31 Days of Disaster Recovery

By Robert L Davis, originally posted on www.sqlsoldier.com

Back in January, 2013, I started a blog series dubbed 31 Days of Disaster Recovery. I posted 31 blog posts (though it took longer than 31 days) covering a variety of disaster recovery topics such as backups, restore, corruption, best practices, etc. I have decided to make the blog post series into a downloadable eBook for easy consumption.

# Day 1: Does DBCC Automatically Use Existing Snapshot?

Welcome to my series on Disaster Recovery. I will spend the entire month of January focusing on all things related to disaster recovery including topics like corruption, data integrity, data loss, DBCC commands, and more.

For my first post of this month, I want to take a look at the myth that the DBCC CHECK commands will automatically use an existing database snapshot if one exists for the database. This is a myth I believed myself at one time and told to others. This was done to prove it to myself as much as to anyone else. This is an attempt to prove definitively that the DBCC CHECK commands will not use an existing snapshot for a database.

## Does DBCC Automatically Use Existing Snapshot?

I was doing some bigger investigations into DBCC CHECK commands (post to come) and was looking for a way to see that hidden snapshot the DBCC CHECK commands create and use. The snapshots are not visible in sys.databases, sys.master\_files, nor any other system catalog that I could find. Additionally, the snapshot creation does not trigger server level events for a DDL trigger nor the database create or database start events for SQL Trace or Extended Events.

I found it in the *databases\_dbcc\_logical\_scan* event via Extended Events. The event reports database\_id of the database the command was run against as well as the database\_id of the database where the action is actually occurring. If the action database is the hidden snapshot, the database\_id will not show up in sys.databases, but the DB\_NAME() function will return the name of the source database. I set up an Extended Events session to capture this event with the columns database\_id and database\_name. I use the ring buffer target because I don’t intend to retain any of this data. This session will not be active yet.

CREATE EVENT SESSION [TestSnap] ON SERVER

ADD EVENT sqlserver.databases\_dbcc\_logical\_scan(

 ACTION(sqlserver.database\_id,

 sqlserver.database\_name))

ADD TARGET package0.ring\_buffer

WITH (MAX\_DISPATCH\_LATENCY=30 SECONDS,

 MAX\_EVENT\_SIZE=0 KB);

Then I create a snapshot of the AdventureWorksDW2012 database called AWSnap so there is an existing snapshot.

CREATE DATABASE AWSnap

 ON (NAME = N'AdventureWorksDW2012\_Data',

 FILENAME = N'C:\bakAdventureWorksDW2012\_Data.ndf')

 AS SNAPSHOT OF AdventureWorksDW2012;

Now, using Object Explore in SQL Server Management Studio (SSMS), I start the Extended Events session (expand Management -> expand Extended Events -> right-click on the session -> click Start Session). Then I right-click on the session and click on Watch Live Data. This allows me to see the events in real-time while DBCC CHECKDB is running against the AdventureWorksDW2012 database. Next I open a new query window and run DBCC CHECKDB.

We can see by looking at any one entry in the Extended Events session that the database\_id and database\_id (Action) are different database IDs. If you query sys.databases, you will see that database\_id of 6 is the AdventureWorksDW2012 database and there is no entry for the database\_id (Action). If we check the action database\_id with the DB\_NAME() function while the CHECKDB is running, it will return AdventureWorksDW2012, but after DBCC completes, it will return NULL.



DBCC Scan Output

For the next part of the test, I close out the Live Data window and reopen it so that it is all clear again. Then I run DBCC CHECKDB against the snapshot AWSnap. checking the Live Data window, we see that the database\_id and database\_id (Action) are now the same value. If you query sys.databases, you will see that the value for both maps to the snapshot AWSnap.



DBCC Scan Output

## Summary

As I have shown here, if you want to control the snapshot that DBCC uses, you need to specifically run DBCC on the snapshot itself. It won’t detect and use the snapshot just because one exists. The reason why this is important to understand will become apparent in a later post. So be sure to check back and see how I build on this information.

# Day 2: Protection From Restoring a Backup of a Contained Database

Welcome to day 2 of my month-long series on Disaster Recovery. For today’s post, I want to talk a little bit about restoring backups of contained databases. In particular, what protections are in place in case you are given a backup of a contained database to restore without being told it has containment enabled.

## Restoring a Contained Database

One of the first concerns I had when I learned about contained database was how am I as a DBA protected against giving someone access to a database without my knowledge. Well, there are several layers of protection starting with the fact that it is up to the administrator to enable containment before it even comes in to play.

## First Layer of Protection

Containment has to be enabled at the server level before a contained database can be restored or created or before an existing database can have containment enabled. As long as you do not enable this setting at the server level, you’re completely protected. You can enable it via sp\_configure or via the Server Properties dialog (Advanced tab). Despite the fact that it is listed on the Advanced tab, it is not an advanced option. You do not have to enable **Show advanced options** before changing or checking this setting. I see a lot of blog posts stating that it is an advanced option and showing that the Show advanced options has to be set, but it is not and does not.

In addition to sp\_configure and the Server Properties dialog, you can check the current setting by querying the sys.configurations table. This is my preferred method for querying for configuration settings because it is a lot easier to use in automation scripts. However, if my intent to change it if it needs it, I generally use sp\_configure.

If you have not enabled containment at the server level at you attempt to restore a database that has containment enabled, the restore will fail with an error telling you that containment has to be enabled first. This is a warning to the DBA that the database is contained. It is up to you to decide whether enabling containment is the proper thing to do or not. Please don’t just enable it because the error message said to. This affects the security of your server and databases.

The error message you will receive is:

|  |  |
| --- | --- |
|  | Msg 12824, Level 16, State 1, Line 1The sp\_configure value 'contained database authentication' must be set to 1 in order to restore a contained database.  You may need to use RECONFIGURE to set the value\_in\_use.Msg 3013, Level 16, State 1, Line 1RESTORE DATABASE is terminating abnormally. |

The option that you are enabling or disabling is **contained database authentication** and the command to enable/disable containment at the server level is as follows:

|  |  |
| --- | --- |
|  | -- Enable "contained database authentication"Exec sp\_configure 'contained', 1;Reconfigure; -- Disable "contained database authentication"Exec sp\_configure 'contained', 0;Reconfigure; |

## Second Layer of Protection

You will notice that I used RECONFIGURE in that code and not RECONFIGURE WITH OVERRIDE. I did so for a very important reason. I will get to that later, but first, let’s take a look at the next layer of protection. Let’s move a step further and see how you are protected after you have enabled containment at the server level. Once the option is enabled, you will get no warnings whatsoever when you restore a contained database. It’s up to you to make sure you know whether or not the backup you are going to restore is contained or not. Fortunately, there is a built-in mechanism for discovering that.

If you are presented with a backup of a new database, you can check for containment yourself. You don’t have to rely on being told about it. A new column (Containment) has been added to the output of the RESTORE HEADERONLY command. You can use this command to inspect the header information for the backup before restoring it. If Containment = 1, containment is enabled in the database. If Containment = 0, it is not enabled.

In this example, I am checking the header information of a backup named c:\bakCDTest.bak:

|  |  |
| --- | --- |
|  | Restore HeaderOnly From Disk = 'c:\bakCDTest.bak'; |

It’s very easy to check and there is no reason not to perform this check before restoring a database if the server has containment enabled.

## Final Layer of Protection

Let’s say you are in a disaster recovery scenario and you need to get the database back online as quickly as possible, but when you attempt to restore the database on a replacement server, you discover that it has containment enabled. Now let’s assume that you had not been aware that this setting was enabled and to the extent of your knowledge and available documentation, the database does not use contained users. Furthermore, let’s assume that there may be some compliancy regulations or security policies that say that you can’t have undocumented contained users. You may not have time to hunt down someone that can explain why there are contained users. You need to get the database restored and also respect your security restrictions.

One option you can do is to enable containment at the server level, restore the database, and then disable containment at the server level. If you try to disable containment at the server level using sp\_configure and RECONFIGURE, you will get an error message stating that you can not disable containment because there are contained databases. You can force disabling of containment at the server level by using RECONFIGURE WITH OVERRIDE. This is why I was sure to use just RECONFIGURE in the earlier example. I want to ensure that I don’t accidentally disable the setting if there are contained databases without realizing it.

Here is what happens when you try to disable containment at the server level when a contained database already exists:

|  |  |
| --- | --- |
|  | -- Disable "contained database authentication"Exec sp\_configure 'contained', 0;Reconfigure; |
|  | Configuration option 'contained database authentication' changed from 1 to 0. Run the RECONFIGURE statement to install.Msg 12818, Level 16, State 1, Line 3RECONFIGURE failed. Attempting to change the 'contained database authentication' value to 0 while there are existing contained databases requires a RECONFIGURE WITH OVERRIDE. |

|  |  |
| --- | --- |
|  | -- Force disabling of "contained database authentication"Reconfigure With Override; |
|  | Command(s) completed successfully. |

## Summary

There are built-in protections and mechanisms for protecting yourself from restoring a contained database, but there is also a responsibility on us as administrators to make sure we understand our actions and don’t get blind-sided by a contained database. These built-in features should be your second line of defense, and good practices and being diligent about being aware of what we are doing and the implications our actions should be our first line of defense. It’s on us to secure our databases and servers and the built-in functionality is just a fail-safe.

# Day 3: Determining Files to Restore Database

Welcome back for day 3 of my month-long series on Disaster Recovery. For today’s post, I want to address something that, in my experience, has really flummoxed a lot of people who find themselves unprepared to handle a disaster scenario. Determining exactly which files you should restore.

## Test Your Restores

Everyone on your team who might be called upon to perform a restore should be performing regular test runs of restoring the backups. It’s this lack of testing that leads to fumbling around trying to figure out what files to restore. Well, whether you’ve practiced or not, you might like a little help to figure out exactly what files are required to restore the database to its most current point. Thus I give you my RestoreScripter script. The script traverses the backup information in the backup tracking tables in the msdb database.

The script automatically handles many of the possible pitfalls that you might experience and gives you the details you need to quickly script out the restores. For example, records with the same “RestoreOrder” are multiple files for the same backup and should be scripted together in a single restore command.

## The Script

As usual, rather than explain the various steps in the script here, I have included extensive comments inline in the script at each step. You can grab the script below or download it as a zipped file here: [RestoreScripter.zip (2 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/RestoreScripter.zip)

Declare @DBName sysname,

 @DBBackupLSN numeric(25, 0);

Declare @Baks Table (

 BakID int identity(1, 1) not null primary key,

 backup\_set\_id int not null,

 media\_set\_id int not null,

 first\_family\_number tinyint not null,

 last\_family\_number tinyint not null,

 first\_lsn numeric(25, 0) null,

 last\_lsn numeric(25, 0) null,

 database\_backup\_lsn numeric(25, 0) null,

 backup\_finish\_date datetime null,

 type char(1) null,

 family\_sequence\_number tinyint not null,

 physical\_device\_name nvarchar(260) not null,

 device\_type tinyint null)

Set NoCount On;

-- Set the name of the database you want to restore

Set @DBName = N'';

-- Get the most recent full backup with all backup files

Insert Into @Baks (backup\_set\_id,

 media\_set\_id,

 first\_family\_number,

 last\_family\_number,

 first\_lsn,

 last\_lsn,

 database\_backup\_lsn,

 backup\_finish\_date,

 type,

 family\_sequence\_number,

 physical\_device\_name,

 device\_type)

Select Top(1) With Ties B.backup\_set\_id,

 B.media\_set\_id,

 B.first\_family\_number,

 B.last\_family\_number,

 B.first\_lsn,

 B.last\_lsn,

 B.database\_backup\_lsn,

 B.backup\_finish\_date,

 B.type,

 BF.family\_sequence\_number,

 BF.physical\_device\_name,

 BF.device\_type

From msdb.dbo.backupset As B

Inner Join msdb.dbo.backupmediafamily As BF

 On BF.media\_set\_id = B.media\_set\_id

 And BF.family\_sequence\_number Between B.first\_family\_number And B.last\_family\_number

Where B.database\_name = @DBName

And B.is\_copy\_only = 0

And B.type = 'D'

And BF.physical\_device\_name Not In ('Nul', 'Nul:')

Order By backup\_finish\_date desc, backup\_set\_id;

-- Get the lsn that the differential backups, if any, will be based on

Select @DBBackupLSN = database\_backup\_lsn

From @Baks;

-- Get the most recent differential backup based on that full backup

Insert Into @Baks (backup\_set\_id,

 media\_set\_id,

 first\_family\_number,

 last\_family\_number,

 first\_lsn,

 last\_lsn,

 database\_backup\_lsn,

 backup\_finish\_date,

 type,

 family\_sequence\_number,

 physical\_device\_name,

 device\_type)

Select Top(1) With Ties B.backup\_set\_id,

 B.media\_set\_id,

 B.first\_family\_number,

 B.last\_family\_number,

 B.first\_lsn,

 B.last\_lsn,

 B.database\_backup\_lsn,

 B.backup\_finish\_date,

 B.type,

 BF.family\_sequence\_number,

 BF.physical\_device\_name,

 BF.device\_type

From msdb.dbo.backupset As B

Inner Join msdb.dbo.backupmediafamily As BF

 On BF.media\_set\_id = B.media\_set\_id

 And BF.family\_sequence\_number Between B.first\_family\_number And B.last\_family\_number

Where B.database\_name = @DBName

And B.is\_copy\_only = 0

And B.type = 'I'

And BF.physical\_device\_name Not In ('Nul', 'Nul:')

And B.database\_backup\_lsn = @DBBackupLSN

Order By backup\_finish\_date Desc, backup\_set\_id;

-- Get the last LSN included in the differential backup,

-- if one was found, or of the full backup

Select Top 1 @DBBackupLSN = last\_lsn

From @Baks

Where type In ('D', 'I')

Order By BakID Desc;

-- Get first log backup, if any, for restore, where

-- last\_lsn of previous backup is >= first\_lsn of the

-- log backup and <= the last\_lsn of the log backup

Insert Into @Baks (backup\_set\_id,

 media\_set\_id,

 first\_family\_number,

 last\_family\_number,

 first\_lsn,

 last\_lsn,

 database\_backup\_lsn,

 backup\_finish\_date,

 type,

 family\_sequence\_number,

 physical\_device\_name,

 device\_type)

Select Top(1) With Ties B.backup\_set\_id,

 B.media\_set\_id,

 B.first\_family\_number,

 B.last\_family\_number,

 B.first\_lsn,

 B.last\_lsn,

 B.database\_backup\_lsn,

 B.backup\_finish\_date,

 B.type,

 BF.family\_sequence\_number,

 BF.physical\_device\_name,

 BF.device\_type

From msdb.dbo.backupset B

Inner Join msdb.dbo.backupmediafamily As BF

 On BF.media\_set\_id = B.media\_set\_id

 And BF.family\_sequence\_number Between B.first\_family\_number And B.last\_family\_number

Where B.database\_name = @DBName

And B.is\_copy\_only = 0

And B.type = 'L'

And BF.physical\_device\_name Not In ('Nul', 'Nul:')

And @DBBackupLSN Between B.first\_lsn And B.last\_lsn

Order By backup\_finish\_date, backup\_set\_id;

-- Get last\_lsn of the first log backup that will be restored

Set @DBBackupLSN = Null;

Select @DBBackupLSN = Max(last\_lsn)

From @Baks

Where type = 'L';

-- Recursively get all log backups, in order, to be restored

-- first\_lsn of the log backup = last\_lsn of the previous log backup

With Logs

As (Select B.backup\_set\_id,

 B.media\_set\_id,

 B.first\_family\_number,

 B.last\_family\_number,

 B.first\_lsn,

 B.last\_lsn,

 B.database\_backup\_lsn,

 B.backup\_finish\_date,

 B.type,

 BF.family\_sequence\_number,

 BF.physical\_device\_name,

 BF.device\_type,

 1 As LogLevel

 From msdb.dbo.backupset B

 Inner Join msdb.dbo.backupmediafamily As BF

 On BF.media\_set\_id = B.media\_set\_id

 And BF.family\_sequence\_number Between B.first\_family\_number And B.last\_family\_number

 Where B.database\_name = @DBName

 And B.is\_copy\_only = 0

 And B.type = 'L'

 And BF.physical\_device\_name Not In ('Nul', 'Nul:')

 And B.first\_lsn = @DBBackupLSN

 Union All

 Select B.backup\_set\_id,

 B.media\_set\_id,

 B.first\_family\_number,

 B.last\_family\_number,

 B.first\_lsn,

 B.last\_lsn,

 B.database\_backup\_lsn,

 B.backup\_finish\_date,

 B.type,

 BF.family\_sequence\_number,

 BF.physical\_device\_name,

 BF.device\_type,

 L.LogLevel + 1

 From msdb.dbo.backupset B

 Inner Join msdb.dbo.backupmediafamily As BF

 On BF.media\_set\_id = B.media\_set\_id

 And BF.family\_sequence\_number Between B.first\_family\_number And B.last\_family\_number

 Inner Join Logs L On L.database\_backup\_lsn = B.database\_backup\_lsn

 Where B.database\_name = @DBName

 And B.is\_copy\_only = 0

 And B.type = 'L'

 And BF.physical\_device\_name Not In ('Nul', 'Nul:')

 And B.first\_lsn = L.last\_lsn)

Insert Into @Baks (backup\_set\_id,

 media\_set\_id,

 first\_family\_number,

 last\_family\_number,

 first\_lsn,

 last\_lsn,

 database\_backup\_lsn,

 backup\_finish\_date,

 type,

 family\_sequence\_number,

 physical\_device\_name,

 device\_type)

Select backup\_set\_id,

 media\_set\_id,

 first\_family\_number,

 last\_family\_number,

 first\_lsn,

 last\_lsn,

 database\_backup\_lsn,

 backup\_finish\_date,

 type,

 family\_sequence\_number,

 physical\_device\_name,

 device\_type

From Logs

Option(MaxRecursion 0);

-- Select out just the columns needed to script restore

Select RestoreOrder = Row\_Number() Over(Partition By family\_sequence\_number Order By BakID),

 RestoreType = Case When type In ('D', 'I') Then 'Database'

 When type = 'L' Then 'Log'

 End,

 DeviceType = Case When device\_type in (2, 102) Then 'Disk'

 When device\_type in (5, 105) Then 'Tape'

 End,

 PhysicalFileName = physical\_device\_name

From @Baks

Order By BakID

;

Set NoCount Off;

**Sample Output**



Sample Restore Files

# Day 4: Back That Thang Up

Here we are at day 4 in my series focusing on disaster recovery. It’s Friday, so this will be a quick one for today. I want to talk about the importance backing up everything that needs it and point out some of the often overlooked things that you should be backing up.

## System Databases

I’ve said it before ([T-SQL Tuesday #19 – Beyond Backups](http://www.sqlsoldier.com/wp/sqlserver/beyondbackups)) and I’ll say it again. Back up your system databases: master, model, msdb, and distribution. You should also be backing up your resource DB as well. You can back up the resource DB (mssqlsystemresource.mdf and mssqlsystemresource.ldf files) by just copying them to your backup location. This can save you from losing a lot of critical data. If you lost master database and you have to rebuild master, you will lose everything in msdb and model as well. It can turn a difficult situation into a terrible one. Being ~~a lazy~~ an efficient dba means doing simple things (like backing up the system databases) to save yourself a lot of work later down the line.

## Certificates and Master Keys

If you are using certificates for creating asymmetric keys, authorizations, or securing securables (like procedures), then you need to consider the ramifications if you need to re-apply the certificate. For example, if you need to move a database with encrypted data to a new server, you may need to decrypt and re-encrypt the data. You will need the original certificate to do perform this action. You can export the certificate to a file using the BACKUP CERTIFICATE command. Likewise, you should export the database master key to a file using the BACKUP MASTER KEY command.

The backed up certificates and master keys should be immediately removed from the server and stored in a safe, secure location. I like to store them in source control that is securely protecting (in a tree only the DBAs have access to) and is itself getting backed up. Additionally, if you use a password to secure the exported files, those passwords should be stored securely using whatever secure method you are using to protect service account passwords. Preferably in something that is secure and gets backed up in it’s own right.

## SSRS Encryption Key

If you lose your SSRS instance and you need to connect a new one to the existing databases, you will need to import the original encryption key to be able to read sensitive data. If you encryption key is not available, you can generate a new encryption key, but it will wipe out all existing sensitive data. I was called in to consult on a scenario where someone had done this, and their reporting server had over 600 subscriptions added by users. They lost all of the user information related to those subscriptions. They had to track down the users and then manually re-enter their email addresses to fix the subscriptions. They also lost the passwords to all of their data sources for the reports, but that was minor to fix compared to the broken subscriptions.

So please ensure that you are exporting the encryption key to a file and storing it securely. Don’t leave it on the server itself. Move it to a secure location that gets backed up like source control.

## SSAS Databases

Yes, there are databases inside Analysis Services, and yes, you can AND SHOULD back those up. You use XMLA to back up the SSAS databases. To automate this process, create a SQL job in the regular database engine that connects to SSAS and executes the XMLA code. It’s actually very simple to do.

Here are sample backup and restore commands in XMLA for reference:

<Backup xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

 <Object>

 <DatabaseID>Adventure Works DW 2012</DatabaseID>

 </Object>

 <File>c:\bakAdventure Works DW 2012.abf</File>

</Backup>

<Restore xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

 <File>c:\bakAdventure Works DW 2012.abf</File>

 <DatabaseName>Adventure Works DW 2012</DatabaseName>

 <AllowOverwrite>true</AllowOverwrite>

 <Security>IgnoreSecurity</Security>

 <DbStorageLocation xmlns="http://schemas.microsoft.com/analysisservices/2008/engine/100/100">C:\Program FilesMicrosoft SQL ServerMSAS11.MSSQLSERVEROLAPDATA</DbStorageLocation>

</Restore>

## Summary

The key point of today’s post was to get you thinking about more than just backing up your main user databases. Yes, those are key in terms of data loss potential, but keeping everything else that is critical is also key in how long you will be down if there is an outage and key in the amount of effort involved with making your system whole again. Please make sure you are backing up the system databases, certificates, master keys, encryption keys for SSRS, and the SSAS databases. If you’re lucky, an outage might not even get noticed. If that happens, you know you’re doing your job right.

# Day 5: Dealing With Corruption in a Nonclustered Index

Welcome to day 5 of my series on disaster recovery. I want to start digging into some corruption scenarios. We’ll start off with the easiest form of corruption to fix, a nonclustered index.

The generic steps we will go through for any corruption scenario are as follows:

1. Identify the corruption (DBCC CHECKDB)
2. Identify the objects and types of objects involved
3. Take the appropriate steps to correct

Sadly, one of the biggest mistakes people make is to jump straight to the third step and start trying to fix things without even knowing what they are up against. Different objects have to be fixed in different ways. Taking the wrong action could cause unrecoverable damage and waste a lot of time. And please, please, PLEASE do not use the repair options of DBCC CHECKDB unless everything else is not possible.

## Identifying Corruption

The first thing you need to do is to identify corruption. You will probably be performing routine integrity checks or you will be responding to a specific alert or error. If you have an error message, you will have the info needed for at least the one page. You may be tempted to take action on that one page, but I advise you to take a step back and run DBCC CHECKDB on the database first. There may be additional pages corrupted that force a different plan of action.

Use DBCC CHECKDB to get the full list of errors so you can see which pages are corrupt. I like to use the No\_InfoMsgs option to reduce unnecessary chatter, the All\_ErrorMsgs option to make sure all errors are returned, and the TableResults undocumented option to output the results in a more readable format. For this demo, I will be running this on a corrupted version of the AdventureWorksDW2012 database.

DBCC CHECKDB(AdventureWorksDW2012)

 With No\_InfoMsgs, All\_ErrorMsgs, TableResults;

This returns a lot of errors for the same things. So it takes a little practice to know what to look for. You need to identify which errors are the real errors and focus on those. You will want to focus on the errors that tell you an object ID, index ID, partition ID, allocation unit ID, file, and page.



DBCC CHECKDB Output — [Click to Enlarge](https://i1.wp.com/www.sqlsoldier.com/wp/wp-content/uploads/2013/01/DBCCOutput.png)

 

MSDB Suspect Pages Output

## Identify the Corrupt Objects

After running DBCC CHECKDB, you can also check the msdb database to see the pages that were identified as being corrupted. It will have an entry for each corrupt page and give you only very base information. It does not give you the object ID or the index ID, but it does give you the database ID, file, and page. You can use this to dump the page using DBCC Page and get the additional info from there. We don’t need to do that in most situations though. We have the C CHECKDB output above. There is only one object ID reported, and I use the OBJECT\_NAME() function to get its name (FactResellerSales). I also see that the index ID is 2. I don’t need to look up the index. The index ID tells me everything I need to know.

**Index ID Mapping**

* ID 0 = heap
* ID 1 = clustered index
* ID > 1 = nonclustered index

## Take Action

Now that we know we’re dealing with just a nonclustered index, we can take the appropriate action. The easiest way to fix a corrupt nonclustered index is to drop and recreate the index. Since the underlying clustered index or heap isn’t corrupted, it can easily recreate it as the data is all still there. However, we are limited to drop and recreate. If we try to rebuild or create with drop\_existing, it will fail. Rebuild and create with drop\_existing will scan the existing index and will fail when it hits the corruption.

Now we do need the index name since we are going to be working with it directly. I query sys.indexes to get the object names.

Select Object\_Name(object\_id) As TableName, name As IndexName

From sys.indexes

Where object\_id = 341576255

and index\_id = 2

This tells me that the table name is FactResellerSales and the index name is IX\_FactResellerSales\_CurrencyKey. So just to illustrate that we have to drop and recreate it, let’s try rebuilding it first.

-- Rebuild the index??

Alter Index IX\_FactResellerSales\_CurrencyKey

 On dbo.FactResellerSales

 Rebuild;

And that gives us this error:

The statement has been terminated.

Msg 824, Level 24, State 2, Line 2

SQL Server detected a logical consistency-based I/O error: incorrect pageid (expected 1:7171; actual 0:0). It occurred during a read of page (1:7171) in database ID 8 at offset 0x00000003806000 in file 'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL13MSSQLDATAAdventureWorksDW2008R2\_Data.mdf'. Additional messages in the SQL Server error log or system event log may provide more detail. This is a severe error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.

Now let’s try using CREATE INDEX … WITH DROP\_EXISTING. For this, I will need to script out the create index statement.

-- Create with drop existing??

Create Index IX\_FactResellerSales\_CurrencyKey

 On dbo.FactResellerSales(CurrencyKey)

 With Drop\_Existing;

This yields the exact same error as the rebuild statement did. This leaves us with drop and recreate. I write the drop statement followed by the create statement and run that:

-- Drop and create?

Drop Index dbo.FactResellerSales.IX\_FactResellerSales\_CurrencyKey;

Create Index IX\_FactResellerSales\_CurrencyKey

 On dbo.FactResellerSales(CurrencyKey);

That completed successfully. We should be free of corruption now. So let’s run DBCC CHECKDB again to verify. This time CHECKDB completes successfully with 0 errors.

## Summary and Follow-up

As I showed above, some forms of corruption are actually very easy to fix. There’s no reason to start panicking when you get a corruption error message. Just follow the steps defined above and then take the appropriate action. I’ll get into some other types of corruption later in the series. In the meantime, you can download the sample corrupt database I used tonight and the code to step you through fixing it yourself.

**Sample database and demo code:** [CorruptionDemo\_AdventureWorksDW2012.zip (12.22 MB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/CorruptionDemo_AdventureWorksDW2012.zip)

# Day 6: Dealing With Corruption in Allocation Pages

Yesterday, I covered [corruption in nonclustered indexes](http://www.sqlsoldier.com/wp/sqlserver/day5of31daysofdisasterrecoverydealingwithcorruptioninanonclusteredindex), the easiest type of corruption to handle. Today, I’m going to move on to something slightly more complex, yet still really simple to manage. Today, I’m going to talk about what to do when you encounter corruption of an allocation page. Allocation pages cannot be repaired not can they be single-page restored. If you have corruption of an allocation page, you need to restore the whole database.

## What are Allocation Pages?

I’ve talked in great detail previously about allocation pages in my posts on dealing with tempdb contention [here (part 1)](http://www.sqlsoldier.com/wp/sqlserver/breakingdowntempdbcontention) and [here (part 2)](http://www.sqlsoldier.com/wp/sqlserver/breakingdowntempdbcontentionpart2). Allocation pages are special pages in the data files that track and manage extent allocations. There are 3 types of allocation pages that we are going to focus on today.

**Global Allocation Map (GAM):** Tracks which extents have been allocated. There is 1 GAM page for every 4 GB of data file. It is always page 2 in the data file and then repeats every 511,232 pages.

* Page ID = 2 or Page ID % 511232

**Shared Global Allocation Map (SGAM):** Tracks which extents are being used as mixed (shared) extents. There is 1 SGAM page for every 4 GB of data file. It is always page 3 in the data file and then repeats every 511,232 pages.

* Page ID = 3 or (Page ID – 1) % 511232

**Page Free Space (PFS):** Tracks the allocation status of each page and approximately how much free space it has. There is 1 PFS page for every 64 MB of data file. It is always page 1 in the data file and then repeats every 8,088 pages.

* Page ID = 1 or Page ID % 8088

If the page ID is 1, 2, or 3, it’s obvious that it’s one of the above allocation pages. If it’s a high number, then you have a couple of ways to figure out what type of page it is. Of course, you can do the math and calculate whether it’s an allocation page or not. I’ve included the below script to do the math for you.

Declare @PageID int;

-- Enter page number

-- e.g., 8088 = PFS page

Set @PageID = 8088;

Select Case

 When @PageID = 1 Or @PageID % 8088 = 0 Then 'Is PFS Page'

 When @PageID = 2 Or @PageID % 511232 = 0 Then 'Is GAM Page'

 When @PageID = 3 Or (@PageID - 1) % 511232 = 0 Then 'Is SGAM Page'

 Else 'Is Not PFS, GAM, or SGAM page'

 End

Another way you could determine the page type is to dump the page using DBCC PAGE and look for the m\_type value in the header output. The type values are different from the page IDs (m\_pageid in DBCC PAGE output).

|  |  |
| --- | --- |
|  **Page Type**  |  **m\_type**  |
|  PFS  |  11  |
|  GAM  |  8  |
|  SGAM  |  9  |

## Follow the Steps

Now that we know what allocation pages are and how to identify them, we can move on to following the three general steps for handling corruption.

1. Identify the corruption (DBCC CHECKDB)
2. Identify the objects and types of objects involved
3. Take the appropriate steps to correct

Step 1 is to identify the corruption. For today’s exercise, I’m going to my cleverly named sample database PFSCorruption. If I run DBCC CHECKDB on the database, I see that it is corrupted.

DBCC CHECKDB(PFSCorruption)

 With No\_InfoMsgs, All\_ErrorMsgs, TableResults;

That gives us a lot of columns that we don’t need, so I’m going to pare that down to just the really important columns. This will make it easier for us to find the bits we really want. I don’t use this when troubleshooting corruption, this is just to make it easier to find the key info for this demonstration.

Declare @DBCC Table (

 Error int,

 Level smallint,

 State tinyint,

 MessageText varchar(2500),

 RepairLevel varchar(30) null,

 Status tinyint,

 DbId int,

 DbFragId int,

 ObjectId int,

 IndexId int,

 PartitionId bigint,

 AllocUnitId bigint,

 RidDbId int,

 RidPruId int,

 [File] int,

 Page int,

 Slot int,

 RefDbId int,

 RefPruId int,

 RefFile int,

 RefPage int,

 RefSlot int,

 Allocation bigint)

Insert Into @DBCC

Exec sp\_executesql N'DBCC CHECKDB(PFSCorruption)

 With No\_InfoMsgs, All\_ErrorMsgs, TableResults;';

Select Level,

 State,

 MessageText,

 RepairLevel,

 ObjectId,

 IndexId,

 [File],

 Page,

 RefFile,

 RefPage

From @DBCC;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  **Level**  |  **State**  |  **MessageText**  |  **RepairLevel**  |  **ObjectId**  |  **IndexId**  |  **File**  |  **Page**  |  **RefFile**  |  **RefPage**  |
|  16  |  5  |  Database error: Page (1:6) is marked with the wrong type in PFS page (1:1). PFS status 0xa expected 0x44.  |  NULL  |  0  |  -1  |  1  |  6  |  1  |  1  |
|  16  |  5  |  Database error: Page (1:7) is marked with the wrong type in PFS page (1:1). PFS status 0xa expected 0x44.  |  NULL  |  0  |  -1  |  1  |  7  |  1  |  1  |
|  10  |  1  |  CHECKDB found 2 allocation errors and 0 consistency errors not associated with any single object.  |  NULL  |  0  |  0  |  0  |  0  |  0  |  0  |
|  10  |  1  |  CHECKDB found 2 allocation errors and 0 consistency errors in database ‘PFSCorruption’.  |  NULL  |  0  |  0  |  0  |  0  |  0  |  0  |

## Identify the Objects

Again, we need to identify the real errors, and those are the ones that are Level 16. The higher the level, the more severe that error is so the higher level errors are the reals one we need to examine. Also, State 5 means that an unknown error occurred that terminated DBCC execution. Each of the 2 error messages mentions 2 different pages. The error messages can be misleading and make you think that pages (1, 6) and (1, 7) are corrupted, but the corruption is page (1, 1). Page (1, 1) has the wrong values for these two pages. So the columns we are interested in this time are RefFile and RefPage rather than File and Page.

We don’t need to do the math in this case. We already know that page 1 is the PFS page and in fact, the error message even called out that it is the PFS page. We know it’s an allocation page, so we know what action is appropriate.

## Taking Appropriate Action

I said earlier that allocation pages cannot be repaired or restored. We need to do a full database restore. At this point, I would check my backup situation, back up the tail of the log, and then begin my restore process. If you don’t have backups to restore from, you will need to export the data, create a new version of the database, and re-import the data. It’s not going to be a fast process. There are a few people in the world that could hack the page with a hex editor to repair it manually, but that’s not something we are going to try to demonstrate here. If you try to do that, I highly advise you to make a copy of the files, and try to fix the copy of it.

**Sample corrupt database and scripts:** [PFSCorruption.zip (116 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/PFSCorruption.zip)

# Day 7: Writing SLAs for Disaster Recover

Today is day 7 in my series on disaster recovery. I thought I would switch gears for today and write about disaster recovery Service-Level Agreements (SLAs). Specifically, I’m talking about Recovery Point Objective (RPO) and Recovery Time Objective (RTO) SLAs.

Recovery SLAs are very rarely written first. They are usually ignored altogether or written as the last step to indicate how long they think their restore process will take. This usually ends up with unhappy business interests when a disaster occurs.

It should be the first thing you write, followed by determining the restore process required to meet those SLAs, and finally writing a backup policy to meet those restore policies. These SLAs should not just be numbers you come up with on your own. You should involve your business parties to make sure that their expectations are the same as your SLAs.

## Recovery Time Objective

Recovery Time Objective (RTO) is the maximum amount of time that is considered acceptable to be down when a disaster event occurs. RTO will dictate the restore path you need to take to restore the database. Typically, a shorter RTO means minimizing the amount of file that must be restored. Log backups restore very quickly, but when you have a high number of backups to restore, it can become a long process very quickly. If you back up the log every half hour (the maximum frequency I recommend), then taking full or differential backup once a day means that you could conceivably have to restore 48 log backups. If you back up the log every 5 minutes, that 48 becomes 288 backups. It quickly escalates into a long recovery period. You may just need to add in some additional differential backups to meet the RTO SLA.

**Recovery Time Objective (RTO):** The amount of time it is acceptable to be down while recovering from a disaster.

## Recovery Point Objective

Recovery Point Objective (RPO) is the point in which you need to recover the data after a disaster occurs. Or in simpler terms, how much data can you lose. This will dictate your recovery model and maximum frequency for the log backups. Unless you have a really high SLA for RPO, then simple recovery model is not going to be an option for you. The amount of data you are susceptible to losing is the maximum length of time between backups. If you are only doing daily full backups, then you are susceptible to losing 24 hours’ worth of data. So chances are good that you will be using full or bulk-logged recovery model in production on almost all cases. I always assume that I will be using full recovery model until someone presents justification for doing either simple or bulk-logged recovery.

An example situation where I have dealt with a nonstandard RPO is for a reporting database in which we could re-import data from source systems. We had an RPO of 4 days and could do a full import of the data (7 TB) in that time frame if we needed to. In this case, we used simple recovery model and performed weekly full backups to give us a more recent starting point and ensure we had a current backup with the schema in it.

**Recovery Point Objective (RPO):** The point in time in relation to the disaster event to which you need to recover the data.

## Working With Business

When I involve business in setting the recovery SLAs, I like to talk about a range of values as a “goal” RTO and “target” RTO. If you just ask the business interests how long you can be down and how much data you can lose, they will almost always say that they want 0 downtime and 0 data loss. That’s not realistic and attempts to approach that can be very expensive. I like to stress to the business that our goals are to try for 0 data loss and 0 downtime, but that we need them to set the realistic target values that will be deemed acceptable.

Here is an example request for the RPO and RTO SLA that I might send to business interests:

In order to determine the most effective and cost-efficient plan for meeting business requirements, we need the Recovery Point Objective (RPO) and Recovery Time Objective (RTO) defined by business. RPO and RTO are defined as ranges.

**Recovery Point Objective:** The point in time in relation to the disaster event to which we need to recover the data.

**Goal RPO:** 0 minutes
**Target RPO:** 30 minutes

**Recovery Time Objective:** The amount of time it is acceptable to be down while recovering from a disaster.

**Goal RTO:** 0 minutes
**Target RTO:** 30 minutes

In order to meet business requirements for RPO and RTO as defined above, we will implement log shipping to a remote location as our primary method of disaster recovery, and we will implement a backup and restore strategy as a secondary method of disaster recovery.

## Summary

One of my favorite sayings is that the first and last thing a DBA should do when inheriting a server is backups. If I inherit a server, the first thing I do is to put my “standard backup process” on the server and make sure that all databases have at least a current full backup. Then once other critical things are taken care of, I will revisit backups and that is when I start with generating SLAs for recovery and then move on to developing the restore plan and finally the backup plan. It’s critical that we get the SLAs in place and then build our recovery plan on that.

# Day 8: Resolutions for All DBAs

This post is not only day 7 of my 31 Days of Disaster Recovery series, it is also participating in [T-SQL Tuesday #38](http://jasonbrimhall.info/2013/01/02/t-sql-tuesday-38-standing-firm/) hosted this month by Jason Brimhall ([blog](http://jasonbrimhall.info)|[@sqlrnnr](http://twitter.com/sqlrnnr)). The theme for this month’s T-SQL Tuesday is **resolutions** or **resolving** or just being **resolute**. For my part in this monthly blogfest, I am going to list some disaster recovery resolutions that I think every DBA should make.

## Disaster Recovery Resolutions

* Ensure that every database is being backed up.
* Prioritize backups. Investigate backup failures as a top priority.
* Verify that all of your databases are using the checksum page verification option.
* Select name, page\_verify\_option\_desc

From sys.databases;

* Use the WITH CHECKSUM option for all database backups and restores
	+ See: [Use All the Checksums](http://www.sqlsoldier.com/wp/sqlserver/day9of31daysofdisasterrecoveryuseallthechecksums).
* Test your backups. Preferably, automate restoring them to a different server.
* Test your recovery plan. Anyone who might need to implement should test regularly with different scenarios.
* Create alerts and send notifications for the following errors:
	+ **823 —** Error reading page at the OS level.
	+ **824 —** Error reading page at the SQL Server level.
	+ **825 —** Error reading page but was successful on retry.
	+ **829 —** Page has been marked as restore pending.
	+ Blog post to follow explaining why.

# Day 9: Use All the Checksums

Welcome to day 9 of my 31 Days of Disaster Recovery series. Today, I want to talk about the three ways you can use CHECKSUM to protect yourself from and identify corruption. Checksum is the default page verification option in SQL Server 2005+ and helps identify corruption to data pages. The other two uses of Checksum are options for the BACKUP and RESTORE commands. When we are done here, hopefully you will be convinced to use all 3 CHECKSUM options.

## CHECKSUM Page Verification

There are 3 page verification options in SQL Server 2005 and newer. Checksum is the current default setting and the most comprehensive. Torn page detection was the default in SQL Server 2000. Torn page detection is less comprehensive because the way it works is to write a 2-bit pattern to the header of the page and then an alternating 2-bit pattern every 512 byte sector. This means it is possible for corruption to occur in a very small area of the page and not be detected. Checksum on the other hand generates a different value if even one bit of data is different. Any amount of corruption will result in a different value and the page will be flagged as corrupted.

Since the default page verification option changed, new databases are protected, but if an older database is upgraded, the page verification option is not changed. If you have really old databases, they may not be fully protected. You should check the page verification option in sys.databases and update those databases.

Here’s the tricky part. If you change the page verification option, you are not automatically protected. It doesn’t automatically rewrite every page to have checksum values when you change it. The next time a page is written to disk, the checksum value is written to the page. My recommendation when changing to checksum page verification is during the next index maintenance period, simply rebuild every index or heap (SQL Server 2008+) rather than rebuilding selectively. This will cause every table and index to be rewritten to disk and ensure that all data structures are protected by checksums.

The third page verification option is NONE. This has NEVER been the default page verification option and if you have databases using this option then it is time for someone to find a new job. This option is only set if someone changes it and whether the change was done maliciously or due to not knowing better, that person should not be touching SQL Server.

## Backup and Restore With Checksum

The final 2 ways to use checksum is with the backup and restore commands. When you use WITH CHECKSUM with the backup and restore commands, it performs extra checks. If checksums exist on a page (see previous section), it will recalculate the checksum values and ensure that they are still corruption free. This does NOT replace the need to run regular integrity checks with DBCC CHECKDB, but it gives you an extra opportunity to catch corruption. Furthermore, if you backup a database that has become corrupted without using this option, it may complete successfully with no warnings or errors. This often leads to corruption getting backed up and restored to other servers or environments without anyone realizing it. Additionally, when the backup completes, it will generate a checksum for the entire backup.

When restoring a backup using WITH CHECKSUM, it performs the same checks. The command does require that the backup was created using the checksum option, but it ensures that you don’t restore a corrupted database without being aware you are doing so. Also, if the backup became corrupt after it was created, then the restore is able to detect that and fail the restore almost immediately rather than waiting until it hits the corrupt page. If the backup file became corrupt, the checksum for the backup will be different and for a very large database, this can save you hours of time.

If you did hit corruption when restoring or backing up with the checksum option, you can complete the backup or restore using the CONTINUE\_AFTER\_ERROR option. Creating a backup of a corrupt database using this option will mark the database as being damaged and will ensure that the backup cannot be restored unless the continue-after-error option is also used for the restore. This will prevent someone from unwittingly restoring a backup of a corrupt database without realizing it.

**EDIT: recommendation from Paul Randal (**[**blog**](http://www.sqlskills.com/blogs/Paul)**|**[**@PaulRandal**](http://twitter.com/PaulRandal)**)**

A great way to check a backup for corruption is to run RESTORE VERIFYONLY … WITH CHECKSUM; to perform the same checks (check the checksum of every page with checksum values and check the checksum of the backup file) without actually restoring it.

## See it in Action

I put together a demo so you can prove to yourself the value of using all 3 checksum options. Rather than explain the demo in the blog post here, the demo code is heavily commented. It uses the two sample databases I’ve used in 2 earlier posts in this series. You may already have them downloaded, or you can download them again. I have provided the demo script and the 2 sample corrupt databases in separate zip files so you can download only what you need.

**Sample corrupt databases:** [SampleCorruptDBs.zip (12.33 MB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/SampleCorruptDBs.zip)
**Demo code:** [Demo\_BackupWithCheckSum.zip (2 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/Demo_BackupWithCheckSum.zip)

-- Check page verify setting

-- Note that both databases have Checksums enable

Select name, page\_verify\_option\_desc

From sys.databases

Where name in ('PFSCorruption', 'AdventureWorksDW2012');

-- Check to see if the databases are corrupted

-- Note that both databases are

DBCC CHECKDB(AdventureWorksDW2012)

 With No\_InfoMsgs, All\_ErrorMsgs, TableResults;

DBCC CHECKDB(PFSCorruption)

 With No\_InfoMsgs, All\_ErrorMsgs, TableResults;

-- Backup PFSCorruption with Checksum

-- Note that it suceeds even though we know it is corrupt

-- Pages have not had checksum values written yet

Backup Database PFSCorruption

 To Disk = 'C:\bakPFSCorruption\_checksum.bak'

 With Init, Checksum;

-- Backup AdventureWorksDW2012 with Checksum

-- Note that this backup fails and notifies us that it is corrupt

-- Pages have had the checksum values written to them

Backup Database AdventureWorksDW2012

 To Disk = 'C:\bakAdventureWorksDW2012\_checksum.bak'

 With Init, Checksum;

-- Backup AdventureWorksDW2012 without Checksum

-- Suceeds with no warning or errors

Backup Database AdventureWorksDW2012 To Disk = 'C:\bakAdventureWorksDW2012\_nochecksum.bak'

 With Init;

-- Use Continue\_After\_Error option to get backup of corrupt database

Backup Database AdventureWorksDW2012

 To Disk = 'C:\bakAdventureWorksDW2012\_checksum.bak'

 With Init, Checksum, Continue\_After\_Error;

-- What happens if we restore AdventureWorksDW2012\_nochecksum.bak?

-- Database restores with no errors. Another lost opportunity to catch corruption.

-- We have potentially spread corruption to different server/environment

Restore Database CorruptDB\_nochecksum

 From Disk = 'C:\bakAdventureWorksDW2012\_nochecksum.bak'

 With Move 'AdventureWorksDW2008R2\_Data' To 'c:\bakAdventureWorksDW2008R2\_Data.mdf',

 Move 'AdventureWorksDW2008R2\_Log' To 'c:\bakAdventureWorksDW2008R2\_Log.ldf';

-- Let's try it again with the checksum option

If DB\_ID('CorruptDB\_nochecksum') Is Not Null

 Drop Database CorruptDB\_nochecksum;

-- Restore fails because we cannot use checksum for restore unless

-- we used it for the backup

Restore Database CorruptDB\_nochecksum

 From Disk = 'C:\bakAdventureWorksDW2012\_nochecksum.bak'

 With Checksum,

 Move 'AdventureWorksDW2008R2\_Data' To 'c:\bakAdventureWorksDW2008R2\_Data.mdf',

 Move 'AdventureWorksDW2008R2\_Log' To 'c:\bakAdventureWorksDW2008R2\_Log.ldf';

-- What happens if we restore AdventureWorksDW2012\_checksum.bak

-- Restore fails because it was marked as a corrupt database

Restore Database CorruptDB\_checksum

 From Disk = 'C:\bakAdventureWorksDW2012\_checksum.bak'

 With Move 'AdventureWorksDW2008R2\_Data' To 'c:\bakAdventureWorksDW2008R2\_Data.mdf',

 Move 'AdventureWorksDW2008R2\_Log' To 'c:\bakAdventureWorksDW2008R2\_Log.ldf';

-- Let's try it again with the checksum option

-- Restore fails because it was marked as a corrupt database

-- Requires using the continue after error option

Restore Database CorruptDB\_checksum

 From Disk = 'C:\bakAdventureWorksDW2012\_checksum.bak'

 With Checksum,

 Move 'AdventureWorksDW2008R2\_Data' To 'c:\bakAdventureWorksDW2008R2\_Data.mdf',

 Move 'AdventureWorksDW2008R2\_Log' To 'c:\bakAdventureWorksDW2008R2\_Log.ldf';

-- Let's try it again with the checksum and Continue\_After\_Error options

-- Suceeds with a low level warning telling us that the database

-- was damaged and should be checked out

Restore Database CorruptDB\_checksum

 From Disk = 'C:\bakAdventureWorksDW2012\_checksum.bak'

 With Checksum,

 Move 'AdventureWorksDW2008R2\_Data' To 'c:\bakAdventureWorksDW2008R2\_Data.mdf',

 Move 'AdventureWorksDW2008R2\_Log' To 'c:\bakAdventureWorksDW2008R2\_Log.ldf',

 Continue\_After\_Error;

-- Cleanup extra database(s)

If DB\_ID('CorruptDB\_checksum') Is Not Null

 Drop Database CorruptDB\_checksum;

If DB\_ID('CorruptDB\_nochecksum') Is Not Null

 Drop Database CorruptDB\_nochecksum;

## Summary

You can see that the checksum options I’ve described in this post are very useful for protecting the integrity of your data and for increasing the chances that you will find corruption early as well as being the quickest way to detect a corrupt backup file. I highly encourage you to step through the demo code and see how it protects you and can save you a lot of hardship down the line.

# Day 10: Monitoring for Corruption Errors

It’s day 10 of my 31 Days of Disaster Recovery series, and I want to talk about monitoring for corruption errors. There are four errors related to corruption for which everyone should raise alerts and send notifications. The four alerts are 823, 824, 825, and 829. The sooner you identify and address corruption, the greater the chance that it can be resolved without data loss and with minimal downtime.

## 823 Errors

An 823 error is raised when a read of a page fails at the OS layer. The OS returns the failure to SQL Server who re-throws the error to the user. The attempt to read the page is retried 3 time and on the fourth failure, the error is raised. This means that when you see this error, it actually failed four times. If you see it frequently, then it has failed four times as many errors. This error is generally a sign that of problems with the storage system. It could be a disk going bad, faulty drivers, incorrectly configured disk subsystem, etc. If you see me at an event someday, ask me to share the funny story about a misconfigured SAN causing rampant file corruption on a production SQL Server (discovered before we put it into active use, fortunately).

When you receive this error, the message text will indicate the underlying cause of the error. It is important to note that not all 823 errors are an indication of corruption. For example, error 21 (Device is not ready) is often the result of the drive the file is ongoing offline or the service account losing permissions to access the drive. If the OS error returned is error 23 (Data error – cyclic redundancy check), then you definitely have a corruption of the file. In fact, OS error 23 is the only 823 error that will trigger automatic page repair in database mirroring and Availability Groups.

If you are receiving 823 errors, I highly advise having the disk system checked as well as it could be an indicator of pending hardware/disk failure.

## 824 Errors

Error 824 means that the read of the page from the file system was successful, but SQL Server detecting corruption. Just like error 823, the read is retried 3 times, and the error is raised only after the final failure. If you receive an 824 error, it has actually occurred 4 times. This error is a clear indicator of corruption and you should take corrective steps.

## 825 Errors

As I explained above, reads encountering 823 and 824 errors are retried 3 times and only raised if they fail all retries. An 825 error indicates that one of the retries was successful. Unlike 823 and 824 errors, 825 is NOT a fatal error and does NOT get bubbled up to the user. 825 errors can occur undetected by anyone. The error is logged in the SQL log and in the event log, so there are many ways you can detect these, but most people don’t think to do so. If you are getting many 825 errors, it can be a sign of impending disk failure, and the disks should be checked as soon as possible.

## 829 Errors

829 errors are lesser known errors, but they are potentially a sign of corruption caused by the disk subsystem. The error means that a page has been marked RestorePending, The only way to fix this problem is to restore the page (or file/database). It can’t be repaired using DBCC. This error, in addition to 823 (CRC failure) and 824, can trigger automatic page repair in database mirroring or Availability Groups. Ironically, when automatic page repair is triggered, it sets the page to a status of RestorePending to ensure that no other transactions can be run against it while the page repair is being performed. If you have this error occurring and it is not the result of automatic page repair, check the disk system.

## Creating Alerts

Alerts can be easily created by using the GUI (under the SQL Server Agent node in Object Explorer) or via T-SQL. First, create an Operator to receive notifications and then create the alerts. Below is an example of how to create an Operator to receive emails and alerts for the four error codes assigned to email the Operator.

USE msdb;

-- Create operator

Exec msdb.dbo.sp\_add\_operator

 @name=N'DBAs',

 @enabled=1,

 @email\_address=N'';

Go

-- Create alert

Exec msdb.dbo.sp\_add\_alert

 @name=N'Corruption (823) detected',

 @message\_id=823,

 @enabled=1;

-- Assign operator to alert

Exec msdb.dbo.sp\_add\_notification

 @alert\_name=N'Corruption (823) detected',

 @operator\_name=N'DBAs',

 @notification\_method = 1;

Go

-- Create alert

Exec msdb.dbo.sp\_add\_alert

 @name=N'Corruption (824) detected',

 @message\_id=824,

 @enabled=1;

-- Assign operator to alert

Exec msdb.dbo.sp\_add\_notification

 @alert\_name=N'Corruption (824) detected',

 @operator\_name=N'DBAs',

 @notification\_method = 1;

Go

-- Create alert

Exec msdb.dbo.sp\_add\_alert

 @name=N'Error 825 occurred',

 @message\_id=825,

 @enabled=1;

-- Assign operator to alert

Exec msdb.dbo.sp\_add\_notification

 @alert\_name=N'Error 825 occurred',

 @operator\_name=N'DBAs',

 @notification\_method = 1;

Go

-- Create alert

Exec msdb.dbo.sp\_add\_alert

 @name=N'Page RestorePending (829) detected',

 @message\_id=829,

 @enabled=1;

-- Assign operator to alert

Exec msdb.dbo.sp\_add\_notification

 @alert\_name=N'Page RestorePending (829) detected',

 @operator\_name=N'DBAs',

 @notification\_method = 1;

Go

## Summary

Hopefully I was successful in stressing the importance of raising alerts and sending notifications for these errors. It is critical that we identify and deal with corruption as soon as possible to reduce the chance that we will lose data. So create these alerts and learn how to handle them.

# Day 11: Converting LSN Formats

Welcome back to my series 31 Days of Disaster Recovery. Today is day 11, and today I want to talk about converting LSN formats. I had intended to write this blog post a long time ago, but I never seemed to get around to it. This started out as a question posted on Twitter’s [#sqlhelp](https://twitter.com/search?q=%23sqlhelp) hash tag.

Someone had read a blog post by Paul Randal ([blog](http://www.sqlskills.com/blogs/paul)|[@PaulRandal](http://twitter.com/PaulRandal)) called [Using fn\_dblog, fn\_dump\_dblog, and restoring with STOPBEFOREMARK to an LSN](http://www.sqlskills.com/blogs/paul/using-fn_dblog-fn_dump_dblog-and-restoring-with-stopbeforemark-to-an-lsn/). In this blog post, Paul explains how to take the LSN for a transaction from a log dump and use it in the RESTORE command with the STOPBEFOREMARK option. In this case, the LSN is in a different format than is required for the RESTORE command. Paul Randal explains how to convert the format, but the explanation was still a little confusing for some people. This post is an attempt to clarify how to convert the LSN format and also provide an automated way to convert it.

## Converting LSN Format

If you get the LSN from a log dump with either fn\_dblog() or DBCC LOG() or fn\_dumpdblog(), the LSN format is in a string format that consists of 3 hexadecimal numbers delimited with colons. The restore command expects the LSN to be in a large integer format (we will actually be converting it to a varchar(26) data type). The process is to take each hexadecimal string and convert it and them put them back together again with the correct number of leading zeros.

We start with this format from the log file: **::** (e.g., 00000467:00001fd8:0001) and convert it to this format . Easy peezy, lemon squeezy.

Paul Randal explained the format conversion really well in his post, but the part that was confusing some people was how to get the hexadecimal string to an integer string. With SQL Server 2008+, this is very simple to do by converting to a varbinary data type with a conversion style of 1 and then cast as integer. In order to convert it to varbinary, you first have to concatenate the string **0x** and enough zeros to the hex string to bring the length of the hex string to 8 characters. Or to put into T-SQL terms:

CAST(CONVERT(VARBINARY, '0x' +

 RIGHT(REPLICATE('0', 8) + @LSN2, 8), 1) As int);

We do this with each of the three hex strings and then concatenate them with the appropriate leading zeros. for example, the resulting integer string would be **1127000000815200001 [1127 + (000000 + 8152) + (0000 + 1)]** If we convert each hex string to variables of @LSN1, @LSN2, and @LSN3, the final concatenation would be:

CAST(@LSN1 as varchar(8)) +

 CAST(RIGHT(REPLICATE('0', 10) + @LSN2, 10) as varchar(10)) +

 CAST(RIGHT(REPLICATE('0', 5) + @LSN3, 5) as varchar(5));

## Testing the Process

First, we will create a database to perform our test in, make sure it is in simple recovery model, and then create a marked transaction:

Create Database TestLSN;

Go

-- Make sure it is simple recovery

Alter Database TestLSN Set Recovery Simple;

Go

Use TestLSN;

Go

-- Create a marked transaction

BEGIN TRAN Tran1 With MARK 'Tran 1';

Select \* Into dbo.MyDatabases

From sys.databases;

Commit

Go

We can look up the LSN for this transaction (ignoring the logmarkhistory table in the msdb database) by dumping the log file and looking for the transaction by name.

-- Find the LSN in the log

Select [Current LSN], [Previous LSN], Operation, Context, [Transaction Name]

From fn\_dblog(null, null)

Where [Transaction Name] = 'Tran1';

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  **Current LSN**  |  **Previous LSN**  |  **Operation**  |  **Context**  |  **Transaction Name**  |
|  0000001e:00000140:0004  |  00000000:00000000:0000  |  LOP\_BEGIN\_XACT  |  LCX\_NULL  |  Tran1  |

But let’s assume that you don’t find about this right away and the transaction is no longer in the active part of the log. Does that mean that we’re out of luck? Here we will take a manual checkpoint and see that the transaction is no longer listed in the log:

-- Manual Checkpoint

Checkpoint;

-- Find the LSN in the log

Select [Current LSN], [Previous LSN], Operation, Context, [Transaction Name]

From fn\_dblog(null, null)

Where [Transaction Name] = 'Tran1';

Maybe we’re not out of luck though. There is a chance it could still be in the inactive portion of the log. We can use undocumented trace flag 2537 to view the inactive portion of the log. There is a bug with this trace flag (which I’ve been told will never be fixed because, after all, it’s undocumented) which prevents it from being used to read the log from a newly restored database.

-- No, really. Find the LSN in the log

DBCC TraceOn (2537);

Select [Current LSN], [Previous LSN], Operation, Context, [Transaction Name]

From fn\_dblog(null, null)

Where [Transaction Name] = 'Tran1';

DBCC TraceOff (2537);

So we have our LSN (Current LSN is the column we want) and can convert it now. Just plug the “Current LSN” into the conversion script to get the properly formatted LSN.

-- Convert LSN from hexadecimal string to decimal string

Declare @LSN varchar(22),

 @LSN1 varchar(11),

 @LSN2 varchar(10),

 @LSN3 varchar(5),

 @NewLSN varchar(26)

-- LSN to be converted to decimal

Set @LSN = '0000001e:00000038:0001';

-- Split LSN into segments at colon

Set @LSN1 = LEFT(@LSN, 8);

Set @LSN2 = SUBSTRING(@LSN, 10, 8);

Set @LSN3 = RIGHT(@LSN, 4);

-- Convert to binary style 1 -> int

Set @LSN1 = CAST(CONVERT(VARBINARY, '0x' +

 RIGHT(REPLICATE('0', 8) + @LSN1, 8), 1) As int);

Set @LSN2 = CAST(CONVERT(VARBINARY, '0x' +

 RIGHT(REPLICATE('0', 8) + @LSN2, 8), 1) As int);

Set @LSN3 = CAST(CONVERT(VARBINARY, '0x' +

 RIGHT(REPLICATE('0', 8) + @LSN3, 8), 1) As int);

-- Add padded 0's to 2nd and 3rd string

Select CAST(@LSN1 as varchar(8)) +

 CAST(RIGHT(REPLICATE('0', 10) + @LSN2, 10) as varchar(10)) +

 CAST(RIGHT(REPLICATE('0', 5) + @LSN3, 5) as varchar(5));

This gives us the LSN in integer string format: **30000000005600001**

## Summary

Restoring data to or immediately before a specific transaction is a key tool in data recovery. If you can identify the LSN of the transaction that deleted data that shouldn’t have been deleted, you can use this process to restore the database to the last point immediately prior to the delete. More than likely, you will be restoring it to a different name and then using INSERT … SELECT queries to restore the data to the live database.

**Download the demo script:** [Demo\_ConvertLSNs.zip (1 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/Demo_ConvertLSNs.zip)

# Day 12: Extreme Disaster Recovery Training

Fittingly, today’s focus on disaster recovery as part of my 31 Days of Disaster Recovery is to announce a full day of training, [Extreme Disaster Recovery](http://sqlbits.com/information/Event11/Extreme_Data_Recovery1/TrainingDetails.aspx), being offered at [SQLBits XI](http://sqlbits.com/). This training will be the only pre-conference session being led by **two (2) Microsoft Certified Masters** (SQL Server). I will be delivering this session along with my good friend and frequent cohort Argenis Fernandez ([blog](http://www.sqlblog.com/blogs/argenis_fernandez)|[@DBArgenis](http://twitter.com/DBArgenis)) with whom I have delivered 3 pre-conference sessions already.

## [Extreme Disaster Recovery](http://sqlbits.com/information/Event11/Extreme_Data_Recovery1/TrainingDetails.aspx)

SQL Bits XI – Extreme Disaster Recovery Here’s your chance to learn data recovery from the big dogs. Two Microsoft Certified Masters (MCMs) will take your Recovery game to an all new level. Take a deep dive into SQL Server recovery and learn how to handle a wide variety of data loss and corruption scenarios. The session will cover how to be prepared for, prevent, and recover data lost due to deletion or corruption.

**Learn the following skills in this session:**

* Built-in functionality in SQL Server for preventing and detecting corruption that you may not even know about
* How to identify a specific transaction in the transaction log and recover data lost from that transaction
* Categories of corruption and how to manage recovery differently for each one


Don’t come empty handed. Bring your laptops, and we’ll practice recovering corrupt databases together.

[**Register here for SQLBits and a Pre-con**](http://sqlbits.com/information/Registration.aspx)

# Day 13: Standard Backup Scripts

Today’s post took longer to prepare than I had anticipated which is why day 13 is being published on day 14. This won’t derail the 31 Days of Disaster Recovery series, even if it runs over into February. Day 13’s topic is standard backup scripts.

I’ve said on numerous occasions that the first thing a DBA should do when they inherit a new server is to make sure it has sufficient backups on it. Then later once everything is under control, circle back around and make sure the backup plan meets the needs of the [restore plan and recovery SLAs](http://www.sqlsoldier.com/wp/sqlserver/day7of31daysofdisasterrecoverywritingslasfordisasterrecover). To facilitate that, I’ve used scripts that I wrote that I call my **Standard Backups Scripts**. Within a few seconds, I have a full backup plan put into place.

**The scripts and documentation can be downloaded as a single zip file:** [StandardBackups.zip (26 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/StandardBackups.zip)

## Standard Backups Script Details

These scripts are for creating and deploying standardized backups. This set of scripts can be used to manage full/differential backups and log backups. The scripts are robust in nature and will automatically process all databases as appropriate on the server.

The scripts can be deployed as is without any modifications. At the same time, a lot effort was put into making the scripts customizable for most scenarios. All parameters are well commented inline and in the SQL jobs that they create.

**Deploy Backups.sql**
DeployBackups.sql contains all other backup scripts together. You can download and execute this single script, and you are done deploying backups to your server. You can download the individual scripts if you prefer, but be sure to create all of the procedures before creating the jobs.

**dba\_BackupDBs.sql**
All of the below default configuration options are customizable. The default configuration for this procedure is as follows:

* Back up all online databases on the server
	+ Can specify a single database via @DBName parameter
* Make the best determination for the location of the backup files (if not passed in to procedure)
	+ Use default backup location, if exists
	+ Use location of last backup taken on server, if exists
	+ Can specify a location via the @BackupLocation parameter
* Follow a schedule of a weekly full backup with daily differential backups
	+ Use the @BackupType and @DayOfFullBackup parameters to customize this schedule
* The weekly full backup will occur on Friday night
	+ Customizable via @DayOfFullBackup parameter
* All system databases (except tempdb) will always have a full backup performed
* Will default to using default server setting for compression
	+ Customize via @UseCompression parameter
* Alerts of failures will not be sent via email
	+ Enable via @SendAlerts parameter
	+ Requires that @AlertRecipients parameter is also customized
	+ Also requires that Database Mail is enabled and configured
* There are no default recipients for the alerts
	+ Set via @AlertRecipients parameter
* The backups will be executed
	+ Use @Debug parameter to output code instead of running backups
* When performing a differential backup, if a full backup does not exist, if will skip the backup and treat the attempt as a failure
	+ Use @CreateFullIfNotExists parameter to create a full backup instead if one does not exist

**For log backups:**

To use this for log backups, you must customize at least 1 parameter, @BackupType. @BackupType = 2 will back up the log of all online databases that are not log shipping participants. Without further customization of the parameters, the procedure will perform the following:

* Back up the log of all online databases that are not participating in log shipping and are in the full or bulk-logged recovery model
	+ Can specify a single database via @DBName parameter
* Make the best determination for the location of the backup files (if not passed in to procedure)
	+ Use default backup location, if exists
	+ Use location of last backup taken on server, if exists
	+ Can specify a location via the @BackupLocation parameter
* Will default to using default server setting for compression
	+ Customize via @UseCompression parameter
* Alerts of failures will not be sent via email
	+ Enable via @SendAlerts parameter
	+ Requires that @AlertRecipients parameter is also customized
	+ Also requires that Database Mail is enabled and configured
* There are no default recipients for the alerts
	+ Set via @AlertRecipients parameter
* The backups will be executed
	+ Use @Debug parameter to output code instead of running backups
* When performing a log backup, if a full backup does not exist, if will skip the backup and treat the attempt as a failure
	+ Use @CreateFullIfNotExists parameter to create a full backup instead if one does not exist

**The customizable parameters are:**

* @DBName sysname — Database name or null for all databases
* @BackupLocation nvarchar(255) — Location where you want the backups
* @BackupType bit — 0 = Full, 1 = Differential, 2 = Log, Null = Follow daily schedule (weekly full, daily diff)
* @UseCompression tinyint — 0 = Never use compression, 1 = Always use compression, Null = Do not specify compression, allow default server setting
* @DayOfFullBackup tinyint — Only applied if @BackupType is null. Use DatePart(dw, getdate()) to determine current day’s value, Sunday = 1, Saturday = 7
* @CreateFullIfNotExist bit — If full backup doesn’t exist, create full backup instead of differential or log backup, 0 = do not create, 1 = create
* @SendAlerts bit — 0 = do not send alerts, 1 = send alerts
* @AlertRecipients varchar(500) — Email address(es) to whom email alerts should go
* @Debug bit — 0 = execute backup, 1 = output the code without executing

**dba\_DeleteDBBackups.sql**
All of the below default configuration options are customizable. The default configuration for this procedure is as follows:

* Delete all old backups that exceed the configured retention level
* Make the best determination for the location of the backup files
	+ Customizable via @BackupLocation parameter
* Delete fulldifferential backups with the file extension of “bak”
	+ Customizable via @FileExtension parameter
* Delete log backups with the file extension of “trn”
	+ Customizable via @LogFileExtension parameter
* Retain only 1 full backup and all differential and log backups created since the full backup
	+ Customizable via @Retention parameter
* The backups will be deleted
	+ Use @Debug parameter to output code instead of running backups

**The customizable parameters are:**

* @BackupLocation nvarchar(255) — Location of the backups
* @FileExtension nvarchar(3) — File extension of full/differential backups
* @LogFileExtension nvarchar(3) — File extension of log backups
* @Retention int — The number of full backups to retain. All differentials since the oldest backup will be retained as well
* @Debug bit — 0 = execute deletion of backups, 1 = output the code without executing

**BackupDatabase\_job.sql**
This script creates the jobs to execute the database backup procedures. The jobs result in the following commands being executed:

* Exec dbo.dba\_DeleteDBBackups;
	+ Daily at 1 AM
* Exec dbo.dba\_BackupDBs;
	+ Daily at 1 AM
* Exec dbo.dba\_BackupDBs @BackupType = 2;
	+ Every half hour

**The scripts and documentation can be downloaded as a single zip file:** [StandardBackups.zip (26 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/StandardBackups.zip)

# Day 14: Fixing a Corrupt Tempdb

Welcome to day 14 of my 31 Days of Disaster Recovery series. I’ve previously discussed handling corruption for [nonclustered indexes](http://www.sqlsoldier.com/wp/sqlserver/day5of31daysofdisasterrecoverydealingwithcorruptioninanonclusteredindex) and [allocation pages](http://www.sqlsoldier.com/wp/sqlserver/day6of31daysofdisasterrecoverydealingwithcorruptioninallocationpages). Today, I’m going to talk about a specific corruption that is very simple to fix; however, it does mean taking the server offline briefly. How brief depends on a few things, such as is the corruption in the log file or the data file. And if it’s in the data file, is Instant File Initialization (IFI) enabled. If it’s not, you should enable IFI now unless you have compliancy restrictions against it. Go do it now, this blog post can wait. I am, of course, talking about corruption in the tempdb database.

## Tempdb Corruption

In the early days of SQL Server 2008 RTM, I saw a lot of tempdb corruption when people started implementing Transparent Data Encryption (TDE). There was a SQL Server bug in RTM that could cause this. Yes, folks, I am admitting that corruption is sometimes caused by SQL Server … like .01% of the time. The fix for tempdb corruption is simple. Restart SQL Server. The files will be reinitialized and corruption should be gone. Many people have come to believe that if you restart SQL Server, the tempdb files will be recreated anew. This is the fault of Books Online. Books Online states that the tempdb files are recreated each time the SQL service starts. This is a myth that was proven false by Jonathan Kehayias ([blog](http://www.sqlskills.com/blogs/jonathan)|[@SQ:Poolboy](http://twitter.com/SQLPoolboy)) here: [does-the-tempdb-log-file-get-zero-initialized-at-startup.aspx](http://sqlblog.com/blogs/jonathan_kehayias/archive/2010/05/13/does-the-tempdb-log-file-get-zero-initialized-at-startup.aspx)

On some rare occasions, simply restarting SQL Server did not resolve the tempdb corruption. It’s been a long time since I encountered this scenario, and my attempts to reproduce the issue were fruitless. You are not likely to encounter an issue where simply restarting SQL Server doesn’t fix the corruption. If you encounter this issue, delete the corrupted file or files while SQL Server is shut down. As I mentioned before, if it is the log file or if IFI is not enabled, then the restart time will likely be longer than usual. Log files must always be zero initialized when created. Just be aware of how big the file is configured to be and be prepared for the startup to be delayed.

Fixing corruption doesn’t get easier than this, but you shouldn’t stop there. Fixing corruption is only part of the job. You need to investigate the root cause of the corruption. If it’s a SQL bug (probably not), then you need to make sure you have applied the fix or contact PSS to get one. You also need to investigate the disk subsystem to see if it is external corruption. Whatever the cause of corruption, you need to address it to prevent it from occurring again. A DBA’s primary job is to protect the data, and that means prevention is just as important as fixing what’s broken.

**Thanks** to Paul Randal ([blog](http://www.sqlskills.com/blogs/Paul)|[@PaulRandal](http://twitter.com/PaulRandal)) for pointing out that deleting the tempdb files are usually not needed.

# Day 15: Running DBCC CHECKTABLE in Parallel Jobs

Welcome back to my 31 Days of Disaster Recovery series. Today is day 15, and I want to answer a question I was asked a while back. Paul Randal ([blog](http://www.sqlskills.com/blogs/paul)|[@PaulRandal](http://twitter.com/PaulRandal)) wrote a blog post explaining alternative options for [checking integrity of a very large database if you are not able to run the full CHECKDB](http://www.sqlskills.com/blogs/paul/checkdb-from-every-angle-consistency-checking-options-for-a-vldb/) process, and the question was borne out of one of the recommendations by Paul. One of the tactics Paul recommends is breaking the process up over multiple nights.

The person I was talking to was planning to run DBCC CHECKALLOC and DBCC CHECKCATALOG the first night of the week and then spread out DBCC CHECKTABLE executions for all of the tables across other nights. His database has a mixture of some very big tables and lots of small tables. His question to me was can he save time by running the DBCC CHECKTABLE commands in parallel threads and process multiple tables at the same time. His thought was that while one job is running DBCC CHECKTABLE against a very large table, another job could be running it against the smaller tables.

At the time, I wasn’t sure how well multiple jobs would co-exist, but I said I would run some tests and see. The results of my findings are below.

## Holy Double DBCC, Batman!

I decided to use my AdventureWorksDW2012 database for these tests. I have a very large table that I added to the database that I use for testing quite often named dbo.FactInternetSalesBig. dbo.FactInternetSalesBig has 30,923,776 rows in it and is basically just a copy of dbo.FactInternetSales table with the data re-inserted repeatedly until it was sufficiently big enough. The original dbo.FactInternetSales table has 60,398 rows in it, so it is a fraction of the size of the big versions. Another difference between the two tables is indexes. The small table has 9 total indexes, a clustered index, 7 nonclustered indexes, and a columnstore index (remnant of a different test) whereas the big table is just a heap.

I already knew from experience that the big table is about 7 GB of data that takes a fair amount of time to load into memory. First thing I did was run DBCC CHECKTABLE on both tables to make sure they both had their data in memory. Next, I gathered run times for the DBCC CHECKTABLE runs individually for my baseline values. Then I captured the run times for running both commands run at the same time in separate query windows. I toyed around with several other combinations of factors, and the ones that seemed to make the most compelling story was playing around with limiting maximum degree of parallelism (max DOP), and creating the database snapshot ahead of time. I had expected that running the DBCC commands against the database would not be ideal as each job would create it’s own database snapshot (see day 1 of this series: [Does DBCC Automatically Use Existing Snapshot?](http://www.sqlsoldier.com/wp/sqlserver/day1of31daysofdisasterrecoverydoesdbccautomaticallyuseexistingsnapshot)).

The code for the DBCC CHECKTABLE commands are shown below. The commands were the same for running against the database snapshot (named AWSnap) except the USE statement point to AWSnap.

Use AdventureWorksDW2012;

Declare @StartTime datetime2 = getdate(),

 @TimeLapse int;

DBCC CHECKTABLE('dbo.FactInternetSalesBig') with No\_InfoMsgs;

Set @TimeLapse = DATEDIFF(ms, @StartTime, getdate());

Select @TimeLapse;

Use AdventureWorksDW2012;

Declare @StartTime datetime2 = getdate(),

 @TimeLapse int;

DBCC CHECKTABLE('dbo.FactInternetSales') with No\_InfoMsgs;

Set @TimeLapse = DATEDIFF(ms, @StartTime, getdate());

Select @TimeLapse;

One of the interesting things I noticed was that running the jobs in parallel caused a relatively small increase in run time for the large table and a large increase in run time for the small table. Initially, I noticed a lot of IOCOMPLETION and PAGEIOLATCH\_SH (shared page IO latch waits) waits on the the hidden snapshot that gets created (e.g., 13:1:1444304) and the occasional PREEMPTIVE\_OS\_FILEOPS (calling out to OS for a file operations) wait. This quickly gave way to CXPACKET (parallelism exchange event) and 1 thread waiting on WRITE\_COMPLETION. Clearly there is contention between the parallel jobs.

Altogether, I ended with 8 different sets of numbers. one thing to note is although I show a single set of numbers for each category, each test was run multiple times and the run times were averaged. Additionally, the cache was pre-warmed for both the database and the pre-existing database snapshot. Some generalizations can be made about my results based on these numbers.

* Running the jobs in parallel tends to take slightly longer than running individually
* Running the jobs with limited max DOP was faster on the database directly, but slower against the snapshot
* The best results were achieved with higher max DOP on the pre-existing snapshot

Though not shown in the numbers, I did try with various levels of max DOP. The test machine has 8 logical CPUs, and the max DOP used for the reported numbers were 0 (number of logical CPUs or 8) and 1 (parallelism disabled). 1 was chosen as the test value because that value demonstrated the best results for running the jobs in parallel against the database and degraded incrementally as max DOP is increased. Running the jobs against the pre-existing snapshot got the worst results with max DOP = 1 and improved incrementally as max DOP is increased.

|  |  |
| --- | --- |
|   |  **Run Time (ms)**  |
|   |  **FactInternetSalesBig**  |  **FactInternetSales**  |  **Run Time**  |
|  Baseline (Individually on database)  |  234274  |  3614  |  237888  |
|  Parallel on database  |  239547  |  237027  |  239547  |
|  Individually on database w/limited DOP  |  147240  |  2467  |  149707  |
|  Parallel on database w/limited DOP  |  153836  |  114123  |  153836  |
|  Individually on snapshot  |  12853  |  263  |  13116  |
|  Parallel on snapshot  |  14617  |  1647  |  14617  |
|  Individually on snapshot w/limited DOP  |  49710  |  660  |  50370  |
|  Parallel on snapshot w/limited DOP  |  50520  |  950  |  50520  |

## Summary

These tests showed that you can indeed run DBCC CHECKTABLE command in parallel jobs to fit more checks into your maintenance window. Creating the database snapshot ahead of time and running all of the checks against the snapshot directly gives the best results speed wise. And as for the best degree of relativity, I believe that your mileage may vary and recommend testing different levels of max DOP to determine the optimal settings.

As Paul notes in the comments below, you should never run DBCC commands in parallel jobs unless you are creating the snapshot ahead of time and running it directly against the snapshot. And it’s important to remember that it’s not enough just to create the snapshot. You have to run the DBCC command in the context of the snapshot. It won’t automatically use the snapshot just because one exists.

**Edit:** A couple of people have asked about the script I used to create the dbo.FactInternetSalesBig table. The script I used was for testing out columnstore indexes and is from Kalen Delaney’s ([blog](http://sqlblog.com/blogs/kalen_delaney/)|[@SQLQueen](http://twitter.com/SQLQueen)) blog: [Geek City: Build a Big Table with a Columnstore Index](http://sqlblog.com/blogs/kalen_delaney/archive/2012/04/26/Build-a-Big-Table-with-a-Columnstore-Index.aspx).

# Day 16: Disaster Recovery Gems From Around the Net

It’s day 16 of my series 31 Days of Disaster Recovery. I’ve seen a lot of great DR related posts recently. I want to do kind of a round-up of some of the other must-read disaster recovery posts and articles out there.

* [Last Time CHECKDB was Run](http://jasonbrimhall.info/2013/01/17/last-time-checkdb-was-run/) by Jason Brimhall ([blog](http://jasonbrimhall.info/)|[@SQLRNNR](http://twitter.com/SQLRNNR)). This post is important because knowledge is power. Especially if you may not be the one directly responsible for the integrity checks at your job. This is a quick way to check the last good DBCC integrity check for the database.
* [Automated Backup Tuning](http://sirsql.net/blog/2012/12/12/automated-backup-tuning) by Nic Cain ([blog](http://sirsql.net)|[@SirSQL](http://sqlsoldier.net/wp/sqlserver/twitter.com/SirSQL)). You can get some good performance boost by tweaking the backup buffers, but you can also send backup performance in the toilet by tweaking it the wrong way. And it depends heavily on many factors so there is only one reliable way to determine the best settings: trial and error. Nic has automated that process for you and it runs the backup through various settings and graphs the performance differences for you so you can easily see which settings give you the best performance.
* [Centralizing and Analyzing SQL Backup Pro Backup and Restore Data](http://www.simple-talk.com/sql/sql-tools/centralizing-and-analyzing-sql-backup-pro-backup-and-restore-data/?utm_source=twitter&utm_medium=discussion&utm_content=rlandrum_centralizingbackup&utm_campaign=sqlbackup) by Rodney Landrum ([blog](http://www.simple-talk.com/community/blogs/rodney/)|[@SQLBeat](http://twitter.com/SQLBea)). It’s absolutely imperative that a DBA know the state of the backups on his servers. If you have a large number of servers, that can be a very daunting task. This process shows one way to automate this process and easily keep on top of a large number of servers.
* [The SQL Server Instance That Will not Start](http://www.simple-talk.com/content/article.aspx?article=1722) by Gail Shaw ([blog](http://sqlinthewild.co.za/)|[@SQLInTheWild](http://twitter.com/SQLInTheWild)). This is a great article about the common problems you may face when rebooting SQL Server and it won’t start. If you haven’t already been there, you will be soon enough. Gail walks you through fixing several different common problems (which believe me is much better than reinstalling production in the heat of the moment).
* [Everything](http://www.sqlskills.com/blogs/paul) by Paul Randal ([blog](http://www.sqlskills.com/blogs/paul)|[@PaulRandal](http://twitter.com/PaulRandal)). Okay, so I’m not referencing a particular blog post here, but if a week goes by without checking Paul’s blog, chances are pretty good that you missed something important. There’s no better source for learning deep internals on disaster recovery, DBCC, etc. Read it all. And then … read it again.

# Day 17: When are Checksums Written to a Page

Today is day 17 of 31 Days of Disaster Recovery. The series has skipped a couple of days due to real life imposing itself, but we’re getting back on track by digging into the Checksum page verification option and offering up some proof that the checksum value doesn’t get written until the page is written to disk. You may also learn some cool tricks for looking at metadata information. Let’s explore.

## Checksum Page Verification

I’m going to start out by creating a new database named TestPageVerify, set page verification to NONE, and add a table with some data in it.

-- Create database for testing

Create Database TestPageVerify;

Go

-- Set page verification to none

Alter Database TestPageVerify Set Page\_Verify None;

Go

-- Switch to the database

Use TestPageVerify;

Go

-- Create a table with some data in it

Select \* Into dbo.AllDBs

From sys.databases;

Go

-- Add a primary key

Alter Table dbo.AllDBs

 Add Constraint PK\_AllDBs\_DBID primary key (database\_id);

Go

Next I will choose a column at random and use the undocumented function sys.fn\_PhysLocFormatter(%%physloc%%) to return the file ID and page ID on which the record is located. %%physloc%% is a binary representation of where the row is located and fn\_PhysLocFormatter breaks that down and returns the data as file ID, page ID, and slot ID formatted as (File:Page:Slot). After getting the results, I made a note of the location and the key value of the record that I’m going to messing with.

-- Find a random data page in the table

-- (<File>:<Page>:<Row>): (1:277:3)

-- database\_id: 4

Select Top(1) sys.fn\_PhysLocFormatter(%%physloc%%),

 database\_id

From dbo.AllDBs

Order By NewID();

Go

If I use DBCC PAGE to dump the page in its current state, we’ll see that it is not protected. The key data flags to note in the header output are m\_flagBits and m\_tornBits. You’ll notice in this case that the flag bits are set to 0x0 meaning that checksum is not enabled and written to the page and torn bits are 0 as no value has been written yet.

-- Dump the page (any dump style)

DBCC TraceOn(3604);

DBCC Page(TestPageVerify, 1, 277, 1)

-- From the header output:

-- m\_flagBits = 0x0

-- m\_tornBits = 0

Go

Excerpt From DBCC Page Output

Now, I’ll enable checksum page verification and recheck the flag bits and torn bits of the page. If you follow it up by running CHECKPOINT manually and rechecking the page, you’ll see that the values don’t change.

Alter Database TestPageVerify Set Page\_Verify CheckSum;

Go

-- Dump the page again

-- Still no change

DBCC Page(TestPageVerify, 1, 277, 1)

-- From the header output:

-- m\_flagBits = 0x0

-- m\_tornBits = 0

Go

The next step is to update our sample record so that the page is dirtied in memory. At this point, dumping the page shows that the checksum value still has not been written and the torn bits is still 0. Running a CHECKPOINT will write the page to disk and cause these values to be updated. The second dump of the page shows that flag bits has been set to 0x200 (checksum page verification is enabled and populated) and torn bits is set to a large integer value.

Begin Tran

 Update dbo.AllDBs

 Set name = name + '\_Test'

 Where database\_id = 4;

Commit

Go

-- Dump the page again

-- Still no change

DBCC Page(TestPageVerify, 1, 277, 1)

-- From the header output:

-- m\_flagBits = 0x0

-- m\_tornBits = 0

Go

CHECKPOINT;

Go

-- Dump the page again

-- Still no change

DBCC Page(TestPageVerify, 1, 277, 1)

-- From the header output:

-- m\_flagBits = 0x200

-- m\_tornBits = 655784296

Go

The next thing I want to test is to add a large, fixed-length column to the table so I can update the record and force it to be moved to a new page. I’ll then use the fn\_PhysLocFormatter function again to identify the new page where the record is located.

-- Add a 7600 fixed length column so we can force a page split

Alter Table dbo.AllDBs Add TestVal nchar(3800) null;

Go

Begin Tran

 Update dbo.AllDBs

 Set TestVal = N'Test'

 Where database\_id = 4;

Commit

Go

-- Find the same page in the table

-- (<File>:<Page>:<Slot>): (1:282:0)

Select Top(1) sys.fn\_PhysLocFormatter(%%physloc%%),

 database\_id

From dbo.AllDBs

Where database\_id = 4;

Go

Dumping the page header will show that the checksum value is not yet set as both flag bits and torn bits show 0. Running a manual CHECKPOINT and re-dumping the new page shows that the checksum value is not written to the page.

-- Dump the new page

-- No checksum info

DBCC Page(TestPageVerify, 1, 282, 1)

-- From the header output:

-- m\_flagBits = 0x0

-- m\_tornBits = 0

Go

CHECKPOINT;

Go

-- Dump the page again

-- Still no change

DBCC Page(TestPageVerify, 1, 282, 1)

-- From the header output:

-- m\_flagBits = 0x200

-- m\_tornBits = 309696659

Go

Nothing left now, but to disable the trace flag we enabled and drop the test database.

-- Disable DBCC Trace

DBCC TraceOff(3604);

-- Cleanup database

Use master;

If DB\_ID('TestPageVerify') Is Not Null

 Drop Database TestPageVerify;

Go

## Summary

Run through the demo above (also attached below) and you will see that page checksums are not written to the page until the page is written to disk. Simply setting page verification to checksum is not sufficient. My recommendation is to plan to rebuild all indexes and heaps at your next index maintenance window to ensure all data gets rewritten. Beginning with SQL Server 2008, you can rebuild a heap with the ALTER TABLE REBUILD; command.

**Demo script:** [WhenChecksumsAreWritten.zip (1 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/WhenChecksumsAreWritten.zip)

# Day 18: How to CHECKDB like a Boss

Day 18 of my 31 Days of Disaster Recovery series is drawing to a close. It’s 11:22 PM here, and I’ve been working feverishly to finish today’s post before the calendar flips over to tomorrow. This started out as sharing a simple script I use for running DBCC CHECKDB against all databases on a server, and like I tend to do, I thought of lots of things I wanted to add to it. I spent a several hours customizing my “this will be a quick blog post because the script is already written” script.

## CHECKDB Like a Boss

As I said, this script started out as a simple script to run DBCC CHECKDB against every database on a server. As I was making it an “official” script, meaning one I feel is suitably evolved for sharing, I came up with several ideas for improving it. The scripts that are attached to this blog posts consists of the T-SQL to create a table to track results of the DBCC CHECKDB run to refer back to later when you see that an error occurred and a stored procedure that you can schedule via a SQL job to run regularly.

I added a couple of parameters to the procedure as follows:

* **@DBName:** Allows you to specify a specific database to run it on. The default is NULL which will run it against all online databases.
* **@UseSnapshotIfExists:** If a database snapshot already exists, this allows you to specify whether DBCC CHECKDB should be run against the snapshot instead of the live database. If multiple snapshots exist, it will run against the most recently created snapshot. A value of 1 means it will run against the existing snapshot, and 0 (default) will ignore database snapshots.

**Example usage:**

Exec dbo.dba\_CHECKDBLikeABoss @UseSnapshotIfExists = 1;

The procedure captures the error output into the table and will raise a Severity 16 error at the end if any DBCC CHECKDB checks failed telling you which databases failed and directing you to check the logging table. The error is also raised to the Windows event log in case you use monitoring software that scans the event log for errors. This process provides several ways to catch the failure (you know, just in case), the job failing, the error raised in SQL, and the error raised in the Windows event log.

I considering writing a script to purge old data from the logging table, but if your server needs regular purging of this table, you’ve got major problems. You should be fine to simply truncate this table any time you’ve been error free for a while and know you no longer need the historical data. This table should not need frequent purging.

**Download the scripts as a zip file:** [CHECKDBLikeABoss.zip (2 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/CHECKDBLikeABoss.zip)

# Day 19: How Much Log Can a Backup Log

It’s day 19 of my 31 Days of Disaster Recovery series, and today I want to talk about how much log is in a backup file. A common misconception is that when you restore a backup, you get an exact copy of the database as it was when it was backed up. That’s mostly true, but there are exceptions to that. For example, if you restore a database to a new server, the TRUSTWORTHY property gets reset as does replication or CDC configurations and objects unless you specify to keep them as part of the restore. Another common one is that the entire transaction log is in the backup. In fact, the backup only has as much a log as is required to bring the database to a consistent state upon recovery. I’m going to demonstrate just how much log a backup can log (or contains).

## Examine the Log

For this demonstration, I’m going to be using undocumented function fn\_dump\_dblog() and undocumented trace flag 2537. The function is similar to other commands with which you may be familiar, DBCC LOG() and fn\_dblog(). You can find some unofficial documentation on fn\_dump\_dblog() on the blog of fellow Certified Master Dimitri Furman ([blog](http://blogs.msdn.com/b/dfurman/)) here: [Reading database transaction log with fn\_dump\_dblog()](http://blogs.msdn.com/b/dfurman/archive/2009/11/05/reading-database-transaction-log-with-fn-dump-dblog.aspx). Also, Paul Randal ([blog](http://www.sqlskills.com/blogs/paul)|[@PaulRandal](http://twitter.com/PaulRandal)) has blogged about using this function. I generally prefer using DBCC LOG() or fn\_dblog() because the parameters to pass in are more manageable. The really cool thing about fn\_dump\_dblog() though is that it can be used to view the transaction log inside of a backup file.

Additionally, trace flag 2537 can be used in conjunction with any of the three log reader functions mentioned above to include the inactive portion of the log file when you view it. I have been told that this trace flag used to be 2536. Not sure exactly when it changed from 2536 to 2537, so if you try this on an older version than SQL Server 2008, you may need to use trace flag 2536 instead.

For this demo, I’m going to start out by creating a new database and switching to it:

Use master;

-- Create Database

Create Database TestBackups;

Go

-- Switch to database

Use TestBackups;

Go

Next step is to run a manual CHECKPOINT to ensure that the log is clear. At this point, we should only see 2 or 3 entries (depending on your version of SQL Server) for the checkpoint operation. I’ll use the fn\_dump\_dblog() function to look at the active log and verify. On a few occasions, the checkpoint will run prior to the database logging everything it needs to do for the initial creation. If you see a lot of transactions in the log, just run this step again.

-- Clear the tran log

Checkpoint;

Go

-- Log entries should be only the checkpoint entries, 3 records

Select \*

From fn\_dump\_dblog(null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null);

Go

Next I create a table and fill it with some data inside of a marked transaction. When you name a transaction, you can find the start of the transaction in the transaction log by looking for the name in the Transaction Name column of the output. After running the named transaction, I will query the log file for the count of records in the log file (lots) and also for the entry for the named transaction (to show that it’s there).

-- Insert some data into a new table in a named transaction

Begin Tran Tran1 With MARK 'Tran 1'

Select \*

INTO dbo.MasterFiles

From sys.master\_files;

Commit

Go

-- Lots of log entries now

Select count(\*)

From fn\_dump\_dblog(null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null);

-- Including our named transaction

Select \*

From fn\_dump\_dblog(null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null)

Where [Transaction Name] = 'Tran1';

Go

Next, I will CHECKPOINT the database manually again to clear the log. Then a dump of the log will show that we are back to only 2 or 3 entries for the CHECKPOINT command. Next, I will enable trace flag 2537 and requery the log for the count of records and for the named transaction. Now that we can also see the inactive portion of the log, we see that there is still a large number of transactions in the log including the named transaction.

-- Clear the tran log again

Checkpoint;

Go

-- Log entries should be only the checkpoint entries, 3 records

Select \*

From fn\_dump\_dblog(null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null);

-- Enable trace flag 2537 to see all log entries (active and inactive)

DBCC TraceOn(2537);

-- Lots of inactive log entries

Select count(\*)

From fn\_dump\_dblog(null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null);

-- Including our named transaction

Select \*

From fn\_dump\_dblog(null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null)

Where [Transaction Name] = 'Tran1';

DBCC TraceOff(2537);

Go

Next I switch to master database, back up the database, and then use fn\_dump\_dblog() to query the log file contained inside of the backup file we just created. We see that there is only a small number of records in the backup file and our named transaction is not among them.

-- Switch to master

Use master;

-- Backup the database

Backup Database TestBackups

 To Disk = 'C:\bakTestBackups.bak'

 With Init;

Go

-- Check log entries in backup

Select count(\*)

From fn\_dump\_dblog(null, null, N'Disk', 1, N'C:\bakTestBackups.bak',

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null);

-- What about our named transaction?

Select \*

From fn\_dump\_dblog(null, null, N'Disk', 1, N'C:\bakTestBackups.bak',

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null)

Where [Transaction Name] = 'Tran1';

Go

And then finally, I drop the database because I’m done.

Use TestBackups;

Alter Database TestBackups Set Single\_User With Rollback Immediate;

Use master;

Drop Database TestBackups;

Go

## Summary

Earlier today, I was lurking around [#sqlhelp on Twitter](https://twitter.com/search?q=%23sqlhelp), and someone pointed out the blog post by Dimitri that I linked to earlier. It struck me that fn\_dump\_dblog() would be a good way to demonstrate that the log file in the backup file did not include the entire log file, just the part required to bring the database to a consistent state. Be sure to work through the demo code above or download the script below:

**Demo script in zip format:** [TranLogInBackup.zip (1 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/TranLogInBackup.zip)

# Day 20: The Case of the Backups That Wouldn't Restore

I have decided to spend day 20 of my 31 Days of Disaster Recovery series by relating a true tale from my harried past days of Production DBA Operations. This is a cautionary tale with an important moral. This is the case of the backups that wouldn’t restore.

## The Case of the Backups That Wouldn’t Restore

When I worked in operations at Microsoft, I was on a team of about 40 operations engineers managing about 80 application. Only a small portion of the engineers were DBAs, so the core set of DBAs were often called in to consult for the other engineers when they needed in-depth SQL Server knowledge for a specific problem. I was contacted one evening by one of the engineers for a critical application. It was a SharePoint application that had been hit by a newly discovered bug that caused SharePoint to corrupt all of it’s content data. To be clear, the content database itself wasn’t corrupted, but the data it contained had been sort of mangled. It wasn’t the kind of thing that could be fixed, they had to restore.

After more than 3 hours of trying to restore the database themselves, they finally called me at home to ask for my help. They told me that most of the backups were corrupted, and they would lose way too much data if they restore the newest backup that works. They were hoping that I would have some trick up my sleeve to prevent them from losing 4 days of data.

I asked them to break down for me what they had been attempting to do and where it was failing. It turns out that they were using my [Standard Backup Scripts](http://www.sqlsoldier.com/wp/sqlserver/day13of31daysofdisasterrecoverystandardbackupscripts) with the default settings. This meant they were doing weekly full backups, daily differentials, and log backups every half hour. They said the full backup restored successfully, but the 3 most recent differential backups would not restore. They could restore the 4th differential backup, but that would be losing too much data. Solution was simple, simply restore the most recent differential backup that could be restored and then restore all of the log backups from that point forward to the last known good point …. right?

I told them my plan, and they didn’t seem very enthused. Turns out that when they were trying to restore the database themselves, they decided to simplify the task in front of them by deleting a bunch of the older log backups. The log backups they had not deleted only went back two days. If there was any hope of using the log backups for the restore, we had to somehow get one of the two most recent differentials to restore. they didn’t recall the exact error they got when they tried to restore the newer differential, so I tried it myself. I tested the differentials with RESTORE VERIFYONLY with no problems. They did not appear to be corrupted so I tried to restore one of them.

This was the error I received when I tried to restore the differential backup:

Msg 3136, Level 16, State 1, Line 6

This differential backup cannot be restored because the database has not been restored to the correct earlier state.

Msg 3013, Level 16, State 1, Line 6

RESTORE DATABASE is terminating abnormally.

It was apparent that someone had created an out-of-band full backup of the database. The operations engineer for the application was insistent that no one that works on the application would do that, much less create the backup and then delete it. Sure enough, a full backup had been created during that time frame, and it was NOT in the backup folder. The database had been backed up to a share on the SharePoint server. The admin had done a full site backup through SharePoint, and he was not aware that a full SharePoint backup included a full backup of the database. I checked the share on the SharePoint server, and the full backup was still there.

I restored to a recent known good point in standby mode so the engineer could query the data and verify that it was good. They wanted to get closer to the point where the data corruption occurred so, I used the technique I outlined in my post [SQLU DBA Week – Recovering Lost Data](http://www.sqlsoldier.com/wp/sqlserver/sqludbaweekrecoveringlostdata) and recovered the log files progressively in standby mode repeatedly querying for the data to ensure we get to the most recent point. Once we found the point we felt was the best restore point, I recovered the database, and they were able to begin repopulating all of their SharePoint catalogs.

## Moral of the Story

There are several things that could be learned from this experience. After recovery had completed, and everyone had gotten a good night’s sleep, we worked on improving their processes to prevent this kind of problem again. Hopefully, this tale will help you avoid making the same mistakes. Get your recovery plans in order and take heed of the below takeaways from this experience:

* **Test your backups –** Backups are critical, but they are useless if they can’t be restored. If they had implemented some process to test restores of their backups, they would have learned several days earlier that the most recent differential backups could not be restored. They could have addressed this problem a day or two prior to the corruption occurring and been ready to restore when the corruption had been found.
* **Practice your recovery process –** One of the reasons you practice your recovery process is so that when something goes wrong, you know what to do because you’ve already practiced that scenario. If they had practiced different scenarios, they might have realized that they could have simply used the log backups to complete the restore process.
* **Don’t be afraid to ask for help –** If things aren’t going well and you need help, don’t spend more than 3 hours trying to figure it out on your own. Ask for help. It would have saved them 3 hours of time, and it would have been early evening instead of almost bed time when they finally called me. It would have been better for them and for me.
* **Know what’s going on in your environment –** Part of their problem was that they were taking weekly full site backups through SharePoint with no idea of what that actually meant. It wasn’t a fluke that they had the problem that week. They would have had the same problem no matter which week it had occurred. This is especially true if you are not the SharePoint admin or “whatever admin”, it is important that you are communicating and planning your disaster recovery together. It doesn’t work well, as this experience showed, if the pieces of the same application are each doing their own thing in terms of disaster recovery.

# Day 21: Who Deleted That Data?

Welcome back for day 21 of my 31 Days of Disaster Recovery series. Today I want to talk about trying to track down who deleted data from a table. This little investigation started out as a question on the [#sqlhelp hash tag](https://twitter.com/search?q=%23sqlhelp) on Twitter from Wayne Sheffield ([blog](http://blog.waynesheffield.com/wayne)|[@DBAWayne](http://twitter.com/DBAWayne)) whom I first met on [SQLCruise Alaska 2012](http://sqlcruise.com/).

The question was asking how to convert the page ID from fn\_dump\_db\_log() to match the integer format for page ID in DBCC IND(). Where we ended up was not even close to where we started.

If you want more info on the undocumented function fn\_dump\_dblog(), check out the following blog post: [Day 19 of 31 Days of Disaster Recovery: How Much Log Can a Backup Log](http://www.sqlsoldier.com/wp/sqlserver/day19of31daysofdisasterrecoveryhowmuchlogcanabackuplog).

## Converting Page ID

I explained that the **Page ID** column in fn\_dump\_dblog() output has file ID and page ID in hexadecimal format as **:**. It was as simple as splitting the value and then converting both pieces to integers. I recommended using the same expression to convert Page ID as I did for converting the LSN in the blog post [Day 11 of 31 Days of Disaster: Converting LSN Formats](http://www.sqlsoldier.com/wp/sqlserver/day11of31daysofdisasterconvertinglsnformats). Wayne’s plan was to search the log backups from the time frame that they believed the deletion had occurred for entries where the page ID matched one of the page IDs output by DBCC IND() for the table in question in hopes of finding a clue as to what performed the delete, a person or the system.

Wayne quickly discovered an idea better than using the page ID to find entries that matched the entire list of page IDs from DBCC IND(). He could use the AllocUnitId column to correlate it to the object ID of the table. The next trick was to get the AllocUnitId to match up to the table’s object ID. We did this by joining fn\_dump\_dblog() to sys.allocation\_units and then joining that to sys.partitions.

The process for finding these records was Wayne’s brain child. I just helped work out some details of the query. I’ll leave the demoing of his process up to him if he chooses to share it. I will share a query for finding the log entries from fn\_dump\_dblog() that correlate to a specific object.

-- Define object from which data was deleted

Declare @ObjectID int;

Set @ObjectID = OBJECT\_ID('dbo.AllDatabases');

-- Query for log file entries

Select DD.\*

From fn\_dump\_dblog(null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null,

 null, null, null, null, null, null, null, null, null, null) DD

Inner Join sys.allocation\_units AU

 On AU.allocation\_unit\_id = DD.AllocUnitId

Inner Join sys.partitions P

 On P.partition\_id = AU.container\_id

Where DD.AllocUnitId Is Not Null

And P.object\_id = @ObjectID;

Go

## Summary

In the end, Wayne was successful in tracking down the entries in the log backup file for the data deletion from the table. Sadly, the log records do not tell you who performed the action. He was able to get the session IDs (SPIDs) for the commands which indicated it was a user thread that ran the delete and not a system thread. Although you can’t find everything you want in the log, you can still find a lot of information and correlate that to certain conclusions. And I especially like the fact that you can use fn\_dump\_dblog() on a log backup rather than a live database because you can muck around in the log data without worry of affecting a live database.

# Day 22: Which DBCC CHECK Commands Update Last Known Good DBCC

The end of the day is quickly approaching as I finish this blog post. This is day 22 in my series 31 Days of Disaster Recovery, and I want to examine which DBCC CHECK commands update the last known good DBCC check that is tracked in the header of the database. To check this value, I could either dump the header page using DBCC PAGE() or I could just output he header info using DBCC DBINFO(). Both of these functions are officially undocumented, but you can find them documented unofficially all over the web. They are known to be safe commands to use; however, I still recommend that you don’t use them to muck around in production.

So the question is which DBCC CHECK commands update the value. Well, let’s find out. Trial and error is our tool of choice here.

## The Test

I executed all of the DBCC CHECK commands, except DBCC CHECKIDENT, of course, because that command isn’t used for consistency checks. After each execution, I would check the value of dbi\_dbccLastKnownGood in the database header. If the value had been updated, I dropped and recreated the database before the next test execution.

Normally, you need to enable trace flag 3604 to see the output of commands like DBCC PAGE() and DBCC DBINFO() in the query window message pane. This trace flag redirects output from the SQL log to the console. However, in this case, I’m using the WITH TABLERESULTS option. This option is documented for some DBCC commands, but it works with almost any DBCC command. If I use this option, the output automatically goes to the query window results pane, so there’s no need to use trace flag 3604. This option also makes it easy to insert into a table to query.

For the testing, I created a new database named TestDBCC, added a filegroup and file to it and then created a table with some data in it.

Use master;

-- Drop database if it exists

If DB\_ID('TestDBCC') Is Not Null

 Drop Database TestDBCC;

Go

-- Create new database

Create Database TestDBCC;

Go

-- Add a filegroup for testing DBCC CHECKFILEGROUP

Alter Database TestDBCC Add Filegroup TestFG;

Go

-- Add a file to the filegroup

Alter Database TestDBCC Add File (

 Name = N'TestFile',

 FileName = N'C:\bakTestFile.ndf')

To Filegroup TestFG;

Go

-- Switch to the database

Use TestDBCC;

Go

-- Create a table for testing DBCC CHECKTABLE

Create Table dbo.AllDBs(

 DBID int not null,

 DBName sysname not null,

 -- Create a check constraint for testing DBCC CHECKCONSTRAINTS

 Constraint ckPK Check(DBID > 0))

On TestFG;

Go

-- Add some data to the table

Select database\_id, name

From sys.databases;

Go

This is the query I used repeatedly to check the last know good value of the database I created for the test:

-- Table for DBCC results

Declare @DBInfo Table (

 ParentObject varchar(255),

 Object varchar(255),

 Field varchar(255),

 Value varchar(255))

-- Insert DBCC DBINFO into table

Insert Into @DBInfo

Exec sp\_executesql N'DBCC DBInfo(''TestDBCC'') With TableResults;';

-- Query for last known good DBCC

Select Value As dbccLastKnownGood

From @DBInfo

Where Field = 'dbi\_dbccLastKnownGood';

The results of the testing was:

|  |  |
| --- | --- |
|  **DBCC Command**  |  **Update Last Known Good DBCC?**  |
|  DBCC CHECKTABLE  |  No  |
|  DBCC CHECKCONSTRAINTS  |  No  |
|  DBCC CHECKFILEGROUP  |  Yes  |
|  DBCC CHECKALLOC  |  No  |
|  DBCC CHECKCATALOG  |  No  |
|  DBCC CHECKDB  |  Yes  |

I also tested several different options to see if any particular option affected whether the last good DBCC got updated. None of the optional settings made a difference as to whether last good DBCC got updated or not except, of course, ESTIMATEONLY because it specifically does not run any consistency checks.

# Day 23: Restoring Differential Backups With New Files

It’s day 23 of my 31 Days of Disaster Recovery series, and today’s blog post is inspired from an email i received in response to day 20’s blog post [The Case of the Backups That Wouldn’t Restore](http://www.sqlsoldier.com/wp/sqlserver/day20of31daysofdisasterrecoverythecaseofthebackupsthatwouldntrestore).

A friend shared a story with me via email about a partial restore scenario that was very complex. These are the complexities he described:

* Weekly full backups and daily differential backups
* New data files are added daily for their partitioned tables
* 1508 total files in the database
* Restore server had a different drive layout, and all files had to be moved to different locations

Additionally, he was performing partial restore of only specific filegroups, so his restore process was even more difficult than what I am going to cover today. Needless to say, he had a lot of move commands to write. The GUI in SQL Server 2012 has an option to move all files of a specific type to the same location, but we’re not going to cover the restore GUI in this blog (ever). This brought back memories of past consultations where people needed to restore a differential file where a new file had been added and the location wasn’t available on the restore server. So in today’s post is about how to restore a differential backup when files have been added.

## Restore Demo

I worked up a demo that you can walk through to see exactly how you would handle new files being added to a database between differential backups. In this scenario, we’re going to simulate a week of differential backups and add a new file each day before the differential backup. Then we are going to restore the full backup and the most recent differential backup to a new location using MOVE arguments.

**Download the scripts in a zip file:** [DifferentialRestoresWithMove.zip (2 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/DifferentialRestoresWithMove.zip)

-- Create unique locations for database files

Exec xp\_create\_subdir 'd:\TestDiffData';

Exec xp\_create\_subdir 'd:\TestDiffLogs';

Go

-- Create new database in unique location

Create Database TestDiff

 On (Name = N'TestDiff',

 FileName = N'd:\TestDiffDataTestDiff.mdf')

 Log On (Name = N'TestDiff\_log',

 FileName = N'd:\TestDiffLogsTestDiff\_log.ldf');

Go

-- Take Full Backup

Backup Database TestDiff

 To Disk = 'd:\BackupTestDiffDay1.bak'

 With Init;

Go

-- Add file for 6 (simulated) days and perform a differential backup

Declare @Cntr int = 1,

 @SQL nvarchar(200);

While @Cntr <= 6

 Begin

 Set @SQL = N'Alter Database TestDiff

 Add File (Name = ''TestDiff' +

 Cast(@Cntr as nvarchar(200)) + ''',

 FileName = ''d:\TestDiffDataTestDiff' +

 Cast(@Cntr as nvarchar(200)) + '.ndf'');';

 Exec sp\_executesql @SQL;

 Set @SQL = N'Backup Database TestDiff

 To Disk = ''d:\BackupTestDiffDay' +

 Cast(@Cntr as nvarchar(200)) + '\_diff.bak''

 With Differential, Init;';

 Exec sp\_executesql @SQL;

 Set @Cntr = @Cntr + 1;

 End

Go

-- Restore the full backup as TestDiff2 to new location

-- Create unique locations for database files

Exec xp\_create\_subdir 'd:\TestDiff2Data';

Exec xp\_create\_subdir 'd:\TestDiff2Logs';

Go

-- Restore backup moving files to new location

Restore Database TestDiff2

 From Disk = 'd:\BackupTestDiffDay1.bak'

 With Move 'TestDiff' To 'd:\TestDiff2DataTestDiff.mdf',

 Move 'TestDiff\_log' To 'd:\TestDiff2LogsTestDiff\_log.ldf',

 NoRecovery;

Go

-- Restore file list of most recent differential backup

Restore FileListOnly

 From Disk = 'd:\BackupTestDiffDay6\_Diff.bak';

Go

-- 6 new database files that need to be accounted for:

-- TestDiff1 d:\TestDiffDataTestDiff1.ndf

-- TestDiff2 d:\TestDiffDataTestDiff2.ndf

-- TestDiff3 d:\TestDiffDataTestDiff3.ndf

-- TestDiff4 d:\TestDiffDataTestDiff4.ndf

-- TestDiff5 d:\TestDiffDataTestDiff5.ndf

-- TestDiff6 d:\TestDiffDataTestDiff6.ndf

-- Restore differential backup moving files

Restore Database TestDiff2

 From Disk = 'd:\BackupTestDiffDay6\_Diff.bak'

 With Move 'TestDiff1' TO 'd:\TestDiff2DataTestDiff1.ndf',

 Move 'TestDiff2' TO 'd:\TestDiff2DataTestDiff2.ndf',

 Move 'TestDiff3' TO 'd:\TestDiff2DataTestDiff3.ndf',

 Move 'TestDiff4' TO 'd:\TestDiff2DataTestDiff4.ndf',

 Move 'TestDiff5' TO 'd:\TestDiff2DataTestDiff5.ndf',

 Move 'TestDiff6' TO 'd:\TestDiff2DataTestDiff6.ndf',

 Recovery;

Go

i also worked up a quick script to generate the MOVE commands. This could come in handy if you ever need to generate the MOVE command for a backup with a large number of files that need to be moved. This is just a quick script, not a well-evolved one, so you will need to take the output and remove the comma from the final line if you use it.

Declare @BackupFile nvarchar(500),

 @FileNumberInBackup int,

 @MoveDataFilesTo nvarchar(500),

 @MoveLogFilesTo nvarchar(500),

 @MoveFilestreamTo nvarchar(500),

 @MoveFTCatalogTo nvarchar(500),

 @RestoreCmd nvarchar(max)

Declare @FileList Table (LogicalName nvarchar(128),

 PhysicalName nvarchar(260),

 Type char(1),

 FileGroupName nvarchar(128),

 Size numeric(20,0),

 MaxSize numeric(20,0),

 FileID bigint,

 CreateLSN numeric(25,0),

 DropLSN numeric(25,0) NULL,

 UniqueID uniqueidentifier,

 ReadOnlyLSN numeric(25,0) NULL,

 ReadWriteLSN numeric(25,0) NULL,

 BackupSizeInBytes bigint,

 SourceBlockSize int,

 FileGroupID int,

 LogGroupGUID uniqueidentifier NULL,

 DifferentialBaseLSN numeric(25,0) NULL,

 DifferentialBaseGUID uniqueidentifier,

 IsReadOnly bit,

 IsPresent bit,

 TDEThumbprint varbinary(32))

-- Define backup file path/name

Set @BackupFile = N'd:\BackupTestDiffDay6\_Diff.bak';

-- Define file number of file in backup (default 1)

Set @FileNumberInBackup = 1;

-- Define destination path (not name) for all file types

Set @MoveDataFilesTo = N'd:\TestDiff2Data';

Set @MoveLogFilesTo = N'd:\TestDiff2Logs';

Set @MoveFilestreamTo = N'd:\TestDiff2FS';

Set @MoveFTCatalogTo = N'd:\TestDiff2FT';

-- Add trailing slash if not exists

If Right(@MoveDataFilesTo, 1) <> ''

 Set @MoveDataFilesTo = @MoveDataFilesTo + '';

If Right(@MoveLogFilesTo, 1) <> ''

 Set @MoveLogFilesTo = @MoveLogFilesTo + '';

-- Restore file list of most recent differential backup

Insert Into @FileList

Exec sp\_executesql N'Restore FileListOnly

 From Disk = @BackupFile

 With File = @FileNumberInBackup;',

 N'@BackupFile nvarchar(500), @FileNumberInBackup int',

 @BackupFile = @BackupFile,

 @FileNumberInBackup = @FileNumberInBackup;

Select MoveCmd = 'Move ''' + LogicalName + ''' To ''' +

 Case Type When 'D' Then @MoveDataFilesTo +

 Right(PhysicalName,

 CharIndex('', Reverse(PhysicalName)) - 1)

 When 'L' Then @MoveLogFilesTo +

 Right(PhysicalName,

 CharIndex('', Reverse(PhysicalName)) - 1)

 When 'F' Then @MoveFilestreamTo

 When 'S' Then @MoveFTCatalogTo

 Else PhysicalName

 End + ','

From @FileList

Where IsPresent = 1

Order By FileID;

Go

## Summary

It’s actually not very difficult to deal with files being added in a differential (or log) backup when you need to move the locations. You simply need to identify the files using RESTORE FILELISTONLY and then use the MOVE argument to define the new location. But it can be tedious when there are a large number of files involved. That’s where scripts like these come in handy. Enjoy.

**Download the scripts in a zip file:** [DifferentialRestoresWithMove.zip (2 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/DifferentialRestoresWithMove.zip)

# Day 24: Handling Corruption in a Clustered Index

Welcome to day 24 of my 31 Days of Disaster Recovery series. Previously, I’ve talked about several different forms of corruption: [Nonclustered Index](http://www.sqlsoldier.com/wp/sqlserver/day5of31daysofdisasterrecoverydealingwithcorruptioninanonclusteredindex), [Allocation Pages](http://www.sqlsoldier.com/wp/sqlserver/day6of31daysofdisasterrecoverydealingwithcorruptioninallocationpages), and [Tempdb](http://www.sqlsoldier.com/wp/sqlserver/day14of31daysofdisasterrecoveryfixingacorrupttempdb). These were all fairly simple to fix. Today I’m going to dive into a scenario that is a little more complex, clustered indexes.

Clustered indexes are the base data. This means we have to go to a restore scenario. Hopefully, the corruption isn’t wide spread. if it’s just 1 or a few pages, we can do page level restores. If it’s a lot of pages, it may be faster just to do a full restore. It may come down to a judgement call as to which you think is faster.

## Identify the Corruption

I’ve created a sample database for us to use for this scenario. It has a table in it with a corrupt clustered index. It also has clean backups from before the corruption occurred so we can perform a page or a full restore. In order to facilitate recovery, I had to do something we tell you to NEVER do with a corrupt database. Instead of providing you with a backup of the corrupt database, I detached the database and copied the files. To run through this demo on your own, you’re going to need to attache the files as a new database.

The scenario here is that the database was offline (let’s say the server rebooted), and corruption occurred during that time. The database is online now and seems fine. We insert some data into our table, and everything works fine. Then we query the data, and we discover the corruption.

-- Take database offline and poof, corruption occurs

-- We don't know that yet though

-- Add more data

Insert Into dbo.FactInternetSales

Select Top(500) \*

From AdventureWorksDW2012.dbo.FactInternetSales;

Go

-- Everything seems to be fine. Let's query the data

Select \*

From dbo.FactInternetSales;

Go

-- Error occurs

Msg 824, Level 24, State 2, Line 2

SQL Server detected a logical consistency-based I/O error: incorrect pageid (expected 1:298; actual 0:0). It occurred during a read of page (1:298) in database ID 8 at offset 0x00000000c14000 in file 'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL12MSSQLDATACorruptDB.mdf'. Additional messages in the SQL Server error log or system event log may provide more detail. This is a severe error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.

This is a fatal error, so our query window gets disconnected when the error pops up. We know this is a serious error. We need to determine the extent of the corruption. You may be tempted to run DBCC CHECKTABLE at this point, but the corruption could be more than just a single object. I highly recommend running the DBCC CHECKDB in almost every case.

-- Let's run DBCC CHECKDB (there may be other corrupt tables)

DBCC CHECKDB(CorruptDB) With No\_InfoMsgs, All\_ErrorMsgs, TableResults;

Go

-- Errors returned for:

-- Database ID 8 (CorruptDB)

-- Object ID 245575913 (dbo.FactInternetSales)

-- Index ID 1 (clustered index)

Okay, CHECKDB shows lots of error messages, but we need to look at the ones with a Severity of 16. It appears that just a single table and a single index is corrupted. At first look, it appears that there are 2 pages that are corrupt. Taking a closer look at the error message for page #299 shows that it is reporting that the pointer to it from page #298 is missing. Page #298 is the only one that is corrupted.

If we take a look at the corrupt page using DBCC PAGE, we can see that about half of the page was overwritten with repeating 0’s. This is a sure sign of disk corruption. You need to follow this up by having the disks checked. Disk corruption may be a sign of a failing disk so **it is important to investigate right away**.

## Decide How to Deal With It

Since it is only a single page that is corrupted, I prefer to do a single page restore here. In order to do that, I have to have certain things already in place:

* Good backup with a non-corrupted copy of the page in it. This is where having tested your backups previously really pays off. I can’t tell you how many times tables have sat corrupted for months on end before someone finds it and by then they no longer have sufficient backups to support a proper restore or they have to wait for many hours or days for another team to recover backups from tape that is in storage. TEST YOUR BACKUPS.
* Log backups to bring the page current with the rest of the database. This means that if we’re in simple recovery mode or if we don’t have all the log backups, we’re dead in the water. Our only choice in that case would be to do a full restore and save as much of the data as possible.

In our case, we see that we have a full backup and a log backup. We will need to do a tail log backup as well to really bring it current. for this restore, I’m going to put the database into restricted-user mode, take the tail log backup, and then proceed with the restore process.

## Page-level Restore

To process this restore, I’m going to put the database in restricted-user mode to kick the non-admin users out of the database. Then I will do the page-level restore including a tail log backup. I’ll restore the full backup specifying the page I want to restore. Then I will restore the existing log backup and then take a tail log backup I just took. Then I can recover the database and run DBCC CHECKDB again to ensure that the corruption is fixed. If the database is clean, we can let the users back in to the database.

-- Only the one page is corrupt, so let's do a page restore

-- Switch to master to restore the damaged page

USE master;

Go

-- Set the database in restricted user mode to keep average users out

Alter Database CorruptDB Set Restricted\_User With Rollback Immediate;

Go

-- Restore the corrupt page from the good full backup

Restore Database CorruptDB

 Page = '1:298'

 From Disk = 'C:\UsersSQLSoldierDocumentsBlogFilesCorruptDBBackupsCorruptDB.bak';

Go

-- Restore the 1st pre=existing log backup to bring the page current

-- SQL knows which transations to apply, no need to specify any special commands

Restore Log CorruptDB

 From Disk = 'C:\UsersSQLSoldierDocumentsBlogFilesCorruptDBBackupsCorruptDB.trn'

 With NoRecovery;

Go

-- If there were more pre-existing log backups, we would restore them in order

-- Now backup the tail of the log...

Backup Log CorruptDB

 To Disk = 'C:\bakCorruptDB\_LOG\_TAIL.trn'

 With init;

Go

-- Restore the tail of the log bringing the page current

Restore Log CorruptDB

 From Disk = 'C:\bakCorruptDB\_LOG\_TAIL.trn'

 With NoRecovery;

Go

-- Finally, recover the database to bring it online

Restore Database CorruptDB With Recovery;

Go

-- Recheck the database for corruption again

DBCC CHECKDB(CorruptDB) With All\_ErrorMsgs, No\_InfoMsgs, TableResults;

Go

-- Allow users back in

Alter Database CorruptDB Set Multi\_User;

Go

-- Run the original query again

Use CorruptDB;

Select \*

From dbo.FactInternetSales;

Go

## Summary

At first glance, a page-level restore may seem a little tricky, but once you’ve done it several times, it starts to become old hand. Just remember the steps I recommend taking, and follow them one at a time. It’s not as difficult as it seems.

**Download the demo scripts and sample corrupt database in zip format:** [Demo\_ClusteredIndexCorruption.zip (5.03 MB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/01/Demo_ClusteredIndexCorruption.zip)

Special thanks to Paul Randal ([blog](http://www.sqlskills.com/blogs/Paul)|[@PaulRandal](http://twitter.com/PaulRandal)) for his guidance when I saw something weird happening. Something weird was happening, but not what I thought was happening. Talking to him about it cleared it up for me and allowed me to see my error that I was overlooking.

# Day 25: Improving Performance of Backups and Restores

My series 31 Days of Disaster recovery has been on hiatus due mostly to illness. I’ve been battling a chest cold that became bronchitis. I’m still fighting cough, but even that has improved to the point that I’m now sleeping longer at night than night. It wasn’t so much that I was too sick to write a blog post as it was that I was too sick to think up good ideas and put them into words. Rather than deliver poorly thought out and poorly articulated blog posts, I opted to wait until I was better in control of my cognitive abilities.

Today we resume with Day 25 of the series and I want to discuss improving performance of backups and restores. The good news is that what works for one generally works for the other. Most actions you take to speed up backups will also speed up restores. Double bonus.

## Adjust the Backup Buffers

I mentioned this first tip earlier in the series when I posted some [DR gems from around the net](http://www.sqlsoldier.com/wp/sqlserver/day16of31daysofdisasterrecoverydisasterrecoverygemsfromaroundthenet). Nic Cain ([blog](http://sirsql.net)|[@SirSQL](http://sqlsoldier.net/wp/sqlserver/twitter.com/SirSQL)) has written a process ([Automated Backup Tuning](http://sirsql.net/blog/2012/12/12/automated-backup-tuning)) to automate determining the best buffer settings for backups. This is great because the only real way to figure out the optimal setting for any given server is through trial and error. Nic’s process automates the trial part of it and lays out the results so you can easily pick out the optimal settings.

## Skip Creating Database Files

When you run a restore, the first step of the restore process is to create the database files to the same sizes they were in the database when it was backed up. There are a couple of ways to skip the creation of some or all of the database files. For a very large database, this can save you a great deal of time in the restore process.

The first way is to make sure you have Instant File Initialization (IFI) enabled. Normally, when SQL Server creates a database file, it has to fill it with zeroes (i.e., zero it out) to mark the limits of the file. IFI is a feature that allows SQL Server to mark the limits of the files without filling them. This means that to create a 100 GB file, for example, it does not have to write 100 GBs of zeroes. The file creation process is almost instantaneous. Unfortunately, the transaction log files cannot be instantly initialized. This only allows us to skip the creation of the data files. This feature is enabled at the OS level by granting **Perform Volume Maintenance Tasks** rights to the SQL Server service account via Local Security Policy Editor or Group Policy Editor.

The other option works for data and log files. If the database files for the database you are restoring are already present, it will reuse the files that already exist. If the files exist and are the right size, then you are basically skipping the file creation process. I’ve seen people delete the existing database before starting the restore instead of restoring over the top without realizing that they are prolonging the process. This is particularly helpful when you need to restore a backup of a very large database. If I have to restore a large backup on a new server, the copy process is going to take a long time. In the meantime while it is copying, I will create an empty version of the database with the exact file specifications of the one I’m going to restore. Once the copy process finish, the creation of the database is probably finished as well and I can skip the file creation step by restoring over the database I just created.

## Multiple Files on Multiple Dedicated Drives

I can’t stress enough that this recommendation has **3 parts**. 1) Multiple files on 2) multiple 3) dedicated drives. You won’t see much if any improvement by simply having multiple files if they are on the same drive. You get 1 backup thread per LUN or mountpoint, not per file. If you write the backup to multiple files on the same drive or even different drives on the same LUN, you only get a single backup thread. No performance boost. The only benefit to this is manageability of moving around and storing smaller files or if you will be able to use multiple dedicated drives for the restore.

A common mistake I see people make is that rather than getting multiple LUNs for backups, they will put one of the backup files on the backup drive and one on a drive that holds database files. If you are trying to write a backup file to a drive that has an active database on it, it will affect performance for the activity of the database as well as the backup. I have seen many cases where this was even slower than writing to a single backup file. It is very important that the LUNs/drives be dedicated to backups only. I have seen directly proportionate improvements in backup time by adding more dedicated backup drives. I used to manage a VLDB that took 6 hours to back up onto a single drive. When we added a 2nd dedicated backup drive, back up time dropped to half, 3 hours. And when we added a 3rd dedicated backup drive, the time dropped to 2 hours.

## Other Tips

* **Use Compression:** Unless you are using Transparent Data Encryption (TDE) for the database, you should be using backup compression if it is available. Either use SQL native compression or a 3rd party compression tool. Either way, this is a real must have.
* **Crank up the SAN Throughput:** People love to talk about the speed of the drives in their SAN or the RAID type, but more often than not, I see SANs bottlenecking on throughput long before they reach the limits of the drives. Work with your SAN admin to increase the queue depth and the number of paths to the SAN. Rule of thumb: more SAN paths (multi-pathing) = more throughput.
* **Use Differential/Partial backups:** not all backups have to be full backups. Figure out a mixture of Full and Differential/Partial backups and log backups that allow you to meet your SLA for recovery time and recovery point.

# Day 26: The Mysterious Case of the Long Backup

Welcome back for day 26 of my series 31 Days of Disaster Recovery. Today I want to share a tale of a mysterious backup that was running too long, and as the SAN admin reported, nothing had changed in terms of configuration of the SAN or our LUNs. We eventually tracked down the issue, and it was something none of us had even considered. Likewise, it was something we never even thought to look for at the time we were investigating.

## Backup Performance

We had 2 databases on the server in question, the small one was 500 GB and the large one was 1.75 TB. The smaller database was basically used for authentication and only had a few hundred updates per day. As such, we rarely focused on this database very much. The main transactional database, the big one was extremely busy 24 hours a day, 7 days a week. There was no maintenance period as it handled transactions from users everywhere. Our busiest times were …. weekends. Followed next by business hours in the United States and business hours in Japan. The system was used by 30,000+ support agents around the globe. Okay, you get the picture. There was no time when it wasn’t busy.

We had highly tuned our backups. We would back up the smaller database at midnight (took less than half an hour) and the large database at 1 AM (US Pacific Time). The large database took 2 hours. We published performance metrics reports daily, and you could see that there was a small performance drop in the main database while the backup was running. From 1 AM to 3 AM, it wasn’t an issue as we had performance to spare. Over time, the backup times kept taking longer and longer. This became an issue when the backup time started taking longer than 4 hours. This put the backup completion time after 7 AM Eastern US Time which meant we were getting close to when business starting picking up again. We were still fine performance-wise in the application, but we were approaching the time that it would become a problem. Furthermore, the smaller database was now taking more than an hour to run and so it was still running when the bigger one started.

In order to maintain the size of this database, we aggressively purged data from it 4 times a day deleting support cases that were closed and had no action on them for at least 90 days. We were deleting millions of rows daily. We tracked and plotted the amount of data that we purged as well as the size of the database in our daily performance reports. There was no significant changes in either of those metrics.

We investigated the amount of activity, also in our performance reports, during the backup time frame, and no big changes there either. All performance metrics throughout the day looked completely normal. No slowness during the day, only during the backup window. We escalated it to the SAN team, and they confirmed that none of our settings on the SAN had changed and that everything looked healthy on the SAN. All SAN metrics looked good. We were on a shared SAN with many other applications, and he said that none of the others were complaining, only us.

Digging deeper, we discovered that while the backups were running, our throughput to the SAN went way down and then sometime in the 3 AM hour, throughput would return to normal. The bulk of the backup was being performed after this time. We had a theory and we needed to confirm it. We asked the SAN admin to validate the same findings on his side of the SAN.

Sure enough, the SAN was being flooded between midnight and 3 AM and bottlenecking on throughput because everyone else on the SAN was running their backups at midnight as well. We changed our backup schedule to work around this. We would back up the smaller database at 11 PM and then start the large database at 3 AM. Backup times returned to normal, and we were good again.

## Summary

When you run into performance problems with your backups, it is important to look at the usual suspects first such as disk performance, activity on the server, etc. Our investigation was made much easier by having baselines of the activity that we could compare to the current levels to determine if anything had truly changed. Ultimately though, we had to trust our findings and look outside the box. We had to look outside our system at external factors that were affecting us.

# Day 27: Restoring Part of a Database

Today is day 27 of my series 31 Days of Disaster Recovery, and I want to talk about restoring a partial database to a server. If you have a very large database with many filegroups, and you need to restore just part of the database, then you can perform a partial restore (Enterprise Edition required) to only restore minimum amount of filegroups online to get access to the part you need. A good use case for this would be if you need to restore data from backup to recover data that was accidentally deleted.

## Partial Restores

As I already stated, partial restores require Enterprise Edition. The way it works is that the database is brought online when the primary filegroup is restored. Then as each successive filegroup is brought online, those filegroups come online as well. This can be very handy for disaster recovery if you have critical data that users need right away in separate filegroups from historical data that is not critical to get online right away. If a user attempts to query any objects in filegroups that are online, the query proceeds as normal. If they attempt to query something in a filegroup that is not online, the query will fail.

For this demo, I’m going to create a new database with 2 additional filegroups. Each filegroup will contain a file which in turn contains a table. One of the filegroups will be marked as read-only and the other will remain read/write.

Use master;

Go

-- Create Database

Create Database TestPiecemealRestores;

Go

-- Make sure recovery is full

Alter Database TestPiecemealRestores Set Recovery Full;

Go

-- Add first filegroup

Alter Database TestPiecemealRestores

 Add FileGroup SecondaryFG;

Go

-- Add third filegroup

Alter Database TestPiecemealRestores

 Add FileGroup TertiaryFG;

Go

-- Add file for first table

Alter Database TestPiecemealRestores

 Add File (

 Name = N'SecondaryFile',

 FileName = N'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL12MSSQLDATATestPiecemealRestores\_Secondary.ndf')

 TO FileGroup SecondaryFG;

GO

-- Add file for second table

Alter Database TestPiecemealRestores

 Add File (

 Name = N'TertiaryFile',

 FileName = N'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL12MSSQLDATATestPiecemealRestores\_Tertiary.ndf')

 TO FileGroup TertiaryFG;

GO

-- Switch to new database

Use TestPiecemealRestores;

go

-- Create table in FG #2

Create Table dbo.SecondaryFGData (

 DataId int identity(1, 1) Not Null primary key,

 DatabaseID int not null,

 DBName sysname not null,

 FileID int not null)

 On SecondaryFG;

Go

-- Insert data into dbo.SecondaryFGData

Insert Into dbo.SecondaryFGData (DatabaseID, DBName, FileID)

Select database\_id,

 DB\_NAME(database\_id),

 file\_id

From sys.master\_files;

Go

-- Create table in FG #3

Create Table dbo.TertiaryFGData (

 DataId int identity(1, 1) Not Null primary key,

 LoginID int not null,

 LoginName sysname not null,

 LoginType char(1) not null)

 On TertiaryFG;

Go

-- Insert data into dbo.TertiaryFGData

Insert Into dbo.TertiaryFGData (LoginID, LoginName, LoginType)

Select principal\_id,

 name,

 type

From sys.server\_principals;

Go

Next, I’m going to set the third filegroup read-only, take a full backup (as best practice after setting filegroup read-only), add more data the second filegroup, and then finally, back up the read/write filegroups in the database using the **Read\_Write\_Filegroups** option for the backup command.

-- Switch to master

Use master;

Go

-- Change TertiaryFG filegroup to read-only

Alter Database TestPiecemealRestores

 Modify FileGroup TertiaryFG Read\_Only;

Go

-- Back up the full database

Backup Database TestPiecemealRestores

 To Disk = 'd:\backupTestPiecemealRestores.bak'

 With init;

Go

-- Switch back to database

Use TestPiecemealRestores;

Go

--Insert more data into dbo.SecondaryFGData

Insert Into dbo.SecondaryFGData (DatabaseID, DBName, FileID)

Select database\_id,

 DB\_NAME(database\_id),

 file\_id

From sys.master\_files;

Go

-- Back up the ReadWrite filegroups (primary and SecondaryFG)

Backup Database TestPiecemealRestores

 Read\_Write\_Filegroups

 To Disk = 'd:\backupTestPiecemealRestores\_RW.bak'

 With init;

Go

No to move on to the restores on a different instance as we have everything we need. I am going to start of by restoring just the read/write filegroups from the read/write filegroups backup that I created. For the demo, I am going to use the **Read\_Write\_Filegroups** option for the restore command even though it really is optional in this example. Since the backup I am using contains only read/write filegroups, I don’t need to tell it to restore them. With the option, SQL will restore everything in the backup file. However, if I had chosen to restore from the full backup I took first, I would have had to specify the **Read\_Write\_Filegroups** option to avoid restoring the whole database.

That’s an important fact to note. You don’t have to perform piecemeal backups in to perform a piecemeal restore. You can pick and choose what you want to restore from any given backup file.

Since my SQL instances are on the same machine, I do not to specify the MOVE option for all files being restored. The primary filegroup is always the first to be restored and must be included in the restore. Additionally, the log file will be restored as well. This means we are restoring 3 files and need to specify a new location for all 3.

Use master;

Go

-- Restore backup creating a new database. Restore

-- only primary filegroup, SecondaryFG filegroup,

-- and log file.

Restore Database TestPiecemealRestores

 Read\_Write\_Filegroups

 From Disk = 'd:\backupTestPiecemealRestores\_RW.bak'

 With Move 'TestPiecemealRestores' To 'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL13MSSQLDATATestPiecemealRestores.mdf',

 Move 'TestPiecemealRestores\_log' To 'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL13MSSQLDATATestPiecemealRestores\_log.ldf',

 Move 'SecondaryFile' To 'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL13MSSQLDATATestPiecemealRestores\_Secondary.ndf';

Go

We can use the following query to look at the filegroups and files. Since the definition for all filegroups, files, and objects are in primary, you will see the objects as included. However, and filegroups that contain files that are not online will be unavailable with a file state of **RECOVERY\_PENDING**. Querying any tables with data that is offline will fail.

-- Attempt select from first table

Select \*

From TestPiecemealRestores.dbo.SecondaryFGData;

Go

-- Attempt select from second table

Select \*

From TestPiecemealRestores.dbo.TertiaryFGData;

Go

The first query above returns data, but the second query returns an error that the table resides in a filegroup that is not online.

Msg 8653, Level 16, State 1, Line 3

The query processor is unable to produce a plan for the table or view ''TertiaryFGData'' because the table resides in a filegroup which is not online.

Now let’s say that I restored the critical filegroups for someone, and they now realize that they also need data from the read-only filegroup. Your first thought may be that you have to start over because the database was not left in a recovering mode. However, if you think back to the query where we looked at the files and filegroups, the read-only filegroup is in a Recovery\_Pending state. So, yes, we can restore that filegroup without starting over.

We don’t have a filegroup backup of the read-only filegroup, so I have to use the full backup I took first. I need to tell it to restore the filegroup by name, and I need to tell it where to put the file since it can’t go into the same spot.

-- Attempt to restore the Read-only filegroup from full backup

Restore Database TestPiecemealRestores

 Filegroup = 'TertiaryFG'

 From Disk = 'd:\backupTestPiecemealRestores.bak'

 With Move 'TertiaryFile' To 'C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL13MSSQLDATATestPiecemealRestores\_TertiaryFile.ndf';

Under normal circumstances, in order to restore the third filegroup, we would have to restore transaction log backups to bring the filegroup current with the rest of the database. However, the database has been read-only since the backup we used for the restore, and SQL is smart enough to know that there are no transactions to add to it. the moment I restored the read-only filegroup, it came online.

If we check the state of the files and filegroups now, we will see that they are all online now. and if we run the same 2 queries as before, both queries will successfully return data.

-- Check state of files in database

-- All filegroups and files should be online

-- Nothing should show as recovery pending

Select DF.name As [File Name],

 DF.type\_desc As [File Type],

 DF.state\_desc As [File State],

 DF.size As [File Size],

 DS.name As [FileGroup Name]

From TestPiecemealRestores.sys.database\_files DF

Left Join TestPiecemealRestores.sys.data\_spaces DS On DS.data\_space\_id = DF.data\_space\_id;

-- Attempt select from first table

Select \*

From TestPiecemealRestores.dbo.SecondaryFGData;

Go

-- Attempt select from second table

Select \*

From TestPiecemealRestores.dbo.TertiaryFGData;

## Summary

The reason I like to talk about talking about things like piecemeal restores, is that it gives us really good insight into how things work. Understanding how things work and why they work is key to figuring out how to do things that aren’t spelled out for you in a manual somewhere. It enables thinking out of the box and doing things that you might otherwise assume is undo-able.

**You can download the code for this demo here:** [Demo\_PiecemealRestoreRestore.zip (2 KB)](http://www.sqlsoldier.com/wp/wp-content/uploads/2013/02/Demo_PiecemealRestoreRestore.zip)

# Day 28: Recovering SQL if the Tempdb Drive Dies

Welcome to day 28 of my series 31 Days of Disaster Recovery. Today I want to talk about recovering SQL Server if your tempdb drive suddenly dies. SQL Server won’t start if it can’t start up tempdb, so it poses a challenge if you can’t get the drive up and running again.

## Recovering Tempdb

So let’s say that you’re running queries on your SQL Server one day and suddenly the query fails saying there was a transport error. You run it again, and the query window is not able to connect to the server again. You check the SQL services and you see that SQL isn’t running. Obviously, you try to start SQL running again via SQL Server Configuration Manager, and it shows that SQL Server started. You try your query again and get the same result. Next, you find the SQL error log and open it in Notepad, and you see the following errors:

Clearing tempdb database.

Error: 5123, Severity: 16, State: 1.

CREATE FILE encountered operating system error 3(The system cannot find the path specified.) while attempting to open or create the physical file 'E:\tempdbtempdb.mdf'.

Error: 17204, Severity: 16, State: 1.

FCB::Open failed: Could not open file E:\tempdbtempdb.mdf for file number 1. OS error: 3(The system cannot find the path specified.).

Error: 5120, Severity: 16, State: 101.

Unable to open the physical file "E:\tempdbtempdb.mdf". Operating system error 3: "3(The system cannot find the path specified.)".

Error: 1802, Severity: 16, State: 4.

CREATE DATABASE failed. Some file names listed could not be created. Check related errors.

Could not create tempdb. You may not have enough disk space available. Free additional disk space by deleting other files on the tempdb drive and then restart SQL Server. Check for additional errors in the event log that may indicate why the tempdb files could not be initialized.

SQL can’t create or open the tempdb files. The errors state that it can’t find the path specified. So you look at disk manager, and the drive is gone. It’s dead. Unrecoverable. Well, we’re not worried about losing anything in tempdb, we just want to get the server up and running again. One option is to bring another drive online with the same name as the old one and let tempdb recreate its files at startup. Let’s assume that’s not an option for you at this time and you’re only recourse is to change where tempdb is looking for its files. How do you do that if you can’t start it to change the file locations of SQL Server?

The solution is to start SQL Server in minimal configuration with trace flag 3608. Trace flag 3608 skips startup of all databases except master. Then connect via SQLCMD and make the changes to the tempdb file paths. Open a command window and run the following command to start SQL Server. In this example, I am starting a named instance named SQL13 so the server name format is MSSQL%. The other options are **/f** to start in minimal configuration mode and **/t3608** to start with trace flag 3608 and not startup any database other than master.

net start mssql$sql13 /f /t3608

Then I connect via SQLCMD and modify the file paths. Since I start SQL in minimal configuration mode, I will need to connect via the Dedicated Admin Connection (DAC) (-A option). Just make sure you created the required directory structure for the tempdb files. The two commands I will run (1 for each tempdb file) are:

sqlcmd -A -S SQLLaptopSQL13 -q"Alter Database tempdb Modify File (Name = 'tempdev', FileName = 'G:\tempdbtempdb.mdf');"

sqlcmd -A -S SQLLaptopSQL13 -q"Alter Database tempdb Modify File (Name = 'templog', FileName = 'G:\tempdbtemplog.ldf');"

Then just restart the service and you’re good to go.

# Day 29: Using Database Snapshots to Restore Replicated Databases in Test

For day 29 of my 31 Days of Disaster Recover series, I want to talk about restoring replicated databases from database snapshots. Someone asked me recently if I had any recommendations for speeding up their unit testing on replicated databases. He has to run unit tests over and over on replicated databases to ensure that they deploy successfully in a replicated environment. His current process was adding too much time to the process and automating it was difficult for him because he doesn’t know replication to a great depth. My recommendation was to use database snapshots.

## Restoring Replicated Databases

My friend’s current process for resetting his test replication environment was to drop replication, restore the publisher and subscriber, and then re-setup replication from scratch. This was time consuming for him, and he didn’t have the replication knowledge to automate the process. He had done some research and was considering using the **sync with backup** option for the distributor and publisher databases so that he could just restore them and they would work. He would still have to restore the subscription database or reinitialize the subscriber to get replication to work

I made the alternate suggestion of using database snapshots instead of regular backups for the restore of the replication databases. Since this is in a non-production environment, they are using Developer Edition even if they are using Standard Edition in productions. The trick is to create the snapshot of the publisher, distributor, and subscriber at the same time while there is no activity occurring and replication is in sync. Then when you need to reset the test environment, simply revert (restore) all 3 databases back to the snapshots at the same time. As far as replication knows, nothing has happened and everything is perfectly in sync. This process requires no replication knowledge to automate as it’s simply creating 3 database snapshots and then reverting the databases. And of course, both the snapshot creation and restore are sub-second processes adding less than a second to the turn-around of the test databases.

I sent him some sample code of how to do it:

Use master;

-- Create database snapshots of all 3 replication databases

-- distributor:

Create Database distribution\_snap

 On (Name = 'distribution',

 FileName = 'D:\Program FilesMicrosoft SQL ServerMSSQL11.SQL12MSSQLDatadistribution.ndf')

 As Snapshot of distribution;

Go

-- Publisher

Create Database Publisher\_snap

 On (Name = 'Publisher',

 FileName = 'D:\Program FilesMicrosoft SQL ServerMSSQL11.SQL12MSSQLDataPublisher.ndf')

 As Snapshot of Publisher;

Go

-- Subscriber

Create Database Subscriber\_snap

 On (Name = 'Subscriber',

 FileName = 'D:\Program FilesMicrosoft SQL ServerMSSQL11.SQL12MSSQLDataSubscriber.ndf')

 As Snapshot of Subscriber;

Go

-- Restore databases as revert to database snapshots

-- Need to ensure that no connections exist in databases

-- distributor:

Use distribution;

Alter Database distribution Set Single\_User With Rollback Immediate;

Use master;

Restore Database distribution

 From Database\_Snapshot = 'distribution\_snap';

Go

-- Publisher

Use Publisher;

Alter Database Publisher Set Single\_User With Rollback Immediate;

Use master;

Restore Database Publisher

 From Database\_Snapshot = 'Publisher\_snap';

Go

-- Subscriber

Use Subscriber;

Alter Database Subscriber Set Single\_User With Rollback Immediate;

Use master;

Restore Database Subscriber

 From Database\_Snapshot = 'Subscriber\_snap';

Go

It is important that the code for the revert to the database snapshot is run as a single batch. Don’t run it command by command. If you pause long enough between the point where you switch to master database after changing the database to single-user, you give time for other processes to get the single connection to the database, and the restore will be blocked. Just run the whole thing as a single batch.

# Day 30: Using Partial Availability and Initialize from Backup to Replicate a Partial Database

It’s been a tough and long road to 31 Days of Disaster Recovery. It’s been very difficult coming up with quality topic ideas for the series as we near the end. For day 30 of the series, I am combining a post on performing piecemeal restores with a post on filegroups for [T-SQL Tuesday #40](http://www.midnightdba.com/Jen/2013/03/invitation-to-t-sql-tuesday-040-file-and-filegroup-wisdom/) and a post on replication.

In case you’re not familiar with T-SQL Tuesday, let me enlighten you. This blog entry is participating in [T-SQL Tuesday #40](http://www.midnightdba.com/Jen/2013/03/invitation-to-t-sql-tuesday-040-file-and-filegroup-wisdom/), hosted this month by the Midnight DBA Jen McCown ([Blog](http://www.midnightdba.com/Jen/)|[@MidnightDBA](http://twitter.com/MidnightDBA)|[@JenniferMcCown](http://twitter.com/JenniferMcCown)). You are invited to visit his blog to join the blog party. You are welcome to write your own participating blog post for the party or just to read more blogs participating in this month’s theme: File and Filegroup Wisdom.

## Replicating a Partial Database

Let’s assume a scenario where you have a very large database (VLDB) and you want to replicate only part of the database, but you don’t want to use a snapshot to initialize the subscription and you don’t want to copy the whole database backup to the subscriber and restore it. For example, if you have a 5 TB database with mostly historical, unchanging data and you only want to replicate a small percentage of the database. For this example, let’s assume that 3 TB of data is old data and 2 TB is current data and you only want to replicate the current data. How would you go about doing it in the fastest, easiest method?

**Initialize from snapshot?** Snapshots are slow and performance impacting. There is limited support for compression in snapshots, but it uses Windows CAB compression that can be CPU intensive and has limitations such as a 2 GB file limit. Maybe your production database can take that sort of performance hit, maybe not.

**Initialize from backup?** The trouble with this option is the documented way to do this is to back up the entire database, copy the entire database over, and restore the entire database. For 5 TB of data, this can take a really long time. Additionally, the entire database gets restored and if you don’t want the extra 3 TB of data in the database, you have to delete it yourself. Even more time and trouble to set up, not to mention the performance hit on the subscriber.

From [Books Online](http://msdn.microsoft.com/en-us/library/ms151705.aspx):

A backup contains an entire database; therefore each subscription database will contain a complete copy of the publication database when it is initialized.
…
It is the responsibility of the administrator or application to remove any unwanted objects or data after the backup has been restored.

But there’s a better way. You can take the one-time hit of separating your tables to different filegroups (assuming they’re not already that way, and they’re probably not) in order to be able to set up replication in a much quicker and simpler way. Once you have the files segregated, you can use partial database availability to restore only the filegroups that contain the tables you want to replicate.

One of the caveats for doing this is that you have to backup the primary filegroup with any other filegroups you want to restore. The primary filegroup always has to be restored as it contains all of the system objects that define everything in the database, and the primary filegroups and any other filegroups must be based on the same recovery point, so they must be backed up together. You can perform a filegroup backup of filegroups by name or just the read/write filegroups which include all filegroups not marked as read-only. On the other hand, the restore can actually be performed from a full backup by specifying which filegroups to restore.

Any filegroups not restored on the subscriber will still be listed as part of the database, but the files in those filegroups will be listed as RECOVERY\_PENDING. Any attempts to query tables in those filegroups will return an error stating that the file is not online. Then it’s a matter of setting up replication subscription to initialize from backup.

You may be wondering, “**Any downsides to this process?**“. Well, partial database availability is an Enterprise Only feature. So, yeah, you must be on Enterprise Edition (or Enterprise or Datacenter Edition if on SQL 2008 R2).

## Steps to Set Up and Demo Code

I’ve done presentations on this process and written a whitepaper on it. I’m not going to repeat all of the details that I’ve presented elsewhere. I recommend downloading my whitepaper [Initializing Replication from Backup](https://www.idera.com/Action/RegisterWP.aspx?WPID=51) and [my slide deck and demo code](http://www.sqlsoldier.com/wp/sqlserver/session-files-from-sql-saturday-107-houston) from SQL Saturday #107 in Houston, TX.

**Whitepaper:** [Initializing Replication from Backup](https://www.idera.com/Action/RegisterWP.aspx?WPID=51)
**Replication Magic: Initializing from Backup:** [ReplicationMagic.zip (5.04 MB)](http://www.sqlsoldier.com/wp/sqlserver/session-files-from-sql-saturday-107-houston)

# Day 31: Backup and Restore of the Resource Database

It has been a long journey to the final day my 31 Days of Disaster Recovery series, but we have finally reached the final post, day 31. My final topic for the series is born out of a conversation I had today with my good friend and fellow DBA, Ed Watson ([blog](http://sqlgator.com)|[@SQLGator](http://twitter.com/SQLGator)). Today, I want to talk about backing up and restoring the resource database.

The resource database stores critical system objects safely separated away from the master database. It is critical in operation of the system and the master database is fairly useless without it. If you lose the resource database, the system cannot start the master database and SQL Server will not being able to start. So we need to back it up and be prepared to restore it if need be.

## Back Up the Resource Database

SQL Server service packs (SPs), cumulative updates (CUs), hotfixes, etc., may or may not upgrade the resource database to a new version. It is highly critical that we back up the resource database after installing an SP, CU, hotfix, etc. to ensure that we have a current version. In some scenarios, you may be able to restore a slightly older version of the resource database and then reapply all patches and updates to bring it current. In some cases, you may not. The safest bet is to always make sure you have a current copy of the database.

If you thought the resource database was backed up with the master database, it is not. You must back it up separately. You can’t, however, back it up using native SQL Server backup. My recommended process is to simply copy the resource database files to the backup directory using simple file copy (robocopy, xcopy, copy-item, etc). Then I am sure that whatever process saves my backups to tape or alternative storage is saving backups of the resource database as well.

Below are a couple examples of how I back up the resource database as part of an automated job. Each example copies the files to the backup folder and adds a data string in the format \_YYYYMMDD to the end of the filename.

**Windows script (DOS):**

|  |  |
| --- | --- |
|  | for /f "tokens=1\* delims= " %%a in ('date/t') do set cdate=%%bfor /f "tokens=1,2 eol=/ delims=/ " %%a in ('date/t') do set mm=%%bfor /f "tokens=1,2 delims=/ eol=/" %%a in ('echo %cdate%') do set dd=%%bfor /f "tokens=2,3 delims=/ " %%a in ('echo %cdate%') do set yyyy=%%bset date=%yyyy%%mm%%dd%xcopy "C:\Program filesMicrosoft SQL ServerMSSQL11.SQL13MSSQLBinnmssqlsystemresource.\*" D:\BackupSQL13mssqlsystemresourcemssqlsystemresource\_%date%.\* /J /Q |

**PowerShell:**

|  |  |
| --- | --- |
|  | $date = "\_" + (get-date -format yyyyMMdd) + "."copy-item -path "C:\Program filesMicrosoft SQL ServerMSSQL11.SQL13MSSQLBinnmssqlsystemresource.\*" -destination D:\BackupSQL13mssqlsystemresource -force -passthru | rename-item -newname { $\_.name -replace '.',$date} |

## Restore the Resource Database

The process of restoring the resource database is just as straightforward as backing it up. If you determine that your resource database is corrupted, fixing it is as simple as copying a good copy of the resource database files in place of the old ones. If corruption is detected, SQL Server will not be able to start and you will find a message in the application event file similar to the following:

SQL Server detected a logical consistency-based I/O error: invalid protection option. It occurred during a read of page (1:3289) in database ID 32767 at offset 0x000000019b2000 in file ‘C:\Program FilesMicrosoft SQL ServerMSSQL11.SQL13MSSQLBinnmssqlsystemresource.mdf’. Additional messages in the SQL Server error log or system event log may provide more detail. This is a severe error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.

If you are wondering, yes, I intentionally corrupted my resource database file to be able to demonstrate this problem. Since SQL Server is offline, I copy a good backup copy of the files in place of the old ones, and then SQL is able to restart successfully. Easy, peezy, lemon-squeezy.

But what happens if you don’t have any backups of the resource database files? Well, you can always rebuild the master database (see Books Online for the process) which means all system databases get rebuilt and you will have to recreate server objects (logins, linked servers, SQL jobs, etc) and settings (server configurations, changes to the model database, replication settings). In short, rebuilding the master database can be a colossal pain.

So be safe and always have backups!