Steve Stedman
Freelance SQL Server Consultant
http://stevestedman.com

Database Corruption Challenge #2 - Alternate Solution

With week two of the Database Corruption Challenge being just as interesting as week one, and after seeing yesterdays blog posting showing the solution by Rob Farley, you might be thinking you are ready to take on corruption. If you are interested in giving the corruption challenge a try, please sign up for my newsletter so that you will be notified of the next corruption challenge.

Lets take a look at one of the more interesting solutions submitted by Bogdan Sahlean, who was the first to solve using DBCC Page to find the exact corrupt bytes, and DBCC WritePage to fix those corrupt bytes.

Now if you are reading this, and don't know about DBCC WritePage, I caution you... DBCC WritePage is not something you should just play with, and it is certainly not something that you should ever run on a production database. Consider yourself warned. One of the dangers of using Write page that it's use will make the DB unsupported if Microsoft support is required.

However after all the warnings about DBCC WritePage are out of the way, Bogdan Sahlean has come up with a really slick solution that just fixed the 8 bytes of corruption that exist in the database. The same solution could be achieved by editing the database with a hex editor, and that solution was provided by another participant. Lets take a look at this solution. First assume that the latest backup provided has been restored to a database called CorruptionChallenge2_Latest_SGBD.

He starts out by finding out what is corrupt:

Which results with this output:

```
Msg 8939, Level 16, State 98, Line 40
Table error: Object ID 2137058649, index ID 1, partition ID 72057594038845440, alloc unit ID 72057594039762944 (type In-row data), page (1:244). Test (IS_OFF (BUF_IOERR, pBUF->bstat)) failed. Values are 133129 and -10.
Msg 8928, Level 16, State 1, Line 40
Object ID 2137058649, index ID 1, partition ID 72057594038845440, alloc unit ID 72057594039762944 (type In-row data): Page (1:244) could not be processed. See other errors for details.
Msg 8976, Level 16, State 1, Line 40
Table error: Object ID 2137058649, index ID 1, partition ID 72057594038845440, alloc unit ID 72057594039762944 (type In-row data). Page (1:244) was not seen in the scan although its parent (1:157) and previous (1:243) refer to it. Check any previous errors.
Msg 8978, Level 16, State 1, Line 40
```

Table error: Object ID 2137058649, index ID 1, partition ID 72057594038845440, alloc unit ID 72057594039762944 (type In-row data). Page (1:245) is missing a reference from previous page (1:244). Possible chain linkage problem.
CHECKDB found 0 allocation errors and 4 consistency errors in table 'Revenue' (object ID 2137058649).
CHECKDB found 0 allocation errors and 4 consistency errors in database 'CorruptionChallenge2_Latest_SGBD'.
repair_allow_data_loss is the minimum repair level for the errors found by DBCC CHECKDB (CorruptionChallenge2_Latest_SGBD).

Bogdan noted

Important info (from my point of view):
Page (1:244) was not seen in the scan although its parent (1:157) and previous (1:243) refer to it.

So he takes a look at what is in the revenue table:

SQL Server detected a logical consistency-based I/O error: invalid protection option.
It occurred during a read of page (1:244) in database ID 7 at offset 0x000000001e8000
in file 'D:\BD\Corruption_Challenge\week2\CorruptionChallenge2_LatestBackup.mdf'.
Additional messages in the SQL Server error log or system event log may provide more detail. This is a severe error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.

Bogdan noted:

Important info (from my point of view):
It occurred during a read of page (1:244) in database ID 7 at offset 0x000000001e8000

Now where the code gets interesting...

So then he starts looking around at the pages to find the problem.

Where he makes this discovery:

```
 1:243 = 01010400 00020001 f2000000 01001400 f4000000   1:244 = 020003ff
ffff0001 f3000000 01001400 f5000000   1:245 = 01010400 00020001 f4000000
01001400 f6000000   1:246 = 01010400 00820001 f5000000 01001400 9f000000
          |---------------|            ^            |-- It seems that fir
st 8 bytes from 1:244                 are "bad" and it  should be "010104
00 00020001"                  instead of "01010400 00020001"
```

Then the fix is made:

The DBCC WRITEPAGE command is writting 8 bytes into page 244 at position 8 to fix the corruption.

Next DBCC CheckDB() comes back clean and the sproc that tests the databse called checkCorruptionChallenge2Result reports success.

Nice job Bogdan Sahlean for finding the 8 specific bytes in the entire database that were corrupt and fixing just those 8 bytes. No data loss, and the database now reports that there is no corruption.

I would again like to reiterate my warning about using DBCC WritePage in production systems. I would certainly NEVER Use this without being absolutely sure that there was a really solid backup. DBCC WritePage is like the voodoo of SQL Server, but sometimes it does the job.