Securing the Tabular BI Semantic Model

SQL Server Technical Article

Writers: Cathy Dumas, Business Intelligence Consultant, Avantage Partners
Kasper de Jonge, Senior Program Manager, Microsoft

Contributors: Marius Dumitru, Akshai Mirchandani, Bob Meyers, Siva Harinath, Bradley Ouellette, Greg Galloway, Howie Dickerman

Technical Reviewer: Nick Medveditskov, Edward Melomed, Owen Duncan, and Dave Wickert, Microsoft; Thomas Ivarsson, Sigma AB

Published: April 2012, Updated April 2016

Applies to: SQL Server 2012, SQL Server 2014, SQL Server 2016, Azure Analysis Services, Power BI

Summary: This paper introduces the security model for tabular BI semantic and Power BI. You will learn how to create roles, implement dynamic security, configure impersonation settings, manage roles, and choose a method for connecting to models that works in your network security context.

Copyright

Contents

Introduction

With the release of Microsoft SQL Server 2012, Analysis Services introduces the BI semantic model. The BI semantic model includes both traditional multidimensional models developed by BI professionals and deployed to SQL Server Analysis Services, and the new tabular model.

Tabular models are secured differently than multidimensional models. Some concepts remain the same – Windows users or groups are added to roles, and a set of security permissions is defined on a per-role basis. These permissions determine whether the user can administer, process, or read data from the model. However, where multidimensional models define read permissions using cell and dimension security, tabular models use a row security model. The set of allowed rows in a table can be restricted by using a row filter defined in Data Analysis Expressions (DAX). Row filters can be static or dynamic. Static filters show the same set of allowed rows to all role members. Dynamic filters vary the set of allowed rows per-user based on the user name or connection string.

This whitepaper describes the tabular security model and demonstrates how to create and manage roles. This paper also describes the security considerations to take for handling credentials when connecting to a tabular model and when connecting to a relational data source for processing.

This paper contains several examples of both static and dynamic security. The dynamic security examples include step-by-step instructions for creating models that implement security based on the currently logged on Windows user name or based on custom data passed to Analysis Services. Two different methods are used to apply row filters – one using many-to-many relationships in DAX, and one using a more computational approach.

You will better understand the examples in this whitepaper if you are already familiar with DAX, especially the concepts of row context, filter context, and many-to-many relationships. For a quick introduction to DAX, see QuickStart: Learn DAX Basics in 30 Minutes (https://support.office.com/en-us/article/QuickStart-Learn-DAX-Basics-in-30-Minutes-51744643-C2A5-436A-BDF6-C895762BEC1A).

With the release of SQL Server 2016, the Tabular model was updated and with it came support for many-to-many patterns that change the way you can do row level security with the tabular model. We will cover both mechanism of row level security in this whitepaper, one pre-SQL Server 2016 and the new technique that can be used with any release post SQL Server 2016 and Azure Analysis Services. Since Power BI shares it's analytical engine with SQL Server Analysis Services the techniques showed here today can also be used in Power BI.

Sample data files

Some examples in this whitepaper are based on the AdventureWorksDW2012 data file. This sample database is available for download at http://msftdbprodsamples.codeplex.com/releases/view/55330.

Other sample files included as part of this whitepaper include:

- The CustomDataExample solution, which includes a tabular model that implements dynamic security using the CUSTOMDATA() function and a C# test client that connects to the tabular model
- The OrganizationalSurveyExample project, which includes a tabular model that implements dynamic security based on a user hierarchy
- The OrganizationalSurveyExample.xlsx file, which is the data source for the OrganizationalSurveyExample project.

These files can be downloaded from http://securitytabularbi.codeplex.com/.

The samples on how to use bi-directional cross filtering for Dynamic security can be found here: https://www.kasperonbi.com/power-bi-desktop-dynamic-security-cheat-sheet/

This sample file applies to SQL Server 2016 and later, Azure Analysis Services and Power BI.

Security model overview

This section describes the core security concepts for the tabular model: permissions, connectivity, and impersonation.

Permissions

Server instances, databases, and rows within a database can be secured. Other objects, such as tables, columns, cells, and perspectives cannot be secured.

At the server instance level, users can be granted administrative permissions. Server administrators have administrative permissions on all databases on the server, so they can read, alter, and process any database on the tabular instance. Also, only server administrators can create, delete, attach, and detach databases on a tabular instance.

At the database level, users can create roles, assign permissions to roles and add other Windows users or groups to roles. Roles can be assigned administrative, process, and/or read permissions. Users in a role with administrative permissions on a database can alter, back up, process, and read all in-memory data in the database. Also, users with administrative permissions on a database can restore the database over a previous version on the server. Users in a role with only processing permissions cannot read any data or metadata in the model; however, they can process data via automation or script. Users in a role with unrestricted read permission can view all in-memory data in the model; however, they cannot read the model metadata.

> **Note** When a model is queried in DirectQuery mode, security permissions defined on the SQL Server data source apply when reading data. For more information, see “Security in DirectQuery enabled models”.

Read permissions can be further refined by applying row filters to tables in the model. A row filter determines the set of rows that a user is allowed to see in a table. Row filters are defined by specifying a DAX expression that evaluates to a Boolean (true/false) value for each row in a table.

> **Note** DirectQuery enabled models do not support row filters.

For example, consider a simple model for analyzing sales, as shown in Figure 1. This model has two tables – Product and Sales – with a one-to-many relationship between the Product and Sales tables. The Sales table can be summarized by sales amount using the Sum of SalesAmount measure.

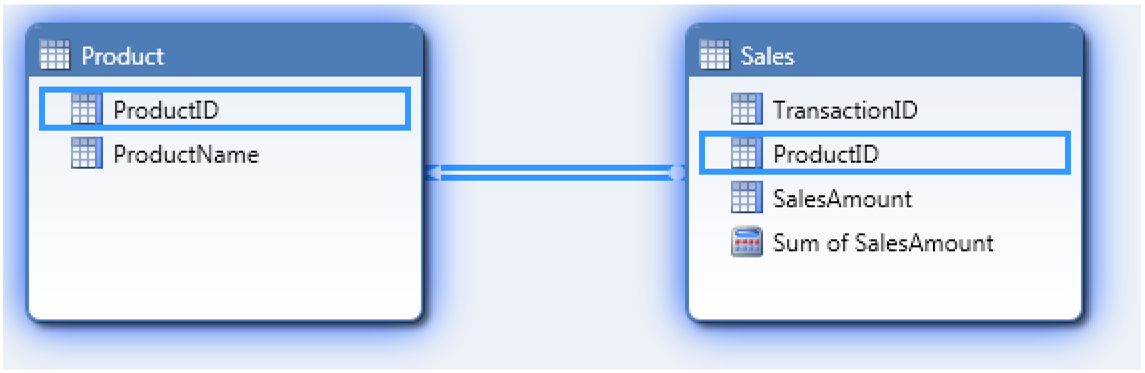


Figure 1: A simple model with a one-to-many relationship

Consider what happens when a row filter is applied to the Product table with the DAX expression `=Product[ProductID]=1`. When a user in the read role issues a query on the Product table, each row is checked to see if the value of the [ProductID] field is 1 and, if so, the row is allowed in the row set and is returned in the query results. Figure 2 shows the process for applying the row filter to the Product table and the resulting pivot table.

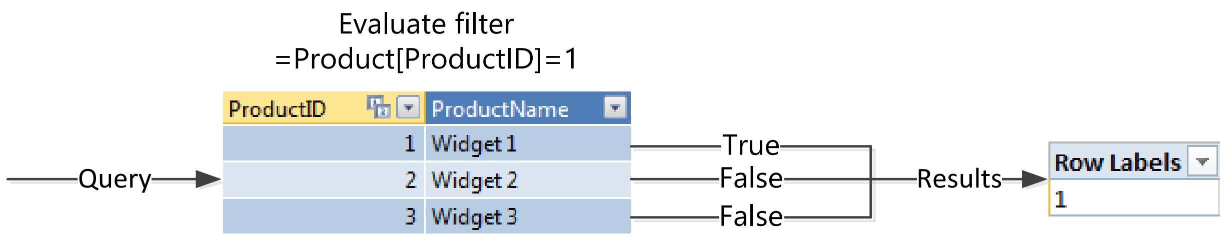
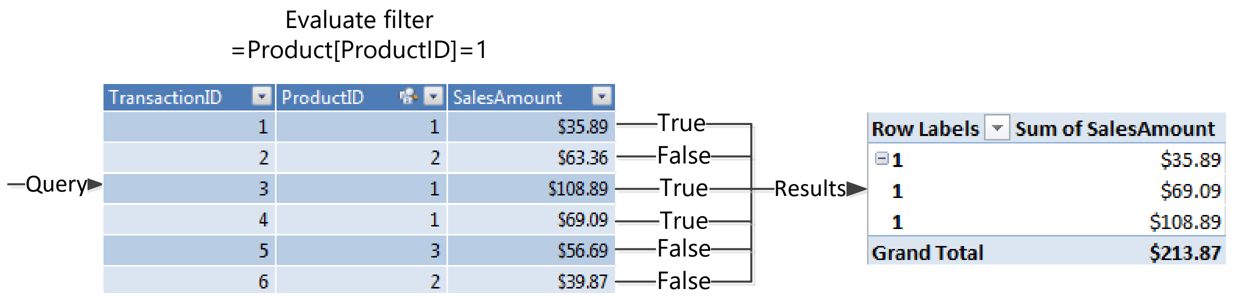


Figure 2: Evaluating an incoming query on the Product table when there is a row filter on that table

Row filters applied to a table also filter related tables. This means that if there is a one-to-many relationship between table A and table B, and a row filter is applied to table A, the set of allowed rows in table B is also restricted according to the filter.

Consider the effect of the row filter defined in Figure 2 on the Sales table. Because there is a one-to-many relationship between Product and Sales, the filter on the Product table also applies to the Sales table. Figure 3 shows the effect of the row filter on the Sales table.



TransactionID	ProductID	SalesAmount
1	1	$35.89
2	2	$63.36
3	1	$108.89
4	1	$69.09
5	3	$56.69
6	2	$39.87

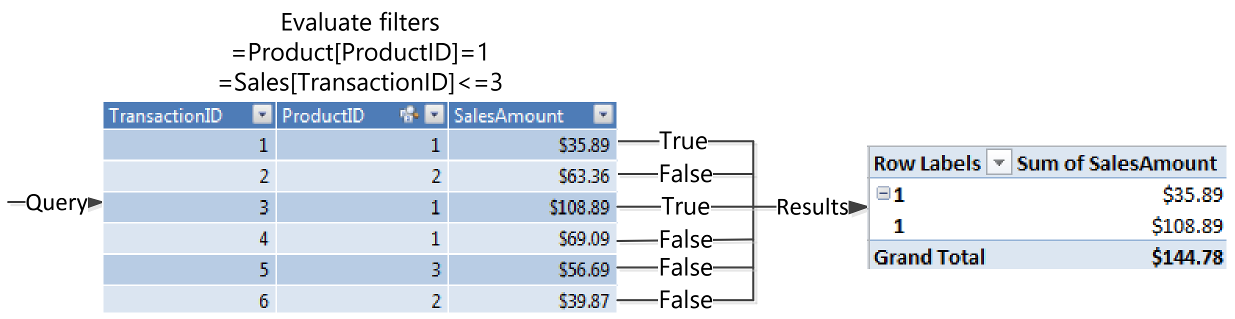
Row Labels	Sum of SalesAmount
1	$35.89
1	$69.09
1	$108.89
Grand Total	**$213.87**

Figure 3: Evaluating an incoming query on the Sales table when there is a row filter on the Product table

Filtering does not work in the opposite direction – any filters applied to table B do not apply to table A. For example, if you define a row filter on the Sales table =Sales[ProductID]=1, this filter has no effect on the Product table. If you want a filter on the many side of a one-to-many relationship to affect the related table, you must apply a filter to both tables and use a many-to-many relationship pattern to restrict the allowed row set.

When multiple row filters are applied to a role, a user in the role can see only the rows allowed by both filters. This means that if a row filter is applied to table A and a row filter is applied to table B, the allowed row set for the user is restricted on both table A and table B. Moreover, if there is a one to many relationship between table A and table B, and a row filter is applied to both tables, the allowed row set for table B is affected by both row filters, and only those rows allowed by both filters are shown.

For example, if you define a row filter on the Product table =Product[ProductID]=1 and a row filter on the Sales table =Sales[TransactionID]<=3, the Sales table is affected by both filters. Figure 4 shows the effect of the row filters on the Sales table.



TransactionID	ProductID	SalesAmount
1	1	$35.89
2	2	$63.36
3	1	$108.89
4	1	$69.09
5	3	$56.69
6	2	$39.87

Row Labels	Sum of SalesAmount
1	$35.89
1	$108.89
Grand Total	**$144.78**

Figure 4: Evaluating an incoming query on the Sales table when there is a row filter on both the Sales and Product tables

Row filters can be static or dynamic. Most row filters are static row filters. When a static row filter is applied, all role members see the same data set. However, when the USERNAME() or CUSTOMDATA() functions are used in the DAX filter, the row filter becomes a dynamic row filter. Dynamic filters change the set of allowed rows based on the currently logged on Windows user name or based on security information passed in the connection string.

A role can be assigned multiple permissions. For example, you can assign both read and process permissions to a role. In this case, role members have all rights associated with all assigned permissions.

You can add a user or group to multiple roles. If a user is a member of multiple roles, they have all of the rights associated with all assigned roles. For example, if a user is a member of both an administrative and read role, the user can administer the database. Another example occurs when a user is a member of multiple read roles with row filters applied. Consider the following scenario: A tabular model is created with two read roles, Role 1 and Role 2. Role 1 restricts access to Table A using a row filter, and Role 2 restricts access to the unrelated Table B using another row filter. If a user is a member of both Role 1 and Role 2, that user can see all data in both Table A and Table B. This is because Role 1 allows unrestricted access to Table B, and Role 2 allows unrestricted access to Table A.

Unlike in the multidimensional model, there is no concept of a deny permission in the tabular model. Permissions simply define the set of operations that users can perform (such as processing or altering the database) and the set of data that users can see. A user in a role with lower permissions can always be granted access to objects and granted permissions to perform operations by adding that user to a different with additional permissions.

Only Windows users and groups can be added to roles. Although users of client applications that query tabular models may connect to reports using credentials other than Windows credentials, these credentials are always mapped to Windows credentials by the application hosting the report (for example, Microsoft Internet Information Services (IIS) or SQL Server Reporting Services).

For more information about roles and permissions, see Roles (https://docs.microsoft.com/sql/analysis-services/tabular-models/roles-ssas-tabular).

Connecting to tabular models

End users connect to tabular models in a number of different ways. For example:

- A user can connect directly to the tabular model from a client application, such as Microsoft Excel, Power BI Desktop, or SQL Server Management Studio, by specifying a connection string to Analysis Services.
- A user can connect to the tabular model using a connection specified in a BI semantic model file (.bism).

- A user can connect to Power View or Reporting Services using a report server data source file (.rsds), and Power View or Reporting Services in turn connects to Analysis Services.
- A user can connect to a middle-tier application, such as an ASP.NET web application, and the middle-tier application in turn connects to Analysis Services.

The method used to connect to the tabular model determines which users you will add to roles in the model, the privileges granted to the Analysis Services service account, and also the privileges granted to service accounts running the Power View, Reporting Services, or middle-tier application connecting to Analysis Services.

Any method for connecting to the tabular model besides directly establishing a connection using a connection string introduces the possibility of a double-hop – that is, it is possible that credentials will need to be passed to an intermediary (such as Reporting Services, SharePoint, or a middle-tier application) which in turn will need to delegate credentials to Analysis Services. Care must be taken to ensure that delegation of credentials is successful. This may involve configuring Kerberos or elevating privileges of the intermediary service account. Later sections of this paper discuss the security requirements in different connectivity scenarios.

Impersonation

When data is loaded into a tabular model, the Analysis Services engine supplies a set of credentials to the relational data source. These credentials are then used to read data from the relational data source. Analysis Services can connect to the relational data source using the credentials of its service account, or it may impersonate another user when connecting to the data source. Impersonation information is specified on each data source in the model, and optionally on the tabular database itself. For more information about impersonation in tabular models, see Impersonation (https://docs.microsoft.com/sql/analysis-services/tabular-models/impersonation-ssas-tabular).aspx).

DirectQuery enabled models support additional impersonation information. This is because DirectQuery enabled models can connect to a relational data source in two scenarios – when the model is queried and when the model is processed (for DirectQuery enabled models in hybrid mode). Because there is one additional scenario for connections, there is one additional set of impersonation settings, called the DirectQuery impersonation setting, that can be configured to allow Analysis Services to impersonate the current user when querying the data source.

Whenever Analysis Services impersonates another user, that user must have at least read access to the relational data source. If the impersonated user does not have sufficient privileges, the query or processing attempt fails.

Creating roles

Roles can be created when the model is developed in SQL Server Data Tools, or when the model is administered in SQL Server Management Studio.

We recommend that you create roles in SQL Server Data Tools. Writing logic in DAX for defining read permission to the database is a task best suited for a BI developer, not a database or server administrator. Using SQL Server Data Tools allows the role definitions to be checked in to source control, so modifications can be tracked over time. The Role Manager in SQL Server Data Tools provides functionality that is unavailable in SQL Server Management Studio, such as validation for DAX syntax and semantic errors. Also, Analysis Management Objects (AMO) is not well suited for creating roles because the AMO object model for modifying role permissions is complex.

We recommend that you add role members after deployment, either in SQL Server Management Studio or using automation. If you add role members in SQL Server Data Tools, these users will be able to read the workspace database as you are developing the model, which is often undesirable. Also, adding role members is a task best suited for an administrator, not a BI developer. BI developers may not have access to the production system, and therefore may not have permission to add the Windows users to roles. Also, role membership changes over time and typically changes more frequently than the model itself.

We also recommend that you add Windows groups, and not individual Windows users, as role members whenever possible. This eases administration because group membership changes in Active Directory Domain Services (AD DS) automatically propagate to Analysis Services.

We do not recommend that you add role members in SQL Server Data Tools. If you add role members in the development environment, you can accidentally overwrite changes made in production to the list of role members when you deploy the model. The only time you should add role members (usually Windows groups) in the development environment is when dynamic security is applied to a model. The easiest way to test dynamic security is to add role members. These role members should be removed before you deploy the model.

Because roles are defined in one place, and role membership is defined in another place, care must be taken when deploying models with security defined. If models are deployed directly from SQL Server Data Tools, the role membership will not be preserved. We recommend that you deploy models with roles using the Deployment Wizard. The Deployment Wizard provides functionality for preserving role membership.

Using the Role Manager in SQL Server Data Tools

Roles are created in SQL Server Data Tools using the Role Manager. For more information about using the Role Manager, see Create and Manage Roles (https://docs.microsoft.com/sql/analysis-services/tabular-models/create-and-manage-roles-ssas-tabular). For a tutorial that includes step-by-step instructions on how to create basic roles and roles with row filters, see Lesson 12: Create Roles (https://docs.microsoft.com/ sql/analysis-services/lesson-11-create-roles).

Testing roles

You can test roles using the Analyze in Excel feature in SQL Server Data Tools. For more information about using Analyze in Excel, see Analyze in Excel (https://docs.microsoft.com/sql/analysis-services/tabular-models/analyze-in-excel-ssas-tabular).

For a tutorial that includes step-by-step instructions on using Analyze in Excel, see Lesson 12: Analyze in Excel (https://docs.microsoft.com/sql/analysis-services/lesson-12-analyze-in-excelFor information about installing Excel on a machine with SQL Server Data Tools installed, see Working with the ACE provider in the tabular designer (http://blogs.msdn.com/b/cathyk/archive/2011/09/29/working-with-the-ace-provider-in-the-tabular-designer.aspx).

You can test security for one or more roles and for one or more role members using Analyze in Excel. If you have not implemented dynamic security, we recommend that you only test roles in the development environment, because you do not need to add role members.

If you cannot install Excel on the machine with SQL Server Data Tools installed, you can test roles during development in one of the following ways:

- You can connect to the workspace database in SQL Server Management Studio using the default connection string and then browse the database, specifying the role or user name in the user interface.
- You can connect to the workspace database in SQL Server Management Studio and specify the name of the user or role that you want to test as an additional connection parameter.
- You can use Excel on another machine to connect to the workspace database and specify the name of the user or role that you want to test in the connection string to Analysis Services.

SQL Server Management Studio provides a user interface that you can use to change the role or user when browsing the model.

To test a role or user in the browser in SQL Server Management Studio:

1. From Object Explorer, right-click the database that you want to test and then click **Browse.**
2. On the **Cube** menu, click **Security Context**.
   or
   On the toolbar in the browser, click the security icon.
3. Select the role(s) or user(s) that you want to test.

You can only use the browser in SQL Server Management Studio to test Multidimensional Expressions (MDX) queries. If you want to test DAX queries or XML for Analysis (XMLA) commands, you must specify the role or user name before connecting to Analysis Services. To do this, add the *Roles* parameter or the *EffectiveUserName* parameter to the connection string for the tabular instance.

To use the *Roles* or *EffectiveUserName* parameters in SQL Server Management Studio:

1. From Object Explorer, click **Connect** and then click **Analysis Services**.
2. Type the name of the tabular instance hosting the workspace database in the **Server Name** box.

3. Click **Options**.
4. Click the **Additional Connection Parameters** tab. In the box, specify one of two things:
    - To test a role, type **Roles=**<*role name*>. To test multiple roles, type **Roles=**<*role name 1*>**,** <*role name 2*>
    - To test a user, type **EffectiveUserName=**<*Windows user name*> where <*Windows user name*> is in the form DOMAIN\username
5. Click **Connect**.

Now you can execute queries and commands that use the security context of the specified role or user. Note that the Object Explorer in SQL Server Management Studio does not change to reflect the effective user name or role; only query or command results are affected by the user or role name change.

Another way to test security is to use Excel on a different machine to connect to the workspace database. To do this, add the *Roles* parameter or the *EffectiveUserName* parameter to the connection string for the tabular instance.

To use the *Roles* or *EffectiveUserName* parameters in Excel:

1. Start Excel.
2. On the **Data** tab, in the **Get External Data** group, click **From Other Sources**, and then click **From Analysis Services**.
3. Type the name of the tabular instance hosting the workspace database in the **Server Name** box and then click **Next**.
4. Select the workspace database from the list of databases and then click **Finish**.
5. In the **Import Data** box, click **Properties**.
6. Click the **Definition** tab.
7. In the **Connection string** box, append one of the following strings to the existing connection string:
    - To test a role, type **;Roles=**<*role name*>. To test multiple roles, type **;Roles=**<*role name 1*>**,** <*role name 2*>
    - To test a user, type **;EffectiveUserName=**<*Windows user name*> where <*Windows user name*> is in the form DOMAIN\username.
8. Click **OK** to close the **Properties** box, and then click **OK** to close the **Import Data** box.

You can only use the *EffectiveUserName* or *Roles* connection string parameters to test security if you are a server administrator on the tabular instance hosting the database that you want to test. If you are testing in a development environment on the workspace database this is not usually a problem because you must be an administrator on the instance to use SQL Server Data Tools.

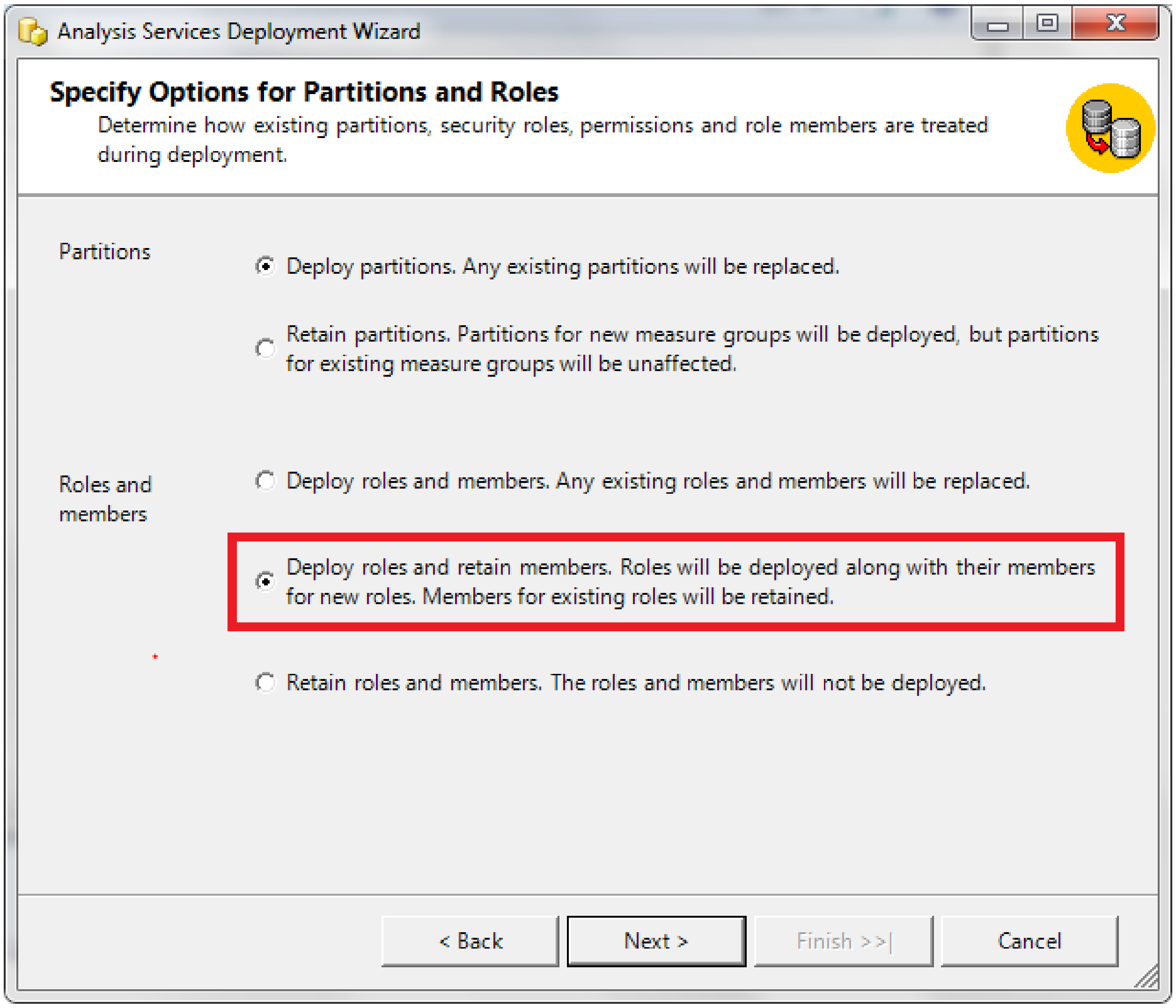
If you are connecting to the workspace database from another machine, ensure that the firewall rules on the machine running the workspace database server instance allow incoming connections to Analysis Services. If you used the default installation options for Analysis Services, you must open port 2383 on the firewall to incoming connections. For more information about firewall configuration, see Configure the Windows Firewall to Allow Analysis

Services Access (https://docs.microsoft.com/sql/analysis-services/instances/configure-the-windows-firewall-to-allow-analysis-services-access).

Deploying models with roles

Because roles are created in one place and role membership is managed in another, we recommend that you use the Deployment Wizard to deploy models with roles. The Deployment Wizard is a stand-alone utility included with the SQL Server management tools that you can use to deploy both multidimensional and tabular models. For more information about using the Deployment Wizard, see Deploy Model Solutions Using the Deployment Wizard (https://docs.microsoft.com/ sql/analysis-services/multidimensional-models/deploy-model-solutions-using-the-deployment-wizard).

When you deploy the model using the Deployment Wizard, ensure that you use the **Deploy roles and retain members** setting, as pictured in Figure 5. This ensures that metadata changes made in the Role Manager in SQL Server Data Tools are deployed to the server, but the role membership remains unchanged.



Specify Options for Partitions and Roles

Determine how existing partitions, security roles, permissions and role members are treated during deployment.

Partitions

- [✓ Deploy partitions. Any existing partitions will be replaced.]
- Retain partitions. Partitions for new measure groups will be deployed, but partitions for existing measure groups will be unaffected.

Roles and members

- Deploy roles and members. Any existing roles and members will be replaced.
- [✓ Deploy roles and retain members. Roles will be deployed along with their members for new roles. Members for existing roles will be retained.]
- Retain roles and members. The roles and members will not be deployed.

Figure 5: Deployment Wizard with the **Deploy roles and retain members** setting highlighted

Dynamic security

Dynamic security is useful in many scenarios, and this paper provides a few examples. In one scenario, access to sales data is granted based on geographical areas. In another scenario, access to organizational data is granted in such a way that users can see data for themselves and for their reports (if any), but cannot see data for their manager or for their peers.

Dynamic security is implemented using the USERNAME() and CUSTOMDATA() functions in the row filter definition for a table. The USERNAME() function returns the name, in the form DOMAIN\user name, of the Windows user connected to the model. This is typically the currently logged on Windows user. However, if an administrator on the database is impersonating a different user by adding the *EffectiveUserName* to the connection string (either manually or using the Analyze in Excel function), this impersonated user's name is returned by the USERNAME() function. The CUSTOMDATA() function returns a string embedded in the connection string used to connect to Analysis Services. This function is helpful when dynamic security is implemented in an environment where the Windows user name is not available.

> **Note** Do not use CUSTOMDATA() to configure security for a model when a client is connecting directly to Analysis Services. This is not secure. A user that manually crafts a connection string may see data that the user is not intended to see, resulting in information disclosure. Only use the CUSTOMDATA() function to configure security for a model that is used by a middle-tier application.

Modeling patterns for dynamic security

There are some common modeling patterns for implementing dynamic security, regardless of the method used to implement security (USERNAME() or CUSTOMDATA()). Allowed values can be stored in a relational data source, such as SQL Server, and managed by an administrator. The table with the allowed values is then imported into the tabular model and related to the table to be secured.

There is a one-to-many relationship between the two tables, with one value in the table to be secured related to many rows in the security table. Recall that row security cascades in the one-to-many direction, not in the many-to-one direction. That means it is impossible to apply a filter directly to the security table and have it apply to the table that you want to secure. Instead, to implement security, a many-to-many relationship pattern must be used. As many-to-many patterns were not supported in the Tabular model until SQL Server 2016, we will discuss both methods: one pre-SQL Server 2016 and one post SQL Server 2016, where we are able to have filters flow in the many-to-one direction.

Enabling dynamic security using Tabular models before SQL Server 2016

To create a model for dynamic security:

1. Create a two-column security table in the relational data source. In the first column, specify the list of Windows user names or custom data values. In the second column,

specify the data values that you want to allow users to access for a column in a given table.
2. Import the security table into the tabular model using the Import Wizard.
3. Using the Import Wizard, create a one column table that contains the user names or custom data values for the model. The values in this table must be unique. This table does not need to be materialized in the relational data source; instead, it can be constructed in the Import Wizard by querying the security table for the unique user names or custom data values.
4. Create a relationship between the security table and the column that is being secured.
5. Create a relationship between the bridge table created in step 3 and the security table created in step 2.
6. [optional] Add supporting calculated columns or measures for computing the row filter on the table that contains the column that you want to secure.
7. Create a role in the Role Manager with Read permission.
8. Set the row filter for the security table to `=FALSE()`.
9. Set the row filter for the bridge table to restrict the allowed row set to the currently logged on Windows user or to the custom data value using a row filter like `=[Column Name]=USERNAME()` or `=[Column Name]=CUSTOMDATA()`.
10. Set the row filter on the table that you want to secure using one of the methods described in the examples that follow.

There should be one security table per column that contains values that you want to secure. If you want to secure multiple values, create multiple security tables and create relationships and row filters as described earlier.

You can take other approaches to data modeling for dynamic security in other scenarios. If you are securing data in a parent/child hierarchy where the allowed rows are defined by the level in the hierarchy, you do not need and an external data source for the security table and you do not need to use the many-to-many modeling pattern. The section “Example: Securing a model using parent/child hierarchies” describes this modeling pattern. Also, instead of storing the security information in a relational data source like SQL Server, you can manage security using security groups in Active Directory Domain Services (AD DS) and query these groups from the tabular model using a third-party component or using the Microsoft OLE DB Provider for Microsoft Directory Services. A detailed discussion of that implementation is out of the scope of this document; however, you can use the many-to-many relational modeling techniques illustrated here to implement dynamic security in that environment.

Example: Implementing security for a model using the USERNAME() function

In this example, security for the Internet sales data in the AdventureWorks database is implemented by sales territory. Each Windows user with access to the tabular model has access to sales by one or more sales territories. The mapping of user names to allowed sales territories is pasted into the tabular model.

Before you begin, select a set of users on your domain to use for testing purposes. You can create test accounts in Active Directory Domain Services (AD DS) or use other users’ accounts on the domain. You do not need to know the passwords to these accounts. It is helpful if these users are all members of one group in the domain, so the group can be added as a role member instead of individual users. Also, if you are not familiar with DAX, you should familiarize yourself with some basic DAX concepts, especially row context and filter context, before you begin. For a quick introduction to DAX, see QuickStart: Learn DAX Basics in 30 Minutes (https://support.office.com/en-us/article/QuickStart-Learn-DAX-Basics-in-30-Minutes-51744643-C2A5-436A-BDF6-C895762BEC1A).

To begin, first import the required data from AdventureWorks. To do this:

1. Launch SQL Server Data Tools and then create a new tabular project.
2. On the **Model** menu, click **Import from Data Source** to launch the Import Wizard.
   or
   On the Analysis Services toolbar, click the database icon.
3. Click **Next** to use Microsoft SQL Server as the data source.
4. Enter the server name and the database name for the AdventureWorksDW2012 data source and then click **Next**.
5. Enter the user name and password for a Windows user with at least read permission on the AdventureWorksDW2012 database and then click **Next**.
6. Click **Next** to import tables using Preview and Filter.
7. Click the check box next to the FactInternetSales table to select it, click **Select Related Tables** to get some related dimensions, and then click **Finish**.

Seven tables and the relationships between them are imported, as shown in Figure 6.

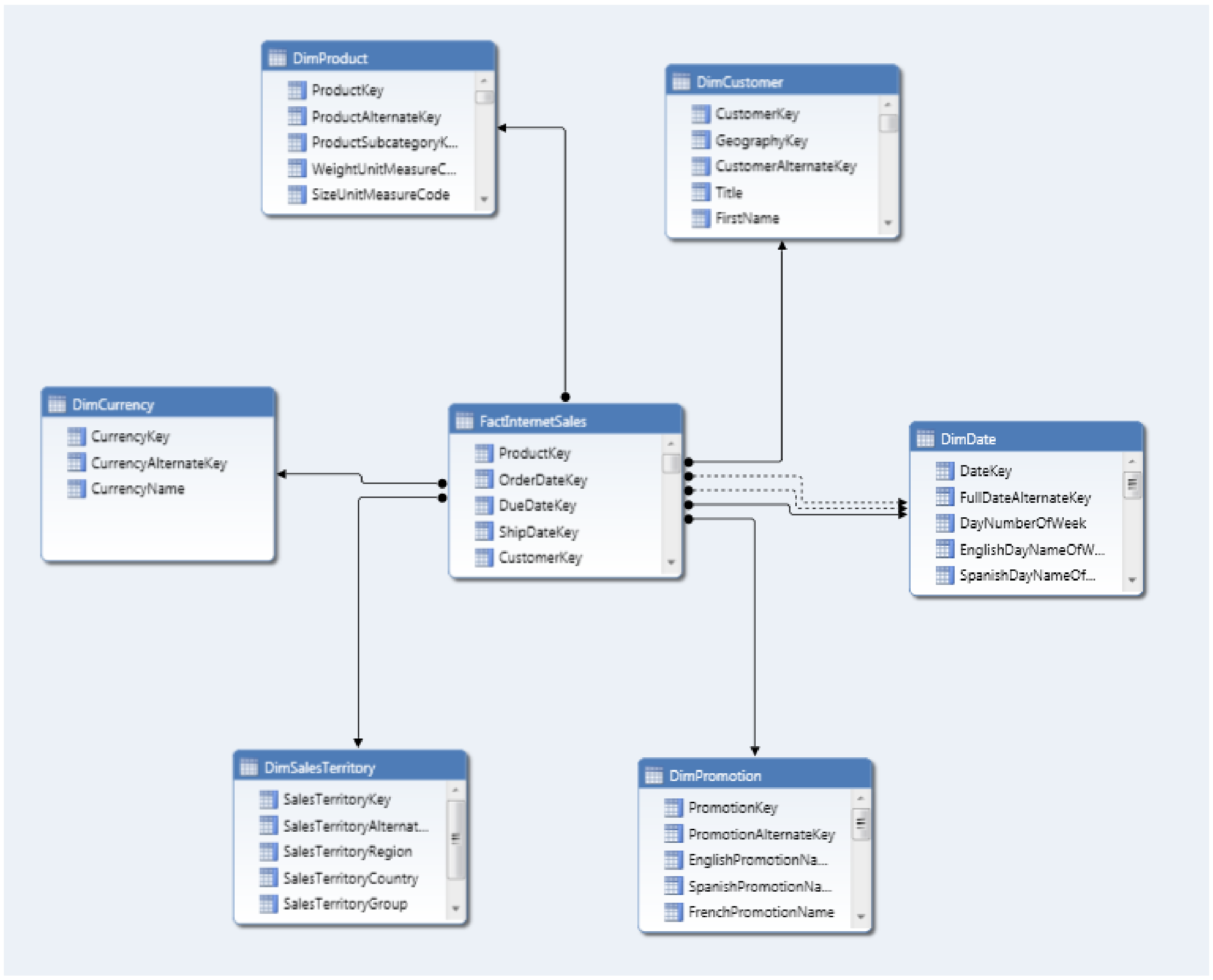
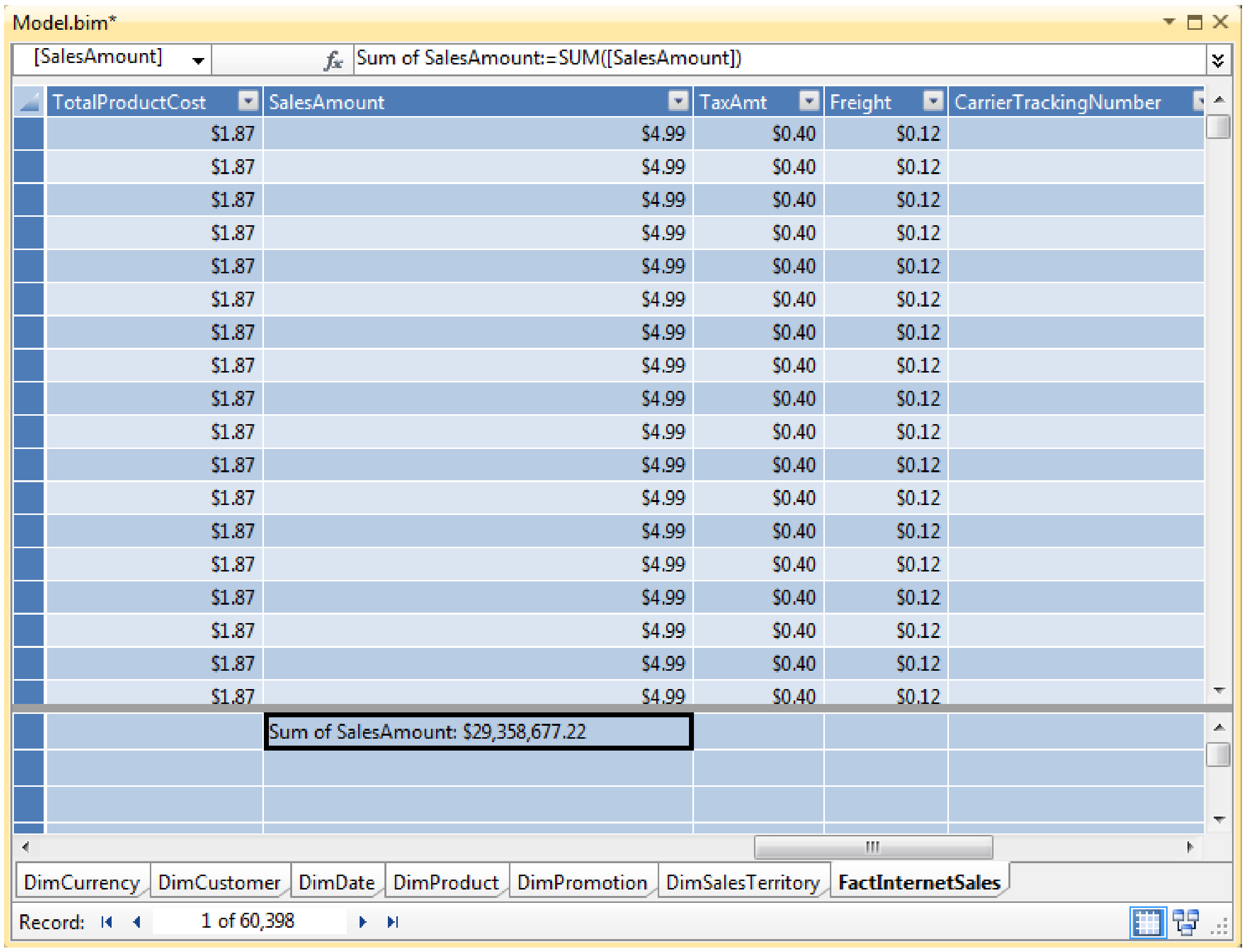


Figure 6: The tabular model after you have imported the tables and relationships from the AdventureWorks database

Next, create a measure to use for testing purposes:

1. With the model in Data View, click the sheet tab for the FactInternetSales table.
2. Select the SalesAmount column.
3. On the **Column** menu, point to **AutoSum** and then click **Sum**.
   Or
   On the Analysis Services toolbar, click the sigma icon.

The tabular model now contains a measure named Sum of SalesAmount that reports on Internet sales, as shown in Figure 7.


Model.bim*

[SalesAmount] Sum of SalesAmount:=SUM([SalesAmount])

TotalProductCost	SalesAmount	TaxAmt	Freight	CarrierTrackingNumber
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	
$1.87	$4.99	$0.40	$0.12	

Sum of SalesAmount: $29,358,677.22

DimCurrency | DimCustomer | DimDate | DimProduct | DimPromotion | DimSalesTerritory | **FactInternetSales**

Record: 1 of 60,398

Figure 7: Sum of the SalesAmount measure

Next, prepare the security table:

1. Copy and paste the following table into Excel:

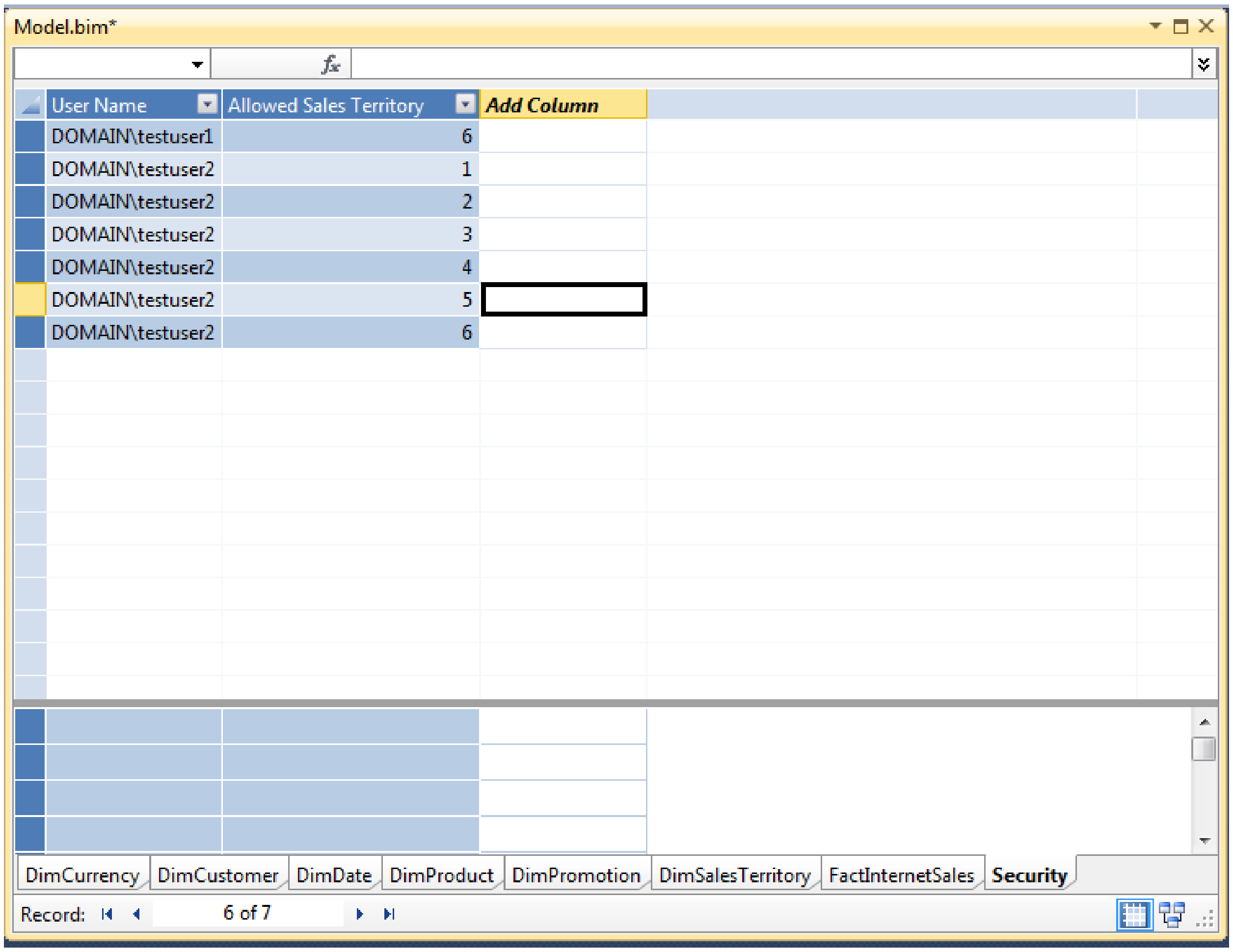
User Name	Allowed Sales Territory
DOMAIN\testuser1	6
DOMAIN\testuser2	1
DOMAIN\testuser2	2
DOMAIN\testuser2	3
DOMAIN\testuser2	4
DOMAIN\testuser2	5
DOMAIN\testuser2	6

Table 1: The Security table

2. Change DOMAIN\testuser1 and DOMAIN\testuser2 to users on your local domain.
3. Select the modified table and then copy it.
4. Click the tabular model in SQL Server Data Tools.
5. On the **Edit** menu, click **Paste**. The **Paste Preview** dialog box appears.

6. Set the table name to Security, and then click **OK**.

The tabular model now contains a pasted security table, as shown in Figure 8.



User Name	Allowed Sales Territory
DOMAIN\testuser1	6
DOMAIN\testuser2	1
DOMAIN\testuser2	2
DOMAIN\testuser2	3
DOMAIN\testuser2	4
DOMAIN\testuser2	5
DOMAIN\testuser2	6

Figure 8: The pasted Security table. Note that you should change the user names to the names of users in your domain before pasting this table into the model.

Next, prepare the user table:

1. Copy and paste the following table into Excel:

User Name
DOMAIN\testuser1
DOMAIN\testuser2

Table 2: The Users table

2. Change DOMAIN\testuser1 and DOMAIN\testuser2 to users on your local domain.
3. Select the modified table and then copy it.
4. Click the tabular model in SQL Server Data Tools.
5. On the **Edit** menu, click **Paste**. The **Paste Preview** dialog box appears.
6. Set the table name to Users, and then click **OK**.

You have now added all of the required tables to the tabular model to implement dynamic security.

Next, create the relationships between the Security table, the Users table, and the DimSalesTerritory table:

1. On the **Model** menu, point to **Model View**, and then click **Diagram View** to switch to Diagram View.
   or
   Click the diagram icon at the bottom right corner of the tabular model.
2. Scroll to the Users table.
3. Drag the User Name column on the Users table and drop it on the User Name column on the Security table to create a relationship.
4. Scroll to the DimSalesTerritory table.
5. Drag the SalesTerritoryKey column on the DimSalesTerritory table and drop it onto the Allowed Sales Territory column on the Security table.

Figure 9 shows a subset of the model after the tables have been added and the relationships created.

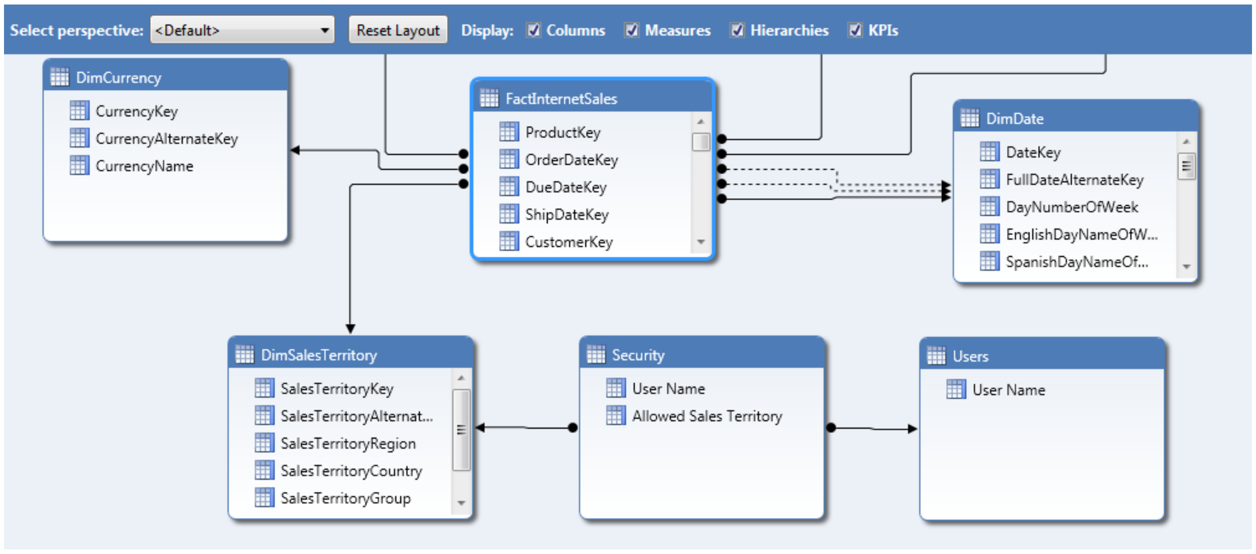


Figure 9: The tabular model after you have added all tables and relationships

Note the following relationships:

- One-to-many relationship between the Users table and the Security table
- One-to-many relationship between the DimSalesTerritory table and the Security table

These relationships will be used by the DAX filters to apply security.

Next, hide the Security and Users tables to prevent them from showing in the field list in client reporting tools like Excel. Note that this is not a security measure. Hiding the fields reduces clutter in the field list, because these tables are not intended to be queried by end users.

To hide the Security and Users tables:

1. Hold down the Ctrl key and then click the Security and Users tables.
2. Right click the tables and then click **Hide from Client Tools**.

Next, apply three row filters:

- The Security table is filtered to prevent users from reading the assigned permissions. This information should not be disclosed to end users. This filter has no effect on any other table in the model because there are no one-to-many relationships from the Security table to other tables.
- The Users table is filtered to restrict the set of allowed rows to the currently logged on Windows user. Alone, this filter would affect the Security table due to the relationship between the Users and the Security table; however, because a more restrictive filter has already been placed on the Security table, the filter has no visible effects. This filter indirectly impacts the DimSalesTerritory table.
- The DimSalesTerritory table is filtered to restrict the list of sales territories to the set of allowed sales territories for the currently logged on Windows user. The filter on the DimSalesTerritory table relies on the existence of the filter on the Users table, because the filtered Users table is used as a parameter in the DAX row filter on the DimSalesTerritory table. The filter on the DimSalesTerritory table directly affects the FactInternetSales table due to the one-to-many relationship between those two tables.

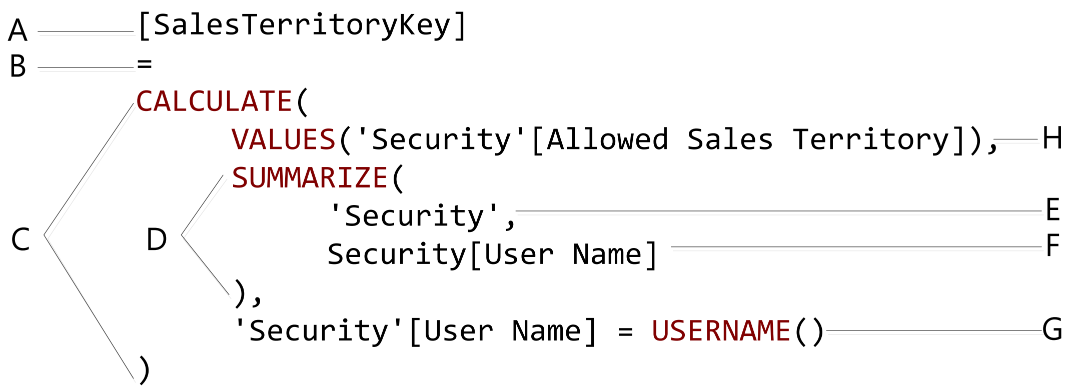
This paper shows two approaches to filtering rows. One approach uses a generalized many-to-many relationship pattern in DAX to look up matching sales territory values based on the filters applied to the Users table. The other approach is more mathematical in nature, and it computes the allowed row set by counting the number of unfiltered rows in the Security table. The second approach is shorter and faster but perhaps less intuitive because it doesn't reference the column being filtered.

To apply the row filters using a generalized many to many DAX pattern:

1. On the **Model** menu, click **Roles**.
   or
   On the Analysis Services toolbar, click the role icon.
2. Click **New** to create a role.
3. Set the permissions for the role to Read. The area for setting DAX filters is enabled.
4. Set the DAX Filter for the Security table to `=FALSE()`. This expression hides all rows from the Security table.
5. Set the DAX Filter for the Users table to `=[User Name]=USERNAME()`.This expression checks each row in the Users table, and allows the row to be shown if the value in the row matches the current Windows user's name. For users with Read access, this row filter will yield either zero or one matching rows in the table, with one row returned if the currently logged on Windows user appears in the Users table and zero rows returned otherwise.

6. Set the DAX Filter for the DimSalesTerritory table to =[SalesTerritoryKey]= CALCULATE(VALUES('Security'[Allowed Sales Territory]), SUMMARIZE('Security', Security[User Name]), 'Security'[User Name]=USERNAME()).

To understand the row filter on the DimSalesTerritory a bit better, let's break it down into the individual syntax elements. Figure 10 shows the syntax elements for the DAX filter.



```
[SalesTerritoryKey]
=
CALCULATE(
     VALUES('Security'[Allowed Sales Territory]),
     SUMMARIZE(
          'Security',
          Security[User Name]
     ),
     'Security'[User Name] = USERNAME()
)
```

Figure 10: The DAX filter for the DimSalesTerritory table

The following list describes the syntax elements as numbered in Figure 10:

A. The left side of the row filter. This is a reference to the SalesTerritoryKey column in the DimSalesTerritory table. This expression is evaluated in a row context, as illustrated in Figures 2 and 3 in the "Permissions" section. Each time this expression is evaluated, the expression returns a single value – the current SalesTerritoryKey value in the current row.
B. The equality operator checks to see whether the current row for DimSalesTerritory table exists in the Security table for the currently logged on user. This is done by comparing the results of expression A with the results of expression C. If A and C are equal, the row in the DimSalesTerritory table is allowed; otherwise, the row is not allowed.
C. The right side of the row filter. The function CALCULATE() evaluates the expression H in the current row context. The row filter uses the CALCULATE() function so that the DAX engine can evaluate the expression with additional filters that take advantage of the many-to-many relationship between the DimSalesTerritory and the Security tables. These filters are specified by expressions D and G.
D. The function SUMMARIZE() creates a single-column in-memory table. This in-memory table contains only the unique user names in the Security table. SUMMARIZE() takes two parameters, E and F, which specify the columns to put in the in-memory table. The SUMMARIZE() reduces the set of user names to only the unique values. This in-memory table is calculated in the context of expression G. This table contains 0 or 1 rows.
E. A reference to the Security table.
F. A reference to the [User Name] column in the Security table.
G. This Boolean expression restricts the set of rows in the in-memory table created by expression D to only those rows corresponding to the currently logged on Windows user.

This restriction is applied by checking to see if the User Name column matches the results returned by the USERNAME() function. If the results match, the user name is added the in-memory table; otherwise, the table is blank.

H. The VALUES() function constructs another in-memory table that contains the set of allowed sales territories for the currently logged on Windows user in the current row context. This calculation is performed by looking up all of the rows for the currently logged on user in the Security table (as filtered by expressions D and G) and fetching the values for the Allowed Sales Territory column. This function returns either one value – the key for the currently selected row in the DimSalesTerritory table – or blank. The reason that at most one value is returned is because of the chain of relationships between the Security table, the Users table and the DimSalesTerritory table. The row context on the DimSalesTerritory table is applied implicitly on the Security table because of the one-to-many relationship between the DimSalesTerritory table and the Security table, thus ensuring a unique value for the Allowed Sales Territory column. The relationship between the Users and the Security table and the filters applied in D and G ensure that there is a unique value for the User Name column.

This calculation is pretty fast because the SUMMARIZE() function in expression D does not iterate over all of the rows of the Security table to create the in-memory table used for the calculation. However, an even faster approach is to count the number of unfiltered rows in the Security table, instead of computing the set of allowed sales territory values.

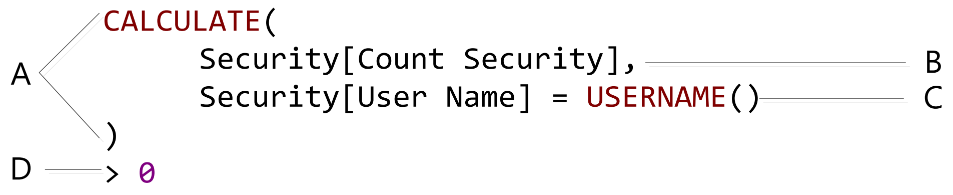
To apply row filters by counting the number of rows in the Security table:

1. On the **Model** menu, point to **Model View**, and then click **Data View** to switch to Data View.
   or
   Click the grid icon at the bottom right corner of the tabular model.
2. Click the sheet tab for the Security table.
3. Click anywhere in the Measure Grid on the Security table, and then type the measure `Count Security:=COUNTROWS(Security)`. This measure counts the number of rows in the Security table.
4. Right click the Count Security measure and then click **Hide from Client Tools**. This measure is not needed for analysis.
5. On the **Model** menu, click **Roles**.
   or
   On the Analysis Services toolbar, click the role icon.
6. Click **New** to create a role.
7. Set the permissions for the role to Read. The area for setting DAX filters is enabled.
8. Set the DAX Filter for the Security table to `=FALSE()`. This expression hides all rows from the Security table.
9. Set the DAX Filter for the Users table to `=[User Name]=USERNAME()`.This expression checks each row in the Users table, and allows the row to be shown if the value in the row matches the current Windows user's name. For users with Read access, this row filter will yield either zero or one matching rows in the table, with one row returned if the

currently logged on Windows user appears in the Users table and zero rows returned otherwise.

10. Set the DAX filter for the DimSalesTerritory table to =CALCULATE(Security[Count Security], Security[User Name]=USERNAME()) > 0

To understand the row filter on the DimSalesTerritory a bit better, let’s break it down into the individual syntax elements. Figure 11 shows the syntax elements for the DAX filter.



```
CALCULATE(
        Security[Count Security],
        Security[User Name] = USERNAME()
)
> 0
```

Figure 11: The DAX filter for the DimSalesTerritory table

The following list describes the syntax elements as numbered in Figure 11.

A. The function CALCULATE() evaluates the expression B in the current row context. The row filter uses the CALCULATE() function so that the DAX engine can evaluate the expression with an additional filter, specified by expression C, that takes advantage of the many-to-many relationship between the DimSalesTerritory and the Security tables.
B. This measure counts the number of rows in the Security table. This measure returns either 1 or blank. The reason the number of rows in the table is at most 1 is because of the chain of relationships between the Security table, the Users table, and the DimSalesTerritory table. The row context on the DimSalesTerritory table is applied implicitly on the Security table because of the one-to-many relationship between the DimSalesTerritory table and the Security table, thus ensuring a unique value for the Allowed Sales Territory column. The relationship between the Users and the Security table and the filter applied in C ensures that there is a unique value for the User Name column, and therefore at most one row to be counted in the table.
C. This Boolean expression restricts the set of rows counted in expression B to only those rows corresponding to the currently logged on Windows user. This restriction is applied by checking to see whether the User Name column matches the results returned by the USERNAME() function. If the results match, the row is counted; otherwise, the row is not counted.
D. If the result of expression B is greater than 0, the row in the DimSalesTerritory table is allowed; otherwise; the row is not allowed.

Now that you have applied the row filter to the table, you can test the role. To test the role using the Analyze in Excel feature:

1. Click the **Members** tab in the Role Manager.
2. Add *DOMAIN\testuser1* and *DOMAIN\testuser2* as role members, substituting users on your domain for these placeholder values.

or

Add a Windows group that has both *DOMAIN\testuser1* and *DOMAIN\testuser2* to the list of role members.

3. Click **OK** to close the Role Manager.
4. On the **Model** menu, click **Analyze in Excel**.
   or
   On the Analysis Services toolbar, click the Excel icon.
5. Check the **Other Windows User** button, and then click **Browse**.
6. In the **Enter the object name to select** box, type *DOMAIN\testuser1*, substituting a user on your domain for this placeholder value.

In Excel, you can now construct a pivot table. Drop DimSalesTerritoryKey on rows and Sum of SalesAmount on values. The pivot table will reflect the row security filters that you applied, and only the values for the test users on your domain appear.

Example: Securing the model using the CUSTOMDATA() function

In this example, security for the Internet sales data in the AdventureWorks database is configured by sales territory, just as in the previous example. A client application maps user names, which do not correspond to Windows user names, to custom data values. These custom data values correspond to one or more allowed sales territories in the tabular model. When users connect to the model using the client application, they can see only the allowed sales territories specified using the custom data value in the connection string.

> **Note** This example is provided for illustration purposes only. Do not secure a client application using custom data in production. Only use custom data to secure middle-tier applications.

The data model for this example is very similar to the one in the previous example, so this paper does not provide step-by-step instructions for preparing the data model. Instead, you can view the tabular model in the CustomDataTest project included with the sample files. You will prepare the data model for your development environment, explore the model, deploy the model, and then test it using the C# client provided.

To configure the CustomDataTest project for your development environment:

1. Open the CustomDataExample solution.
2. From Solution Explorer, double-click to open the Model.bim file.
3. On the **Model** menu, click **Existing Connections**.
   or
   On the Analysis Services toolbar, click the connection icon.
4. Click **Edit** to modify the connection string to the AdventureWorks database.
5. Change the **Server name** to the name of the SQL Server instance hosting your copy of the AdventureWorks database. Save your changes.
6. On the **Model** menu, point to **Process** and then click **Process All**.

or
On the Analysis Services toolbar, click the process icon.
7. On the **Model** menu, click **Roles**.
   or
   On the Analysis Services toolbar, click the role icon.
8. Click the **Members** tab.
9. Add a user on your domain to the list of members. This user must not have administrative access to the Analysis Services instance to which you are deploying the model. Also, you must know the password of this user, as you must start the test client with this user's credentials.
10. Click **OK** to close the Role Manager.

Now that you have configured the model and loaded data into it, you can explore the model. There are a few differences from the previous example. The Security table contains the following values.

CustomDataValue	Allowed Sales Territory
MichaelBlythe	6
HanyingFeng	1
HanyingFeng	2
HanyingFeng	3
HanyingFeng	4
HanyingFeng	5
HanyingFeng	6

Table 3: The Security table

The User Name column was replaced with a CustomDataValue column, which contains the custom data values used in the client application.

Also, the Users table from the previous example was replaced with a CustomDataValues table. That table contains the following values.

CustomDataValue
MichaelBlythe
HanyingFeng

Table 4: The CustomDataValues table

Again, this table is very similar to the structure of the Users table. The CustomDataValues table contains the set of unique values for the custom data values. It serves the same purpose as the Users table as well – the table functions as a bridge table between the Security table and the DimSalesTerritory table. Relationships are defined between the Security table, the CustomDataValues table, and the DimSalesTerritory table so that the row filters cross-filter the appropriate tables.

Open the Role Manager and take a look at the row filters defined on the Read role. The filter on the Security table remains the same – it is set to `=FALSE()` so that users are not allowed to see the allowed sales territories in the Security table. The row filter on the CustomDataValues table is set to =[CustomDataValue]=CUSTOMDATA(). Again, this filter is conceptually similar to the one set on the Users table in the previous example, except that the custom data value passed to Analysis Services in the connection string is used for dynamic security instead of the currently logged on Windows user's name. The row filter on the DimSalesTerritory table takes the generalized many-to-many approach presented in the previous example. The row filter is set to =[SalesTerritoryKey]=CALCULATE(VALUES('Security'[AllowedSalesTerritory]), SUMMARIZE('Security', Security[CustomDataValue]), 'Security'[CustomDataValue]=CUSTOMDATA()). Again, the only difference between the custom data example and the username example is the substitution of the [CustomDataValue] column for the [User Name] column and the substitution of the CUSTOMDATA() function for the USERNAME() function.

After you have configured and explored the model, you can deploy it and test it. The sample application is configured to deploy the model to a tabular instance located at localhost\TABULAR. To deploy to a different location, change the **Server** property in the CustomDataTest project properties before deployment. To deploy the project, from Solution Explorer right-click the CustomDataTest project and then click **Deploy**.

You can now test the dynamic security using the test client. The code for the test client is in the WinFormsCustomDataExample project in the CustomDataExample solution. As the name of the project suggests, the test client has a GUI interface written in WinForms. Figure 12 shows the test client.

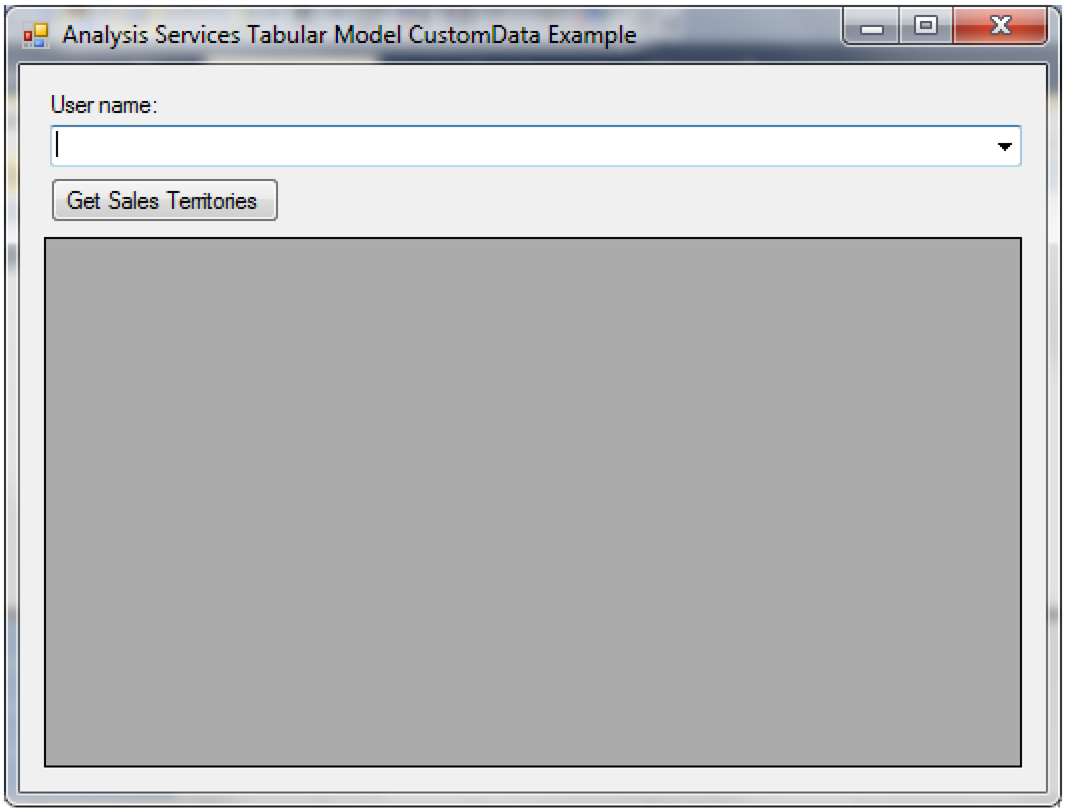


Figure 12: The test client for the model using CUSTOMDATA() for security

To use the test client, select one of the values in the **User name** box and then click **Get Sales Territories**. The click event handler for the Get Sales Territories button maps the supplied user name to a custom data value for the model, appends the custom data value to the connection string for Analysis Services, establishes an ADOMD.NET connection to the model, executes a DAX query to get the data from the DimSalesTerritory table, and then displays the results in a data grid.

The following code listing shows the logic in the Get Sales Territory button click event handler. The logic for supplying the custom data value to Analysis Services is highlighted.

```
DataTable results = new DataTable();
String connectionString = "Data Source=localhost\\Tabular;Initial
Catalog=CustomDataTest;Provider=MSOLAP.5;Integrated Security=SSPI;Impersonation
Level=Impersonate;CustomData=" + comboUserName.SelectedItem.ToString().Replace("
", string.Empty);

using (AdomdConnection connection = new AdomdConnection(connectionString))
{
    connection.Open();
    AdomdCommand command = new AdomdCommand("evaluate DimSalesTerritory order by
DimSalesTerritory[SalesTerritoryKey]", connection);
    AdomdDataAdapter adaptor = new AdomdDataAdapter(command);
    adaptor.Fill(results);
    gridViewResults.DataSource = results.DefaultView;
}
```

The client sends the custom data value to the tabular model in the connection string to Analysis Services.

You cannot verify the functionality of the test client by running it from the development environment. This is because you have administrative permission on the deployment server, so you can read all of the data in the model. To verify the functionality, you must build the test client and run it as the user with read permission on the tabular model instead.

To run the test client:

1. If you did not deploy your tabular model to an instance named localhost\Tabular, change the value of the `connectionString` variable to match the name of the deployment server.
2. From Solution Explorer, right-click the WinFormsCustomDataExample project and then click **Build**.
3. Right-click the WinFormsCustomDataExample project and then click **Open Folder in Windows Explorer**.
4. Copy the file bin\Debug\WinFormsCustomDataExample.exe to a location accessible to the user that you granted read access to the tabular model. For simplicity, this example will assume that you copied the file to C:\ WinFormsCustomDataExample.exe.
5. From the **Start** menu, click **Command Prompt**.
6. At the prompt, type `runas /user:`*`DOMAIN`*`\`*`username`*` C:\WinFormsCustomDataExample.exe` Replace the placeholder with the name of the user with read access to the tabular model.

You can now experiment with the test client and verify that the security works correctly.

The logic in the button click event handler can be migrated to a web application such as an ASP.NET web application. If you are using this code from ASP.NET, grant read permission on the tabular model to the user running the Application Pool for the web application. End users of the web application must not have read access to the tabular model. The web application must authenticate the end users, map the users to a custom data value, and then connect to Analysis Services, passing the appropriate custom data value in the connection string.

Note that you can only use this example security technique from a middle-tier application that has full control over the connection string to Analysis Services, so it can add custom parameters to the connection string. If you are using a middle-tier application that doesn't control the connection string directly on a per-connection basis (such as Reporting Services), you cannot use this security method.

Example: Securing a model using parent/child hierarchies

In this example, information given by employees in response to a survey will be secured. Employees are members of a single organization, with a ragged organizational hierarchy that is up to six levels deep. Each Windows user with access to the tabular model will have access to survey responses for themselves and for those reporting to them (directly or indirectly) in the organization. Users cannot see responses from their peers or responses from those higher up in the organization.

This example is based on data in an Excel workbook. Parts of the tabular model were constructed for you. Your task is to add row security to the model, write a measure for analysis, and construct a pivot table with row security in place.

Before you begin, prepare the data in the Excel workbook:

1. Copy the OrganizationalSurveyExample.xlsx file to a location accessible to the Analysis Services workspace database server. Analysis Services will read data from this file.
2. Open the OrganizationalSurveyExample.xlsx file.
3. Click the Employees tab.
4. Change the UserAlias for a few users to the Windows user names of users in your local domain. You should change the UserAlias values for users in different levels of the hierarchy to test the effects of the row filter. Changing the aliases of andreas0, andy0, and douglas0 achieves this goal.
5. Save and close the Excel file.

Next, prepare the OrganizationalSurveyExample project for analysis:

1. Open the OrganizationalSurveyExample project.
2. On the **Model** menu, click **Existing Connections**.
   or
   On the Analysis Services toolbar, click the connection icon.
3. Click **Edit** to modify the connection string to the Excel workbook.

4. Click **Browse** and then navigate to the location of OrganizationalSurveyExample.xlsx. Save your changes.
5. On the **Model** menu, point to **Process** and then click **Process All**.
   or
   On the Analysis Services toolbar, click the process icon.

The data is loaded into the tabular model and you are ready to add security.

First, take some time to explore the data model in Data View. There are five tables in this model. There is a table of employee information, a table with survey questions, a table of survey responses, and two tables that provide more information about questions and responses.

The Employees table contains several columns of source data and several calculated columns. Two columns of source data, UserID and Manager, form a parent/child relationship. To turn this parent/child relationship into a hierarchy, some DAX calculations are performed in calculated columns. The UserAliasWithDomain column prepends the domain name to the user alias. The OrgHierarchyPath column expands fully the parent/child relationships between the employee and the manager. The individual levels in the hierarchy are computed in the Level1 through Level6 calculated columns. The PATH() and PATHITEM() functions perform these calculations. For more information about creating calculations for parent/child hierarchies, see PowerPivot Denali: Parent child using DAX (http://www.powerpivotblog.nl/powerpivot-denali-parent-child-using-dax). Note that the OrgHierarchyPath contains user aliases, not user IDs, because this makes the row filter for dynamic security easier to understand.

Switch to the Diagram View and look at the Employees table. The calculated columns for computing the levels of the hierarchy are greyed out, indicating that they are hidden from the client tools. Also, you can now see the Employee hierarchy, as illustrated in Figure 13. This hierarchy contains the Level1 through Level6 calculated columns in order. This Employee hierarchy can be dropped into a pivot table, and users can click to expand each individual node in the hierarchy to view child levels. For more information about building hierarchies, see Create and Manage Hierarchies (https://docs.microsoft.com/sql/analysis-services/tabular-models/create-and-manage-hierarchies-ssas-tabular) and Lesson 9: Create Hierarchies (https://docs.microsoft.com/sql/analysis-services/lesson-9-create-hierarchies).

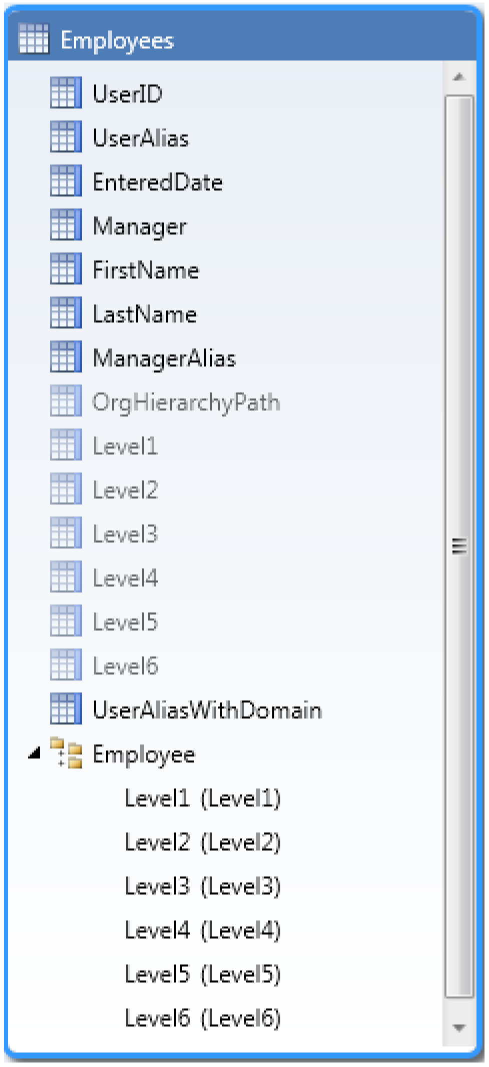


Figure 13: The Employee hierarchy in the Employees table

With this data model in place, you can now implement dynamic security.

To implement dynamic security:

1. With the tabular model in Data View, click the Employees table.
2. Click the UserAliasWithDomain column.
3. In the formula bar, replace *domain* with the name of your domain, in lowercase letters.
4. On the **Model** menu, click **Roles**.
   or
   On the Analysis Services toolbar, click the role icon.
5. Click **New** to create a role.
6. Set the permissions for the role to Read. The area for setting DAX filters is enabled.
7. Set the DAX Filter for the Employees table to =PATHCONTAINS([OrgHierarchyPath], USERNAME())
8. Click the **Members** tab.
9. Click **Add**.

10. In the **Enter the object name to select** box, type *domain\andreas0, domain\douglas0, and domain\andy0*, substituting users on your domain for these placeholder values.

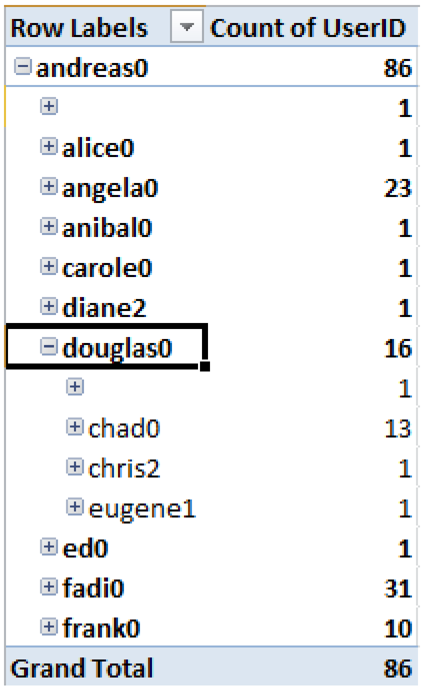
    or

    Click **Object Types** and then click **Groups**. In the **Enter the object name to select** box, type the name of a Windows group that contains the users andreas0, douglas0, and andy0.

The row filter uses the PATHCONTAINS function to see if the hierarchy for the current row contains the name of the currently logged on Windows user and, if so, allows the row to be added to the allowed row set. The first parameter to PATHCONTAINS, [OrgHierarchyPath], contains the hierarchy. The second parameter, USERNAME(), contains the user name. PATHCONTAINS returns true if the currently logged on Windows user's alias exists in the hierarchy, thus allowing the row, and false otherwise.

Compared to the previous examples, implementing dynamic security here is relatively simple. This is because the security filter is set on a single table, so there is no need to traverse relationships to calculate the allowed row set.

Next, compute the number of people in a user's organization, either directly or indirectly. Normally when you use hierarchies, this calculation is simple: if you create a measure on the Employees table with the formula COUNT[UserID] , results roll up all the way to the parent. Figure 14 shows the number of employees in the user andreas0's organization using this formula, as shown when andreas0 connects to the tabular model.

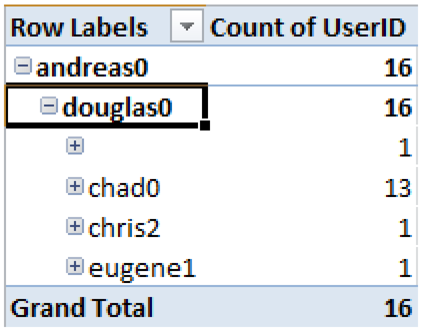


Row Labels	Count of UserID
⊟andreas0	86
⊞	1
⊞alice0	1
⊞angela0	23
⊞anibal0	1
⊞carole0	1
⊞diane2	1
⊟douglas0	16
⊞	1
⊞chad0	13
⊞chris2	1
⊞eugene1	1
⊞ed0	1
⊞fadi0	31
⊞frank0	10
Grand Total	86

Figure 14: A pivot table showing the values for a measure with the formula COUNT[UserID] when using andreas0's credentials

When row security is applied, parent rows can be filtered out of the allowed row set. Although a user can see the parent in a hierarchy, because the name of the parent is stored in a calculated column in an allowed row, the user is not allowed to see the total number of employees for the parent. This is because visual totals are always enabled, so users can see totals only for those rows they are allowed to see. Since the parent's total includes rows that the

Figure 15 shows the same pivot table when connected to the tabular model as douglas0.



Row Labels	Count of UserID
⊟ andreas0	16
⊟ douglas0	16
⊞	1
⊞ chad0	13
⊞ chris2	1
⊞ eugene1	1
Grand Total	16

Figure 15: A pivot table showing the values for a measure with the formula COUNT[UserID] when using douglas0's credentials

The number of users in andreas0's organization has reduced from 86 to 16, which does not accurately reflect the total number of people in andreas0's organization.

For this reason, we do not recommend using hierarchies in pivot tables when row security applies. Although there is a way to compute the correct total values in the hierarchy, the computation requires some complex DAX expressions and is therefore not recommended.

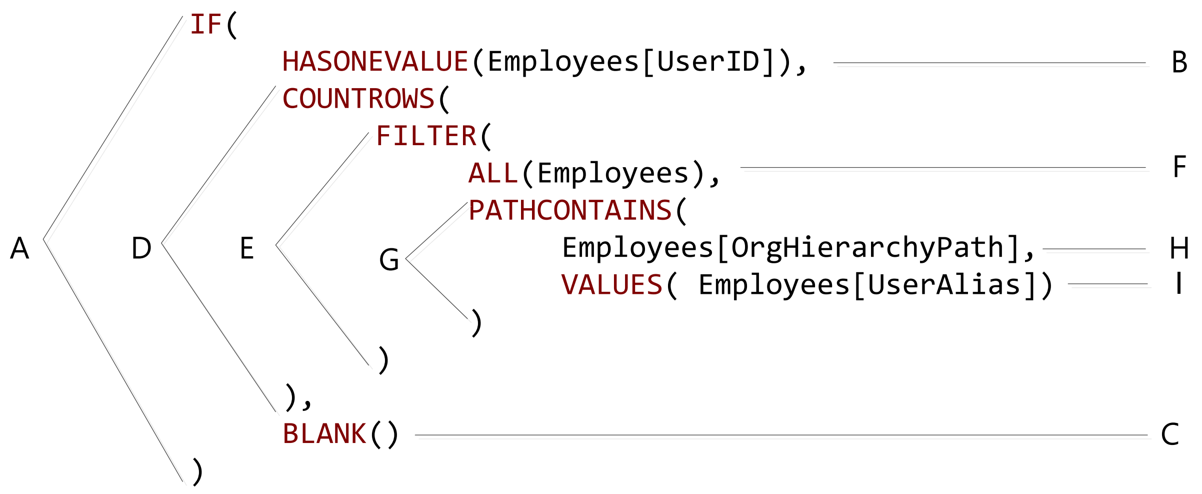
When row security is applied to hierarchies, you should only drop columns into pivot tables. The DAX formulas for measures should then be written so that they are accurate when columns are used in analysis.

The formula for counting the number of employees in a user's organization becomes more complex when you are not using the hierarchy to automatically count the number of employees at each level of the hierarchy. You must do this summarization in DAX.

To create a measure that counts the number of employees in a user's organization:

1. With the tabular model in Data View, click the sheet tab for the Employees table.
2. Click anywhere in the Measure Grid.
3. Paste the following measure into the formula bar:
```
Number in Organization:=IF(HASONEVALUE(Employees[UserID]),
COUNTROWS(FILTER(ALL(Employees),PATHCONTAINS(Employees[OrgHierarchyPath],VALUES(Employees[Us
erAlias])))),
BLANK())
```

To understand the measure a bit better, let's break it down into the individual syntax elements. Figure 16 shows the syntax elements for the measure.



```
IF(
    HASONEVALUE(Employees[UserID]),
    COUNTROWS(
        FILTER(
            ALL(Employees),
            PATHCONTAINS(
                Employees[OrgHierarchyPath],
                VALUES( Employees[UserAlias])
            )
        )
    ),
    BLANK()
)
```

Figure 16: The syntax elements for the Number in Organization measure

The following list describes the syntax elements as numbered in Figure 16.

A. The IF() function performs basic validation. The number of people in the organization is returned only if one employee is selected in the filter context; otherwise, the measure returns blank. The IF function takes three parameters: expressions B, D, and C.

B. The HASONEVALUE() function checks to see whether there is one single value for the UserID column and returns TRUE in that case. Usually you will get a single value by dropping a unique column, such as UserAlias or UserID, into the Row Labels area of a pivot table,

C. The BLANK() function is the return value for the measure if there is more than one employee selected.

D. The COUNTROWS() function counts the number of people in the organization of the selected employee. COUNTROWS() counts the number of rows in an in-memory table constructed by expression E.

E. The FILTER() function constructs an in-memory table with the list of all people in the selected user’s organization. FILTER() takes two parameters: expressions F and G.

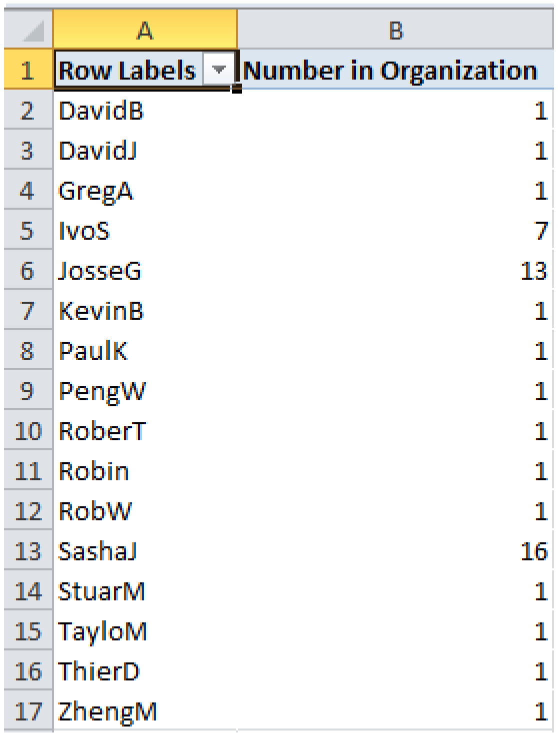
F. The ALL() function removes the filter context from the Employees table so that all rows in the Employee table are available for use by expressions D and E. Previously, expression B verified that there is exactly one row in the Employee table – the row for the currently selected employee. The ALL() function removes this filter.

G. This parameter of expression E adds a filter to the table calculated in expression F, restricting the number of rows in the table to be counted by expression D. The PATHCONTAINS() function is evaluated for each row in the table constructed in expression F, and returns true (thus allowing the row to remain in the table constructed in expression E) if the currently selected user exists in the organizational hierarchy for the row and false (thus removing the row from the table constructed in expression E) otherwise. PATHCONTAINS() takes two parameters, H (the user hierarchy) and I (the search value).

H. A reference to the OrgHierarchyPath column in the Employees table.

I. The VALUES() function returns the string representation of the currently selected employee's alias. Again, an employee is usually selected by dropping the ID or Alias of the employee into the Row Labels area of a pivot table, and the selection reflects the row in the pivot table. The VALUES() function takes a single parameter, which is a reference to the UserAlias column. The UserAlias column is used because the organizational hierarchy is constructed of aliases, so an alias must be used to search the hierarchy.

Figure 17 shows the number of employees for each user in douglas0's organization:



	A	B
1	Row Labels	Number in Organization
2	DavidB	1
3	DavidJ	1
4	GregA	1
5	IvoS	7
6	JosseG	13
7	KevinB	1
8	PaulK	1
9	PengW	1
10	RoberT	1
11	Robin	1
12	RobW	1
13	SashaJ	16
14	StuarM	1
15	TayloM	1
16	ThierD	1
17	ZhengM	1

Figure 17: A pivot table showing the values for the Number in Organization measure when using douglas0's credentials

Notice that row security restricts the number of rows to only those directly or indirectly reporting to Douglas0, and the number in organization is computed per user. Note that this measure is not additive and should never be summarized in a grand total calculation.

Enabling dynamic security using Tabular models using bi-directional relationships.

To create a model for dynamic security:

1. Create a security table in the relational data source that at least contains a column with a list of Windows user names or custom data values. Every user needs to be unique in this table.

2. Create a two-column bridge table in the relational data source. In the first column, specify the same list of Windows user names or custom data values. In the second column, specify the data values that you want to allow users to access for a column in a given table. Usually the same values as you want to secure in your dimension.
3. Import both tables into the tabular model using the Table Import Wizard.
4. Create a relationship between the bridge table and the column that is being secured in the dimension table. The key here is to *turn on Bi-directional relationships* for this relationship and also enable the security filter.
5. Create a relationship between the bridge table created in step 2 and the security table created in step 1.
6. By using Role Manager, create a role with Read permission.
7. Set the row filter for the security table to =[Column Name]=USERNAME() or =[Column Name]=CUSTOMDATA().

There should be one security table per column that contains values that you want to secure. If you want to secure multiple values, create multiple security tables and create relationships and row filters as described earlier.

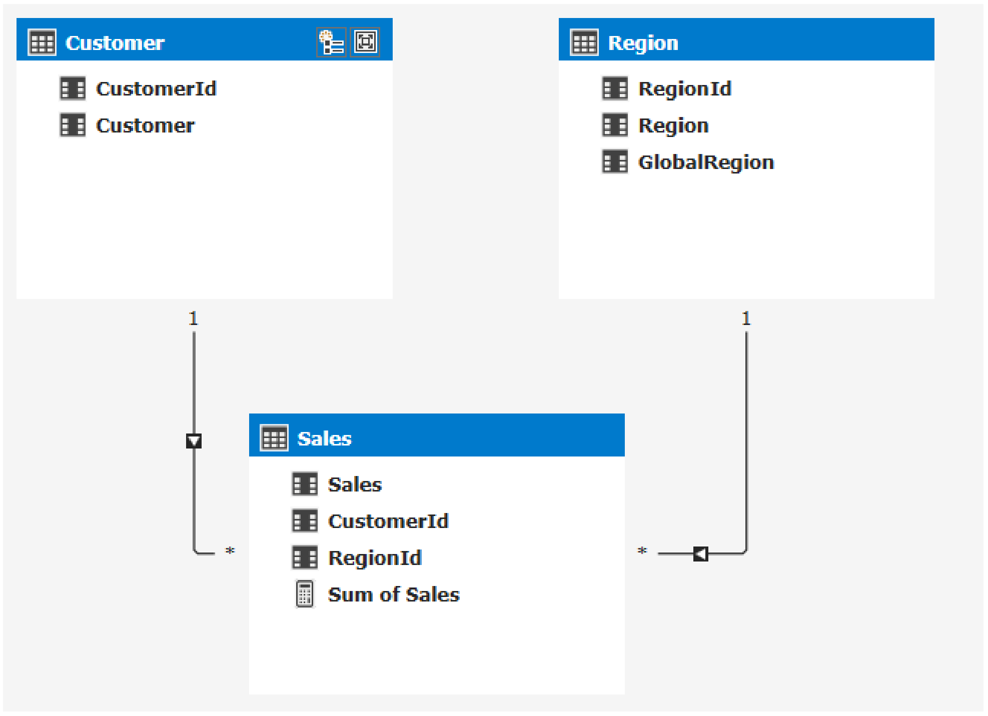
Example: Dynamic row level security and bi-directional relationships

In this example, we will look at how to implement Dynamic Row Level security by using Bi-Directional Cross filtering as it is available in SQL Server 2016, Power BI and Azure Analysis Services. More information on Bi Directional Cross filtering is available in this whitepaper.

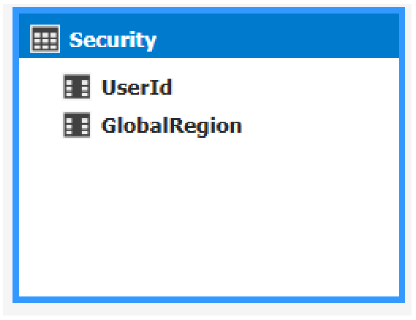
Both RLS and bi-directional relationships work by filtering data. Bi-directional relationships explicitly filter data when the columns are actually used on axis, rows, columns or filters in a PivotTable or any other visuals. RLS implicitly filters data based on the expressions defined in the roles.

In this example, we have three tables: Customer, Region and their Sales. We want to add dynamic row level security to this model based on the user name or login id of the user currently logged on, I want to secure my model not per region, but a grouping of regions called GlobalRegion.

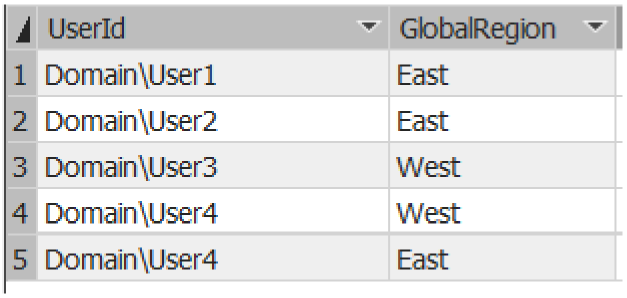
This is the schema for the model:



Next, I add a table that declares which user belongs to a globalregion:



The goal here is to restrict access to the data based on the user name the user uses to connect to the model. The data looks like this:

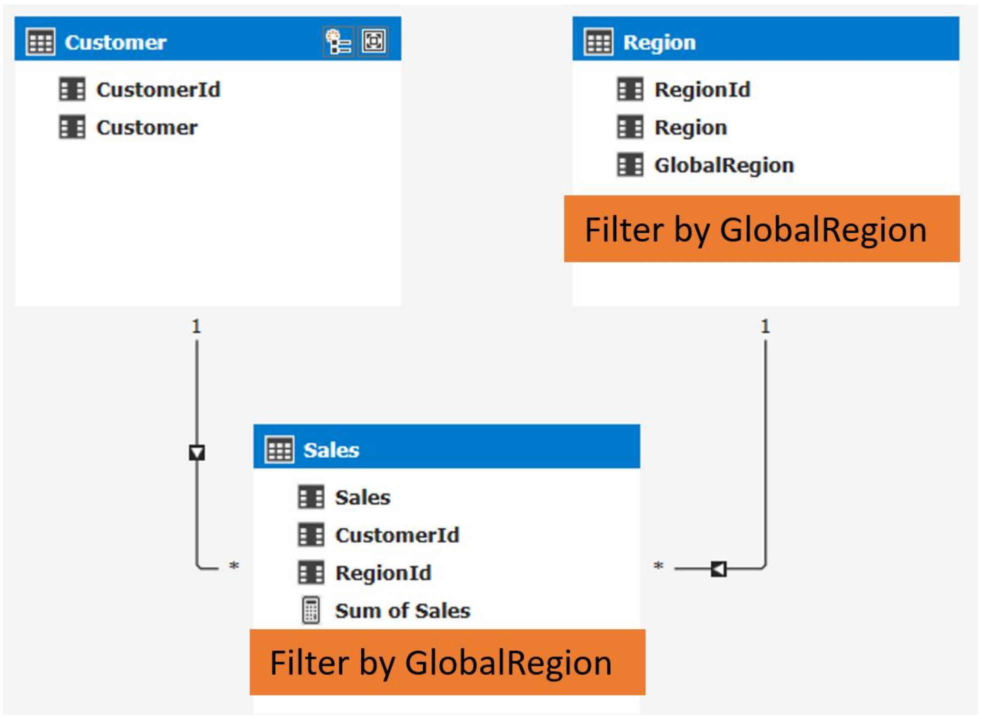


	UserId	GlobalRegion
1	Domain\User1	East
2	Domain\User2	East
3	Domain\User3	West
4	Domain\User4	West
5	Domain\User4	East

To solve this problem without bi-directional cross-filtering, we would use some complicated DAX to a row level security expression on the region table that will determine the global region the current connected user has access to. To do that we can create this DAX expression:

```
=Region[GlobalRegion]=LOOKUPVALUE(Security[GlobalRegion]
          , Security[UserId], USERNAME()
          , Region[GlobalRegion], Security[GlobalRegion])
```

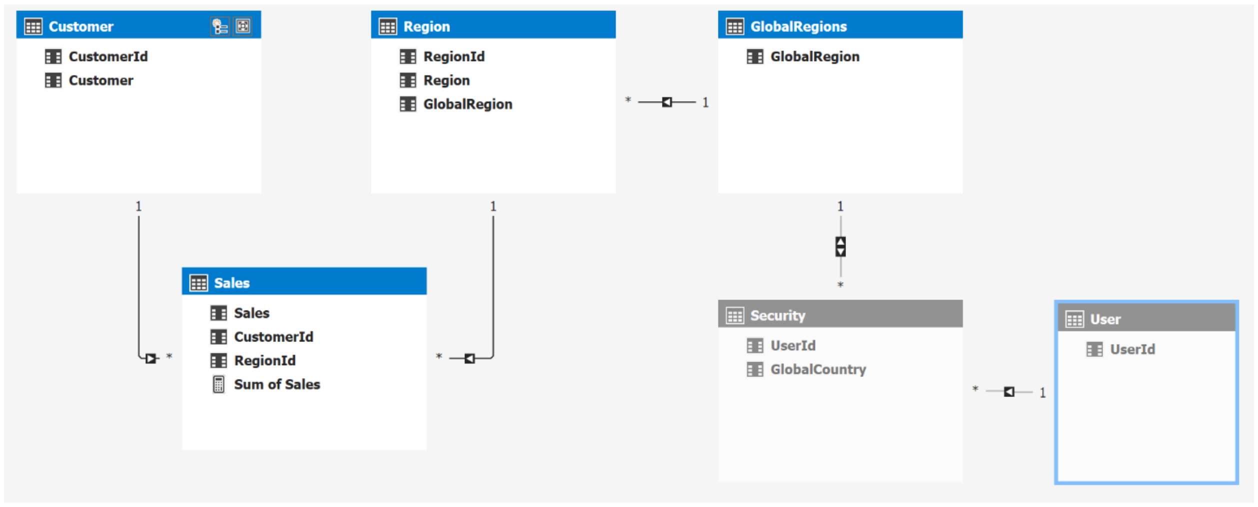
This would create a filter on the Region table and return just the rows in the Region table as returned by the DAX expression from the Security table. This DAX expression will look up any values for GlobalRegion based on the username of the user connecting to the SSAS model.



This is a pretty complicated scenario and won’t work for models in DirectQuery mode as the LOOKUPVALUE function isn’t supported. For that reason, we introduced a new property in SQL Server 2016 that will allow you to leverage bi-directional cross-filtering to enable the same scenario.

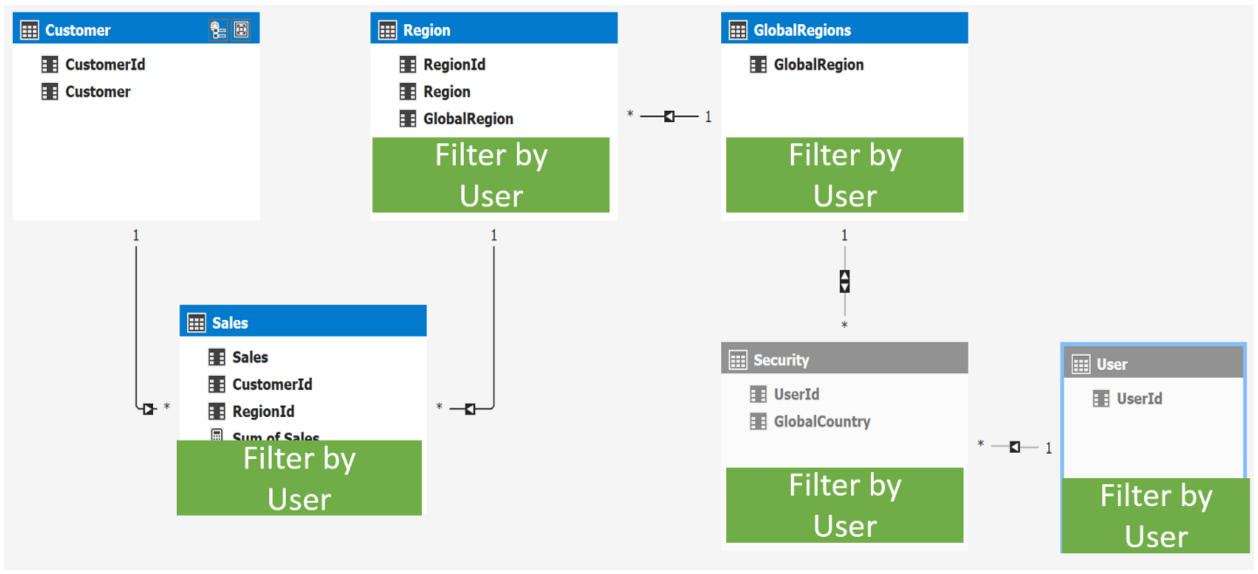
> Note: when using Power BI you should use the USERPRINCIPALNAME() function instead of the USERNAME() function. This will make sure the actual username will get returned, USERNAME will give you an internal representation of the username in Power BI. For Azure Analysis service both functions will return the Azure AD UPN.

Let’s take a look at how we would model this using bi-directional relationships. Instead of leveraging the DAX function to find the username and filter the table, we’ll be using the relationships in the model itself. By extending the model with some additional tables and the new bi-directional cross-filter relationship we can make this possible:



Observe we now have a separate User table and a separate GlobalRegions tables with an bridge table in the middle that connects the table to be secured with the security table. The relationship between User and GlobalRegion is a many to many relationship as a user can belong to many global regions and a global region can contain many users.

The idea here is that we can add a simple row level security filter on the User[UserId] column like =USERNAME() and this filter will now be applied to all of the related tables:



There one more item to cover, as you can see the relationship between Security and GlobalRegions is set to bidirectional cross-filtering, by default this will not be honored for RLS scenarios. Luckily there is a way to get this working for RLS scenarios as well. To turn it on I go to the diagram view in SSDT and select the relationship:

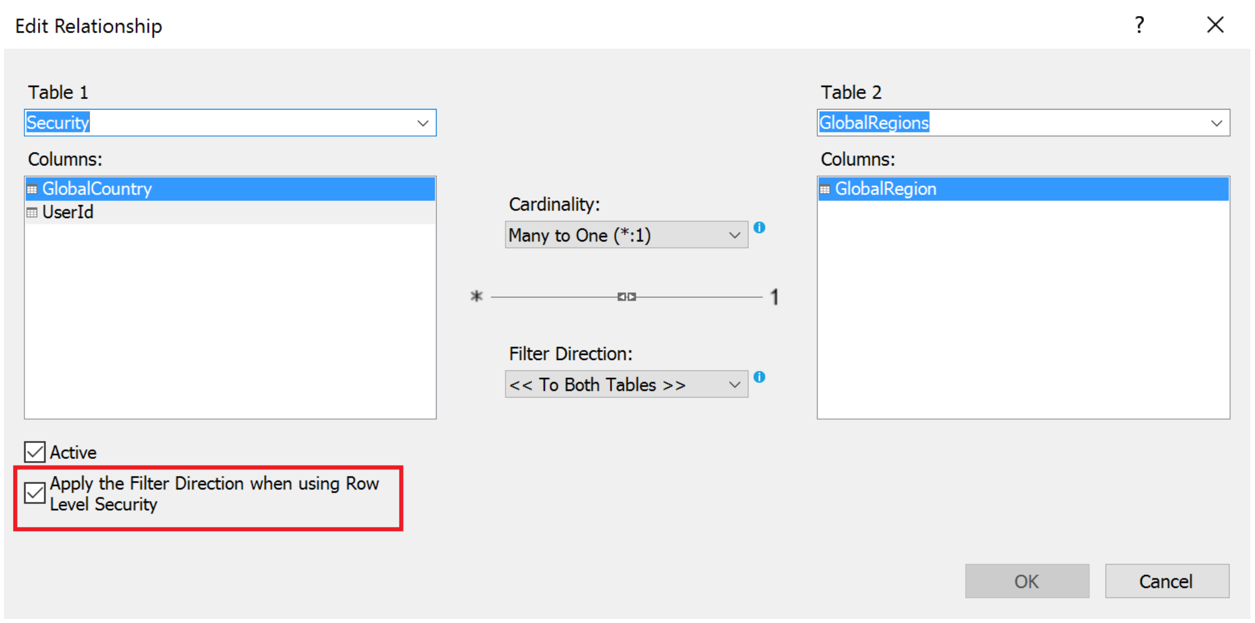

Edit Relationship

Table 1: Security

Columns:
- GlobalCountry
- UserId

Cardinality: Many to One (*:1)

Filter Direction: << To Both Tables >>

Table 2: GlobalRegions

Columns:
- GlobalRegion

☑ Active

☑ Apply the Filter Direction when using Row Level Security

Here I can select the property that applies the filter direction when using Row Level Security.

This property only works for certain models and is designed to follow the above pattern with dynamic row level security as shown in the example above. Trying to use this in for other patterns might not work and Analysis Services might throw an error to warn you.

Now this is the basic pattern for dynamic row level security using bi-directional relationships, many of the other examples like using CUSTOMDATA() from above can be used as variation on this theme as well.

Managing roles and permissions

After you have deployed a model with roles, you (or your server administrator) must perform some security-related management tasks. Usually, this is limited to adding or removing role members, but you can also create, edit, and delete roles and add or remove administrators on the tabular instance. You can perform management tasks using any management tool for Analysis Services, including SQL Server Management Studio, AMO, and AMO for PowerShell. You can also use XMLA scripts to manage roles, though a discussion of XMLA for role management is outside the scope of this document.

Adding and removing administrators from a tabular instance

If you installed your tabular instance of Analysis Services using the default configuration settings, the following users are administrators on the tabular instance:

- Members of the local administrator group on the machine where Analysis Services is installed
- The service account for Analysis Services
- Any user that you specified as an administrator in SQL Server setup

You can change these settings after you have installed the tabular instance by modifying the server properties in SQL Server Management Studio.

To change the permissions for the local administrator group:

1. From Object Explorer, right-click the instance that you want to change and then click **Properties**.
2. Click **General**.
3. Click **Show Advanced (All) Properties**.
4. Change the value of the **Security \ BuiltInAdminsAreServerAdmins** property. To disallow administrative permissions on Analysis Services, set the property to false; otherwise, set the property to true (the default).

To change the permissions for the service account:

1. From Object Explorer, right-click the instance that you want to change and then click **Properties**.
2. Click **General**.
3. Click **Show Advanced (All) Properties**.
4. Change the value of the **Security \ ServiceAccountIsServerAdmin** property. To disallow administrative permissions on Analysis Services, set the property to false; otherwise, set the property to true (the default).

To allow or revoke administrative permission on Analysis Services to individual users or groups:

1. From Object Explorer, right-click the instance that you want to change and then click **Properties**.
2. Click **Security**.
3. Add or remove Windows users or groups from the list of users with administrative permission to the server.

Using SQL Server Management Studio to manage roles

You can create, edit, and delete roles from SQL Server Management Studio. For more information about how to use SQL Server Management Studio to manage roles, see Manage Roles by Using SSMS (https://docs.microsoft.com/sql/analysis-services/tabular-models/manage-roles-by-using-ssms-ssas-tabular). We recommend that you use SQL Server Management Studio primarily to add and remove role members. Roles are best created in SQL Server Data Tools.

You can also script out a role definition from SQL Server Management Studio.

To script out a role definition:

1. From Object Explorer, navigate to the role that you want to script out.
2. Right-click the role and then click **Properties**.
3. Click **Script**.

Only the script created from the Properties page contains the row filter definitions. If you right-click the role in Object Explorer and then click a scripting option (create, alter, or delete), the script generated does not include the row filters and cannot be used for administrative purposes.

If you are an administrator on the tabular instance, you can also use SQL Server Management Studio to test security for the tabular model and to browse a tabular model using a different role or user name. For more information, see “Testing roles”.

Using AMO to manage roles

You should only use AMO to add and remove role members. Although the AMO framework does have methods to create roles, delete roles, and modify role permissions, these methods may not be supported in future releases of SQL Server.

Programs that add or remove role members must be run by users with administrative permission on the tabular instance or database that is being modified. Users with only read or process permission do not have sufficient rights to add or remove role members.

The following C# code shows how to add a role member by name. This code uses the role created in the CustomDataTest example provided earlier.

```
Server s = new Server();
s.Connect(".\\TABULAR");

Database db = s.Databases.FindByName("CustomDataTest");
Role r = db.Roles.FindByName("Role");

r.Members.Add(new RoleMember("domain\\username"));
r.Update();

s.Disconnect();
```

The code does the following:

- Connects to a tabular instance named “.\TABULAR”.
- Gets the role named “Role” from the “CustomDataTest” database. This is a two-step operation: first the program finds the database and then the program finds the role.
- Adds the user named “domain\username” to the role named “Role”. This is a two-step operation: first the program queues up the role member addition and then the program updates the role on the server.
- Disconnects from the server.

The following C# code shows how to remove a role member by name.

```
Server s = new Server();
s.Connect(".\\TABULAR");

Database db = s.Databases.FindByName("CustomDataTest");
Role r = db.Roles.FindByName("Role");
```

```
r.Members.RemoveAt(r.Members.IndexOf(r.Members.Cast<RoleMember>().First(user =>
user.Name == "domain\\username")));
r.Update();

s.Disconnect();
```

The code does the following:

- Connects to a tabular instance named ".\TABULAR".
- Gets the role named "Role" from the "CustomDataTest" database. This is a two-step operation: first the program finds the database and then the program finds the role.
- Removes the user named "domain\username" to the role named "Role". Because the list of role members is of type RoleMemberCollection, which implements IList, the role member list must be cast to the generic IList<RoleMember> before the program can search for a role member by name using the LINQ extension method First(). After the program queues up the role member removal, the program then updates the role on the server.
- Disconnects from the server.

Notice that in both examples, the roles on the server are not updated until r.Update() is called. This means that you can queue many add or remove operations before updating the server, which improves performance.

For readability, these programs do not handle errors. Before using these examples in production, add some error handling to the code.

Using AMO for PowerShell to manage roles

SQL Server 2012 introduced 11 Windows PowerShell cmdlets for administering both multidimensional and tabular instances of Analysis Services. For general information about using PowerShell with Analysis Services, see PowerShell scripting in Analysis Services (https://docs.microsoft.com/sql/analysis-services/instances/powershell-scripting-in-analysis-services). For a list of available cmdlets, see Analysis Services PowerShell Reference (https://docs.microsoft.com/sql/analysis-services/powershell/analysis-services-powershell-reference).

You can use the Add-RoleMember and Remove-RoleMember cmdlets to add and remove role members. These cmdlets can be used interactively in the sqlps shell or unattended in scripts or SQL Server Agent jobs. For more information about Add-RoleMember, see Add-RoleMember cmdlet (https://docs.microsoft.com/sql/analysis-services/powershell/add-rolemember-cmdlet). For more information about Remove-RoleMember, see Remove-RoleMember cmdlet (https://docs.microsoft.com/sql/analysis-services/powershell/remove-rolemember-cmdlet).

If you are using these cmdlets interactively, you must be an administrator on the tabular instance or be a member of a role with administrative permission on the tabular database for these cmdlets to execute successfully. If these cmdlets are part of an unattended script or a SQL Server agent job, the user running the script or job must have administrative permission on the tabular instance or database for the cmdlets to execute successfully. Users with only read or process permission do not have sufficient rights to add or remove role members.

Managing connections to relational data sources from Analysis Services

This section of the whitepaper describes some security considerations for connecting to relational data sources. This information is relevant for tabular models that are not DirectQuery enabled. DirectQuery enabled models have a different set of considerations and a discussion of those considerations is out of the scope of this document. For more information about DirectQuery, see Using DirectQuery in the Tabular BI Semantic Model (http://msdn.microsoft.com/en-us/library/hh965743.aspx).

Figure 18 shows a typical machine topology for an in-memory tabular workload.

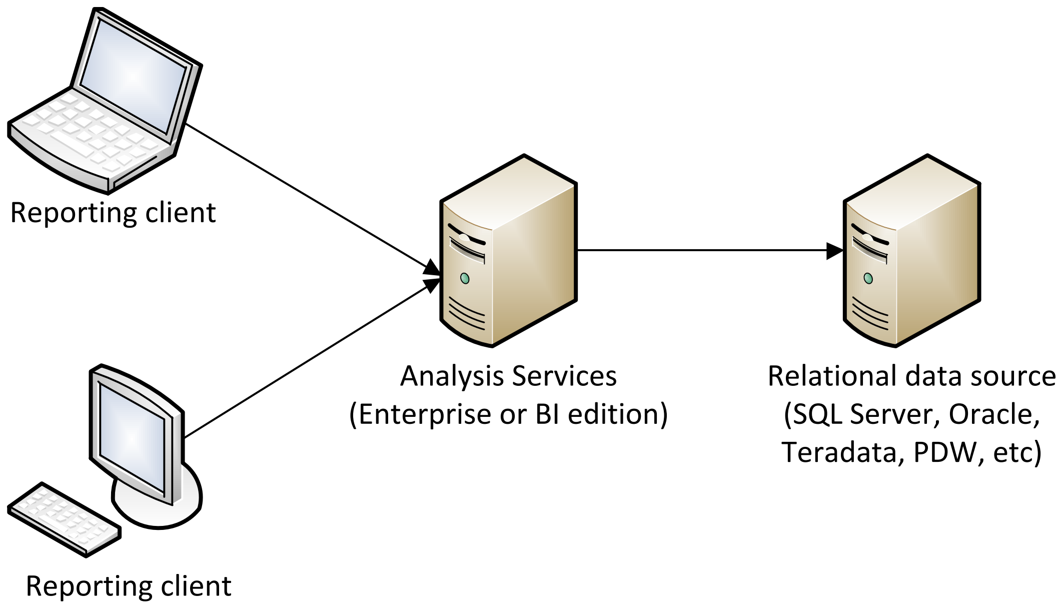


Figure 18: A typical machine topology for an in-memory tabular workload.

Analysis Services queries one or more relational data sources to the fetch data that is loaded into the tabular model. End users of reporting clients query Analysis Services, which returns results based on data that it previously loaded into memory.

Analysis Services uses the impersonation settings to determine the credentials to use when it queries the relational data source. For tabular models running in in-memory mode, three types of impersonation are supported:

- Impersonating a named Windows user
- Impersonating the Analysis Services service account
- Inheriting the impersonation settings defined on the tabular database

The impersonation settings for a data source are initially defined in SQL Server Data Tools when data is imported using the Import Wizard. These settings can be modified in SQL Server Data Tools in the **Existing Connections** dialog box. Only the first two options are available in the Import Wizard.

Before you deploy the model, you can change the impersonation settings in the Deployment Wizard so that you are using the appropriate settings for the production data source. After deployment, you can change the impersonation settings by modifying the connection properties in SQL Server Management Studio.

We recommend that if you are connecting to SQL Server using integrated security (the default), configure your model to impersonate a specific Windows user for connecting to a data source when processing. The Analysis Services service account should have the least possible permissions on the server, and generally it should not have access to data sources.

If Analysis Services and the relational data source are in the same domain, Analysis Services can simply pass the credentials to the data source without additional configuration. However, if there is a domain or forest boundary between Analysis Services and the data source, there must be a trust relationship between the two domains.

Impersonation credentials are required even if another method, such as SQL authentication, is used by the relational data source to authenticate the user before returning query results. The impersonation credentials are used to connect to the data source. This connection attempt may fail if the user that Analysis Services is impersonating does not have network access to the data source. After the connection is established, the second authentication method (if applicable) is used by the data source to return the query results.

SQL Server Data Tools also connects to relational data sources while modeling. Figure 19 shows a typical machine topology in a development environment.

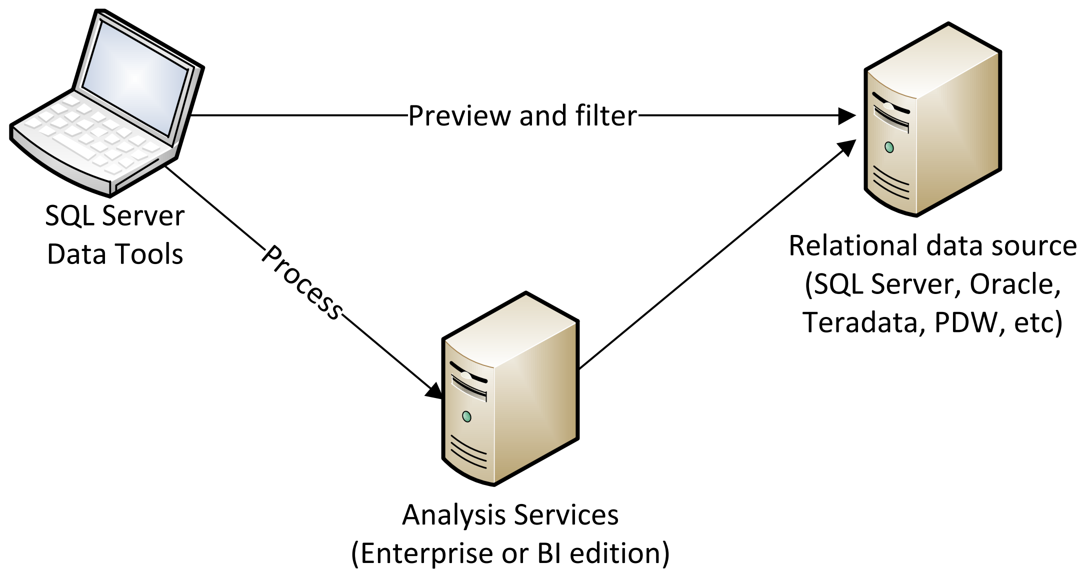

Preview and filter
SQL Server Data Tools
Process
Relational data source (SQL Server, Oracle, Teradata, PDW, etc)
Analysis Services (Enterprise or BI edition)

Figure 19: A typical topology in a development environment.

SQL Server Data Tools queries the relational data source when the user previews data from the relational data source in the Import Wizard, in the **Table Properties** dialog box, or in the Partition Manager. When SQL Server Data Tools connects to the relational data source, it impersonates the current user's credentials. This is because SQL Server Data Tools does not have access to the impersonation credentials stored in the workspace database and these preview and filter operations have no impact on the workspace database.

When the user processes data in SQL Server Data Tools, Analysis Services uses the credentials specified in the Import Wizard or the **Existing Connections** dialog box to query the data source.

Managing connections to the tabular model

As described in the section "Connecting to tabular models", there are several ways that users can connect to tabular models. Users can connect directly using a connection string or .bism file, or indirectly using an .rsds file or through a middle-tier application. Each connection method has its own set of security requirements. This section of the whitepaper describes the requirements in each scenario.

Connecting directly to a tabular model using a connection string

Many client reporting applications, such as Excel, allow users to specify a connection string to connect to Analysis Services. These connections can be entered manually by the user or connections can be saved in an Office Data Connection (.odc) file for reuse. Note that Power View cannot connect to tabular models using this method.

The following table summarizes the major security design aspects for using direct connections to the tabular model.

Design aspect	Configuration
Topology	Usually, reporting clients and the tabular instance reside on the same domain.
Credentials	All users must use Windows credentials. These credentials are passed directly to Analysis Services from the reporting client.
Authentication	Analysis Services authenticates end users of the reporting client using their Windows credentials unless they connect to Analysis Services over HTTP.
Permissions	All users of the reporting client must be members of a role with read permission on the tabular database.
Dynamic security	Dynamic security can be implemented using the USERNAME() function. The CUSTOMDATA() function should not be used for security when clients connect directly to Analysis Services.
Kerberos	Not required between client and tabular instance if there is a single hop between the client and the tabular instance. If there are multiple hops, for example if domain boundaries are crossed, Kerberos is required.

Table 5: Security considerations for using direct connections to Analysis Services

Usually, clients connect directly to Analysis Services by specifying the server and instance name. However, you can configure a tabular instance for HTTP or HTTPS access instead. A discussion of configuring HTTP access to Analysis Services is outside of the scope of this document. For more information about configuring HTTP access, see Configure HTTP Access to Analysis Services on Internet Information Services 7.0 (http://msdn.microsoft.com/en-us/library/gg492140.aspx). When HTTP access is configured, authentication happens first on IIS. Then, depending on the authentication method used for http access, a set of Windows user credentials is passed to Analysis Services. For more information about IIS authentication, see Configure HTTP Access to Analysis Services on IIS 8.0 (https://docs.microsoft.com/en-us/sql/analysis-services/instances/configure-http-access-to-analysis-services-on-iis-8-0).

Also, reporting clients typically reside on the same domain as the tabular instance. However, reporting clients can communicate with tabular instances across domain or forest boundaries if there is a trust relationship between the two domains. A discussion of this configuration is outside of the scope of this document.

Connecting to a tabular model using a .bism file

The BI semantic model connection file is a new content type for SharePoint that enables you to connect to Analysis Services databases or PowerPivot for SharePoint models from reporting clients. This content type is available when PowerPivot for SharePoint is configured on the farm. For more information about .bism files, see PowerPivot BI Semantic Model Connection (https://docs.microsoft.com/sql/analysis-services/power-pivot-sharepoint/power-pivot-bi-semantic-model-connection-bism).

.bism files are especially useful when Power View is the reporting client for the tabular model. When there is a .bism file in a SharePoint library, you can create a Power View report using the **Create Power View Report** quick launch command. You can also use .bism files from other reporting clients, including Excel, to establish a connection to a tabular model.

The following table summarizes the major security design aspects when .bism files are used to connect to a tabular model from Power View.

Design aspect	Configuration
Topology	There is a SharePoint farm that hosts PowerPivot for SharePoint and Reporting Services. The tabular instance of Analysis Services typically resides outside of the SharePoint farm.
Credentials	All users must use Windows credentials to connect to Power View.
Authentication	Reporting Services first tries to connect to Analysis Services by impersonating the user running Power View. If Kerberos constrained delegation is configured, this connection succeeds. Otherwise, Reporting Services tries to connect to Analysis Services using the credentials of the Reporting Services service account, specifying the name of the Power View user as the

	EffectiveUserName in the connection string. If the Reporting Services service account is an administrator on the tabular instance, the connection succeeds, otherwise the connection attempt fails with an error.
Permissions	All Power View users must be members of a role with read permission on the tabular database. If Kerberos with constrained delegation is not configured, the Reporting Services service account must be an administrator in the tabular instance.
Dynamic security	Dynamic security can be implemented using the USERNAME() function. The CUSTOMDATA() function cannot be used for security because additional connection properties cannot be added to .bism files using the .bism file editor in SharePoint.
Kerberos	Kerberos is required, unless the Reporting Services service account is an administrator on the tabular instance or the tabular instance is on a machine inside the SharePoint farm.

Table 6: Security considerations when using .bism files to connect to tabular models from Power View

For more information about configuring Power View and .bism files Analysis Services is outside of the SharePoint farm, see Power View, Tabular mode databases, Kerberos and SharePoint (http://www.powerpivotblog.nl/power-view-tabular-mode-databases-sharepoint-and-kerberos) and http://www.sqlpass.org/summit/2011/Speakers/CallForSpeakers/SessionDetail.aspx?sid=2027.

Connecting to a tabular model from Excel using a .bism file has a different set of security considerations than those for Power View. This is because credentials are not delegated from SharePoint to Analysis Services when Excel is used; instead, credentials are passed directly from Excel to Analysis Services.

The following table summarizes the major security design aspects when .bism files are used to connect to a tabular model from Excel.

Design aspect	Configuration
Topology	There is a SharePoint farm that hosts PowerPivot for SharePoint. The tabular instance of Analysis Services typically resides outside of the SharePoint farm. Excel clients typically reside in the same domain as the tabular instance.
Credentials	All users must use Windows credentials to connect to Excel.
Authentication	Analysis Services authenticates Excel users using their Windows credentials.
Permissions	All Excel users must be members of a role with read permission on the tabular database.
Dynamic security	Dynamic security can be implemented using the USERNAME() function. The CUSTOMDATA() function cannot be used for security if you are using the quick launch commands in SharePoint to create Excel files because additional connection properties cannot be added to .bism files.

Kerberos	Not required between client and tabular instance when both are on the same domain.

Table 7: Security considerations when using .bism files to connect to tabular models from Excel

The same security considerations about HTTP, HTTPS, and cross-domain connectivity described in the section “Connecting directly to a tabular model using a connection string” also apply when connecting to a tabular instance from Excel using a .bism file. For details, see the previous section.

You can use .bism files to connect to tabular models in other scenarios. For example, you can use a .bism filename in the place of a database name in the connection string to Analysis Services. The security considerations in these scenarios are similar to those when connecting to a tabular model from Excel using a .bism file. The main difference is that if you are implementing a middle-tier application that connects to a tabular model using a .bism file, you can use CUSTOMDATA() to secure the tabular model.

Connecting to a tabular model using an .rsds file

.rsds files are used by Reporting Services to connect to Analysis Services models when Reporting Services is running in SharePoint Integrated Mode. For more information about .rsds files, see Create, Modify, and Delete Shared Data Sources (SSRS)(https://docs.microsoft.com/sql/reporting-services/report-data/create-modify-and-delete-shared-data-sources-ssrs).

.rsds files are useful in the following scenarios:

- Use when Power View is configured on the SharePoint farm but PowerPivot for SharePoint is not. .rsds files can be used as the data source for Power View reports instead of .bism files.
- Use when connecting to Reporting Services reports using credentials that are not Windows credentials.
- Use when Analysis Services resides outside of the SharePoint farm, and configuring Kerberos or elevating the privileges of the account running the Reporting Services application are not acceptable. You can configure .rsds files to use stored credentials, and these credentials do not have to be delegated.

The following table summarizes the major security design aspects when .rsds files are used to connect to a tabular model.

Design aspect	Configuration
Topology	The .rsds file is uploaded to the SharePoint farm hosting Reporting Services. The tabular instance typically resides on a machine outside of the SharePoint farm.
Credentials	End users can connect to Reporting Services using Windows credentials, no credentials, or other credentials.

	Reporting Services either impersonates the Windows user connected to the report or sends stored credentials to the tabular instance.
Authentication	If impersonation is used, Analysis Services authenticates the users connected to the reports. If stored credentials are used, Reporting Services authenticates the users connected to the reports and then passes a set of stored credentials to Analysis Services. Analysis Services only authenticates the stored credentials.
Permissions	When impersonation is used, all Reporting Services users must be members of a role with read permission on the tabular database. When stored credentials are used, only the account specified in the .rsds file must be a member of a role with read permission on the tabular database.
Dynamic security	Dynamic security can be implemented using the USERNAME() function. The CUSTOMDATA() function cannot be used for security because users can manually edit the connection string in the .rsds file, potentially causing unintended data access.
Kerberos	Not required unless impersonating a Windows user across SharePoint farm boundaries.

Table 8: Security considerations when using .rsds files

For more information about configuring Reporting Services to connect to tabular databases, see http://www.sqlpass.org/summit/2011/Speakers/CallForSpeakers/SessionDetail.aspx?sid=2027

Connecting to a tabular model from a middle-tier application

One advantage of using custom middle-tier applications to connect to a tabular instance is that you can use custom dynamic security using the CUSTOMDATA() function. You can use CUSTOMDATA() in this scenario because access to Analysis Services is restricted to only the user specified in the middle-tier application. End users cannot craft their own connection strings, so they can’t get access to data unintentionally.

Otherwise, using a custom middle-tier application is conceptually similar to other multi-hop scenarios using .bism or .rsds files.

The following table summarizes the major security design aspects when .rsds files are used to connect to a tabular model.

Design aspect	Configuration
Topology	The middle-tier application typically connects to a tabular instance on a different machine in the same domain.
Credentials	End users can connect to the reporting client application using Windows credentials, no credentials, or other credentials. The middle-tier application either delegates Windows user credentials to Analysis Services or, if an authentication method

	other than Windows authentication is used, maps the supplied credentials to a Windows user account and connects to Analysis Services using the credentials stored in the middle-tier application.
Authentication	If credentials are delegated, Analysis Services authenticates the users connected to the reports. If the middle-tier application uses stored credentials, Analysis Services only authenticates the stored credentials. The middle-tier application authenticates the end users of reports.
Permissions	When credentials are delegated, all users of the reporting client must be members of a role with read permission on the tabular database. When stored credentials are used, only the account specified by the middle-tier application must be a member of a role with read permission on the tabular database.
Dynamic security	Dynamic security can be implemented using the USERNAME() function. The CUSTOMDATA() function can be used when the middle-tier application uses stored credentials to connect to the tabular model, and end users of reports do not have access to the underlying tabular database.
Kerberos	Required if delegating credentials. Not required if using stored credentials or if using the EffectiveUserName connection string parameter to delegate a user's credentials.

Table 9: Security considerations when connecting to a middle-tier application from Analysis Services

Security in the modeling environment (SQL Server Data Tools)

There are some special security considerations for the SQL Server Data Tools modeling environment. These considerations pertain to the configuration of the workspace database instance, the handling of tabular project backups, and the impersonation settings used by SQL Server Data Tools when connecting to data sources.

Before you can use SQL Server Data Tools, you must specify a tabular instance to use as the workspace database server instance. You must be an administrator on this tabular instance. While you are working, all changes that you make in SQL Server Data Tools are automatically reflected on the workspace database instance.

Any administrator on the tabular instance and any role member for the model can access the workspace database, even before you deploy the model. If you do not want others to see changes before the model is deployed, install the tabular instance on the same machine as SQL Server Data Tools, ensure that no other users have administrative access to the machine, and ensure that the Analysis Services instance cannot be reached through the firewall. This is especially important for DirectQuery enabled models, because the workspace database contains cached data by default, even if the model is a DirectQuery only model and will not contain any cached data when it is deployed.

Also, when choosing a workspace database instance, you must consider the permissions given to the service account running the Analysis Services instance. By default, the service account is set to the NT Service\MSSQLServerOLAPService account, which does not have permission to access files on network shares or in user folders such as the My Documents folder. To enable all functionality in SQL Server Data Tools, including importing metadata and data from PowerPivot workbooks and taking backups of tabular projects, the service account for the workspace database instance must have access to these file locations. Consider changing the user running the service account to one that has sufficient permission to access these common file locations. For general information about configuring service accounts, see Configure Service Accounts (https://docs.microsoft.com/sql/analysis-services/instances/configure-service-accounts-analysis-services). For information about configuring the workspace database server and its service account, see Configuring a workspace data server (http://blogs.msdn.com/b/cathyk/archive/2011/10/03/configuring-a-workspace-database-server.aspx).

There are a few other security considerations aside from workspace database server configuration. If you have configured SQL Server Data Tools to take backups of tabular projects when you save the project, these backups are unencrypted. Ensure that these files are stored in a secure location that is inaccessible to other users. For more information about using backups in SQL Server Data Tools, see Working with backups in the tabular designer (http://blogs.msdn.com/b/cathyk/archive/2011/09/20/working-with-backups-in-the-tabular-designer.aspx). Also, the user running SQL Server Data Tools must have access to relational data sources for some user interface components to work correctly. For more information, see “Managing connections to relational data sources from Analysis Services”.

Security in DirectQuery enabled models before SQL Server 2016

Securing a DirectQuery enabled model is fundamentally different from securing an in-memory model. Many of the security designs discussed earlier in this whitepaper do not apply to DirectQuery enabled models because DirectQuery enabled models do not support row security or dynamic security. Also, because DirectQuery enabled models connect to the data source in two scenarios – when querying the model and when processing the model – the impersonation requirements change. The DirectQuery engine can impersonate the current user when querying the data source, thus allowing the SQL Server security model to be used and therefore changing the way that the data warehouse is secured.

An in-depth discussion of security in DirectQuery enabled models is out of the scope of this document. For an in-depth discussion of security considerations in DirectQuery enabled models, see the DirectQuery in the BI Semantic Model whitepaper (http://msdn.microsoft.com/en-us/library/hh965743.aspx).

Security in DirectQuery enabled models post SQL Server 2016

SQL Server 2016 adds support for row security or dynamic security for DirectQuery enabled models. Regardless if the model is in DirectQuery mode or not using bi-directional cross filter to secure your models will work as described in the chapter “Enabling dynamic security using Tabular models using bi-directional relationships.”

DirectQuery models have one additional benefit for security, you can pass the user who is connecting to Analysis Services on to the underlying database, thus leveraging any security placed on the data source directly. To do this we can use an option available in Analysis Services that allows us to connect to a data source using the credentials of the current user as is described here: https://docs.microsoft.com/en-us/sql/analysis-services/multidimensional-models/set-impersonation-options-ssas-multidimensional#bkmk_options

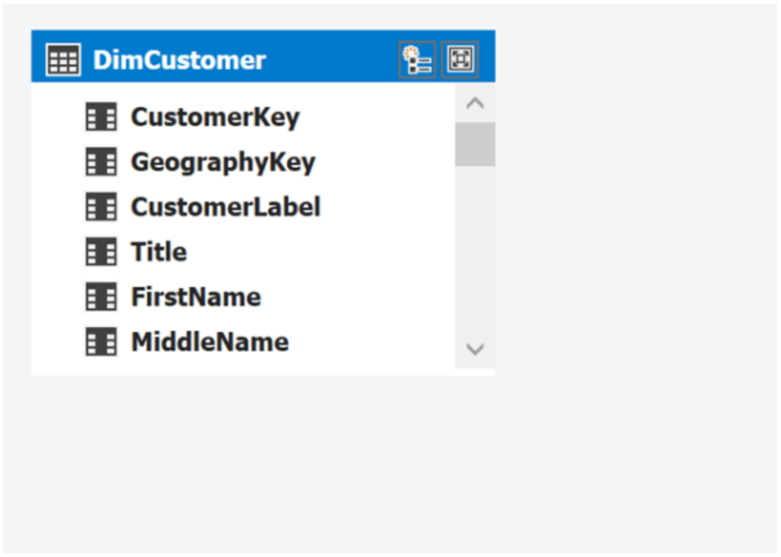
For more information on Direct Query the whitepaper “Direct Query in SQL Server 2016 Analysis Services” is available.

Example: Leveraging SQL Server Row Level Security

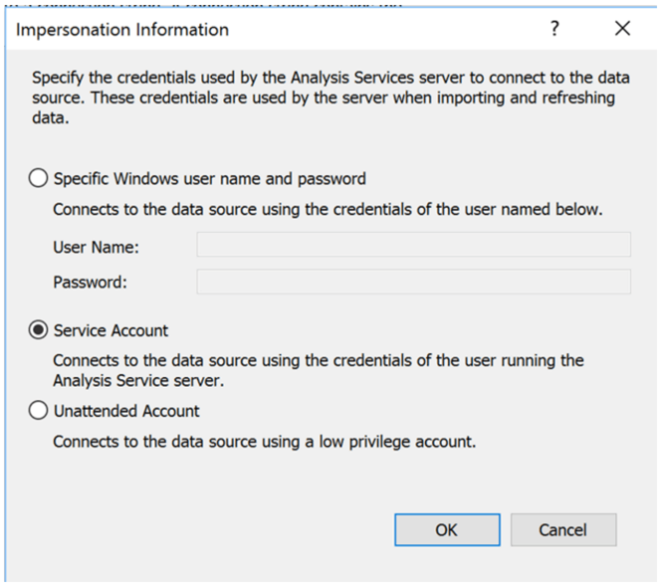
Let’s look at an example where we pass on the username from SSAS to the underlying database. In this case, you could have something like the following architecture: Dashboards and reports in Power BI that connect to an on premise SSAS server running DirectQuery mode connecting to SQL Server. All with the same username flowing from Power BI all the way down to SQL Server.

Let’s see how this works in SSAS. The Power BI and gateway part work as usual - no changes are needed there. The example below is using Tabular DirectQuery, but the same works for Multidimensional ROLAP.

I have this simple model with just 1 table set to DirectQuery mode:



I am using the service account to connect to the data source (go to Existing connections, Edit Datasouce, Impersonations to get this dialog)


Impersonation Information

Specify the credentials used by the Analysis Services server to connect to the data source. These credentials are used by the server when importing and refreshing data.

○ Specific Windows user name and password

Connects to the data source using the credentials of the user named below.

User Name:

Password:

● Service Account

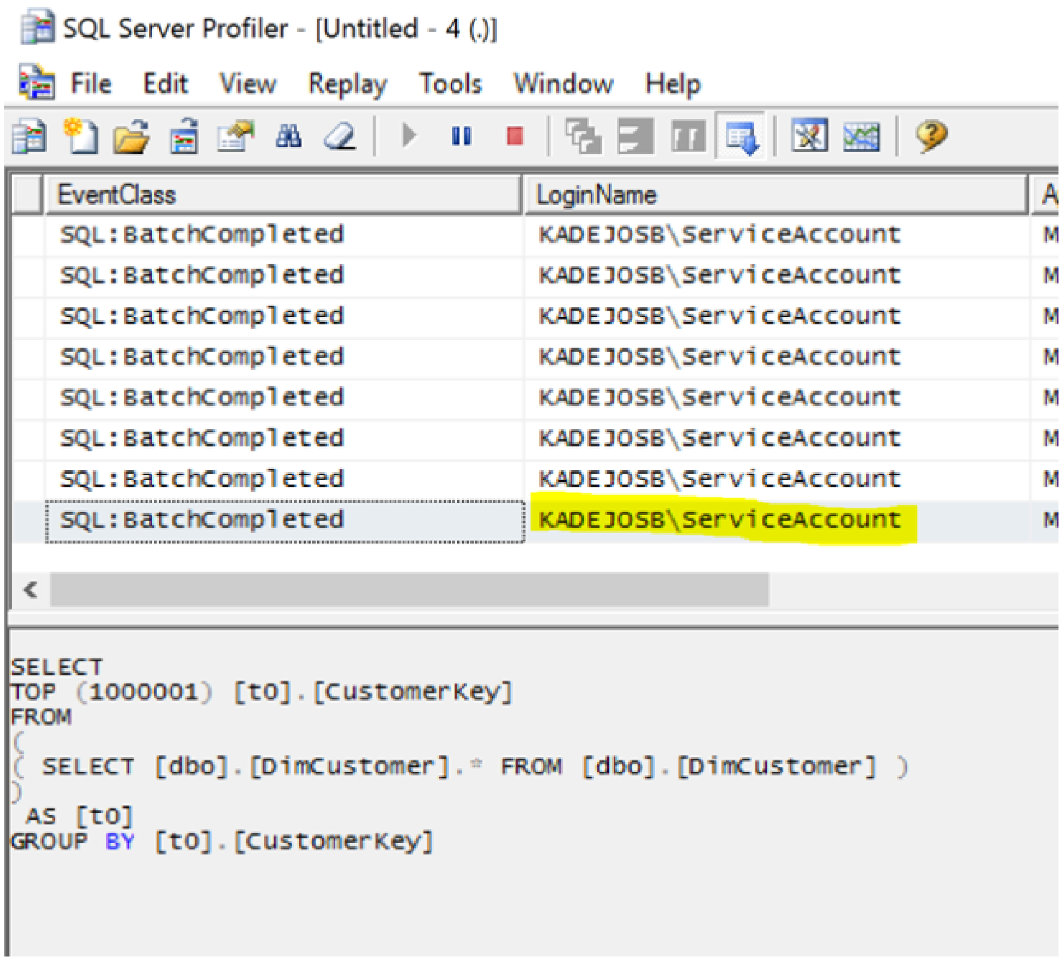
Connects to the data source using the credentials of the user running the Analysis Service server.

○ Unattended Account

Connects to the data source using a low privilege account.

Note: Best practice is to not use the service account as impersonation setting in any production environment

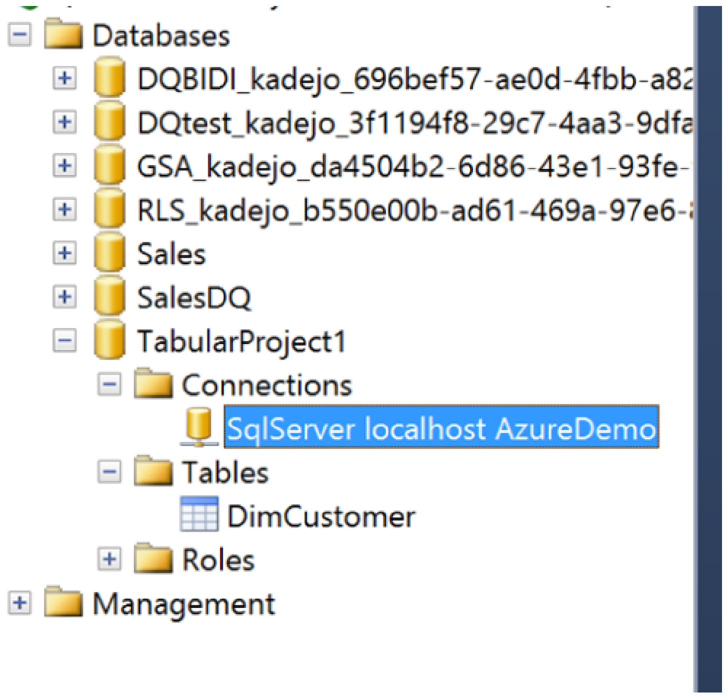
Now, if I connect to this model with Excel and run profiler on SQL Server we can see the user that is running the queries to SQL Server and logging into SQL using the service account SSAS is running under:


SQL Server Profiler - [Untitled - 4 (.)]
File Edit View Replay Tools Window Help

EventClass	LoginName	A
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M
SQL:BatchCompleted	KADEJOSB\ServiceAccount	M

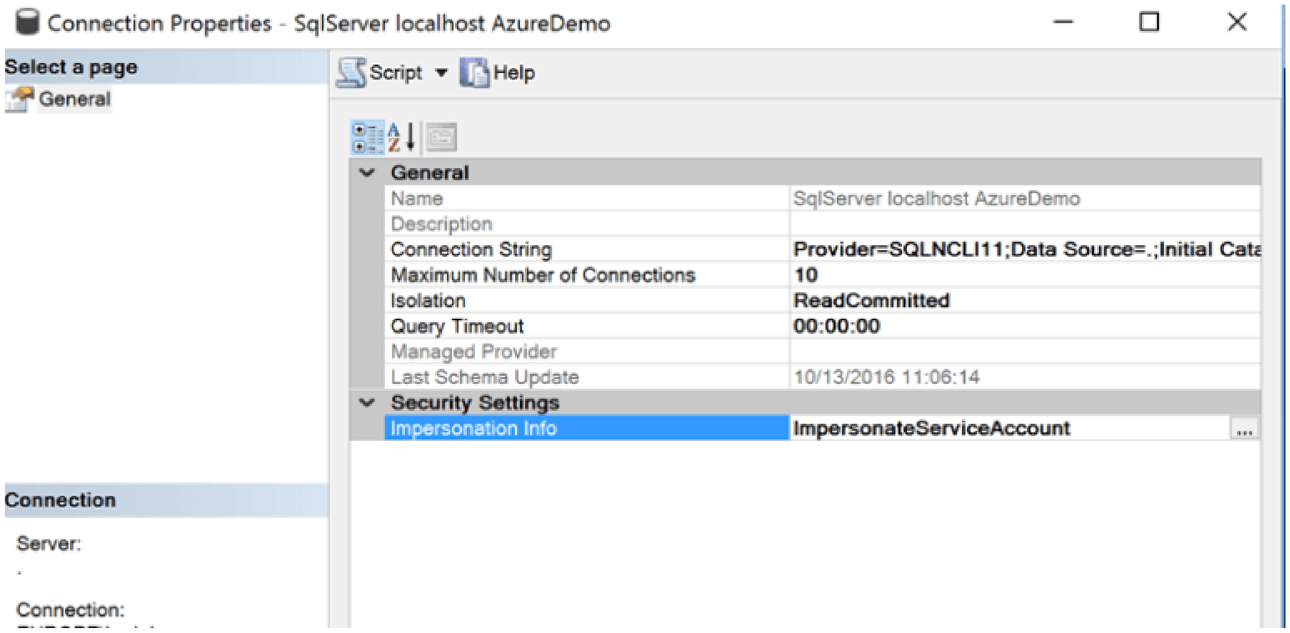
```
SELECT
TOP (1000001) [t0].[CustomerKey]
FROM
(
( SELECT [dbo].[DimCustomer].* FROM [dbo].[DimCustomer] )
)
 AS [t0]
GROUP BY [t0].[CustomerKey]
```

Now instead of the Service Account I want to use the credentials of the user who is connecting to my model and use that to connect to the SQL database. You cannot do this in the SSDT UI. Instead I publish the model to my server and open the Connection Dialog in SSMS:



Databases
DQBIDI_kadejo_696bef57-ae0d-4fbb-a82
DQtest_kadejo_3f1194f8-29c7-4aa3-9dfa
GSA_kadejo_da4504b2-6d86-43e1-93fe-
RLS_kadejo_b550e00b-ad61-469a-97e6-
Sales
SalesDQ
TabularProject1
Connections
SqlServer localhost AzureDemo
Tables
DimCustomer
Roles
Management

As you can see this is still set to impersonate the service account:



Connection Properties - SqlServer localhost AzureDemo

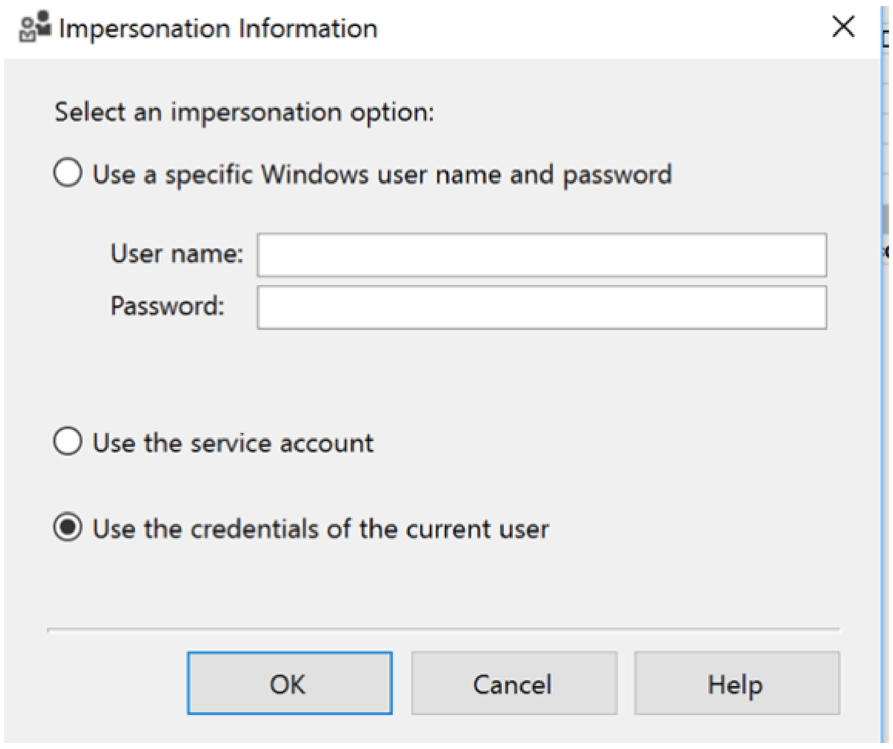
General

Name	SqlServer localhost AzureDemo
Description	
Connection String	Provider=SQLNCLI11;Data Source=.;Initial Cata
Maximum Number of Connections	10
Isolation	ReadCommitted
Query Timeout	00:00:00
Managed Provider	
Last Schema Update	10/13/2016 11:06:14

Security Settings

Impersonation Info	ImpersonateServiceAccount

Clicking here will give me a new option that I didn't have in SSDT, one that allows me to use the credentials of the current user:


Impersonation Information

Select an impersonation option:

○ Use a specific Windows user name and password

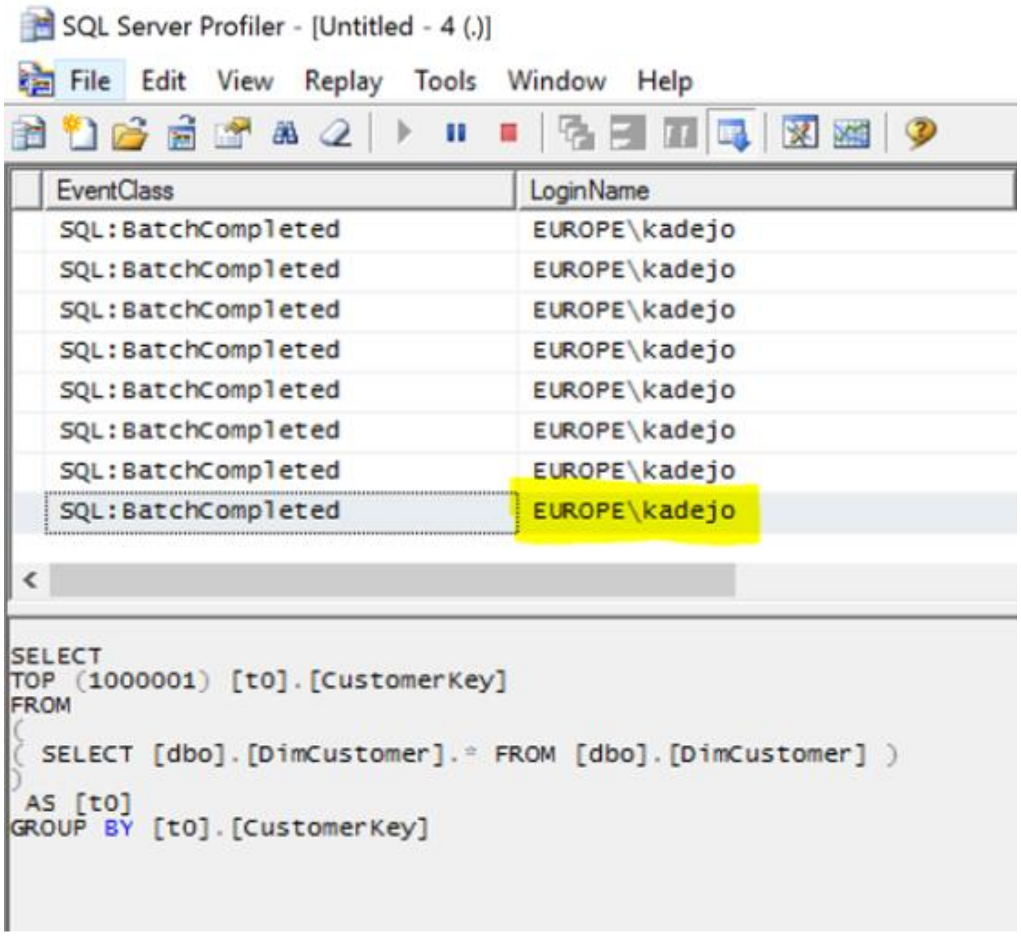
User name:

Password:

○ Use the service account

● Use the credentials of the current user

I press OK and connect again with Excel to my model. Now I will be able to see that my username is being used to connect to SQL Server:


SQL Server Profiler - [Untitled - 4 (.)]
File Edit View Replay Tools Window Help

EventClass	LoginName
SQL:BatchCompleted	EUROPE\kadejo
SQL:BatchCompleted	EUROPE\kadejo
SQL:BatchCompleted	EUROPE\kadejo
SQL:BatchCompleted	EUROPE\kadejo
SQL:BatchCompleted	EUROPE\kadejo
SQL:BatchCompleted	EUROPE\kadejo
SQL:BatchCompleted	EUROPE\kadejo
SQL:BatchCompleted	EUROPE\kadejo

```
SELECT
TOP (1000001) [t0].[CustomerKey]
FROM
(
( SELECT [dbo].[DimCustomer].* FROM [dbo].[DimCustomer] )
)
 AS [t0]
GROUP BY [t0].[CustomerKey]
```

And that's it. Now SSAS passes on the credentials of the current user. If you have RLS configured at your SQL Server SQL database, it will now only return the rows approved for this user. This option works is available for both Multidimensional and Tabular models.

Of course, passing along credentials between machines can also complicate matters. In my scenario things are easy as I have all services on a single box. But in a real-world scenario, SSAS and the SQL database system will not be on the same machine. In those cases, you will need to configure Kerberos delegation between the two machines. Below you can find multiple blog post of SSAS and similar services that should give you a starting point on how to configure this:

https://msdn.microsoft.com/en-us/library/dn194199.aspx

http://sqlmag.com/sql-server-reporting-services/implement-kerberos-delegation-ssrs

https://shuggill.wordpress.com/2015/03/06/configuring-sql-server-kerberos-for-double-hop-authentication/

Conclusion

Securing a tabular BI semantic model is conceptually simple. Windows users or groups are added to roles, and a set of security permissions are defined on a per-role basis. These permissions determine if the user can administer, process, or read the model. Read permission can be further refined by adding row filters to tables in the model. These filters can be static or dynamic. In practice, there are many factors to take in to consideration when implementing and deploying a tabular model with security defined. You must plan to address all of these factors when developing your tabular models.

This paper gave an overview of the tabular security model, explained how to implement static and dynamic security in several scenarios, described how to manage roles and permissions, and gave suggestions for how to manage connections to Analysis Services and to relational data sources.

Additional resources

Roles (SSAS Tabular) https://docs.microsoft.com/sql/analysis-services/tabular-models/roles-ssas-tabular

Impersonation (SSAS Tabular) https://docs.microsoft.com/sql/analysis-services/tabular-models/impersonation-ssas-tabular

Kerberos by CSS SQL Server Engineers http://blogs.msdn.com/b/psssql/archive/tags/kerberos/

Deployment checklist: Reporting Services, Power View, and PowerPivot for SharePoint http://msdn.microsoft.com/en-us/library/hh231687(v=sql.110).aspx

DirectQuery in SQL Server Analysis Services 2016 https://blogs.msdn.microsoft.com/analysisservices/2017/04/06/directquery-in-sql-server-2016-analysis-services-whitepaper/

Programming AMO Security Objects https://docs.microsoft.com/sql/analysis-services/multidimensional-models/analysis-management-objects/programming-amo-security-objects

QuickStart: Learn DAX Basics in 30 Minutes https://support.office.com/article/QuickStart-Learn-DAX-Basics-in-30-Minutes-51744643-C2A5-436A-BDF6-C895762BEC1A

For more information:

http://www.microsoft.com/sqlserver/: SQL Server Web site

https://azure.microsoft.com/services/analysis-services/: Azure Analysis Services

http://technet.microsoft.com/en-us/sqlserver/: SQL Server TechCenter http://msdn.microsoft.com/en-us/sqlserver/: SQL Server DevCenter