Protect the Metadata

Sourc e	http://www.microsoft.com/technet/technetmag/issues/2006/01/ProtectMetaData/default.aspx

Metadata is data about data. It not only describes the structure and meaning of your data, but also the structure and meaning of your applications and processes. In this article, I will discuss the concept of metadata visibility and focus on the visibility restrictions introduced in SQL Server™ 2005. I'll look at the tools available for accessing the metadata, how the new restrictions work, and to whom they apply. Finally, I'll look at the impact that some of the new restrictions can have when upgrading to SQL Server 2005 and suggest best practices for migrating applications affected by these changes.

Legacy Access to Metadata

In earlier versions of SQL Server (prior to SQL Server 2005), all data, including metadata, was available through relational tables called system tables. Every database in a SQL Server instance had its own set of system tables and, in addition, there were system tables that existed only in the master database. System tables have several properties by which they can be recognized: names that start with "sys", xtype values in the sysobjects table of "S", and Object ID values less than 100.

Prior to SQL Server 2005, most of the data in the system tables was visible to any user, except for data in a few columns with limited visibility, such as password columns. However, even though users could directly select data from the system tables, it was recommended that other tools be used for the purpose of looking at the metadata.

Luckily, SQL Server 7.0 and SQL Server 2000 have several such tools, including system stored procedures (sp_help), property functions (OBJECTPROPERTY), system functions (OBJECT_ID), and information schema views (INFORMATION_SCHEMA.TABLES).
Some of these interfaces do restrict the results so that not all users can see all the available metadata. The stored procedure sp_helpdb, for example, returns one row for each database in the instance to which the current user has access. An administrator can see all the databases, but all other users can only see rows for databases to which they have been allowed access. There is nothing magic in how SQL Server determines which databases to display when sp_helpdb is executed. The procedure itself includes a test for database access using the system function has_dbaccess. If an administrator does not want this restriction, he can actually rewrite the sp_help procedure to remove the test and unconditionally return

information for all databases. However, in SQL Server 7.0 and SQL Server 2000 any user can still see information about all the databases even if the administrator hasn't given the user approval, because the sysdatabases system table itself is unprotected. In those versions, any user can execute the following, and see one row for each database:

```
SELECT * FROM sysdatabases
```

The information schema views were introduced in SQL Server 7.0 to provide an ANSI-mandated interface for accessing metadata. The objects are defined as a set of views on top of the system tables, and all the views restrict the data returned to include only the data to which the current user has rights. So basically, you can think of them as working on a need-to-know basis. For example, a user who executes a select statement will see one row for each table or view for which he has permissions, as illustrated in the following:

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

However, just as I previously mentioned for running the sp_help procedure, this isn't really a serious restriction because the metadata is always available using the system table directly.

Users accessing SQL Server using the provided GUI tools are also subject to inconsistent restrictions on metadata visibility in these previous versions. When using Query Analyzer, only those databases to which a user has access will be displayed in the Object Browser and in the database dropdown list, but using Enterprise Manager, all databases will be shown in the object tree. In both tools, all objects are always visible for any database once the user accesses that database, whether or not the user has any rights to the objects.

SQL Server 2005 Security

In SQL Server 2005, there are extensive changes to the entire security model. The biggest change that will most likely impact your applications is the separation between users and schemas. In earlier versions, object names were fully qualified by including the name of the user who owned the object, while ANSI standard SQL specified that an object qualification should include the "schema" in which an object was contained. Until the release of SQL Server 2005, SQL Server really didn't have a concept of schemas. To prepare to upgrade, you should get in the habit right now of starting to qualify all objects with their user name so that qualifying by schema name will be easy when you need to do it in SQL Server 2005.

The new security model is based on a hierarchy of entities known as "securables". Your servers contain databases, which contain schemas, which contain even more finite

securables, such as tables and views. (See Figure 1 for a diagram of the new security hierarchy.) Securables can contain data, or they can contain executable code. Permissions can be granted at any level, on any securable, or on all securables at a particular level. So you could GRANT SELECT on a table, or on all tables in a schema, or on all schemas in a database. In addition, there are enhancements which will make it easier to define your own security framework by granting or denying more specific rights than are possible in SQL Server 7.0 or SQL Server 2000.

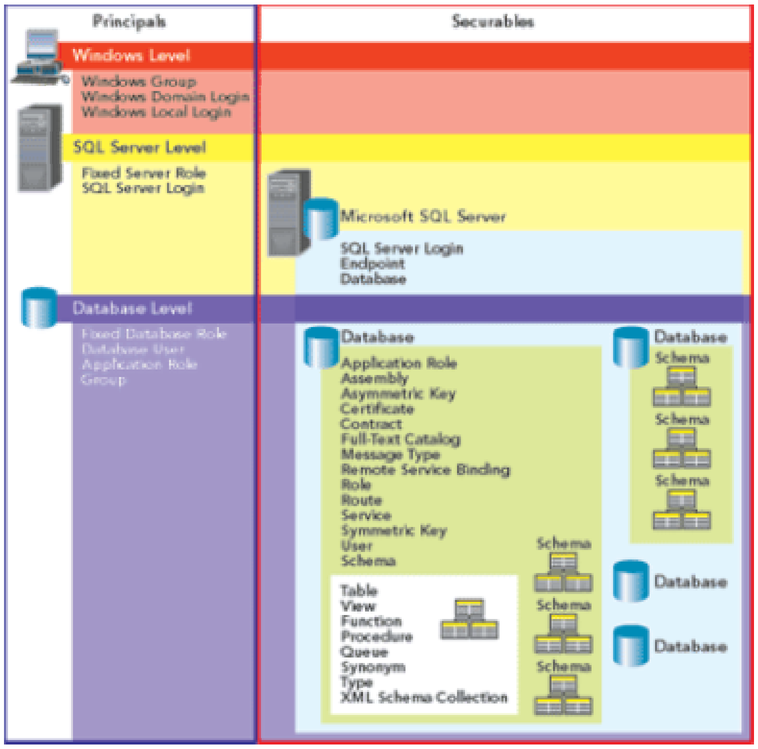

Figure 1 Security Hierarchy

The details of all the security changes in SQL Server 2005 are beyond the scope of this article, but I strongly suggest you read "Security Considerations for SQL Server" and "Security Considerations for Databases and Database Applications" as soon as you can.

The one change we are most concerned with here is that of metadata visibility restrictions. In SQL Server 2005, almost all visibility of metadata is restricted, and the only metadata that a user can see is that to which the user has some rights. In fact, the system tables are so restricted that not even a user in the sysadmin role can SELECT from them. All metadata access is through a set of views, called Catalog Views. Although the names of the catalog views will seem similar to the system table names in SQL Server 2000, the contents are frequently quite different. For example, there is a catalog view called objects, which

contains one row for each object in the current database. This view is in a schema called "sys", so to access this view, the following statement can be used:

```
SELECT * FROM sys.objects
```

Although this looks similar to accessing the sysobjects table in SQL Server 2000, the data columns returned will be different.

To allow a little more backward compatibility, there is also a set of views called compatibility views which have exactly the same names, and present data in the same form as the SQL Server 2000 system tables. For example, there is a compatibility view called sysobjects, which can be accessed through the dbo schema like so:

```
SELECT * FROM dbo.sysobjects
```

This view will return exactly the same columns as the sysobjects table in SQL Server 2000. In SQL Server 2005, the catalog views and the compatibility views have restricted visibility, so by default they only return information about objects to which the user has some rights. The only exceptions to restrictions on visibility apply to the metadata stored in the msdb database, to control and monitor activities such as log shipping, backup and restore, and replication. Figure 2 details the scope of restrictions.

If an administrator on SQL Server 2005 wants full backward compatibility, and does not want to restrict metadata visibility, a new SQL Server 2005 permission can be used. The permission, VIEW DEFINITION, allows a user or role to see the definition of an object or all objects within a particular scope. So if an administrator ran the following two statements, all users would be able to view all the metadata on a SQL Server instance:

```
GRANT VIEW ANY DEFINITION TO public
GRANT VIEW SERVER STATE TO public
```

Both of these statements grant server-level permissions. The first GRANT statement allows all server logins to view any metadata in any database. The second allows all logins to see the dynamic management views that contain server run-time information. These replace the SQL Server 2000 virtual tables, including sysprocesses, which was an unrestricted table in older versions.

Without the word ANY, the first GRANT statement grants database-level permissions to view all definitions to all users of one particular database. Figure 3 shows the relationship of the

more global permissions available in SQL Server 2005 to the specific permissions just mentioned. Thus a login with the CONTROL SERVER permission would automatically have both the permissions needed to view all server metadata, and a user with CONTROL permission on a database would be able to see all the metadata within that database.

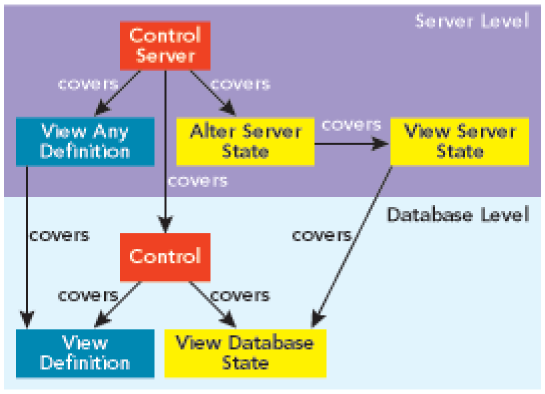

Figure 3 Permissions Relationships

Within a database, the GRANT VIEW DEFINITION statement can be further qualified to grant metadata access for a particular schema or object. Please see "VIEW DEFINITION Permission" for more details.

The new model of restricted metadata visibility is intended to provide better control for DBAs. Although there is a great deal of information on the new security model, some of the features dealing with metadata visibility are not well documented. Several of these features can have an impact on your applications when upgrading to SQL Server 2005, and you should be aware of which ones they are.

Database Visibility

A new permission, called VIEW ANY DATABASE, allows the grantee to see the metadata that describes all databases, regardless of whether the login owns or can actually use a particular database. Without this VIEW ANY DATABASE permission, the normal rules of metadata visibility apply, and you can't see any information about databases for which you do not have rights. However, to determine if you have rights for a database, SQL Server needs to open the database. This can be a very expensive operation on a busy server with lots of databases. And if the database is offline, it will look like you don't have access to the database, whether or not that is the case.

You need a way to get a list of all the databases quickly and reliably, and VIEW ANY DATABASE provides that functionality. In SQL Server 2005, this permission is granted to public by default. When VIEW ANY DATABASE is revoked, a user can only see master, tempdb, any database he owns, and the user's current database context.

Distributed Partitioned Views

Distributed queries need to evaluate CHECK constraints on the remote servers by querying the remote server's catalog. However, the check constraints are only visible to the object owners or users who have been granted CONTROL, ALTER, TAKE OWNERSHIP, or VIEW DEFINITION permissions. If a client trying to access a distributed partitioned view doesn't have permission to access the CHECK constraint's definition, the remote query will fail. The solution is to grant one VIEW DEFINITION on the view to whichever roles or users need to access the view.

In the case of a local partitioned view, the evaluation of the CHECK constraints happens inside the local server and runs under a system context, so no additional permissions other than SELECT from the view are needed. For a distributed view to access the remote server's catalog, it executes under the caller's credentials.

Dependency Management

Restrictions in metadata visibility mean that you might not be able to see objects that affect your own actions in your own schemas. Suppose user Sue owns sue_schema and has created a user-defined type sue_int. Sue then grants user Dan REFERENCES permission on sue_int. Suppose Dan then creates a table in his schema dan_schema, using sue_int as the type of one column. If Sue tries to drop sue_int, she will get an error that the type is currently in use. If she looks in the sys.columns view, she may get no results, because she doesn't have permissions on the table that contains her type:

```
SELECT * FROM sys.columns
WHERE user_type_id = type_id('sue_schema.sue_int')
```

The solution is to use a usage view, which is provided in SQL Server 2005 to show ID bindings. Available usage views include:

```
module_assembly_usages
type_assembly_usages
fulltext_index_catalog_usages
parameter_type_usages
column_type_usages
parameter_xml_schema_collection_usages
column_xml_schema_collection_usages
```

The one you need is column_type_usages, and the following query will show Sue which table ID is using sue_int:

```
SELECT * FROM sys.column_type_usages AS ctu
   JOIN sys.types AS t ON t.user_type_id = ctu.user_type_id
WHERE t.name = ‘my_int’;
```

Potential Grantees

Restricting metadata visibility means that commands like sp_helpuser and SELECT * FROM sys.database_principals will only return my name and the fixed roles. However, if I want to grant access to other users on a table I have just created, I have no way to find out names of other users.

A solution is to use a DDL (Data Definition Language) trigger that fires every time a new user or role is created. One such trigger, shown in Figure 4, will extract the type of event (create user or create role) and the name of the principal being created, and then generate a dynamic SQL statement to create view definition on that principal to public. So now, every time a new user or role is added, everyone in the database can see the name.

Application Role Limitations

With restrictions on metadata visibility, application roles are completely limited to seeing only objects in the database in which they were created. Application roles can see nothing in the master database, including syslogins, sysservers, and sysprocesses (and their SQL Server 2005 replacements).

There are two solutions for when you need an application role to have some access to the master database. First, a new trace flag has been introduced. If you enable traceflag 4616, SQL Server turns off catalog permission checks completely for application roles. This solution is recommended for upgrading old applications to the new version, but probably leaves your master database more exposed than you might like, as it applies to all tables in the master database. It should be used as a temporary solution only.

A second solution is to create a procedure requiring a feature in SQL Server 2005 called Module Signing, which accesses the desired objects in the master database. The procedure requires a signature in order to be executed by the application role. This can be used to solve various problems concerning limited visibility. Module Signing is beyond the scope of this article, but you should keep it in mind when upgrading applications that use application roles.

Conclusion

Access to metadata is a privilege in SQL Server 2005. In general, users have automatic access to data they need to see, including metadata for databases and objects to which they have rights. But, access to other metadata must be granted. After reading this article you should understand how metadata access works, how you can properly grant access, and how to diagnose any problems you may have seeing metadata in your own servers and databases.

***~~~ End of Article ~~~***