DBCC CHECKDB

Use and Abuse

By Richard Fryar

This description of DBCC CHECKDB and how to use it is an update of an article I wrote on SQL Server Pro several years ago.

When I first wrote this article I was in the process of recruiting a senior DBA and I was surprised at the lack of knowledge about database corruption and how to deal with it!

Most candidates were aware that they should run DBCC CHECKDB to check for corruption, but most also thought that the solution should be to run it again with one of the repair options! This is dangerous, as it can cause data loss.

So here is how to use DBCC CHECKDB, and what to do when you have database corruption.

So How Do I Use It?

The primary purpose is to check for consistency errors, and should ideally be run every day.

The basic syntax is:

```
DBCC CHECKDB ('YourDatabase') WITH NO_INFOMSGS
```

NO_INFOMSGS prevents an excessive number of informational messages from being generated. There are several other options, but this is the syntax you should aim to use as it performs all integrity checks.

This may take a long time on large databases and you may want to specify the PHYSICAL_ONLY option. This checks physical on-disk structures, but omits the internal logical checks. The syntax is:

```
DBCC CHECKDB ('YourDatabase') WITH PHYSICAL_ONLY
```

It Has Found A Problem - What Do I Do?

This article is about the approach to take to recover the corrupt data, but don't forget to investigate the cause. Corruption, thankfully, is very rare but make sure your disks and/or SAN are checked thoroughly to identify the cause. It is also worth looking at the Windows event logs - sometimes you will see an error message that helps your investigation.

By far the best option for fixing the corruption is to restore from a backup, but let's look at how you investigate which pages are affected and what type of data is affected:

Look at the output from DBCC CHECKDB. You may see something like this:

```
Object ID 2088535921, index ID 0, partition ID 72345201021503994, alloc unit ID 72345201051571606 (type In-row data): Page (1:94299) could not be processed. See other errors for details. Msg 8939, Level 16, State 98, Line 1 Table error: Object ID 2088535921, index ID 0, partition ID 72345201021503994, alloc unit ID 72345201051571606 (type In-row data), page (1:94299). Test (IS_OFF (BUF_IOERR, pBUF->bstat)) failed. CHECKDB found 0 allocation errors and 2 consistency errors in table 'yourtable' (object ID 2088535921). CHECKDB found 0 allocation errors and 2 consistency errors in database 'yourdb'. repair_allow_data_loss is the minimum repair level for the errors found by DBCC CHECKDB (YourDatabase).
```

From this you can see which page is corrupted (1:94299)

The first thing to do is check if it is data in a heap, in a clustered index, or in a non-clustered index. In the above text you can see it is index ID 0. You could also examine the page (1:94299 in database 'YourDatabase') as follows:

```
DBCC TRACEON (3604, -1)
GO
DBCC PAGE('YourDatabase', 1, 94299, 3)
GO
```

In the output you will see something like:

Metadata: IndexId = n

If n is greater than 1 it is a non-clustered index and can safely be dropped and recreated. If n is 0 or 1 you have data corruption and need to perform one of the options described below.

Restoring from a backup

If the recovery model is FULL (or BULK_LOGGED, with some limitations), you can backup the tail of the log, perform a restore (with norecovery) from the last clean full backup, followed by subsequent log backups and finally the tail of the log.

If only a few pages are affected you have the option of selectively restoring only the bad pages, as follows:

```
RESTORE DATABASE YourDatabase PAGE = '1:94299' FROM DISK = 'C:\YourDatabase.bak' WITH NORECOVERY
```

If the recovery model is simple you don't have that option, and have to accept that a restore from the last full backup will result in subsequent transactions being lost. In this case, or if you have no backups at all, you may decide that an automatic repair is the only option.

Automatic Repair Options

First let me emphasise the importance of running a backup BEFORE you go any further.

Have a look at the output of the original CHECKDB. It will specify the minimum repair level.

REPAIR_REBUILD

If the minimum repair level is REPAIR_REBUILD you have been lucky.

The syntax is:

```
DBCC CHECKDB('DB Name', REPAIR_REBUILD)
```

REPAIR_ALLOW_DATA_LOSS

This attempts to repair all errors. Sometimes the only way to repair an error is to deallocate the affected page and modify page links so that it looks like the page never existed. This has the desired effect of restoring the database's structural integrity but means that something has been deleted (hence the ALLOW_DATA_LOSS). There are likely to be issues with referential integrity, not to mention the important data that may now be missing.

The syntax is:

```
DBCC CHECKDB('DB Name', REPAIR_ALLOW_DATA_LOSS)
```

Make sure you run DBCC CHECKCONSTRAINTS afterwards so you are aware of referential integrity issues and can take the appropriate action.

And Finally

My original reason for writing this was to stress that the correct action when faced with corruption is nearly always to restore from a backup. Only use the automatic repair options as a last resort, and with full understanding of the damage this may do.

Just as important is that regular backups are an essential part of a DBA's responsibilities, and you should use the FULL recovery model with regular log backups for all but the most trivial databases.

DBCC CHECKDB is a powerful tool, but also very dangerous in the wrong hands.

Maybe instead of adding the REPAIR_ALLOW_DATA_LOSS option, Microsoft should have created a separate DBCC command called:

```
DBCC DELETE_DATA_TO_FIX_CORRUPTION
```

A bit wordy, but DBAs would be under no illusion about exactly what damage they could be doing, and it may make them think twice before running it!