SQL Server Backup Questions We Were Too Shy to Ask

Grant Fritchey

simple talk

redgate

Grant Fritchey, SQL Server MVP, works for Red Gate Software as Product Evangelist. In his time as a DBA and developer, he has worked at three failed dot–coms, a major consulting company, a global bank and an international insurance & engineering company. Grant volunteers for the Professional Association of SQL Server Users (PASS). He is the author of the books SQL Server Execution Plans (Simple-Talk) and SQL Server 2008 Query Performance Tuning Distilled (Apress). He is one of the founding officers of the Southern New England SQL Server Users Group (SNESSUG) and its current president. He's earned the nickname "The Scary DBA". He even has an official name plate, and displays it proudly.

Contents

Instead of writing yet another article about how you should do your backups, or why you should do your backups, or even why you should test your backups (and you really should test your backups), we've decided that we're just going to tackle some of the harder, more interesting questions that we've heard asked about SQL Server backup and restore. These questions, collected up over time, are...

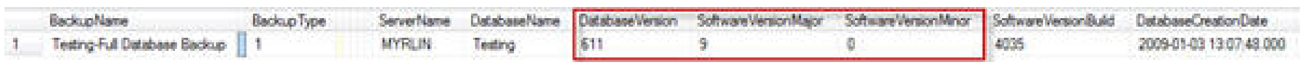
Can I restore a backup onto a different version of SQL Server? What snags might I hit?

You can restore to a different version of SQL Server, but you can only restore upwards. In other words, you can restore from 2000 to 2005 or from 2005 to 2008R2 or from 2008 to 2012, but you can never restore in the reverse direction. Each version of SQL Server makes modifications to the binary of the database and its storage. Microsoft doesn't go back in time and rewrite the previous versions to support these changes. If you really need to restore downwards, you might need to try scripting out the schema and data (see, for example, Downgrading from SQL 2008 to 2005 by Jonathan Kehayias).

In order to know from which version of SQL Server a backup originated, you need to look at the header of the backup file:

```
RESTORE HEADERONLY FROM DISK = 'd:\bu\mm.bak';
```

The information returned includes details of the Major, Minor and Build versions of the SQL Server instance for that backup, as shown in the screenshot below. This allows you to validate that you are attempting to restore an appropriate version.



	BackupName	BackupType	ServerName	DatabaseName	DatabaseVersion	SoftwareVersionMajor	SoftwareVersionMinor	SoftwareVersionBuild	DatabaseCreationDate
1	Testing-Full Database Backup	1	MYRLIN	Testing	611	9	0	4035	2009-01-03 13:07:48.000

When restoring a database to a newer version of SQL Server, it is possible to hit incompatibilities within the database. The safest approach is to run Microsoft's Upgrade Advisor (a free tool available for each version of SQL Server) on the database you wish to migrate, in order to ensure that it's ready, and then take a backup and restore it to the new instance (don't do it the other way round i.e. attempt the restore then run the advisor).

After the restore, the database will be running in the compatibility mode that it had prior to the upgrade. This means that it will be set to support the functionality of the version of SQL Server from which you migrated. In order to take full advantage of the new version of SQL Server, you will need to modify the compatibility level of your database. You can do this through the GUI, or by running this T-SQL:

```
ALTER DATABASE MyDB SET COMPATIBILITY_LEVEL = 110;
```

The different numbers represent different versions of SQL Server: 90 for 2005, 100 for 2008 and 2008R2 and 110 for 2012.

Finally, not all upgrades are possible. Microsoft only allows you to jump forward a maximum of two versions. For example, you cannot restore a SQL Server 2000 database backup to a 2012 instance. You would first need to restore it to 2008, update the compatibility level, perform a backup and then restore that backup to 2012.

Can I use a restore to copy a database? What could go wrong?

Yes, you can use a restore to copy a database. If you are restoring to a new server, you just need to make sure that your drives are the same on that server or that you map the files of the database to the appropriate drives on the new server using the WITH MOVE option of the RESTORE DATABASE command:

```
RESTORE DATABASE NewDBName
FROM  DISK = ‘c:\bu\mm.bak’
WITH  MOVE ‘OldDB’ TO ‘c:\data\new_mm.mdf’,
MOVE ‘OldDB_Log’ TO ‘c:\data\new_mm_log.ldf’;
```

Database files have a logical name and a physical file name. You just have to identify the logical name of each file and assign it to a new location.

The main issues you might encounter are in the form of errors arising from insufficient space for the restored database in the new location, or because you forgot to supply a new name and attempted to restore over the top of your existing database.

When restoring the database on a different instance of SQL Server you might hit problems with 'Orphaned Users', if the database user is mapped to a SID that is not present in the new server instance. You would need to fix this.

Why can you mount a database from the MDF file even if you don't have the transaction log?

The only time that you can do this is when the DB was shut down cleanly before the log file was lost. It's still not a good idea. You need the transaction log file as well as the data files because SQL Server references the log file when attaching the DB - it's the crash recovery process. While attaching a data file without the log file may be possible in some circumstances, it is not a recommended approach and is only for cases when the log file has been damaged or lost due to hardware problems and there are no backups available. Of course, you cannot have a database without a log file, so SQL Server just recreates a log file when you attach the database.

Attaching a data file without the log file breaks the log chain and may render the database transactionally or structurally inconsistent depending on the state of the database before the log file was lost. It may also result in the database failing to attach at all, no matter what steps are taken.

You can detach the data and log files for a database but you need to ensure that these log files either have been detached from a server, or copied after shutting down the server. This ensures that all transactions for that database are complete.move them to another server and put them on that server in a process that is much faster than a restore, although not necessarily as safe, since you have to copy or move your files. Note again that you can only move upwards, not downwards in SQL Server versions.

My database is on a SAN. I'm told that the SAN backups are sufficient. Is this true?

It might be true. The critical point is that your SAN must support transactions within the SQL Server system. If so, then it will be aware of the fact that SQL Server databases have transactions and that these transactions mean that the data in the data files may be incomplete because the process of writing the transaction isn't complete at the point that the backup is taken. SQL Server native backups, of course, always consider this!

Data Domain from EMC, a combination of software and a SAN, certainly offer transactional support, as do other vendors, but check your SAN documentation. You're looking for phrases like 'transaction consistency' or 'transaction aware' or similar. If you don't see them, then I strongly suggest you test a restore of a database before you assume that the SAN backups alone will support your backup requirements. Even if you do see them, don't automatically assume you don't need to take SQL Server database backups. If you need to support point-in-time restores, for example, then you will need to be taking database and log backups.

A SAN backup solution that does support SQL Server will hook into the SQL Server VDI interface and quiesce the database prior to taking the backup. If you were to run such a backup and then look in the SQL Server error log, there will be messages stating that the IO was frozen on the database.

If you are relying on SAN backup, you're still going to have to check for database consistency, either by running DBCC checks on the live database, or by restoring a database from a SAN backup and running the checks on that. Otherwise, you may just be backing up a corrupted database.

Why can't I just backup SQL Server's data files with my Windows backup tool? I don't need up-to-the-minute backups.

SQL Server is not like a word processing application. It manages its own files internally in order to guarantee the ACID (Atomic, Consistent, Isolated, Durable) properties of its databases. In a nutshell, to ensure that a transaction completes successfully, SQL Server maintains very tight access control over its files and it modifies these files as it sees fit.

If you simply copy the data file, ignoring the locks and ignoring the transactions that may be currently in progress, it means that when you attempt to attach that database later you will have a database file that is in an inconsistent state. It will generate errors.

Only if you are in a situation where the database is completely unchanging, ever, could you reliably copy the file and attach it later. However, if there is even a remote possibility of a transaction being open on the system when you copy the file, you're extremely likely to have an unsuccessful backup. The only safe way to ensure no transactions are running when you run your Windows backup tool would be to take the database offline first.

It's is much safer and easier to use SQL Server backups in order to ensure that you have a safe copy of your database that respects the ACID properties of the transactions within it. It is much easier to do a normal database backup.

It is only a small database. Why can't I just write every table to disk to back it up?

Yes, you can use a tool like SQLCMD to write the tables to a flat file but then, instead of a straightforward, single statement to restore the database, you will need a whole series of commands. First, you'll need to create an empty database. Then, you'll need a process to create and load each table. If any of the tables has an IDENTITY column, you'll need to SET IDENTITY_ INSERT on for each of those tables. You'll also have to worry about the order in which you load data into the tables, to take into account any relational integrity between them.

Even then, you'll have written each table to disk at a separate point in time, so if any data modifications were in progress at the time, the data wouldn't necessarily be consistent between related tables. If it isn't then you'll have to clean the data by hand afterwards before you can enable constraints.

You absolutely can do all that work. Alternatively, you can just run a backup and then, when needed, a restore, which is the much smarter approach.

SQL Server has native backup. Why pay money for a tool to do it?

There are three primary reasons why you might want to purchase a tool to help with your backups: guidance, automation and utility. If you are just getting started as a DBA, or, you're not a DBA at all but you're managing a database system as a secondary part of your job, you may not know a lot about how, where and why to set up SQL Server backups. A good third party product (e.g SQL Backup Pro) will provide you with exactly the kind of guidance you need to help you get your databases protected through the backup process.

SQL Server backups work extremely well, but you're going to have to do some work to get them set up and even more to get them automated. A good third party product will make this automation process very simple. Further, you can get automation of other processes related to backups such as mirroring/log shipping and backup verification.

Finally, while native SQL Server backups do what you need, they may not be doing so in a "best of breed" fashion. For example, some tools are more efficient at compressing backups, saving even more disk space and time during your backup processes. They also add functionality such as encryption of the backup files, something only possible natively through SQL Server if you've also encrypted the database.

If I leave a backup on a network share, surely nobody can read it?

Unless you encrypt the backup file directly, then yes, that is a readable file. If someone got access to the network share, he or she could read the backup file directly using a text editor, or simply copy the file and run a restore on another instance of SQL Server.

It's even possible to extract the schema code or the data from the backup file without restoring it. If you have SQL Data Compare, then the /Export command line parameter wil allow you to read all the data out of the backup, in CSV format, by comparing the backup to a blank database, without any password being required. Likewise, a similar process with SQL Compare will allow you to create a complete script to recreate the database schema.

You must take steps to prevent unauthorized access of a backup file. First off, make sure the network location where you're storing backups is only available to a select group of people. Next, only keep the backups there that you actually need. Don't keep extra copies of a backup locally unless you have to. Finally, if you're using a third party tool (e.g SQL Backup Pro), you can encrypt the backup to ensure that even if someone inappropriate has access to the file, he or she won't be able to read it.

With native backups, you'd have to first encrypt the actual data and log files, using Transparent Data Encryption (TDE).

The very best solution for security involves defense in depth, meaning, do all of the above.

Can someone change the contents of a backup?

There is no direct way to modify the contents of a backup file. Since the backup is a page-by-page copy of the database, as it existed the moment you took the backup, a restored copy of that database will be in exactly the same state as it was the moment you took the backup.

When SQL Server reads each page during the restore, it will calculate a checksum, based on its contents, and compare it to the value when it read the page during the backup (assuming you enabled checksums during backup). If someone did manage to change the data within the backup, these values won't match and SQL Server will flag the page as corrupt.

If I've set the verify flag when doing a backup, surely it can always be restored.

If by the verify flag, you mean that your backup process includes running a RESTORE VERIFYONLY operation after the backup is complete, then no, you won't be assured that the backup can be restored. The RESTORE VERIFYONLY operation can perform two sets of checks.

Firstly, it reads the backup header to ensure that it is correct. If some corruption had affected the header, this would prevent your restore from running correctly.

```
RESTORE VERIFYONLY
FROM DISK= ‘<Backup_location>’
```

A second check becomes possible only if you are also running BACKUP WITH CHECKSUM for your backup operations. This means that during the backup SQL Server will recalculate the checksums as it reads each page to back up. If it finds a page that fails this test, the backup will fail. If it completes successfully, BACKUP WITH CHECKSUM computes a checksum over the backup file.

Subsequently, we can use the RESTORE VERIFYONLY to re-validate the backup file checksum to help ensure that nothing has corrupted the file during storage.

```
RESTORE VERIFYONLY
FROM DISK= '<Backup_location>'
WITH CHECKSUM
```

The problems come in two places. First, the header verification process of VERIFYONLY doesn't check every single aspect of the header that could affect the restore process. This means that you can pass the VERIFYONLY test, but still fail a restore.

Second, CHECKSUM cannot detect corruption that may have occurred in-memory. If a data page cached in memory is updated, and then some in-memory corruption occurs before the modified page is written back to disk (and subsequently to backup), then the checksums will not catch this corruption. CHECKSUM just validates that your backup wrote the pages that it read. If those pages are corrupted internally in the database, CHECKSUM won't tell you and your restore can fail.

The only way to know for sure that a backup will restore is to restore it and preferably run a database consistency check on the restored copy.

Does a backup contain more than the data? Can someone read passwords from it?

A backup does contain more than just the data. It contains the entire structure of the database. This includes all the data structures and data, of course, but it also includes all the procedures, views, functions and any other code. It also contains the settings and definitions of the database. Finally, it includes the users of the database. For a regular database, each user maps to a login on the server. The passwords for those users would be stored with the login, not with the user, so these passwords would not be available through the backup.

However, contained databases have a concept called USER WITH PASSWORD, since the idea behind a contained database is to have a database that is divorced from relationships with the server. In this case, the password will be stored with the backup, which does make it subject to the possibility of hacking attempts. The passwords are not stored in plain text; they’re hashed the same way as the login passwords (which would be in the master database and hence in the backups of master).

Microsoft discusses some of the security best practices within MSDN as they relate to contained databases.

Do I have to backup indexes, statistics and other stuff that is easy to recreate? It seems a waste.

On the contrary, I'd argue that it's a waste of time to try to split things out like this, and only backup up some of it. Firstly, how would you do it? For example, how do you back up the data without backing up the clustered indexes? You can't; those clustered indexes are the data pages. To all intents and purposes, those are your tables, so now you have to include clustered indexes. Possibly, you could split the non-clustered indexes into a separate filegroup that you don't back up, but then during a restore, you're still going to have to bring that filegroup back online and rebuild the indexes, so what have you gained?

Another problem comes from statistics on your indexes. SQL Server backs up these statistics with a database (and they take up very little room, since the histogram that defines a set of statistics is only 200 rows long), and restores them with the database. However, if you have to recreate your indexes after a restore operation, because you opted not to back them up, then you also have to recreate your statistics. This extends still further the time it takes to restore your database, keeping the business offline longer.

Overall, I would challenge you to define "easy to recreate", because getting these things back online in the event of an emergency could be a very complicated, multi-step process that will inevitably leave the business people and customers who are dependent on your database waiting much longer during disaster recovery.

The whole idea of having a backup is to be able to do a restore quickly and efficiently. The more difficult you make the restore process, the more you lessen the utility of that backup. Yes, there is some additional space needed to backup indexes, foreign key constraints, users and stored procedures, but the increased speed you get from having everything in one place is well worth that small increase.

OMG! I've just deleted a table! I know it is in the log. How do I get it back?

Once a transaction is committed, there is no undo from SQL Server. A DELETE operation and a TRUNCATE operation remove data in slightly different ways. The DELETE removes the data through a transaction that removes each row. The TRUNCATE simply de-allocates the page storage of the data. Neither one can easily be recreated by going through the log manually. Instead, you need to perform a process called a point in time recovery. You should immediately take a log backup of your database in order to preserve the changes up to the moment when you accidentally removed the data from the table. Then, you should follow the steps outlined in Chapter 6 of SQL Backup and Restore to restore to a point in time.

Another option you have is to open up SQL Backup Pro, which has a built-in wizard that will run you through an object-level restore process. It will find all the appropriate backups to allow you to retrieve your table and get it back online.

I just want to extract a build script from a backup without restoring it…?

No native method will allow you to extract a build script from a backup, without running a restore. However, a tool such as SQL Compare will allow you to generate a complete database build script from a backup. Although it is easy to do it from the GUI, you can also do it from PowerShell

```
& 'C:\Program Files (x86)\Red Gate\SQL Compare 8\SQLCompare.exe' /
Backup1:C:\MyBackups\MyBackupFile.bak  /MakeScripts:"C:\MyScripts\
MyBackupScript"
```