Dynamic Security in Analysis Services

Why use dynamic security?

Administering security in Microsoft Analysis Services can be especially challenging when large numbers of users are restricted to different parts of your cube. With SP2, Analysis Manager can suffer performance slowdowns when managing more than 150-200 roles.

While Analysis Manager performance has improved in SP3, managing large numbers of roles is still difficult, especially because of the large number of items displayed in the tree view.

Using dynamic security, you can avoid these problems. In this approach you will:

- Store your security permissions in relational tables, which become the source for an Analysis Services cube.
- Define only one role on your main cube.
- When a user logs on, query the 'permissions cube' to find that user's access rights in the multi-dimensional data space.
- Apply that security filtering to the main cube.
- Allow the user to access the main cube

Dynamic security is only possible with Analysis Services SP1 or higher and can be used with either dimension or cell security settings.

Note: *Dynamic security does not offer any advantages in terms of memory usage over non-dynamic, standard roles*.

How to set up dynamic security

Preliminary Step: Restore the Demonstration Databases

The zip file associated with this document contains two .CAB files and a Microsoft Access database. These resources are required for the working demonstrations of dynamic dimension security. One example will demonstrate security at the leaf level (the lowest level of detail in the data), and another will feature security at all levels of a dimension.

To install these databases on your server, follow these steps:

1. Extract the contents of the zip file to a directory on your server
2. Open the Access database, and edit the data in the Users table
    - Change the UserName field, in the single row in that table, to the USERID that you will use to run Analysis Manager on your server
3. Open Analysis Manager and expand the node on the "Servers" tree that represents your server

4. Right-click on this node and select “Restore Database” from the pop-up menu.
    - Then select the .CAB file you wish to restore, and click “Open”, then “Restore”
5. Expand the new database in the tree view, and then expand the “Data Sources” folder
6. Right-click on the data source; a dialog box will appear. Select “Edit”
    - Select the “Connection” tab, then edit the database filename so that it contains the path of the Access database you have just extracted.

Remember that when you browse data in the Cube Editor as an OLAP Administrator, you bypass any security filtering and see all the data in the cube. To see the effect of security restrictions, right-click on the “SalesAndPermissions” virtual cube and select “Manage Roles”, then select “Test Role”.

Example 1: Dynamic Security at the Leaf Level

This section describes the steps required to create the first demonstration database, ‘Dynamic_Security_Demo’. This database demonstrates how to set-up simple dynamic security at the leaf level (or lowest level) of a dimension.

The objective is as follows:

- In the ‘Sales’ cube, there is one measure called ‘Sales’, and two dimensions: ‘Geography’ and ‘Product’.
- You must limit user access to specified members on the Geography dimension.

STEP 1: Create the relational USERID table

In your relational database, create a new table named ‘Users’ with two columns:

- A key column, call it “UserKey”
- A text column containing the USERIDs of your individual users
    - User the format ‘DomainName\UserName’.

It’s also a good idea to add USERIDs of any OLAP Administrators doing development work. Otherwise, if they try to use the ‘Test Role’ functionality in Analysis Manager (see above), they will have no permissions and will be unable to browse the data.

STEP 2: Create the Permissions table

Create a second relational table named 'Permissions'. This will be used to create the Permissions Cube which stores the security permissions in multidimensional space. This table should have three columns:

- The UserKey of a user
- The key of a leaf-level (lowest level) member on the Geography dimension that this user is allowed to access.
- A field which always contains the value "1". This will be the single measure in the cube.

Each row in the table represents permission for one user to see one member on the Geography dimension.

If the user is allowed access to more than one leaf-level member, add a row for each

Note: If you want to implement dynamic security on multiple dimensions in a cube, you should create separate permissions tables for each dimension.

STEP 3: Build the Permissions Cube

Once you have created the relational tables, build a multidimensional cube to store the permissions in Analysis Services.

- Use the Permissions table as a fact table with the column containing the '1' value as the single measure called 'Permission'.
- Create a new dimension from the Users table
- Add the Geography dimension from that table,

You will create a cube which returns a non-empty value for each dimension member that a specified user is allowed to access.

Note: The column containing the '1' is only included in the Permissions table because the Cube Wizard in Analysis Manager requires that you have at least one regular measure in a cube. If you created the cube in the Cube Editor, you could instead create a 'computed' measure with a Source Column property of '1'. You would not then need the column in the Permissions table.

STEP 4: Apply the Permissions Cube the data cube

Join this Permissions cube to your main data cube as a virtual cube, so that your real data and the permissions data is available in the same place. Remember to deny all users all access to these two cubes afterwards. You can also hide the Permissions measure and the Users dimension by setting their Visible property to False.

Finally, create a role on the new, virtual cube. In the **Membership** dialog add all the names of the users to it.

Then, in the Restricted Dimensions dialog select **Geography**, set **Rule** to 'Custom' in the dropdown box and then click on Custom Settings.

In the **Advanced** tab of the **Custom Dimension Security** dialog which then appears, you should paste the following MDX:

```
STRTOSET(
IIF(USERNAME="",
"{}",
SETTOSTR(
NONEMPTYCROSSJOIN(
[Geography].[Geography].MEMBERS,
{STRTOMEMBER("[USERS].[All USERS].[" + USERNAME + "]")},1)
)))
```

This expression takes the set of all members, on the leaf level of the Geography dimension, and uses NONEMPTYCROSSJOIN to return only those members where a row exists in the Permissions table for the current user.

Note: A bug in the USERNAME function, in certain circumstances, may cause it to return an empty string instead of a valid USERID (see RAID database Plato7x bug #12653). There are two possible workarounds available, such as:

- Add a member to your Users dimension with a blank string as a name and grant access to the "All" member,
- Trap this possibility in the MDX itself as shown above: if USERNAME does return an empty string, the expression simply returns a set containing just the All Member.

Note: In SP2 there were a several bugs relating to security – especially dynamic cell security – and the creation of local cubes, some of which were fixed in SP3 and some of which weren't (see, for example, RAID database Plato7x bug #12740). If your requirements include both dynamic security and the creation of local cubes, then test your application very thoroughly to ensure that these bugs are not an issue.

Example 2: Dynamic Security above the Leaf Level

You may also need to grant users access to dimension members above the leaf or bottom level. You could achieve this by granting permission to all the descendants of that member down to the leaf level, but this could be impractical. The second

demonstration database, 'Dynamic_Security_Demo_MultiLevel' shows how you can implement this requirement on a Geography dimension that contains two levels.

STEP 1: Create a "flattened" Geography Dimension Table

Create another table named 'FlatGeographyMultiLevel' which contains a 'flattened' version of the Geography dimension, i.e. with a single column containing all members of all levels on the dimension.

Make sure that the dimension has keys which are unique across all levels, so that the same keys can be used in the single level of the flattened dimension

- We will be using the LINKMEMBER function to match members on the flattened dimension to the equivalent members on the real dimension.

Using this flattened table, you can store access rights for all members on this dimension in your Permissions table, because all members have been forced to the flattened leaf level.

STEP2: Build the Permissions Cube with the Flatten Dimension

Build your Permissions cube as described in the previous section, but instead of adding the real Geography dimension, create a new dimension called Flat_Geography from the flattened dimension table.

STEP3: Create a virtual cube combining the Permissions and data cubes

Create your virtual cube from the Sales cube and the Permissions cube. Rremember to hide the Flat_Geography dimension as well as the Users dimension and the Permissions measure.

In this example, the MDX used defining the role is slightly more complicated:

```
STRTOSET(
IIF(USERNAME="",
"{}",
SETTOSTR(
GENERATE(
NONEMPTYCROSSJOIN(
[Flat_Geography].[Geography].MEMBERS,
{STRTOMEMBER("[USERS].[All USERS].[" + USERNAME + "]")},1)
,{LINKMEMBER([Flat_Geography].CURRENTMEMBER, [Geography])}
))))
```

This expression is similar to the one in the previous section, but this time the NONEMPTYCROSSJOIN is directed to the flattened version of the Geography dimension to return the set of members to which the current user is permitted to access. The LINKMEMBER and GENERATE functions convert this set of members on Flat_Geography to the equivalent members on Geography, which may be above the leaf level.

Example 3: Controlling Upper and Lower Levels Visibility

From SP3 onwards, you can write dynamic MDX expressions to control the upper and lower levels in the dimension available to the user. These expressions are entered in the **Custom Dimension Security** dialog on the Advanced tab, just above where the MDX for the two previous sections was entered.

For example, in the 'Dynamic_Security_Demo_MultiLevel' database, you might restrict visibility of Geography level that user cannot access.

- The Geography dimension has two levels, Country and City
- If a user is only allowed access to Geography dimension members at the Country level, they should not be able to drill down to the City level
- This is be achieved by using the following MDX as the 'Bottom Level' expression in the **Custom Dimension Security** Dialog:

```
GEOGRAPHY.LEVELS(
MAX(
STRTOSET(
IIF(USERNAME="",
"{}",
SETTOSTR(
GENERATE(
NONEMPTYCROSSJOIN(
[Flat_Geography].[Geography].MEMBERS,
{STRTOMEMBER("[USERS].[All USERS].[" + USERNAME + "]")},1)
,{LINKMEMBER([Flat_Geography].CURRENTMEMBER, [Geography])}
))))
, GEOGRAPHY.CURRENTMEMBER.LEVEL.ORDINAL))
```

This is the same MDX expression used to return the set of members the user is permitted to access, wrapped in a MAX function to find the maximum level ordinal in the set, and a LEVELS function to return the level of the deepest member.