**Persisting DBCC output data**

**FROM:** [**http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/**](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/)

Most of the DBCC commands return their results as textual output, even if you have SSMS configured to return result sets to a grid. This makes examining the output a manual process that is prone to errors. It sure would be nice if there was a way to return the output to a grid.

If we were to examine Books Online (BOL) for DBCC in 2000 (<http://technet.microsoft.com/en-us/library/aa258281%28v=sql.80%29.aspx>) and 2005 (<http://msdn.microsoft.com/en-us/library/ms188796%28v=sql.90%29.aspx>), we would notice a section titled “Using DBCC Result Set Output”, with the phrase: “Many DBCC commands can produce output in tabular form by using the WITH TABLERESULTS option. This information can be loaded into a table for additional use.” This section has been removed from more recent versions, most likely because the results returned are not documented for the individual DBCC commands and are thus subject to change without notice.

Okay, let’s try this out. At the above BOL links, there are two DBCC commands that are documented to use the TABLERESULTS option: OPENTRAN and SHOWCONTIG. Testing all of the other DBCC commands shows that this option can also be used on the CHECKALLOC, CHECKDB, CHECKFILEGROUP and CHECKTABLE commands. I’m going to continue this post with using DBCC CHECKDB. If you check [BOL](http://msdn.microsoft.com/en-us/library/ms176064.aspx), it does not mention the TABLERESULTS option. Let’s first see the results we get without the TABLERESULTS option by running:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_1" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_1) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_1) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DBCC CHECKDB (master);

For the consistency check that is performed on the master database, I end up with over 300 lines of output. The key lines in the output are:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_2" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_2) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_2) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

…

CHECKDB found 0 allocation errors and 0 consistency errors in database 'master'.

…

And the final two lines:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_3" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_3) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_3) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

CHECKDB found 0 allocation errors and 0 consistency errors in database 'mssqlsystemresource'.

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

(Notice that when we run CHECKDB against the master database, the hidden database mssqlsystemresource is also checked.)

If we were to modify the above statement to use the TABLERESULTS option:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_4" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_4) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_4) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DBCC CHECKDB (master) WITH TABLERESULTS;

we would actually get two result sets – one for the master database, and one for the hidden mssqlsystemresource database. Notice that the “DBCC execution completed…” line is not in either of the result sets – it is still displayed on the Messages tab.

Now, most people usually modify this command to suppress informational messages and to show all error messages (it defaults to “only” the first 200 error messages per object), so the command that is normally used would be:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_5" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_5) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_5) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DBCC CHECKDB (master) WITH ALL\_ERRORMSGS, NO\_INFOMSGS;

And, hopefully, this only returns the message “Command completed successfully”. Let’s modify this command to use the TABLERESULTS option:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_6" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_6) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_6) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DBCC CHECKDB (master) WITH ALL\_ERRORMSGS, NO\_INFOMSGS, TABLERESULTS;

If there is no corruption in the database, it still returns only the message “Command completed successfully”. However, if there is corruption, you will get a result set back. So, I’m now going to run this against a database (Lab) where I have engineered some corruption. First off, let’s run CHECKDB without the TABLERESULTS option to check the initial output:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_7" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_7) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_7) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DBCC CHECKDB (Lab) WITH ALL\_ERRORMSGS, NO\_INFOMSGS;

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_8" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_8) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_8) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

Msg 8939, Level 16, State 98, Line 1

Table error: Object ID 885578193, index ID 0, partition ID 72057594039042048, alloc unit ID 72057594043432960 (type In-row data), page (1:328). Test (IS\_OFF (BUF\_IOERR, pBUF->bstat)) failed. Values are 133129 and -4.

Msg 8928, Level 16, State 1, Line 1

Object ID 885578193, index ID 0, partition ID 72057594039042048, alloc unit ID 72057594043432960 (type In-row data): Page (1:328) could not be processed. See other errors for details.

CHECKDB found 0 allocation errors and 2 consistency errors in table 'Tally' (object ID 885578193).

CHECKDB found 0 allocation errors and 2 consistency errors in database 'Lab'.

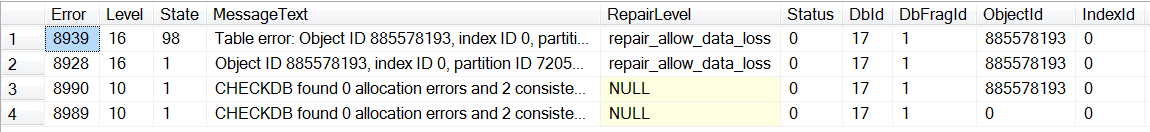
repair\_allow\_data\_loss is the minimum repair level for the errors found by DBCC CHECKDB (Lab).

Here we can see that we indeed have corruption. Quickly now… is any of this corruption in a non-clustered index?

Running this statement with the TABLERESULTS option, we get a grid of the results:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_9" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_9) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_9) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DBCC CHECKDB (Lab) WITH ALL\_ERRORMSGS, NO\_INFOMSGS, TABLERESULTS;

[](http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/uploads/2014/11/DBCCTABLE1.png)

(This screen shot is only showing some of the columns. Note that the message of the previous output for the minimum repair level is still on the Messages tab.)

As you can see, this already makes examining your errors somewhat easier. For instance, you can easily scroll through this output to see if the corruption is in non-clustered indexes (IndexId > 1), where the corruption could easily be fixed by scripting out the index definition, dropping and then re-creating the non-clustered index. Suppose you had over 100 errors… you can see how much faster this would be.

If you recall, the BOL description says that this data can be loaded into a table for further processing. Furthermore, this post is about persisting DBCC output data, which implies storing it into a table. So, let’s make a table, put these results into the table, and then run a query against it. First off, let’s make a local temporary table to hold the results (you could put this into a permanent table if you so desire):

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_10" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_10) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_10) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

IF OBJECT\_ID('tempdb.dbo.#DBCCCHECKDB') IS NOT NULL

DROP TABLE #DBCCCHECKDB;

CREATE TABLE #DBCCCHECKDB (

Error INTEGER,

[Level] INTEGER,

[State] INTEGER,

MessageText VARCHAR(MAX),

RepairLevel VARCHAR(MAX),

[Status] INTEGER,

[DbId] INTEGER,

DbFragId INTEGER,

ObjectId INTEGER,

IndexId INTEGER,

PartitionId BIGINT,

AllocUnitId BIGINT,

RidDbId INTEGER,

RidPruId INTEGER,

[File] INTEGER,

[Page] INTEGER,

Slot INTEGER,

RefDbId INTEGER,

RefPruId INTEGER,

RefFile INTEGER,

RefPage INTEGER,

RefSlot INTEGER,

Allocation INTEGER);

Since the output from using TABLERESULTS isn’t documented, I’ve had to make some assumptions about the data types for these columns. To actually insert the output, we need to use the INSERT INTO … EXECUTE statement:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_11" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_11) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_11) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

INSERT INTO #DBCCCHECKDB

EXECUTE ('DBCC CHECKDB (Lab) WITH ALL\_ERRORMSGS, NO\_INFOMSGS, TABLERESULTS;');

Now that we have the data stored in a temporary table, let’s run a query against the table to return some aggregated data.

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/" \l "codesyntax_12" \o "Click to show/hide code block) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_12) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/#codesyntax_12) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

-- Get the objects, # of errors, and repair level

SELECT [DbId],

database\_name = DB\_NAME([DbId]),

ObjectId,

[OBJECT] = OBJECT\_SCHEMA\_NAME(ObjectId, [DbId])

+ '.' +

OBJECT\_NAME(ObjectId, [DbId]),

ErrorQty = COUNT(\*),

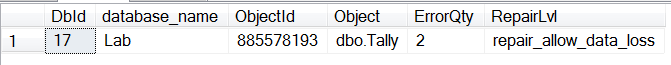
RepairLvl = MAX(RepairLevel)

FROM #DBCCCHECKDB

WHERE Error NOT IN (8989,8990)

GROUP BY [DbId], ObjectId;

With which I get the following results:

[](http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/uploads/2014/11/DBCCTABLE2.png)

So, there we go. [Easy-peasy](http://en.wiktionary.org/wiki/easy_peasy). By utilizing the TABLERESULTS option, the output of the DBCC CHECKDB command has been persisted into a table, and we are now able to run our own queries against that data. In the event that there is corruption in multiple indexes in a table, this query could easily be extended to get the number of errors in each index.

**Related**

[Automating DBCC Page](http://www.sqlsolutionsgroup.com/automating-dbcc-page/)December 23, 2014In "SQL Group"

[Automating DBCC DBTABLE - obtaining the disk Sector Size](http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/)April 28, 2015In "SQL"

**Automating DBCC Page**

**FROM:** [**http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts\_hit=1&relatedposts\_origin=1759&relatedposts\_position=0**](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0)

Way back in 2006, Paul Randal documented DBCC PAGE on his Microsoft blog at <http://blogs.msdn.com/b/sqlserverstorageengine/archive/2006/06/10/625659.aspx>. In his post, you will notice that in order to return the output from DBCC PAGE to the screen, you need to enable trace flag 3604 first. The above blog post shows a few examples of the results and utilizing them for further actions. Unfortunately, this method requires manual intervention to get the necessary data from the page in order to work with it further.

You know, it sure would be nice if the manual intervention could be removed and to completely automate the task that you are looking for. This blog post is going to show you how this can be done, and it will give a few examples of doing so.

In [my last blog](http://bit.ly/1x0awIS) post, I introduced an optional setting to several DBCC commands: WITH TABLERESULTS. This returns the output in tabular format, in a manner that allows the output to be consumed. As it turns out, this optional setting also works with DBCC PAGE. You can see this by examining the database information page (page 9) of the master database:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_1) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_1) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_1) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DBCC PAGE ('master', 1, 9, 3) WITH TABLERESULTS;

This statement, which can be run without the trace flag, returns the data on the page in 4 columns: ParentObject, Object, Field and VALUE. As it turns out, this particular page of every database has a lot of interesting information on it. Let’s automate grabbing a specific piece of information out of this page:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_2) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_2) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_2) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

IF OBJECT\_ID('tempdb.dbo.#DBCCPAGE') IS NOT NULL DROP TABLE #DBCCPAGE;

CREATE TABLE #DBCCPAGE (

ParentObject VARCHAR(255),

[OBJECT] VARCHAR(255),

Field VARCHAR(255),

[VALUE] VARCHAR(255));

INSERT INTO #DBCCPAGE

EXECUTE ('DBCC PAGE (''master'', 1, 9, 3) WITH TABLERESULTS;');

SELECT LastGoodDBCCDate = CONVERT(DATETIME, VALUE)

FROM #DBCCPAGE

WHERE Field = 'dbi\_dbccLastKnownGood';

This wonderful snippet of code returns the last time a successful DBCC CHECKDB was run against the master database. **Whoa… how sweet is that**? When you discover a server where the CHECKDB job has been failing to the point that the job only has failed entries in the job history, you can now find out just how long it’s been since it was last run successfully.

The “trick” to making this work is to encapsulate the DBCC command as a string, and to call it with the EXECUTE () function. This is used as part of an INSERT INTO / EXECUTE statement, so that the results from DBCC PAGE are inserted into a table (in this case a temporary table is used, although a table variable or permanent table can also be used). There are three simple steps to this process:

1. Create a table (permanent / temporary) or table variable to hold the output.
2. Insert into this table the results of the DBCC PAGE statement by using INSERT INTO / EXECUTE.
3. Select the data that you are looking for from the table.

By utilizing a cursor, you can easily spin through all of the databases on your instance to get when they last had a successful DBCC CHECKDB run against them. The following utilizes sp\_MSforeachdb (which itself uses a cursor to spin through all of the databases) to do just this:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_3) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_3) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page/?relatedposts_hit=1&relatedposts_origin=1759&relatedposts_position=0#codesyntax_3) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

IF OBJECT\_ID('tempdb.dbo.#DBCCPAGE') IS NOT NULL DROP TABLE #DBCCPAGE;

IF OBJECT\_ID('tempdb.dbo.#CheckDBDates') IS NOT NULL DROP TABLE #CheckDBDates;

CREATE TABLE #DBCCPAGE (

ParentObject VARCHAR(255),

[OBJECT] VARCHAR(255),

Field VARCHAR(255),

[VALUE] VARCHAR(255));

CREATE TABLE #CheckDBDates (

database\_name sysname,

LastCheckDB DATETIME);

EXECUTE sp\_MSforeachdb '

TRUNCATE TABLE #DBCCPAGE;

INSERT INTO #DBCCPAGE

EXECUTE (''DBCC PAGE (''''?'''', 1, 9, 3) WITH TABLERESULTS;'');

INSERT INTO #CheckDBDates

SELECT ''?'', CONVERT(DATETIME, VALUE)

FROM #DBCCPAGE

WHERE Field = ''dbi\_dbccLastKnownGood'';';

SELECT database\_name,

LastCheckDB

FROM #CheckDBDates;

And here we have every database on your instance, and the date that DBCC CHECKDB was last run successfully against it – well, at least for the databases that sp\_MSforeachdb didn’t skip (see <http://www.mssqltips.com/sqlservertip/2201/making-a-more-reliable-and-flexible-spmsforeachdb>/ for more information about this).

In summary, by utilizing the “WITH TABLERESULTS” option of DBCC, we can automate processes that use DBCC PAGE, instead of needing manual intervention to work your way through this.

In the next few installments, we’ll look at other uses of using “WITH TABLERESULTS” to automate DBCC output.

Previous related posts:

[Persisting DBCC Output](http://bit.ly/1x0awIS)

**Related**

[Automating DBCC DBTABLE - obtaining the disk Sector Size](http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/)April 28, 2015In "SQL"

[Automating DBCC PAGE – Part 3 (Is a Clustered Index physically sorted on disk?)](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/)March 25, 2015In "SQL Group"

[Automating DBCC PAGE, part 2 - determining object names on pages](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/)February 3, 2015In "SQL Group"

By [Wayne Sheffield](http://www.sqlsolutionsgroup.com/author/wsheffield/)|2014-12-23T15:31:20+00:00December 23rd, 2014|[SQL Group](http://www.sqlsolutionsgroup.com/category/sql-group/)|[0 Comments](http://www.sqlsolutionsgroup.com/automating-dbcc-page/#respond)

**Share This Story, Choose Your Platform!**

**Automating DBCC PAGE, part 2 – determining object names on pages**

**FROM:** [**http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/**](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/)

In [my last post](http://bit.ly/1E7UhP4), I demonstrated how using DBCC PAGE can be automated by using the “WITH TABLERESULTS” option. In this post, we will continue with another look at how this can be done.

On a nice wintry day, your city ended up being covered in several feet of snow. During the course of the night, your own house had several power outages. Being concerned about your databases, you shoveled your way into your office, so that you could check on things. (Okay… actually you would just VPN in, but this is my story after all…)

Once you get into your server, you check the jobs and find that your job that runs DBCC CHECKDB has failed. Let’s assume that a power glitch has caused corruption in your database. In order to find out what all is affected, you run DBCC CHECKDB WITH NO\_INFOMSGS, ALL\_ERRORMSGS. But, this happens to be on your 2TB database, and it will take a while for CHECKDB to finish so that you can find the scope of corruption.

You’d really like to know what tables are affected without having to wait. Luckily(?), this corruption was recorded in msdb.dbo.suspect\_pages, and having just recently read Paul Randal’s post at [here](http://www.sqlskills.com/blogs/paul/finding-table-name-page-id/), we know we can use DBCC PAGE to determine this information. And, after having read my last blog post, you know that we can automate DBCC PAGE, so we can use our new friend “WITH TABLERESULTS” to find out what objects have been corrupted.

The suspect\_pages table, documented [here](http://technet.microsoft.com/en-us/library/ms174425%28v=sql.110%29.aspx), has three particular columns of interest: database\_id, file\_id and page\_id. These correspond nicely to the first three parameters needed for DBCC PAGE. To automate this, we need to know what information we need to return off of the page – and from Paul’s post, we know that this is the field “METADATA: ObjectId”. For this code example, let’s assume that this corruption is on page 11 of the master database (just change “master” to the name of your 2TB database).

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_1) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_1) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_1) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

IF OBJECT\_ID('tempdb.dbo.#DBCCPAGE') IS NOT NULL DROP TABLE #DBCCPAGE;

CREATE TABLE #DBCCPAGE (

ParentObject VARCHAR(255),

[Object] VARCHAR(255),

Field VARCHAR(255),

[VALUE] VARCHAR(255));

INSERT INTO #DBCCPAGE

EXECUTE ('DBCC PAGE (''master'', 1, 11, 3) WITH TABLERESULTS;');

SELECT schema\_name = OBJECT\_SCHEMA\_NAME(ca.object\_id),

object\_name = OBJECT\_NAME(ca.object\_id)

FROM #DBCCPAGE

CROSS APPLY (SELECT CONVERT(INTEGER, VALUE)) ca(object\_id)

WHERE Field = 'Metadata: ObjectId';

And there you go… you now know which object it is that has the corruption. Another interesting field that is returned is the IndexId – the Field value is “Metadata: IndexId”. It would be a similar exercise to grab that from this page also, an exercise that I’ll leave to you.

An automated method for getting the object from all suspect pages would entail encapsulating this logic into a cursor to spin through each row in the suspect\_pages table (and I’ll even throw in getting the index\_id also):

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_2) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_2) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_2) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

IF OBJECT\_ID('tempdb.dbo.#DBCCPAGE') IS NOT NULL DROP TABLE #DBCCPAGE;

IF OBJECT\_ID('tempdb.dbo.#SuspectObjects') IS NOT NULL DROP TABLE #SuspectObjects;

CREATE TABLE #DBCCPAGE (

ParentObject VARCHAR(255),

[Object] VARCHAR(255),

Field VARCHAR(255),

[VALUE] VARCHAR(255));

CREATE TABLE #SuspectObjects (

database\_id INTEGER,

file\_id INTEGER,

page\_id INTEGER,

object\_id INTEGER,

index\_id INTEGER);

DECLARE @database\_id INTEGER,

@file\_id INTEGER,

@page\_id INTEGER,

@SQLCMD NVARCHAR(MAX);

DECLARE cCrackSuspectPages CURSOR LOCAL FAST\_FORWARD FOR

SELECT 'EXECUTE (''DBCC PAGE (' +

CONVERT(VARCHAR(15), database\_id) + ', ' +

CONVERT(VARCHAR(15), file\_id) + ', ' +

CONVERT(VARCHAR(15), page\_id) + ') WITH TABLERESULTS;'');',

database\_id, file\_id, page\_id

FROM msdb.dbo.suspect\_pages;

OPEN cCrackSuspectPages;

FETCH NEXT FROM cCrackSuspectPages INTO @SQLCMD, @database\_id, @file\_id, @page\_id;

WHILE @@FETCH\_STATUS = 0

BEGIN

TRUNCATE TABLE #DBCCPAGE;

INSERT INTO #DBCCPAGE EXECUTE (@SQLCMD);

INSERT INTO #SuspectObjects

(database\_id,

file\_id,

page\_id,

object\_id,

index\_id

)

SELECT @database\_id, @file\_id, @page\_id,

(SELECT CONVERT(INTEGER, VALUE)

FROM #DBCCPAGE dp

WHERE dp.Field = 'Metadata: ObjectId'),

(SELECT CONVERT(INTEGER, VALUE)

FROM #DBCCPAGE dp

WHERE dp.Field = 'Metadata: IndexId');

FETCH NEXT FROM cCrackSuspectPages INTO @SQLCMD, @database\_id, @file\_id, @page\_id;

END

CLOSE cCrackSuspectPages;

DEALLOCATE cCrackSuspectPages;

SELECT database\_name = DB\_NAME(database\_id),

database\_id,

file\_id,

page\_id,

schema\_name = OBJECT\_SCHEMA\_NAME(object\_id, database\_id),

object\_name = OBJECT\_NAME(object\_id, database\_id),

index\_id

FROM #SuspectObjects;

If you happen to be on SQL 2012 or higher, this can be greatly simplified by using the new (undocumented) DMO function sys.dm\_db\_database\_page\_allocations (and it also takes away the need to crack the page using DBCC PAGE).

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_3) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_3) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-part-2/#codesyntax_3) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

SELECT database\_name = DB\_NAME(sp.database\_id),

sp.database\_id,

sp.FILE\_ID,

sp.page\_id,

schema\_name = OBJECT\_SCHEMA\_NAME(dpa.OBJECT\_ID, sp.database\_id),

OBJECT\_NAME = OBJECT\_NAME(dpa.OBJECT\_ID, sp.database\_id),

dpa.index\_id

FROM msdb.dbo.suspect\_pages sp

CROSS APPLY sys.dm\_db\_database\_page\_allocations(sp.database\_id, NULL, NULL, NULL, 'LIMITED') dpa

WHERE sp.FILE\_ID = dpa.allocated\_page\_file\_id

AND sp.page\_id = dpa.allocated\_page\_page\_id;

And there we go – yet another time when you might want to automate using DBCC PAGE. By now you should be able to see other uses for it – as long as you can get the database\_id, file\_id and page\_id, you can automate the usage of it to retrieve the information that you are looking for.

Previous related posts:

[Persisting DBCC Output](http://bit.ly/1x0awIS)  
[Automating DBCC Page](http://bit.ly/1E7UhP4)

**Related**

[Automating DBCC Page](http://www.sqlsolutionsgroup.com/automating-dbcc-page/)December 23, 2014In "SQL Group"

[Persisting DBCC output data](http://www.sqlsolutionsgroup.com/persisting-dbcc-output-data/)November 26, 2014In "SQL Group"

[Automating DBCC DBTABLE - obtaining the disk Sector Size](http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/)April 28, 2015In "SQL"

**Automating DBCC PAGE – Part 3 (Is a Clustered Index physically sorted on disk?)**

**FROM:** [**http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts\_hit=1&relatedposts\_origin=1914&relatedposts\_position=1**](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1)

In my last few blog posts, I’ve shared several methods of getting internal information from a database by using the DBCC PAGE command and utilizing the “WITH TABLERESULTS” option to be allowed to automate this process for further processing.

This post will also do this, but in this case, we’ll be using it to bust a common myth—data in a clustered index is physically stored on disk in the order of the clustered index.

To bust this myth, we’ll create a database, put a table with a clustered index into this database, and then we’ll add some rows in random order. Next, we will show that the rows are stored on the pages in logical order, and then we’ll take a deeper look at the page internals to see that the rows are not stored in physical order.

To start off with, let’s create a database and a table, and add a few rows to this table:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_1) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_1) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_1) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

IF DB\_ID('CIPageTest') IS NULL

CREATE DATABASE CIPageTest;

GO

-- use the database

USE CIPageTest;

GO

-- if the PageTest table exists, then drop it to start all over

IF OBJECT\_ID('dbo.PageTest','U') IS NOT NULL DROP TABLE dbo.PageTest;

GO

-- create the dbo.PageTest table

CREATE TABLE dbo.PageTest (

RowID INTEGER PRIMARY KEY CLUSTERED,

Col1 VARCHAR(1000)

);

INSERT INTO dbo.PageTest (RowID, Col1) VALUES (1, REPLICATE('Row01', 100));

INSERT INTO dbo.PageTest (RowID, Col1) VALUES (3, REPLICATE('Row03', 100));

INSERT INTO dbo.PageTest (RowID, Col1) VALUES (5, REPLICATE('Row05', 100));

INSERT INTO dbo.PageTest (RowID, Col1) VALUES (2, REPLICATE('Row02', 100));

INSERT INTO dbo.PageTest (RowID, Col1) VALUES (4, REPLICATE('Row04', 100));

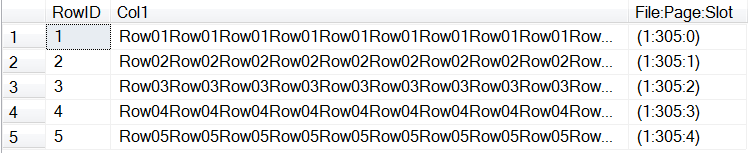
What we have is a table with 5 rows. The table’s clustered key is the RowID integer column. The rows are inserted so that the odd rows are inserted first, followed by the even rows.

At this point, let’s look at where the system reports these rows to be at. To do this, we’ll utilize two undocumented system commands. The first is %%physloc%%, which returns the physical location in a hexadecimal format. The second is fn\_PhysLocFormatter, which converts this into a format of FileID:PageID:SlotID. This is just simply added to the select clause, so the query is:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_2) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_2) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_2) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

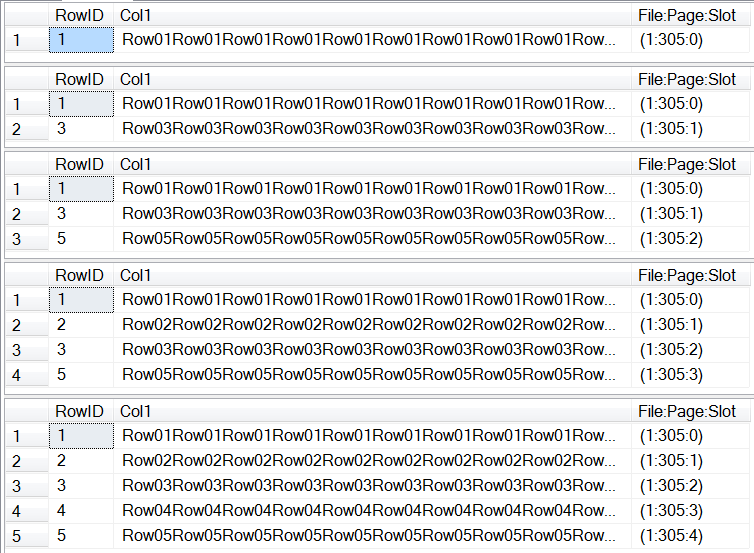
SELECT RowID, Col1, [File:Page:Slot] = sys.fn\_PhysLocFormatter(%%physloc%%) FROM dbo.PageTest;

This query produces these results:

[](http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/uploads/2014/12/DBCCPage3-1.png)

From these results, you can see that the rows are all on the same page, and that the slot in the slot array on that page are incrementing for the appropriate RowID value. Remember also that the slot array is zero-based, where the first slot is slot #0. This is the logical order.

To see this changing around as the rows are inserted, just put the select statement (which I just introduced) after each of the prior insert commands and run the script to create the table and add the rows again. You will initially see RowID=1 put into Slot #0, RowID =3 into Slot #1 and RowID=5 into slot #2. When you then add RowID=2, this needs to be between RowID #s 1 and 3, so #2 is now in slot array #1, #3 moves to slot array #2, and #5 is moved to slot array #3. When you add RowID=4, it gets inserted into slot #3 and RowID#5 is again pushed down, to slot #4. The logical order follows what we are expecting:

[](http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/uploads/2014/12/DBCCPage3-3.png)

In order to show that these are not physically stored in this order, we will need to crack this page and look internally at where the data **is** actually stored. We will accomplish this with this code:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_3) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_3) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_3) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DECLARE @Page INTEGER,

@SQLCMD VARCHAR(MAX);

DECLARE @DBCCIND TABLE (

PageFID INTEGER,

PagePID INTEGER,

IAMFID INTEGER,

IAMPID INTEGER,

ObjectID INTEGER,

IndexID INTEGER,

PartitionNumber INTEGER,

PartitionID BIGINT,

iam\_chain\_type VARCHAR(100),

PageType INTEGER,

IndexLevel INTEGER,

NextPageFID INTEGER,

NextPagePID INTEGER,

PrevPageFID INTEGER,

PrevPagePID INTEGER

);

INSERT INTO @DBCCIND EXECUTE ('DBCC IND (CIPageTest, ''dbo.PageTest'', -1)');

SELECT @Page = PagePID

FROM @DBCCIND

WHERE PageType = 1;

SET @SQLCMD = 'DBCC PAGE (CIPageTest, 1, ' + CONVERT(VARCHAR(15), @Page) + ', 3) WITH TABLERESULTS';

DECLARE @DBCCPAGE TABLE (

RowID INTEGER IDENTITY,

ParentObject VARCHAR(255),

Object VARCHAR(255),

Field VARCHAR(255),

Value VARCHAR(255));

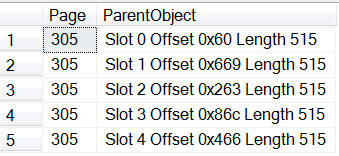
INSERT INTO @DBCCPAGE EXECUTE (@SQLCMD);

SELECT DISTINCT Page = @Page, ParentObject

FROM @DBCCPAGE

WHERE ParentObject LIKE 'Slot%Offset%';

Which produces these results:

[](http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/uploads/2014/12/DBCCPage3-2..png)

As we look at these results, pay close attention to the Offset. This is where the page is physically located on the page. You can see how for RowID #2, that this offset is higher than the offset for RowID #3… and even RowID #5. When the row was added, the data was added to the end of the other existing rows on the page, and the offset where this data starts was entered into the slot array, after having the remaining slot array entries pushed down in order to maintain the logical order. We can see this happening once again when RowID #4 is inserted.

And, finally, let’s use DBCC PAGE to crack open this page and look at the raw data. For this, we want to use dump style 2:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_4) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_4) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/automating-dbcc-page-3/?relatedposts_hit=1&relatedposts_origin=1914&relatedposts_position=1#codesyntax_4) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

DECLARE @Page INTEGER;

SET @Page = xyz; --<< set to the appropriate page from above

DBCC PAGE (CIPageTest, 1, @Page, 2) WITH TABLERESULTS;

If you expand out the VALUE column, you can see it go from Row01 to Row03 to Row05, then to Row02 and Row04.

So there you go… the data is physically stored in the next available space on the disk; **however**, the slot array is in the clustered key order. Furthermore, we now know that we can look at the offset and determine the order on the page. And we also have yet another way to use DBCC PAGE in an automated manner to show this.

Previous related posts:

[Persisting DBCC Output](http://bit.ly/1x0awIS)  
[Automating DBCC Page](http://bit.ly/1E7UhP4)  
[Automating DBCC Page, Part 2](http://bit.ly/1E7VBkY)

**Automating DBCC DBTABLE – obtaining the disk Sector Size**

**FROM:** [**http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/?relatedposts\_hit=1&relatedposts\_origin=1935&relatedposts\_position=1**](http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/?relatedposts_hit=1&relatedposts_origin=1935&relatedposts_position=1)

I was recently reading [this msdn article](http://blogs.msdn.com/b/sqljourney/archive/2012/07/28/an-in-depth-look-at-ghost-records-in-sql-server.aspx) on Ghost Records, and it mentioned that you could get the number of ghost records on a page with DBCC DBTABLE… and it also mentioned that you need to be sure that you enable Trace Flag 3604 in order to see the results. So, two things immediately jumped out at me. First, I wanted to look at this to see where the ghost records were located. Secondly, I’ve just written a few articles ([here](http://bit.ly/1x0awIS), [here](http://bit.ly/1E7UhP4), [here](http://bit.ly/1E7VBkY) and [here](http://bit.ly/1xfm77y)) where I’ve been able to use the “WITH TABLERESULTS” option on the DBCC command to avoid using this trace flag and to provide automation for the process, and I wanted to see if that would work here also.

The good news is that “WITH TABLERESULTS” does indeed work with DBCC DBTABLE. The bad news is that I could not find the ghost record count in the results.

When I was looking for this information, I noted that the results meta-data are identical to the way DBCC PAGE has its output, so this means that the automation processes already developed will work for them. And as I was looking through the results, looking for a ghost record counter, I noticed two interesting fields:

|  |  |
| --- | --- |
| m\_FormattedSectorSize | 4096 |
| m\_ActualSectorSize | 512 |

Hmm, this is showing me the disk Sector Size of each database file. After checking things on a few different systems, it looks like the m\_ActualSectorSize is what the sector size is for the disk that the database file is currently on, and the m\_FormattedSectorSize appears to be the sector size for when the database was created – and it is copied from the model database, so it appears to be what the disk was like when Microsoft created the model database.

Since it’s a best practice to have the disk sector size (also known as the allocation unit size or block size) set to 64kb (see [this white paper](http://msdn.microsoft.com/en-us/library/dd758814%28v=sql.100%29.aspx)), I decided to programmatically get this information. After digging through the Object and ParentObject columns, this script to get the current allocation using size (Sector Size) for each drive was developed:

|  |  |
| --- | --- |
| [Source code](http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/?relatedposts_hit=1&relatedposts_origin=1935&relatedposts_position=1#codesyntax_1) | [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/code.png](http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/?relatedposts_hit=1&relatedposts_origin=1935&relatedposts_position=1#codesyntax_1) [http://i0.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/printer.png](http://www.sqlsolutionsgroup.com/obtaining-the-disk-sector-size/?relatedposts_hit=1&relatedposts_origin=1935&relatedposts_position=1#codesyntax_1) [http://i2.wp.com/www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/themes/default/images/info.gif](http://www.sqlsolutionsgroup.com/wp-content/plugins/wp-synhighlight/About.html) |

USE master;

GO

IF OBJECT\_ID('tempdb.dbo.#DBTABLE') IS NOT NULL DROP TABLE #DBTABLE

CREATE TABLE #DBTABLE (

ParentObject VARCHAR(255),

Object VARCHAR(255),

Field VARCHAR(255),

Value VARCHAR(255));

INSERT INTO #DBTABLE

EXECUTE ('DBCC DBTABLE WITH TABLERESULTS');

WITH cte1 AS

(

-- get the objects for the dbt\_dbid. Distinct to return only one per database

SELECT DISTINCT Object

FROM #DBTABLE

WHERE Field = 'dbt\_dbid'

), cte2 AS

(

-- get the objects related to the dbt\_dbid for the m\_Startup% field

-- SQL 2005/2008/2008R2 - looking for m\_StartupState

-- SQL 2012+ - Looking for m\_StartupPhase

-- So use m\_Startup%

SELECT DISTINCT t1.Object

FROM #DBTABLE t1

JOIN cte1 ON cte1.Object = t1.ParentObject

WHERE t1.Field LIKE 'm\_Startup%'

), cte3 AS

(

-- get the filepath and sector size for each file

SELECT fcb\_filepath = MAX(CASE WHEN Field = 'fcb\_filepath' THEN Value ELSE NULL END),

m\_ActualSectorSize = MAX(CASE WHEN Field = 'm\_ActualSectorSize' THEN Value ELSE NULL END)

FROM #DBTABLE t1

JOIN cte2 ON cte2.Object = t1.ParentObject

WHERE t1.Field IN ('fcb\_filepath', 'm\_ActualSectorSize')

GROUP BY cte2.Object, t1.Object

)

-- and now get the distinct list of drives and their sector sizes

SELECT DISTINCT Drive,

m\_ActualSectorSize,

is\_64kb = CASE WHEN m\_ActualSectorSize % 65535 = 0 THEN 1 ELSE 0 END

FROM cte3

CROSS APPLY (SELECT Drive = UPPER(LEFT(fcb\_filepath, 2))) ca

ORDER BY Drive;

And here we have yet another way for how a process can be automated by using “WITH TABLERESULTS” on a DBCC command. I think that this one is a particularly good one to show the possibilities – to get this information you have to hit multiple parts of the DBCC results, and repeat it for each file in each database. Doing this by using the 3604 trace flag, finding the appropriate piece and then proceeding on to the piece would be very time consuming to do manually.

Finally, a quick note here: there are better ways of getting the disk sector size – since you can get it with WMI calls, you can get it with PowerShell (or even dos), and there are also command line utilities that will also get you this information. This is just a way to do it from within SQL Server. Note also that this only gets the drives that contain database files on this SQL Server instance – if you are looking for other drives, then this won’t work for you.

Take a look at the other fields that are available in DBTABLE – you just might find another item that you’d like to be able to retrieve.

Previous related posts:

[Persisting DBCC Output](http://bit.ly/1x0awIS)  
[Automating DBCC Page](http://bit.ly/1E7UhP4)  
[Automating DBCC Page, Part 2](http://bit.ly/1E7VBkY)  
[Automating DBCC Page, Part 3](http://bit.ly/1xfm77y)