**Automate Data Purity Investigation**

**By** [**Richard Fryar**](http://www.sqlservercentral.com/Authors/Scripts/Richard_Fryar/430436/)**, 2013/08/01**

**FROM:** [**http://www.sqlservercentral.com/scripts/DBCC+CHECKDB/100463/**](http://www.sqlservercentral.com/scripts/DBCC+CHECKDB/100463/)

A couple of weeks ago I wrote an article at [www.sqlcopilot.com/dbcc-checkdb-with-data\_purity.html](http://www.sqlcopilot.com/dbcc-checkdb-with-data_purity.html) about the DATA\_PURITY option of DBCC CHECKDB, and how to identify the affected columns.

One of the ways to find affected columns is to run a SELECT with a WHERE clause to return out-of-range data. But this doesn't always work. It is possible for data to be within a valid range but still fail the data purity check. It is also possible for the column to be totally corrupt, resulting in an arithmetic overflow error when you attempt to SELECT it.

The method I showed in my article for when a SELECT doesn't help was to use DBCC PAGE. But if you have more than a few columns with invalid data, it can be quite time consuming to do. This was the case for me recently when I migrated a database from SQL Server 2000 to 2008 R2 and the CHECKDB found 540,000 data purity errors!

A SELECT for values outside the range for the datatype (decimal(23, 8) in this case) didn't return any data and so DBCC PAGE was the only option - but obviously there was no way I could run it manually 540,000 times!

The script attached to this article was my solution to the problem.

1. It runs DBCC CHECKDB(dbname) WITH DATA\_PURITY, NO\_INFOMSGS, TABLERESULTS and captures the results in a temporary table

2. It extracts the page, slot, object id, column name and data type for each row returned

3. It loops through the results performing a DBCC PAGE for each one to get the primary key values of the rows containing the invalid data.

Within this loop, there is also code to derive the condition for the primary key (allowing for multi-column keys). The key value is then used to query the table to get the current value of the affected column. The results are stored in a table, tmp\_final\_results. This can be dropped once all investigation is complete.

There is further code, commented out, that I will explain shortly.

**How to use the script**

This is a 3 stage process.

**1. First run the script within the context of the database to check.**

It may take several hours to run, especially if you have a few hundred thousand data purity errors.

When it has finished, the table tmp\_final\_results will contain a row for each out-of-range column.

**2. Use the first commented out section of the script to view the results.**

Note: the conversion of [Value] to varchar is necessary, as attempting to retrieve some out-of-range data may result in arithmetic overflow errors. By converting them to varchar, these are displayed as -1.#IND instead.

**3. Fix the data**

Now you have to decide what to set each one to.

If you are lucky you may find that all the values are within a valid range for their datatypes, so a straightforward UPDATE to their existing value will fix the problem. The third section of my script (also commented out) generates an UPDATE statement for each column.

However, you may find out-of-range values, and some may be displayed as -1.#IND. For these you have to decide what they should be set to, and this means speaking with someone who knows the application well and getting them to look at your data.

**4. Tidy up**

The final commented out section needs to be run to drop the tables generated by the script.

Please remember that this script is a tool to aid in the identification of columns that have failed a data purity check. You should not blindly update the columns - ensure you are 100% confident of the correct values. However, the code in section 3 of the script is available if you decide the suggested values can be used. And of course, backup the database before making any data changes.



/\*

Data Purity Investigation

STEP 1

Run this script within the context of the database you want to check (use [dbname];)

Once it has finished, which may take several hours if there are a lot of data purity errors, run the commented out SQL under step 2

to view the results, or use step 3 to generate UPDATE statements (be careful - read the article first).

Finally, run the commented out code under step 4 to tidy up.

\*/

SET NOCOUNT ON;

SET ANSI\_WARNINGS OFF;

DECLARE

@db sysname,

@ver varchar(20),

@sql varchar(max),

@record varchar(255)

SELECT @db = DB\_NAME()

SELECT @ver = CONVERT(varchar(20), SERVERPROPERTY('ProductVersion'))

SELECT @sql = 'DBCC CHECKDB (''' + @db + ''') WITH DATA\_PURITY, NO\_INFOMSGS, TABLERESULTS'

-- Create temporary tables. These will need to be dropped manually once the results of the script have analysed

CREATE TABLE tmp\_dbcc\_results

(

[ObjectId] int NULL,

[Column] sysname NULL,

[File] int NULL,

[Page] int NULL,

[Slot] int NULL

)

CREATE TABLE tmp\_page\_results

(

[ParentObject] sysname NULL,

[Object] sysname NULL,

[Field] sysname NULL,

[Value] sysname NULL

)

CREATE NONCLUSTERED INDEX idx\_slot

ON tmp\_page\_results([Object], [Field])

INCLUDE([Value])

CREATE TABLE tmp\_final\_results

(

[Id] int IDENTITY(1, 1) PRIMARY KEY CLUSTERED,

[Table] sysname NULL,

[Key] varchar(1000) NULL,

[Column] sysname NULL,

[Type] varchar(50) NULL,

[Value] sql\_variant NULL

)

CREATE NONCLUSTERED INDEX idx\_val

ON tmp\_final\_results([Value], [Table])

CREATE TABLE tmp\_primary\_keys

(

[Table] sysname, [Column] sysname, [Type] varchar(50)

)

-- Perform the data purity check. The output is version dependent, hence the condition code

IF @ver LIKE '9%' OR @ver LIKE '10%'

BEGIN

CREATE TABLE tmp\_dbcc\_results\_to\_2008\_r2

(

[Error] int NULL,

[Level] int NULL,

[State] int NULL,

[MessageText] nvarchar(2048) NULL,

[RepairLevel] nvarchar(100) NULL,

[Status] int NULL,

[DbId] int NULL,

[ObjectId] int NULL,

[IndexId] int NULL,

[PartitionId] bigint NULL,

[AllocUnitId] bigint NULL,

[File] int NULL,

[Page] int NULL,

[Slot] int NULL,

[RefFile] int NULL,

[RefPage] int NULL,

[RefSlot] int NULL,

[Allocation] int NULL

)

INSERT tmp\_dbcc\_results\_to\_2008\_r2

EXEC(@sql)

INSERT tmp\_dbcc\_results

SELECT [ObjectId],SUBSTRING(MessageText, CHARINDEX('). Column "', MessageText) + 11, CHARINDEX('"', MessageText, CHARINDEX('). Column "', MessageText) + 11) - CHARINDEX('). Column "', MessageText) - 11),[File],[Page],[Slot]

FROM tmp\_dbcc\_results\_to\_2008\_r2

WHERE [Error] = 2570

DROP TABLE tmp\_dbcc\_results\_to\_2008\_r2

END

ELSE IF @ver LIKE '11%'

BEGIN

CREATE TABLE tmp\_dbcc\_results\_from\_2012

(

[Error] int NULL,

[Level] int NULL,

[State] int NULL,

[MessageText] nvarchar(2048) NULL,

[RepairLevel] nvarchar(100) NULL,

[Status] int NULL,

[DbId] int NULL,

[DbFragId] int NULL,

[ObjectId] int NULL,

[IndexId] int NULL,

[PartitionId] bigint NULL,

[AllocUnitId] bigint NULL,

[RidDbId] int NULL,

[RidPruId] int NULL,

[File] int NULL,

[Page] int NULL,

[Slot] int NULL,

[RefDbId] int NULL,

[RefPruId] int NULL,

[RefFile] int NULL,

[RefPage] int NULL,

[RefSlot] int NULL,

[Allocation] int NULL

)

INSERT tmp\_dbcc\_results\_from\_2012

EXEC(@sql)

INSERT tmp\_dbcc\_results

SELECT [ObjectId],SUBSTRING(MessageText, CHARINDEX('). Column "', MessageText) + 11, CHARINDEX('"', MessageText, CHARINDEX('). Column "', MessageText) + 11) - CHARINDEX('). Column "', MessageText) - 11),[File],[Page],[Slot]

FROM tmp\_dbcc\_results\_from\_2012

WHERE [Error] = 2570

DROP TABLE tmp\_dbcc\_results\_from\_2012

END

-- Run DBCC PAGE for each error.

-- CHECKDB only generates one error per row, even if there are multiple columns with data purity issues.

-- This code checks the DBCC PAGE output to ensure these missing errors are captured.

DECLARE

@ObjectId int,

@Column nvarchar(255),

@File int,

@Page int,

@Slot int

DECLARE c CURSOR FOR

SELECT [ObjectId],[Column],[File],[Page],[Slot]

FROM tmp\_dbcc\_results

OPEN c

FETCH NEXT FROM c INTO @ObjectId, @Column, @File, @Page, @Slot

WHILE @@FETCH\_STATUS = 0

BEGIN

SELECT @sql = 'DBCC PAGE (' + @db + ', ' + CONVERT(varchar(10), @File) + ', ' + CONVERT(varchar(20), @Page) + ', 3) WITH TABLERESULTS'

INSERT tmp\_page\_results

EXEC (@sql)

IF NOT EXISTS(SELECT 1

FROM tmp\_primary\_keys

WHERE [Table] = OBJECT\_NAME(@ObjectId))

BEGIN

INSERT tmp\_primary\_keys

(

[Table],[Column],[Type]

)

SELECT OBJECT\_NAME(@ObjectId),cu.COLUMN\_NAME,t.name

FROM INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE cu

JOIN sys.columns c ON c.object\_id = @Objec

AND c.name = cu.COLUMN\_NAME

JOIN sys.types t ON c.user\_type\_id = t.user\_type\_id

JOIN INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS tc ON tc.TABLE\_NAME = OBJECT\_NAME(@ObjectId)

AND tc.CONSTRAINT\_NAME = cu.CONSTRAINT\_NAME

AND tc.CONSTRAINT\_TYPE = 'PRIMARY KEY'

END

SELECT @record = 'Slot ' + CONVERT(varchar(10), @Slot) + ' %'

INSERT tmp\_final\_results

(

[Table],[Key],[Column],[Type],[Value]

)

SELECT OBJECT\_NAME(@ObjectId),(SELECT LEFT(x.pk, LEN(x.pk) - 4)

FROM (SELECT '[' + pk.[Column] + '] = ' + CASE

WHEN pk.[Type] LIKE '%char%'

OR pk.[Type] LIKE '%date%' THEN ''''

ELSE ''

END + pr.Value + CASE

WHEN pk.[Type] LIKE '%char%'

OR pk.[Type] LIKE '%date%' THEN ''''

ELSE ''

END + ' and ' AS [text()]

FROM tmp\_primary\_keys pk

JOIN tmp\_page\_results pr ON pr.[Object] LIKE @record

AND pr.[Field] = pk.[Column]

WHERE pk.[Table] = OBJECT\_NAME(@ObjectId)

FOR xml path('')) x(pk)),ic.Field,ict.name,NULL

FROM tmp\_page\_results ic

JOIN sys.columns icc ON icc.object\_id = @ObjectId

AND icc.name = ic.[Field]

JOIN sys.types ict ON icc.user\_type\_id = ict.user\_type\_id

WHERE ic.[Object] LIKE @record

AND ( ic.[VALUE] = 'INVALID COLUMN VALUE'

OR ic.[Field] = @Column )

SELECT @sql = (SELECT 'update tmp\_final\_results set [Value] = (select [' + [Column] + '] from [' + [Table] + '] where ' + [Key] + ') where [Id] = ' + CONVERT(varchar(10), [Id]) + ';'

FROM tmp\_final\_results

WHERE [Table] = OBJECT\_NAME(@ObjectId)

AND [Value] IS NULL

FOR xml path(''))

EXEC(@sql)

TRUNCATE TABLE tmp\_page\_results

FETCH next FROM c INTO @ObjectId, @Column, @File, @Page, @Slot

END

CLOSE c

DEALLOCATE c

/\*

-- STEP 2

-- Run this to view the results.

-- The conversion to varchar is necessary in some cases to prevent arithmetic overflow errors

SELECT [Table], [Key], [Column], [Type], CONVERT(varchar(255), [Value]) AS [Value]

FROM tmp\_final\_results

\*/

/\*

-- STEP 3

-- Run this to generate the UPDATE statements that will fix the data purity issues

-- IMPORTANT - READ ACCOMPANYING ARTICLE FIRST - DO NOT RUN THE GENERATED UPDATE STATEMENTS UNLESS 100% SURE THEY ARE CORRECT

SELECT 'UPDATE [' + [Table] + '] SET [' + [Column] + '] = ' + CASE

WHEN [Type] LIKE '%char%'

OR [Type] LIKE '%date%' THEN ''''

ELSE ''

END + CONVERT(varchar(255), [Value]) + CASE

WHEN [Type] LIKE '%char%'

OR [Type] LIKE '%date%' THEN ''''

ELSE ''

END + ' WHERE ' + [Key] + '; '

FROM tmp\_final\_results

\*/

/\*

-- STEP 4

-- Finally run this bit to drop all temporary tables

DROP TABLE tmp\_primary\_keys

DROP TABLE tmp\_final\_results

DROP TABLE tmp\_page\_results

DROP TABLE tmp\_dbcc\_results

\*/