Do you understand how SQL backups work? Are you using a 3rd party package and relying on it to handle your backups and restores? Are you sure that your backups are working properly? Check again, are you REALLY sure? Are you willing to bet your job on it? If not, maybe it's time you take a few minutes and review how SQL Backups work and how to restore them. The time to test your knowledge of backing up and restoring is not when you're restoring after a computer crash.

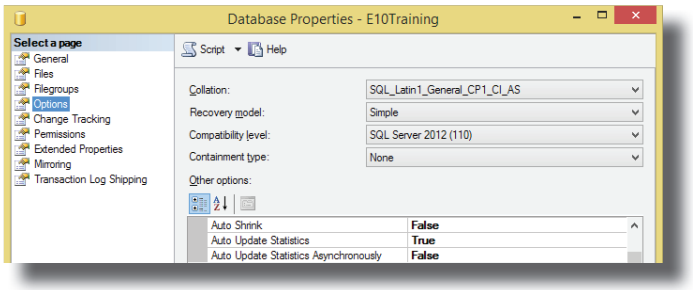
It is easy to rely on someone else (especially a third party package) and assume that all is working properly. You need to understand some basic things about SQL to ensure, even with that third party package, that you have configured things properly. We're going to walk through how SQL backups work and how to properly restore them so that you KNOW you are ready for disaster.

WHAT IS THE POINT OF A BACKUP?

The main goal of a backup is to recover from file corruption, hardware failure, and disaster with minimal downtime and data loss. Depending on the process you use, you can also use the backup to "back out" mistakes (such as when Bob deletes every customer in the database). To meet these goals, we need to understand how SQL can back up the database and what would have to be done to restore a database from backup.

DATABASE RECOVERY MODEL

Startup SQL Management Studio and navigate to one of your databases. Right-click on the database and choose Properties. Now go to the Options page. Look at the Recovery Model drop down. Do you know how the recovery model affects your backup? Each one handles transactions a little differently. The recovery model you choose will affect what you have to do when you restore. Let's take a look at each of the recovery model types.



SIMPLE

With this recovery model as soon as a transaction completes, it is taken out of the log file. The log only exists so that transactions can roll back if needed. When you perform a backup, you do a full backup. Restoring is done from a full backup. Although the database still has a log file, it is not used for the recovery process.

Per Microsoft, "The simple recovery model is inappropriate for production systems for which loss of recent changes is unacceptable."

WHAT'S A TRANSACTION?

A simple definition: a transaction is any operation that changes data in the database such as insert, update, and deletes. Things that modify the database structure itself (the schema) are also logged as transactions.

FULL

This is probably the most powerful of the three models and the one most commonly used in production systems. As transactions are performed on the database, they are written to a log file as well as to the database. Backing up involves backing up both the database and the transaction log (more on this later). To restore, you restore the database backup(s) and then you apply the transaction log(s) in order.

Bulk-logged

The bulk-logged recovery model is meant to be used temporarily while you do bulk operations. For example, if you needed to insert 25,000 new parts into your table, you might do this as a bulk operation. There's no need to track the individual inserts as you're not going to roll them back. With the Bulk-logged recovery model, normal transactions are logged as usual, but bulk operations are "minimally logged". You'll be able to restore from backups while using the bulk-logged model, but as soon as you finish your bulk operations, you should switch back to the full model, make a full backup, and then resume business as usual.

MINIMALLY LOGGED

- bcp
- Bulk Insert
- Insert...Select
- Select into
- Some index operations

The bulk-logged recovery model still requires that you have transaction logs. If you need to do a restore, you won't be able to restore to a point in time during the time that you are bulk-logging and until you do a full backup of the database.

TYPES OF BACKUPS

So now we know that unless we're bulk-loading data we need to be in the Full recovery model, next we need to understand the different types of backups. By understanding how the different backups work and how you restore from them, you'll be able to develop a strategy for your own situation.

BACKUP WHILE "IN-USE"

You don't have to stop using a database to back it up. Users can be active in the system while a backup is being performed.

There are three types of backups that you can do with SQL Server: Full, Differential, and Transaction Log. Each is described below and has an impact on your backup and restore strategy.

Full

A Full Backup contains a copy of the database as it was when the backup was made. All the information in the database is in the backup. This is the one type of backup where a single file can be used for restoring the database. A Full Backup does not back up nor clear out the transaction log. Backing up your transaction log is a separate operation (see below).

Differential

A Differential Backup contains the changes since the most recent full backup. Without going into all the details of how SQL stores information, we can say that this backup tracks the "bits" that have changed since the last Full backup. There's more to it than that, but this will suffice for our needs now.

Every time a differential backup is performed, it contains all the information from the most recent Full backup up to the time that the Differential backup happens. This includes the changes that are recorded in previous Differential backups.

Since the backup contains only the changed information, typically this is a smaller backup file and is faster to perform compared to a full backup.

Transaction Log

Transaction Log backups are sometimes called Incremental backups to fit within the normal backup/restore terminology that one would use when backing up file storage. A Transaction Log backup does exactly what you think it would: it backs up the transaction log. So all the transactions that have happened since the last transaction log backup are written to the backup file.

When it does this, a marker is written to the transaction log so that SQL knows that the transactions have been backed up. This way SQL can "re-use" the file space for future transactions. It's doing these transaction log backups that keep the log file from growing out of control. That would be a bad thing as it will effectively stop your database from accepting any further transactions (inserts, updates, deletes, etc.). A Full backup will not affect the transaction log, so it is very important that you perform transaction log backups on a regular basis.

These backups are typically quick (since they are only the transactions). You'll want to monitor your log file to find out the maximum time span between transaction log backups so that the transaction log does not fill up and prevent future transactions on the database.

You'll need to decide how much data you're willing to lose as that will indicate the minimum time span between log file backups. For example, if your system is not frequently used, perhaps you can handle re-entering four hours worth of information. In that case, you can do a transaction log every four hours. Instead, if your system is heavily used, maybe you can only afford to lose 15 minutes of data or else there would be too much work lost. Then you'd want to back up your transaction log at least every 15 minutes.

HOW DO I RESTORE?

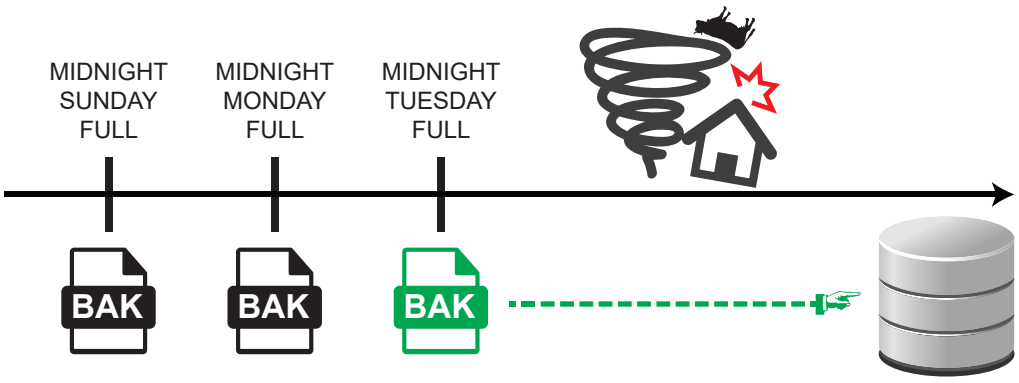
Understanding what the backups do is one thing, but when disaster strikes, you need to be able to get your database back online. Depending on the choices you make for your backup, the steps you take to restore your database will change. Once you understand how to do the restores, you can develop a backup strategy that works for you and your company.

One thing to keep in mind with any of the restore paths: **always** restore the full backup first.

FULL BACK UP RESTORES

Let's say you're using the Simple recovery model (otherwise you'd have some transactions log backups as well, right?). You are doing a full back up every night at midnight. The system crashes sometime on Tuesday. What do you do?

This is the easy one. All that you can restore is the full back up from midnight Tuesday. Any data changed in the system since midnight Tuesday will not be recovered. Once you restore Tuesday's full backup, you're back online. That Full backup has all you need to get your database back up and going again. Simple. Easy. Quick.



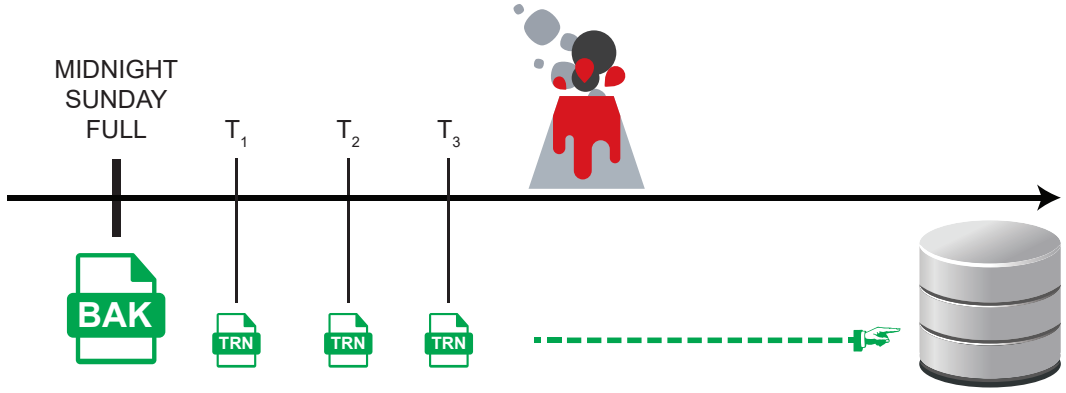
With the full backup and the full restore, there's not the option to restore to a point in time like there is with the other two restore methods that we'll discuss next.

Transaction Log Backup and Restore

Restoring with transaction log backups takes a little more effort, but also gives you a little more power. The general concept here is that you restore the Full backup, then you apply the transactions logs taken since the Full backup. Sometimes this process is called "playing forward" the transaction logs. In this way you're essentially redoing every transaction involved since the full backup. Once complete, you'll be at the state the database was at when the last transaction log was taken.

In this scenario, you're using the Full recovery model. Every night you do a full back up at midnight. A transaction log backup is performed multiple times a day. Disaster happens after the third transaction log backup for the day. What do you do?

Remember, the first thing you'll do is restore the full back up from midnight. Next you'll restore each of the transaction logs in order. When you're done, your database will be at the point it was just after the third transaction log backup. You will only lose the information taken between the third transaction log backup and the disaster. In our example we show three transaction log backups, but in reality you would run the transaction log backups on a frequency for the amount of data loss you are willing to have.



With this method, you also have the capability to restore the full back up, transaction log 1 (T_1), T_2, and then T_3 up to 20 minutes after T_2 was taken. Effectively backing out anything entered 20 minutes after T_2 was taken.

SQL Server makes restoring the chain of log files easy: when you restore, you can select the full backup and all the transaction logs together. SQL Server will read the files and place them in the proper sequence. SQL will also notify you if the log chain is broken and the restore cannot be performed.

During each of the backups, there is a "marker" that is written to the database and to the backup file that indicates what information was backed up. This is known as the Logical Sequence Number or LSN. The backup file will contain information about the first and last LSN that the backup file contains. When restoring multiple files, SQL Server will look at this LSN information to order the files properly. If there is a "gap" in the LSNs, you won't be able to restore past that gap. This is known as "breaking the chain" since the sequence of transactions is interrupted. When you restore you must have all the files in the sequence to the point you wish to restore. You cannot skip a file as SQL Server won't be able to rebuild the database.

Point in Time Recovery

You don't have to restore to the latest transaction that is recorded in a backup. With point in time recovery, you can restore to a specific time. For example, Bob accidentally deletes all the customers in the database at 9:05 am. Using the 9:10 am transaction log backup, you can restore the database up to 9:04:50 am; essentially backing out Bob's delete. How cool is that?

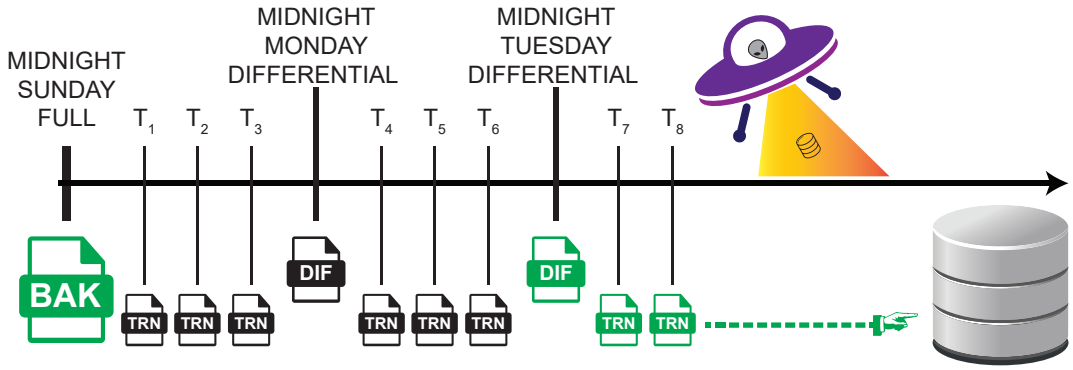
The **first_lsn** of B equals the **last_lsn** of A.

Differential backup with transaction log backups and restore

Differential backups are a little more complicated, but sometimes they are the fastest to restore. The general pattern is to restore the Full backup, restore the most recent differential, then apply the transaction logs since the most recent differential.

This restore path also relies on the database being in the Full recovery model. In our example, you do a full back up on Sunday at midnight, a differential backup every weekday at midnight, and transaction log backups multiple times a day. Disaster happens Tuesday after the second log backup (T_8). What do you do now?

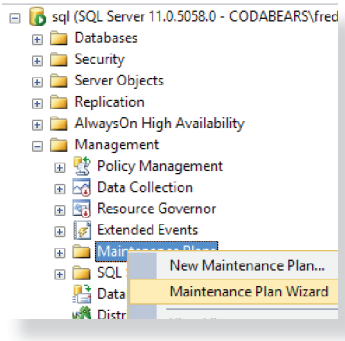
Restore the full backup first, then the most recent differential backup. In our example, that would be Tuesday's differential backup. There is no need to restore Monday's differential as Monday's changes are included in Tuesday's differential backup. Finally restore the transaction logs that were taken since the most recent differential backup. In our case that would be T_7 and T_8. We do not need to restore T_1-T_6 as those changes are accounted for in the Tuesday differential backup file. Once complete, the database would be at the state it was when we took the T_8 backup.



How do I create backups?

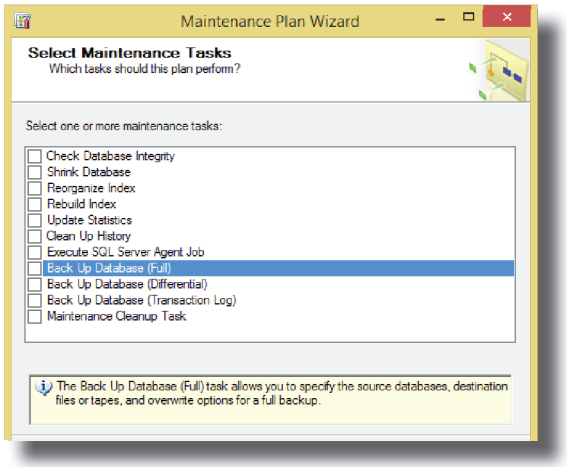
Maintenance plans

The easiest and most common way to create backups is to create a maintenance plan in SQL Management Studio. You can right-click on the Maintenance Plans node and choose "Maintenance Plan Wizard".

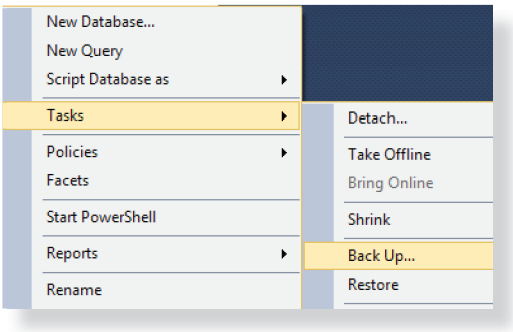


After specifying the name of the plan and the schedule, the wizard will display a dialog allowing you to select the tasks you wish to perform during this plan. Although the dialog allows you to specify additional tasks, we're going to stay focused on the backup tasks for this paper. The wizard will walk you through the additional steps to gather the information needed for the tasks you select. When the wizard completes, the job will be created in SQL Server Agent (assuming you specified a schedule).

For more information, please look to SQL Books Online.



Manually



Sometimes you just need to make a one-time backup of your database. With SQL Server, you can do this manually. In SQL Management Studio, right-click on a database, choose Tasks, and the select Backup. This will present you with a dialog asking for more information about the backup (such as the type, location, compression, etc.).

But what's this copy-only backup checkbox? There is an option when you perform a backup to perform a copy-only backup. Imagine if you wanted to take a copy of your production database and put it into your test environment. If you took a Full back up, then you would update the marker in the database of when the last full backup occurred. If you then moved this full back up to a different machine, you would no longer have it available to restore. The LSNs would no longer be in sequence when you attempt to restore. This would limit your point-in-time recovery to the log file that is before the full backup you did for the test server. To not break the chain, check the "copy only" checkbox and SQL will make a backup file, but not update the markers so that the chain remains unbroken.

Using 3rd party software

There are several 3rd party products available that perform backups of machines, disks, and also SQL databases. If you are using (or looking to use) one of these packages, please make sure to check the following:

- The package supports backing up and restoring SQL databases. The software cannot just back up the MDF and LDF files that SQL uses. Restoring these as files will typically result in a corrupted, unusable database. Instead the software **must** interface with SQL's back up functions and perform a true SQL backup.
- How does the software back up the database and log file? Some packages will allow you to truncate the log file after performing a backup. Be aware that this can break the log chain and you won't be able to restore to a point in time after the truncation. Know what the software is doing and how it affects your backup plans. More importantly, know how it affects your restore plans!
- Do not implement maintenance plans to back up the database unless you thoroughly understand when each event happens and you verify that one process is not interfering with the other. If your maintenance plan performs a backup and that backup doesn't go where the 3rd party software restores from, you will not be able to restore your database using the 3rd party tool. You'll need to coordinate these two tools if you are going to be using both.
- Test backing up and restoring your database. The 3rd party tool may have specific processes that must be followed for the restore to work properly. Familiarize yourself with these steps before you have to use them. Make sure that your backups are functional and that you know the chain is unbroken. Test restoring the backup so that you know the process and can easily do it under pressure.

So what's your strategy?

Now that you understand what happens during SQL backups and how to restore the files properly, you can make decisions about what backups you should perform and when. Your plan should address these issues:

1. Make sure all who have a stake in your SQL Server know the plan, understand it, and agree to it. If they don't know how long it will take you to restore, when the need arises you'll be dealing not only with the pressure of getting servers back online, but also with the CEO breathing down your neck asking you when it will be done.

2. How long can you be down when you need to restore a database? The different backup types offer different restore times. You'll need to test your backups to ensure you know how long a restore will take. If you have to tell the CEO, "Yes, we have backups, but it's going to take me a full day to restore," you're going to probably be looking for a new job….well, at least after you get the restore done.

3. How much data can you lose when you restore? This answer is going to tell you how often you need to perform a backup so that you meet this goal.

4. How much disk space can you dedicate to your backups? Disks are exceedingly cheap nowadays. How much would it really cost to allow you to have an extra couple days' worth of backups? Remember, as your database grows and you add more databases to your server, you'll have more to backup. This is not a "set it and forget it" process. You need to remember to check the processes. Verify you don't have a problem creeping up on you. Surprises are great for birthdays, not for IT.

5. How much time does it take to back up your database? You want to have as short a window for the backup as is reasonable.

   a. Using fast disks on the local machine, you can get very good backup speeds which make for quick backups and restores.

   b. If your database is significantly large, you may need to write the backup to multiple files to increase the speed of the backup. For more information on this refer to SQL's Books Online.

   c. Back up to fast disks locally, then copy your file across to a network location. Then you'll have the speed of a local backup, but you'll also have an off-machine copy of the file in case your SQL Server crashes.

6. Where are the backup files located?

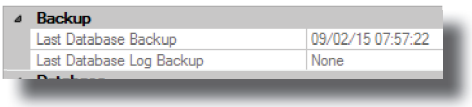
   a. If your backup files are located on the same drive as your databases, when that drive crashes, you've now lost the databases and the backups.

   b. If your backups are on the server that houses your databases, what happens when that server crashes or there's a power surge that fries the disks? You've lost both the database and the backup files at the same time again.

   c. OK, store the files on a network location separate from the server. Your files are now safe as long as the building stays intact. For this reason, it is common to take a copy of backups offsite so that in the case of complete physical disaster, you still have a copy of the backups to perform a restore.

7. Do you need point-in-time recoverability? If you do, make sure you understand how to restore with this option and that your backup strategy supports it.

8. How often are you testing your backups? Did you restore a backup to a test system to make sure that the files are not corrupt? Don't assume that things are working. Make sure that they work so that you don't have even more problems when you go to restore. To verify the backups ran, check the database properties to make sure that the database is being backed up when you think it is. If you do this, you'll likely catch a problem when there's a change. For example, if you make a change to using a new software, make sure you test restoring from that new software package as well. If it doesn't restore properly during the test, there's no way it will restore properly when you need it to. Be proactive, not reactive!



9. DOCUMENT THE PROCESS!!! You may know what's going on, but if you're swept up by that tornado that took out your server, someone else is going to have to do the restore. Document how to do the restore and alternative contacts. Your coworkers will thank you as you're exploring Oz.

10. Unless you are using the Simple recovery model, make sure you are doing some form of transaction log backup or your transaction log file will continue to grow.

11. If you change your strategy or backup controller, INFORM ANYONE WHO MAY HAVE A STAKE IN YOUR SQL SERVER! If you are relying on an outside company to do the restore for you, but you change how things are being backed up, be prepared for them to tell you they'll do their best, but they may not be able to help you.

REFERENCES

SQL Server Books Online

https://msdn.microsoft.com/en-us/library/ms189275.aspx

https://msdn.microsoft.com/en-us/library/ms190925.aspx#MinimallyLogged

http://www.sqlskills.com/blogs/paul/misconceptions-around-the-log-and-log-backups-how-to-convince-yourself/

School of Hard Knocks