Understanding SQL Server fixed database roles

Join

By: K. Brian Kelley | Updated: 2009-12-14 | Comments (18) | Related: More > Security

Problem

I know there are fixed database roles that come with SQL Server. How do I best use them within my installations? What should I watch out for? In this tip we will cover each of the database roles and recommendations on when to and when not to use them.

Solution

Within each database, SQL Server does have fixed database roles, ones which are standard and included in every database. These differ from the database roles you can create and use yourself in that they have pre-assigned permissions. The fixed database roles are:

- db_owner
- db_securityadmin
- db_accessadmin
- db_backupoperator
- db_ddladmin
- db_datareader
- db_datawriter
- db_denydatareader
- db_denydatawriter

Like with the server roles, let's look at each in turn.

db_owner

We'll start with the biggest role: db_owner. A member of the db_owner role can do anything inside the database. Now there is a difference between a member of the db_owner role and the dbo user. That difference is that if someone maps into the database as the dbo user, that person bypasses all security checks. An example of this is anyone who is a member of the sysadmin fixed server role. They map in as dbo. And as a result, they don't receive security checks.

If a user is not dbo but is a member of the db_owner role, it does receive a security check. Of course, unless you've explicitly used DENY to block access, that user can do what he or she wants. The DENY stops them cold (it does not stop dbo). However, a member of the db_owner role could remove the DENY, so effectively that person can do anything, even if you put roadblocks in place. Therefore, the db_owner role should be given out only when necessary. Some applications will require it, which is a headache, but rarely will actual people need it (unless the application is connecting using their credentials). So you should be able to keep a reasonable tight control over this role. Like sysadmin, which is returned as a member of every fixed server role if you use the IS_SRVROLEMEMBER() function, if you query for someone who is a member of the db_owner role to determine if that user is a member of any other fixed database role, it will return true, even if the user is not explicitly a member of that role. For instance, a user who is a member of db_owner but who is not a member of db_securityadmin will still return a 1 if you execute the following query:

```
SELECT IS_MEMBER('db_securityadmin');
```

Things to remember:

- The db_owner role allows a user to do anything within the database.
- DBAs who are already members of the sysadmin fixed server role come in as dbo and don't need this role explicitly granted to them.
- Normal users should not be a member of this role.
- Applications might require their user account to be a member of this role.

db_securityadmin

Like the securityadmin fixed server role, the db_securityadmin fixed database role manages security. In this case, it manages role membership (with the exception of db_owner) as well as permissions on securables. As a result, it's another role you want to keep a close eye on. Generally speaking, I've not seen a lot of folks use this role. Typically the DBAs manage security within the database and they're already coming in as dbo. There may be some rare instances where it would be used, but I would flag those as exceptions. Therefore, if you see any members of this role within a database, it's worth checking out.

- The db_securityadmin role can manage role membership and permissions on securables.
- Again, since DBAs usually manage security and are usually coming in as dbo, this role is little used.
- Normal users should not be a member of this role.
- Applications should tend not to need this role.
- Since it's little used, you should audit its membership for exceptions.

db_accessadmin

The db_accessadmin role also manages security, but handles access to the database, as the name implies. The db_accessadmin role grants, denies, or revokes permission to enter the database for logins. Combined with db_securityadmin, and you can completely manage security into and throughout the database. Like db_securityadmin, though, access into the database is usually handled by DBAs. If they aren't members of the sysadmin fixed server role, they are members of the securityadmin fixed server role. As a result, this role should also be rarely used.

- The db_accessadmin role can allow access into or block access to the database for logins.
- Again, since DBAs usually manage security and have an appropriate server-level role, this role is little used.
- Normal users should not be a member of this role.
- Applications should tend not to need this role.
- This is another role you should audit for membership exceptions.

db_backupoperator

The db_backupoperator allows a member of the role to take backups of the database. However, it's only going to allow native backups, as in the standard backups through SQL Server itself. If you're using a third party product, chances are it is usually the methods which allow for high speed backups. Unfortunately, these methods require the login executing them to be a member of the sysadmin fixed server role. As a result, this role tends to be of limited usefulness. Add to it that you're backing up to a local drive, and it's rare to see a non-DBA having this level of access, even in a development system. Because of all these things, this is another role that is typically not used much.

- The db_backupoperator role allows a user to take backups of the database.
- Most 3rd party backup utilities utilize methods that require sysadmin rights, which this doesn't give.
- Another role that is little used because this functionality is usually handled by DBAs or a service account.
- Normal users should not be a member of this role.
- Applications should tend not to need this role, though I have seen exceptions.

db_ddladmin

The db_ddladmin is another powerful role because it allows a user to create, drop, or modify any objects within a database, regardless of who owns it. So a user could alter a stored procedure owned by dbo, for instance. This role is sometimes given to developers on non-production systems as they built custom applications. However, there is typically no reason anyone should be a member of this role on a production database. One thing the db_ddladmin does not do is allow the user to alter permissions on the objects. So a member of this role can create or modify the object, such as a stored procedure, but not alter the permissions on it unless he or she is the owner. So, for instance, a member of this role could create a stored procedure in a schema owned by dbo, but couldn't grant the ability to execute it.

- The db_ddladmin role can create, drop, and alter objects within the database, regardless of who the owner is.
- The db_ddladmin role cannot alter security.
- It is not unusual to grant this role to developers in a non-production environment.
- Normal users should not be a member of this role.
- Applications should not need this role.
- No one should normally be a member of this role on a production database.

db_datareader

The db_datareader role allows a user to be able to issue a SELECT statement against all tables and views in the database. DENY for a user (or a role the user is a member of) will still block the SELECT, however. But if there are no permissions set, whatsoever, the user will have the ability to SELECT against the table or view. The catch with this role is that the permission is implicit. That means if you query sys.database_permissions, you will not see any permission granted, either to the db_datareader role or directly to the user. Therefore, if you need to audit for everyone who has SELECT access to particular tables in a database, you'll have to query the membership of this group via the use of sp_helprolemember:

```
EXEC sp_helprolemember 'db_datareader';
```

It is not unusual to see the db_datareader role used in databases. It's an easy way to grant SELECT permissions to everything without having to worry about it. However, due to the fact that it uses implicit permissions, I prefer to create a user-defined database role and explicitly grant permissions. With that said, here are things to remember:

- The db_datareader role gives implicit access to SELECT against all tables and views in a database.
- In SQL Server 2005 and up, an explicit DENY will block access to objects.
- It is not unusual to see this role used in production for developers.
- It is not unusual to see this role used in production for normal users.
- Applications will occasionally need this role.
- Creating a user-defined database role and explicitly defining permissions is still preferred over the use of this role.

db_datawriter

The db_datawriter role is like the db_datareader role in that it gives implicit access to tables and views within a database. It also can be blocked by an explicit DENY for the user or for a role the user is a member of. Unlike db_datareader, however, db_datawriter gives INSERT, UPDATE, and DELETE permissions . Again, since the permission is implicit, you will not see these rights show up in sys.database_permissions. And like with db_datareader, you'll have to check the membership of this role to determine actual permissions in the event of an audit.

- The db_datawriter role gives implicit access to INSERT, UPDATE, and DELETE against all tables and views in a database.
- In SQL Server 2005 and up, an explicit DENY will block access to objects.
- Typically developer are not members of this role in production unless all users are.
- While less common than with db_datareader, it is not all that unusual to see this role used in production for normal users.
- Applications will occasionally need this role.
- Creating a user-defined database role and explicitly defining permissions is still preferred over the use of this role.

db_denydatareader

Unlike the previous two roles, db_denydatareader denies access. In this case, the db_denydatareader is the same as having a DENY for SELECT on all tables and views in the database. Because DENY trumps everything else, this is not a role I've seen used frequently. If there are no permissions for a given user on an object, such as the user has no SELECT permissions on a table, then SQL Server blocks access. Therefore, if a user doesn't have SELECT permission on TableA, then the user cannot successfully issue a SELECT query against TableA. An explicit DENY is not needed. And since this affects all tables and views, that adds to the reason this database role is typically not used. And like db_datareader and db_datawriter, the DENY is implicit, meaning you'll have to query for membership in this role to determine who is affected.

- The db_denydatareader role is denied access to SELECT against any table or view in the database.
- Typically this role is not used.
- The DENY is implicit.
- Creating a user-defined database role and explicitly defining permissions is still preferred over the use of this role.

db_denydatawriter

Wrapping up our list of roles is db_denydatawriter. The db_denydatawriter has an implicit DENY on INSERT, UPDATE, and DELETE for all tables and views in the database. Again, this is not a role that sees much use, for the same reasons as db_denydatareader.

- The db_denydatawriter role is denied access to INSERT, UPDATE, or DELETE against all tables and views in the database.
- Typically this role is not used.
- The DENY is implicit.

- Creating a user-defined database role and explicitly defining permissions is still preferred over the use of this role.

Next Steps

Last Updated: 2009-12-14

About the author

K. Brian Kelley is a SQL Server author and columnist focusing primarily on SQL Server security.

View all my tips

Related Resources

More SQL Server DBA Tips...