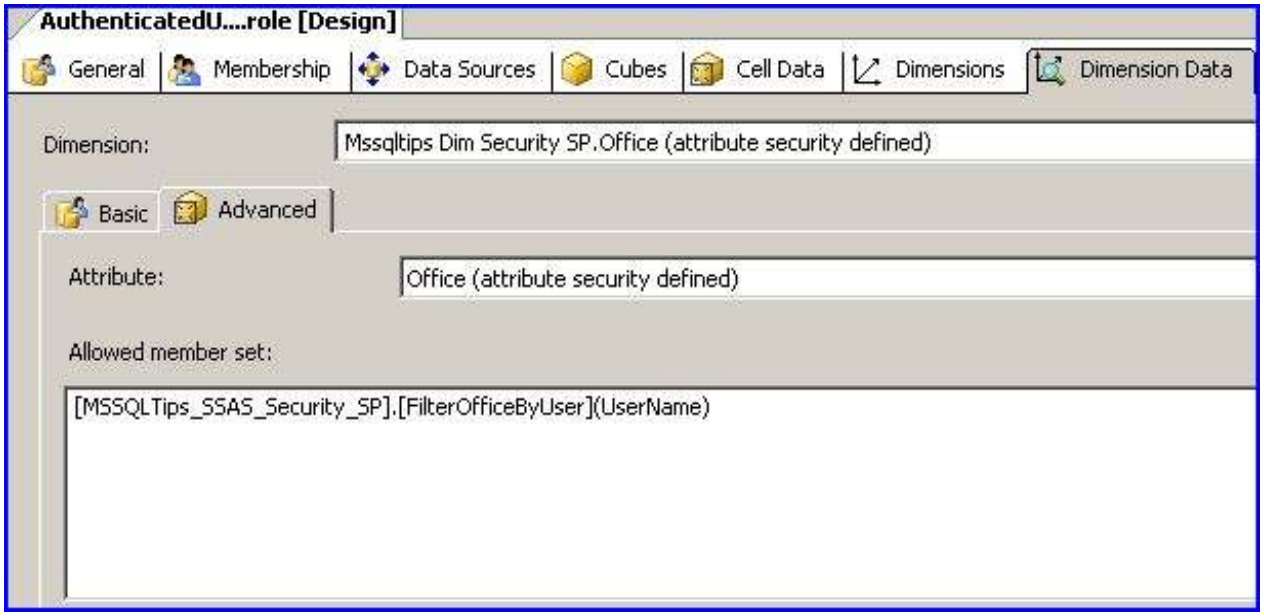


In my example I'm working on a Windows 2003 Server that is not a member of a domain. I selected the Authenticated Users group because the role-based security that I'm setting up is intended to work for any user.

Click on the Cubes tab and allow the role read access to the cube:



Click on the Dimension Data tab to specify the SSAS stored procedure to be invoked to determine the allowed member set as shown below:



In the screen shot above you need to click on the Advanced tab, select the dimension, then enter in the statement to execute the SSAS stored procedure. The statement can be described as follows:

- [MSSQLTips_SSAS_Security_SP] is the name of the class library project (discussed earlier) and when you deploy it to SSAS the Assembly name defaults to the project name (or the name of the DLL if you deploy via SSAS)
- [FilterOfficeByUser] is the name of the method in the .NET class (also discussed above)
- UserName is a built-in MDX function that returns the current user in the form of DOMAIN\USER and

is being passed as a parameter to the FilterOfficeByUser method

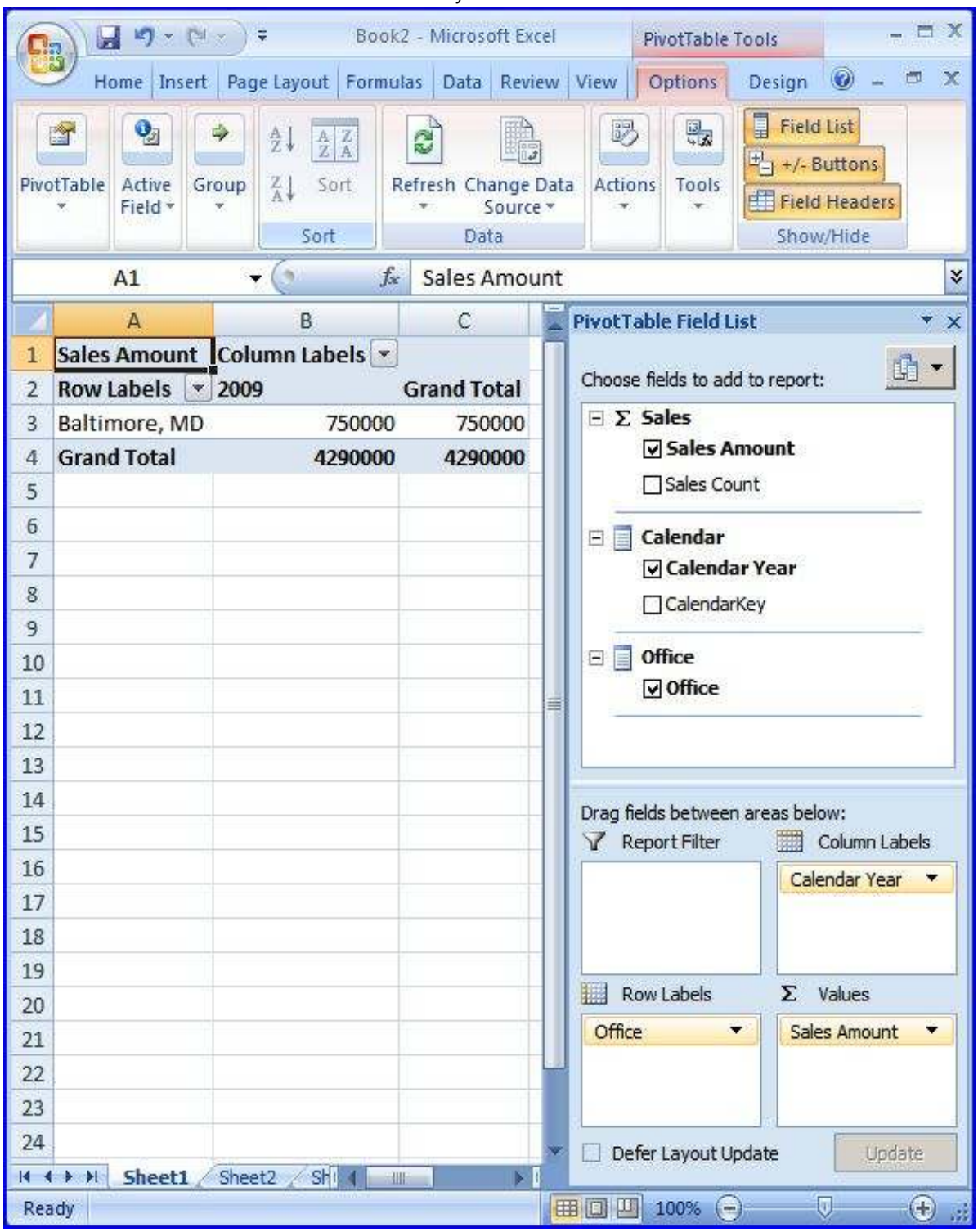
Testing Role-Based Security in the Cube

To test our role-based security setup we'll create an Excel 2007 pivot table by connecting to the cube. We'll launch Excel from a Command Prompt window using the runas command which allows us to specify the user credentials to run Excel. The following is a sample command:

```
runas /user:vssqldb02\awilson
      "C:\Program Files\Microsoft Office\Office12\EXCEL.EXE"
```

Note that this command needs to by entered on a single line. The runas command specifies the user, vssqldb02 is the name of my test server, and awilson is the user I want to test.

After executing the above command to launch Excel, I connect to the cube, insert a pivot table, and drag and drop the fields as follows: Office onto the Row Labels, Sales Amount onto the Values, and Calendar Year onto the Column Labels. I now have the following pivot table:



The point of this test is that the user AWilson is restricted to just the Baltimore, MD office. This is exactly what we expect based on the role-based security that we setup. We can verify the above result by executing the FilterOfficeByUser stored procedure in our data source (this is what the SSAS stored procedure does that we setup in our role-based security). Execute the stored procedure in an SSMS query window and the results are shown below:

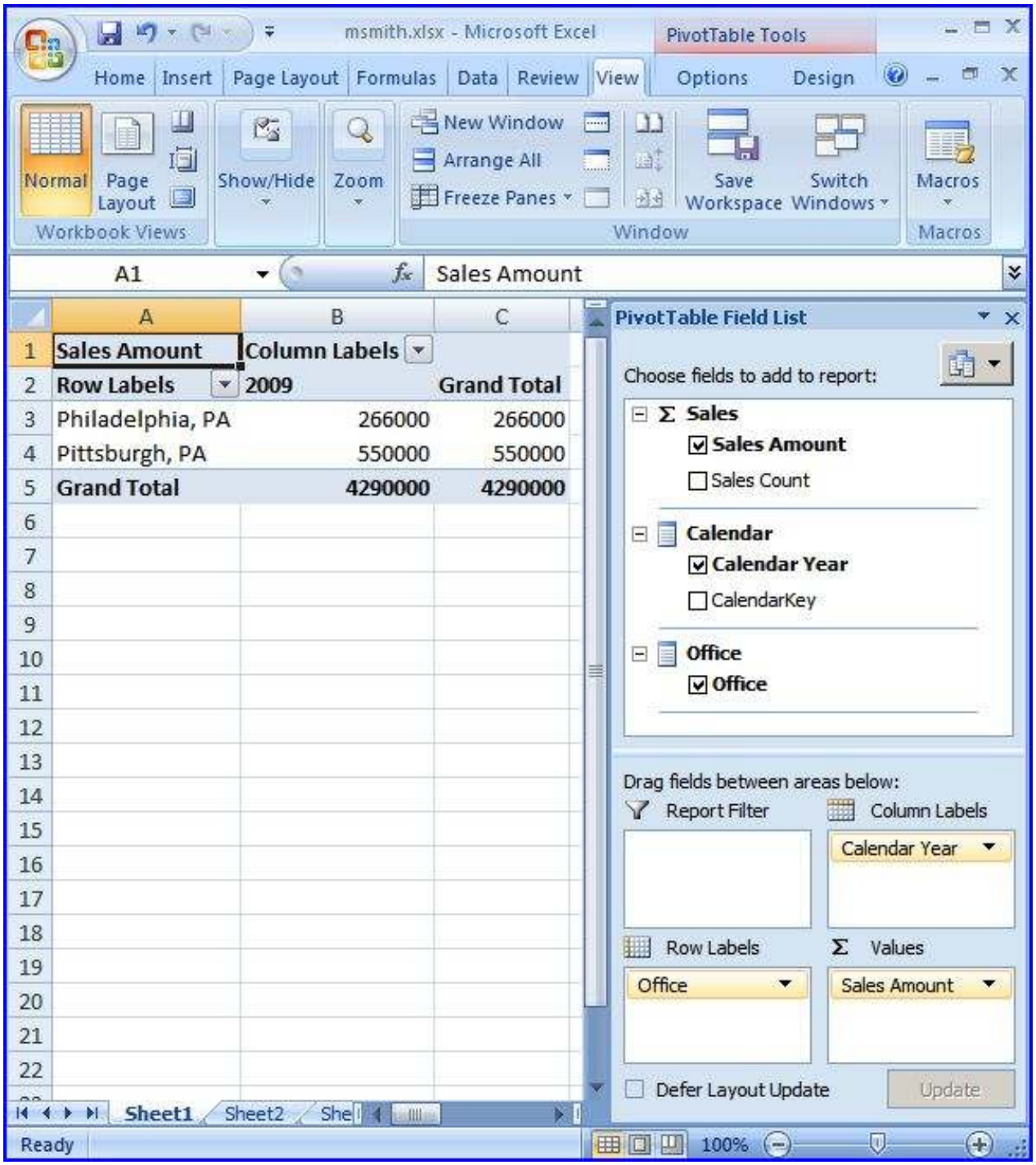
```
exec dbo.FilterOfficeByUser 'vssqldb02\awilson'
exec dbo.FilterOfficeByUser 'vssqldb02\msmith'
```

Results | Messages

	OfficeKey	OfficeName
1	1	Baltimore, MD

	OfficeKey	OfficeName
1	3	Pittsburgh, PA
2	4	Philadelphia, PA

I have executed the FilterOfficeByUser stored procedure for two users and shown the query results . If I repeat the steps above to launch Excel as the user MSmith and create a pivot table, I get the following:



Again this is what we expect as the query results above show that MSmith is allowed to see the Philadelphia, PA and Pittsburgh, PA offices.

Next Steps

- Download the sample code and experiment with it. The archive includes a backup of the SQL Server database used to build the cube and a solution that contains the class library for the SSAS stored procedure and the SSAS project to build the cube.
- Dimension security in an SSAS cube is much more robust than what we get with a SQL Server relational database. This example demonstrates a powerful technique where we can specify dimension security in our cube by leveraging an existing security implementation that lives outside of the cube.
- Kudos to Teo Lachev who authored the excellent book Applied Microsoft Analysis Services 2005 and Microsoft Business Intelligence Platform which I used to gain an understanding of how to leverage SSAS stored procedures for dimension security.