IDERA® | SQL Server Whitepaper

INITIALIZING REPLICATION FROM BACKUP

BY **KENNETH FISHER** and **ROBERT L DAVIS**

Applies to: SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, SQL Server 2014, SQL Server 2016

SUMMARY

This whitepaper provides clear guidance on the perceived limitations and restrictions on initializing transactional replication subscribers from a backup.

SQL Server 2005 introduced initializing a transactional replication subscriber from a backup. When working with very large databases (VLDBs), initializing a subscription from a snapshot can have a considerable performance impact and reinitializing a subscriber can be prohibitive. Many administrators are already restoring a backup to the subscriber to ensure that the non-replicated objects and user permissions are in place.

The next logical step in the evolution of replication was to make better use of the backups that were already being applied to the subscribers. Despite the manageability improvements of initializing from a SQL backup, this methodology has not been widely adopted. There are perceived limitations and restrictions that would unnecessarily prevent administrators from implementing initialization from backup for transactional replication.

Reasons often given for not using initialization from backup include a need to use a third party compression for full backups, especially in SQL Server 2005 tool for encryption or to gain maximum compression ratios; a belief that the subscriber must be initialized from a full backup; and a requirement to limit subscribers to a subset of tables. All of these limitations are easy to handle with initialization from backup.

THE TROUBLE WITH SNAPSHOT INITIALIZATION

This whitepaper is by no means advocating that initialization from a snapshot should not be used. Snapshot initialization is very useful and would be the recommended process in many cases. It is especially useful for small databases and remains the simplest way to set up transactional replication. For large databases or datasets, snapshot creation can cause performance impact on the publication database, and that can be a real pain point for database administrators.

Snapshot replication does not take advantage of native SQL Server compression. Snapshots consist of straight bcp (bulk copy program) of data out of the publication and into the subscriber. If the dataset is large, the data files will in turn be large as well. Snapshot generation can be a lengthy process as can the time required to apply the snapshot to the subscriber.

There is a compression option for snapshots that uses the Windows CAB compression to compress the data files after they have been created. This option may help speed up the delivery of the snapshot, especially when there is limited network bandwidth between the publisher and subscriber, but there are a number of limitations. This process can have a noticeable effect on CPU usage during the compression phase. Snapshot compression should not be used if the server is already using very high levels of CPU. It is also limited by the operating system to files no larger than 2 GB. If a data file larger than 2 GB is created, the process will fail when it tries to compress it. The snapshot process is pretty good about breaking large tables up into files of 2 GB or smaller, but every once in a while, you hit one that is slightly larger. This option is not supported via the replication wizard but can be configured by using T-SQL when creating a publication or the properties dialog when altering a publication on the snapshot options tab.

Note that in addition to the @compressed_snapshot parameter, you must also provide an alternate snapshot location via the @alt_snapshot_folder parameter. A snapshot in the default location cannot be compressed, and only the snapshot generated in the alternate location will be compressed. Unlike backup compression, snapshot compression will make the snapshot generation run longer, not shorter. Compressed snapshots are first created uncompressed in the primary snapshot location and then compressed to the alternate snapshot folder. Yes, that means with compressed snapshots, you get two copies of the snapshot; one compressed, one not compressed.

WHICH BACKUPS ARE SUITABLE FOR INITIALIZATION

Initialization from backup is most beneficial with very large databases (VLDBs) as it enables the replication administrator to deliver a very large amount of data to a subscriber with minimal impact on the publisher and takes advantage of other technologies like backup compression. Yet the process was not widely adopted when it was released with SQL Server 2005 due to the lack of native backup compression in SQL Server.

In order to initialize a subscriber from a backup, SQL Server must be able to natively read directly from the backup file. This precluded the use of a backup that was compressed by 3rd party utilities. In SQL Server 2008 and newer, the replication engine is able to read natively compressed backups, so this limitation is no longer a factor. Furthermore, there is no requirement to use a full backup as the initialization source for a subscriber, so this limitation was not a total blocker for using initialization from backup in SQL Server 2005.

Books Online states that “any recent backup can be used if it was taken after the publication was enabled for initialization with a backup.” See http://msdn.microsoft.com/en-us/library/ms151705.aspx.

This clearly indicates that we can initialize a subscriber from an alternate type of backup such as a differential or log backup. If relying on third party backup utilities, you can still use the third party tool for the full backup and then use a much smaller backup, such as a log backup, as the initialization source for the subscriber.

The process to initialize a subscriber from a log backup would be similar to the following:

1. Create a full backup of the publisher
   a. Can be compressed via third party tool and sometimes this is still preferred as modern compression tools can achieve a higher compression ratio.
   b. Can be natively compressed or encrypted.

c. If the recovery model is FULL take a log backup as part of your regular processes in order to minimize the size of the log backup used to initialize the subscriber.

2. Restore the full backup on the subscriber.
   a. Use the NORECOVERY option so further backups can be restored.
3. Configure the replication publication.
4. Allow initialization from backup.
   a. Any backup created after this option is enabled can be used to initialize a subscriber.
5. Take a non-full backup of the publisher.
   a. If the recovery model is FULL then a log backup would be appropriate.
   a. If the recovery model is SIMPLE then use a differential backup. While the most recent full backup will work, taking and using a new full backup will minimize the size of the differential backup
6. Restore the non-full backup on the subscriber.
   a. Use the RECOVERY option to bring the subscriber database online.
7. Configure the subscription.
   a. Specify the log file or differential backup used in step 6 as the initialization source.

The replication wizard does not support initializing a subscriber from backup. If using this option, you must create the subscriber using T-SQL. The below code snippet is an example of how to create a push subscription initialized from backup. These commands would be executed in the publisher database.

```
-- Add subscription specifying the backup to use for initializing
-- Modify the below to use your backup location and server/ instance
    EXEC sys.sp_addsubscription
    @publication = N'<Name of publication>',
    @subscriber = N'<Subscriber server\instance>',
    @destination_db = N'<Subscriber database name>',
    @subscription_type = N'Push',
    @sync_type = N'initialize with backup',
    @backupdevicetype = N'Disk',
    @backupdevicename = N'<Path and name of backup file used for initialization>',
    @update_mode = N'read only',
    @subscriber_type = 0;

-- Add subscription agent
-- Modify the below to use your server/instance name
    EXEC sys.sp_addpushsubscription_agent
    @publication = N'<Name of publication>',
    @subscriber = N'<Subscriber server\instance>',
    @subscriber_db = N'<Subscriber database name>';
```

BEYOND SIMPLE INITIALIZATION FROM BACKUP

One of the other reasons some administrators have not adopted initialization from backup is because the full backup will contain the entire database. This generally means that the DBA has to drop any unwanted tables manually or via a script. If the database is very large, this could result in transferring a lot more data than is required and in a lot more work by the administrator. This spawned a discussion of alternate methods.

There was a discussion on Twitter one day regarding initialization from backup. David Levy (@Dave_Levy) proposed the idea of initializing a subscriber from a filegroup backup to avoid transferring unwanted tables to the subscriber. I discussed this idea briefly with Dave and Argenis Fernandez (@DBArgenis) before I was finally nominated to attempt to make it work.

As stated earlier, Books Online states that any backup can be used to initialize a subscriber as long as it was created after initialization from backup was enabled. This would seem to imply that a filegroup backup could be used. The bigger question is can a filegroup backup be used without first restoring a full backup.

Books Online further states the following on page 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."

This statement would seem to indicate that a full backup must first be restored before using any other type of backup to initialize a subscriber. However, I was able to make initialization from a filegroup backup without first transferring a full backup to the subscriber work with some caveats. The below limitations apply to using this process.

> **The filegroup restore must include the primary filegroup as it contains all system objects and definition of all user objects**
> **Relies on partial database availability which is an Enterprise-only feature**
> **All filegroups being restored must have the same base differential LSN**
Back up the filegroups together

To take full advantage of this process, I back up just the filegroups I want to restore in its own backup. This is not a requirement. I can do a partial restore of just select filegroups from a full backup or from a partial backup containing additional filegroups. For example, if I already had a full backup created, I could use the full backup for the restore and specify which filegroups to restore.

INITIALIZING FROM A FILEGROUP BACKUP

The process being outlined here is attempting to reduce the burden of backing up, copying, and restoring the full database. Even though the process will work with a full backup, the following describes the process using only filegroup backups.

CREATING THE FILEGROUP BACKUPS

There are two methods that you can use for creating the filegroup backup. If a portion of your database is read-only, you can back up the read-only filegroups in a separate process and the read-write filegroups together. Assuming that all of the filegroups that contain tables to be replicated are read-write filegroups, you can then transfer just that backup and restore the read-write filegroups only. You can also specify filegroups by name for the restore process if the read-write filegroup backup contains additional filegroups.

With this method, the read-only filegroup must be backed up by name to back up separately, but you can specify the read-write filegroups only by including the READ_WRITE_FILEGROUPS parameter for the BACKUP command. The syntax of this backup command would be:

```
BACKUP DATABASE <Database Name>
READ_WRITE_FILEGROUPS
TO DISK = '<Path to and name of backup file>'
WITH INIT;
```

Alternatively, you can specify the filegroups by name for the backup command. The syntax of the backup command is similar to backing up read-write filegroups except you specify the filegroups by name separated by commas if more than one. For example, this is the syntax for the backup command if backing up two filegroups (remember that it must include primary filegroup):

```
BACKUP DATABASE <Database Name>
FILEGROUP = 'primary',
FILEGROUP = '<Filegroup name>'
TO DISK = '<Path to and name of backup file>'
WITH INIT;
```

SEPARATION OF FILEGROUPS

This process does rely on separation of tables by filegroups. Most often, this is not the state of a database. To really take advantage of this process, it may mean taking one-time downtime to move some of the tables to alternate filegroups. This is no simple process for a very large database and may need to be performed piecemeal wise over a long period of time. To optimize the database for initializing replication from a backup it is best to move all tables that will be replicated into their own filegroup(s). A simple filegroup setup might look like this:

Primary

One best practice suggests creating all user defined objects in FGs other than primary and keeping it only for the system objects.

ReplicatedRW

A read/write filegroup that contains only the writeable tables that will be replicated.

ReplicatedRO

A read-only filegroup that contains only the read-only tables that will be restored to the replication subscriber.

NonReplicatedRW

A read/write filegroup that contains only the writeable tables that will NOT be replicated.

All code would go in this filegroup.

NonReplicatedRO

A read-only filegroup that contains only the read-only tables that will NOT restored to the replication subscriber.

Obviously only create those filegroups that are needed and additional filegroups could be created for additional separation (partitioning for example). Using a layout like this will not only require the least amount of data to be transferred but will eliminate the need to delete unnecessary objects from the replicated database.

RESTORING THE FILEGROUP BACKUPS

If you are restoring specific filegroups from a full backup or a backup with additional filegroups in it, then you must identify the filegroups you want restored. This may mean using the READ_WRITE_FILEGROUPS option or by specifying filegroups by name. Either way, you can use the exact same parameter in the RESTORE command that you used for the backup command. However, if the backup only includes the filegroups you want to restore, the parameters are optional. If you do not specify which filegroups to restore, the restore process will restore everything included in the backup. All filegroups in the backup will be restored unless you specify otherwise.

The following is sample restore commands demonstrating the restore of specific filegroups:

```
RESTORE DATABASE <Database Name>
READ_WRITE_FILEGROUPS
FROM DISK = '<Path to and name of backup file>'
WITH RECOVERY;
RESTORE DATABASE <Database Name>
FILEGROUP = 'primary',
FILEGROUP = '<Filegroup name>'
FROM DISK = '<Path to and name of backup file>'
WITH RECOVERY;
```

VERIFYING FILEGROUP STATE

After the restore is complete, the database will be online. Any files contained in those filgroups that were not restored will have a state of Recovery Pending. You can check the state of the files and filegroups by querying sys.database_files and sys.data_spaces inside the database you restored with the following query:

```
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 sys.database_files DF
LEFT JOIN sys.data_spaces DS
ON DS.data_space_id = DF.data_space_id
```

STEP-BY-STEP PROCESS

The process for initializing a subscriber from a backup file is not very different from initializing it from a log backup. The big differences are that you don't perform a full backup and restore first, and you specify the filegroups involved. We basically start at step 3 as follows:

1. Configure the replication publication.
2. Allow initialization from backup.
   a. Any backup created after this option is enabled can be used to initialize a subscriber
3. Take a backup of the filegroups on the publisher that you want to replicate.
4. Restore the filegroup backup on the subscriber.
   a. Use the RECOVERY option to bring the subscriber database online.
5. Configure the subscription.
   a. Specify the log file backup used in step 6 as the initialization source.

CONCLUSION

Initialization of a transactional replication subscriber is still a highly valuable tool. For very large databases (VLDBs), snapshot generati on and application can be performance impacting and take a long time. Initialization from backup was designed to alleviate some of the problems faced by administrators of VLDBs. Initialization from backup is a tool that anyone who needs to replicate a VLDB should consider.

There are perceived limitations regarding initialization from backup, but this paper has demonstrated that those are not insurmountable barriers to using this process. You can still use initialization from backup with a little creativity and utilizing log or filegroup backups.

ABOUT IDERA

IDERA understands that IT doesn't run on the network – it runs on the data and databases that power your business. That's why we design our products with the database as the nucleus of your IT universe.

Our database lifecycle management solutions allow database and IT professionals to design, monitor and manage data systems with complete confidence, whether in the cloud or on-premises.

We offer a diverse portfolio of free tools and educational resources to help you do more with less while giving you the knowledge to deliver even more than you did yesterday.

Whatever your need, IDERA has a solution.