Introduction to SQL Server 2005 Database Backups

Written by **Mladen Prajdić** on **19 December 2007**
http://www.sqlteam.com/article/introduction-to-sql-server-database-backups

Backing up a database is one of the most important things you need to do when having a database driven application. It 's only all of your data in there, right? But often developers and management don't realize the importance of backups and overall proper backup strategy for the most important side of the business – data and it's consistency.

How backup works

When you run a backup database command SQL Server performs a Checkpoint on the data pages in memory. Checkpoint means that all transactionally committed dirty pages are written to disk. Dirty pages are simply changed pages in memory that haven't been written to disk yet. After this the data on the disk is backed up in one or multiple files depending on your requirements. A backup must be able to be restored to a transactionally consistent state which means that it will also contain the data from the transaction log needed to undo all of the transactions which are running while the backup is taken.

Recovery models

Full recovery model

Every DML (insert, update, delete) statement is fully logged. Also every bulk insert operation (BCP, BulkInsert, SqlBulkCopy in .Net, SELECT INTO) is fully logged for each row. This recovery model also logs every CREATE INDEX and ALTER INDEX statement which are DDL operations. This means that when you recover a transaction log that contains logged CREATE INDEX you don't have to rebuild an index since it's already built. This behavior has changed since SQL Server 2000 where only the index creation event was logged, and not the whole index. The downside of this model is that it can consume a lot of disk space very fast.

Bulk-Logged recovery model

This model differs from the Full recovery in that it doesn't log every row insert on BULK operations I mentioned in Full recovery model. SQL Server does log that the operation has been executed and information about the data page allocations. However all data can still be restored. This is handled by the Bulk Change Map (BCM). SQL Server keeps track of the changed extents under this model by setting a bit representing an extent to 1 in the BCM, if that extent has changed. That is why the BULK operations are also faster under this model than under the Full Recovery, since logging each row is much slower than just setting a bit to 1. When you take a transaction log backup, all changed extents are also backed up by reading the BCM. This means that the transaction log itself is smaller but the transaction log backup can be a lot larger that the one under Full recovery model. Note that under this model you can't do point-in-time recovery on any transaction log backup that contains a bulk-logged transaction.

Simple recovery model

Under this recovery model you can't backup a transaction log at all. An attempt to do so results in an error, since there's nothing to update. The transaction log gets truncated at every checkpoint (writing data from a log to a disk) which happens at predetermined intervals. Also changing the database recovery model to Simple will immediately truncate the transaction log. A common misunderstanding is that nothing is being logged under this model. That is NOT TRUE. Everything is logged, you just don't have the point-in-time recovery ability. Bulk operations are minimally logged as in Bulk-Logged recovery model.

Transaction log marks

These are only available under Full and Bulk logged recovery models. Log marks are set in the transaction you want to recover to. You can do this with the begin transaction statement:

```
BEGIN TRANSACTION TranMark1 WITH MARK 'The mark description'
```

The name of the mark in the transaction log is TranMark1. After this transaction commits the mark is inserted into the logmarkhistory table in the msdb database and into the transaction logs of other related databases. Related databases are beyond the scope of this article, but simply put: the databases are related when we make related updates to them (e.g.: update to table1 in db1 has to be followed by update to table2 in db2). With log marks over multiple databases we can recover all databases to a specific related state to each other.

Types of Backups

Full database Backup

This backs up the whole database. In order to have further differential or transaction log backups you have to create the full database backup first.

```
-- Back up the AdventureWorks as full backup
BACKUP DATABASE AdventureWorks
   TO DISK = N'c:\AdventureWorksDiff.bak'
```

Differential database backup

Differential database backups are cumulative. This means that each differential database backup backs up the all the changes from the last Full database backup and NOT last Differential backup.

```
-- Back up the AdventureWorks as differential backup
BACKUP DATABASE AdventureWorks
   TO DISK = N'c:\AdventureWorksDiff.bak' WITH DIFFERENTIAL
```

Transaction log backup

Transaction log backup aren't possible under the Simple Recovery model. Two transaction logs can't contain the same transactions which means that when restoring you have to restore every backup in the order they were taken

```
-- Back up the AdventureWorks transaction log
BACKUP LOG AdventureWorks
         TO DISK = N'c:\AdventureWorksLog.bak'
```

Tail log backup

There seems to be a lot of confusion about this one since it's a new term in SQL Server 2005 (I haven't heard it being used in SS2k). A Tail log backup is the last Transaction log backup that you make prior to restoring a database. What this means is that if your db crashes for whatever reason, you have to backup your transaction log so that you can do point in time recovery. This last backup is called Tail log backup. If your data file (MDF) is unavailable you need to use WITH NO_TRUNCATE option:

```
-- Back up the AdventureWorks tail-log
BACKUP LOG AdventureWorks
   TO DISK = N'c:\AdventureWorksTailLog.bak' WITH NO_TRUNCATE
```

If your database is in OFFLINE or EMERGENCY state then tail log backup isn't possible.

Mirrored backup

Mirrored backups simply write the backup to more than one destination. You can write up to four mirrors per media set. This increases the possibility of a successful restore if a backup media gets corrupted. Following statement gives us two backups that we can restore from:

```
-- make one backup on d: disk and a mirror on e: disk
BACKUP DATABASE AdventureWorks
       -- media family 1
       TO DISK = 'd:\AdventureWorksMirror_1.bak'
   MIRROR
       -- media family 1
       TO DISK = 'e:\AdventureWorksMirror_2.bak'
 -- create a new mirrored backup set
   WITH FORMAT;
```

Copy-only backup

Copy-only backups are new in SQL Server 2005 and are used to create a full database or transaction log backup without breaking the log chain. A copy-only full backup can't be used as a basis for a differential backup, nor can you create a differential copy only backup.

```
-- Back up the AdventureWorks database as copy only
BACKUP DATABASE AdventureWorks
   TO DISK = N'c:\AdventureWorksDiff.bak' WITH COPY_ONLY

-- Back up the AdventureWorks transaction log as copy only
```

```
BACKUP LOG AdventureWorks
   TO DISK = N'c:\AdventureWorksLog.bak' WITH COPY_ONLY
```

File and file group backup

If you have your data spread across multiple files or filegroups you can take a full or differential backup of file(s) or filegroup(s):

```
-- Backup AdventureWorks_file1 file of the
-- AdventureWorks database to disk
-- note that AdventureWorks has to have a AdventureWorks_file1 file
BACKUP DATABASE AdventureWorks
   FILE = 'AdventureWorks_file1'
   TO DISK = 'e:\AdventureWorks_file1.bak'
GO

-- Backup AdventureWorks_filegroup1 filegroup of the
-- AdventureWorks database to disk
-- note that AdventureWorks has to have a
-- AdventureWorks_filegroup1 filegroup
BACKUP DATABASE AdventureWorks
   FILEGROUP = 'AdventureWorks_filegroup1'
   TO DISK = 'e:\AdventureWorks_filegroup1.bak'
GO
```

For differential backups just add WITH DIFFERENTIAL option to the upper examples.

Partial Filegroup backup

Partial filegroup backups are used to backup large databases with one or more read-only filegroups. In a way they are similar to full database backup, but by default they don't include read-only files or filegroups. This means that they contain the primary filegroup, every read/write filegroup and an optional number of read only files or filegroups.

```
-- Create a partial filegroup backup. backs up all read/write
filegroup
-- and the read only AW_ReadOnly_FileGroup1 filegroup
BACKUP DATABASE AdventureWorks
   READ_WRITE_FILEGROUPS, FILEGROUP = 'AW_ReadOnly_FileGroup1'
   TO DISK = 'e:\AdventureWorks_partial.bak'
```

For differential backups just add WITH DIFFERENTIAL option to the upper example.

Note that all file and file group names are logical and not physical names.

Conclusion

In the next article I'll cover some possible data loss scenarios, types of backups for each recovery model discussed here and how to minimize data loss of your database.