ITPro with deep vertical Knowledge

RECENT

PRACTICAL SQL SERVER

SQL Server Database Corruption, Part XII: Recovery Sample

overachiever | May 31, 2012

In my last post in this ongoing series on SQL Server database corruption I mentioned that my next post would be to provide a 'soup to nuts' sample or example of how you can test corruption and recovery in your own environment – as a means of better getting familiar with exactly what corruption is, what it looks like, and how to address it.

Simulating Corruption

To that end, this post closely follows on a previous post where I showcased, in step-by-step fashion, how to simulate corruption in a SQL Server database by using a hex editor to go in and 'hack' some of the raw bits of data stored in a copy of an AdventureWorks database.

Order of Operations

Of course, that previous post just showcases how to 'trash' a SQL Server database – and doesn't show how to recover from that 'trashing'. As such, if you'd like to walk through the process of 'recovering' from corruption (in a simulated environment or setting), here's what you'll want or need to do.

1. Start with a fresh/non-trashed copy of AdventureWorks.

2. Make sure that the DB is in FULL recovery.

3. Clean out msdb.dbo.suspect_pages.

4. Create a new FULL backup of your AdventureWorks database.

5. Create a t-log backup as well. (You'll be using these to work through the recovery process.)

6. Force your SQL Server down (using the SHUTDOWN WITH NOWAIT command). And, yes, obviously, you DON'T want to do this on a production server.

7. Open up the AdventureWorks.mdf file with a Hex Editor – as outlined in this post and 'hack' some of the values in the 02177190 offset range by setting the 'data' in this range of 'trashed' space to all 0's or whatever floats your boat.

8. Save your Hex-Edited changes.

9. Restart SQL Server.

10. Zip in to the AdventureWorks database and run a sample query such as SELECT * FROM sysobjects.

11. Then check for corruption using DBCC CHECKDB(AdventureWorks) WITH NO_INFOMSGS, ALL_ERRORMSGS.

12. Review the error messages, and then get a list of which pages were 'taco-ed' by your corruption efforts by executing: SELECT * FROM msdb..suspect_pages.

13. Start the Recovery Process by switching to the master database.

14. Execute a page-level restoration operation by means of running the RESTORE DATABASE command against AdventureWorks with the PAGE option specified. (Where you'll want to target the list of all pages identified in the list of pages returned by querying msdb's suspect_pages.

15. Make sure that your first recovery operation specifies WITH NORECOVERY.

16. Then apply your transaction log backup as well – and (once all t-log backups that you've made have been applied) go ahead and RECOVER your database. (And the reason I say 'all' t-log backups is because my hope/goal is that you'll poke around with this list of operations and EXPAND upon the minimalistic scenario I've provided here to try some things on your own – including marshaling changes in to the db and watching what happens when you have (or don't have) different t-logs on hand to try and 'recover' from whatever evil you've managed to perpetrate on your test db.)

17. Run DELETE FROM msdb..suspect_pages to clear out all previously logged suspect pages.

18. Run DBCC CHECKDB(AdventureWorks) WITH NO_INFOMSGS, ALL_ERRORMSGS to ‘clear’ the database and make sure that all corruption was checked. (And note that in a production setting you MIGHT want to leave the db in SINGLE_USER/RESTRICTED_USER and make this check BEFORE releasing the db back for general-usage (though doing so IS a size-of-data operation and can add some appreciable down-time into the mix – so only do this if you’re SUPER paranoid/concerned about the negatives that could ensue from not correcting corruption before releasing your db into usage).)

19. Re-truncate/delete suspect_pages in msdb – as ‘cleanup’ of this table is always 100% on you as the DBA and you therefore don’t want any ‘lingering’ pages left in this table the next time you run a DBCC CHECK.

Taking this Process for a Test Drive

Of course, the list of operations I’ve provided above are a BARE MINIUMUM set of steps that you’ll want or need to get your hands dirty with corruption. As such, I’m hoping that you’ll run through the steps listed above until you get comfortable with what’s going on. Then I’m hoping you’ll get a bit reckless and start ‘adding’ to the steps above by playing around with many of the different questions you’re bound to have about what happens when you do X or Y or don’t have Z and so on. Because, again, the whole point of this post is to provide you with a ‘skeleton’ framework (if you will) that will let you try out dealing with corruption in a controlled environment – so that you’re THAT much more comfortable and capable of dealing with it in production WHEN it occurs as corruption is always a case of WHEN not IF.

And, to that end, I realize that some of the steps outlined above MIGHT be a bit cryptic if you’re just getting started with SQL Server or haven’t played with some of the commands, options, arguments listed above. As such, what follows is a full-blown script that I’ve used to demonstrate the whole process outlined above in previous demos/presentations:

```
-- set recovery to FULL:
ALTER DATABASE AdventureWorks
SET RECOVERY FULL
GO

-- PURGE suspect_pages as part of the demo:
TRUNCATE TABLE msdb..suspect_pages
GO

-- Get a new Backup:
BACKUP DATABASE AdventureWorks TO DISK = 'D:\SQLBackups\AdventureWorks.BAK'
GO

-- and a t-log backup:
BACKUP LOG AdventureWorks TO DISK = 'D:\SQLBackups\AdventureWorks.TRN'
```

```
BACKUP LOG AdventureWorks TO DISK = 'D:\SQLBackups\AdventureWorks.TRN'
GO

-- Turn SQL Server off to start simulating corruption:
SHUTDOWN WITH NOWAIT

-- Use a hex editor (HxD) and go, for example, to offset 02177190
--      to the AW00xxx keys, replace a bunch of them with 000000s.

-- Then restart SQL Server

-- show that the DB is online...
SELECT * FROM AdventureWorks..sysobjects

-- check for corruption:
DBCC CHECKDB(AdventureWorks) WITH NO_INFOMSGS, ALL_ERRORMSGS

-- Talk through demo and stuff... about how to repair/recover and so on.

-- then, talk about backups and stuff (show that we took them earlier)

-- Get a report on Corruption:
SELECT * FROM msdb..suspect_pages


-- Recover:
USE master
GO

ALTER DATABASE AdventureWorks SET RESTRICTED_USER
GO

-- TODO: replace the page #(s) here with data from suspect_pages.
RESTORE DATABASE AdventureWorks
PAGE = '1:xxxx'
FROM DISK = 'D:\SQLBackups\AdventureWorks.BAK'
WITH
        NORECOVERY
GO

-- previous log (from before corruption):
RESTORE LOG AdventureWorks
FROM DISK = 'D:\SQLBackups\AdventureWorks.TRN'
WITH
        NORECOVERY
GO

-- CURRENT log:
BACKUP LOG AdventureWorks
TO DISK = 'D:\SQLBackups\AdventureWorks_TAIL.TRN'
GO
```

```
RESTORE LOG AdventureWorks
FROM DISK = 'D:\SQLBackups\AdventureWorks_TAIL.TRN'
WITH
        RECOVERY
GO

-- now query it:
SELECT * FROM AdventureWorks..sysobjects
GO

-- and check it:
DBCC CHECKDB(AdventureWorks) WITH ALL_ERRORMSGS, NO_INFOMSGS
GO


-- bring it back online for users:
ALTER DATABASE AdventureWorks SET MULTI_USER
GO

-- and truncate/cleanup suspect_pages:
TRUNCATE TABLE msdb..suspect_pages
GO
```

Just make sure to pay attention to the comments and follow all instructions and you should be fine.

Part XIII: Recap

TAGS: MICROSOFT SQL SERVER SOFTWARE DEVELOPMENT

2 COMMENTS

```
SELECT STRING_AGG(CHAR(n), '')
  WITHIN GROUP (ORDER BY CHAR(n) COLLATE Latin1_General_CS_
FROM dbo.GetNums(0, 255)
WHERE CHAR(n) COLLATE Latin1_General_CS_AS LIKE '[A-Z]';


SELECT STRING_AGG(CHAR(n), '')
  WITHIN GROUP (ORDER BY CHAR(n) COLLATE Latin1_General_BIN
FROM dbo.GetNums(0, 255)
WHERE CHAR(n) COLLATE Latin1_General_BIN LIKE '[A-Z]';
```

BUSINESS INTELLIGENCE > MICROSOFT SQL SERVER

The Challenge Is on: Maintain a Custom Letter-Based Sequence

If you need to implement a custom sequence solution, and you want it to perform as well as possible, you need to ask yourself which guarantees the solution needs to provide versus which it doesn't.

Itzik Ben-Gan | Apr 01, 2018

Recently I delivered an advanced T-SQL class in Europe, and one of my students, Nicolas Barreau of Avanade France SAS, presented a nice little puzzle. The task wasn't very complex, but it was

interesting and fun to work on, and I hope you will enjoy working on it, too.

The Challenge

The challenge Nicola presented was to maintain a custom, letter-based, cycling, gap-allowing, sequence, with values in the form LLL, where L is an alphabet letter in the range A through Z (26 symbols). Nicola's company uses the sequence value as a match flag for settlement, for instance, between payment and invoice. The first value produced should be AAA, and it should advance as if the letters were digits in base 26, like so:

AAA

AAB

READ MORE

TAGS: SOFTWARE DEVELOPMENT

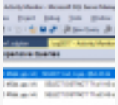
0 COMMENTS

RELATED

The Challenge Is on: Multi-Way Number Partitioning with T-SQL and SQL CLR
FEB 21, 2018

SQL Server: Avoiding a Sort with Descending Order
JAN 19, 2018

SQL Server: Avoiding a Sort with Merge Join Concatenation
DEC 18, 2017



T-SQL Puzzle Challenge: Grouping Connected Items
NOV 27, 2017

Sign up for the ITPro Update newsletter.