ITPro with deep vertical Knowledge

PRACTICAL SQL SERVER

SQL Server Database Corruption, Part IX: Responding to Corruption

overachiever | Apr 14, 2012

In a previous post we looked at the importance of backups when it comes to dealing with logical (human or application caused) or physical (IO subsystem caused) corruption. In this post we'll look at some best practices for responding to physical corruption when it happens or is detected.

Responding to Corruption When it Happens

When corruption is encountered, there's a definite order of operations to follow as a means of both decreasing the potential for data loss and as a means for decreasing down-time while remedying the situation. That said, all forms of corruption are NOT created equally and are NOT the same in potential scope or impact.

For examples, sometimes corruption occurs within typical data pages that store the actual data needed by your database to keep it running and viably storing data. Or, at other times, corruption might simply occur within an index or other simple structure that is effectively an 'ancillary' feature of your database – meaning that you MIGHT be able to recover from this form of corruption by merely dropping the index and recreating it (with effectively no down time and only a brief 'disruption' in the form of the locks/blocks necessary to replace the index). Yet, at other

times, corruption COULD happen within key (allocation) pages used by the database to MAP where your data, indexes, or other KEY structures are found – and in cases where corruption occurs in these highly specialized pages (such as page 9 – the boot page), this corruption can be FATAL – meaning that there is NO option for recovery other than to revert to backups.

As such, the following, ordered, list outlines key considerations to address as a means of 'escalating' through available options in order to try and minimize the scope and down-time associated with recovering from corruption.

1. **No, REALLY: Don't Reboot Your Server.** While some weird Windows issues will go away by rebooting the box, the presence of corruption will NOT go away if you reboot your server. Instead, it's VERY possible that **a reboot will actually make things MUCH worse** as it may act as the means for causing SQL Server to take your database offline and render it SUSPECT – making it MUCH harder to deal with than it might otherwise be.

2. **Do NOT Attempt to Detach / Re-attach Your Databases.** As with the previous bullet point, this is an absolute worst practice (no matter WHAT you might have read in some forums online). Detaching and re-attaching your database when corruption is present is almost certainly going to make recovery MUCH harder.

3. **Determine the Nature and Scope of the Corruption BEFORE Attempting ANY Repairs.** If you're getting IO errors (823, 824, 825 errors) from SQL Server, then you may be at the cusp or leading edge of running into problems with corruption. As soon as possible, run DBCC CHECKDB([yourDbName]) WITH NO_INFOMSGS, ALL_ERRORMSGS as soon as possible to see if there are problems. Or, if you've found problems by means of regular checks or backups, make sure to evaluate ALL of the error messages returned and document or save them to a safe location somewhere (i.e., copy/paste to notepad or save as a .sql file as needed).

4. **Consider re-running DBCC CHECKDB if there is Only a Single or Minor Problem Reported**. Yes, this sounds lame and even superstitious, BUT in some **_very_** rare cases if you've just encountered a minor bit of corruption (i.e. just one or two errors) you can actually re-run DBCC CHECKDB and the corruption will have disappeared – simply because SQL Server wrote to a bad spot (or the 'write' was 'mangled') and SQL Server was able to 'recover' transparently, and then write to a new location, or execute a write that 'completed correctly'. Such scenarios are VERY rare but do happen. Care should be taken in these kinds of scenarios too to watch for a pattern – because while one instance of corruption here or there is sadly to be allowed/tolerated/expected, multiple problems over a short period of calendar time typically indicate a pattern of systemic problems and you need to start contemplating hardware replacement as needed. Otherwise, IF you want to take a chance that this is going on and you're working with a small database, the factors to consider here are that if you can re-run checks quickly, it may be in your best interest to TRY this option – but don't HOPE too much for problems to magically disappear. AND, the upsize of 'getting lucky' with this seemingly feeble attempt is that you ALWAYS want to 100% size up the specifics of corruption whenever it occurs – as in think of the old adage: "measure twice, and cut once".

5. **Size Up Corruption and Remediation Options BEFORE doing ANYTHING.** Otherwise, when you do run into problems, make sure to WAIT until DBCC CHECKDB() has successfully completed. There are simply too many horror stories out there about DBAs who 'jumped the gun' at the first sign of trouble and either DESTROYED data unnecessarily or who INCREASED down-time by responding to SYMPTOMS of corruption instead of root-causes. Accordingly, make sure you review EVERY single error after SQL Server completes reporting about errors because in some cases the 'screen' may be chock-full of red error messages about

corruption here and there in various pages, but you might find out at the end of the check that almost all of these errors are caused by a non-clustered index that can be simply dropped and recreated.
So don't jump the gun.
Likewise, some forms of corruption are simple issues where the meta-data used by SQL Server to keep track of free space and other 'internals' get a bit out of 'sync' with reality – and in these cases SQL Server may tell you that you can safely just run DBCC UPDATEUSAGE() or another operation to CLEANLY and EASILY restore from corruption without incurring any downtime OR losing any data. So, again, size up your options accordingly – and after you have all of the facts.

6. **Validate ANY Repair Options in a Test Environment First.** If possible… [blah blah]. Unless you spend gobs of time working with Corruption, various modes and types of repair are GOING to be things that you're going to potentially cut your teeth on – meaning that you'll probably learn some GREAT and VALUABLE lessons the HARD way. So why do that on your PRODUCTION data? Spin up a copy of your database somewhere, and practice/validate your intended changes there. Yes, this will probably take longer in most cases than just executing your changes in production. Then again, if you screw something up in production, THEN which approach ends up taking longer? (Personally, I ALWAYS approach every recovery operation in looking for any options that leave me with INCREASED fall-back options and capabilities as I move forward with remediation efforts.)

7. **Escalate to DBCC CHECKDB() and REPAIR_REBUILD IF Needed.** In many cases when corruption is minimal, SQL Server might inform you that the REPAIR_REBUILD option may be a viable approach to recovering data. If this is the case, or if you just want to 'check' and see if it will work, you can safely run DBCC CHECKDB([yourDBName]) WITH REPAIR_REBUILD **with no worries of data loss**. The only thing you stand to lose would be **TIME** – meaning you MUST put the database into Single User mode to execute this option. So, if you think this has a potential to correct your errors, it's a viable approach. If SQL Server indicated something more severe (that requires the use of backups or repair options that require data loss) then running this will JUST waste time.

8. **Execute Page-Level Restore Operations if Possible.** If you've got full-blown corrupt data within a few pages (as opposed to being in indexes that could be recreated), then you'll be able to spot those by querying msdb's suspect pages table, like so:

```
SELECT * FROM msdb..suspect_pages
GO
```

Then, from there, it's possible to effectuate a page-level restore from SQL Server using your existing backups. And what this means is that you'll instruct SQL Server to 'reach in' to previous backups, 'grab' the data from the pages that were corrupt, and then REPLACE the corrupted pages with known-safe pages from your backups. More importantly, since any operations since your last backup will also have been logged (assuming FULL recovery and regular FULL/DIFF + Transaction Log backups), you'll be able to 'replay' any changes against that data by means of replaying the data in the transaction log. (For more information on how this works, see the following video.) As such, make sure to back up the 'tail end' of your transaction log BEFORE beginning this operation.
NOTE: *I'll follow up with an EXAMPLE of how to pull this off in a subsequent post.*

9. **Execute a Full Recovery.** If there are large numbers of suspect/corrupted pages (i.e., so many that manually recovering each one would take longer than a full recovery) or if critical system pages have been destroyed, then you'll need to execute a full recovery. As with the previous operation, make sure you commence this operation by backing up the tail end (or non-backed up part) of your current transaction log – to ensure that you don't lose any

operations that haven't been backed up.
NOTE: *I'll ALSO follow up with an EXAMPLE of how to do this (along with best-practices/guidance) in a subsequent post as well.*

10. **STOOP To Using DBCC CHECKDB and REPAIR_ALLOW_DATA_LOSS if ALL Other Hope is Lost.** Using this option WILL result in the loss of data so it's not recommended. Furthermore, if you're going to run this option, Microsoft recommends that you initiate a full backup BEFORE running this option as once complete you have no other option for undoing the loss you will have caused.
As such, that begs the question: "Why would you want to use this technique?" And there are two answers.
First, you would use this technique IF you had no existing backups that you could use to recover from corruption otherwise. Therefore, don't let this ever become a need – by making sure you always have viable backups.
Second, there are some EDGE cases where **SOME** databases might actually FAVOR uptime over data-purity (meaning that these kinds of databases would prefer to avoid down-time at the expense of data-continuity or purity) and in cases like this there are ADVANCED scenarios where the use of REPAIR_ALLOW_DATA_LOSS might be acceptable (assuming you understand the trade-offs). And for more info on these kinds of scenarios, or where this would make sense, take a peek at my previous post where I provide a link to a Tech Ed presentation made by Paul Randal showing some of the raw kung-fu moves you'd need to pull off correction of these sorts of problems – assuming you felt you were in a scenario where you favored up-time over correctness.

Long story short: by keeping a level head, and by understanding what's going on AND by having regularly tested backups that you can fall back upon, addressing problems of corruption can be scary and ugly but non-fatal.

Moreover, if you regularly practice the techniques or details outlined in the scripts and examples that I'll post in my next two blog posts in this series, then, when it comes time to deal with ANY sort of disaster – including corruption – you'll be able to handle these these emergencies MUCH better than if you wait until a disaster strikes to learn these techniques and practices.

Part X: Page Level Restore Operations

TAGS: MICROSOFT SQL SERVER SECURITY

0 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

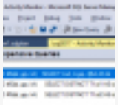
0 COMMENTS

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.

Email address United States SIGN UP ▸

Getty Images

SECURITY > ENDPOINT SECURITY

Apple Is Ready for the Tech Backlash

The only real guarantee that companies won't go overboard in invading customers' privacy is that they have large revenue streams which make it an unnecessary risk.

Bloomberg | Mar 30, 2018

(Bloomberg View) --It's easy to understand Apple CEO Tim Cook's somewhat self-righteous comments on the Facebook-Cambridge Analytica scandal: His company has never emphasized customer data monetization, and privacy has been one of its persistent selling points. Apple still collects our data — and reserves the right to share it — but the company has a much better track record than newer tech titans