ITPro with deep vertical Knowledge

RECENT

PRACTICAL SQL SERVER

SQL Server Database Corruption, Part III: Preventing Corruption

overachiever | Feb 09, 2012

In previous posts we touched upon an overview of what causes corruption and looked at ways to simulate corruption. In this post we'll talk about more about corruption – in terms of addressing how you can prevent it. (Though, if you've been paying attention in previous posts, you'll probably guess what's coming up).

How to Prevent SQL Server Database Corruption

In answer to the question: "How can I prevent corruption?" the simple, and honest, answer is: "You really can't". As covered in previous posts, corruption is 'more or less' a random problem that occurs at the disk subsystem level and really can't be prevented. (Or as I like to tell all of my consulting clients: "There's a reason that EVERY major Operating System out there ships with something like CHKDSK.")

So, while disk/storage errors should be rare within any production system, the fact is that these kinds of errors aren't so much a question of IF they'll happen, but WHEN they'll happen – simply because the reality is that magnetic storage is insanely complex and subject to the occasional 'hiccup' that can result in corruption.

However, given that corruption SHOULD be a relatively rare problem that you will encounter in your environment, it IS safe to say that if you keep bumping into seemingly-regular or semi-regular problems with corruption, it's fair to say that you may need to start becoming concerned about whether there's a problem with your hardware or possibly even drivers and so on. (The problem, however, is that troubleshooting these kinds of problems is insanely hard – so if you do start to recognize regular problems with corruption it may be time to start thinking about migrating critical systems to new hardware.)

However, even though you can't prevent corruption, that doesn't mean that you are powerless against its effects. In fact, not only does SQL Server readily accept or anticipate that disk subsystem errors can and WILL occur, but it also provides a number of great tools and functionality that can be used to address and compensate for corruption when it happens. Consequently, by taking advantage of these features, it's possible to become very protected against the kinds of damage that corruption could otherwise cause. All you really need, then, to protect 'against' corruption is a decent amount of pro-active effort that can help protect you against the EFFECTS of corruption.

Caveat: Incurable Corruption

Of course, while the underlying purpose behind this series of posts is to provide a solid overview of ways to detect, correct, and protect against corruption, there ARE some kinds of corruption from which you can't recover – such as corruption that happens in the 'boot page' (or page 9) as mentioned in my previous post. So, in cases where you bump into these kinds of corruption problems, the only thing you're really going to be able to do is resort to using backups. That said, backups happen to PRETTY MUCH be the KEY to responding to most OTHER forms of corruption as well. So, in other words: one huge component to being able to properly deal with corruption when it happens (not if it happens) is to have viable, regularly tested, backups on hand – so that you'll have every option possible at your disposal for dealing with corruption.

The Key to Dealing With Corruption

So, given that corruption can't really be prevented and given that some forms of it are actually so nasty that you'll need backups (though the truth is that many/most forms of it can be nasty enough that you'll need PART of your backups in many cases), the next question becomes: "What is the key to being able to successfully deal with corruption when it occurs?"

And, in my experience, Without a doubt, the best and easiest way to address with database corruption within SQL Server environments is to detect it early and address it BEFORE options to address it run out – because, as we saw in the previous post, it IS possible for corruption to happen and then, effectively, go unnoticed for a potentially LONG time if nothing ends up querying the locations or pages that were trashed by corruption.

In other words, imagine a mission-critical database that happens to encounter corruption during when writing data that IS important but which, for the sake of argument, maybe doesn't get queried except at the end of every month. If this corruption goes unnoticed for, say, a few weeks its entirely possible in many organizations that the use of rolling backups will severely hamper the ability to go in and address this problem once it's discovered – simply because the backups that could have best been used to recover from this issue will have been purged, lost, etc. Whereas, on the other hand, if this problem were found within just a few days (for example) then there would likely be more options available in terms of addressing this problem.

Consequently, the biggest key to being able to deal with corruption is to have as many options and possibilities at your disposal as possible – and that, in most cases, can only happen when corruption is discovered relatively early on. Otherwise, corruption almost always becomes more expensive and less easy to deal with.

Up Next

So, with those ideas in mind, my next post will take a look at how and why the use of CHECKSUM page verification is such a great thing. Then, in a subsequent post we'll take a look at ways to pro-actively be alerted to whenever SQL Server detects any kind of disk problem related to data purity, and then we'll keep ploughing our way through some tips, techniques, approaches, and examples of how to deal with corruption once it's been discovered.

Part IV: CHECKSUM Page Verification

TAGS: MICROSOFT SQL SERVER

0 COMMENTS

Internet of Things World Series
March 26-30, 2018
iot Internet of Things INSTITUTE
THE WORLD'S FIRST IOT DIGITAL WEEK
5 Days of IoT Focused Content!
REGISTER NOW

```
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]';
```


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

READ MORE

TAGS: SOFTWARE DEVELOPMENT

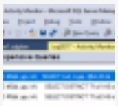
0 COMMENTS

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



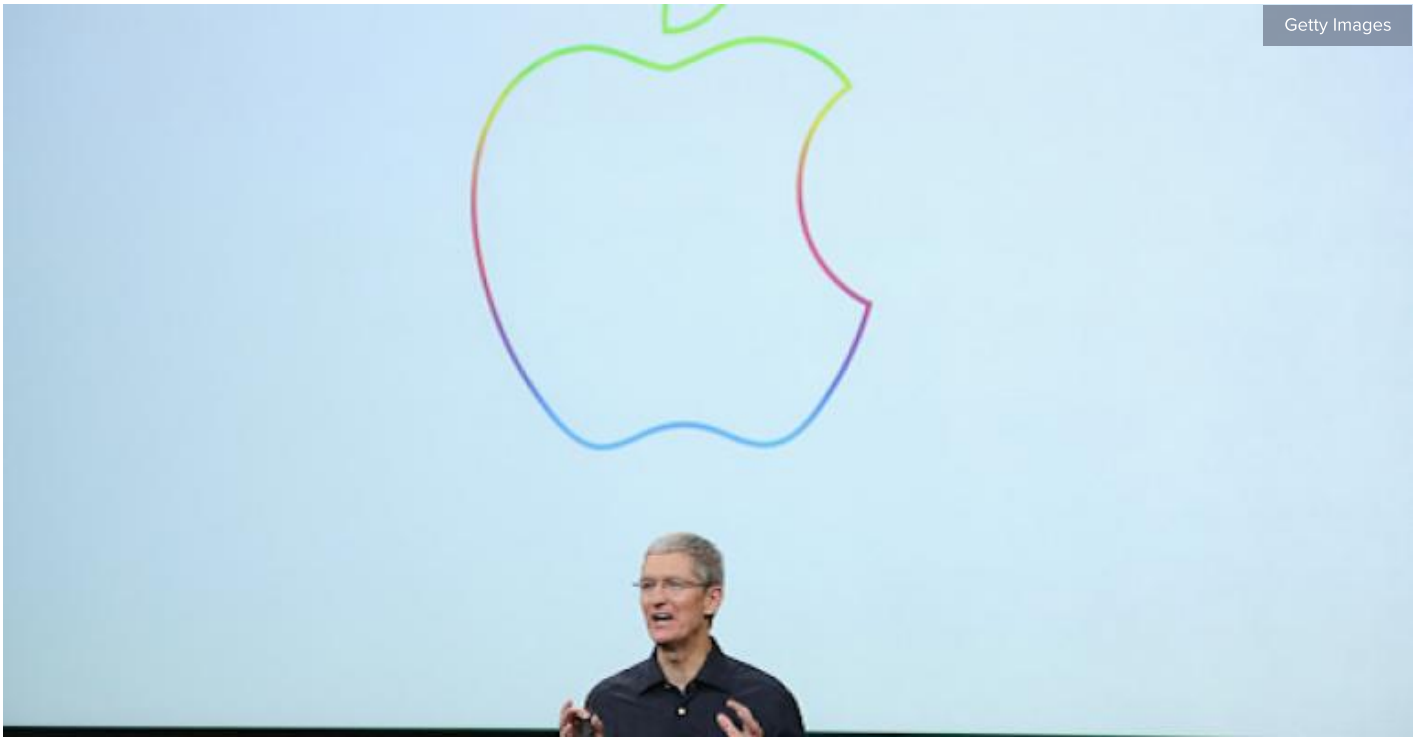
T-SQL Puzzle Challenge: Grouping Connected Items
NOV 27, 2017

Sign up for the ITPro Update newsletter.

Email address

United States

SIGN UP ▸



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.

"The truth is, we could make a ton of money if we monetized our customer — if our customer was our product," Cook said on Wednesday. "We've elected not to do that." Asked what he'd do if he were in Facebook CEO Mark Zuckerberg's shoes, he replied, "I wouldn't be in this situation."

In reality, Apple collects more information about its customers than Facebook because it offers more products and services. It knows everything from an iPhone's location to its owner's taste in movies and music. It's all in the company's privacy statement, including the frank admission that Apple will share the data with "strategic partners that work with Apple to provide products and services, or that help Apple market to customers" and with law enforcement. Apple also lets applications that run on its platform (including Facebook) hoover up data. And it does have a small ad platform that allows targeted advertising.

Apple's closest equivalent in terms of data collection, sharing and "pass-through" to app developers is Google, which offers roughly the same array of services (plus search) and also owns mobile and desktop operating systems. The difference is that Apple has imposed some voluntary restrictions on itself: For example, it anonymizes location data (unless you're using the "Find My iPhone" feature) and generally employs "differential privacy" — a cryptography-based practice of obtaining usage and preference data without linking it to specific users. Google and Microsoft also use that to some