

Module 10: Corruption Detection and Recovery

Jonathan Kehayias
Jonathan@SQLskills.com



Why Is This Module Important?

- **Corruption does happen, mostly caused by I/O subsystem problems**
- **People don't realize they have corruption until too late**
 - Either they don't know how to check for corruption or they miss the warning signs
- **People don't know what to do when they do have corruption, leading to:**
 - More data loss and downtime than necessary
 - Monetary and even job losses
 - Overall lowered perception of SQL Server
 - Makes it harder to convince management that SQL Server is Enterprise capable



What Can Happen to an Unprepared DBA Confronted by Corruption?



- Image from http://commons.wikimedia.org/wiki/File:Panic_button.jpg

Practice Makes Perfect

- **If you're recovering from a disaster, or helping a client recover from a disaster:**
 - You must know what you're doing
 - You must have practiced
 - You must NOT make things worse
- **Most of you have restored simple backups or run DBCC CHECKDB, but there are some things that most people have not done**
 - These are the things that will set you apart from others
 - Your colleagues and clients will love you if you can save them while minimizing downtime and data loss

Root Cause Analysis

- **No matter what method you use to recover from corruption, you should always determine why it happened to avoid future problems**
- **It may be obvious what happened**
 - E.g. a known SAN problem or power failure
- **You may have no idea what happened, so what to do?**
 - Google/Bing for the corruption message you saw
 - Run I/O subsystem and server memory diagnostics
 - Examine the SQL Server error log and Windows event logs for clues
 - Check that firmware is up-to-date
 - Investigate NTFS filter drivers
 - Possibly contact Microsoft Customer Support for assistance

Overview

- **How corruption occurs, I/O errors, and page protection**
- **Consistency checking options**
- **DBCC CHECKDB**
 - What it does
 - Best practices and FAQ
 - Interpreting CHECKDB output
 - How repair works
- **Recovering from corruption**
- **Horror stories from the wild**

How Does Corruption Occur? (1)

- If a data file page is “good” when written out of SQL Server’s memory, but “bad” when read back into memory, that’s corruption
 - Log files can be corrupt too
- It’s almost always the I/O subsystem
- The I/O subsystem means anything “underneath” SQL Server
 - Windows operating system
 - File system filter drivers
 - E.g. antivirus, defraggers, encryption
 - Network cards, switches, cables
 - SAN controllers, RAID controllers, disks
- Lots of moving parts, lots of code, potential for bugs
- Could also be bad memory, SQL Server bugs, human error

How Does Corruption Occur? (2)

- Consider the Mean Time Between Failure (MTBF) of disks
 - Various sources online give figures around 1.5 million hours as MTBF as long as the operating environment is ideal
 - Dust, high/low temperature, vibration, power cycling are not ideal
 - It’s a *mean*, which means there’s a bell-curve of failure times
 - Someone’s going to get the low-end of the curve
 - And this also applies to SSD/flash storage
- Jim Gray likened a disk head in a 15,000rpm disk to a Boeing 747 flying at 500mph about ¼ inch above the ground
 - What are the tolerances involved?
 - What happens in a crash?

What Does NOT Cause Corruption

- Many misconceptions about what can cause corruption
- Corruptions are not caused by:
 - Anything an application can do
 - Anything you can do in SQL Server with supported, documented commands
 - Interrupting a database shrink, index rebuild, or long-running batch
 - Shutting down SQL Server
- In a very rare case, you may hit a bug in SQL Server

Corruption Propagation To Remote Servers

- SQL Server redundancy technologies do not propagate data file corruptions caused by the I/O subsystem
 - Database mirroring, log shipping, replication, availability groups
- These technologies send log records (or actions based on log records), not data file pages from disk
- Possible to have a “second-order” corruption propagated
 - Corrupt page is not detected on the local server and used to calculate a result which is stored locally, and sent to the remote server
- Backing up a database should detect corruption, preventing propagation to a remote server during restore
 - As long as the corruption happened in the I/O subsystem
- SAN replication should also not propagate corruption
 - Replicating changed disk blocks as they’re written, not read from disk

Disappearing Corruption

- **Typical description of the phenomenon:**
 - Consistency-checking job fails during the night
 - DBA runs a consistency check again in the morning, but no corruption!
 - DBA questions where the initial report of corruption was correct
- **What's really happening:**
 - Corruption is reported on some pages allocated to an index
 - After the consistency-checking job runs, an index maintenance job runs
 - The index containing the corrupt pages is rebuilt, which builds a new index with new pages, and then de-allocates the old index and pages
 - Another consistency check runs but doesn't find any corruption
 - The second consistency check is reading different pages from the first one, so doesn't see the corrupt pages
- **Consistency checks validate those data file pages that are allocated to tables and indexes in the database**

DBCC WRITEPAGE

- **DBCC WRITEPAGE is used to directly update a page**
 - dbcc writepage (('dbname' | dbid), fileid, pageid, offset, length, data [, directORbufferpool])
- **USE THIS COMMAND ENTIRELY AT YOUR OWN RISK!**
- **Extremely useful for creating corrupt databases to test with**
- **Paul added the directORbufferpool option in SQL Server 2005 to allow page checksum failures to be simulated**
 - The buffer pool is flushed for the database
 - The file's real FCB is unhooked and a new one created
 - The page is read directly into memory and updated
 - The page is written directly back to disk, bypassing the buffer pool
 - The file's real FCB is put back
- **Also useful for manually attempting to fix database corruption**
- **See Paul's blog post at <http://bit.ly/XfMkOv>**

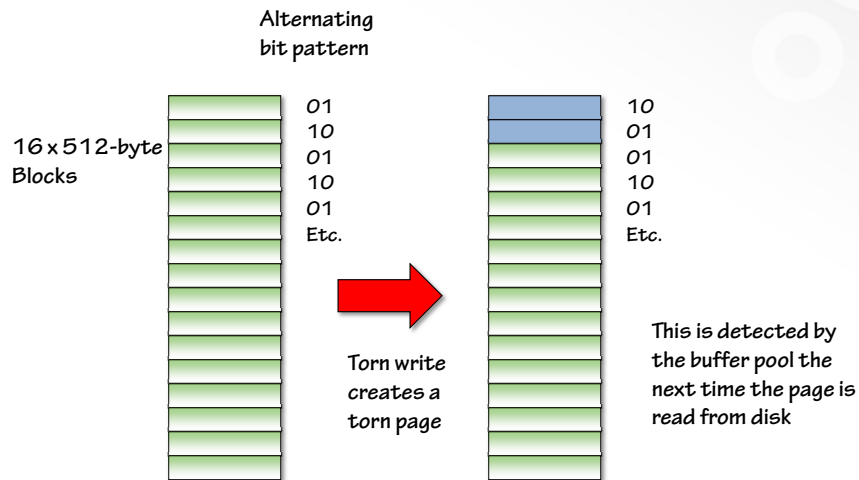
Demo

DBCC WRITEPAGE and creating corruption

Torn-Page Detection (1)

- An 8KB data file page is really 16 x 512-byte disk sector
- It is possible for a page to be partially written in the event of a power failure
- Torn-page detection allows SQL Server to detect incomplete writes
 - Takes two-bits from each sector, stores them in the page header (a 32-byte metadata structure in each data file page) and writes an alternating bit pattern in each sector
 - 01 in the first, 10 in the second, 01 in the third, etc.
 - The bit pattern flips each time the page is written to disk
 - On a subsequent read, if the pattern is disrupted, the page is torn
- This does not detect corruptions within a disk sector though

Torn-Page Detection (2)



Page Checksums (1)

- **SQL Server 2005 introduced per-page checksums**
 - On by default for new databases from SQL Server 2005 onward
 - Added to *tempdb* from SQL Server 2008 onward
- **Performed by the buffer pool**
 - Calculated as a four-byte value and stored in the page header as the very last thing SQL Server does on a physical write
 - Recalculated and checked against the stored value as the very first thing SQL Server does on a physical read
- **Upgraded databases must enable it**
 - Switching it on doesn't do anything until pages are written
 - No easy method to force all pages to get a page checksum
 - Switching it on does not erase existing torn page detection
- **Negligible CPU overhead as simple checksum algorithm**
 - Error detecting, not error correcting

Page Checksums (2)

- Provides the “smoking gun” that the error is not due to SQL Server
 - Helps in hardware vendor vs. Microsoft arguments
- Checked when:
 - Page is read normally
 - Page is read during consistency checks
 - Page is read during BACKUP ... WITH CHECKSUM
 - Page is read from within a checksum'd backup
- **Make sure ALL your databases have CHECKSUM enabled!**

I/O Errors

- 823: a hard I/O error
- 824: a soft I/O error
- Example error message

Msg 824, Level 24, State 2, Line 1

SQL Server detected a logical consistency-based I/O error: incorrect checksum (expected: 0x7232c940; actual: 0x720e4940). It occurred during a read of page (1:143) in database ID 8 at offset 0x0000000011e000 in file 'c:\sqlskills\broken.mdf'. Additional messages in the SQL Server error log or system event log may provide more detail. This is a severe error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.

- Logged in msdb.dbo.suspect_pages
 - Finite size so maintenance required to purge old rows
 - Input into single-page restore operations

Read-Retry

- Reads that fail because of I/O errors are retried 4 times
- Eventual failure results in an I/O error
- Eventual 'success' results in an error in the error log:
 - A read of the file <<FILE NAME>> at offset <<PHYSICAL OFFSET>> succeeded after failing <<RETRY COUNT>> time(s) with error: <<DETAILED ERROR INFORMATION>>. Additional messages in the SQL Server error log and system event log may provide more detail. This error condition threatens database integrity and must be corrected. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.
- *Early warning of impending doom as read-retry means the I/O subsystem is returning incorrect data to SQL Server!*
- KB article: <http://support.microsoft.com/kb/2015757>

Memory Corruption

- The buffer pool periodically validates page checksums of clean pages in the buffer pool
 - Failures results in an 832 error (do not confuse with 823)
 - In 2012+, you'll see 854, 855, 856 errors instead of 832
- SQL Server 2012+ on Windows Server 2012+ also supports automatic correction of some memory corruptions of buffer pool memory
 - Corrupt clean pages are re-read from disk
 - Corrupt dirty pages cannot be repaired
 - See Glenn's post at <http://bit.ly/SmZLLA> for more details

Automatic Page Repair

- Corrupt pages can possibly be automatically repaired in database mirroring and availability groups
- Works for 824 'soft-I/O' errors, some 823 'hard-I/O' errors, 829 'in restore' errors
 - Errors are hit by queries reading the page or by DBCC CHECKDB
- Corrupt pages on the principal AND mirror (primary replica and secondary replicas) can be repaired
- Repairs are asynchronous, corrupt pages are unusable until fixed
- NOTE: Only meant to be a band-aid to prevent downtime. It is NOT a substitute for having alerts for high-severity errors and taking affirmative action to rectify/prevent them.

DBCC CHECKDB (1)

- This is the main consistency checking command
- Runs all consistency checks against a single database
- Very simple to execute:
 - DBCC CHECKDB (N'database')
 - Or change context to the desired database and just do DBCC CHECKDB
- The only way to read all allocated pages in the database
 - Use to force page checksums to be checked
- Possible to change some of the behavior using a variety of options
 - We'll cover these later
- Very resource intensive with CPU, memory, and I/O
- Will use multiple threads on Enterprise Edition
 - Parallelism can be disabled using documented trace flag 2528
 - Parallelism can be limited with MAXDOP option in SQL Server 2016+

DBCC CHECKDB (2)

- Many algorithms have been added to minimize runtime and run online without blocking locks since SQL Server 2000
 - Compared with 6.5 or 7.0, which used locking all the time
 - Performance and scalability has increased from version to version
- Features worth mentioning from SQL Server 2005 onward:
 - Progress reporting
 - Data purity checks
 - "Last known good" stored in the database boot page
 - No false failures like in SQL Server 2000
- Paul's comprehensive blog post series is at <http://bit.ly/TFvxmG>
- Detailed information (~90 pages) in the *SQL Server 2008 Internals* and *SQL Server 2012 Internals* books

Does DBCC CHECKDB Run During Startup?

- When SQL Server starts, you may see messages in the error log about DBCC CHECKDB
 - 2016-02-18 12:02:34.23 spid4s CHECKDB for database 'master' finished without errors on 2016-02-16 03:10:47.163 (local time). This is an informational message only; no user action is required.
- This does not mean that DBCC CHECKDB was performed during instance startup
- It's reporting "last known good" time for databases on the instance
- This is the time that DBCC CHECKDB last finished without finding any corruptions
 - Stored in the boot page of the database (page 9 in file 1)
 - Not propagated to availability group replicas
 - Can be viewed using DBCC DBINFO

How Are DBCC CHECK* Commands Online?

- SQL Server 2000 used transaction log analysis, which was slow
- Since SQL Server 2005, DBCC commands use a database snapshot, which is a transactionally-consistent, point-in-time view of the database
 - Not required if the database is read-only, single-user, or the target database is itself a database snapshot
- The database snapshot is “hidden” prior to SQL Server 2014
 - Created as NTFS alternate streams of the existing data files
 - Created as real files in SQL Server 2014+ to allow ReFS support
- Sometimes problems can occur:
 - The database snapshot runs out of space or hits an NTFS limit
 - See KB <https://support.microsoft.com/en-us/kb/2002606> and <http://bit.ly/1xLnHg>
 - 3rd-party NTFS filter drivers that do not support alternate streams correctly
- Solution: create your own database snapshot and check that

What Exactly Does CHECKDB Do? (1)

- Primitive checks of critical system tables
 - Problems? Game over.
- Allocation checks
- Logical checks of critical system tables
- Logical checks of all other tables
 - Including nonclustered indexes, LOB, FILESTREAM...
- Service Broker data validation
- Metadata checks
- Indexed view, XML index, spatial index checks
 - Only under EXTENDED_LOGICAL_CHECKS option in SQL Server 2008+
- Repairs are done at each stage (if specified and possible)
- What about when WITH PHYSICAL_ONLY is used?

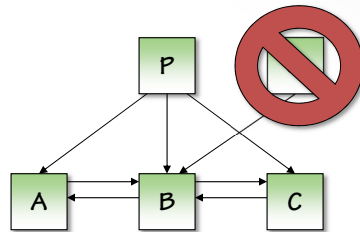
What Exactly Does CHECKDB Do? (2)

- **Another way to think of it is that CHECKDB does:**
 - Primitive system table checks, then
 - DBCC CHECKALLOC, then
 - DBCC CHECKTABLE of system tables, then
 - DBCC CHECKTABLE of user tables, then
 - DBCC CHECKCATALOG
- **It does not do:**
 - DBCC CHECKIDENT
 - DBCC CHECKCONSTRAINTS

Fact Processing (1)

- **DBCC CHECKDB reads all allocated pages in the database in a very efficient manner**
- **It does not use the obvious link-following recursive algorithm**
- **It reads the pages in allocation order, driving its own readahead, and generates “facts” about what it has seen**
 - E.g. facts about pages, b-tree linkages, off-row text linkages
 - Driven by an internal structure called the MultiObjectScanner
- **Each fact has a multi-part key**
 - Including object ID, index ID, allocation unit ID, page ID
- **Facts are passed to the Query Processor for efficient sorting and hashing, then passed back to DBCC for aggregation**
 - Aggregation means that facts cancel each other out
 - When extra or missing facts are detected, that’s a corruption

Fact Processing (2)



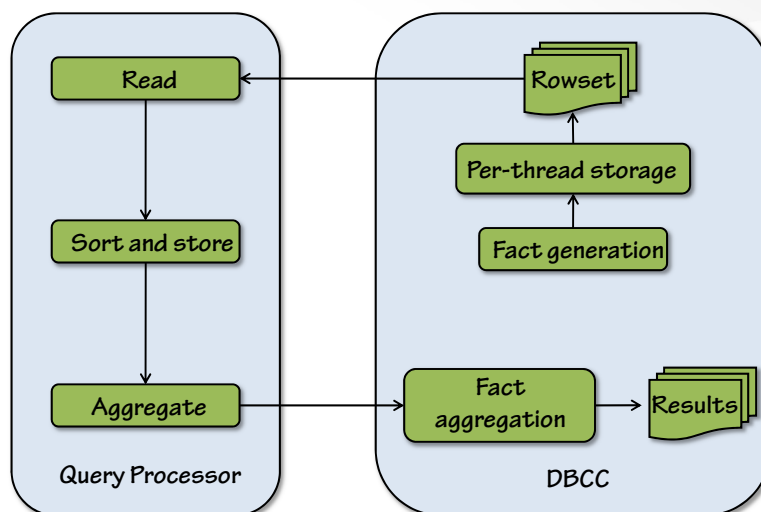
B-Actual
 B-Sibling-A
 B-Sibling-C
 A-Actual
 A-Sibling-B
 P-Actual
 P-Parent-A
 P-Parent-B
 P-Parent-C
 Q-Actual
 Q-Parent-B



B-Actual
 A-Sibling-B
 C-Sibling-B
 P-Parent-B
 Q-Parent-B

ERROR!!

Fact Processing (3)



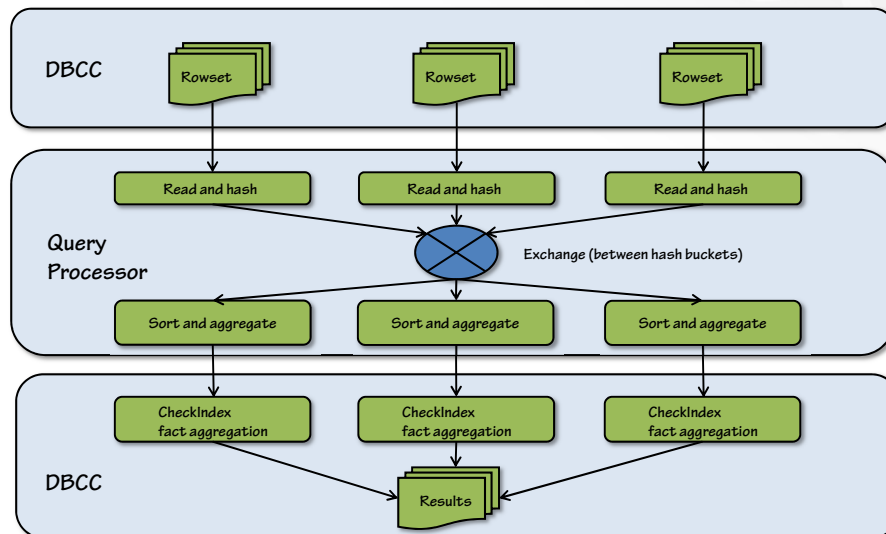
Batch Processing

- The fact-storage worktable usually requires more memory than is available so it spills into *tempdb*
- The batch size is limited to prevent excessive *tempdb* space usage
 - A batch's "size" is the number of tables and associated indexes
- The minimum batch size is a single table and its nonclustered indexes as everything about a table must be checked at the same time
- Tables are added to the batch until one of the following occurs:
 - The total number of indexes in the batch becomes more than 512
 - The estimate for all necessary facts for the batch becomes more than 32 MB
 - If a repair option is specified, the batch size is always limited to a single table
- Read and write performance of *tempdb* can greatly affect the run time

Parallelism (1)

- DBCC CHECKDB will run in parallel on Enterprise Edition
 - Also applies to DBCC CHECKTABLE and DBCC CHECKFILEGROUP
 - DBCC CHECKALLOC and DBCC CHECKCATALOG are always single-threaded
- Each batch executes the DBCC CHECKDB internal "query" and the Query Processor decides how far to parallelize
 - Up to the limit imposed by the sp_configure MAXDOP setting or the Resource Governor workload group MAX_DOP setting
 - DBCC CHECKDB only has a MAXDOP option in SQL Server 2016+
- Each thread is given a page to generate facts for, and then calls into the MultiObjectScanner to get the next page to process
- Threads then participate in fact aggregation once all pages have been read and processed for facts
- Parallelism can be disabled using documented trace flag 2528
 - Greatly increases run time, but greatly decreases the resources required

Parallelism (2)



Potential Problems with DBCC CHECKDB (1)

- **Hidden database snapshots use NTFS alternate streams on existing database files**
 - Some 3rd-party software uses kernel filter drivers that do not cope with these correctly and return garbage data so DBCC CHECKDB reports corruption
- **The initial phase where the database snapshot is created cannot be interrupted due to limitations in the Engine**
 - This can make killing a DBCC CHECKDB impossible, and the SPID to appear as if it is in rollback
- **Database snapshots in general have an issue where they can run into NTFS limitations on size due to file-system fragmentation**
 - See <http://support.microsoft.com/kb/2002606> for more details

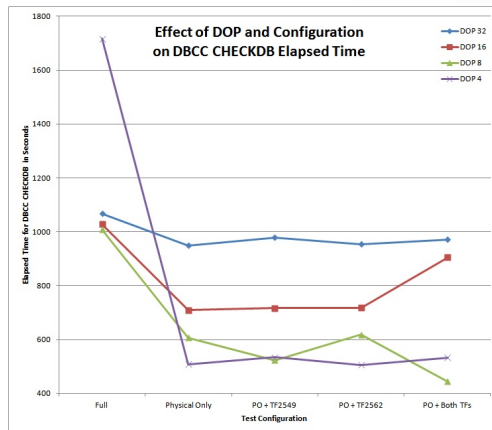
Potential Problems with DBCC CHECKDB (2)

- **Beware of unexpected very long run times**
 - Most likely a corruption has triggered an algorithm to look deeper into the database structure to determine exactly where the problem is
- **Consistency checks make use of *tempdb* for temporary data storage, so *tempdb* must be sized appropriately**
 - Use WITH ESTIMATEONLY to get an estimate of how much space is required
 - Beware that in some current builds that functionality is broken

Making DBCC CHECKDB Go Faster (1)

- **Recent work has been done to improve the performance of DBCC CHECKDB by:**
 - Allowing it to check all tables as a single batch
 - Slightly changing the I/O pattern it generates
 - Reducing contention on the DBCC_MULTIOBJECT_SCANNER latch
- **KB article describing these is at**
<http://support.microsoft.com/kb/2634571>
- **Ported to SQL Server 2008 and later versions**
 - Uses documented trace flags 2549 and 2562
 - Check the KB article to see which trace flags are required in which versions
- **Be aware these changes may push the I/O subsystem much harder, and you may not see any changes at all**
- **Bob Ward's blog post is at** <http://bit.ly/yAFY7W>

Making DBCC CHECKDB Go Faster (2)



- Consider reducing DOP to lower run time when using the option WITH PHYSICAL_ONLY
- On systems with high numbers of **physical** cores (32+), consider reducing DOP to 4 or 8 to lower run time even without using PHYSICAL_ONLY
- Resource Governor can help with this
- MAXDOP option in SQL Server 2016
- Source: Paul's blog post at <http://bit.ly/10rgH70> (those are zeroes, not Os)

Making DBCC CHECKDB Go Faster (3)

- (Note: This problem is fixed in SQL Server 2014+, so this does not apply, as described in this CSS blog post: <http://bit.ly/1wLc4lm>)
- Consider reducing the memory available to DBCC CHECKDB
- The Query Processor by default will grant DBCC CHECKDB a very large query execution memory grant, potentially causing buffer pool shrinkage
 - Requested grant is many terabytes because of unknown cardinality
 - On a system with 100 GB of memory, grant may be as high as 10 GB!
- Performance testing by Jonathan Kehayias has shown that limiting the amount of memory for DBCC CHECKDB to 1 GB can lead to 5-10% performance increase, and very limited buffer pool pressure
- Use Resource Governor to do this
- Jonathan's blog post at <http://bit.ly/YBb7iH>

Making DBCC CHECKDB Go Faster (4)

- **Indexes on computed columns drastically slow down DBCC CHECKDB**
 - Also applies to DBCC CHECKTABLE and DBCC CHECKFILEGROUP
- **Paul discovered this problem during benchmarking tests for the degree-of-parallelism data on one of the previous slides**
- **Any time a table has an index on a computed column, all threads bottleneck on the DBCC_OBJECT_METADATA latch**
 - This controls access to an Expression Evaluator construct in the Query Processor that evaluates computed column expressions
 - The latch can only be acquired in EX mode, hence the bottleneck
- **This can lead to 20x performance decrease**
- **Solution is to disable such indexes while DBCC CHECKDB is running**
- **Paul's blog post on this is at <http://bit.ly/YAzY6h>**

Which Databases Should Be Checked?

- **All databases should be consistency checked**
 - Include read-only databases
- **Definitely include system databases**
 - *master*, as it contains information about databases, users, logins
 - Corruptions in it can cause an instance to fail to start
 - *mssqlsystemresource* is checked automatically when master is checked
 - *msdb*, as it contains SQL Agent jobs, schedules, backup history, and more
 - *model*, as its content is copied to create new databases
 - *tempdb*, as corruptions in it can cause an instance to shut down
 - Note that if you use the Maintenance Plan Wizard to create a job to check integrity, *tempdb* is excluded
 - *distribution*, to avoid replication problems

How To Run Consistency Checks?

- **Three options for running consistency checks:**
 - Manually
 - Automated script, using SQL Agent
 - Automated using a SQL Server Maintenance Plan
 - See the Pluralsight course *SQL Server: Maintenance Plans* for details
- **If you can, set NO_INFOMSGS (and MAXDOP in SQL Server 2016+)**
- **Free, widely used, and easy-to-use set of maintenance scripts available from Ola Hallengren at <http://ola.hallengren.com/>**

How Often Should Consistency Checks be Run?

- **It all depends on a combination of:**
 - Stability of I/O subsystem
 - Backup strategy
 - Acceptable downtime if corruption occurs
 - Acceptable data loss if corruption occurs
 - Time window available to take the extra I/O and CPU load
 - What kind of system it is (e.g. production, test, backup)
- **I always like to recommend at least once a week**
- **Examples:**
 - No backups and persistent corruptions
 - VLDB with very low downtime/data loss tolerance

How Long Will CHECKDB Take to Run?

- **Depends on many factors:**
 - Size of the database
 - Concurrent I/O load on the server
 - Concurrent CPU activity on the server
 - Concurrent update activity on the database
 - Throughput capabilities of the I/O subsystem
 - Number of CPUs on the server
 - Speed of the disks where TEMPDB is placed
 - Complexity of the database schema
 - Which options were specified
 - Number and type of corruptions that exist
- **See blog post for more details:**
 - <http://www.sqlskills.com/blogs/paul/checkdb-from-every-angle-how-long-will-checkdb-take-to-run/> (<http://bit.ly/RRL570>)

How To Consistency Check a VLDB?

- **Consistency checking a multi-terabyte database can take many hours!**
- **Four options for reducing the run time:**
 - Don't run any checks (don't do this!)
 - Use DBCC CHECKDB options to reduce time and resources necessary
 - Break up the checks over time using other DBCC commands
 - Use another system to run the consistency checks

Consistency Checking Using Another Server (1)

- This method removes the consistency checking workload from the production server
- **Methodology:**
 - Perform full database backup on the production server
 - Restore database on another server
 - Run consistency checks on the restored database
- **What to do if a corruption is found?**
 - Is it the I/O subsystem on this server?
 - Is it the backup file?
 - Is it the database on the production server?
- **If corruption is found, which should be rare, it forces you to run consistency checks on the production server**

Consistency Checking Using Another Server (2)

- Some people advocate consistency checking a mirror database or an availability group secondary replica as a way of consistency checking the principal database or primary replica
- This is not valid
- The various servers are stored on different I/O subsystems and corruptions do not propagate between servers
 - E.g. ensuring that a mirror database has no corruption does not imply anything about the state of the principal database
- Consistency checking should ideally be performed on all copies of a database
 - Especially true for availability groups if a secondary replica is being used to offload full backups from the primary replica

Consistency Checking Survey (1)

What method do you use to run consistency checks on your production database (s)?

Run DBCC CHECKDB with no options on the production database		62%	261
Run DBCC CHECKDB WITH PHYSICAL_ONLY on the production database		10%	42
Run DBCC CHECKDB on a restored backup on another server		11%	47
Run DBCC CHECKDB on a database snapshot on a mirror database		1%	4
Spread consistency checking over several days using DBCC CHECKTABLE		0%	1
Spread consistency checking over several days using DBCC CHECKFILEGROUP		1%	4
Use BACKUP WITH CHECKSUM to validate page checksums, no DBCCs		1%	4
Run DBCC CHECKDB on a log shipping secondary, or 2012 AG secondary		1%	3
Run DBCC CHECKDB on a SAN copy/mirror		2%	7
Create your own database snapshot of the production database and run DBCC CHECKDB on it		0%	2
Other? Enter here...		11%	47
Total: 422 responses			

- Source: <http://www.sqlskills.com/blogs/paul/importance-of-how-you-run-consistency-checks/> (<http://bit.ly/Hq3m77>)



47

© SQLskills, All rights reserved.
<http://www.SQLskills.com>

Consistency Checking Survey (2)

How often do you run consistency checks? (regardless of 'how' you run them)

What are consistency checks?		9%	25
Never		5%	14
Only when corruption is detected some other way		8%	22
Only during an event like an upgrade or migration		1%	2
Regularly, but less than monthly		3%	8
Monthly		5%	14
Weekly		37%	103
Daily		25%	69
More frequently than daily		1%	3
Only after performing a restore, or after a failover		1%	3
Other? Enter here...		5%	13
Total: 276 responses			

- Source: <http://www.sqlskills.com/blogs/paul/importance-of-running-regular-consistency-checks/> (<http://bit.ly/1mUuRw>)



48

© SQLskills, All rights reserved.
<http://www.SQLskills.com>

Did Something Go Wrong?

- So you setup a regular job to run consistency checks, and turned on page checksums – how can you tell if it goes wrong?
- There has to be some kind of monitoring otherwise you'll never know!
- Manual monitoring is time-consuming and prone to being forgotten
- **Solution: Agent alerts, or other monitoring (e.g. SCOM)**
- **Create alerts for:**
 - Severity 19 errors and above
 - Any user-defined errors (e.g. flagging that CHECKDB job failed)
 - Anything else you're interested in
- See <http://www.sqlskills.com/blogs/paul/easy-monitoring-of-high-severity-errors-create-agent-alerts/> (<http://bit.ly/hrBiTK>)
- And <http://spaghettidba.com/2011/11/28/email-alert-dbcc-checkdb/> (<http://bit.ly/TFwKdK>)

First Hints of Corruption...

- Application/user connections get broken
- Users report 823 or 824 errors
- Backup jobs start failing
 - Error 3043 – backup detected checksum errors
- Agent alerts start firing
- Maintenance jobs start failing
- Errors in the SQL Server error log
- All these hint that you've got corruption

As Soon As Corruption Is Suspected

- **No need to panic!**
 - Panic leads to mistakes and downtime/data-loss
- **Go to the DR run-book**
- **Determine the extent of the corruption**
 - Run DBCC CHECKDB
 - Look in the SQL Server error log, Windows event log
 - Check maintenance job history
- **Check what backups are available**
- **Wait for CHECKDB to finish before doing anything else**
 - You may not NEED to do anything intrusive/destructive

Running DBCC CHECKDB

- **Use DBCC CHECKDB (yourdb) WITH NO_INFOMSGS**
 - Add ALL_ERRORMSGs if using older than SQL Server 2008 SP1
- **Good idea to know how long it usually takes to run for a database**
 - Allows you to report to management how long before results are known when a disaster occurs
 - Longer run time usually indicates some corruption has been found
- **Wait for the command to complete**
 - It's the only way to know what corruptions you have
 - Don't jump the gun and assume you need to restore
- **Look through the output for anything significant**

Interpreting DBCC CHECKDB Output (1)

- There are over 100 errors that DBCC CHECKDB can output, some with over 200 message states
 - Effectively there are roughly a thousand different corruption conditions that can be reported
- Figuring out what one corruption means isn't too bad
 - MSDN has some of them published
- Figuring out multiple corruptions can become very hard and usually isn't worth the time
- There are some tips and tricks you can use to determine the course of action to take

Interpreting DBCC CHECKDB Output (2)

- Did DBCC CHECKDB fail?
 - If it stops before completing successfully, something bad has happened that is preventing it from running correctly
 - This means there is no choice but to restore from a backup or try exporting the data, as DBCC CHECKDB cannot be forced to run (and hence repair)
- Examples of fatal (to DBCC CHECKDB) errors:
 - 7984 – 7988: corruption in critical system tables
 - 8967: invalid states within DBCC CHECKDB itself
 - 8930: corrupt metadata such that DBCC CHECKDB could not run
- The SQL Server error log message will list an error state
 - See the “Understanding DBCC Error Messages” portion of the Books Online for DBCC CHECKDB for details at <http://bit.ly/179p6At>

Demo

Example fatal errors to CHECKDB

Interpreting DBCC CHECKDB Output (3)

- **Are the corruptions only in nonclustered indexes?**
 - If recommended repair level is REPAIR_REBUILD, then yes
 - Otherwise, check all the index IDs in the errors and if they're all greater than 1, then yes
 - If yes, you don't need to restore or run repair
- **Was there an un-repairable error found?**
 - Examples:
 - 2570 error: invalid data for the column type (data purity error)
 - 8992 error: CHECKCATALOG (metadata mismatch) error
 - 8909, 8938, 8939 (page header corruption) errors where type is 'PFS'
 - None of these can be automatically repaired so your options are to restore or to attempt manual repairs

Manually Fixing Nonclustered Indexes

- It doesn't make sense to put the database offline and run DBCC CHECKDB or DBCC CHECKTABLE with REPAIR_REBUILD to fix corrupt nonclustered indexes
- All it will do is essentially disable and rebuild the index, so why not do it yourself?
- You cannot just rebuild the index
 - Index rebuilds read the old index to build the new index
- Steps to use, inside a transaction:
 - ALTER INDEX name ON tablename DISABLE
 - ALTER INDEX name ON tablename REBUILD
- Using a transaction is necessary to prevent any index-enforced constraints from being violated while the index is disabled

Demo

Nonclustered index corruption

Manually Fixing Data Purity Errors

- **If a 2570 data purity error was found, SQL Server cannot repair it**
 - The error is that a column value is out-of-bounds
 - Which value should SQL Server pick to repair it?
- **You must manually repair data purity errors**
 - Work out which row is invalid using SELECT or DBCC PAGE
 - Update the column to a valid value
 - <http://support.microsoft.com/kb/923247> explains the options
- **Make sure you choose a value that makes sense for your business logic and the data the column value represents**

Demo

Fixing data purity errors

Interpreting DBCC CHECKDB Output (4)

- **Everything else I haven't mentioned I classify as "general corruptions"**
 - Your options are to restore or repair or export to a new database
- **The more corruptions there are, the harder it is to figure out what's actually broken**
 - It also depends on what kind of page is corrupt
- **For example, for a data page in the leaf level of a clustered index:**
 - There may just be one error about one row on the page
 - And there may also be a matching error about each nonclustered index
 - And there may also be a matching error about each off-row LOB value in the row
 - Etc...
 - There may be an error that the entire page cannot be processed
 - And there will also be errors about broken links to other pages
 - And there may also be errors about nonclustered indexes
 - Etc...
 - Etc...

Recovering Using Backups

- **Best way to avoid data loss**
- **Not necessarily the best way to avoid downtime**
 - Depends what kind of backups are available
 - Although data and backup compression help...
- **Plethora of options available**
 - Full database backup is a good starting point
 - Series of transaction log backups as well is much better
- **Remember:**
 - Backups have to exist to be useful
 - Backups have to be valid to avoid data loss

What if SQL Server is Unavailable?

- You always want to try a tail-of-the-log backup
- If the SQL Server instance is unavailable, have to perform a 'hack-attach' of the log file to another instance to perform the backup
- Steps to perform:
 - Create dummy database with same name
 - Set database offline
 - Delete all data and log files
 - Drop in salvaged log file you want to back up
 - Try to bring database online
 - Perform tail-of-the-log backup
- **Note: only works on the same version of SQL Server**

Demo

Last resort tail-of-the-log backup

Single-Page Restore

- **One or more pages can be restored**
 - Incredibly valuable if database not architected for partial database availability
- **Backups must exist to allow the page to be restored to same point in time as PRIMARY filegroup**
- **Not all page types can be single-page restored**
 - Boot page
 - Fileheader pages
 - Allocation bitmaps (not including IAM pages)
 - Certain pages in hidden, critical system catalogs

Demo

Using single-page restore

Restore vs. Repair

- **Did DBCC CHECKDB fail?**
 - Yes – you must restore or export, as you cannot run repair
- **Is it just nonclustered indexes that are damaged?**
 - Yes – neither restore or repair, manually rebuild them
- **Are there any un-repairable errors?**
 - Yes – you must restore or export, or potentially manually repair them
 - Some manual repairs are trivial, but most are (usually very) advanced
- **If you're still able to make a repair vs. restore choice:**
 - Consider your down time and data loss Service Level Agreements
 - Use whichever option you can which allows you to limit down time and data loss while still staying within the SLAs
- **There is a comprehensive flow chart of the decision making process available at <http://bit.ly/VGFOH5>**

How Does Repair Work?

- **What is the purpose of repair?**
 - Make the database structurally consistent
 - Tries to be as fast as possible
- **How does it know what to repair?**
 - From the list of corruptions it just found
- **How does it choose what to repair first?**
 - Runs most intrusive errors first
- **Did it repair everything?**
 - Check the output – count of errors found and fixed
 - Be careful – some corruptions could be masked by others
- **Why aren't repairs online?**
 - Hard enough to get them right *offline*...

Running Repair

- **Not always a last resort compared to restore**
 - Depends what backups are available
 - Depends if downtime is more important than data loss
- **All repair options require SINGLE_USER mode, i.e. they're offline**
 - REPAIR_REBUILD – only rebuilds indexes
 - REPAIR_ALLOW_DATA_LOSS – repairs that will likely lose data
- **Specifying a repair option makes DBCC CHECKDB run single-threaded**
- **Try to repair smallest thing**
 - DBCC CHECKDB then DBCC CHECKTABLE then DBCC CHECKALLOC
- **You can put a transaction around your repair operation and roll it back if you don't like what repair did**
- **Note: most system tables can't be repaired successfully**

Beware of REPAIR_ALLOW_DATA_LOSS

- **It was very deliberately named**
- **It usually fixes structural inconsistencies by de-allocating**
 - This is the fastest and most provably correct way
- **It doesn't take into account:**
 - Foreign-key constraints
 - Inherent business logic and data relationships
 - Replication
- **Before running repair, protect yourself**
 - Take a backup and quiesce replication topologies involved
- **After running repair, check the data**
 - Run DBCC CHECKDB again to make sure all corruptions were repaired
 - Run DBCC CHECKCONSTRAINTS if necessary
 - Reinitialize any replication topologies involved

Misconceptions Around Repair

- Repair will not cause data loss (it depends)
- Repair should be run as the default (no)
- You can run repair without running DBCC CHECKDB (no)
- As soon as you've run repair, continue as normal (no)
- Repair can always fix everything (no)
- Repair is safe on system databases (no)
- You can run repairs online (no)
- REPAIR_REBUILD will fix everything (no)
- Repair fixes up constraints (no)
- Repairs are propagated to replication subscribers (no)

If You Are Forced To Use Repair...

- That implies that your backup strategy does not allow you to meet your downtime and data loss Service Level Agreements
- Update your backup strategy!
 - Figure out what restores you need to be able to perform
 - Change the backup strategy to perform the backups that will allow those restores to take place
 - Implement regular backup validation
- Also make sure that:
 - You check any constraints that may be affected based on which tables were repaired
 - You check to see what data was lost
 - You reinitialize any affected replication topologies
 - Perform root-cause analysis

Demo

Using repair

Metadata Corruption

- Someone manually changed system tables in SQL Server 2000...
- Hidden system tables are accessible only when the Dedicated Admin Connection (DAC) is used
 - sqlcmd -A or prefix the SSMS connection string with "admin:"
 - 2012+: Error message in SSMS can be ignored, it's from Intellisense
 - 2012+: TF7806 at startup plus running SQL Browser allows DAC connections under all circumstances, including named instances
 - SQL Performance blog post on this at <http://bit.ly/RKxGOV>
- These tables cannot be changed unless the server is booted with -m
- From SQL Server 2008 onward, if a system table is directly modified, it is noted in the database metadata
 - **This modification makes your database technically unsupported**
 - Check DBCC DBINFO
 - If you update a system catalog using 3-part naming, the dbi_updSysCatalog of the database you're in is erroneously updated

Rebuilding System Table Indexes

- Structurally corrupt system table clustered indexes cannot be repaired successfully
- Structurally corrupt system table nonclustered indexes sometimes can be rebuilt by repair
 - You may have to play around with DBCC CHECKTABLE to find which table is corrupt as DBCC CHECKDB may fail
 - This is not guaranteed to work...be careful as it may make things worse
- I would try to avoid running repair on system tables completely
 - Or at least test it on a copy of the database

Demo

Fixing metadata corruption

Damaged/Missing Log with Clean Shutdown

- The transaction log is essential for crash recovery to work
- However, a damaged or missing transaction log does not matter if the database was cleanly shut down
 - The transaction log is not required as there is no crash recovery to run
- In that case, a new transaction log can be created
 - Requires re-attaching the database using CREATE DATABASE and using either the FOR ATTACH or FOR ATTACH_REBUILD_LOG options
 - FOR ATTACH creates a single minimum-size transaction log file, but only works if there was previously one transaction log file
 - FOR ATTACH_REBUILD_LOG also creates a single minimum-size transaction log file, but works no matter how many log files there were previously
- Make sure to provision the correct size and auto-growth afterwards

EMERGENCY Mode

- Without a clean shutdown, if the transaction log is damaged or missing, the database will be in one of two states:
 - RECOVERY_PENDING: crash recovery has to run but could not start
 - SUSPECT: crash recovery started but could not complete
- EMERGENCY mode is also known as “bypass recovery” mode, as this is exactly what it instructs the Storage Engine to do
 - ALTER DATABASE mydb SET EMERGENCY
- Although this allows access to the database, the data is transactionally inconsistent (and maybe structurally inconsistent)
- You can try to export data out into a new database
- Without backups, the only way to recover the existing database is to try EMERGENCY-mode repair

EMERGENCY-Mode Repair

- This was added to SQL Server 2005 as a documented and supported way to attempt recovery with a damaged transaction log
 - It used to be possible using DBCC REBUILD_LOG in SQL Server 2000
- The database must be set to EMERGENCY and SINGLE_USER
- Running DBCC CHECKDB (mydb, REPAIR_ALLOW_DATA_LOSS) does the following in EMERGENCY mode:
 - Run as much crash recovery as possible, skipping damaged log records
 - Build a new transaction log file
 - Run a full DBCC CHECKDB with REPAIR_ALLOW_DATA_LOSS
 - Try to bring the database online
- **This is not guaranteed to work in all scenarios**
 - Severe damage to data files may prevent the database being brought online
 - File system damage may prevent the log file being recreated

Detached SUSPECT Database

- When attaching a database, crash recovery must be completed before the database will attach
- If a detached database requires crash recovery and the transaction log is damaged or missing, the attach will fail
- Prior to SQL Server 2008 it was possible to detach a SUSPECT database
- Reattaching a database in this state requires the following:
 - Create a database with the same name and file IDs as the detached database
 - Set the dummy database offline and delete the log and data files
 - Copy in the files from the detached database
 - Try to bring the database online again
- Once the database is attached, you can proceed to EMERGENCY mode

Damaged/Missing Log of Attached Database

- If the SQL Server runtime encounters transaction log corruption, for instance while rolling back a transaction, the database will be set offline with a status of SUSPECT
- Recover from this using backups or EMERGENCY mode
- If there are no active transactions that could encounter the corruption, DBCC CHECKDB or a transaction log backup may fail
- To recover from this:
 - Switch the database to the SIMPLE recovery model
 - Issue a CHECKPOINT to force the transaction log to clear
 - Switch back to the original recovery model
- This should hopefully clear the damaged log records
- Note: this breaks the log backup chain

Demo

Using EMERGENCY mode

Hex Editor Salvage of Boot Page

- If the boot page (1:9) or file header pages of data files in the primary filegroup are damaged, you cannot attach or online a database
- These pages cannot be repaired
- Simplest fix is to copy them from an older restored backup
- Use a hex editor to do this
 - XVI32 or HxD are great freeware hex editors
 - XVI32 will truncate files bigger than 2GB!
- Open both files at the correct offset: select, copy, paste, success!
 - Offset of the boot page will be $9 \times 8192 = 73728$
 - Select 8192 bytes, copy, paste at the same offset in the corrupt file

Salvage From Nonclustered Indexes

- If repair is the only option and table structures are damaged, you may be able to recover data from nonclustered indexes
 - May also be the case if repair is impossible because of corrupt metadata
- Create SELECT statements to:
 - Figure out which table rows will be lost from running repair
 - Force column selection from non-damaged nonclustered indexes
- Success rates will vary depending on the amount of column coverage in the nonclustered indexes
- Do this BEFORE running repair, as the repair will rebuild nonclustered indexes based on the repaired heap or clustered index

The Ultimate Advanced Recovery

- **Manually editing data files using a hex editor or DBCC WRITEPAGE is the most advanced method of corruption recovery**
 - Usually requires very detailed knowledge of database structures
- **Always take a backup before manually editing a database**
 - If something goes wrong, you can still get back to the starting point
- **Paul usually does this once or twice a year for clients**
 - Some examples:
 - Manually swapping allocation units in broken system tables
 - Deallocating a broken page to allow data export when other metadata problems prevent DBCC CHECKDB repair from running
 - Fixing a broken boot page by replacing with an old one
- **Always run DBCC CHECKDB to make sure you haven't made things worse and have fixed the original problem**
- **Always do it on a copy of the corrupt database**
 - If a backup/restore won't work, use a file system copy and hack attach

Any Other Ways to Salvage Data?

- **Possible other places to get data:**
 - Data warehouse
 - Test/dev systems
 - 3rd-parties that host/consume some of your data
 - Replication subscribers
 - Hard-copy printouts
- **I've heard some incredible stories of tenacity in the face of corruption**
 - "Unbelievable tale of disaster recovery" at <http://bit.ly/185Z2bl>

Customer Examples to Learn From

- All of the following examples happened while Paul was at Microsoft
- Paul was involved with each customer
- No names will be divulged
- In increasing magnitude of loss
- These are just the worst ones...

Examples to Learn From #1

- **Story**
 - Database is corrupt and takes literally *days* to bring online
 - Customer is a small bank in the US
- **Cause**
 - Customer only had a full backup from January 2004 plus log backups every ½ hour until April 2004 when corruption hit
- **Resolution**
 - Restored ALL the backups (approx 5000!) – luckily none were corrupt
- **Cost**
 - Aggravation to bank customers while bank was down for several days
- **Lesson Learned**
 - Plan a backup strategy that allows an efficient and quick restore

Examples to Learn From #2

- **Story**
 - Database lost with corrupt backup
 - Customer is a large investment firm and the database stores all 401k data for all its customers
- **Cause**
 - Database was backed up prior to moving to new hardware
 - Old h/w flattened *before* database restored on new h/w
 - Backup corrupt
- **Resolution**
 - Force corrupt backup to restore and patch together broken data
- **Cost**
 - Job losses (from DBA to VP level)
- **Lessons Learned**
 - Always validate backups
 - Always restore and check *then* delete

Examples to Learn From #3

- **Story**
 - Database and application down for 3 days after corruption
 - Database stores stock trades for UK customers of a large US bank
- **Cause**
 - Started DBCC CHECKDB but took too long so killed it
 - Started to restore from backups but found they were corrupt
 - Re-ran DBCC CHECKDB again and let it complete
- **Resolution**
 - Eventually found it was a nonclustered index corruption that could have been fixed with an index rebuild
- **Cost**
 - Customer fined >\$10m by UK government
- **Lessons Learned**
 - Know how long DBCC CHECKDB takes, and it may take longer with corruption
 - Validate backups

Examples to Learn From #4

- **Story**
 - DBA comes to work to find the database and all backups gone
 - Database stores all tax records for a US state
- **Cause**
 - 3rd party technician accidentally formatted the drive where the database *AND* backups (!!!) were stored while installing hardware
- **Resolution**
 - 200 staff members spent 3 weeks entering data for all state residents from scratch
- **Cost**
 - Unsubstantiated report of cost to the state of more than \$2 *billion* in lost tax revenue
 - No job losses...
- **Lesson Learned**
 - Always have multiple backups, including off-site
 - Carefully guard access to infrastructure

Review

- I/O errors and what they mean
- Page protection options
- DBCC CHECKDB
 - What it does
 - Best practices and FAQ
 - Interpreting CHECKDB output
 - How repair works
- Recovering from corruption
- Horror-stories from the wild

Blog Posts

- **Blog post series**

- <http://www.sqlskills.com/blogs/paul/category/corruption/>
- <http://www.sqlskills.com/blogs/paul/category/checkdb-from-every-angle/>
- <http://www.sqlskills.com/blogs/paul/category/disaster-recovery/>
- <http://www.sqlskills.com/blogs/paul/dbcc-writepage/>

- **Pluralsight**

- SQL Server: Detecting and Recovering from Database Corruption
- SQL Server: Advanced Corruption Recovery Techniques



93

© SQLskills, All rights reserved.
<http://www.SQLskills.com>

Questions?

