Smartlist Builder & Cross DB Queries

Problem:

When using Smartlist Builder with a custom SQL view that is a "cross database query" or any table from a custom database a GP user that is setup in SQL as a "sysadmin" will have no trouble seeing data in the created Smartlist assuming appropriate Smartlist Builder security has been granted for the user inside GP (see pg. 76 of the April 2008 edition *Smartlist Builder User Guide with Excel Report Builder*).

However, if you are in GP as a non-admin user, there is a good chance that you would not be able to see data on Smartlists created in Smartlist Builder that utilize these views or tables even if all GP security is set properly for your user. Furthermore, it is possible that you won't be able to see the new Smartlists at all.

```
USE TWO
GO

IF object_id(N'SampleView', 'V') IS NOT NULL
    DROP VIEW SampleView
GO

CREATE VIEW SampleView
AS  SELECT  *
    FROM    TWO..SOP10100 T1
            INNER JOIN myDatabase..WebOrders T2 ON T1.SOPNUMBE = T2.OrderNum
```

Figure 1.0 – Sample Cross Database Query to create a SQL View.

Corey Clay, MCS
Technical Consultant
Nextec Group, Inc.
Cleveland, Ohio
http://www.nextecgroup.com

Resolutions:

There are 2 possible options for resolving the issue with having the ability to see the data on these new Smartlists. However, you must first confirm that the "DYNGRP" database role is granted "SELECT" permission on the view or table used in your new Smartlist (See Figure 1.1 for an example). This will enable the user to at least see the new Smartlist even it won't populate the data.

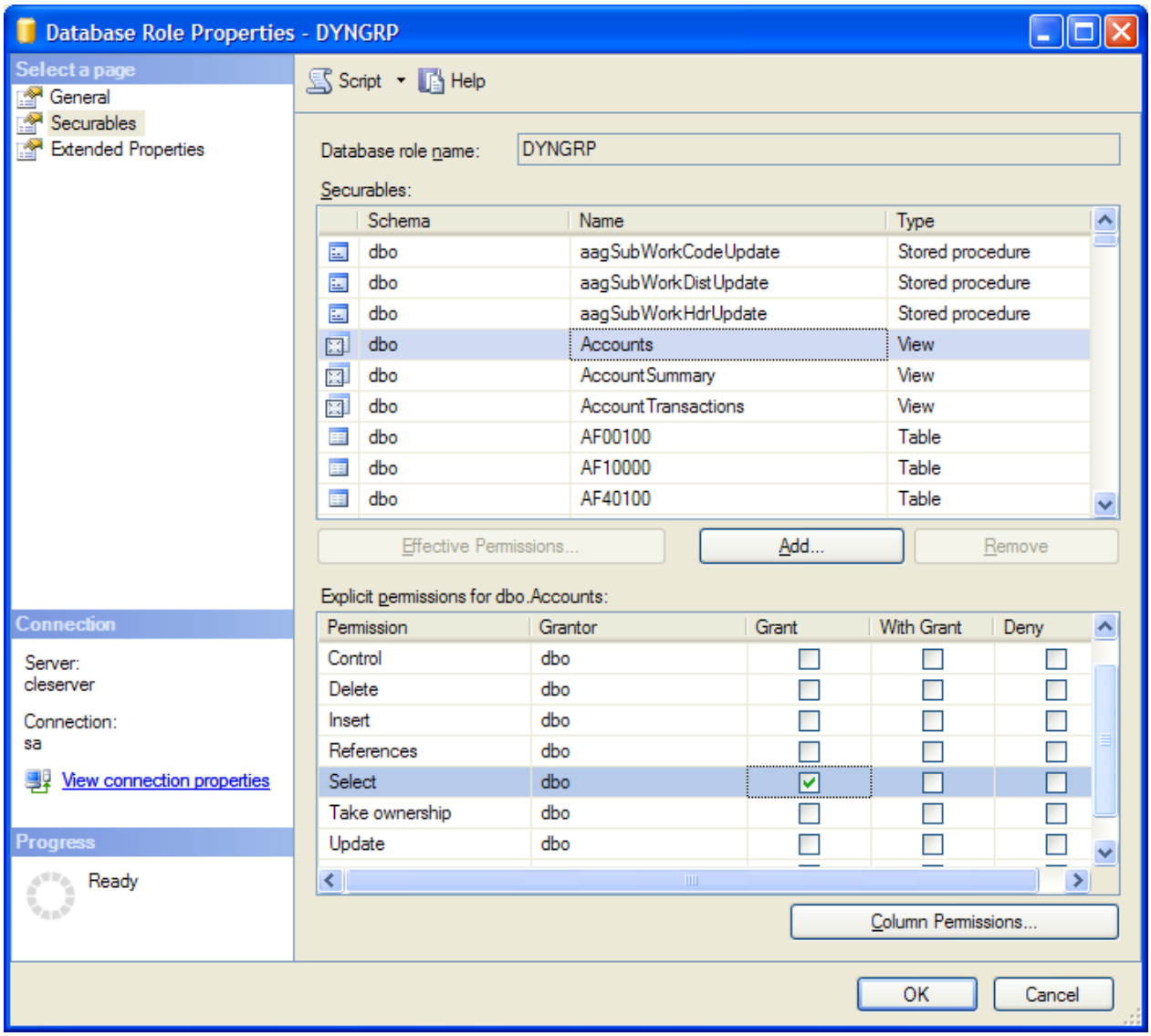


Figure 1.1 – Database Role Properties Window -> "Securables" Tab

Option 1: In order for the data to appear in the new Smartlist, the user that is trying to run the Smartlist must exist as a SQL user in all of the databases that are being queried by the new Smartlist. This option could become very cumbersome if there are a lot of users that would need to have the ability to run the Smartlist because each user would have to be added to the "foreign database". In the case of my example script in above (Figure 1.0) the user would have to exist in both the TWO database (which we would know they do) as well as the database called "myDatabase".

Option 2: SQL Server has a setting that is disabled by default, but can be manually enabled called *Cross Database Ownership Chaining*. This feature can be enabled either by script or via the server properties window as shown below in figure 1.2. The feature is described in depth in Appendix A & Appendix B which are both taken from the Online Help for Microsoft SQL 2005. Ownership Chaining will allow the query to run as long as the TWO database and "myDatabase" have the same "db_owner". When the view tries to reference a table in the foreign database "myDatabase", ownership chaining will kick in and check to see that the calling database and the target database have the same owner, if they do, there will be no further security checking. ***Please note that, when enabling this option, the SQL server must be restarted for the change to take effect.***

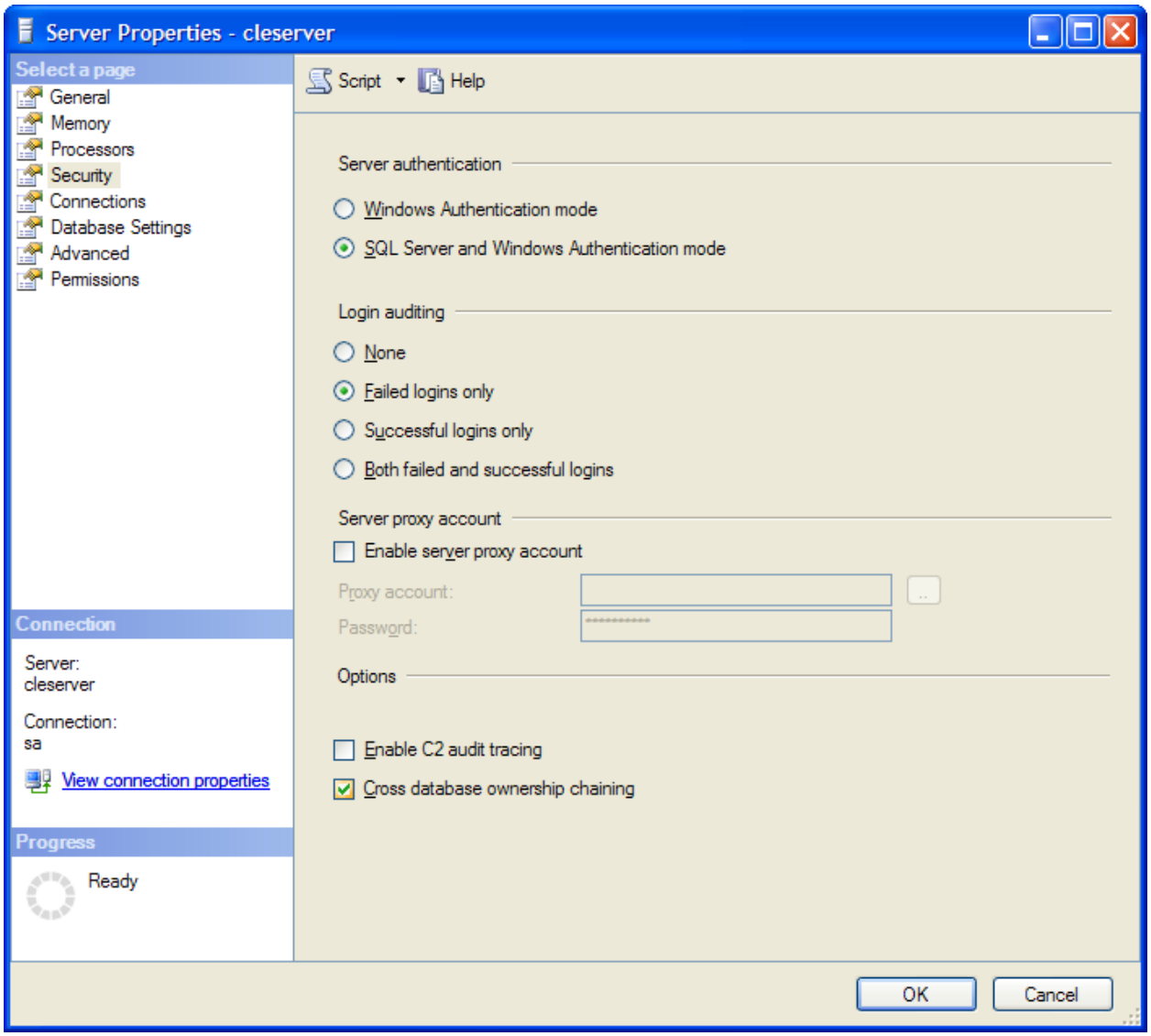


Figure 1.2 – Server Properties Window -> Security Tab -> Options -> Cross database ownership chaining

SQL Server 2005 Books Online (September 2007)

Ownership Chains

Security Considerations for Databases and Database Applications >

When multiple database objects access each other sequentially, the sequence is known as a *chain*. Although such chains do not independently exist, when SQL Server 2005 traverses the links in a chain, SQL Server evaluates permissions on the constituent objects differently than it would if it were accessing the objects separately. These differences have important implications for managing security.

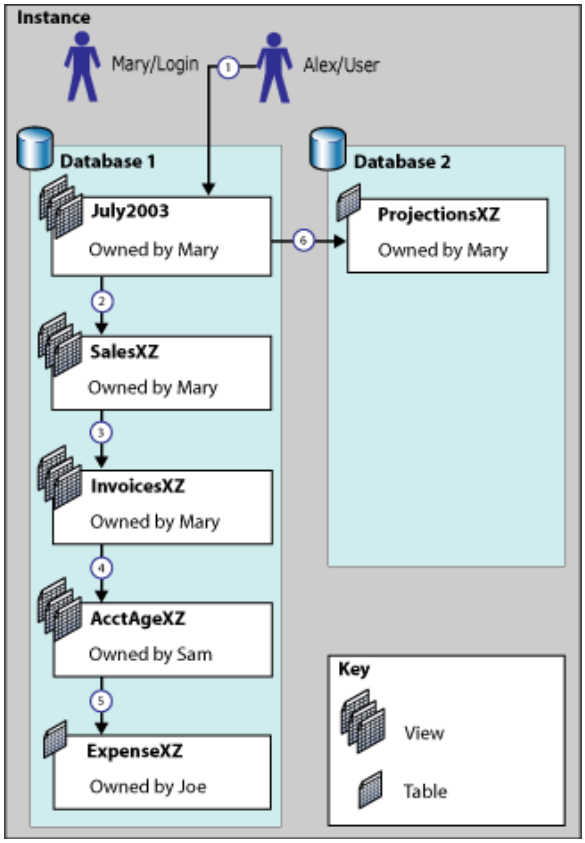
Ownership chaining enables managing access to multiple objects, such as multiple tables, by setting permissions on one object, such as a view. Ownership chaining also offers a slight performance advantage in scenarios that allow for skipping permission checks.

How Permissions Are Checked in a Chain

When an object is accessed through a chain, SQL Server first compares the owner of the object to the owner of the calling object. This is the previous link in the chain. If both objects have the same owner, permissions on the referenced object are not evaluated.

Example of Ownership Chaining

In the following illustration, the **July2003** view is owned by Mary. She has granted to Alex permissions on the view. He has no other permissions on database objects in this instance. What happens when Alex selects the view?



1. Alex executes SELECT * on the **July2003** view. SQL Server checks permissions on the view and confirms that Alex has permission to select on it.
2. The **July 2003** view requires information from the **SalesXZ** view. SQL Server checks the ownership of the **SalesXZ** view. Because this view has the same owner (**Mary**) as the view that calls it, permissions on **SalesXZ** are not checked. The required information is returned.

3. The **SalesXZ** view requires information from the **InvoicesXZ** view. SQL Server checks the ownership of the **InvoicesXZ** view. Because this view has the same owner as the previous object, permissions on **InvoicesXZ** are not checked. The required information is returned. To this point, all items in the sequence have had one owner (**Mary**). This is known as an *unbroken ownership chain*.

4. The **InvoicesXZ** view requires information from the **AcctAgeXZ** view. SQL Server checks the ownership of the **AcctAgeXZ** view. Because the owner of this view is different from the owner of the previous object (**Sam**, not **Mary**), full information about permissions on this view is retrieved. If the **AcctAgeXZ** view has permissions that allow access by **Alex**, information will be returned.

5. The **AcctAgeXZ** view requires information from the **ExpenseXZ** table. SQL Server checks the ownership of the **ExpenseXZ** table. Because the owner of this table is different from the owner of the previous object (**Joe**, not **Sam**), full information about permissions on this table is retrieved. If the **ExpenseXZ** table has permissions that allow access by **Alex**, information is returned.

6. When the **July2003** view tries to retrieve information from the **ProjectionsXZ** table, the server first checks to see whether cross-database chaining is enabled between **Database 1** and **Database 2**. If cross-database chaining is enabled, the server will check the ownership of the **ProjectionsXZ** table. Because this table has the same owner as the calling view (**Mary**), permissions on this table are not checked. The requested information is returned.

Cross-Database Ownership Chaining

SQL Server can be configured to allow ownership chaining between specific databases or across all databases inside a single instance of SQL Server. Cross-database ownership chaining is disabled by default, and should not be enabled unless it is specifically required.

Potential Threats

Ownership chaining is very useful in managing permissions on a database, but it does assume that object owners anticipate the full consequences of every decision to grant permission on a securable. In the previous illustration, Mary owns most of the underlying objects of the **July2003** view. Because Mary has the right to make objects that she owns accessible to any other user, SQL Server behaves as though whenever Mary grants access to the first view in a chain she has made a conscious decision to share the views and table it references. In real life, this might not be a valid assumption. Production databases are far more complex than the one in the illustration, and the permissions that regulate access to them rarely map perfectly to the administrative structures of the organizations that use them.

You should understand that members of highly privileged database roles can use cross-database ownership chaining to access objects in databases external to their own. For example, if cross-database ownership chaining is enabled between database **A** and database **B**, a member of the **db_owner** fixed database role of either database can spoof her way into the other database. The process is simple: Diane (a member of **db_owner** in database **A**) creates user **Stuart** in database **A**. **Stuart** already exists as a user in database **B**. Diane then creates an object (owned by **Stuart**) in database **A** that calls any object owned by **Stuart** in database **B**. Because the calling and called objects have a common owner, permissions on the object in database **B** will not be checked when Diane accesses it through the object she has created.

SQL Server 2005 Books Online (September 2007)

cross db ownership chaining Option

Administering the Database Engine > Managing Servers > Setting Server Configuration Options >

Use the **cross db ownership chaining** option to configure cross-database ownership chaining for an instance of Microsoft SQL Server.

This server option allows you to control cross-database ownership chaining at the database level or to allow cross-database ownership chaining for all databases:

- When **cross db ownership chaining** is off (0) for the instance, cross-database ownership chaining is disabled for all databases.

- When **cross db ownership chaining** is on (1) for the instance, cross-database ownership chaining is on for all databases.

- You can set cross-database ownership chaining for individual databases using the SET clause of the ALTER DATABASE statement. If you are creating a new database, you can set the cross-database ownership chaining option for the new database using the CREATE DATABASE statement.

  Setting **cross db ownership chaining** to 1 is not recommended unless all of the databases hosted by the instance of SQL Server must participate in cross-database ownership chaining and you are aware of the security implications of this setting. For more information, see Ownership Chains.

Controlling Cross-Database Ownership Chaining

Before turning cross-database ownership chaining on or off, consider the following:

- You must be a member of the **sysadmin** fixed server role to turn cross-database ownership chaining on or off.

- Before turning off cross-database ownership chaining on a production server, fully test all applications, including third-party applications, to ensure that the changes do not affect application functionality.

- You can change the **cross db ownership chaining** option while the server is running if you specify RECONFIGURE with **sp_configure**.

- If you have databases that require cross-database ownership chaining, the recommended practice is to turn off the **cross db ownership chaining** option for the instance using **sp_configure**; then turn on cross-database ownership chaining for individual databases that require it using the ALTER DATABASE statement.