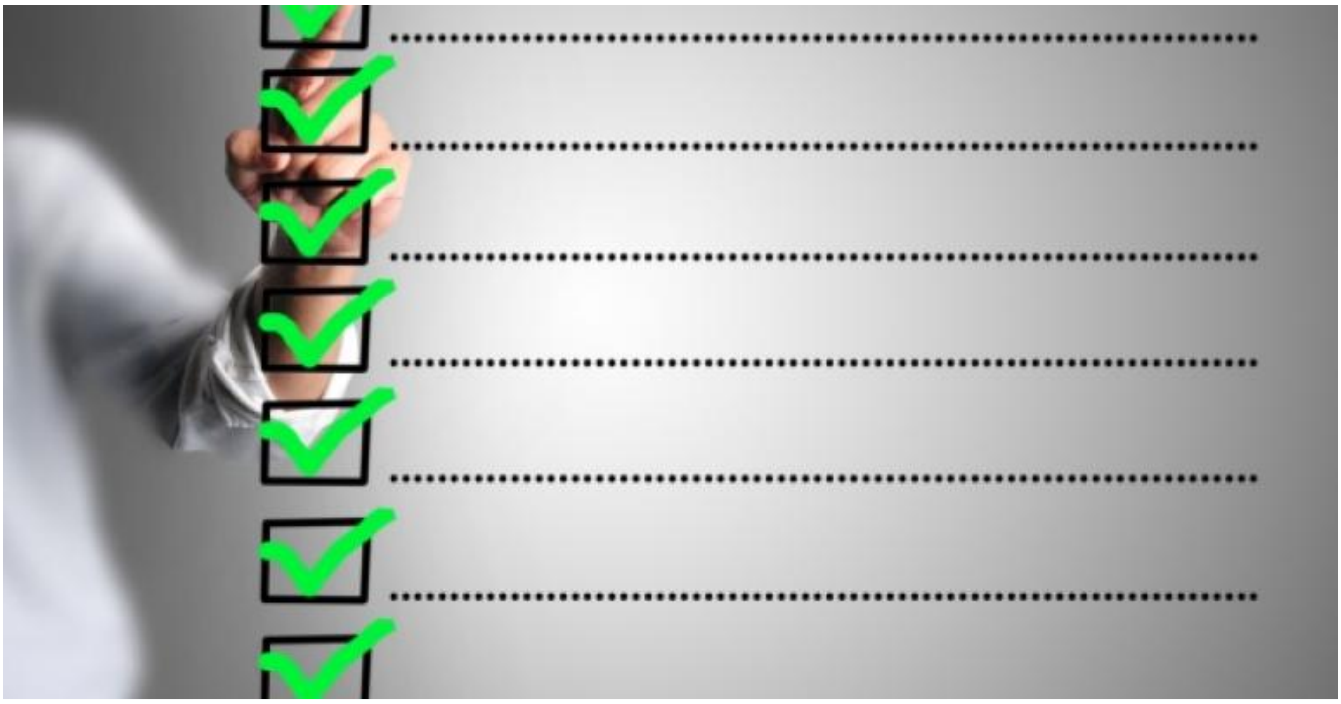
PRACTICAL SQL SERVER



SQL Server Database Corruption, Part VI: Regular Corruption Checks

overachiever | Mar 07, 2012

Previously, we looked at how to set up alerts for cases where SQL Server encounters issues with the storage subsystem. Setting those up is a key component to staying alerted to when problems happen – but another great mechanism for detecting corruption early is to set up regular checks for corruption.

Regular Checks for Corruption

Of course, in some environments, you may find that SQL Server will write data to disk and then may not use it for long periods of time afterwards. Therefore, a best practice in addressing the possibility of corruption is to run regular checks (typically once or twice a week as possible) to aggressively verify data integrity and validate many forms of constraints present in the database itself.

To do this, you'll just want to take advantage of an insanely powerful and robust tool that ships with SQL SERVER, known or accessed via DBCC CHECKDB (and associated) commands. And while the scope of this tool and its capabilities, optimizations, and functionality are really beyond the scope of this document (simply because an entire book could easily be written about DBCC CHECKDB), documentation for this tool can be found here. (Note too that DBCC CHECKDB is also mentioned/used later in this document not just as a means of detecting corruption, but REPORTING on the exact specifics about the scope, nature, impact, and type of corruption found when it is indeed found.)

Otherwise, when it comes to regularly detecting corruption, a best practice is simply to run DBCC CHECKDB to run on a regular basis – as a scheduled job. So, for example, assume that you wanted to run DBCC CHECKDB against your database(s) every Sunday and Wednesday nights (periods where performance isn't critical and, for example, periods right before making FULL/DIFFERENTIAL backups). To institute these checks, you'd just need to create a new SQL Server Agent Job as follows:

PartVI_Figure1

Then, after giving the new Job a name in the General tab (and assigning an owner – SysAdmin is the best owner), create a new step as shown below:

PartVI_Figure2

As for the actual command that you'll specify for this step, you'll want to put in the following:

```
DBCC CHECKDB([databaseNameHere]) WITH NO_INFOMSGS, ALL_ERRORMSGS
GO
```

```
GO
```

making sure to replace [databaseNameHere] with the name of your own database – surrounded by square brackets. But note that this will ONLY check a single database. You CAN easily ‘chain’ statements one after in the command-text for this step to have it check multiple databases, like so:

```
DBCC CHECKDB([someDb]) WITH NO_INFOMSGS, ALL_ERRORMSGS
GO

DBCC CHECKDB([anotherDB]) WITH NO_INFOMSGS, ALL_ERRORMSGS
GO
```

Or, a better approach is to use Aaron Betrand’s great replacement for sp_msForEachDB by visiting this site/link, getting the script provided, running it on your server to create a new stored procedure that will safely and accurate ‘loop’ over all databases, and then pasting in the following as your command:

```
EXEC master..sp_foreachdb
        @command = 'DBCC CHECKDB(?) WITH NO_INFOMSGS, ALL_ERRORMSGS'
GO
```

Because this approach will cause this single job step to check ALL databases on your system – a much better option as you never know when or where corruption might occur and encountering it in the master or other system databases can be fatal if not addressed quickly.

Otherwise, the key thing to note about whichever method used is that both approaches use the same ‘switches’ or ‘options’ in that they both suppress information messages (which otherwise ‘clog the screen’ with useless noise that provides no real value) and ensure that ALL error messages are output – instead of the first 200 errors that would NORMALLY be output by default otherwise. (Which can be a huge issue, because some corruption errors CAN look very nasty until you notice (near the end of reporting) that maybe they’re all caused by a single, simple, problem that would have gone unnoticed if only 200 errors were being output.)

Reaching 3.4 million
IT Professionals

Likewise, when setting up this job, make sure to switch to the Notifications tab, and ensure that an email/etc it sent WHEN this job fails – as shown below (where a single operator called 'General' has been set up to send out emails to alerts@somecompany.com or something similar):

PartVI_Figure3

Additionally, to make the output of DBCC CHECKDB when run unattended, make sure to modify the Advanced tab for the Job Step created previously so that job output is put into the SQL Server logs – as this will provide additional detail when errors are encountered – as per the following screenshot:

PartVI_Figure4

And, of course, don't forget to visit the Schedules tab and set up an applicable schedule for this job.

Conclusion

By following the steps outlined above, you'll have successfully set up your server to regularly check databases for corruption – which is a key component of being able to detect it early. Again, running this once or even twice a week is typically going to be adequate – but the sensitivity/importance of your data contrasted with the overhead this imposes on the IO subsystem is going to be the key factor in determining how regularly you need to run DBCC CHECKDB in your own environment.

Otherwise, this post hasn't even touched (at all) upon what actually goes ON when DBCC CHECKDB is run. For basic information on what is actually going on under the covers, I recommend taking a peek at Books Online. And, if you REALLY want to get a feeling for how amazing, powerful, and well designed DBCC CHECKDB is, then I heartily recommend that you check out Chapter 11 of SQL Server 2008 Internals – where Paul Randal provides a more in-depth overview of DBCC CHECKDB (and friends) in all of its glory.

Likewise, to gain just a SENSE of how complex DBCC CHECKDB really is, take a peek at this blog post by Bob Ward over on the CSS blog where he outlines all of the effort that went into

enabling a faster CHECKDB.

ENGAGE

Finally: Running DBCC CHECKDB() is an aggressive operation. It's been designed to mitigate the full scope of the performance hit needed to aggressively scour your database by means of some VERY intelligently defined algorithms that make it a HIGHLY EFFICIENT means for detecting corruption. But, even so, it's still an aggressive operation and should not be run during peak-load times unless there is a very real need to check for corruption. In other words: try to schedule regular DBCC checks at night or during off-peak hours when the contention it causes for system resources (particularly disk) will not be as severe.

Part VII: Backups

TAGS: MICROSOFT SQL SERVER

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

…

AAY

READ MORE

TAGS: SOFTWARE DEVELOPMENT

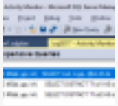
0 COMMENTS

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 ▸