Menu

Martin's Microsoft SQL Code Snippets - v20170816

I use this page myself.

DBDeving and DBAing are large and complex subjects, and sometimes it is good to have a little library of most commonly used snippets.

WARNING: Some of these snippets are intended to be used by experienced DBDev and DBA.

If you do not understand what the snippet does, or *how* it does it, **DO NOT USE**.

Snippets that beginners should really avoid will be marked with a warning sign and have a "WARNING" alert box (same as this text is in), describing some of the dangers involved in using the snippet.

You have been *warnededed*...

- Items to still document:
  - etc

GO?

```
GO
```

TOP

Unzombie Users

This is useful especially after restoring a DB from one server to another (say you are refreshing a QA box with the newest PROD data).

It is a relative safe operation, and will not touch or break existing and working DB users.

Get SQL here

```
/*
Source: Various
Usage: Run as DBO on SQL2008R2 or newer.
Caveat: Update the USE statement for the DB that youa re working on.
*/

-- USE [--INSERT DB NAME HERE--]

EXECUTE sp_change_users_login 'report' ; --See all Zombie users in the database.

GO

DECLARE @ZombieUsers TABLE
(
  idKey Int IDENTITY(1,1) PRIMARY KEY,
  UserName SysName,--nVarChar(128)
  UserSID  VarBinary(85)
)

INSERT INTO @ZombieUsers
EXECUTE sp_change_users_login 'report' ;

DECLARE @CRLF as nVarChar = CHAR(10) + '&' + CHAR(13)
DECLARE @Sql as nVarChar(MAX) = N''
DECLARE @idKey as Int = 1
DECLARE @MaxidKey as Int
    SET @MaxidKey = ( SELECT COUNT(*) FROM @ZombieUsers ) ;
DECLARE @theCount as Int = 0
DECLARE @UsersFixed as nVarChar(MAX) = N''
DECLARE @UserName as SysName -- Zombied Database user.

WHILE ( @idKey <= @MaxidKey )
BEGIN
  SET @UserName = ( SELECT UserName FROM @ZombieUsers WHERE idKey = @idKey ) ;
  IF ( SELECT COUNT(*) FROM sys.server_principals WHERE Name = @UserName ) = 1
  BEGIN
    SET @Sql = @Sql + 'EXECUTE sp_change_users_login ''update_one'', [' + @UserName + '], ['
    SET @UsersFixed = @UsersFixed + @UserName + ', ' ;
    SET @theCount = @theCount + 1 ;
  END
  SET @idKey = @idKey + 1 ;
END

PRINT @Sql ;
EXECUTE sp_executesql @Sql ;
PRINT   'Total fixed: ' + CAST(@theCount as VarChar) + '.  Users Fixed: ' + @UsersFixed ;
SELECT ( 'Total fixed: ' + CAST(@theCount as VarChar) + '.  Users Fixed: ' + @UsersFixed ) A

GO

EXECUTE sp_change_users_login 'report' ; --See all Zombie users still in the database.

GO
```

```
/* END */
```

Shrinking TempDB

For many years the common concession from most DBA was that shrinking the TEMPDB was just... ugh.

As you know (I hope), TEMPDB is dropped and recreated every time the SQL server is restarted.

But most of us DBA try not to restart our prod server unless required - for various reasons.

And yes, if you set up your server correctly and TEMPDB friendly, then there will never be a reason to shrink it. But we all have those inherited servers that we cannot (easily) refactor.

So sometimes you just need to be able to shrink the TEMPDB - for example in that emergency where the app owners forgot to inform anyone of the massive import they are running overnight, which grows the TEMPDB beyond recognition on a server config that is not TEMPDB friendly.

PS: Microsoft recently officially wrote "Sure, go ahead and shrink TEMPDB if you want to."

Bonus Note: the first methd works well on any DB - just replace the file identifiers "TEMPDEV" and "TEMPLOG" with whatever your specific DB uses. And if your DB has more than just a MDF and LDF, just add a new line of "SHRINKFILE" for each additional file.

Get SQL here

Note: The following snippets are based on a default TEMPDB, as is installed by default. If you have a custom TEMPDB config (multi-files, different names, etc) you will need to adjust below a bit.

```
/*
        Method one, try this first.
*/
        USE [TEMPDB];
GO
        DBCC FREESYSTEMCACHE('ALL');
GO
        DBCC SHRINKFILE (TEMPLOG,0);
        DBCC SHRINKFILE (TEMPDEV,0);
GO
```

Moving TempDB

If you need to move your TEMPDB, here is the simple way to do so.

Note: This script assumes you have a default TEMPDB setup.

WARNING: Use at own risk!
Make sure you enter a valid path with enough space for your typical TEMPDB use case.

```
/*
        Get the old TEMPDB name, and edit below if needed.
*/
        SELECT NAME, PHYSICAL_NAME AS CURRENTLOCATION
        FROM SYS.MASTER_FILES
        WHERE DATABASE_ID = DB_ID( N'TEMPDB' );
GO
        USE MASTER;
GO
        ALTER DATABASE TEMPDB
                MODIFY FILE ( NAME = TEMPDEV,
                FILENAME = 'D:\SQL_TEMP\TEMPDB.MDF');
GO
        ALTER DATABASE TEMPDB
                MODIFY FILE ( NAME = TEMPLOG,
                FILENAME = 'D:\SQL_TEMP\TEMPLOG.LDF');
GO

/*
        Restart SQL service.
        Check if it worked!
*/

        SELECT NAME, PHYSICAL_NAME AS CURRENTLOCATION, STATE_DESC
        FROM SYS.MASTER_FILES
        WHERE DATABASE_ID = DB_ID( N'TEMPDB' );

/*
        Finally: Delete tempdb.mdf and templog.ldf in the old location.
*/
```

Drop All Tables - SCRIPT

This just creates the script to drop either all tables or a filtered set, depending on our use case (some configuration required.)

The nice thing is the drop does not happen immediatly, so you can generate the DROP TABLE script for use in some future work.

WARNING: Use at own risk!
Before running the resulting script - Make a backup first!

```
/*
        Should work SQL 2005 and up.
*/
        SELECT
                'DROP TABLE [' + TABLE_SCHEMA + '].[' + TABLE_NAME +']'
        FROM
                INFORMATION_SCHEMA.TABLES;
GO
/*
        And you can filter, if you need to, here just an example:
*/
        SELECT
                'DROP TABLE [' + TABLE_SCHEMA + '].[' + TABLE_NAME + ']'
        FROM
                INFORMATION_SCHEMA.TABLES
        WHERE
```

```
                TABLE_SCHEMA IN ('DBO')
                OR TABLE_NAME LIKE 'F01%'
        ;
GO
```

Check DB UserMode

```
/*
        Run this in MASTER.
*/
        USE [master];
GO
        DECLARE @USERMODE as nvarchar(100);
        SET @USERMODE = (SELECT user_access_desc FROM sys.databases WHERE name = DB_NAME());
        IF @USERMODE = 'SINGLE_USER'
        BEGIN
                PRINT '['+DB_NAME()+'] is in single user mode! ['+@USERMODE+']'
        END
        ELSE
        BEGIN
                PRINT 'No, ['+DB_NAME()+'] is in mode ->['+@USERMODE+'].'
        END
GO
```

Backup All DB

One of the few times I use cursors.

This handy script can even run in a batch or job on a schedule, when a proper backup system is missing.

Edit - and uncomment - the ignore filter as required.

Get SQL here

```
/*
Source: Various
Usage: Update the @path variable to fit your Drive configuration.
Caveat: @path _HAS_ to be on the local SQL machine.
Does not support over the network backups without additonal config.
*/
USE [MASTER] ;
GO

DECLARE @name nVARCHAR(50) -- database name
DECLARE @desc nVARCHAR(50) -- desc
DECLARE @path nVARCHAR(256) -- path for backup files
DECLARE @fileName nVARCHAR(256) -- filename for backup
DECLARE @fileDate nVARCHAR(20) -- used for file name
-- specify database backup directory
SET @path = 'C:\SQLBACKUP\'  --- ' -- EDIT THIS! -----------------
-- specify filename format
```

```
SELECT @fileDate = CONVERT( VARCHAR(20), GETDATE(), 112 ) ;
DECLARE db_cursor CURSOR FOR
SELECT name
FROM master.dbo.sysdatabases

/*Uncomment the follwoing if you wish to exclude System Databases */
/* -- remove this line to uncommnet --
WHERE name NOT IN
(
'master','model','msdb','tempdb'
,'ReportServer','ReportServerTempDB'
,'stress_test','TEST'
)  -- exclude these databases
-- remove this line to uncommnet -- */

OPEN db_cursor ;
FETCH NEXT FROM db_cursor INTO @name ;
WHILE @@FETCH_STATUS = 0
BEGIN
        SET @fileName = @path + @name + '_' + @fileDate + '.BAK' ;
        SET @desc = @name + N' - Database Backup' ;
        select @desc , @fileName ;
        BACKUP DATABASE @name TO  DISK = @fileName
        WITH NOFORMAT, NOINIT,  NAME = @desc
        , SKIP, NOREWIND, NOUNLOAD, COMPRESSION,  STATS = 5 ;
        FETCH NEXT FROM db_cursor INTO @name ;
END
/* CLEANUP CURSOR */
CLOSE db_cursor ;
DEALLOCATE db_cursor ;

GO
```

TOP

List Failed Logins

Safe for anyone.

This simply reads the SQL LOGS - the last 4 sets in this case - and pulls out any mention of a failed login for the last 7 days.

You can easily adjust the code below to change how far back you want to list failed logins.

Note: You need to have Auditing Failed Logins enabled on your SQL server instance.

Get SQL here

```
/*
Note: You need to have Auditing Failed Logins enabled on your SQL server instance.
*/
USE MASTER ;
GO

DECLARE @LOGINLOG TABLE
(
        LOGDATE DATETIME,
        PROCESSINFO VARCHAR(10),
        TEXT VARCHAR(1000)
);
```

```
INSERT INTO @LOGINLOG EXEC SYS.SP_READERRORLOG 0, 1, 'FAILED', 'LOGIN' ;
INSERT INTO @LOGINLOG EXEC SYS.SP_READERRORLOG 1, 1, 'FAILED', 'LOGIN' ;
INSERT INTO @LOGINLOG EXEC SYS.SP_READERRORLOG 2, 1, 'FAILED', 'LOGIN' ;
INSERT INTO @LOGINLOG EXEC SYS.SP_READERRORLOG 3, 1, 'FAILED', 'LOGIN' ;
SELECT
        COUNT(*) AS [FAILED TIMES]
        , SUBSTRING(SUBSTRING(TEXT,CHARINDEX('''',TEXT)+1,LEN(TEXT)
                        -CHARINDEX('''',TEXT)),0,CHARINDEX('''',SUBSTRING(TEXT,
                        CHARINDEX('''',TEXT)+1,LEN(TEXT)
                        -CHARINDEX('''',TEXT)))) AS [LOGIN NAME]
        , DATEADD(DD, 0, DATEDIFF(DD, 0, LOGDATE)) AS LOGDATE
FROM
        @LOGINLOG
WHERE
        LOGDATE > DATEADD(DAY,-7,GETDATE())
GROUP BY
        SUBSTRING(SUBSTRING(TEXT,CHARINDEX('''',TEXT)+1,LEN(TEXT)
                -CHARINDEX('''',TEXT)),0,CHARINDEX('''',SUBSTRING(TEXT,
                CHARINDEX('''',TEXT)+1,LEN(TEXT)
                -CHARINDEX('''',TEXT))))
        , DATEADD(DD, 0, DATEDIFF(DD, 0, LOGDATE))
;
GO
```

TOP

List Table Triggers

Lists all the table triggers in a given DB.

Table triggers run in the same transaction as the triggering operation, so knowing what triggers are on your DB is essential is being aware of blocking risks.

Something most keep forgetting, is that TRUNCATE TABLE does nto fire a TRIGGER.
That can be a real pain if you are running a form of CDC that uses triggers...

```
/*
        Change the USE to the DB you want to inspect.
*/
        USE [myDB];
GO
        SELECT
                sysobjects.name AS trigger_name
                , USER_NAME(sysobjects.uid) AS trigger_owner
                , s.name AS table_schema
                , OBJECT_NAME(parent_obj) AS table_name
                , OBJECTPROPERTY( id, 'ExecIsUpdateTrigger') AS isupdate
                , OBJECTPROPERTY( id, 'ExecIsDeleteTrigger') AS isdelete
                , OBJECTPROPERTY( id, 'ExecIsInsertTrigger') AS isinsert
                , OBJECTPROPERTY( id, 'ExecIsAfterTrigger') AS isafter
                , OBJECTPROPERTY( id, 'ExecIsInsteadOfTrigger') AS isinsteadof
                , OBJECTPROPERTY(id, 'ExecIsTriggerDisabled') AS [disabled]
        FROM sysobjects
        /* -- Uncomment this is you want to add fields from sysusers - might b useful in you
                INNER JOIN sysusers
                        ON sysobjects.uid = sysusers.uid
        */
        INNER JOIN sys.tables t
                ON sysobjects.parent_obj = t.object_id
        INNER JOIN sys.schemas s
```

```
                ON t.schema_id = s.schema_id
        WHERE sysobjects.type = 'TR';
GO
```

TOP

Compression Generator

I like using SQL generators. This is related to dynamic SQL, but more controlled. Copying the result into a new query window gives me a chance to check the code before I run it. WIth dynamic SQL, the execute happens immediatly, and depending on what the script does, could be catastrophic.

Also, one can pre-generate the code and include it in a CHANGE REQUEST for use in both development and production environments (assuming you followed best practice, and refreshed your non-prod ENV from PROD, first, to make sure they are identical...)

Get SQL here

```
/*
This is a script generator.

Set the output to TEXT, then copy the result ot a new query window.

Add a where clause to the filter on tables or schmema, as required.

Switch the compression type to ROW from PAGE as required...
*/

SELECT
'ALTER TABLE ' + s.name + '.' + t.name + ' REBUILD PARTITION = ALL  WITH (DATA_COMPRESSION =
GO
'
FROM sys.tables AS t INNER JOIN sys.schemas AS s
ON t.schema_id = s.schema_id ;

GO

/* Handy script to list compressions */

SELECT
st.name
, ix.name
, st.object_id
, sp.partition_id
, sp.partition_number
, sp.data_compression
, sp.data_compression_desc
FROM sys.partitions SP
INNER JOIN sys.tables ST
ON st.object_id = sp.object_id
LEFT OUTER JOIN sys.indexes IX
ON sp.object_id = ix.object_id AND sp.index_id = ix.index_id
WHERE sp.data_compression <> 0
ORDER BY st.name, sp.index_id ;

GO
```

TOP

GO?

GO

TOP

GO?

GO

TOP

GO?

GO

TOP