

SQL Server Database Restore

Documentation

Table of Contents

1 Introduction.....	3
2 Technical preview.....	4
3 Deployment	5
4 After deployment	5
4.1 Direct messages	5
4.2 Stored procedures	5
4.3 Tables.....	6
5 Execution of stored procedures	6
5.1 Parameters	6
5.2 Restore of database.....	7
5.3 Restore of database that is joined in Availability Group	7
5.4 Executing stored procedure AddDatabaseOnSecondary	8
5.5 Messages	8
6 Possible problems.....	9
6.1 Reporting issues.....	9
7 Change history	10

1 Introduction

This document describes stored procedure covering all required actions to properly restore database to SQL Server instance. Including scenarios where restoring database that is part of Availability Group, and joining it to all secondaries. Restore process created based on experience with restoring databases especially in scenarios where restores needed on regular basis. Whole restore process is described in details further in document. Way of restoring can differ from your own process, but that is not definitely bad it is only my approach to doing so and can be wrong, but I have some bullet proof arguments, so feel free to start discussion.

Tested on SQL Server versions \geq 2008, so all older versions are not supported and you are running scripts/procedures on your own risk!

2 Technical preview

Whole solution consist of two stored procedures, that can be called directly or from SQL Agent job steps. One procedure is needed for all restore scenarios, and another only needed on Availability Group (only AG in further writing) secondary replicas to be able to join database to AG.

- RestoreDatabase – perform every restore
- AddDatabaseOnSecondary – only needed on secondary replicas

Both procedures using pure T-SQL approach, I know similar operations can be performed by PowerShell and maybe more efficiently, but I like T-SQL way.

Both procedures cooperating with Ola Hallengreen's maintenance solution procedures (visit here for more details <https://ola.hallengren.com/>), using its **CommandLog** table for tracking operations done during execution and **CommandExecute** for executing commands within script. Both table and procedure is created during deployment and you are informed about it.

All important information included in every procedures header also, for example any versions info and small release notes can be found there also

```
/*
Purpose:      This procedure can be used for regular restores of database that is part of availability group. Taking
               care of all actions needed for proper restore process of database in Availability group.
               It is also writing its actions to CommandLog which is able from popular Olla Hallengreen's maintenance.
Author:       Tomas Rybnicky
Version:      0.1
Last modified: 30.10.2018

Execution example:
-- restore database
EXEC [master].[dbo].[usp_RestoreDatabase]
@BackupFile = N'\\SQLDRPROD01\\SQL_Backups\\DYNAMICSPROD01\\DYNAMICSPROD01_healthcheck_BackupFull.bak',
@Database   = N'healthcheck',
@LogToTable = 'Y'

-- restore database and add to Availability Group
EXEC [master].[dbo].[usp_RestoreDatabase]
@BackupFile = N'\\SQLDRPROD01\\SQL_Backups\\DYNAMICSPROD01\\DYNAMICSPROD01_healthcheck_BackupFull.bak',
@Database   = N'healthcheck',
@AvailabilityGroup = N'SQLCLO2PRODAG',
@SharedFolder = N'\\SQLCLO2PROD01\\AGShare',
@LogToTable = 'Y'
*/
```

Figure 1 - procedure header info

3 Deployment

Only thing you have to do is to copy deployment script from its storage on [GitHub](#). Copy script to SQL Server Management Studio and run it against SQL Server instance you are connected to or use multiquery from Registered Servers. Running script using multi-query is especially beneficial when creating procedures on AG replicas, you will avoid unnecessary clicking when connecting to every replica and running one by one.

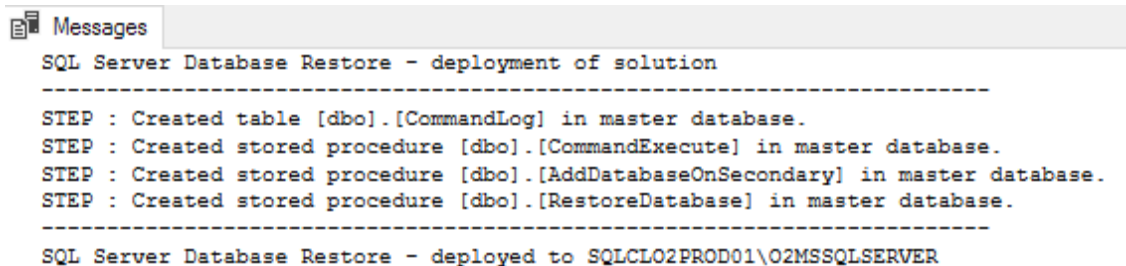
You can also change script body if there is something that you do not like there, but only by your own responsibility!

Deployment script is doing nothing magical, only creating two stored procedures in master database. First it checks if stored procedure already exists and drop and re-create it. **So please be aware of overwriting of your procedure if you have same naming.** You can also find some important info in script header, rather to read it before running anything against your servers, you should be aware of what you are doing also on non-production servers.

4 After deployment

4.1 Direct messages

After proper execution you can check messages for detailed steps which have been done over instance and also for possible related error messages.



```
SQL Server Database Restore - deployment of solution
-----
STEP : Created table [dbo].[CommandLog] in master database.
STEP : Created stored procedure [dbo].[CommandExecute] in master database.
STEP : Created stored procedure [dbo].[AddDatabaseOnSecondary] in master database.
STEP : Created stored procedure [dbo].[RestoreDatabase] in master database.
-----
SQL Server Database Restore - deployed to SQLCLO2PROD01\O2MSSQLSERVER
```

Figure 2 Script messages

4.2 Stored procedures

You can see two new stored procedures in master database + CommandExecute procedure from Ola Halengreen (if not there already)

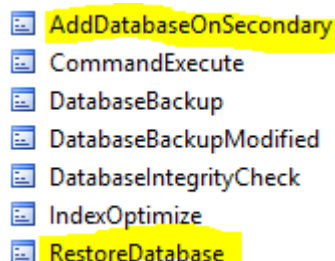


Figure 3 Stored procedures in master database

4.3 Tables

There is one table created within deployment – CommandLog if it already did not exist before deployment. It is borrowed from Ola's maintenance solution. If you are using Ola Halengreen's maintenance solution you can just see new records related to restores in this table.

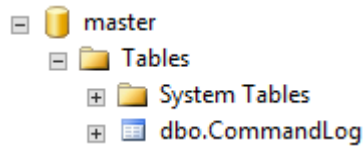


Figure 2 Script messages

5 Execution of stored procedures

OK so you are all set now and you can start enjoying new stored procedures. You can use *RestoreDatabase* procedure to make common restore to standalone SQL Server database or you can use it to refresh database that is part of Availability Group

5.1 Parameters

Both stored procedures are expecting some input parameters to work with. In below lists you can find some short description of parameters for both procedures

Procedure RestoreDatabase:

- **@BackupFile** *NVARCHAR(1024)* - Backup file that is to be used for restore
- **@Database** *SYSNAME* - Name of restored database
- **@AvailabilityGroup** *SYSNAME = NULL* - Name of Availability Group that is to be used for database. When NULL then normal restore operation happening
- **@SharedFolder** *NVARCHAR(2048) = NULL* - Path to shared network location accessible by all replicas. Required when adding to Availability group
- **@LogToTable** *CHAR(1) = 'N'* - Flag if restore commands are to be tracked in CommandLog table

Procedure AddDatabaseOnSecondary:

- **@FullBackupFile** *NVARCHAR(1024)* - Database backup file taken on primary replica
- **@TlogBackupFile** *NVARCHAR(1024)* - Transaction log backup file taken on primary replica
- **@Database** *SYSNAME* - Name of database
- **@AvailabilityGroup** *SYSNAME* - Name of Availability Group that is to be used for database
- **@LogToTable** *CHAR(1) = 'N'* - Flag if restore commands are to be tracked in CommandLog table

5.2 Restore of database

Lets describe process done behind the scenes when you are doing restore to common database not participating in any HADR, lets call it simple database.

```
EXEC [master].[dbo].[RestoreDatabase]
@BackupFile = N'\\SERVER\Backups\TesDB.bak',
@Database = N'TestDB_Restored',
@LogToTable = 'Y'
```

During restore of database folowing tasks are done in specified order:

1. Checking – core requirements before any executing
 - a. permissions
 - b. procedure CommandExecute
 - c. table CommandLog
2. Preparing – collect some important data and build restore command
 - a. gathering instance info
 - b. gathering backup file info
 - c. building restore command
3. Restoring database – performing restore from given backup using CommandExecute stored procedure (include logging in CommandLog table)
4. Post configuration – doing some reconfigurations to restored database
 - a. shrink log file
 - b. rename files
 - c. set multi user
 - d. set online
5. Informing about sucessfull restore

5.3 Restore of database that is joined in Availability Group

Lets describe process done behind the scenes when you are doing restore to database that is joined in Availability Group. Database can be already joined or is to be joined after restore.

```
EXEC [master].[dbo].[RestoreDatabase]
@BackupFile = N'\\SERVER\Backups\TesDB.bak',
@Database = N'TestDB_Restored',
@AvailabilityGroup = N'AVAILABILITYGROUPNAME',
@SharedFolder = N'\\SERVER\SharedFolder',
@LogToTable = 'Y'
```

During restore of database that is already or going to be part of Availability Group there are some prerequisites that have to be met to start data synchronizing across replicas. See [this article](#) for more info about all prerequisites. From beginning it is doing the same as [restore of common database](#) so only additional steps described here (what continues after restore of database):

1. Take full backup
2. Take backup of transaction log
3. Add database to Availability Group on primary replica
4. Iterate all secondary replicas
 - a. Check if linked server for secondary replica exists othwerwise create it
 - b. Check RPC Out config for linked server
 - c. Adding database to Availability Group by calling stored procedure AddDatabaseOnSecondary – see more info in [next section](#)
5. Informing about sucessfull joining on all secondaries
6. Informing about sucessfull restore

5.4 Executing stored procedure AddDatabaseOnSecondary

This procedure is executed only on secondary replicas where called by procedure RestoreDatabase from primary replica via linked server. So you only need this procedure if using restores of databases that are part of Availability Group. You can remove it once when planning to use RestoreDatabase only for common databases.

Folowing command is constructed automatically within RestoreDatabase execution but you can call procedure directly if you want:

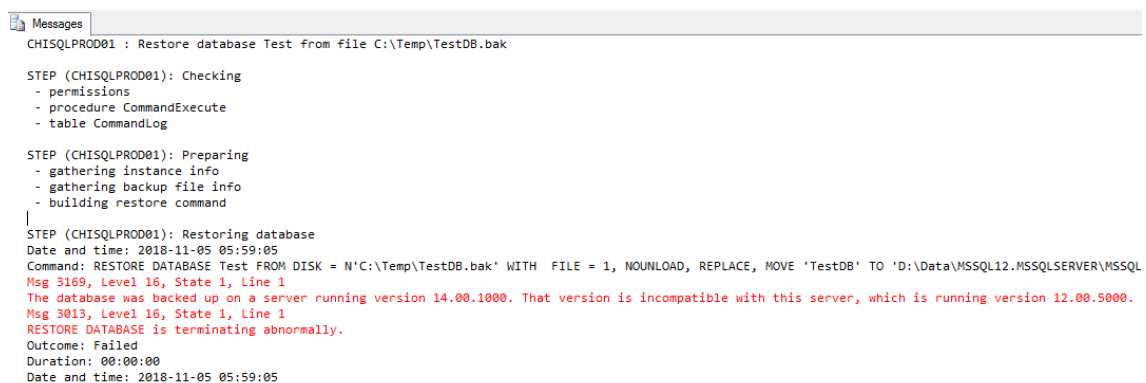
```
EXEC [master].[dbo].[AddDatabaseOnSecondary]
@FullBackupFile = N'\\SQLCLO2PROD01\AGShare\healthcheck_AG_init.bak',
@TlogBackupFile = N'\\SQLCLO2PROD01\AGShare\healthcheck_20181031164726',
@Database = N'healthcheck',
@AvailabilityGroup = N'SQLCLO2PRODAG',
@LogToTable = 'Y'
```

Folowing actions done during execution of this procedure:

1. Checking requirements
 - a. permissions
 - b. procedure CommandExecute
 - c. table CommandLog
 - d. availability group
2. Restoring database
 - a. restore full backup
 - b. restore of transaction log
3. Add database to Availability Group
 - a. add on secondary replica
4. Informing about sucessfull joining on secondary replica

5.5 Messages

Stored procedures are informing you via messages about its execution steps in pretty detailed info messages. Also you can find possible error descriptions in messages after execution failed.



The screenshot shows the 'Messages' window in SQL Server Enterprise Manager. The title bar reads 'Messages'. The main text area contains the following log entries:

```
CHISQLPROD01 : Restore database Test from file C:\Temp\TestDB.bak

STEP (CHISQLPROD01): Checking
- permissions
- procedure CommandExecute
- table CommandLog

STEP (CHISQLPROD01): Preparing
- gathering instance info
- gathering backup file info
- building restore command

STEP (CHISQLPROD01): Restoring database
Date and time: 2018-11-05 05:59:05
Command: RESTORE DATABASE Test FROM DISK = N'C:\Temp\TestDB.bak' WITH FILE = 1, NOUNLOAD, REPLACE, MOVE 'TestDB' TO 'D:\Data\MSSQL12.MSSQLSERVER\MSSQL
Msg 3169, Level 16, State 1, Line 1
The database was backed up on a server running version 14.00.1000. That version is incompatible with this server, which is running version 12.00.5000.
Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE is terminating abnormally.
Outcome: Failed
Duration: 00:00:00
Date and time: 2018-11-05 05:59:05
```

Figure 3 Procedure outcome in messages - failure

6 Possible problems

There was testing of the solution ongoing for several weeks for debugging and tuning purposes and all known problems has been fixed already, but as everything also this script can cause some issues in different environments.

I'm assuming only following possible issues:

- problems with accessing secondary replica via linked server - [Login failed for User 'NT AUTHORITY\ANONYMOUS LOGON'](#)
- When executing from SQL Agent job, ensure that account that is used for execution has sufficient permissions, especially in case restoring database into Availability Group as there are actions done on all secondary replicas.

And some other possible problems can be related to OH stuff in the solution so, please be so kind and try to check this FAQ <https://ola.hallengren.com/frequently-asked-questions.html> first before asking me directly.

6.1 Reporting issues

Please report all found issues, current version of the solution is the first one and require some debugging to be "perfect". Here are some contacts you can write to via email or LYNC:

- tomas.rybnicky@wetory.eu
- Use GitHub issues channel <https://github.com/wetory/SQL-Server-Database-Restore>

7 Change history

Version	Date	Author	Approved by	Change history
V1.0	02.11.2018	Tomáš Rybnický	-	First version of this document