Solution to Week 10 Part 1 Database Corruption Challenge

If you are looking for more info on the corrupt database with Week 10 of the Database Corruption Challenge, you can take a look at the week 10 Challenge Details.

With this being the final competition of the Database Corruption Challenge, I had to make it more difficult than the others, so I added 3 different types of corruption. This challenge was about the equivalent of any 3 of the other challenges.

I have split the solution to the Week 10 Challenge into 3 different posts, with each post explaining how to fix one of the three specific corrupt areas.

Corruption in the Orders Table

WARNING: DBCC WritePage is a dangerous command, that should never be used on any production database. It may invalidate your ability to get support from Microsoft on issues that arise with that database going forward. It is not my intention to encourage anyone to use DBCC WritePage ever. This is just what I used to create a corrupt database, and since creating corrupt databases is not part of the role of most DBAs, you should not use DBCC WritePage. Consider yourself warned.

The corruption in the orders table was unlike any of the other corruption challenges. This was something new. For details on the way that a table scan uses a linked list of pages see my blog post on SQL Server Tables – BTree or Linked List.

Lets start with how I created the orders table and made it corrupt.

I then added rows and deleted rows, until I ended up with a the orders table having 59019 rows in it.

My goal was to corrupt the table in a way that it would create an infinite loop when doing a table scan (or clustered index scan) on the orders table. Here is how I did that.

First I used

First I used DBCC IND to get a list of the pages in the orders table (clustered index).

Next I used DBCC page to pick a page somewhere near the beginning of the table, so I chose page 2323. Using DBCC Page, I looked up the m_nextPage and m_prevPage pointers, and

recorded them. I then modified the pointer to the next page to point to the previous page creating the infinite loop, or infinite query.

At this point it turned out that the query wasn't entirely infinite. After a while my SQL Server ran out of memory and the query stopped at about 33 million rows.

Here I realized that it would be too easy, all you needed to do at this point was just select in the opposite direction and you could get all the data out.

Being the final week of the challenge, I couldn't leave it at just that, instead, I had to mess with one of the m_prevPage pointers near the end of the table to cause a similar loop, so I did the following:

Yes... at this point you were unable to SELECT * from the table with either ASC or DESC.

So to make it a little worse, I decided to mess with another page somewhere in the middle of the table, and overwrite the page header (similar to an early challenge). So I stomped on another page, this time not touching the prev page or next page pointer.

I then backed up the database and we have a corrupt database with the orders table.

Fixing the Corruption - DBCC WritePage

So now to fix the corruption. Some people were able to track down the problem, and just used DBCC WritePage to the same addresses to fix the corruption. Those solutions looked something like this solution from Rob Farley:

Many solved the challenge with a similar solution to this. There were some however who solved the corruption by reading each page, and reconstructing the table, specifically Patrick Flynn who had a solution very similar to his solution to the Week 7 Challenge.

Not everyone used DBCC WritePage to solve the challenge this week, however many did.

I hope this helps get a better understanding Fixing the Corruption in a bizarre corruption example in the Database Corruption Challenge.

Related Links

- Current Scores
- Newsletter Sign up
- Database Corruption Challenge T-Shirt
- Week 1, Week 2, Week 3, Week 4, Week 5, Week 6, Week 7, Week 8, Week 9, Week 10
- Slow DBCC CheckDB
- Database Corruption Worksheet

Database Corruption Challenge Sponsored by