Week 7 Database Corruption Challenge Results

The seventh week of ten in the Database Corruption Challenge went well. There were a few new participants, and many of those who have competed week over week. The first to submit a correct answer this week was Raul Gonzalez.

What was interesting about this challenge was that it was not your typical corruption where you could just run DBCC CheckDB and find the problem. To understand exactly what happened, solving this required reading through the transaction log that had not yet been backed up and finding the specific problem.

The original post describing the challenge stated the following:

> Your goal if you choose to accept it is to get restore all the data in the [OPEN_NFIRS].[Record1000] table prior to its disappearance. Note: there may have been multiple DBA's working on this system in several different time zones across the world.

Which contained some valid information, and some distracting information. First the distracting part, the fact that the DBAs working on the solution were from multiple time zones across the world. This had nothing to do with anything, and some people asked about it. That statement were purely there to distract. The key items to take away from this comment were:

- Multiple DBAs (not necessarily at the same location).
- The table contents just disappeared.

There was another key clue in the original post...

> Not all missing data is caused by hardware failures or issues, sometimes it is caused by people. – Unknown DBA

Where I was going with this was that the table appeared to be "corrupt", but after initial investigation it could be discovered that someone who was logged in as the user 'sa' ran a delete command that possibly forgot a where clause, and dumped the entire [OPEN_NFIRS].[Record1000] table. There was an extra point offered for those who figured that part out.

So once you figured that part out all you had to do was the following:

- Take a tail of the log backup from the 3 database files included in the challenge.

- Figure out the specific time or LSN of the delete statement from the log.
- Restore the provided full backup.
- Restore the provided log backups.
- Restore the tail of the log backup just up to the point in time before the DELETE statement was run.

The following solution, all the code from Raul Gonzales, and the commentary and screen shots are from me. I changed a couple of paths to match my configuration when testing this:

The database attaches just fine, and upgrades the files to the current version of SQL Server.

Next take a look to see what the recovery model is on the database. This will determine if the tail of the log backup may be available.

I like how Raul included his results as comments in the code. This allows me to confirm that I see the same thing that he did going through the process.

So yes, the database CorruptionChallenge7 is in FULL recovery model. Lets take a look at the transaction log.

Yes, loads of things in the transaction log, 56591 to be exact.

So now its time to create a tail of the log backup.

At this point we at least know that using backups we can restore to the exact point in time that we

are at now. We also know that we can restore to any point along the way if we know where to stop.

So next we run the final check query that needs to be submitted with the solution to see what the table looks like, and we select everything out of the [Record1000] table.

Which shows there are no rows in the table.

So where did the rows go?

Restore all the transaction logs without the tail of the log backup.

Now lets take a look at how things look.

No corruption remaining, so lets look at the data.

Showing 1010 rows, and some checksums. But it can be that simple...

Using the fn_dup_dblog function to scan the tail of the log transaction log backup to find the interesting DELETE statement, which occurred at 17:57:56:278.

So lets look at what happened after that DELETE statement.

Scrolling through the list after the DELETE, no other commands, just SQL Server doing its thing to cleanup after the DELETE statement. So now we know everything that is needed to restore to the exact point in time (or just before it). Run the same list of transaction log backups we did before, but add in the tail of the log backup and tell it to stop at the DELETE statement.

The tail of the log backup is restored to just before the DELETE statement.

So now run the final checksum query that needed to be submitted as proof of completion.

And that's how Database Corruption Challenge #7 was won by Raul Gonzalez.

Not all "corruption" is what you traditionally think of as corruption.

Raul received 4 points this week:

- One point for being first to complete the challenge correction.
- Two points for solving it before any additional clues where given.
- One point for providing that the DELETE was performed by the user 'sa'.

Almost all of the winning solutions this week were very similar to this, however some used the LSN on the final restore, rather than the timestamp. Those solutions looked something like this:

The solutions using the LSN required converting it from hex to decimal, which isn't exactly obvious. I created a function to do the LSN conversion, and included it in the extra clue this week. Another participant Dan Andrei Stefan provided an alternate function to do the same thing. His function was the inspiration to create my own function to do the conversion. Here is my function.

There was one solution provided by Patrick Flynn which restored all of the data without using any of the backups. He only used the 3 database files (MDF, LDF, NDF). I will post this solution tomorrow, so check back to see how he did it.

So that wraps up Database Corruption Challenge #7. I hope everyone enjoyed this weeks competition.

-Steve Stedman

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
- Slow DBCC CheckDB
- Database Corruption Worksheet

Database Corruption Challenge Sponsored by