http://www.sqlservercentral.com/articles/Administration/61648/
Printed 2007/12/04 05:28AM

Fixing Orphaned Users

By Timothy Ford, 2007/11/30

Billy the Bard and the Merry Orphans of Windsor

Migrating databases between production, test, development and train servers is something that a DBA needs to deal with constantly. During migration, one of the inevitable issues we need to deal with is concept of *Orphaned Users*. Orphaned users occur when a SQL login's SID in **master..syslogins** does not match from server-to-server.

All the World Is… Setting the Stage

For this exercise, you will need two SQL instances to work with. For the sake of this discussion, we will refer to these instances as **SQL1** and **SQL2**.

Run the following script on the both SQL instances (**SQL1** and **SQL2**). It will create the database we will be using for this little exercise. Note that you will need to substitute a file name that is acceptable to your environment for the data and log files:

```
USE [master]
GO
CREATE DATABASE [Test_DB] ON  PRIMARY
  (
        NAME = N'Test_DB',
        FILENAME = N'D:\MSSQL\Data\Test_DB.mdf' ,
        SIZE = 102400KB ,
        MAXSIZE = UNLIMITED,
        FILEGROWTH = 102400KB
  )
     LOG ON
  (
        NAME = N'Test_DB_log',
        FILENAME = N'E:\MSSQL\Logs\Test_DB_log.ldf' ,
        SIZE = 51200KB ,
        MAXSIZE = 2048GB ,
        FILEGROWTH = 51200KB
  )
     COLLATE SQL_Latin1_General_CP1_CI_AS
GO
```

Next, run the following script to create two logins and to grant user rights to the new database on both SQL instances:

```
USE [master]
GO
CREATE LOGIN [TestLogin1] WITH PASSWORD=N'foo',
	DEFAULT_DATABASE=[master],
	CHECK_EXPIRATION=OFF,
	CHECK_POLICY=OFF
GO
	USE [Test_DB]
GO
CREATE USER [TestLogin1] FOR LOGIN [TestLogin1]
ALTER USER [TestLogin1] WITH DEFAULT_SCHEMA=[dbo]
EXEC sp_addrolemember N'db_datareader', N'TestLogin1'
EXEC sp_addrolemember N'db_datawriter', N'TestLogin1'
GO
	USE [master]
GO
CREATE LOGIN [TestLogin2] WITH PASSWORD=N'foo',
	DEFAULT_DATABASE=[master],
	CHECK_EXPIRATION=OFF,
	CHECK_POLICY=OFF
GO
	USE [Test_DB]
GO
CREATE USER [TestLogin2] FOR LOGIN [TestLogin2]
ALTER USER [TestLogin2] WITH DEFAULT_SCHEMA=[dbo]
EXEC sp_addrolemember N'db_owner', N'TestLogin2'
GO
```

I do believe we are ready to do a bit of querying against the system tables (system views) in the master database. Before we do so, a quick aside.

ASIDE:

By now most of you are aware that in SQL 2005, the system tables we were always told not to rely on, but always did, vanished into the ether. Well, actually, they just vanished into the Resource database: that shadowy database that runs things behind the scenes – the database on the grassy knoll – the database behind the curtain. Pay no attention to the database behind the curtain however! Thankfully, Microsoft replaced the system tables with system views that rely on those tables, maintain the same naming convention as those tables (albeit with the new schema of sys) and have the same structure as those old SQL 2000 system tables. You'll see throughout this article my reference to system tables/views. It all depends upon which version of SQL you are working with. I've been using these "compatibility views", such as **sysusers**, for the 3+ years that they been in place, since they are backward compatible to SQL 2000. However, certain columns may be subject to an "overflow" problem (described here: http://technet.microsoft.com/en-us/library/ms187376.aspx) and some new SQL 2005 features, such as Service Broker and partitioning, require use of the new **catalog views**, rather than these compatibility views.

If you're likely to be affected by this, you'll want to swap the references to the compatibility views to the equivalent catalog views. For example, the equivalent of **sysusers** in the catalog views is **sys.database_principals**.

First let's look at the results of the following query on both SQL instances:

```
--Using SQL1 Instance
SELECT SL.name, SL.sid, SL.isntgroup, SL.isntname, SL.loginname, 'SQL1' AS Instance
  FROM MASTER..syslogins SL
  WHERE SL.name LIKE 'TestLogin%'
```

name	sid	isntgroup	isntname	loginname	Instance
TestLogin1	0x4E3710A5718DF6428DDE04F7EC833A720	0	0	TestLogin1	SQL1
TestLogin2	0x9CF711D7B570604C89CF067101AF9595	0	0	TestLogin2	SQL1

```
--Using SQL2 Instance
SELECT SL.name, SL.sid, SL.isntgroup, SL.isntname, SL.loginname, 'SQL2' AS Instance
  FROM MASTER..syslogins SL
  WHERE SL.name LIKE 'TestLogin%'
```

name	sid	isntgroup	isntname	loginname	Instance
TestLogin1	0x22EC7B3BAC9A95479CE35A0B1B70AC77	0	0	TestLogin1	SQL2
TestLogin2	0xAF59FDD422E39C4BAADCA7DAD056CD2C	0	0	TestLogin2	SQL2

In comparing both instances you can see that, at this point, all things being equal, the sids for the logins are different for each SQL instance.

Let us now turn our attention to the test databases and their **sysusers** system table (system view):

```
--Using SQL1 Instance
SELECT SU.name, SU.sid, SU.islogin, SU.isntgroup, SU.isntname, 'SQL1' AS Instance
  FROM Test_DB..sysusers SU
  WHERE SU.name LIKE 'TestLogin%'
```

name	sid	islogin	isntgroup	isntname	Instance
TestLogin1	0x4E3710A5718DF6428DDE04F7EC833A721	0	0	SQL1	
TestLogin2	0x9CF711D7B570604C89CF067101AF9595	1	0	0	SQL1

```
--Using SQL2 Instance
SELECT SU.name, SU.sid, SU.islogin, SU.isntgroup, SU.isntname, 'SQL2' AS Instance
  FROM Test_DB..sysusers SU
  WHERE SU.name LIKE 'TestLogin%'
```

name	sid	islogin	isntgroup	isntname	Instance
TestLogin1	0x22EC7B3BAC9A95479CE35A0B1B70AC77	1	0	0	SQL2
TestLogin2	0xAF59FDD422E39C4BAADCA7DAD056CD2C	1	0	0	SQL2

At this time, if we were to attempt to connect to the **Test_DB** database on either instance, with either the TestLogin1/foo, or TestLogin2/foo login/password, we would get into the database with no issues arising.

A Login by Any Other Name Would Work…

Fast forward a few weeks; we've been using **SQL1** as our development server and **SQL2** is our

production server. We are preparing to take our database live by migrating it from **SQL1** to **SQL2**. I will not delve into what is or is not best practice for backing up and restoring databases between instances. To do so would be akin to me telling everyone that chocolate is the best flavor ice cream of all (which it is of course!), and would open up a heated and distracting debate. However, here are the scripts I use for my backup and restore:

```
--Using SQL1 Instance
  BACKUP DATABASE [Test_DB]
       TO  DISK = N'\\SQL\Backup\Test_DB.bak'
       WITH NOFORMAT, INIT, SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO
  --Using SQL2 Instance
       RESTORE DATABASE [Test_DB]
  FROM  DISK = N'\\SQL\Backup\Test_DB.bak'
       WITH  FILE = 1, NOUNLOAD,  STATS = 10, REPLACE
       MOVE N'Test_DB' TO N'D:\MSSQL\Data\Test_DB.mdf',
       MOVE N'Test_DB_log' TO N'E:\MSSQL\Logs\Test_DB_log.ldf'
```

Once restored on **SQL2**, if we re-execute the following query we get different results in the **Test_DB** database:

```
--Using SQL2 Instance
SELECT SU.name, SU.sid, SU.islogin, SU.isntgroup, SU.isntname, 'SQL2' AS Instance
  FROM Test_DB..sysusers SU
  WHERE SU.name LIKE 'TestLogin%'
```

name	sid	islogin	isntgroup	isntname	Instance
TestLogin1	0x4E3710A5718DF6428DDE04F7EC833A721	0	0		SQL2
TestLogin2	0x9CF711D7B570604C89CF067101AF9595	1	0	0	SQL2

Do those SID values look familiar? Of course they do. They are the same values as we saw for the query against **SQL1.Test_DB..sysusers**. This does make sense; we've restored the database and all of its contents over to the SQL2 instance.

Now, what happens if we attempt to connect to the database with either **TestLogin1** or **TestLogin2**? A large bucket of failure is cast at your feet:

```
--Msg 916, Level 14, State 1, Line 1
The server principal "TestLogin1" is not able to access the database "Test_DB" under the current
security context.
```

Why did this happen? Let's take a step back and look at how the users in the databases are resolved to the logins at the instance level.

By The Pricking Of My Thumbs, Something Wicked This Way Comes

A login against a SQL instance is initially checked for validity of the login/password combination. Once confirmed as valid, a login's SID must match between the instances' **master..syslogins** table (view) and

a database's **sysusers** table (view). SIDs do not follow a consistent evolution from instance to instance. It would be quite a security issue if they did, wouldn't it? This is why you see different SID values for the same login on between **SQL1** and **SQL2**. The logins are really the same in name only.

The users are orphaned after the database restore because the SID values in the **sysusers** table (view) do not match the SID value for the corresponding login in the **master..syslogins** table (view) on **SQL2**. Thankfully, Microsoft has the functionality to reconcile the users and the logins.

Friends, Romans, Contractors, Lend Me a Stored Procedure

The **sp_change_users_login** stored procedure will synchronize SID values for SQL logins. The syntax for this system stored procedure in the master database is as follows, per SQL Books Online:

sp_change_users_login[**@Action** =] '*action*'
  [**,** [**@UserNamePattern** =] '*user*']
  [**,** [**@LoginName** =] '*login*']
    [**,** [**@Password** =] '*password*']

Where **@Action** can be one of three values:

- **Auto_Fix**, will link a user in the current database's sysusers table with a SQL Server login of the same name in the master database's syslogins table. If the login does not exist in master then it will be created. This means you'll need to supply a password if the login does not exist.
- **Report**, simply lists the user(s) and corresponding SID(s) in the current database that are not linked to a SQL login in the master database.
- **Update_One**, which is the focus of this article, will link a specified user in the current database to an existing login in the master database's syslogins table. It is only a matter of supplying the stored procedure with the "Update_One" parameter, along with values for the following two parameters:

The **@UserNamePattern** is the name of the orphaned user in the current database and the **@LoginName** parameter must be a valid login name as it exists in **master..syslogins**. In both cases the datatype is **sysname** and they default to NULL.

Now the fun begins. With a simple cursor (yes, I said C-U-R-S-O-R), we can loop through any SQL users in the database and reconcile their SID values to those of their corresponding logins. You may be asking why I am only interested in SQL logins. It is simply because only SQL logins are orphaned. This is another reason for using integrated security whenever possible.

On that note, let's look at the code that will save you time every time it's used.

```
--Using Test_DB
DECLARE @user SYSNAME
DECLARE @SQL NVARCHAR(300)
DECLARE cur_Users CURSOR FOR
SELECT name
  FROM sysusers
  WHERE islogin = 1
    AND isntname = 0
    AND NAME NOT IN ('guest', 'dbo', 'sys', 'INFORMATION_SCHEMA')
```

```
  ORDER BY name
OPEN cur_Users
      FETCH NEXT
  FROM cur_Users INTO @user
WHILE @@FETCH_STATUS = 0
  BEGIN
  SELECT @SQL = 'EXEC sp_change_users_login ' + "" + 'UPDATE_ONE'
+ "" + ', ' + "" + @user + "" + ', ' + "" + @user + ""
  EXEC sp_executesql @SQL
        FETCH NEXT
    FROM cur_Users INTO @user
  END
      CLOSE cur_Users
      DEALLOCATE cur_Users
```

This script will loop through all SQL logins that are not aliased as guest, dbo, sys, or **INFORMATION_SCHEMA**. If, by chance the SIDs match then the following message will be presented:

The number of orphaned users fixed by updating users was 0.

If the user does not exist in **master..syslogins**, you'll receive an error. To demonstrate this, let's assume for a second that, during the development phase, we added another login and corresponding user to **SQL1**:

```
USE [master]
GO
CREATE LOGIN [TestLogin3] WITH PASSWORD=N'foo',
      DEFAULT_DATABASE=[master],
      CHECK_EXPIRATION=OFF,
      CHECK_POLICY=OFF
GO
      USE [Test_DB]
GO
CREATE USER [TestLogin3] FOR LOGIN [TestLogin1]
ALTER USER [TestLogin3] WITH DEFAULT_SCHEMA=[dbo]
EXEC sp_addrolemember N'db_datareader', N'TestLogin3'
EXEC sp_addrolemember N'db_datawriter', N'TestLogin3'
GO
```

After restoring **Test_DB** to **SQL2** you would have a user in the database that has no corresponding login on the server. You would receive the following error when the user reconciliation code cursor runs against the **TestLogin3** user:

```
Msg 15291, Level 16, State 1, Procedure sp_change_users_login, Line 131
Terminating this procedure. The Login name 'TestLogin3' is absent or invalid.
```

You can add the login to the SQL instance and re-run the script thereby reconciling the user and login.

Our Remedies Oft In Ourselves Do Lie

When you have to support 800+ databases on your own it becomes a managerial nightmare to have a different set of code commands for each database's user reconciliation process. By harnessing the metadata of your SQL instance, you are able to accomplish mighty things with little effort. Sure, the caveats of not relying on system tables need to be stated, but they are a powerful tool to wield until they are removed from general use. Any method that you can harness to increase your productivity is worth the initial effort.

As Billy "The Bard" Shakes stated: "Better three hours too soon than a minute too late"
