**Automated Rapid Recovery during Database Code Releases using Snapshots**

**By** [**Warren Campbell**](http://www.sqlservercentral.com/Authors/Articles/Warren_Campbell/479574/)**, 2015/12/08**

**FROM:** [**http://www.sqlservercentral.com/articles/Snapshots/133563/**](http://www.sqlservercentral.com/articles/Snapshots/133563/)

**Background**

In VLDB environments, database backup and restore is not always a feasible option for recovery from errors or problems with deployment of new database code. I’ve often deployed code across scores of databases simultaneously, or near simultaneously and each database was hundreds of GB or multiple TB in size. In the event of a widespread deployment error or code problem, recovering many huge databases from backup would likely to take more time than is available in a typical deployment window and the result could be an outage lasting longer than planned.

Fortunately, SQL Server provides another option to allow for quick rollback in just this type of situation: database snapshots. We can use the custom stored procedures included in this article to simplify creation of database snapshots before deployment, allow for quick rollback to snapshots, and then drop the snapshots when we are confident our deployment was successful.

**Database Snapshots Primer**

A comprehensive description of database snapshot functionality is outside the scope of this post, but it’s worthwhile to give a brief overview of how snapshots work so that we can understand their advantages and caveats. Unlike full database backups, which contain all the data pages that make up the database, snapshots contain only data pages which have been modified since the snapshot was created. SQL Server uses the original, source database as a point of reference for pages that have not changed, and the snapshot as the source for pages that have changed. Because of this, SQL Server can revert to the state that the database was in at the time the snapshot was created.

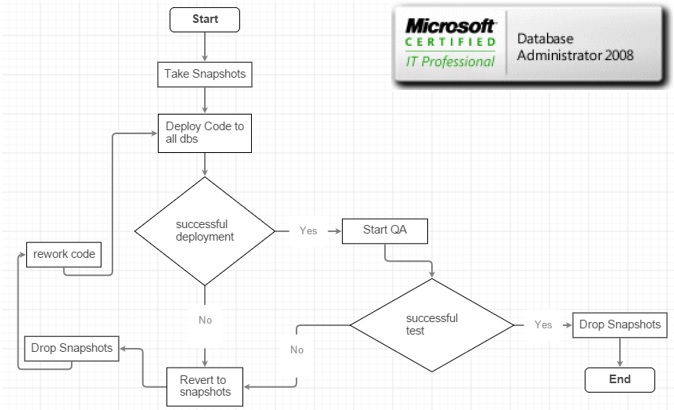
When a small percentage of pages have been changed, as is often the case during database code deployments on VLDBs, reverting to snapshots is much faster than restore and recovery from backup. While full backups might take many minutes or hours to be created, and similar durations to restore, snapshots on a quiet database are created almost instantly and with our typical patch deployments rollback takes seconds.

It’s worth cautioning that snapshots do not negate the need for backups. Reverting to snapshots requires an intact source database, so there are many recovery situations where snapshots are not an option. Also note, when a database is reverted to a snapshot, all changes to the database since the snapshot was created are lost, so it important that we use this method only during our maintenance windows, when we know that code deployment will be the only changes being made.

Finally, we should consider that database snapshots create overhead in terms of system performance and storage (since each page which is modified in the source database must also be written to in the snapshot database). Code deployment will take longer on a database which has a snapshot than on one that does not. This is especially true if the code changes a lot of pages as might occur with a schema change to a large table. Of course, when our deployment changes only a tiny percentage of our database, this overhead is likely to be negligible. In any event, it is critical that we ensure database snapshots are dropped at the end of our maintenance window on highly transactional systems as snapshot overhead during peak hours would be costly.

**Snapshot Protected Deployment Method**

Since our code deployment most often touches all or nearly all databases on our clusters, it makes sense for us to take snapshots of all user databases prior to deployment and drop them when the deployment is complete. The entire process looks like this:



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

That flowchart looks a bit complicated, but the process is actually pretty simple and is just a few steps. Once we are in the maintenance window and no one is accessing the databases, we take snapshots of all non-system databases by executing one stored procedure SP\_SNAPSHOT\_ALL\_DATABASES. Next we deploy our code. If there are no errors, we proceed with QA testing and if testing is successful, we execute another stored procedure, SP\_DROP\_ALL\_SNAPSHOTS, to drop all the snapshots.

If there are errors during deployment, or if QA smoke testing finds a problem serious enough to warrant canceling the deployment, we run a third stored procedure SP\_REVERT\_ALL\_TO\_SNAPSHOTS, to roll back all the databases back to the state they were at the time the snapshots were created. We can then deploy again, or drop all the snapshots using SP\_DROP\_ALL\_SNAPSHOTS.

**Stored Procedures Code**

You should note that these procedures that I’ve written, (and in fact this entire process) is based on the premise that there are no pre-existing database snapshots. These should not be used on systems which have existing snapshots.   The procs could of course be modified to do snapshots of a subset of databases as is appropriate in your environment.

In any event, you should be sure you understand exactly what is happening before using these scripts. I make no warrantees or guarantees whatsoever about this code. Only the person who executes this code is responsible for the outcome.

USE DBA;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SP\_SNAPSHOT\_ALL\_DATABASES] Script Date: 10/28/2015 11:20:46 \*\*\*\*\*\*/

SET ANSI\_NULLS ON;

GO

SET QUOTED\_IDENTIFIER ON;

GO

CREATE PROC dbo.SP\_SNAPSHOT\_ALL\_DATABASES

AS

BEGIN

DECLARE @sql TABLE ( stmt VARCHAR(4000) );

DECLARE @stmt NVARCHAR(4000);

INSERT @sql

(

stmt

)

SELECT

'create database ' + DB\_NAME(database\_id) + '\_Snapshot on (name='

+ name + ',filename=''' + physical\_name + '.ss'') as snapshot of '

+ DB\_NAME(database\_id)

FROM

sys.master\_files

WHERE

DB\_NAME(database\_id) NOT IN ( 'tempdb', 'model', 'master', 'dba' )

AND type = 0;

DECLARE sqlcursor CURSOR

FOR

SELECT

stmt

FROM

@sql;

OPEN sqlcursor;

FETCH NEXT FROM sqlcursor INTO @stmt;

WHILE @@FETCH\_STATUS = 0

BEGIN

SELECT

@stmt;

EXEC sys.sp\_executesql

@stmt;

FETCH NEXT FROM sqlcursor INTO @stmt;

END;

END;

GO

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

USE DBA;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SP\_REVERT\_ALL\_TO\_SNAPSHOTS] Script Date: 10/28/2015 11:21:15 \*\*\*\*\*\*/

SET ANSI\_NULLS ON;

GO

SET QUOTED\_IDENTIFIER ON;

GO

CREATE PROC dbo.SP\_REVERT\_ALL\_TO\_SNAPSHOTS

AS

BEGIN

DECLARE @sql TABLE ( stmt VARCHAR(4000) );

DECLARE @stmt NVARCHAR(4000);

INSERT @sql

(

stmt

)

SELECT

'alter database ' + a.name

+ ' set single\_user with rollback immediate'

FROM

sys.databases a

JOIN

sys.databases b

ON

a.database\_id = b.source\_database\_id

WHERE

b.source\_database\_id IS NOT NULL

AND a.name NOT IN ( 'DBA' );

INSERT @sql

(

stmt

)

SELECT

'restore database ' + a.name + ' from DATABASE\_SNAPSHOT='''

+ b.name + ''''

FROM

sys.databases a

JOIN

sys.databases b

ON

a.database\_id = b.source\_database\_id

WHERE

b.source\_database\_id IS NOT NULL

AND a.name NOT IN ( 'DBA' );

DECLARE sqlcursor CURSOR

FOR

SELECT

stmt

FROM

@sql;

OPEN sqlcursor;

FETCH NEXT FROM sqlcursor INTO @stmt;

WHILE @@FETCH\_STATUS = 0

BEGIN

SELECT

@stmt;

EXEC sys.sp\_executesql

@stmt;

FETCH NEXT FROM sqlcursor INTO @stmt;

END;

END;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

USE DBA;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SP\_DROP\_ALL\_SNAPSHOTS] Script Date: 10/28/2015 11:21:36 \*\*\*\*\*\*/

SET ANSI\_NULLS ON;

GO

SET QUOTED\_IDENTIFIER ON;

GO

CREATE PROC dbo.SP\_DROP\_ALL\_SNAPSHOTS

AS

BEGIN

DECLARE @sql TABLE ( stmt VARCHAR(4000) );

DECLARE @stmt NVARCHAR(4000);

INSERT @sql

(

stmt

)

SELECT

'drop database ' + name

FROM

sys.databases

WHERE

source\_database\_id IS NOT NULL;

DECLARE sqlcursor CURSOR

FOR

SELECT

stmt

FROM

@sql;

OPEN sqlcursor;

FETCH NEXT FROM sqlcursor INTO @stmt;

WHILE @@FETCH\_STATUS = 0

BEGIN

SELECT

@stmt;

EXEC sys.sp\_executesql

@stmt;

FETCH NEXT FROM sqlcursor INTO @stmt;

END;

END;

GO

--