IDERA® | SQL Server Whitepaper

5 COMMON INTEGRITY ISSUES

Authors: **Kenneth Fisher and Robert Davis**

The word "integrity" has multiple meanings in the SQL Server world. One meaning deals with the quality of data stored within SQL Server. This form of integrity is maintained through proper data typing, constraints (default, check, unique, and key), and data validation processes. This is the application/database design side of things. The other definition of integrity deals with the logical and physical consistency of the underlying structures of a database. This is an important part of keeping the lights on and the focus of this whitepaper.

WHAT IS INTEGRITY?

Integrity is ensuring that the logical and physical data stored in SQL Server are structurally sound and consistent. Simply put, it means that the data present in SQL Server is written correctly and is where it is expected to be. Why is this important? Simply put if SQL Server can't read or find the data then it may as well no longer exist. One bit off in the wrong page can cause a loss of the entire database. There are different tolerances for data loss of course, but I think we can all agree that it's bad.

SQL Server has functionality built in for protecting the integrity of data written to the disk. Some of this functionality is enabled by default and some is not. Even the integrity functionality that is enabled by default is not in use by a lot of the systems out there. You can turn it off without realizing if you are not careful.

PAGE VALIDITY

To start with there is a setting that tells SQL Server to verify the validity of each page as it is read from disk into memory. This setting is called PAGE_VERIFY. Beginning with SQL Server 2005, the default page verification method is CHECKSUM. If you create a new database, it will automatically inherit this setting. CHECKSUM is the recommended method of page verification. However, prior to the introduction of CHECKSUM, the default page verification method was TORN PAGE DETECTION. If you upgrade an earlier version of a database to SQL Server 2005 or newer, the page verification method will not be updated. If you do not manually change this setting, your database will not be as protected as it could be.

There are three options for the PAGE_VERIFY setting of a database:

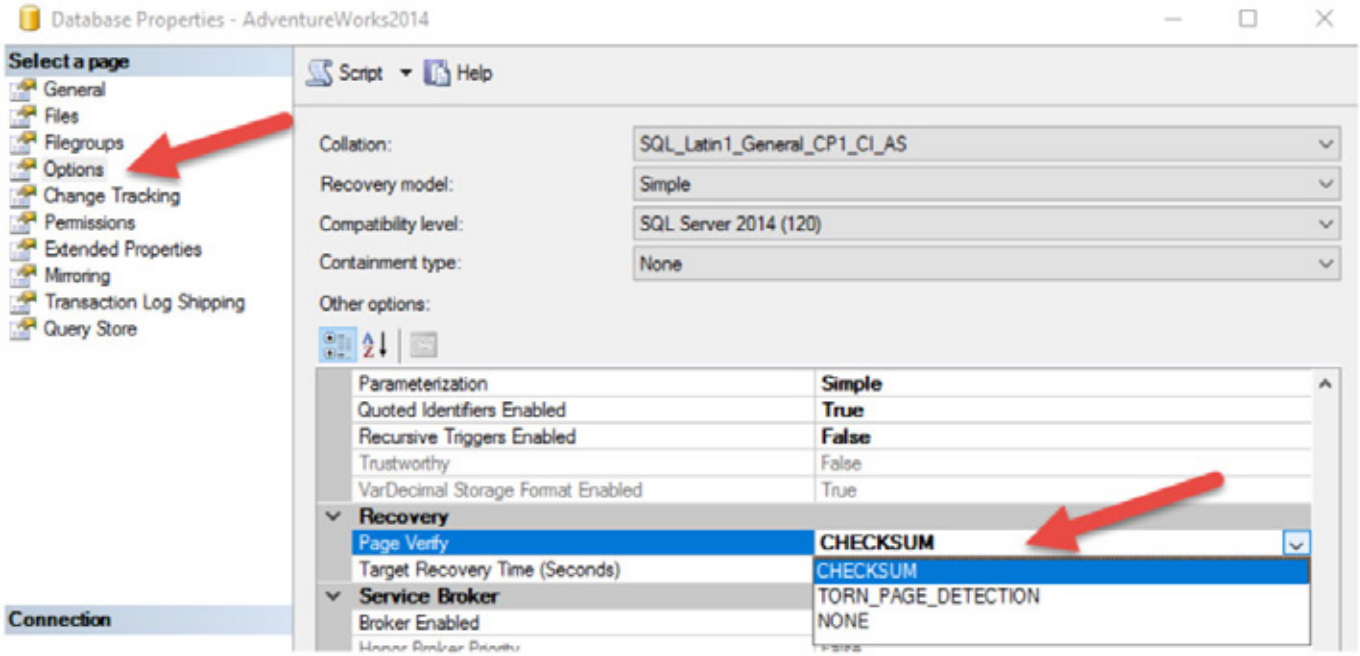
- **CHECKSUM:** The most comprehensive page verification option. When a page is written, a checksum value for the whole page is written in the header of the page. When the page is read back into SQL Server, the checksum value is recalculated, and if the new value does not match, then the page is deemed corrupted. Any change whatsoever to the page after it is written will result in a different checksum value at read time.
- **TORN_PAGE_DETECTION:** The second best option for page verification. TORN PAGE DETECTION works by writing a two-bit pattern in the page header and repeatedly throughout the data page. The two-bit pattern alternates at each point, and if the correct pattern is not found at all points on the page, the page is considered torn or incompletely written. It may be that the write of the page to disk was interrupted and did not complete.
- **NONE:** There is no method of page verification in use. NONE has never been the default page verification setting. Prior to TORN_PAGE_DETECTION, there was not a PAGE_VERIFY setting. If you find databases on your servers that are set to NONE, then someone has probably deliberately changed the setting. If someone makes that change, have them read this document. If they still feel that NONE is an appropriate setting it might be time to consider a new profession.

You can check the page verification method of a database by looking at the Database Properties dialog or by querying the sys.databases system catalog. Additionally, the DATABASEPROPERTYEX() system function can be used to determine if TORN_PAGE_DETECTION is enabled.

```
Select  name,
        page_verify_option,
        page_verify_option_desc
From  sys.databases;

Select  DATABASEPROPERTYEX(N'<database name>',
'IsTornPageDetectionEnabled');
```

Any databases that are not currently set to CHECKSUM can easily be modified by using the ALTER DATABASE command or in the options tab of the database properties dialog.




```
ALTER DATABASE <dbname> SET PAGE_VERIFY CHECKSUM;
```

A very important thing to remember about both CHECKSUM and TORN_PAGE_DETECTION is that simply enabling the option does not mean that the data is protected. The checksum value and torn page bits do not get written to the data pages immediately. Each page will be protected the next time it is written out to disk. The approach I like to recommend is to rebuild all indexes and tables (heaps) during the next opportunity. Rebuilding all indexes and tables will cause all data to be re-written to disk and ensure that all data is protected as well as performing optimally.

BACKUP CHECKSUMS

In addition to being a more comprehensive method of page verification, the CHECKSUM option provides functionality for additional checks of the data pages. The BACKUP and RESTORE commands also have a CHECKSUM option. This option is not enabled by default, and you have to explicitly use it in your commands. When SQL Server is writing the data pages to the backup file and if the data pages have checksum values applied to them, it will recalculate the checksum values and fail the backup if the checksum value does not match. Likewise, when restoring a backup with the CHECKSUM option, it will recalculate any existing checksum values as it reads the data pages and fail the restore if corruption is found. For the performance minded among you, this option has a small but negligible cost.

```
BACKUP DATABASE AdventureWorks2014 TO DISK = 'C:\Backups\AdventureWorks2014.bak' WITH INIT;
GO
BACKUP DATABASE AdventureWorks2014 TO DISK = 'C:\Backups\AdventureWorks2014.bak' WITH INIT, CHECKSUM;
GO
```

```
Processed 25032 pages for database 'AdventureWorks2014', file 'AdventureWorks2014_Data' on file 1.
Processed 2 pages for database 'AdventureWorks2014', file 'AdventureWorks2014_Log' on file 1.
BACKUP DATABASE successfully processed 25034 pages in 8.887 seconds (22.006 MB/sec).
Processed 25032 pages for database 'AdventureWorks2014', file 'AdventureWorks2014_Data' on file 1.
Processed 2 pages for database 'AdventureWorks2014', file 'AdventureWorks2014_Log' on file 1.
BACKUP DATABASE successfully processed 25034 pages in 9.031 seconds (21.655 MB/sec).
```

The CHECKSUM option also generates a checksum value for the backup file and evaluates that at restore time before restoring the database. If any part of the backup file is corrupted after being written to disk, the checksum value will not match and the restore will fail immediately rather than running until the actual corrupt page is discovered. This can save you a lot of time when restoring a corrupted backup file of a large database.

The main benefits of using the CHECKSUM option is quickly finding potential corruption and preventing the spread of corruption to other servers. Without the CHECKSUM option, you can back up a corrupted database and restore it in another environment or server without noticing it.

This does bring up a good point. Before making any attempt to fix a corrupted database it's a good idea to take a backup just in case. In order to take a backup on a corrupted database you will need to exclude the CHECKSUM option and include the CONTINUE_AFTER_ERROR option. Based on the name it should be pretty obvious that this option allows the backup to continue even if there are errors during the backup.

WHICH PAGES FAILED?

Any time one of these tests is failed the failing page is noted in the msdb.dbo.suspect_pages system table. This table can be read from with a simple select.

```
SELECT * FROM msdb.dbo.suspect_pages;
```

Be warned that this table can only hold 1000 entries and must be cleared out manually. Of course if you have had more than 1000 bad pages in any reasonable amount of time you may very well have more serious problems than this table being full.

CALL TO ACTION

There are four things that every database administrator should do to protect the integrity of their databases.

- Make sure all databases are being backed up frequently
- Ensure that all databases are set to PAGE_VERIFY of CHECKSUM
- Use CHECKSUM for all backups and restores
- Execute regular integrity checks of all of your databases

We have already covered two of these things just by discussing integrity. The last action item is the main focus of this whitepaper. It is very crucial that you regularly check the integrity of your databases. This paper will show you how to do that and how to handle the five most common integrity issues administrators may encounter.

READING INTEGRITY CHECK OUTPUT

SQL Server also provides a DBCC command for checking the integrity of a database. There are a variety of DBCC commands available, but the only command we are concerned with for this topic is DBCC CHECKDB. The DBCC CHECKDB command will check the internal structures, metadata, and data in a database.

Unfortunately DBCC CHECKDB does not check memory-optimized tables. This means that in order to check the integrity of a memory-optimized filegroup you will need to take a backup with CHECKSUM. Depending on the layout of your database a filegroup backup could be faster than a regular full backup. If the backup fails with a CHECKSUM error then you have corruption. At this point there are a limited number of solutions. You can either export your data and rebuild the tables, or you can restore from a good backup. You also need to be aware that if the instance starts up and corruption in the memory-optimized tables is discovered as the data is loaded into memory, the entire database will be unavailable.

For all other parts of the database DBCC CHECKDB is an integral part of maintaining the integrity of the database. In its simplest form, it can be executed by calling DBCC CHECKDB (<Database Name>);. For example, if I wanted to check the integrity of tempdb, **I would execute the following:**

```
DBCC CHECKDB (tempdb);
```

The output of DBCC CHECKDB can be confusing even for a clean result with no corruption. DBCC CHECKDB outputs a lot of information, most of it informational unless corruption is found. When output is dumped to the messages tab of a SQL Server Management Studio query window the errors are in red making them somewhat easier to see. Unfortunately the amount of data generate can still make these difficult to find and once the text is removed from the messages tab the red highlight is gone. Fortunately, the command has optional arguments that allow us to filter out the informational messages and focus on the error messages, if any are returned.

There are three options that you should commit to memory:

- **NO_INFOMSGS:** filters out all informational messages in the output.
- **ALL_ERRORMSGS:** returns all error messages. Prior to SQL Server 2008 SP1, CHECKDB only returns the first 200 error messages by default. Beginning with SQL Server 2008 SP1, running the command outside of SQL Server Management Studio (SSMS) returns all error messages by default and this argument has no effect. In SSMS with SQL 2008 SP1 and newer, SSMS limits the output to 1,000 error messages unless the argument is used.
- **TABLERESULTS:** returns the information in a neat, easy-to-read grid format rather than free-form text.

The image below is sample output in table format taken from Idera's free tool SQL Integrity Check. The sample corrupt databases used in the examples in this whitepaper can be downloaded here:
(RecoveringFromCorruption.zip (13.9 MB)

Database	Last Known Good	Last Checked	Status
PFSCorruption	Never	2016-10-26 20:36:23	Possible Corruption

Results for: PFSCorruption

Error	Level	State	Message	Repair Level
8948	16	5	Database error: Page (1:6) is marked with the wrong type in PFS page (1:1). PF...	
8948	16	5	Database error: Page (1:7) is marked with the wrong type in PFS page (1:1). PF...	
8954	10	1	CHECKDB found 2 allocation errors and 0 consistency errors not associated wi...	
8989	10	1	CHECKDB found 2 allocation errors and 0 consistency errors in database 'PFSC...	

The command I would execute to see the limited output for this sample database is:

```
DBCC CHECKDB (PFSCorruption) WITH NO_INFOMSGS, ALL_ERRORMSGS, TABLERESULTS;
```

To see the full supported list of arguments see the SQL Server Books Online entry for DBCC CHECKDB: http://technet.microsoft.com/en-us/library/ms176064.aspx.

If you look at the Books Online page listed above, you will see that one of the arguments I listed is not mentioned. The TABLERESULTS option is not documented for this command (it is documented for a few DBCC commands) and is therefore not officially supported. The argument works for almost all DBCC commands though, and it is safe to use.

When you encounter corruption, the output can be somewhat daunting (and a little scary) at first. The key is to know where to focus your attention. If you look at the Level column, you will see that it returns different values for different messages. Level correlates directly to the severity of the error message. The higher the Level, the more severe and more important the message is. Your first tip is to focus on the messages with the highest level of 16.

If you are using SQL Integrity Check that I mentioned above, you can simply click on the Level column header to sort the output by Level. Click it again to sort in the opposite direction. If running the command manually, you will need to sort through the messages manually to find the most severe messages.

The MessageText column (or Message column if using SQL Integrity Check) for the Level 16 error messages will contain just about everything you need to know to figure out what kind of corruption you have, how wide-spread the corruption is, and how best to deal with it with minimal or no data loss. The message will tell which pages in which files have corruption in the format of (<file number>:<page number>). In the image above, you see three different pages listed in the Level 16 messages: (1:1), (1:6), and (1:7). The messages are referring to page numbers 1, 6, and 7 in file number 1.

You can query sys.database_files or sys.master_files to see which file holds the corrupted page. These particular messages indicate a problem with only one page though. If you read the messages carefully, it says that page (1:1) has incorrect values for pages (1:6) and (1:7). Page (1:1) is the only corrupt page detected in this example. This brings me to the second tip.

Once you know which messages to focus on, identify what is corrupted. Determine the object that is corrupted and that will tell you what kind of corruption you have. The messages can tell you a wide range of ID values depending on what is corrupted. Focus on the IDs that it gives you and narrow it down to exactly what is corrupted.

If it reports the Object ID, this will tell you which table contains the corruption. Along with the Object ID, it should report an index ID as well. If you are dealing with a corrupt index or heap, the index ID will tell you what type of index contains the corruption. The type of index dictates how you will handle the corruption. We will discuss how to handle corruption in clustered indexes or heaps (index ID 1 or 0) and nonclustered indexes (index IDs greater than 1) later on.

The last two common integrity issues we will talk about are found in the wording of the error messages and not by the ID values being reported. We will look at issues when the value stored in a table is out of range for its data type, also known as data purity errors. Lastly, we will look at chain linkage problems where pointers from one page are pointing to an incorrect page. For this particular error, the error message clearly calls out that there are possible chain linkage problems.

Before we start talking about how to deal with the common integrity issues, I have one final tip for reading the CHECKDB output, and it is a very important one. Even if you forget the other two tips, please commit this one to memory.

The final tip is to ignore the RepairLevel column (Repair Level in SQL Integrity Check). Do not consider using the repair options for CHECKDB until absolutely all other avenues have been exhausted. The repair options should be your absolute last resort only. If you're not an expert at dealing with corruption, you should enlist the help of an expert before running any repair on the database. The repair options may delete data pages in order to get rid of corruption, and once deleted, they cannot be recovered except by manually re-entering the data.

To summarize the tips:

1. Limit and simplify the DBCC CHECKDB output by using NO_INFOMSGS, ALL_ERRORMSGS, and TABLERESULTS
2. Focus in on the error messages with a Level 16 severity
3. Read the error messages carefully
4. Check the IDs reported in the error messages to determine exactly what objects and indexes are affected by the error
5. Ignore the repair options and do not consider using repair until it is your only remaining option

CORRUPTION IN ALLOCATION PAGES

For the above example, the only key IDs the messages gave us were the page IDs. The first 8 pages of a data file are system pages commonly referred to as allocation pages. The pages actually contain more than just allocation information, but for consistency with standard terminology, we will use the term allocation pages. **The first 8 pages of a file are:**

- **Page 0:** File header page
- **Page 1:** First Page Free Space (PFS) page
- **Page 2:** First Global Allocation Map (GAM) page
- **Page 3:** First Shared Global Allocation Map (SGAM) page
- **Page 4:** No longer used
- **Page 5:** No longer used
- **Page 6:** First Differential Changed Map (DCM) page
- **Page 7:** First Bulk Changed Map (BCM) page

Pages 1, 2, 3, 6, and 7 also repeat throughout the file. For the sake of brevity, I'm not going to explain the purpose of the different allocation pages; however, it is important to know how often they repeat throughout the files. If the corruption is in a very high number page, you will have to do some calculations to determine what type of page is affected by the corruption. It is only necessary to calculate this when there is no Object ID reported in the error messages. If it is associated with an object, it is not an allocation page. The formula for determining the type of page experiencing contention is:

- File Header: Page ID = 0 (does not repeat)
- PFS: Page ID = 1 or Page ID % 8088
- GAM: Page ID = 2 or Page ID % 511232
- SGAM: Page ID = 3 or (Page ID – 1) % 511232
- DCM: Page ID = 6 or (Page ID – 6) % 511232
- BCM: Page ID = 7 or (Page ID – 7) % 511232

The following query will calculate the page ID for you and tell you if it is an allocation page:

```
DECLARE @Page int;
SET @Page = <page ID>;
SELECT PageType = CASE
        WHEN @Page = 0 Then 'File Header Page'
        WHEN @Page = 1 Or @Page % 8088 = 0
            THEN 'PFS Page'
        WHEN @Page = 2 Or @Page % 511232 = 0
            THEN 'GAM Page'
        WHEN @Page = 3 Or (@Page - 1) % 511232 = 0
            THEN 'SGAM Page'
        WHEN @Page = 6 Or (@Page - 6) % 511232 = 0
            THEN 'DCM Page'
        WHEN @Page = 7 OR (@Page - 7) % 511232 = 0
            THEN 'BCM Page'
        ELSE 'Not an allocation page'
END;
```

Once you know that you are dealing with an allocation page, deciding on a course of action is simple. Allocation pages cannot be single-page restored (more on this process later). They cannot be repaired by the CHECKDB repair process. You are facing a restore of the whole database. The good news is that you can back up the tail of the log file (if in full or bulk-logged recovery model) so that you can restore with no data loss. This is assuming that you have good backups to restore from. If you were backing up with the CHECKSUM option like I mentioned previously, the odds of having good backups to restore from goes way up.

If you do not have backups to restore from or you are in simple recovery mode and would lose all data since the last full or differential backup, your last remaining option is to try to BCP as much data as possible out of the database, manually recreate the database and all objects, and reimport the data.

CORRUPTION IN CLUSTERED INDEXES AND HEAPS

If the corruption occurs in index ID of 0 or 1, then you are dealing with a heap or a clustered index. One of the key things to keep in mind here is that a heap or clustered index is the base underlying data. You cannot fix corruption here by rebuilding or recreating anything. You will likely hear recommendations to drop and recreate the clustered index or rebuild the clustered index or table. These actions cannot make missing data magically reappear.

If you run DBCC CHECKDB on the sample corrupt database CorruptDB (Demo_ClusteredIndexCorruption.zip (5.03 MB), you will receive a message similar to the below message:), you will receive a message similar to the below message:

```
Object ID 245575913, index ID 1 ... Page (1:298) could not be processed. See other errors for details.
```

The key pieces of information here are the Object ID (resolves to dbo.FactInternetSales), index ID of 1 (clustered index), and page 1:298. It is a single page of a clustered index.

If the corruption is in a heap or clustered index, the proper way to fix it is through restore. If the corruption is not too wide-spread, single-page restores is the quickest way to get a large database fully back online. If the corruption is wide-spread, it may be quicker to restore the full database than to perform a series of single-page restores. At that point, it's a judgment call.

If you decide to try a single-page restore, the first thing you should do is to verify your backups. You will need a full or differential backup with a good version of the page in it, and you will need all transaction log backups, including the current tail of the log, since the full or differential backup to bring the page current with the rest of the database. The transaction log backups are absolutely required. Do not even try a single-page restore without all of the required backups.

We identified above in the sample corrupt database that it is a single-page of a clustered index. After verifying that I have proper backups to support a single-page restore, I start the restore process. The steps I follow are:

1. Put the database in RESTRICTED_USER mode
2. Restore the specific page from a full or differential backup
3. Restore all transaction log files since the full or differential backup using NORECOVERY
4. Back up the tail of the log file
5. Restore the tail of the log with NORECOVERY
6. Recover the database
7. Run DBCC CHECKDB to ensure corruption is gone
8. Set the database back to MULTI_USER

If you are new to performing restores, this may seem daunting the first few times because the restore wizard does not support the options you need. You must perform the restores using T-SQL. The SQL commands are actually very simple. The initial restore of the page from the full or differential backup requires specifying which page to restore. The transaction log backups are very simple because you don't have to specify any special settings other than NORECOVERY.

Let's walk through the restore process for the sample corrupt database:

```
-- Only the one page is corrupt, so let's do a page restore
-- Switch to master to restore the damaged page
USE master;
GO
 -- Set the database in restricted user mode to keep average users out
ALTER DATABASE CorruptDB SET RESTRICTED_USER WITH ROLLBACK IMMEDIATE;
GO
-- Restore the corrupt page from the GOod full backup
RESTORE DATABASE CorruptDB
PAGE = '1:298'
FROM DISK = '<Path to full backup>\CorruptDB.bak';
GO
-- Restore the 1st pre=existing log backup to bring the page current
-- SQL knows which transations to apply, no need to specify any special commands
RESTORE LOG CorruptDB
FROM DISK = '<Path to log backups>\CorruptDB.trn'
WITH NORECOVERY;
GO
-- If there were more pre-existing log backups, we would restore them in order
-- Now backup the tail of the log...
BACKUP LOG CorruptDB
TO DISK = '<Path to tail log backup>\CorruptDB_LOG_TAIL.trn'
WITH INIT;
GO
```

```
-- Restore the tail of the log bringing the page current
RESTORE LOG CorruptDB
FROM DISK = '<Path to tail log backup>\CorruptDB_LOG_TAIL.trn'
WITH NORECOVERY;
GO
-- Finally, recover the database to bring it online
RESTORE DATABASE CorruptDB WITH RECOVERY;
GO
-- Finally, recover the database to bring it online
RESTORE DATABASE CorruptDB WITH RECOVERY;
GO
-- Recheck the database for corruption again
DBCC CheckDB (CorruptDB) WITH ALL_ERRORMSGS, NO_INFOMSGS, TABLERESULTS;
GO
-- Allow users back in
ALTER DATABASE CorruptDB SET MULTI_USER;
GO
```

If CHECKDB reports that there is no more corruption, then you know you have been successful. You quickly recovered corrupt data with no data loss and minimal downtime. If it still reports that the corruption exists or if new corruption is found, it's time to consider doing a full restore instead of a single-page restore.

CORRUPTION IN NONCLUSTERED INDEXES

After you have dealt with several incidents of corruption, you will find yourself hoping for an index ID greater than 1 when you encounter corruption. Corruption in nonclustered indexes is the easiest form of corruption to fix as long as the underlying heap or clustered index is not corrupted.

When the error message from DBCC CHECKDB tells us an index ID, we can tell what kind of index it is by the ID value. A heap is index ID = 0, a clustered index is index ID = 1, and a nonclustered index is index ID > 1. To fix this corruption scenario, we need to know the names of the object and index involved.

If we run DBCC CHECKDB on the sample corrupt database (CorruptionDemo_AdventureWorksDW2012.zip (12.22 MB), it will report an Object Id = 341576255 and an index ID = 2. We can query sys.indexes to determine both of these values:

```
SELECT Object_Schema_Name(object_id) + N'.' + Object_Name(object_id) AS TableName,
        name AS IndexName
FROM sys.indexes
WHERE object_id = 341576255
    AND index_id = 2;
```

Fixing this form of corruption is simple. You need to recreate the index. Unfortunately, you cannot rebuild the index because the rebuild process uses the existing index to build the new one. The same is true for creating the index with the DROP EXISTING option. You have two options for this. You can drop the existing index and then recreate it new or you can disable the existing index and rebuild it. When you rebuild a disabled index, it uses the underlying data rather than the existing index to rebuild it.

Fixing the sample corrupt database is very simple:

```
-- Drop and create
DROP INDEX dbo.FactResellerSales.IX_FactResellerSales_CurrencyKey;

CREATE INDEX IX_FactResellerSales_CurrencyKey
        ON dbo.FactResellerSales(CurrencyKey);
```

CHAIN LINKAGE PROBLEMS

The next integrity issue is a metadata corruption issue. It is possible in certain situations for the links from one page to another to get corrupted. All pages in an object or index have pointers to the previous and next pages in the object or index. If these links point to the wrong page, it can result in pages getting skipped and can even cause an infinite loop in a scan. One way this can occur is if the server crashes in the middle of pages getting flushed to disk. If some pages are updated and others are not, some of the pages may point to the wrong page

It is tempting to perform a repair of the table with the invalid links. The repair option here fixes linkage problems by deleting the bad pages and making sure everything that remains has valid links. In other words, data loss. Do not try this option unless it is your last resort or if you don't care about data loss. There are a couple of tactics you can take with this issue. If the issue is with the base table (heap or clustered index), you can try to .BCP the data out of the table, truncate or drop and recreate the table, and then reimport the data. This should get rid of all of the tables with linkage problems and leave you with a clean table if that was your only problem. If it is with a nonclustered index, just drop and recreate the index or disable and rebuild the index.

If the linkage issues are coupled with other forms of corruption, I will often opt to go straight to a full restore of the database. Trying to fix multiple forms of corruption can be problematic as one form of corruption can sometimes prevent the preferred fix for another form of corruption. For example, if the linkage problem is with a nonclustered index and the clustered index also has corruption, then the clustered index corruption would prevent us from recreating the nonclustered index.

In short, the best options are to reload or recreate the table or index.

DATA PURITY ERRORS

Data purity issues are when the value stored in a column is out of range for the data type. This is a rare error for databases created in SQL Server 2005 or newer. The most common occurrence of this integrity issue is for databases that were originally created in SQL Server 2000 or older. It was a lot easier for invalid values to end up in columns in older versions. Datetime, decimal, and approximate data types like float are affected by this issue.

When you upgrade a database to SQL Server 2005 or SQL Server 2008 from an earlier version of SQL Server (or one that has been upgraded in the past), you should run DBCC CHECKDB with the DATA_PURITY option. This option will validate the data correctness for its data type. If it finds any data purity errors, you must fix the errors manually and re-execute DBCC CHECKDB again. Continue this process until you get a good, clean CHECKDB result.

Once you get a clean result with data purity, the data purity checks will be performed automatically every time DBCC CHECKDB runs. Until you get the clean manual result, you must add the DATA_PURITY option to the command to perform the checks.

Let's assume that you have a SQL Server 2000 database that you just restored on a SQL Server 2008 instance. The database has been upgraded and now you need to run the data purity checks to make sure the data is correct. If it finds out-of-range data, it will report error 2870:

```
Msg 2570, Level 16, State 3, Line 1
Page (1:279), slot 1 in object ID 341576255, index ID 1 ... (type “In-row data”). Column “TaxRate”
value is out of range for data type “float”. Update column to a legal value.
```

How do you fix these errors then? You have to identify the records with the invalid data and update the out-of-range data with valid values. You may have to inspect the data closely, but once you identify the data and update it, you should be able to complete the CHECKDB with no problems. It can be a tedious and frustrating process to find and fix all of the bad data, but it really is your only option other than deleting the data. Better to fix the invalid values now than to have user queries fail because an implicit conversion fails.

CONCLUSION

Corruption and other data integrity issues can be scary to deal with, but if you follow the advice and tips provided in this whitepaper, you will limit your exposure to corruption and always be able to take the quickest route to resolution with minimal or no data loss and minimal downtime. Just remember to stay calm and walk through the steps.

Simply put, the following steps will see you through any corruption scenario:

1. Identify the corruption (run DBCC CHECKDB)
2. Identify the objects and types of objects involved
3. Take the appropriate steps to correct

Step 3 is where the tricky stuff can come into play, but the real deciding factor of success is first doing steps 1 and 2. Simple integrity issues can become major problems when you try to jump straight to step 3. As DBAs, we instinctively want to fix something that is broken, but we have to be careful because jumping to action without forethought can cause more damage than good. Oftentimes, the reactive steps people take can result in a non-recoverable situation.

ABOUT IDERA

IDERA understands that IT doesn't run on the network – it runs on the data and databases that power your business. That's why we design our products with the database as the nucleus of your IT universe.

Our database lifecycle management solutions allow database and IT professionals to design, monitor and manage data systems with complete confidence, whether in the cloud or on-premises.

We offer a diverse portfolio of free tools and educational resources to help you do more with less while giving you the knowledge to deliver even more than you did yesterday.

Whatever your need, IDERA has a solution.