ITPro with deep vertical Knowledge

PRACTICAL SQL SERVER

SQL Server Database Corruption, Part I: What Is Corruption?

overachiever | Jan 27, 2012

When it all comes down to it there are really only two main things that DBAs need to worry about: making data available to the proper people, and making sure it's inaccessible to the not-so-proper people. All other considerations are really just appendages to these two concerns. For example, performance is just an extension of making data available to the right users – because if they can't

access it in timely fashion, then it's not really as 'available' to them as it should be.

Part II: Simulating Corruption

As such, concerns around disaster prevention and recovery should rank high on every DBA's list. Yet, surprisingly, I find that most DBAs (especially the 'reluctant' ones) don't have enough of a grasp on how to sufficiently protect against database corruption.

I therefore thought it would be fun to launch into a multi-part set of posts that provide a hands-on overview of SQL Server Database Corruption Basics – including an overview of what corruption is, why you can't really prevent it – and what you can do to deal with that reality such that you're able to ensure proper availability and uptime of your data even when corruption occurs.

What is Database Corruption?

For the purposes of this series of posts, database corruption is defined as a problem associated with the improper storage of the actual zeroes and ones needed to store you database data at the disk or IO sub-system level.

In this sense, the corruption addressed in this series of posts is VERY different from other kinds of 'disasters' that might render business data useless (such as when a developer or admin runs an UPDATE statement without a WHERE clause or a software glitch incorrectly 'mangles' unit prices). However, while user/application errors or 'disasters' are technically much different than disasters caused by corruption, it's important to note that many of the mitigation strategies outlined in this series of posts for addressing SQL Server database corruption ALSO represent best-practices that can easily be leveraged as a means for addressing other types of database disasters.

What Causes Corruption?

In virtually all cases of SQL Server database corruption (over 99.99% by most accounts), the root cause of corruption is due to problems at the IO subsystem level – meaning that the root cause is a problem with drives, controllers, and possibly even drivers. And while specific root causes can vary widely (simply because of the sheer amount of complexity involved in dealing with magnetic storage), a key thing to remember about disk systems is that anyone in IT knows that all major operating systems ship with the equivalent of some sort of Disk-Check utility (CHKDSK) that can scan for bad sectors, mangled entries, and other storage problems that can and will creep up in

storage environments.

Database Corruption and Physics

In other words, when you think about all of the complex operations going on (at insane speeds) when it comes to saving and retrieving data, it shouldn't come as a surprise that things might go wrong every once in a while. In fact, in an a fantastic bit of one of the best 'hard core' science-fiction books that I've read in a while, author Christian Cantrell reviews the perils of magnetic storage through the eyes of his protagonist in *Containment* as follows:

> *Eliminating moving parts from machinery was the best way to improve reliability. Since the subatomic laws of the universe dictate that it's physically impossible for any moving part to move in precisely the same way every single time it moves, each and every moving part in a system represents some measure of unpredictability and unaccounted-for variability. A vulnerability you might even say. What if one critical part didn't move quite enough? what if it moved too much, or too quickly, or with slightly less velocity than the last time it moved? What if it didn't move at all? How many times can it move before it wears out? How long will it take to wear down the lubricant designed to reduce friction? How does temperature affect the properties of that lubricant?*
>
> ***Christian Cantrell** – Containment **(Chapter 30)***

Stated differently: Corruption really isn't a question of if. It's more a question of when. And, in that sense, when SQL Server databases become corrupt, the source of this corruption is almost always the result of some sort of failure at the disk-subsystem level in the sense that the highly-structured data being 'saved' to disk was merely written improperly. Consequently, when SQL Server goes to re-load this data from disk, it encounters 'zeroes and ones' that are out of order or 'mangled' and corrupt – which, in turn, means that databases can and WILL lose data when corruption at the IO subsystem level is encountered within SQL Server databases.

Up Next

So, now that we've covered what corruption is and how it's caused, I thought it would be fun to take a look at a hands-on example of creating or simulating corruption – since most people tend to learn better by example. So, in my next blog post I'll provide a 'hands-on' overview of ways in which you can purposefully go in and 'mangle' or 'corrupt' your own non-production databases as a means of being able to see exactly how corruption works – and to serve as the basis for further examples and 'demos' where we look at ways to repair that corruption.

- Part II: Simulating Corruption
- Part III: Preventing Corruption
- Part IV: CHECKSUM Page Verification
- Part V: Storage Problem Alerts
- Part VI: Regular Corruption Checks
- Part VII: Backups
- Part VIII: The Importance of Backups
- Part IX: Responding to Corruption
- Part X: Page Level Retore Operations
- Part XI: Full Recovery Operations
- Part XII: Recovery Sample
- Part XIII: Recap

TAGS: MICROSOFT SQL SERVER

4 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

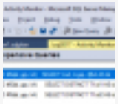
0 COMMENTS

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 ▸