MINION CHECKDB: QUICK START

Minion CheckDB by MinionWare is a free stand-alone integrity check solution that can be deployed on any number of servers. Minion CheckDB is comprised of SQL Server tables, stored procedures, and SQL Agent jobs. For links to downloads, tutorials, and articles, see www.MinionWare.net.

This document explains Minion CheckDB by MinionWare ("Minion CheckDB"), its uses, features, moving parts, and examples.

For video tutorials on Minion CheckDB, see the Minion CheckDB playlist on our YouTube channel: https://www.youtube.com/MidnightDBA

Minion CheckDB is one module of the Minion suite of products.
There are three easter eggs in this documentation; find all three and email us at MinionWareSales@MidnightDBA.com for three free licenses of Minion Enterprise!
(First time winners only, please.)

Quick Start

System requirements:

- SQL Server 2008 or above.
- The sp_configure setting **xp_cmdshell** must be enabled*.
- PowerShell 3.0 or above; execution policy set to **RemoteSigned**.

Once the installer has been run, nothing else is required. From here on, Minion CheckDB will run regularly for **all** non-TempDB databases. The CheckDB routine automatically handles databases as they are created, dropped, or renamed.

** xp_cmdshell can be turned on and off with the database PreCode / PostCode options, to help comply with security policies.*

For more information on xp_cmdshell, see "Security Theater" on www.MidnightDBA.com/DBARant.

This entire document is also available within the installed Minion CheckDB database using the SQL stored procedure Minion.HELP.

Read the *Minion Install Guide.docx* (contained within the MinionCheckDB1.0.zip file) for full instructions and information on using the installer. The basic steps to installing Minion CheckDB are:

1. Download **MinionCheckDB1.0.zip** from MinionWare.net and extract all files to the location of your choice, replacing any existing MinionWare installer folders from previous downloads.
2. Open Powershell as an administrator, and use **Get-ExecutionPolicy** to verify the current execution policy is set to Unrestricted or RemoteSigned. If it is not, use **Set-ExecutionPolicy RemoteSigned to allow the installer to run.**
3. Right-click on each of the following files, select Properties, and then "**Unblock**" the file if necessary. (This allows you to run scripts downloaded from the web using the RemoteSigned execution policy.)
   a. …\MinionWare\MinionSetupMaster.ps1
   b. …\MinionWare\MinionSetup.ps1
   c. …\MinionWare\Includes\CheckDBInclude.ps1
4. Run MinionSetupMaster.ps1 in the PowerShell administrator window as follows:
   .\MinionSetupMaster.ps1 <servername> <DBName> <Product>

   Examples:
   .\MinionSetupMaster.ps1 localhost master CheckDB
   or
   .\MinionSetupMaster.ps1 YourServer master CheckDB

Note that you can install multiple products, and to multiple servers. For more information, see the *Minion Install Guide.docx*.

For simplicity, this Quick Start guide assumes that you have installed Minion CheckDB on one server, named "YourServer".

Customizing Schedules

Minion CheckDB offers a choice of scheduling options. This quick start section covers the default method of scheduling: table based scheduling. We will cover parameter based schedules, and hybrid schedules, in the section titled "How To: Change Schedules". For more information, see "About: Scheduling".

Table based scheduling

In conjunction with the "MinionCheckDB-AUTO" job, the Minion.CheckDBSettingsServer table allows you to configure flexible CheckDB scheduling scenarios. By default, Minion CheckDB is installed with the following configuration:

The *MinionCheckDB-AUTO* job runs hourly, checking the Minion.CheckDBSettingsServer table to determine what operation should be run.

In the Minion.CheckDBSettingsServer table:

- System database CheckDB operations are scheduled **daily at 10:00pm**.
- User database CheckDB operations are scheduled for **Saturdays at 11:00pm**.

The following table displays the first few columns of this default scenario in Minion.CheckDBSettingsServer:

ID	DBType	OpName	Day	ReadOnly	BeginTime	EndTime	MaxForTimeframe
1	System	CHECKDB	Daily	1	22:00:00	22:30:00	1
2	User	CHECKDB	Saturday	1	23:00:00	23:30:00	1

Note: There is also an inactive row for User databases to run AUTO operations Saturday at 11:00 pm. For information about OpName = AUTO, see "About: Dynamic Thresholds" and "How to: Configure Dynamic Thresholds". For an example of a complex scenario that includes OpName=AUTO, see "About: Minion CheckDB Operations".

Let's walk through two different schedule change scenarios:

Scenario 1: Run CheckDB on user databases daily. To change the default setup to run daily CheckDBs on all user databases, update the row with DBType='User' & OpName='CHECKDB', setting the Day field to "Daily".

Scenario 2: Run CheckTable twice daily for specific schemas. To change the default setup in order to run CheckTable twice daily on two specific schemas (in this example, Import and Ace), insert a new row to Minion.CheckDBSettingsServer for CheckDBType='CheckTable' and Schemas='Import,Ace':

```
INSERT  INTO Minion.CheckDBSettingsServer
        ( DBType
        , OpName
        , Day
        , ReadOnly
        , BeginTime
        , EndTime
        , MaxForTimeframe
        , FrequencyMins
        , Schemas
        , Debug
        , FailJobOnError
        , FailJobOnWarning
        , IsActive
        , Comment
        )
VALUES  ( 'User'            -- DBType
        , 'CHECKTABLE'      -- OpName
        , 'Daily'           -- Day
        , 1                 -- ReadOnly
        , '04:00:00'         -- BeginTime
        , '18:00:00'         -- EndTime
        , 2                 -- MaxForTimeframe
        , 720               -- FrequencyMins
        , 'Import,Ace'      -- Schemas
        , 0                 -- Debug
        , 0                 -- FailJobOnError
        , 0                 -- FailJobOnWarning
        , 1                 -- IsActive
        , 'Twice daily CHECKTABLE operations'  -- Comment
        );
```

In the scenario above there are a few critical concepts to understand:

- **Execution Window:** The BeginTime and EndTime settings will restrict this CheckTable entry to between 4:00am and 6:00pm. Minion CheckDB will ignore this entry outside of that execution window.
- **Frequency:** FrequencyMins=720 means that this schedule (row) will only run once in any 720 minute (12 hour) period, regardless of how many times Minion CheckDB is schedule to run.
- **Always set the MaxForTimeframe field.** This setting determines the maximum number of times an operation may be executed in the defined timeframe. In the insert statement above, MaxForTimeframe is set to 2, because we only want to allow a maximum of 2 CheckTable operations during the daily window (between 4am and 6pm).
- **The Schemas setting applies to all databases:** What's more, Schemas='Import,Ace'. This means that the run will only apply to tables within the "Import" and "Ace" schemas *in any database on the system.* (The Schemas and Tables fields apply to all databases.)

Default Settings

Minion CheckDB stores default settings for the entire instance in two rows (where DBName='MinionDefault') in the Minion.CheckDBSettingsDB table.

Warning: **Do not delete the MinionDefault rows, or rename the DBName for the MinionDefault row, in Minion.CheckDBSettingsDB!**

To change the default settings, run an update statement on the MinionDefault / CHECKDB row (or the MinionDefault / CHECKTABLE row) in Minion.CheckDBSettingsDB. For example:

```
UPDATE  Minion.CheckDBSettingsDB
SET     NoInfoMsgs = 1
```

```
    , HistRetDays = 75
    , ResultMode = 'Summary'
WHERE   DBName = 'MinionDefault'
        AND OpName = 'CHECKDB';
```

MINION CHECKDB

Contents in Brief

Top Features

Minion CheckDB is a stand-alone database integrity check module. Once installed, Minion CheckDB automatically checks all online databases on the SQL Server instance, and will incorporate databases as they are added or removed.

Some of the very best features of Minion CheckDB are, in a nutshell:

1. **Dynamic Thresholds** – Minion CheckDB allows you to automate whether databases get a DBCC CheckDB operation, or a DBCC CheckTable operation.
2. **Remote CheckDB** – Automatically run DBCC CheckDB remotely for any database.
3. **Dynamic Remote CheckDB** – Allows you to set a tuning threshold, so the CheckDB will run remotely only if it is above that threshold.
4. **Custom Snapshots** – Choose to create a custom snapshot, for versions of SQL Server that support custom snapshots. This allow you to determine where your snapshot file(s) will be located.
5. **Custom Dynamic Snapshots** – For CheckTable operations, you can configure “rotating” dynamic snapshots that drop and recreate every few minutes.
6. **Multithreaded database processing –** Run multiple DBCC CheckDB operations in parallel.

7. **Multithreaded table processing –** Run multiple DBCC CheckTable processes at the same time.
8. **Rotational scheduling** – Minion CheckDB allows you to define a rotation scenario for your operations. For example, a nightly round of 10 databases would perform integrity checks on 10 databases the first night, another 10 databases the second night, and so on. You can also use the rotational scheduling to limit operations by time; for example, you could configure MC to cycle through DBCC CheckDB operations for 90 minutes each night.
9. **Operation ordering** – Run DBCC CheckDB and CheckTable operations in exactly the order you need.
10. **Extensive, useful logging** – Use the Minion CheckDB log for estimating the end of the current CheckDB run, troubleshooting, planning, and reporting. Errors are reported in the log table instead of text files.
11. **Run code before or after CheckDBs and CheckTables –** This is an extraordinarily flexible feature that allows for nearly infinite configurability.
12. **Integrated help –** Get help on any Minion CheckDB object without leaving Management Studio, with the Minion.HELP stored procedure.
13. **Clone Settings –** Use the new CloneSettings procedure to generate template insert statements for any table, based on an example row in the table.
14. **Scenario testing —** Test the settings to be used at any given time for any database.
15. **Automated installation** – Run the Minion CheckDB installation scripts, and it just goes. You can even rollout to hundreds of servers almost as easily as you can to a single server.
16. **Granular configuration without extra jobs –** Configure extensive settings at the default, database, and/or table levels with ease. Say good-bye to managing multiple jobs for specialized scenarios. Most of the time you'll run MC with a single job.
17. **Live Insight** – See what Minion CheckDB is doing every step of the way. You can even see the percent complete for each operation as it runs.
18. **Flexible include and exclude –** Perform integrity checks on only what you need, using specific database names, LIKE expressions, and even regular expressions. Further restrict operations by including or excluding by schemas and/or tables.
19. **Inline Tokens** – Inline Tokens allow you use defined patterns to create dynamic names. For example, MC comes with the predefined Inline Token "Server" and "DBName". For more information, see the "About: Inline Tokens" section.

For links to downloads, tutorials and articles, see www.MinionWare.nethttp://www.MidnightSQL.com/Minion.http://www.MidnightSQL.com/Minion

Architecture Overview

Minion CheckDB is made up of SQL Server stored procedures, functions, tables, and jobs. The tables store configuration and log data; stored procedures perform CheckDB operations; and the jobs execute and monitor those operations on a schedule.

This section provides a brief overview of Minion CheckDB elements at a high level.

Note: Minion CheckDB is installed in the master database by default. You certainly can install Minion in another database (like a DBAdmin database), but when you do, you must also verify that the job steps point to the appropriate database.

Configuration Settings Hierarchy

Configuration settings for integrity check operations are stored in tables: Minion.CheckDBSettingsDB and Minion. CheckDBSettingsTable. A default row in Minion.CheckDBSettingsDB (DBName='MinionDefault') provides settings for any database that doesn't have its own specific settings. This is a hierarchy of granularity, where more specific configuration levels completely override the less specific levels. That is:

- Insert a row for a specific database (for example, DBName='DB1') into Minion.CheckDBSettingsDB, and that row will override ALL of the default settings for that database.
- Insert a row for a specific table in Minion.CheckDBSettingsTable, and that row will override ALL of the default (or, if available, database-specific) settings for that particular table.

In other words, a database-specific row completely overrides the MinionDefault rows, for that particular database. And a table-specific row overrides the MinionDefault settings for that particular table.

Note: A value left at *NULL* in one of these tables means that Minion will use the setting that the SQL Server instance itself uses.

Additionally, you can configure settings to apply only on specific days, or during certain hours of the day. (For more information, see the "Discussion: Hierarchy and Precedence" section in "About: Scheduling".)

IMPORTANT: Each level of settings in Minion.CheckDBSettingsDB (that is, the MinionDefault level, and each specified database level) should have one row for CHECKTABLE and one row for CHECKDB.

Example: Proper Configuration

Let us take a simple example, in which these are the contents of the Minion.CheckDBSettingsDB table (not all columns are shown here):

ID	DBName	OpLevel	OpName	Exclude	NoInfoMsgs
1	MinionDefault	DB	CHECKDB	0	0
2	MinionDefault	DB	CHECKTABLE	0	1
3	DB1	DB	CHECKDB	1	0
4	DB1	DB	CHECKTABLE	1	0

There are 30 databases on this server. As Minion CheckDB runs, the settings for individual databases will be selected as follows:

- CheckDB operations of database **DB1** will use only the settings from the row with ID=3 or ID=4. (Since Exclude = 1, that means DB1 will not get integrity checks).
- **All other databases** will use the settings from the row with ID=1 (for CheckDB) or ID=2 (for CheckTable).

Database Include and Exclude Precedence

Minion CheckDB allows you to specify lists of databases to include in a CheckDB/CheckTable routine, in a couple of different ways.

Include and Exclude strings

One way to identify which databases should have their integrity checked, is with the Minion.CheckDBSettingsServer Include and Exclude fields; or, for manual executions, the @Include and @Exclude parameters in the Minion.CheckDBMaster stored procedure.

Note: For the purposes of this discussion, we will refer to the @Include/@Exclude parameters, but be aware that the same principles apply to the Include/Exclude fields.

@Include and @Exclude may each have one of three kinds of values:

- 'All' or *NULL* (which also means 'All')
- 'Regex'
- An explicit, comma-delimited list of database names and LIKE expressions (e.g., @Include='DB1,DB2%').

Note: For this initial discussion, we are ignoring the existence of the Exclude *bit*, while we introduce the Include and Exclude *parameters*. We'll explain the Exclude bit concept in at the end of the section.

The following table outlines the interaction of Include and Exclude:

	@Exclude='All' or IS NULL	**@Exclude=[Specific list]**
@Include='All' or IS NULL	Run all CheckDBs	Run all, minus databases in the explicit @Exclude list
@Include=[Specific list]	Run only for databases specified in the @Include list.	Run only specific includes, minus explicit exclude. (But, why would you do this?)

Note that regular expressions phrases are defined in a special settings table (Minion.DBMaintRegexLookup).

Let us look at a couple of scenarios, using this table:

- **@Include IS NULL, @Exclude IS NULL** – Run all CheckDBs.
- **@Include = 'All', @Exclude = 'DB%'** – Run all CheckDBs except those beginning with "DB".

Exclude bit

In addition to the @Include and @Exclude parameters, Minion CheckDB also provides an "Exclude" bit in the primary settings table (Minion.CheckDBSettingsDB), which that allows you to exclude all operations for a specific database.

For example, if you wished to exclude all integrity check operations for database DB1, insert two rows (one for CheckDB and one for CheckTable) to the Minion.CheckDBSettingsDB table with Exclude = 1. From then on, DB1 will not be included in any scheduled operation.

The following table outlines the interaction of the @Include parameter and the Exclude *bit*:

	Exclude=0	**Exclude=1**
@Include='All' or IS NULL	Run all operations as specified (CheckDB or CheckTable)	Run all operations, minus excluded databases' CheckDB types
@Include=[Specific list]	Run only specific includes	Run only specific includes

IMPORTANT: The Exclude bit, like the @Exclude parameter, only applies for instances where @Include (or the column, "Include") is NULL. **Whether @Include is Regex or is a specific list, an explicit @Include should be the final word.** This is because we never want a scenario where a database simply *cannot* have CheckDB performed.

Table Include and Exclude Precedence

Minion CheckDB allows you to specify lists of tables to include in a DBCC CheckTable routine.

Include Strings

One way to identify which tables should have their integrity checked, is with the Minion.CheckDBSettingsServer Schemas and Tables fields; or, for manual runs, the ExcMinion.CheckDBMaster @Schemas and @Tables parameters.

Note: For the purposes of this discussion, we will refer to the @Schemas/@Tables parameters, but be aware that the same principles apply to the Schemas/Tables fields.

@Schemas and @Tables may each have one of two kinds of values:

- *NULL* (which means 'All')
- An explicit, comma-delimited list of database names and LIKE expressions (e.g., @Schemas='Sch1,Sch2%').

Note: For this initial discussion, we are ignoring the existence of the Exclude *bit* in Minion.CheckDBSettingsTable, while we introduce the Schemas and Tables *parameters*. We'll fold the Exclude bit concept back in at the end of the section.

The following table outlines the interaction of Schemas and Tables:

	@Tables IS NULL	**@Tables=[Specific list]**
@Schemas IS NULL	Run all CheckTables	Run only specific tables
@Schemas=[Specific list]	Run only specific schemas	Run all tables in schemas, *plus* specific tables

Note that @Schemas and @Tables *do not limit each other*.

Let us look at a couple of scenarios, using this table:

- **@Schemas IS NULL, @Tables IS NULL** – Run all CheckTabless.
- **@Schemas = 'MySchema', @Tables = 'DB%'** – Run all tables in "MySchema", **PLUS** all tables beginning with DB. Note that the DB% tables will automatically receive the default schema defined in Minion.CheckDBSettingsDB, because there is no schema provided within the @Tables parameter.

Exclude Bit

Minion CheckDB provides an "Exclude" bit in the Minion.CheckDBSettingsTable table, which allows you to exclude CheckTables for a particular table.

The following table outlines the interaction of the @Schemas / @Tables parameters, and the Exclude *bit*:

	Exclude=0	**Exclude=1**
@Schemas IS NULL	Run all CheckTables.	Run all CheckTables except those excluded.
@Schemas=[Specific list]	Run CheckTables only for tables in the listed schemas.	Run CheckTables for tables in the listed schemas, except those excluded.
@Tables IS NULL	Run all CheckDBs.	Run all CheckTables except those excluded.
@Tables=[Specific list]	Run Checktables only for the listed tables.	Run CheckTables only for the listed tables; ignores the Exclude bit in the settings table.

Let us look at a handful of scenarios, using this table:

- **@Schemas='Minion', @Exclude = 1 for Minion.T1** – Run all CheckTables except Minion.T1
- **@Tables IS NULL, Exclude bit=0** – Run all CheckTables.
- **@Tables= 'dbo.T1', Exclude = 1 for DB2** (Minion.CheckDBSettingsDB) – Run CheckTable for dbo.T1.

IMPORTANT: You will note that the Exclude bit is ignored in any case where Tables is not NULL. **An explicit @Tables should be the final word.** The reason for this rule is that we never want a scenario where a table simply *cannot* have CheckTable performed.

Run Time Configuration

The main Minion CheckDB stored procedure – Minion.CheckDBMaster – can be run in one of two ways: with table configuration, or with parameters.

Run Minion.CheckDBMaster using table configuration: If you run Minion.CheckDBMaster without parameters, the procedure uses the Minion.CheckDBSettingsServer table to determine its runtime parameters (including the schedule of DBCC CheckDB and DBCC CheckTable jobs, and which databases to Include and Exclude). This is how MC operates by default, to allow for the most flexible integrity check scheduling with as few jobs as possible.

For more information, see the sections "How To: Change Schedules", "Minion.CheckDBSettingsServer", and "Minion.CheckDBMaster".

Run Minion.CheckDBMaster with parameters: The procedure takes a number of parameters that are specific to the current maintenance run. For example:

- Use @DBType to specify 'System' or 'User' databases.
- Use @OpName to specify CHECKDB, CHECKTABLE, or AUTO.
- Use @StmtOnly to generate integrity check statements, instead of running them.
- Use @Include to specify a specific list of databases, or databases that match a LIKE expression. Alternately, set @Include='All' or @Include=*NULL* to include all databases.
- Use @Exclude to exclude a specific list of databases from CheckDB.
- Use @ReadOnly:
    1. to include ReadOnly databases,
    2. to exclude ReadOnly databases, or
    3. to only include ReadOnly databases.

For more information, see the section "How To: Change Schedules" and "Minion.CheckDBMaster".

Moving Parts

Overview of Tables

The tables in Minion CheckDB fall into four categories: those that store configured **settings**, those that **log** operational information, **debug tables**, and **work tables**.

The **settings** tables are:

- **Minion.CheckDBSettingsAutoThresholds** – This table allows you to set thresholds to automate whether databases get a CheckDB operation, or a CheckTable operation.
- **Minion.CheckDBSettingsDB** – This table contains the essential CheckDB and CheckTable settings for databases, including processing order, history retention, database pre-and postcode, native settings, and more. It holds settings at the default level, database level, and operation level. You may insert rows to define CheckDB/CheckTable settings per database (etc); or, you can rely on the system-wide default settings (defined in the "MinionDefault" rows); or a combination of these.
- **Minion.CheckDBSettingsRemoteThresholds** – This table allows you to define thresholds to prevent smaller databases from taking part in remote DBCC CheckDB operations.
- **Minion.CheckDBSettingsRotation** – This table holds the rotation scenario for your operations (e.g., "run CheckDB on 10 databases every night; the next night, process the next 10; and so on").
- **Minion.CheckDBSettingsServer** – This table contains server-level CheckDB settings, including schedule information. The primary Minion CheckDB job "MinionCheckDB-AUTO" runs regularly in conjunction with this table to provide a wide range of CheckDB options, all without introducing additional SQL Agent jobs.
- **Minion.CheckDBSettingsSnapshot** – This table holds the settings for database snapshots.
- **Minion.CheckDBSettingsTable** – Minion.CheckDBSettingsTable allows you to configure table-level exceptions to the CHECKTABLE settings defined in Minion.CheckDBSettingsDB.
- **Minion.CheckDBSnapshotPath** – This table allows you to configure snapshot file path settings for local custom snapshots. You can specify one row per snapshot file.

The **log** tables are:

- **Minion.CheckDBLog** – Holds an operation-level summary of integrity check operations. It contains one time-stamped row for each execution of Minion.CheckDBMaster, which may encompass several database level integrity check operations. This is updated as each CheckDB occurs, so that you have **Live Insight** into active operations.
- **Minion.CheckDBLogDetails** – Holds a log of CheckDB activity at the database level. This table is updated as each operation occurs, so that you have **Live Insight** into active operations.
- **Minion.CheckDBResult** – Keeps the results from DBCC CheckDB operations (as opposed to outcome and associated operational data in the "Log" tables).
- **Minion.CheckDBSnapshotLog** – This table keeps a record of snapshot files (one row per file). This includes files created as part of local custom snapshots, and as part of snapshot files created locally from a remote server's "remote CheckDB" process.

- **Minion.CheckD BCheckTableResult** – This keeps the results from DBCC CheckTable operations (as opposed to outcome and associated operational data in the “Log” tables).

The **debug** tables are:

- **Minion.CheckDBDebug** – This table holds high level debugging data from Minion CheckDB runs where debugging was enabled.
- **Minion.CheckDBDebugLogDetails** – This table holds detailed debugging data from Minion CheckDB runs where debugging was enabled.
- **Minion.CheckDBDebugSnapshotCreate** – This table holds custom snapshot-related debugging data from Minion CheckDB runs where debugging was enabled.
- **Minion.CheckDBDebugSnapshotThreads** – This table holds thread-related debugging data from Minion CheckDB runs where debugging was enabled.

The **work** tables – which are for internal use, and so are not fully documented – are:

- **Minion.CheckDBCheckTableThreadQueue –** Information gathered in preparation for a CheckTable run is stored here.
- **Minion.CheckDBRotationDBs** – Internal use only.
- **Minion.CheckDBRotationDBsReload** – Internal use only.
- **Minion.CheckDBRotationTables** – Internal use only.
- **Minion.CheckDBRotationTablesReload** – Internal use only.
- **Minion.CheckDBTableSnapshotQueue** – Internal use only.
- **Minion.CheckDBThreadQueue** – Internal use only.

Settings Table Detail

Minion.CheckDBSettingsAutoThresholds

This table allows you to automate whether databases get a DBCC CheckDB operation, or a DBCC CheckTable operation. These settings only apply to runs of the stored procedure Minion.CheckDBMaster where OpName = ‘Auto’ in Minion.CheckDBSettingsDB (or, for a manual run, where @OpName = ‘Auto’).

The default entry that comes installed with Minion CheckDB sets a threshold by size, at 100 GB. What this means is that by default – when Minion.CheckDBMaster runs with @OpName = ‘Auto’, **any database under 100 GB gets a CheckDB operation instead of a CheckTable operation**.

Note: As outlined in the “Configuration Settings Hierarchy” section, more specific settings in a table take precedence over less specific settings. So if you insert a database-specific row for DB1 to this table, that row will be used for DB1 (instead of the “MinionDefault” row in this table).

Name	Type	Description
ID	int	Primary key row identifier.
DBName	varchar	Database name.
ThresholdMethod	varchar	The method by which to measure.

		Valid values: **SIZE**
ThresholdType	varchar	The threshold type, as it relates to ThresholdMethod. ***NULL*** (this is the same as Data) **Data** **DataAndIndex** **File**
ThresholdMeasure	Varchar	The measure for our threshold value. Valid inputs: **GB**
ThresholdValue	Int	The correlating value to ThresholdMeasure. If ThresholdMeasure is GB, then ThresholdValue is the value – the number of gigabytes.
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

Example:

```
-- Insert a row for DB1, threshold 50GB
INSERT  INTO Minion.CheckDBSettingsAutoThresholds
        ( [DBName]
        , [ThresholdMethod]
        , [ThresholdType]
        , [ThresholdMeasure]
        , [ThresholdValue]
        , [IsActive]
        , [Comment]
        )
SELECT  'DB1' AS [DBName]
        , 'Size' AS [ThresholdMethod]
        , 'DataAndIndex' AS [ThresholdType]
        , 'GB' AS [ThresholdMeasure]
        , 50 AS [ThresholdValue]
        , 1 AS [IsActive]
        , 'DB1' AS [Comment];
```

Minion.CheckDBSettingsDB

Minion.CheckDBSettingsDB contains the essential CheckDB settings for databases, including process order, history retention, pre-and postcode, native settings, and more.

Minion.CheckDBSettingsDB is installed with default settings already in place, via the system-wide default rows (identified by DBName = "MinionDefault"). If you do not need to fine tune your integrity checks at all, no action is required, and all operations will use these default configurations.

IMPORTANT: Do not delete the MinionDefault rows!

For more information on DBCC CheckDB options, see the MSDN article on DBCC CHECKDB (https://msdn.microsoft.com/en-us/library/ms176064.aspx).

Name	**Type**	**Description**
ID	Int	Primary key row identifier.
DBName	nvarchar	Database name.
Port	int	Port number for the instance. If this is NULL, we assume the port number is 1433. Minion CheckDB includes the port number because certain operations that are shelled out to sqlcmd require it.
OpLevel	varchar	The level of object that the operation applies to. **Note:** This is not currently in use, but we recommend setting all OpLevel values to 'DB' for future functionality. Valid values: **DB**
OpName	Varchar	The name of the operation (usually, as passed into Minion.CheckDBMaster). **Note:** Each level of settings (that is, the default level, and each database level) should have one row for CHECKTABLE and one row for CHECKDB. For more information, see "Configuration Settings Hierarchy". Note that AUTO is a valid value for the Minion.CheckDBMaster @OpName parameter, but it is NOT valid as a setting in this table (which defines settings for specific operations). Valid values: **CHECKTABLE** **CHECKDB** **CHECKALLOC**
Exclude	Bit	Exclude database from operations.

		For more on this topic, see "How To: Exclude databases from operations" and "Include and Exclude Precedence".
GroupOrder	int	The operation order within a group. Used solely for determining the order in which databases should be processed. By default, all databases and tables have a value of 0, which means they'll be processed in the order they're queried from sysobjects. Higher numbers have a greater "weight" (they have a higher priority), and will be processed earlier than lower numbers. We recommend leaving some space between assigned order numbers (e.g., 10, 20, 30) so there is room to move or insert rows in the ordering. For more information, see "How To: Process databases in a specific order".
GroupDBOrder	int	Group to which this database belongs. Used solely for determining the order in which databases should be processed. By default, all databases have a value of 0, which means they'll be processed in the order they're queried from sysobjects. Higher numbers have a greater "weight" (they have a higher priority), and will be processed earlier than lower numbers. The range of GroupDBOrder weight numbers is 0-255. For more information, see "How To: Process databases in a specific order".
NoIndex	bit	Enable NOINDEX. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
RepairOption	varchar	The repair option to use. **This field is not yet in use.** Future valid values may include: *NULL* NONE REPAIR_ALLOW_DATA_LOSS REPAIR_FAST

		REPAIR_REBUILD For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
RepairOptionAgree	bit	Signifies that you agree to the repair option specified in the RepairOption column. This is in place because some repair options (i.e., “REPAIR_ALLOW_DATA_LOSS”) can cause you to lose data. **This field is not yet in use.** For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
WithRollback	varchar	**This field is not yet in use.**
AllErrorMsgs	bit	Enables or disables the ALL_ERRORMESSAGES option, which displays all reported errors per object. This is on by default. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
ExtendedLogicalChecks	bit	Enables or disables the EXTENDED_LOGICAL_CHECKS option, which performs logical consistency checks where appropriate. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
NoInfoMsgs	bit	Enables or disables the NO_INFOMSGS option, which supresses informational messages. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
IsTabLock	Bit	DBCC CheckDB option -tablock. Causes DBCC CHECKDB to obtain locks instead of using an internal database snapshot. **IMPORTANT: We do not recommend using tablock on production systems!**

IntegrityCheckLevel	varchar	DBCC CheckDB option. This controls whether or not you include physical only, data purity, or neither. Valid values: *NULL* PHYSICAL_ONLY DATA_PURITY
DisableDOP	bit	Enable or disable the trace flag that allows CheckDB to run with a degree of parallelism. Allows you to use or not use multithreading. **Note:** DisableDOP = 1 will *disable* DBCC CheckDB internal multithreading. However, Minion CheckDB multithreading is not affected by this setting. For more information, see "About: Multithreading operations".
IsRemote	bit	Enable or disable remote integrity checks. **Note:** Remote operations only apply to DBCC CheckDB. MC does not support remote CheckTable. **IMPORTANT:** IsRemote = 1 turns on remote CheckDB for *all* databases (that the given row applies to). If you wish to handle remote operations dynamically, based on database size, set IsRemote = 0 and configure remote thresholds. Performing remote integrity checks requires additional setup. See "Minion.CheckDBSettingsRemoteThresholds" and "How to: Set up CheckDB on a Remote Server".
PreferredServer	varchar	The server on which you would like to perform remote CheckDB operations. **Note:** This field does not accept Inline Tokens. Valid inputs: *NULL* <specific server or server\instance name> For more information, see "How to: Set up CheckDB on a Remote Server".
PreferredServerPort	int	The port of the server on which you would like to perform remote CheckDB operations.

		If this value is NULL, the port is assumed to be 1433. Valid values: NULL <specific port>
PreferredDBName	varchar	The database you want to run remote checks against on the remote server. This field is ignored if you're running operations locally. **Note:** Remote operations only apply to DBCC CheckDB. MC does not support remote CheckTable. This field accepts Inline Tokens *and* LIKE expressions. Valid values: NULL <specific database name> For more information, see "About: Remote CheckDB" and "How to: Set up CheckDB on a Remote Server".
RemoteJobName	varchar	The name of the temporary CheckDB job on the remote server. If the RemoteCheckDBMode is "Connected", this can be NULL. Otherwise, RemoteJobName must be populated. This field accepts Inline Tokens. Valid values: *NULL* <job name> For more information, see "About: Remote CheckDB" and "How to: Set up CheckDB on a Remote Server".
RemoteCheckDBMode	varchar	The mode of the remote CheckDB operation, if any. ***NULL*** means that remote CheckDB is not in use for this entry. **Connected** mode runs CheckDB from the local server against the remote server (very like running it against a remote server from SQL Server Management Studio).

		Disconnected mode creates a setup so that CheckDB runs entirely on the remote server. All objects are created on the remote server, and the remote server runs operations independently and reports back. **Note:** Connected mode has fewer moving parts; but Disconnected mode has higher tolerance for things like network fluctuations. Valid values: Connected Disconnected For more information, see "About: Remote CheckDB" and "How to: Set up CheckDB on a Remote Server".
RemoteRestoreMode	varchar	The method by which MC will restore a backup to the remote server, for remote integrity check operations. **Note:** Remote restores apply only to CheckDB operations, not CheckTable. Valid values: NONE LastMinionBackup NewMinionBackup For more information, see "About: Remote CheckDB" and "How to: Set up CheckDB on a Remote Server".
DropRemoteDB	bit	Determines whether the remote CheckDB process drops the remote database after the operation. You might not want to drop the database if, for example, it's supposed to be there for development or QA purposes. For more information, see "About: Remote CheckDB" and "How to: Set up CheckDB on a Remote Server".
DropRemoteJob	bit	Determines whether the remote CheckDB process drops the remote database after the operation. By default, this should be enabled.

		For more information, see "About: Remote CheckDB" and "How to: Set up CheckDB on a Remote Server".
LockDBMode	varchar	**This field is not yet in use.**
ResultMode	varchar	This determines how much detail of the integrity check results to keep in the Minion.CheckDBResult table. *NULL* and SUMMARY will keep only the rows like 'CHECKDB found%allocation errors and %consistency errors in database%'. FULL will keep everything from a run. NONE keeps nothing from a run. Valid values: ***NULL*** (this is the same as SUMMARY) **SUMMARY** **FULL** **NONE**
HistRetDays	int	Number of days to retain a history of operations (in Minion CheckDB log tables). Minion CheckDB does not modify or delete information in system tables. **Note:** This setting is also optionally configurable at multiple levels. So, you can keep log history for different amounts of time for one database vs another
PushToMinion	varchar	Determines whether log data is only stored on the local (client) server, or on both the local server and the remote server. Valid values will include: Local Remote
MinionTriggerPath	varchar	UNC path where the Minion logging trigger file is located. Not applicable for a standalone Minion CheckDB instance.
AutoRepair	varchar	**This field is not yet in use.**
AutoRepairTime	varchar	**This field is not yet in use.**
DefaultSchema	varchar	If you define specific tables to undergo DBCC CHECKTABLE, and you do not define a schema for those tables, then the system uses this DefaultSchema.

		Note: This only applies to rows with OpName=CHECKTABLE. If you leave this value NULL, MC will automatically use the dbo schema.
DBPreCode	varchar	Code to run for a database, before the operation begins for that database. For more on this topic, see "How To: Run code before or after integrity checks".
DBPostCode	varchar	Code to run for a database, after the operation completes for that database. For more on this topic, see "How To: Run code before or after integrity checks".
TablePreCode	varchar	Code to run for a database, before the operation begins for each included table. For more on this topic, see "How To: Run code before or after integrity checks".
TablePostCode	varchar	Code to run for a database, after the operation completes for each included table. For more on this topic, see "How To: Run code before or after integrity checks".
StmtPrefix	nvarchar	This column allows you to prefix *every* integrity check statement with a statement of your own. This is different from the precode and postcode, because it is run in the same *batch*. Whereas, precode and postcode are run as completely separate statements, in different contexts. Code entered in this column MUST end in a semicolon. For more on this topic, see "How To: Run code before or after integrity checks".
StmtSuffix	nvarchar	This column allows you to suffix *every* integrity check statement with a statement of your own. This is different from the precode and postcode, because it is run in the same *batch*. Whereas, precode and postcode are run as completely separate statements, in different contexts. Code entered in this column MUST end in a semicolon.

		For more on this topic, see "How To: Run code before or after integrity checks".
DBInternalThreads	tinyint	The number of CheckTable operations to run simultaneously. **Note:** If you specify DBInternalThreads in Minion.CheckDBSettingsServer, that value takes precedence over this field. Warning: You can max out server resources very quickly if you use too many concurrent operations. If for example you're running 5 databases simultaneously, and each of those operations runs 10 tables simultaneously, that can add up very quickly!
DefaultTimeEstimateMins	Int	How long you estimate the operation will take, in minutes. If you want to limit the operation based off of time (e.g., run for two hours), and the database has never been run before. So, the system has no way to know how long the operation will take.
LogSkips	bit	Whether or not you want to log skipped objects. For example: You have limited the operation to an hour, and it is cycling through CheckTable opeartions. Some tables will be skipped if the time limit is exceeded. Do you want to add those to the log, to see which ones were skipped? It can be a good idea to set LogSkips to 0 (i.e., "do not log tables that were skipped") if you routinely have a very high number of tables that will be skipped; this prevents log bloat.
BeginTime	varchar	The start time at which this configuration applies. **IMPORTANT:** Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
EndTime	Varchar	The end time at which this configuration applies.

		IMPORTANT: Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both ’00:00:00’ and ’08:15:00:000’ are valid times, but ‘8:15:00:000’ is not (because single digit hours must have a leading 0).
DayOfWeek	varchar	The day or days to which the settings apply. Valid inputs: Daily Weekday Weekend [an individual day, e.g., Sunday]
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

IMPORTANT: Remote restores apply only to CheckDB operations, not CheckTable.

Minion.CheckDBSettingsRemoteThresholds

Minion CheckDB provides remote integrity checks, where a database may be restored to another instance for DBCC CheckDB operations. **This table allows you to define thresholds to prevent smaller databases from taking part in remote CheckDB operations.**

Note: Remote operations only apply to DBCC CheckDB. MC does not support remote CheckTable.

Minion.CheckDBSettingsRemoteThresholds is very similar to Minion.CheckDBSettingsAutoThresholds, except that this table does not have a ThresholdMethod column; the method here will only ever be *size*.

To turn on this feature, Minion.CheckDBSettingsDB IsRemote must be set to 0. While this may seem counterintuitive, IsRemote = 1 turns on remote CheckDB for *all* databases (that the given row applies to). If you wish to handle remote operations dynamically, based on database size, set IsRemote = 0 – meaning, “I want operations to be local *unless* a database crosses the threshold”.

For full instructions on configuring remote CheckDB, see the remote thresholds section of “How to: Set up CheckDB on a Remote Server”. Also see “About: Remote CheckDB”.

Name	Type	Description
ID	int	Primary key row identifier.
DBName	varchar	Database name.
ThresholdType	varchar	The threshold type, as it relates to ThresholdMethod. ***NULL*** (this is the same as Data) **Data**

		DataAndIndex **File**
ThresholdMeasure	varchar	The measure for our threshold value. Valid inputs: **GB**
ThresholdValue	int	The correlating value to ThresholdMeasure. If ThresholdMeasure is GB, then ThresholdValue is the value – the number of gigabytes.
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.

Minion.CheckDBSettingsRotation

Minion CheckDB allows you to define a rotation scenario for your operations. For example, a nightly round of 10 databases would perform integrity checks on 10 databases the first night, another 10 databases the second night, and so on.

You can also use the rotational scheduling to limit operations by time; for example, you could configure MC to cycle through DBCC CheckDB operations for 90 minutes each night.

This table holds the rotation scenario for your operations (e.g., "run CheckDB on 10 databases every night; the next night, process the next 10; and so on"). This table applies to both CheckDB and CheckTable operations.

For more information, see "About: Rotational Scheduling" and "How to: Configure Rotational Scheduling".

Name	Type	Description
ID	bigint	Primary key row identifier.
DBName	varchar	Database name. Note that this field only applies to rows with OpName = 'CHECKTABLE'. For CHECKDB rows, feel free to use 'MinionDefault' or leave it NULL.
OpName	varchar	The name of the operation to be performed. Valid values: CHECKTABLE CHECKDB
RotationLimiter	varchar	The method by which to limit the rotation. **DBCount** limits the number of databases processed in a single operation; this only applies to CHECKDB operations.

		TableCount limits the number of tables processed in a single operation; this only applies to CHECKTABLE operations. Time limits the operation by a number of minutes. Valid values: DBCount TableCount Time
RotationLimiterMetric	varchar	The metric by which the RotationLimiter is defined. In Minion CheckDB 1.0, each RotationLimiter has only one possible metric: DBCount and count, TableCount and count, Time and Mins (minutes). Valid values: Count Mins
RotationMetricValue	int	The number associated with the RotationLimiter, e.g., 10 for 10 databases, or 120 for 120 Mins.
RotationPeriodInDays	int	**This field is not yet in use.**
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

Minion.CheckDBSettingsServer

This table contains server-level integrity check settings, including schedule information. The primary Minion CheckDB job "MinionCheckDB-AUTO" runs regularly in conjunction with this table to provide a wide range of CheckDB options, all without introducing additional jobs.

Name	**Type**	**Description**
ID	int	Primary key row identifier.
DBType	varchar	Database name.
OpName	varchar	The name of the operation (usually, as passed into Minion.CheckDBMaster). The AUTO option allows Minion CheckDB to choose the appropriate operation per database, based on settings in the Minion.CheckDBSettingsAutoThresholds table. For more information on this, see the section

		titled "How to: Configure Minion CheckDB Dynamic Thresholds". Valid values: CHECKTABLE CHECKDB AUTO CHECKALLOC
Day	varchar	The day or days to which the settings apply. See the discussion below for information about Day hierarchy and precedence. Note that the least frequent "Day" settings – FirstOfYear, LastOfYear, FirstOfMonth, LastOfMonth – only apply to user databases, not to system databases. Valid values: Daily Weekday Weekend [an individual day, e.g., Sunday] FirstOfMonth LastOfMonth FirstOfYear LastOfYear
ReadOnly	tinyint	Readonly option; this decides whether or not to include ReadOnly databases in the operation, or to perform operations on *only* ReadOnly databases. A value of 1 includes ReadOnly databases; 2 excludes ReadOnly databases; and 3 only includes ReadOnly databases. Valid values: 1 2 3
BeginTime	varchar	The start time at which this schedule applies. **IMPORTANT:** Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
EndTime	varchar	The end time at which this schedule applies.

		IMPORTANT: Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
MaxForTimeframe	int	Maximum number of iterations within the specified timeframe (BeginTime to EndTime). For more information, see "Table based scheduling" in the "Quick Start" section.
FrequencyMins	int	The frequency (in minutes) that the operation should occur. Note that actual frequency also depends on the SQL Agent job schedule. If FrequencyMins = 60, but the job runs every 12 hours, you will only get this operation every 12 hours. However, if FrequencyMins = 720 (12 hours) and the job runs every hour, this CheckDB will occur every 720 minutes.
CurrentNumOps	int	Count of operation attempts for the particular DBType, OpName, and Day, for the current timeframe (BeginTime to EndTime).
NumConcurrentOps	tinyint	The number of concurrent processes used. This is the number of databases that will be processed simultaneously. This applies to both DBCC CheckDB or DBCC CheckTable. Warning: You can max out server resources very quickly if you use too many concurrent operations. For more information, see "About: Multithreading operations".
DBInternalThreads	tinyint	The number of tables that will be processed in parallel. This only applies to DBCC CheckTable operations. This setting overrides the **DBInternalThreads** column in Minion.CheckDBSettingsDB. Warning: You can max out server resources very quickly if you use too many concurrent operations.

		For more information, see "About: Multithreading operations".
TimeLimitInMins	int	The time limit to impose on this opertion, in minutes.
LastRunDateTime	datetime	The last time an operation ran that applied to this particular scenario (DBType, OpName, Day, and timeframe).
Include	nvarchar	The value to pass into the @Include parameter of the Minion.CheckDBMaster job; in other words, the databases to include in this attempt. This may be left NULL (meaning "all databases").
Exclude	nvarchar	The value to pass into the @Exclude parameter of the Minion.CheckDBMaster job; in other words, the databases to exclude from this attempt. This may be left NULL (meaning "no exclusions").
Schemas	nvarchar	The schemas on which to perform operations. May be a single schema, an explicit list, and/or LIKE expressions. Applies only to CHECKTABLE operations. Note that schemas apply to all databases. If you choose to limit to the dbo schema, the operation is limited to the dbo schema in all applicable databases.
Tables	nvarchar	The tables on which to perform operations. May be a single schema, an explicit list, and/or LIKE expressions. Applies only to CHECKTABLE operations. Note that tables apply to all databases. If you choose to limit to tables named 'T%' schema, the operation is limited to 'T%' tables in all applicable databases.
BatchPreCode	varchar	Precode to run before the entire operation.
BatchPostCode	varchar	Precode to run after the entire operation.
Debug	bit	Enable logging of special data to the debug tables. For more information, see "Minion.CheckDBDebug" and "Minion.CheckDBDebugLogDetails".
FailJobOnError	bit	Cause the job to fail if an error is encountered. If an error is encountered, the rest of the batch will complete before the job is marked failed.

FailJobOnWarning	bit	Cause the job to fail if a warning is encountered. If a warning is encountered, the rest of the batch will complete before the job is marked failed.
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

Example: Daily PHYSICAL_ONLY, weekly complete full DBCC CHECKDB

We can use this table to define a new integrity check time scenarios:

- Full system DBCC CheckDBs on Saturday, one time between 6pm and 11pm.
- Full user DBCC CheckDBs on Sunday, one time between 6pm and 11pm.
- PHYSICAL_ONLY system DBCC CheckDBs on every other day (Monday-Friday), one time each between 6pm and 11pm.

The basic process is:

1. Make sure the Minion CheckDB job runs frequently enough.
2. Schedule the operations in Minion.CheckDBSettingsServer.
3. Configure the settings, and when they apply, in Minion.CheckDBSettingsDB.

Make sure the Minion CheckDB job runs frequently enough. Set the MinionCheckDB-AUTO job to run at least daily during the 6pm-11pm window.

Schedule the operations in Minion.CheckDBSettingsServer. Define the following rows. (Note that some of the table columns are omitted here, for presentation purposes.)

ID	DBType	OpName	Day	ReadOnly	BeginTime	EndTime	MaxForTimeframe
1	System	CHECKDB	Saturday	1	18:00:00	23:00:00	1
2	User	CHECKDB	Sunday	1	18:00:00	23:00:00	1
3	System	CHECKDB	Weekday	1	18:00:00	23:00:00	1

Note that if you have two operations slated for the same window of time, a System database operation takes precedence over a User database operation; and a CHECKDB or AUTO operation takes precedence over a CHECKTABLE operation.

Configure the settings, and when they apply, in Minion.CheckDBSettingsDB. The schedule above doesn't actually cover the "PHYSICAL_ONLY" aspect for our scenario. So, we must configure PHYSICAL_ONLY in Minion.CheckDBSettingsDB, with the proper time window. The following statement inserts rows for PHYSICAL_ONLY that applies to Weekdays (one row for CheckDB settings and one for CheckTable settings):

```
INSERT  INTO [Minion].CheckDBSettingsDB
```

```
(           [DBName], [OpLevel], [OpName], [Exclude], [GroupOrder], [GroupDBOrder],
            [NoIndex], [RepairOption], [RepairOptionAgree], [AllErrorMsgs],
            [ExtendedLogicalChecks], [NoInfoMsgs], [IsTabLock], [IntegrityCheckLevel],
            [IsRemote], [ResultMode], [HistRetDays], [DefaultSchema], [DBInternalThreads],
            [LogSkips], [BeginTime], [EndTime], [DayOfWeek], [IsActive], [Comment]
)
      VALUES ('    MinionDefault'   -- DBName
           , 'DB'                       -- OpLevel
           , 'CHECKDB'                  -- OpName
           , 0                  -- Exclude
           , 0                  -- GroupOrder
           , 0                  -- GroupDBOrder
           , 0                  -- NoIndex
           , 'NONE'  -- RepairOption
           , 0                  -- RepairOptionAgree
           , 1                  -- AllErrorMsgs
           , 0                  -- ExtendedLogicalChecks
           , 0                  -- NoInfoMsgs
           , 0                  -- IsTabLock
                       , 'PHYSICAL_ONLY'                -- IntegrityCheckLevel
           , 0                  -- IsRemote
           , 'Full'    -- ResultMode
           , 60                 -- HistRetDays
           , 'dbo'     -- DefaultSchema
           , 1                  -- DBInternalThreads
           , 1                  -- LogSkips
           , '00:00:00'         -- BeginTime
           , '23:59:00'         -- EndTime
           , 'Weekday'          -- DayOfWeek
           , 1                  -- IsActive
           , 'MinionDefault PHYSICAL_ONLY CHECKDB on weekdays.')  -- Comment
                       ,
                       ('        MinionDefault'   -- DBName
           , 'DB'                       -- OpLevel
           , 'CHECKTABLE'               -- OpName
           , 0                  -- Exclude
           , 0                  -- GroupOrder
           , 0                  -- GroupDBOrder
           , 0                  -- NoIndex
           , 'NONE'  -- RepairOption
           , 0                  -- RepairOptionAgree
           , 1                  -- AllErrorMsgs
           , 0                  -- ExtendedLogicalChecks
           , 0                  -- NoInfoMsgs
           , 0                  -- IsTabLock
            , 'PHYSICAL_ONLY'                  -- IntegrityCheckLevel
           , 0                  -- IsRemote
           , 'Full'    -- ResultMode
           , 60                 -- HistRetDays
           , 'dbo'     -- DefaultSchema
           , 1                -- DBInternalThreads
           , 1                -- LogSkips
           , '00:00:00'       -- BeginTime
           , '23:59:00'       -- EndTime
           , 'Weekday'        -- DayOfWeek
           , 1                -- IsActive
           , 'MinionDefault PHYSICAL_ONLY CheckTable on weekdays.');          -- Comment
```

We also need to update the two existing ‘MinionDefault’ rows, so they only apply to the weekend:

```
UPDATE  Minion.CheckDBSettingsDB
SET     DayOfWeek = 'Weekend'
WHERE   DBName = 'MinionDefault'
        AND IntegrityCheckLevel IS NULL;
```

The final result in Minion.CheckDBSettingsDB is:

DBName	OpLevel	OpName	IntegrityCheckLevel	BeginTime	EndTime	DayOfWeek
MinionDefault	DB	CHECKDB	*NULL*	00:00:00	23:59:00	Weekend
MinionDefault	DB	CHECKTABLE	*NULL*	00:00:00	23:59:00	Weekend
MinionDefault	DB	CHECKDB	**PHYSICAL_ONLY**	00:00:00	23:59:00	**Weekday**
MinionDefault	DB	CHECKTABLE	**PHYSICAL_ONLY**	00:00:00	23:59:00	**Weekday**

Minion.CheckDBSettingsSnapshot

This table holds the settings for custom database snapshots.

> “A database snapshot is a read-only, static view of a SQL Server database (the source database). The database snapshot is transactionally consistent with the source database as of the moment of the snapshot's creation. A database snapshot always resides on the same server instance as its source database. As the source database is updated, the database snapshot is updated.”
> - MSDN article “Database Snapshots” (https://msdn.microsoft.com/en-us/library/ms175158.aspx)

When you run DBCC CheckDB or DBCC CheckTable, behind the scenes SQL Server creates a snapshot of the database to run the operation against. SQL Server decides where to place the files for these snapshots, and deletes the snapshot after the operation is complete.

If your version of SQL Server supports it, you can also choose to create a custom snapshot (CustomSnapshot=1). For more information, and to learn how to configure custom snapshots, see “About: Custom Snapshots” and “How to: Configure Custom Snapshots”.

Note: SQL Server 2016 and earlier versions only allow custom snapshots for Enterprise edition. SQL Server 2016 SP1 allow custom snapshots in any edition.

Note that Minion CheckDB comes with two “MinionDefault” rows in this table – one for CHECKDB and one for CHECKTABLE – both with CustomSnapshot = 0. These are example rows so you can easily enable custom snapshots.

Name	Type	Description
ID	int	Primary key row identifier.
DBName	varchar	Database name.
OpName	varchar	The name of the operation (usually, as passed into the Minion.CheckDBMaster procedure from Minion.CheckDBSettingsDB). Valid values: CHECKTABLE CHECKDB
CustomSnapshot	bit	Enable or disable custom snapshots. **IMPORTANT:** If custom snapshots are enabled, MC *requires* active rows in Minion.CheckDBSnapshotPath to determine where the custom snapshot will go. **Note:** If CustomSnapshot is enabled and your version of SQL Server doesn’t support it, that integrity check operation will complete using the default internal snapshot. For more information, see the “Custom snapshots fail” section under Troubleshooting.
SnapshotRetMins	int	The number of minutes to retain a snapshot, before recreating it. This only applies to rows with OpName=’CHECKTABLE’. For more information, see “How to: Configure Custom Snapshots”.
SnapshotRetDeviation	int	**This field is not yet in use.**
DeleteFinalSnapshot	bit	Whether to delete the last snapshot taken during an operation.
SnapshotFailAction	varchar	The action to take if the custom snapshot fails. For example, if you the custom snapshot location doesn’t exist, or you don’t have permissions to it, or some other problem exists, then this field determines how to proceed. **FAIL** will fail with a logged error. Default behavior. **CONTINUE** will allow MC to continue with an internal snapshot, and will log the error in the Warnings column of the log table.

		Valid values: *NULL* <this is the same as FAIL> FAIL CONTINUE CONTINUEWITHTABLOCK
BeginTime	Varchar	The start time at which these settings apply. Can be NULL, meaning "no start limit". **IMPORTANT:** Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
EndTime	varchar	The end time at which these settings apply. Can be NULL, meaning "no end limit". **IMPORTANT:** Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
DayOfWeek	varchar	The day or days to which the settings apply. Valid inputs: *NULL* (meaning, all days) Daily Weekday Weekend [an individual day, e.g., Sunday]
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

Minion.CheckDBSettingsTable

Minion.CheckDBSettingsTable allows you to configure table-level exceptions to the CHECKTABLE settings defined in Minion.CheckDBSettingsDB.

IMPORTANT: Minion.CheckDBSettingsDB *must* have settings for CHECKTABLE operations defined. This table is used to define individual exceptions.

For more information on DBCC CheckTable options, see the DBCC CheckTable article on MSDN: https://msdn.microsoft.com/en-us/library/ms174338.aspx

Name	Type	Description
ID	int	Primary key row identifier.
DBName	varchar	Database name. Required.
SchemaName	varchar	Schema name. Required.
TableName	varchar	Table name. Required.
IndexName	varchar	**This field is not yet in use.**
Exclude	bit	Exclude database (or, if specified, the specific table) from operations. For more on this topic, see "How To: Exclude databases from operations" and "Include and Exclude Precedence".
GroupOrder	int	The operation order within a group. Used solely for determining the order in which tables should be processed. By default, all tables have a value of 0, which means they'll be processed in the order they're queried from sysobjects. Higher numbers have a greater "weight" (they have a higher priority), and will be processed earlier than lower numbers. We recommend leaving some space between assigned order numbers (e.g., 10, 20, 30) so there is room to move or insert rows in the ordering. For more information, see "How To: Process databases in a specific order".
GroupTableOrder	int	Group to which this table belongs. Used solely for determining the order in which tables should be processed. By default, all tables have a value of 0, which means they'll be processed in the order they're queried from sysobjects. Higher numbers have a greater "weight" (they have a higher priority), and will be processed earlier than lower numbers. The range of GroupTableOrder weight numbers is 0-255. For more information, see "How To: Process databases in a specific order".
DefaultTimeEstimateMins	int	How long you estimate the operation will take, in minutes. If you want to limit the operation based off of time (e.g., run for two hours), and the table has never been run before. So, the system has

		no way to know how long the operation will take.
PreferredServer	varchar	For remote CheckDB runs, the name of the remote server.
TableOrderType	varchar	Order the table using different metrics, such as size, usage, etc. **This field is not yet in use.**
NoIndex	bit	DBCC CheckTable option NOINDEX. Specifies that intensive checks of nonclustered indexes for user tables should not be performed.
RepairOption	varchar	The repair option to use. **This field is not yet in use.** Future valid values may include: *NULL* NONE REPAIR_ALLOW_DATA_LOSS REPAIR_FAST REPAIR_REBUILD
RepairOptionAgree	bit	Signifies that you agree to the repair option specified in the RepairOption column. This is in place because some repair options (i.e., "REPAIR_ALLOW_DATA_LOSS") can cause you to lose data. **This field is not yet in use.**
AllErrorMsgs	bit	DBCC CheckTable option ALL_ERRORMSGS.
ExtendedLogicalChecks	bit	DBCC CheckTable option EXTENDED_LOGICAL_CHECKS.
NoInfoMsgs	bit	DBCC CheckTable option NO_INFOMSGS. Suppresses all informational messages.
IsTabLock	bit	DBCC CheckTable option -tablock. Causes DBCC CHECKTABLE to obtain a shared table lock instead of using an internal database snapshot. **IMPORTANT:** We do not recommend using tablock on production systems!
ResultMode	varchar	This determines how much detail of the integrity check results to keep in the Minion.CheckDBCheckTableResult table. *NULL* and SUMMARY will keep only the rows like 'CHECKDB found%allocation errors and %consistency errors in database%'. FULL will keep everything from a run.

		NONE keeps nothing from a run. Valid values: ***NULL*** (this is the same as SUMMARY) **SUMMARY** **FULL** **NONE**
IntegrityCheckLevel	varchar	DBCC CheckTable option. This controls whether or not you include physical only, data purity, or neither. Valid values: *NULL* PHYSICAL_ONLY DATA_PURITY
HistRetDays	int	Number of days to retain a history of operations (in Minion CheckDB log tables). Minion CheckDB does not modify or delete information in system tables. **Note:** This setting is also optionally configurable at multiple levels. So, you can keep log history for different amounts of time for one database vs another.
TablePreCode	varchar	Code to run for a table, before the operation begins for that table. For more on this topic, see "How To: Run code before or after integrity checks".
TablePostCode	varchar	Code to run for a table, after the operation begins for that table. For more on this topic, see "How To: Run code before or after integrity checks".
StmtPrefix	nvarchar	This column allows you to prefix *every* integrity check statement with a statement of your own. This is different from the precode and postcode, because it is run in the same *batch*. Whereas, precode and postcode are run as completely separate statements, in different contexts. Code entered in this column MUST end in a semicolon. For more on this topic, see "How To: Run code before or after integrity checks".
StmtSuffix	nvarchar	This column allows you to suffix *every* integrity check statement with a statement of your

		own. This is different from the precode and postcode, because it is run in the same *batch*. Whereas, precode and postcode are run as completely separate statements, in different contexts. Code entered in this column MUST end in a semicolon. For more on this topic, see "How To: Run code before or after integrity checks".
BeginTime	varchar	The start time at which this configuration applies. **IMPORTANT:** Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
EndTime	varchar	The end time at which this configuration applies. **IMPORTANT:** Must be in the format *hh:mm:ss,* or *hh:mm:ss:mmm* (where *mmm* is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
DayOfWeek	varchar	The day or days to which the settings apply. Valid inputs: Daily Weekday Weekend [an individual day, e.g., Sunday]
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

Minion.CheckDBSnapshotPath

This table allows you to configure snapshot file path settings for local custom snapshots. You can specify one row per snapshot file, or you can specify one location for all snapshot files using FileName='MinionDefault'.

Note: SQL Server does not allow you to specify the snapshot log file location, and so neither does this table.

For more information, see "How to: Configure Custom Snapshots".

Name	**Type**	**Description**
ID	int	Primary key row identifier.
DBName	varchar	Name of the origin database. 'MinionDefault' applies to all databases. Valid values: <specific database name> MinionDefault
OpName	varchar	The name of the operation used. Valid values: CHECKDB CHECKTABLE
FileName	varchar	Name of the file. 'MinionDefault' applies to all files. Valid values: <specific file name> MinionDefault
SnapshotDrive	varchar	Snapshot drive. This is only the drive letter of the snapshot destination. **IMPORTANT**: If this is drive, this must end with colon-slash (for example, 'M:\'). If this is UNC, use the base path (for example, '\\server2\')
SnapshotPath	varchar	Snapshot path. This is only the path (for example, 'SnapshotCheckDB\') of the snapshot destination.
ServerLabel	varchar	A user-customized label for the server name. It can be the name of the server, server\instance, or a label for a server.
PathOrder	Int	If a snapshot goes to multiple drives, then PathOrder is used to determine the order in which the different drives are used. **IMPORTANT:** Like all ranking fields in Minion, PathOrder is a weighted measure. Higher numbers have a greater "weight" - they have a higher priority - and will be used earlier than lower numbers.
IsActive	bit	Whether the current row is valid (active), and should be used in the Minion CheckDB process.
Comment	Varchar	For your reference only. You can label each row with a short description and/or purpose.

Minion.DBMaintInlineTokens

Minion CheckDB 1.0 and MinionBackup 1.3 introduce a new feature to the Minion suite – Inline Tokens. Inline Tokens allow you use defined patterns to create dynamic names. For example, MC comes with the predefined Inline Token "Server" and "DBName".

In this version of MC, inline tokens are accepted for remote CheckDB operations. Specifically, the PreferredDBName and RemoteJobName in the Minion.CheckDBSettingsDB table:

```
UPDATE Minion.CheckDBSettingsDB
SET   PreferredDBName = '%Server%_%DBName%',
        RemoteJobName = 'MinionCheckDB_%Server%_%DBName%';
```

MC recognizes %Server% and %DBName% as Inline Tokens, and refers to the Minion.DBMaintInlineTokens table for the definition. Note that custom tokens must be used with pipe delimiters, instead of percent signs: '|MyCustomToken|'.

For more information, see "About: Inline Tokens".

Note that this table is shared between Minion modules.

Name	Type	Description
ID	Int	Primary key row identifier.
DynamicName	varchar	The name of the dynamic part, e.g., "Date". We recommend you do not include any special symbols – only alphanumeric characters.
ParseMethod	varchar	The definition of the dynamic part. Typically, this is a TSQL expression that resolves to the value desired. For example, the ParseMethod for "Millisecond" is **DATEPART(MILLISECOND, @ExecutionDateTime)** **Note:** Custom inline tokens cannot use internal variables like @ExecutionDateTime; only SQL functions and @@ variables.
IsCustom	bit	Whether this is a custom dynamic part, or one that came with the product originally.
Definition	varchar	This is the official description of the dynamic part. Example (BackupTypeExtension): "Returns a dynamic backup file extension based on the backup type."

		Note that certain built-in token definitions are hard coded in the procedure; entries here are simply a placeholder. So, do not modify or disable definitions
IsActive	bit	The current row is valid (active), and should be used in the Minion Backup process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

Log Table Detail

The data in log tables is retained according to the HistRetDays columns in Minion.CheckDBSettingsDB and Minion.CheckDBSettingsTable.

Minion.CheckDBLog

Contains records of integrity check operations. It contains one time-stamped row for each run of Minion.CheckDBMaster, which may encompass several database integrity check operations. This table stores status information for the overall operation. This information can help with troubleshooting, or just information gathering when you want to see what has happened between one backup run to the next.

Name	**Type**	**Description**
ID	Bigint	Primary key row identifier.
ExecutionDateTime	datetime	Date and time of the operation.
Status	varchar	Current status of the operation. If Live Insight is being used the status updates will appear here. When finished, this column will typically either read 'Complete' or 'Complete with warnings'. If, for example, the process was halted midway through the operation, the Status would reflect the step in progress at the time the operation stopped.
DBType	varchar	Database type. Valid values: System User
OpName	varchar	The name of the operation (usually, as passed into Minion.CheckDBMaster). Valid values: CHECKTABLE CHECKDB AUTO
NumConcurrentProcesses	tinyint	The number of concurrent processes used.

		This is the number of databases that will be processed simultaneously (CheckDB or CheckTable).
DBInternalThreads	tinyint	If CheckTable, this is the number of tables that will be processed in parallel.
NumDBsOnServer	int	Number of databases on server.
NumDBsProcessed	int	Number of databases processed in this operation.
RotationLimiter	varchar	The method that was used to limit the rotation (DBCount, TableCount, or Time).
RotationLimiterMetric	varchar	The metric by which the RotationLimiter was defined (count, or minutes).
RotationMetricValue	int	The number associated with the RotationLimiter, e.g., 10 for 10 databases, or 120 for 120 Mins.
TimeLimitInMins	int	The time limit imposed on this opertion, in minutes.
ExecutionEndDateTime	datetime	Date and time the entire operation completed.
ExecutionRunTimeInSecs	float	The duration, in seconds, of the entire operation.
BatchPreCodeStartDateTime	datetime	Start date of the batch precode.
BatchPostCodeStartDateTime	datetime	Start date of the batch postcode.
BatchPreCode	varchar	Precode set to run before the entire operation. This code is set in the Minion.CheckDBSettingsServer table.
BatchPostCode	varchar	Precode set to run after the entire operation. This code is set in the Minion.CheckDBSettingsServer table.
Schemas	varchar	The schema or schemas that were passed in to the operation. Schemas = *NULL* means the maintenance was not limited by schema. See the @Schema entry for Minion.CheckDBMaster for more information.
Tables	varchar	The table or tables that were passed into the operation. Tables = *NULL* means the maintenance was not limited by table. See the @Table entry for Minion.CheckDBMaster for more information.
IncludeDBs	varchar	A comma-delimited list of database names, and/or wildcard strings, included in the operation.
ExcludeDBs	varchar	A comma-delimited list of database names, and/or wildcard strings, excluded from the operation.

RegexDBsIncluded	varchar	A list of databases included in the backup operation via the Minion CheckDB regular expressions feature.
RegexDBsExcluded	varchar	A list of databases excluded from the backup operation via the Minion CheckDB regular expressions feature.

Minion.CheckDBLogDetails

Contains records of individual integrity check operations. It contains one time-stamped row for each individual DBCC CheckDB or DBCC CheckTable operation. This table stores the parameters and settings that were used during the operation, as well as status information. This information can help with troubleshooting, or just information gathering when you want to see what has happened between one backup run to the next.

Name	**Type**	**Description**
ID	bigint	Primary key row identifier.
ExecutionDateTime	datetime	Date and time of the operation.
Status	varchar	Current status of the operation. If Live Insight is being used the status updates will appear here. For a full description of status messages, see the discussion below.
PctComplete	tinyint	Operation percent complete (e.g., 50% complete).
DBName	varchar	Database name.
CheckDBName	varchar	The database name; or, the name of the database in the case of a remote CheckDB or custom snapshot.
ServerLabel	varchar	A user-customized label for the server name. It can be the name of the server, server\instance, or a label for a server.
NETBIOSName	varchar	The name of the server on which the database resides. If the instance is on a cluster, this will be the name of the cluster node SQL Server was running on. If it's part of an Availability Group, the NETBIOSName will be the physical name of the Availability Group replica.
IsRemote	bit	Whether this is a remote CheckDB operation, or not.
PreferredServer	varchar	For remote CheckDB runs, the name of the remote server.
PreferredDBName	varchar	For remote CheckDB runs, the raw database name from the Minion.CheckDBSettingsDB table (including Inline Tokens, if any). You can use this to compare to the CheckDBName field, to see what the expression (if any) evaluated to.

RemoteCheckDBMode	Varchar	The mode of the remote CheckDB operation, if any. Valid values: *NULL* Connected Disconnected
RemoteRestoreMode	Varchar	The mode of the remote restore, if any. Valid values include: None LastMinionBackup NewMinionBackup
IsClustered	bit	Whether or not the server is clustered.
IsInAG	bit	Whether or not the server is in an Availability Group.
IsPrimaryReplica	bit	Whether or not the server is the primary replica.
DBType	varchar	Database type. Valid values: User System
OpName	varchar	The name of the operation (usually, as passed into Minion.CheckDBMaster). Valid values: CHECKTABLE CHECKDB AUTO
SchemaName	varchar	Schema name.
TableName	varchar	Table name.
IndexName	varchar	**This field is not yet in use.**
IndexID	bigint	**This field is not yet in use.**
IndexType	varchar	**This field is not yet in use.**
GroupOrder	int	The operation order within a group. Used solely for determining the order in which databases should be processed. By default, all databases have a value of 0, which means they'll be processed in the order they're queried from sysobjects. Higher numbers have a greater "weight" (they have a higher priority), and will be backed up earlier than lower numbers. We recommend leaving some space between assigned back up order numbers (e.g., 10, 20, 30) so there is room to move or insert rows in the ordering.

		For more information, see "How To: Process databases in a specific order".
GroupDBOrder	int	Group to which this database belongs. Used solely for determining the order in which databases should be processed. By default, all databases have a value of 0, which means they'll be processed in the order they're queried from sysobjects. Higher numbers have a greater "weight" (they have a higher priority), and will be backed up earlier than lower numbers. The range of GroupDBOrder weight numbers is 0-255. For more information, see "How To: Process databases in a specific order".
SizeInMB	float	Database size, in MB.
TimeLimitInMins	int	The time limit imposed on this opertion, in minutes.
EstimatedTimeInSecs	int	The estimated time to complete the operation.
EstimatedKBperMS	float	The estimated speed of the operation, as measured in KB per millisecond.
LastOpTimeInSecs	int	The time taken to complete the previous operation for this database.
IncludeRemoteInTimeLimit	int	Whether or not the remote operation (if any) is included in the time limit (if any).
OpBeginTime	datetime	Date and time of the operation start.
OpEndTime	datetime	Date and time of the operation end.
OpRunTimeInSecs	float	Operation duration, measured in seconds.
CustomSnapshot	bit	Whether a custom snapshot used.
MaxSnapshotSizeInMB	float	The total size of all snapshot files. This total comes from Minion.CheckDBSnapshotLog.
CheckDBCmd	varchar	The command statement used.
AllocationErrors	int	Number of allocation errors found.
ConsistencyErrors	int	Number of consistency errors found.
NoIndex	Bit	Whether NOINDEX was enabled. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
RepairOption	Varchar	The repair option used. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx

RepairOptionAgree	bit	The RepairOptionAgree value used in the operation. (See the Minion.CheckDBSettingsDB and Minion.CheckDBSettingsTable entries.)
WithRollback	varchar	The WithRollback value used in the operation. (See the Minion.CheckDBSettingsDB entry.) **This field is not yet in use.**
AllErrorMsgs	bit	The value used for the DBCC option ALL_ERRORMESSAGES. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
ExtendedLogicalChecks	Bit	The value used for the DBCC option EXTENDED_LOGICAL_CHECKS. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
NoInfoMsgs	bit	The value used for the DBCC option NO_INFOMSGS. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
IsTabLock	Bit	The value used for the DBCC option TABLOCK. For more information, see the DBCC CheckDB article on MSDN: https://msdn.microsoft.com/en-us/library/ms176064.aspx
IntegrityCheckLevel	Varchar	Integrity check level (ESTIMATEONLY, PHYSICAL_ONLY).
DisableDOP	bit	Whether parallelism (multithreading) was enabled or disabled. **IMPORTANT:** DisableDOP = 1 *disables* multithreading – i.e., processing multiple databases at the same time – in Minion CheckDB! For more information, see "About: Multithreading operations".
LockDBMode	varchar	**This field is not yet in use.**
ResultMode	varchar	How much detail of the integrity check results to keep in the Minion.CheckDBResult table. The

		operation can save either the full results, just the summary results, or no results. Valid values: FULL SUMMARY NONE
HistRetDays	Int	Number of days to retain a history of operations (in Minion CheckDB log tables). Minion CheckDB does not modify or delete information in system tables.
PushToMinion	varchar	Determines whether log data is only stored on the local (client) server, or on both the local server and the remote server. Valid values will include: Local Remote
MinionTriggerPath	varchar	UNC path where the Minion logging trigger file is located. Not applicable for a standalone Minion CheckDB instance.
AutoRepair	varchar	**This field is not yet in use.**
AutoRepairTime	varchar	**This field is not yet in use.**
LastCheckDateTime	datetime	The last time a CheckDB operation was run (as determined by either database properties or Minion.CheckDBLogDetails).
LastCheckResult	varchar	The status of the last CheckDB operation.
DBPreCodeStartDateTime	datetime	The date and time that the database precode began.
DBPreCodeEndDateTime	datetime	The date and time that the database precode ended.
DBPreCodeTimeInSecs	int	The duration of the database precode run.
DBPreCode	varchar	Code that ran before the operation completed for that database.
DBPostCodeStartDateTime	datetime	The date and time that the database postcode began.
DBPostCodeEndDateTime	datetime	The date and time that the database postcode ended.
DBPostCodeTimeInSecs	int	The duration of the database postcode run.
DBPostCode	varchar	Code that ran after the operation completed for that database.
TablePreCodeStartDateTime	datetime	The date and time that the table precode began.
TablePreCodeEndDateTime	datetime	The date and time that the table precode ended.
TablePreCodeTimeInSecs	int	The duration of the table precode run.
TablePreCode	varchar	Code that ran before the operation completed for that table.

TablePostCodeStartDateTime	datetime	The date and time that the table postcode began.
TablePostCodeEndDateTime	datetime	The date and time that the table postcode ended.
TablePostCodeTimeInSecs	int	The duration of the table postcode run.
TablePostCode	varchar	Code that ran after the operation completed for that table.
StmtPrefix	Nvarchar	The code, if any, prefixed to the integrity check statement with a statement of your own.
StmtSuffix	Nvarchar	The code, if any, suffixed to the integrity check statement with a statement of your own.
ProcessingThread	tinyint	In a multithreaded run, the number of the thread assigned to this operation. Used to query with GROUP BY to see the distribution of threads. (E.g., did one thread handle most of the work, or was there a reasonably good distribution of work?) For more information, see "About: Multithreading operations".
Warnings	varchar	Warnings encountered for the operation.

Discussion – Status messages. The Status column of Minion.CheckDBLogDetails can be any of the following:

- **Complete** - operation completed without errors.
- **Complete with errors** - operation completed, but it reported errors. Check the Consistency and AllocationErrors columns, and the Minion.CheckDBResults, table for full details.
- **Complete with Warnings** - operation completed, but there was an error with the process somewhere along the way. This is usually seen on remote CheckDB operations when the process has a problem getting the results back to the primary server. There are other circumstances that can complete with warning. There could be problems deleting the snapshot, or something else. The point is that the integrity check finished, but something else failed and it's impossible to say what the state of the error reporting will be.
- **Complete with Errors and Warnings** - a combination of the above two.
- **Complete with No Status** - This means the integrity check operation completed, but we specifically couldn't parse the error results. Again, this usually happens on remote runs when we can't figure out how many allocation or consistency errors there are, but it could happen on a local run if Microsoft sneaks in a new column into the result table. To get a "Complete" status, we rely on being able to parse the output; so when you get this message, it usually means that you don't have that return data from CheckDB/CheckTable/etc.

Minion.CheckDBResult

This keeps the actual results of DBCC CheckDB operations (as opposed to outcome and associated operational data in the "Log" tables). The level of detail kept in this table per operation is determined by the ResultMode column in Minion.CheckDBSettingsDB (e.g., SUMMARY, FULL, or NONE).

Name	Type	Description
ExecutionDateTime	datetime	Date and time of the operation.
DBName	nvarchar	Database name.
BeginTime	datetime	The date and time that the operation began.
EndTime	datetime	The date and time that the operation finished.
Error	int	The error number. (E.g., error number 8989.)
Level	int	The error level.
State	int	The error state.
MessageText	varchar	The message text. E.g., "CHECKDB found 0 allocation errors and 0 consistency errors in database 'ABC'."
RepairLevel	nvarchar	The repair level used for the operation.
Status	int	The status of the operation. (0 = Success.)
DbId	int	Database ID.
DbFragId	int	CHECKDB TABLERESULTS output; not documented.
ObjectId	bigint	Object ID.
IndexID	int	Index ID.
PartitionId	bigint	Partition ID.
AllocUnitId	int	Allocation unit ID.
RidDBId	bigint	Undocumented.
RidPruId	bigint	Undocumented.
File	int	Undocumented.
Page	bigint	Undocumented.
Slot	bigint	Undocumented.
RefDbId	int	Undocumented.
RefPruId	int	Undocumented.
RefFile	int	Undocumented.
RefPage	bigint	Undocumented.
RefSlot	bigint	Undocumented.
Allocation	int	Undocumented.

Minion.CheckDBSnapshotLog

This table keeps a record of snapshot files (one row per file). This includes files created as part of local custom snapshots.

For more information, see "How to: Configure Custom Snapshots".

Name	Type	Description
ID	int	Primary key row identifier.
ExecutionDateTime	datetime	Date and time of the operation.

OpName	varchar	The name of the operation used: CHECKDB or CHECKTABLE.
DBName	varchar	Name of the origin database.
SnapshotDBName	varchar	Name of the snapshot database.
FileID	int	ID of the file within database.
TypeDesc	varchar	Description of the file type. E.g., ROWS, LOG.
Name	varchar	Logical name of the file in the database.
PhysicalName	varchar	Operating system file name.
Size	bigint	The file size (in 8 KB pages).
IsReadOnly	bit	Whether the file is read only, or not.
IsSparse	bit	Whether the file is sparse, or not.
SnapshotDrive	varchar	Snapshot drive. This is only the drive letter of the snapshot destination (in the format 'M:\', or if this is UNC, the base path ('\\server2\').
SnapshotPath	varchar	Snapshot path. This is only the path (for example, 'SnapshotCheckDB\') of the snapshot destination.
FullPath	varchar	The full path without filename. For example: "C:\SnapshotCheckDB\".
ServerLabel	varchar	A user-customized label for the server name. It can be the name of the server, server\instance, or a label for a server.
PathOrder	Int	If a snapshot goes to multiple drives, then PathOrder is used to determine the order in which the different drives are used. **IMPORTANT:** Like all ranking fields in Minion, PathOrder is a weighted measure. Higher numbers have a greater "weight" - they have a higher priority - and will be used earlier than lower numbers.
Cmd	varchar	The snapshot's "CREATE DATABASE" statement used.
MaxSizeInMB	float	The size of the snapshot file (not of the entire snapshot), in MB.

Minion.CheckDBCheckTableResult

This keeps the results from DBCC CheckTable operations (as opposed to outcome and associated operational data in the "Log" tables). The level of detail kept in this table per operation is determined by the ResultMode column in Minion.CheckDBSettingsTable (e.g., SUMMARY, FULL, or NONE).

Name	**Type**	**Description**
ExecutionDateTime	datetime	Date and time of the operation.
DBName	nvarchar	Database name.
SchemaName	varchar	Schema name.
TableName	varchar	Table name.
IndexName	varchar	**This field is not yet in use.**

IndexType	varchar	**This field is not yet in use.**
BeginTime	datetime	The date and time that the operation began.
EndTime	datetime	The date and time that the operation finished.
Error	int	The error number. (E.g., error number 8989.)
Level	int	The error level.
State	int	The error state.
MessageText	varchar	The message text. E.g., "CHECKDB found 0 allocation errors and 0 consistency errors in database 'ABC'."
RepairLevel	nvarchar	The repair level used for the operation.
Status	int	The status of the operation. (0 = Success.)
DbId	int	Database ID.
DbFragId	int	CHECKDB TABLERESULTS output; not documented.
ObjectId	bigint	Object ID.
IndexID	int	Index ID.
PartitionId	bigint	Partition ID.
AllocUnitId	bigint	Allocation unit ID.
RidDBId	int	Undocumented.
RidPruId	int	Undocumented.
File	int	Undocumented.
Page	bigint	Undocumented.
Slot	bigint	Undocumented.
RefDbId	int	Undocumented.
RefPruId	int	Undocumented.
RefFile	bigint	Undocumented.
RefPage	bigint	Undocumented.
RefSlot	bigint	Undocumented.
Allocation	int	Undocumented.

Debug Table Detail

Note: The data in "Debug" tables (like Minion.CheckDBDebug) is useful to Minion support. Contact us through www.MinionWare.net for help with your Minion CheckDB scenarios and debugging.

Minion.CheckDBDebug

This table holds high level debugging data from Minion CheckDB runs where debugging was enabled. The Minion.CheckDBCheckTable stored procedure allows you to enable debugging.

Minion.CheckDBDebugLogDetails

This table holds detailed debugging data from Minion CheckDB runs where debugging was enabled. The Minion.CheckDBCheckTable stored procedure allows you to enable debugging.

Minion.CheckDBDebugSnapshotCreate

This table holds custom snapshot-related debugging data from Minion CheckDB runs where debugging was enabled. The Minion.CheckDBCheckTable stored procedure allows you to enable debugging.

Minion.CheckDBDebugSnapshotThreads
This table holds thread-related debugging data from Minion CheckDB runs where debugging was enabled. The Minion.CheckDBCheckTable stored procedure allows you to enable debugging.

Work Table Detail
Generally speaking, the data in work tables only lasts as long as the operation they are being used for. In other words, there is no guarantee that data in work tables will be retained for any period of time. (That's what log tables are for!)

Minion.CheckDBCheckTableThreadQueue
This table is for internal use. Information gathered in preparation for a CheckTable run is stored here.

You can use the stored procedure Minion.CheckDBCheckTable with @PrepOnly = 1 to populate this table, and then modify / add / delete the results as needed for custom or dynamic solutions.

Future solutions may include instructions on how to modify this table for custom scenarios.

Minion.CheckDBRotationDBs
This table is for internal use only.

Future solutions may include instructions on how to modify this table for custom scenarios.

Minion.CheckDBRotationDBsReload
This table is for internal use only.

Minion.CheckDBRotationTables
This table is for internal use only.

Minion.CheckDBRotationTablesReload
This table is for internal use only.

Minion.CheckDBTableSnapshotQueue
This table is for internal use only. Do not modify in any way.

Minion.CheckDBThreadQueue
This table is for internal use.

Future solutions may include instructions on how to modify this table for custom scenarios.

Minion.WorkingForTheWeekend
This table is entirely made up. If you have this table in your system, that's on you.

Overview of Views

Minion CheckDB comes with two views:

- **Minion.CheckDBLogDetailsCurrent** – Provides the most recent batch of integrity check operations.
- **Minion.CheckDBLogDetailsLatest** – Gets the latest operation for each database. This is different from the "current" view, in that the current view gets the latest operation without regard to what databases or tables were in it. In this view, we're interested in the last time a database was run.

Overview of Procedures

- **Minion.CheckDB** – This procedure runs a DBCC CheckDB operation for an individual database.
- **Minion.CheckDBCheckTable** – This procedure runs DBCC CheckTable operations for one or more individual tables.
- **Minion.CheckDBCheckTableThreadRunner** – Internal use only.
- **Minion.CheckDBMaster** – The Minion.CheckDBMaster procedure is the central procedure of Minion CheckDB. It uses the parameter and/or table data to make all the decisions on which databases to run CheckDB, and what order they should be in.
- **Minion.CheckDBRemoteRunner** – Internal use only.
- **Minion.CheckDBRotationLimiter** – Internal use only.
- **Minion.CheckDBSnapshotDirCreate** – Internal use only.
- **Minion.CheckDBSnapshotGet** – Creates the statement to create custom snapshots for CheckDB or CheckTable.
- **Minion.CheckDBStatusMonitor** – This procedure updates the status of running operations, in Minion.CheckDBLogDetails.
- **Minion.CheckDBThreadCreator** – Internal use only.

Procedures Detail

Minion.CheckDB

This procedure runs a DBCC CheckDB operation for an individual database. Minion.CheckDB is the procedure that creates and runs the actual DBCC CHECKDB statements for databases which meet the criteria stored in the settings table (Minion.CheckDBSettingsDB).

IMPORTANT: We HIGHLY recommend using Minion.CheckDBMaster for all of your integrity check operations, even when operating on a single database. Do not call Minion.CheckDB to perform integrity checks.

The Minion.CheckDBMaster procedure makes all the decisions on which databases to process, and what order they should be in. It's certainly possible to call Minion.CheckDB manually, to process an individual database, but we instead recommend using the Minion.CheckDBMaster procedure (and just include the single database using the @Include parameter). First, it unifies your code, and therefore minimizes your effort. By calling the same procedure every time you reduce your learning curve and cut down on mistakes.

Second, future functionality may move to the Minion.CheckDBMaster procedure; if you get used to using Minion.CheckDBMaster now, then things will always work as intended.

Name	Type	Description
@DBName	nvarchar	Database name.
@Op	varchar	Operation name. Valid inputs: **CHECKDB** **CHECKALLOC**
@StmtOnly	bit	Only generate CheckDB statements, instead of running them.
@ExecutionDateTime	datetime	The date and time of the batch operation; this is passed in from the Minion.CheckDBMaster procedure. Leave this NULL when running this stored procedure explicitly.
@Debug	bit	Enable logging of special data to the debug tables. For more information, see "Minion.CheckDBDebug", "Minion.CheckDBDebugSnapshotCreate", and "Minion.CheckDBDebugSnapshotThreads".
@Thread	tinyint	Internal use only.

Example execution:

```
-- Generate DBCC CHECKDB statements for database DB2, as applicable:
EXEC [Minion].[CheckDB]
        @DBName = 'DB2',
        @Op = 'AUTO',
        @StmtOnly = 1;
```

Example execution:

```
-- Generate DBCC CHECKALLOC statements for database DB1:
EXEC [Minion].[CheckDB]
        @DBName = 'DB1',
        @Op = 'CHECKALLOC',
        @StmtOnly = 1;
```

Minion.CheckDBCheckTable

This procedure runs DBCC CheckTable operations for one or more individual tables.

Minion.CheckDBCheckTable is the procedure that creates and runs the actual DBCC CHECKTABLE statements for tables, as determined in the Minion.CheckDBMaster stored procedure, and the settings tables (Minion.CheckDBSettingsDB and Minion.CheckDBSettingsTable).

IMPORTANT: We HIGHLY recommend using Minion.CheckDBMaster for all of your integrity check operations, even when operating on a single table. Do not call Minion.CheckDBCheckTable to perform integrity checks.

The Minion.CheckDBMaster procedure makes all the decisions on which databases and tables to process, and what order they should be in. It's certainly possible to call Minion.CheckDBCheckTable manually, to process an individual table, but we instead recommend using the Minion.CheckDBMaster procedure (and just include the single table using the @Tables parameter). First, it unifies your code, and therefore minimizes your effort. By calling the same procedure every time you reduce your learning curve and cut down on mistakes. Second, future functionality may move to the Minion.CheckDBMaster procedure; if you get used to using Minion.CheckDBMaster now, then things will always work as intended.

Name	**Type**	**Description**
@DBName	nvarchar	Database name.
@Schemas	varchar	Limits maintennce to just a single schema, or list of schemas. See the @Schema entry for Minion.CheckDBMaster for more information.
@Tables	varchar	Limits maintennce to just a single table, or list of tables.
@StmtOnly	bit	Only generate CheckDB statements, instead of running them.
@PrepOnly	bit	Only determines which tables require CheckTable at this time, and saves this information to a table (Minion.CheckDBCheckTableThreadQueue). This feature is used automatically (and internally) for use multi-threaded CheckTable work. **Note:** This *can* also be used by users. For example, if you wanted to edit the Minion.CheckDBCheckTableThreadQueue table after the list off tables was added to it, but before the actual CheckTable run.
@RunPrepped	bit	If you've run Minion.CheckDBCheckTable with @PrepOnly=1 (and so the list of tables to be checked is already in the Minion. Minion.CheckDBCheckTableThreadQueue table), then you can use this option to actually run CheckTable operations.

		This feature is used automatically (and internally) for use multi-threaded CheckTable work. **Note:** This can also be used by users. See the “Note” in the @PrepOnly entry above.
@ExecutionDateTime	datetime	Date and time the CheckTable took place. If this stored procedure was called by Minion.CheckDBMaster, @ExecutionDateTime will be passed in here, so this operation is included as part of the entire (multi-table or multi-database) CheckTable operation.
@Thread	tinyint	For internal use only.
@Debug	bit	Enable logging of special data to the debug tables. For more information, see “Minion.CheckDBDebug”, “Minion.CheckDBDebugSnapshotCreate”, and “Minion.CheckDBDebugSnapshotThreads”.

Example execution:

```
-- Generate DBCC CHECKTABLE statements for database DB2, as applicable:
EXEC [Minion].[CheckDBCheckTable]
	@DBName = 'DB2',
	@StmtOnly = 1;
```

Example execution:

```
-- Generate DBCC CHECKTABLE statements for database DB1:
EXEC [Minion].[CheckDBCheckTable]
	@DBName = 'DB1',
	@StmtOnly = 1;
```

Minion.CheckDBCheckTableThreadRunner

This procedure runs the CheckTable threads.

This procedure is for internal use only.

Minion.CheckDBMaster

The Minion.CheckDBMaster procedure is the central procedure of Minion CheckDB. It uses the parameter and/or table data to make all the decisions on which databases to run CheckDB, and what order they should be in. This stored procedure calls either the Minion.CheckDB stored procedure, or the Minion.CheckDBCheckTable.

IMPORTANT: We HIGHLY recommend using Minion.CheckDBMaster for all of your integrity check operations, even when operating on a single database. Do not call Minion.CheckDB to perform integrity checks.

In addition, Minion.CheckDBMaster performs extensive logging, runs configured pre- and postcode, enables and disables the status monitor job (which updates log files for Live Insight, providing percent complete for each CheckDB), and more.

In short, Minion.CheckDBMaster decides on, runs, or causes to run every feature in Minion CheckDB.

Name	**Type**	**Description**
@DBType	varchar	The type of database. Valid inputs: **System** **User**
@OpName	varchar	Operation name. Default value is CHECKDB. The AUTO option allows Minion CheckDB to choose the appropriate operation per database, based on settings in the Minion.CheckDBSettingsAutoThresholds table. For more information on this, see the section titled "How to: Configure Minion CheckDB Dynamic Thresholds". Using *NULL* allows the system to choose the appropriate settings from the Minion.CheckDBSettingsServer table. Valid inputs: **CHECKDB** **CHECKTABLE** **CHECKALLOC** **AUTO** *NULL*
@StmtOnly	bit	Only generate CheckDB statements, instead of running them.
@ReadOnly	tinyint	Readonly option; this decides whether or not to include ReadOnly databases in the operation, or to perform operations on *only* ReadOnly databases. A value of 1 includes ReadOnly databases; 2 excludes ReadOnly databases; and 3 only includes ReadOnly databases. Valid values: 1 2

		3
@Schemas	varchar	This allows you to limit the operations to just a single schema, or list of schemas. Without further filtering (using the Tables column), all objects in this/these schemas will be targeted. Note that this places no limit on the database. For example: If you specify Schemas='Minion', and you have a "Minion" schema in multiple databases, MC will operate on the Minion schema across any database that has it. @Schemas = *NULL* will run maintenance on all schemas. @Schema will also accept a comma-delimited list of database names and LIKE expressions (e.g., 'Minion, Test%, Bravo').
@Tables	varchar	This allows you to limit the operations to just a single table, or list of tables. Note that this places no limit on the database. For example: If you specify Tables='Minion', and you have a "Minion" table in multiple databases, MC will operate on the Minion table across any database that has it. @Tables = *NULL* will run maintenance on all tables (unless otherwise filtered, e.g., by the @Schemas parameter). @Table will also accept a comma-delimited list of database names and LIKE expressions (e.g., 'Minion, Test%, Bravo').
@Include	Varchar	Use @Include to run CheckDB on a specific list of databases, or databases that match a LIKE expression. Alternately, set @Include='All' or @Include=*NULL* to run maintenance on all databases. If, during the last backup run, there were backups that failed, and you need to back them up now, just call this procedure with @Include = 'Missing'. The stored procedure will search the log for the backups that failed in the previous batch (for a given BackupType and DBType), and back them up now. Note that the BackupType and DBType must match the errored out backups.

		Valid inputs: *NULL* Regex Missing *<comma-separated list of DBs including wildcard searches containing '%'>*
@Exclude	varchar	Use @Exclude to skip backups for a specific list of databases, or databases that match a LIKE expression. Examples of valid inputs include: DBname DBName1, DBname2, etc. DBName%, YourDatabase, Archive%
@NumConcurrentProcesses	tinyint	The number of concurrent processes to use for this operation. This is the number of databases that will be processed simultaneously (CheckDB or CheckTable). Default value is 3.
@DBInternalThreads	tinyint	If CheckTable, this is the number of tables that will be processed in parallel.
@TestDateTime	datetime	A "what if" parameter that allows you to see what schedule will be used at a certain date and time. This returns the settings from Minion.CheckDBSettingsServer that would be used at that date and time, and a list of databases (and their order) to be included in the batch. IMPORTANT: To ONLY run the test, and not the actual operations, run with @StmtOnly = 1. For example: **EXEC Minion.CheckDBMaster @StmtOnly = 1, @TestDateTime = '2016-09-28 18:00';**
@TimeLimitInMins	int	The time limit to impose on this opertion, in minutes.
@FailJobOnError	bit	Cause the job to fail if an error is encountered. If an error is encountered, the rest of the batch will complete before the job is marked failed.
@FailJobOnWarning	bit	Cause the job to fail if a warning is encountered. If a warning is encountered, the rest of the batch will complete before the job is marked failed.
@Debug	bit	Enable logging of special data to the debug tables.

		For more information, see "Minion.CheckDBDebug" and "Minion.CheckDBDebugLogDetails".

Example execution:

```
-- Run database integrity check operations for all databases, allow 3 concurrent processes:
EXEC [Minion].[CheckDBMaster]
   @DBType = 'User',
   @OpName = 'AUTO',
   @StmtOnly = 0,
   @NumConcurrentProcesses = 3;
```

Example execution:

```
-- Run DBCC CHECKDB for all user databases named like Minion%, allow 2 concurrent processes:
EXEC [Minion].[CheckDBMaster]
   @DBType = 'User',
   @OpName = 'CHECKDB',
   @StmtOnly = 0,
   @Include = 'Minion%',
   @NumConcurrentProcesses = 2;
```

Example execution:

```
-- Run DBCC CHECKDB for all user databases EXCEPT "TestRun" and those named like %Archive:
EXEC [Minion].[CheckDBMaster]
   @DBType = 'User',
   @OpName = 'CHECKDB',
   @StmtOnly = 0,
   @Exclude = '%Archive, TestRun';
```

Example execution:

```
-- Generate database integrity statements for all system databases:
EXEC [Minion].[CheckDBMaster]
   @DBType = 'System',
   @OpName = 'AUTO',
   @StmtOnly = 1;
```

Minion.CheckDBRemoteRunner

This procedure creates the remote job for remote CHECKDB mode, and runs it.

This procedure is for internal use only.

Minion.CheckDBRotationLimiter

This procedure manages which databases and tables have already been run within the rotation period, and makes sure that only the desired databases are run. It maintains a list of the databases or tables that have run during the current rotation period.

This procedure is for internal use only.

For more information, see "About: Rotational Scheduling" and "How to: Configure Rotational Scheduling".

Minion.CheckDBSnapshotDirCreate

This procedure is for internal use only.

Minion.CheckDBSnapshotGet

Creates the statement to create custom snapshots for CheckDB or CheckTable.

This procedure is meant for internal use.

Future solutions (or MinionWare support) may include instructions on how to use this procedure for troubleshooting or custom scenarios.

Note: SQL Server 2016 and earlier versions only allow custom snapshots for Enterprise edition. SQL Server 2016 SP1 allow custom snapshots in any edition.

For more information, see "How to: Configure Custom Snapshots".

Name	Type	Description
@DBName	varchar	Database name.
@OpName	varchar	Operation name.

Minion.CheckDBStatusMonitor

This procedure updates the status of running operations, in Minion.CheckDBLogDetails. It is automatically started at the start of an integrity check operation, and automatically stopped at the end of the last operation.

Name	Type	Description
@Interval	varchar	The amount of time to wait before updating the table again. Default is '00:00:05' (5 seconds).

Minion.CheckDBThreadCreator

This procedure is for internal use only.

Minion.CloneSettings

This procedure allows you to generate an insert statement for a table, based on a particular row in that table.

We made this procedure flexible: you can enter in the name of any Minion table, and a row ID, and it will generate the insert statement for you.

Note that this function is shared between Minion modules.

WARNING: This generates a clone of an existing row as an INSERT statement. Before you run that insert, be sure to change key identifying information - e.g., the DBName - before you run the INSERT statement; you would not want to insert a completely identical row.

Name	Type	Description
@TableName	Varchar	The name of the table to generate an insert statement for. **Note:** This can be in the format "Minion.CheckDBSettingsDB" or just " CheckDBSettingsDB".
@ID	Int	The ID number of the row you'd like to clone. See the discussion below.
@WithTrans	Bit	Include “BEGIN TRANSACTION” and “ROLLBACK TRANSACTION” clauses around the insert statement, for safety.

Discussion:

Because of the way we have writte Minion CheckDB, you may often need to insert a row that is nearly identical to an existing row. If you want to change just one setting, you still have to fill out 40 columns. For example, you may wish to insert a row to Minion.CheckDBSettingsDB that is only different from the MinionDefault rows in two respects (e.g., DBName and GroupOrder).

We created Minion.CloneSettings to easily duplicate any existing row in any table. This "helper" procedure lets you pass in the name off the table you would like to insert to, and the ID of the row you want to model the new row off of. The procedure returns an insert statement so you can change the one or two values you want.

Discussion: Identity columns

If the table in question has an IDENTITY column, regardless of that column’s name, Minion.CloneSettings will be able to use it to select your chosen row. For example, let’s say that the IDENTITY column of Table1 is *ObjectID*, and that you call Minion.CloneSettings with @ID = 2. The procedure will identify that column and return an INSERT statement that contains the values from the row where ObjectID = 2.

Minion.DBMaintDBSettingsGet

Determines which settings from the **Minion.CheckDBSettingsDB** table apply for a given database and operation, at a given time. This procedure is generally for internal use, but you can use it manually as needed.

Note that this function is shared between Minion modules.

Also: To determine the settings from the **Minion.CheckDBSettingsServer** table will be used, use the **Minion.CheckDBMaster** procedure with @StmtOnly = 1, and @TestDateTime populated.

Name	**Type**	**Description**
@Module	varchar	The name of the Minion module. Valid inputs include: CHECKDB
@DBName	varchar	Database name.
@OpName	varchar	Operation name. Valid inputs: NULL **AUTO** **CHECKDB** **CHECKTABLE**
@SettingID	int	An output parameter that provides the ID of the row in Minion.CheckDBSettingsDB that applies to the module, database, operation, and time provided.
@TestDateTime	datetime	The date and time of the operation. Automatic operations provide the present date and time to get the applicable settings. If you're running Minion.DBMaintDBSettingsGet by hand, you can pass in any date and time as a "what if" to see what settings would be used at that time.

Example execution:

```
DECLARE @SettingID INT;
EXEC Minion.DBMaintDBSettingsGet
        @Module = 'CHECKDB',
        @DBName = 'Demo',
        @OpName = 'CHECKDB',
        @SettingID = @SettingID OUTPUT,
        @TestDateTime = '2016-10-22 16:00:00';
SELECT  @SettingID AS SettingID;
```

Minion.DBMaintDBSizeGet

Determines the size of the database passed in through @DBName, as determined by the ThresholdType and ThresholdValue fields in the Minion.CheckDBSettingsAutoThresholds table.

Note that this function is shared between Minion modules.

Name	Type	Description
@Module	varchar	The name of the Minion module. Valid inputs include: CHECKDB
@OpName	varchar	An output parameter that provides the operation name (e.g., CHECKDB).
@DBName	varchar	Database name.
@DBSize	decimal	An output parameter that provides the database size, as measured in GB.

Minion.DBMaintServiceCheck

This procedure checks the SQL Agent run status and returns the result in an output parameter.

Note that this function is shared between Minion modules.

Name	Type	Description
@ServiceStatus	bit	Output column that returns the state of the SQL Agent service: running (1), or not running (0).

Example:

```
DECLARE @ServiceStatus BIT;
EXEC Minion.DBMaintServiceCheck @ServiceStatus = @ServiceStatus OUTPUT
SELECT  @ServiceStatus;
```

Minion.DBMaintStatusMonitorONOff

This procedure is used to turn the status monitor job on or off.

NOTE: This procedure is used internally; it is not meant to be called manually.

Note that this function is shared between Minion modules.

Functions Detail

Minion.DBMaintSQLInfoGet

This function returns a table with information about the current server instance: VersionRaw, Version, Edition, OnlineEdition, Instance, InstanceName, and ServerAndInstance.

Note that this function is shared between Minion modules.

Example execution:

```
SELECT  VersionRaw
      , Version
      , Edition
```

```
        , OnlineEdition
        , Instance
        , InstanceName
        , ServerAndInstance
    FROM    Minion.DBMaintSQLInfoGet();
```

Overview of Jobs

When you install Minion CheckDB, it creates two new jobs:

- ***MinionCheckDB-AUTO*** – Runs every hour. This job consults the **Minion.CheckDBSettingsServer** table to determine what, if any, integrity check operations are slated to run at that time. By default, the Minion.CheckDBSettingsServer table is configured with Saturday full CheckDBs, daily weekday differential CheckDBs, and log CheckDBs every half hour.
- ***MinionCheckDBStatusMonitor*** – Monitor job that updates the log tables with “CheckDB percentage complete” data. By default, this job runs continuously, updating every 10 seconds, while a Minion CheckDB operation is running.

"About" Topics

About: Minion CheckDB Operations

A baseline run of Minion CheckDB operates like this:

1. **"Master" SP:** The job MinionCheckDB-AUTO runs and calls the **Minion.CheckDBMaster** procedure, without parameters.
2. **Schedule from SettingsServer:** Minion.CheckDBMaster then consults the **Minion.CheckDBSettingsServer** table to determine what operation is currently scheduled. Let's say this run of the job sees a "User CHECKDB" operation is in order.
3. **Settings from SettingsDB and SettingsTable:** The Master procedure then checks the table **Minion.CheckDBSettingsDB** to work out which databases (if any) should be excluded from the run, and what settings to apply. (Note that a CHECKTABLE operation consults both the Minion.CheckDBSettingsDB table *and* Minion.CheckDBSettingsTable).

That's a default, no-special-configurations run of Minion CheckDB. Other options configurable in the product add additional steps, but these base steps remain the same.

Those other options include (but of course, may not be limited to):

- **Dynamic thresholds**, which let MC determine whether to run a CheckDB or a CheckTable (based on your configured criteria). Related table: Minion.CheckDBSettingsAutoThresholds.
- **Remote CheckDB**, which allows you to configure CheckDB operations on remote servers. Related table: Minion.CheckDBSettings.RemoteThresholds.
- **CheckTable rotations ("rotational scheduling")**, which allow you to define a rotation scenario for your operations. Related table: Minion.CheckDBSettingsRotation.
- **CheckDB rotations ("rotational scheduling")**, which allow you to define a rotation scenario for your operations. Related table: Minion.CheckDBSettingsRotation.
- **Custom snapshots**, which allow you to set the location (and, for CheckTable operations, the snapshot frequency) of custom snapshots. Related table: Minion.CheckDBSettingsSnapshot and Minion.CheckDBSnapshotPath.
- **Inline Tokens**, which allows you use defined patterns to create dynamic names. Related table: Minion.DBMaintInlineTokens.

CHECKTABLE operations

In step 3 above, we noted that a CHECKTABLE operation consults both the Minion.CheckDBSettingsDB table *and* Minion.CheckDBSettingsTable.

For CHECKTABLE operations, Minion CheckDB uses settings as appropriate from Minion.CheckDBSettingsDB where OpName='CHECKTABLE'. Then, if there are table-level settings in Minion.CheckDBSettingsTable, those settings take precedence for those tables.

For example: In this example, we have the following settings in Minion.CheckDBSettingsDB:

ID	DBName	OpLevel	OpName	Exclude	…	IsActive
1	MinionDefault	DB	CHECKDB	0	…	1
2	MinionDefault	DB	CHECKTABLE	0	…	1
3	DB1	DB	CHECKDB	0	…	1
4	DB1	DB	CHECKTABLE	0	…	1

And the following settings in Minion.CheckDBSettingsTable:

ID	DBName	SchemaName	TableName	Exclude	…	IsActive
1	DB1	dbo	MyTable	0	…	1
2	DB1	dbo	OtherTable	1	…	1
3	DB2	dbo	ASDF	0	…	1

With these settings in place:

- A CHECKDB run will use settings from Minion.CheckDBSettingsDB, either row 3 (for database DB1) or row 1 (for any other database).
- A CHECKTABLE run for DB5 will use settings from Minion.CheckDBSettingsDB, row 2.
- A CHECKTABLE run for DB1 will use settings from Minion.CheckDBSettingsDB, row 3; EXCEPT for tables "MyTable" and "OtherTable".
- A CHECKTABLE run for DB2 will use settings from Minion.CheckDBSettingsDB, row 3; EXCEPT for table "ASDF".

Complex scenarios

It's useful for you to know how different features of Minion CheckDB play together. In this section, we'll look at the logical ordering and interplay between features. This section will be expanded in future updates to the documentation.

To demonstrate how it all fits together, let's say you have a very complex scenario for DB1, with the following settings configured:

- Weekly AUTO schedule
- Auto Threshold set at 100 GB
- Remote Threshold set at 50 GB
- Custom snapshot
- CheckTable rotation
- CheckDB rotation

Database is under the auto threshold and remote threshold: Right now, DB1 is 40GB. So, the logical order of operations for this scenario is:

1. A run of the MC job determines that it's time to run the AUTO schedule.

2. It checks the DB size, and finds it under the auto threshold of 100; so, it's assigned a CheckDB operation.
3. DB1 is also under the remote threshold, so the operation will be local.
4. DB1 is on an edition of SQL Server that supports custom snapshots, so the custom snapshot settings apply.
5. MC figures out which databases to run next in the CheckDB rotation, and runs them. (Note that as this is a CheckDB operation, CheckTable rotation isn't in play.)

Database is under the auto threshold, but over the remote threshold: DB1 has grown to 65 GB. The logical order of operations for this scenario is:

1. A run of the MC job determines that it's time to run the AUTO schedule.
2. It checks the DB size, and finds it under the auto threshold of 100; so, it's assigned a CheckDB operation.
3. DB1 is OVER the remote threshold, so the operation will be remote.
4. The remote server supports custom snapshots, AND remote CheckDB is set to "Disconnected" mode, AND custom snapshots are configured on the remote server. So the custom snapshot settings apply there. (For more on Disconnected and Connected modes, see the discussion below, "Minion.CheckDBSettingsDB", "About: Remote CheckDB", and "How to: Set up CheckDB on a Remote Server".)
5. MC continues with the next database in the CheckDB rotation, and runs it using the same decision making process. (Note that as this is a CheckDB operation, CheckTable rotation isn't in play.)

Database is over both the auto threshold and the remote threshold: DB1 has grown to 110 GB. The logical order of operations for this scenario is:

1. A run of the MC job determines that it's time to run the AUTO schedule.
2. It checks the DB size, and finds it OVER the auto threshold of 100; so, it's assigned a CheckTable operation. (CheckTable operations are not eligible for remote integrity checks.)
3. The local server supports custom snapshots, AND custom snapshots are configured on the server for CheckTable operations. So the custom snapshot settings apply here.
4. MC determines which tables to run next in the CheckTable rotation, and runs them.
5. When DB1 is completed, MC continues with the next database in the rotation (whether that's the next database in the CheckTable rotation, or the next database which might have either CheckDB or CheckTable), and runs it using the same decision making process.

Again, this example isn't a recommendation, but simply a demonstration of how different features of MC work around one other.

Discussion: Disconnected mode. In the second scenario above, remote CheckDB was set to Disconnected mode, and so the remote server's custom snapshot settings came into play. Connected mode, however, just runs the DBCC CheckDB commands generated from the local server, on the remote server. Connected mode does not consult the custom snapshot settings at all; by default, it will use an internal snapshot. However, you *could* force a "custom snapshot" in connected mode, by (1) restoring the database in question to the remote server (outside of the MC process; you'd have to use RemoteRestoreMode=NONE), (2) creating your own custom snapshot (outside of the MC process), and (3) pointing the PreferredDBName at that snapshot. And because PreferredDBName accepts inline tokens and LIKE expressions, you could potentially

About: Feature Compatibility

Most Minion CheckDB features apply to both DBCC CheckDB operations and DBCC CheckTable operations, but there are exceptions:

	CheckDB Operations	**CheckTable Operations**
Dynamic Thresholds	YES	YES
Remote CheckDB	YES	**No**
Dynamic Remote CheckDB	YES	**No**
Custom Snapshots	YES	YES
Custom Dynamic Snapshots	**No**	YES
Multithreading operations	YES	YES
Rotational Scheduling	YES	YES

Additionally, some features are cross-compabitble with one another, and some are not, and quite a lot of them have footnotes:

	Dynamic Thresholds	**Remote CheckDB**	**Dynamic Remote CheckDB**	**Custom Snapshots**	**Custom Dynamic Snapshots**	**Multithreading operations**	**Rotational Scheduling**
Dynamic Thresholds	-	Yes[1]	Yes	Yes	Yes[2]	Yes	Yes
Remote CheckDB	Yes[1]	-	No[3]	Yes[4]	No	Yes	Yes
Dynamic Remote CheckDB	Yes	No[3]	-	Yes[4,5]	No	Yes	Yes
Custom Snapshots	Yes	Yes[4]	Yes[4,5]	-	Yes	Yes	Yes
Custom Dynamic Snapshots	Yes[2]	No	No	Yes	-	No	Yes
Multithreading operations	Yes	Yes	Yes	Yes	No	-	Yes
Rotational Scheduling	Yes	Yes	Yes	Yes	Yes	Yes	-

[1]Dynamic Thresholds decides whether to do a CheckDB or a CheckTable, based on thresholds you configure. Remote CheckDB cannot perform a CheckTable operation. So technically speaking, you *can* set both of these options up, and if MC chooses CheckTable, it won't consult the remote CheckDB settings; it will simply do a local CheckTable.

[2]Dynamic Thresholds decides whether to do a CheckDB or a CheckTable, based on thresholds you configure. Custom Dynamic Snapshots are not available for CheckDB. So, if MC chooses CheckDB, it won't consult the dynamic snapshot; it will simply use an internal snapshot.

[3]To enable dynamic remote snapshots, "remote snapshots" (IsRemote) must be disabled. For more information, see "Minion.CheckDBSettingsDB".

[4]For more information, see "**Discussion: Disconnected mode**" in the "About: Minion CheckDB Operations" section.

[5]If MC decides to perform a local operation, then the custom snapshot settings are back in play.

About: Scheduling

Minion CheckDB offers you a choice of scheduling options:

- You can use the Minion.CheckDBSettingsServer table to configure flexible scheduling scenarios;
- Or, you can use the traditional approach of one job per integrity check schedule;
- Or, you can use a hybrid approach that employs a bit of both options.

For more information, see "Changing Schedules" in the Quick Start section, and "How To: Change Schedules".

Table based scheduling

When Minion CheckDB is installed, it uses a single job (MinionCheckDB-AUTO) to run the stored procedure Minion.CheckDBMaster with no parameters, once every hour.

When the Minion.CheckDBMaster procedure runs without parameters, it uses the Minion.CheckDBSettingsServer table to determine its runtime parameters (including the schedule of jobs per database type). This is how MC operates by default, to allow for the most flexible scheduling with as few jobs as possible.

Table based scheduling presents multiple advantages:

- **A single job** – Multiple jobs are, to put it simply, a pain. They're a pain to update and slow to manage, as compared with using update and insert statements on a table.
- **Fast, repeatable configuration** – Keeping your schedules in a table saves loads of time, because you can enable and disable schedules, change frequency and time range, etc. all with an update statements. This also makes standardization easier: write one script to alter your schedules, and run it across all Minion CheckDB instances (instead of changing dozens or hundreds of jobs).
- **Mass updates across instances** – With a simple PowerShell script, you can take that same script and run it across *hundreds* of SQL Server instances, standardizing your entire enterprise all at once.
- **Transparent scheduling** – Multiple jobs tend to obscure the maintenance scenario, because each piece of the configuration is displayed in separate windows. Table based scheduling allows you to see all aspects of the schedule in one place, easily and clearly.

- **Boundless flexibility** – Table based scheduling provides an amazing degree of flexibility that would be very troublesome to implement with multiple jobs. With one job, you can schedule all of the following:
    - System DBCC CheckDBs three days a week.
    - User DBCC CheckDBs on weekend days and Wednesday.
    - User DBCC CheckTables twice daily for specific schemas.
    - …and each of these can also use Dynamic Thresholds, which can also be slated for different sizes, applicable at different times and days of the week and year.

Parameter Based Scheduling

Other SQL Server native integrity check solutions traditionally use one job per schedule. Typically and at a minimum, that means one job for system database CheckDBs, and another job for user database CheckDBs.

Note: Whether you use table based or parameter based scheduling, we *highly* recommend always using the Minion.CheckDBMaster stored procedure to run integrity check operations. While it is possible to use the Minion.CheckDB procedure or Minion.CheckDBCheckTable to execute integrity checks, doing so will bypass much of the configuration and logging benefits that Minion CheckDB was designed to provide.

Discussion: Hierarchy and Precedence

There is an order of precedence to the schedule settings in Minion.CheckDBSettingsServer, from least frequent (First/LastOfYear) to most frequent (daily); **the least frequent setting, when it applies, takes precedence over all others.** For example, if today is the first of the year, and there is a FirstOfYear setting, that's the one it runs.

The full list, from most frequent, to least frequent (and therefore of highest precedence), is:

1. Daily
2. Weekday / Weekend
3. Monday / Tuesday / Wednesday / Thursday / Friday / Saturday / Sunday
4. FirstOfMonth / LastOfMonth
5. FirstOfYear / LastOfYear

Note that the least frequent "Day" settings – FirstOfYear, LastOfYear, FirstOfMonth, LastOfMonth – only apply to user databases, not to system databases. System databases may have "Day" set to a day of the week (e.g., Tuesday), WeekDay, WeekEnd, Daily, or *NULL* (which is equivalent to "Daily").

Discussion: Overlapping Schedules, and MaxForTimeframe

The Minion.CheckDBSettingsServer table allows you to have integrity check schedule settings that overlap.

Note: We recommend against overlapping schedules, as there is no guarantee of precedence. If you have a day and time window scheduled for DB1 CheckDB, for example, and an overlapping window for DB1 CheckTable, there is no set precedence for which one will run.

Use adjacent day and time windows for individual databases or sets of databases. For example, we could perform DBCC CheckTable operations on specific DB1 tables every 6 hours from 1am to 7pm, and then run a full DBCC CheckDB every night at 11pm. For this scenario, we would:

- Insert 1 row for the DB1 CheckTable, with a MaxForTimeframe value of 4 and FrequencyMins = 360 (6 hours). Set BeginTime = 01:00:00, and EndTime = 19:00:00.
- Insert one row for the DB1 CheckDB, with a MaxForTimeframe value of 1. Set BeginTime = 23:00:00, and EndTime = 23:59:00.

The sequence of job executions then goes like this:

1. The MinionCheckDB-AUTO job kicks off at 1:00 am.
2. MC determines that a CheckTable operation is slated for DB1 tables, and executes the CheckTable operation.
3. MC also increments the CheckTable row's CurrentNumCheckDBs for that timeframe.
4. The MinionCheckDB-AUTO job continues to run hourly until 7am, when MC sees that it's time for another CheckTable run (based on the MaxForTimeframe field).
5. Steps 2-4 repeat, CheckTable running again at 1pm and 7pm.
6. At 11pm, MC sees that the CheckDB is due, runs it, and increments the CheckDB row's CurrentNumCheckDBs.

Discussion: Using FrequencyMins

The FrequencyMins column allows you to run the "MinionCheckDB-AUTO" SQL Agent job as often as you like, but to space operations out by a set interval. Let's say that the job runs every hour, but DBCC CheckDB (PHYSICAL_ONLY) for DB1 should only run every 12 hours. Just set FrequencyMins = 720 for the CheckDB/DB1 row.

About: Dynamic Thresholds

Minion CheckDB allows you to automate whether databases get a DBCC CheckDB operation, or a DBCC CheckTable operation. Configure dynamic thresholds in the Minion.CheckDBSettingsAutoThresholds table. These settings only apply to runs of the stored procedure Minion.CheckDBMaster where OpName = 'Auto' in Minion.CheckDBSettingsDB (or, for a manual run, where @OpName = 'Auto').

The default entry that comes installed with Minion CheckDB sets a threshold by size, at 100 GB. What this means is that by default – for Minion.CheckDBMaster runs with @OpName = 'Auto', **any database under 100 GB gets a CheckDB operation instead of a CheckTable operation**.

Note: As outlined in the "Configuration Settings Hierarchy" section, more specific settings in a table take precedence over less specific settings. So if you insert a database-specific row for DB1 to this table, that row will be used for DB1 (instead of the "MinionDefault" row).

For more information, see "How to: Configure Dynamic Thresholds".

About: Remote CheckDB

Minion CheckDB provides remote integrity checks, where a database may be restored to another instance for DBCC CheckDB operations.

Note: Remote operations only apply to DBCC CheckDB operations. Minion CheckDB does not support remote CheckTable. For more information, see "About: Feature Compatibility".

Requirements

IMPORTANT: Remote CheckDB has a few requirements:

- The source server's **SQL Agent service account must have rights** on the remote server, including permissions to create jobs. And of course, the two servers must be able to "see" each other.
- You must either be using **Minion Backup 1.3** on the source server; or there must be an external process that restores the database(s) in question to the remote server for CheckDB operations (RemoteRestoreMode=NONE). This also means that the remote server must be a **compatible version of SQL Server**, that the database can restore to.
- **The remote server must have Minion CheckDB installed in the same database** as the local ("source") server. So, if MC is installed in master on the source server, the remote server must have MC installed in master, too. (Officially speaking, for Connected mode, you only need the Minion.CheckDBResult table.)
- Remote CheckDB currently only supports Windows authentication.

For full instructions on configuring remote CheckDB, see "How to: Set up CheckDB on a Remote Server".

Remote CheckDB modes: Connected vs Disconnected

Remote CheckDB has the option of Connected mode or Disconnected mode:

Connected mode is equivalent to running DBCC CheckDB commands from SQL Server Management Studio on Svr1, against Svr2. Connected mode maintains the connection throughout the operation(s). It does **not** require a full Minion CheckDB installation on the remote server. Connected mode is good for when you don't have permissions from the remote server back to the primary server. Connected mode has fewer moving parts internally than Disconnected mode.

Disconnected mode requires a full Minion CheckDB installation on the remote server. Disconnected mode requires the most permissions, but is also the more robust option; it has higher tolerance for things like network fluctuations.

Remote Restore Modes

As you will see in the "How to: Set up CheckDB on a Remote Server" section, remote CheckDB is configured in the Minion.CheckDBSettingsDB table. The RemoteRestoreMode field has three options:

NONE – MC performs no database restore to the remote server. If you already have a process in place for restoring databases to a remote server – whether it's a third party backup and restore proess, home grown, detatch/attach, or anything else – "NONE" allows MC to fold into the existing scenario easily. Note that with the benefit of Inline Tokens, the remote database could be named the same as the source database, or the name could be generated in some rolling process with (for example) the date or a number, like DB1.20170101. In that case, we can set PreferredDBName = '%DBName%.%Date%', or the less specific '%DBName%.%'. **When in doubt, Minion CheckDB will select the most recently created database that fits the naming scheme.**

LastMinionBackup – Restores the last backup from Minion Backup to the remote server, then runs CheckDB against it.

NewMinionBackup – Takes a new backup using Minion Backup, restores it to the remote server, then runs CheckDB against it.

Dynamic Remote CheckDB

You can also define thresholds for remote CheckDB, so the operation will run remotely only if it is above that threshold.

To turn on this feature, the Minion.CheckDBSettingsDB column IsRemote must be set to 0. While this may seem counterintuitive, IsRemote = 1 turns on remote CheckDB for *all* databases (that the given row applies to). If you wish to handle remote operations dynamically, based on database size, set IsRemote = 0 – meaning, "I want operations to be local *unless* a database crosses the threshold".

For full instructions on configuring remote CheckDB, see "How to: Set up CheckDB on a Remote Server".

About: Custom Snapshots

When you run DBCC CheckDB or DBCC CheckTable, behind the scenes, SQL Server creates a snapshot of the database to run the operation against. If your version of SQL Server supports it, you can also choose to create a custom snapshot and configure where its files are created.

Note: SQL Server 2016 and earlier versions only allow custom snapshots for Enterprise edition. SQL Server 2016 SP1 allow custom snapshots in any edition.

You might want to create a custom snapshot if an operation takes long enough that the internal snapshot would grow too large (and risk filling up the drive), which would stop the operation. You can also – for CheckTable operations only – create and recreate "Custom Dynamic Snapshots" (see the following section) at timed intervals, to prevent the snapshot file from getting too large.

Minion CheckDB provides several options for custom snapshots:

- Assign a different drive for each file, or put them all onto a single drive.
- Change the location for just one file.
- Delete the snapshot after your operation is done, or keep it to fold it into your normal snapshot rotation.

Note: If CustomSnapshot is enabled and your version of SQL Server doesn't support it, that integrity check operation will complete using the default internal snapshot. For more information, see the "Custom snapshots fail" section under Troubleshooting.

IMPORTANT: SQL Server does not allow you to specify log files or filestream files in a CREATE SNAPSHOT statement. The MSDN article "FILESTREAM Compatibility with Other SQL Server Features" (https://msdn.microsoft.com/en-us/library/bb895334.aspx#DatabaseSnapshot) provides more information: "When you are using FILESTREAM, you can create database snapshots of standard (non-FILESTREAM) filegroups. The FILESTREAM filegroups are marked as offline for those database snapshots."

For more information, see:

- the section "About: Custom Snapshots"
- the video "Custom Snapshot Basics": https://youtu.be/0PVFXm6KDr0
- the video "Custom Snapshot for CheckTable": https://youtu.be/1wda8fYBVk4
- the video "Custom Snapshot for Multiple Files": https://youtu.be/Le43dzFBOVM

Custom Dynamic Snapshots

Custom snapshots allow you to determine where the snapshot file(s) will be located. For CheckTable, custom snapshots allow you both to set the file locations, *and* to drop and recreate the snapshot every few minutes (which we call "custom dynamic snapshots").

IMPORTANT: Custom dynamic snapshots for CheckTable are only available for *single-threaded operations*. This means that you must set DBInternalThreads in Minion.CheckDBSettingsDB, and DBInternalThreads in Minion.CheckDBSettingsServer, to 1 for custom dynamic snapshots.

Note: The only difference between custom snapshots for CheckTable, and "rotating" custom dynamic snapshots for CheckTable – those that drop and recreate every few minutes – is that a rotating snapshot has "SnapshotRetMins" set to a value greater than zero.

Important notes:

- **Minion.CheckDBSettingsSnapshot (DeleteFinalSnapshot):** It's a good idea to delete the snapshot after your operation is done, but it's not necessary. You might want to fold it into your normal snapshot rotation.
- **Minion.CheckDBSettingsSnapshot (SnapshotRetMins):** You can set up dynamic snapshots that are dropped and recreated every N minutes, for CheckTable oeprations. (The SnapshotRetMins column does not apply to CheckDB operations, as you can only drop and recreate the snapshot for CheckTable.)
- **Hierarchy rules:** The same rules apply in both **Minion.CheckDBSettingsSnapshot** and **Minion.CheckDBSnapshotPath** for database overrides: Make sure you have one row for CheckDB and one for CheckTable for MinionDefault, and CheckDB/CheckTable rows for each individual database you configure in these tables.

- **Logging:** The Minion.CheckDBSnapshotLog table shows you all the files and the statement used to create the snapshot. This is mostly for troubleshooting, but it also has a column that shows you the maximum size that each of the files reached. This is for planning; you can make sure that any given disk will have enough space. You're welcome.

Notes for troubleshooting:

- This table is where we store the "create database" snapshot command for custom snapshots.
- You can read from this table to make sure the files are being created, that they're being created in the right location, and with the correct name, and so on.
- One of the last columns in this table is the **MaxSizeInMB** column, which shows you the size of the snapshot. That can help you plan the size of the drives you need to put the snapshots on.

Note: If you run a CheckDB operation from SvrA remotely (in disconnected mode) on SvrB, *and* if SvrB has custom snapshots configured, then this table will hold records of the custom snapshot file(s) in this table on SvrB. For more information, see "About: Remote CheckDB", "How to: Set up CheckDB on a Remote Server", and the "Complex Scenarios" section under "About: Minion CheckDB Operations".

About: Inline Tokens

Minion CheckDB 1.0 and MinionBackup 1.3 introduce a new feature to the Minion suite – Inline Tokens. Inline Tokens allow you use defined patterns to create dynamic names and paths. For example, MC comes with the predefined Inline Token "Server" and "DBName".

In this version of MC, inline tokens are accepted for remote CheckDB operations. Specifically, the PreferredDBName and RemoteJobName in the Minion.CheckDBSettingsDB table:

```
UPDATE Minion.CheckDBSettingsDB
SET    PreferredDBName = '%Server%_%DBName%',
          RemoteJobName = 'MinionCheckDB_%Server%_%DBName%';
```

From then on, the preferred database name on server "RemoteServer" for database "DB1" will be created as "RemoteServer_DB1", and the job created will be named "MinionCheckDB_RemoteServer_DB1".

MC recognizes %Server% and %DBName% as Inline Tokens, and refers to the Minion.DBMaintInlineTokens table for the definition.

Note: PreferredDBName accepts LIKE expressions, in addition to inline tokens. So you could set PreferredDBName to %DBName%%, and (for example) for the DB1 database, it would work out to PreferredDBName = 'DB1%'. If there is more than one database that matches that pattern, MC will choose the database with the most recent create date.

Create and use a custom Inline Token

To create a custom token, insert a new row to the Minion.DBMaintInlineTokens table. Guidelines:

- **DynamicName**: Use a unique DynamicName.
- **ParseMethod**: Custom inline tokens can't use internal variables (such as @ExecutionDateTime) like the built-in tokens can. Custom tokens can only use SQL functions and @@variables.
- **IsCustom**: Mark IsCustom = 1.
- **Definition**: Provide a descriptive definition, for the use of you and your DBA team.

For example, we can use the following statement to create an Inline Token to represent the full day name (like Monday, etc.):

```
INSERT  INTO Minion.DBMaintInlineTokens
        ( DynamicName
        , ParseMethod
        , IsCustom
        , Definition
        , IsActive
        )
VALUES  ( 'DayNameFull'
        , 'DATENAME(dw, GetDate())'
        , 1
        , 'Returns the full name of the current day (e.g. Monday, Tuesday, etc.).'
        , 1
        );
```

IMPORTANT: The syntax for using this custom Inline Token is "|DayNameFull|". Notice that default tokens (like Server) use percent signs ("%Server%"), while custom tokens use pipe delimiters ("|DayNameFull|").

You can now use this custom token in fields that accept them. See the following section for more information.

Fields that accept Inline Tokens

You can use Inline Tokens in specific fields, in specific tables.

In Minion CheckDB, the table **Minion.CheckDBSettingsDB**:

- PreferredDBName
- RemoteJobName

In Minion Backup, fields in the tables **Minion.BackupSettingsPath** and **Minion.BackupRestoreSettingsPath**.

Custom Inline Tokens

We do have a few guidelines for creating your own tokens:

- **Naming DynamicName:** We recommend you do not include any special symbols – only alphanumeric characters. We also recommend against using the underscore symbol.

- **Defining ParseMethod:** Custom inline tokens can't use internal variables (such as @ExecutionDateTime) like the built-in tokens can. Custom tokens can only use SQL functions and @@variables.
- **Uniqueness**: Be aware that there is a unique constraint on DynamicName and IsActive; so you can only have one active "Date", and one inactive "Date" (as an example).
- **IsCustom**: Set IsCustom = 1 for your custom dynamic names.

IMPORTANT: Custom inline tokens must be surrounded by pipes, not percent signs.

Inline Token Internals

The shorthand for this section looks like this: Tokens in settings tables -> MC stored procedures -> Minion.DBMaintInlineTokensParse stored procedure -> Minion.DBMaintInlineTokens table.

In Minion Backup, multiple tables have fields that accept Inline Tokens; in Minion CheckDB 1.0, only the table Minion.CheckDB does. As a part of normal (or manual) CheckDB operations, the Minion.CheckDB stored procedure must access these fields and have the tokens translated.

This procedure in turn uses the stored procedure Minion.DBMaintInlineTokensParse to parse the token into its value. The DBMaintInlineTokensParse, of course, gets the token definition from the table Minion.DBMaintInlineTokens.

About: Multithreading operations

Minion CheckDB allows you to run multiple DBCC CheckDB processes, or multiple DBCC CheckTable processes, at the same time.

- To configure database multithreading, set the NumConcurrentOps value greater than one in Minion.CheckDBSettingsServer. This applies to both DBCC CheckDB and DBCC CheckTable operations.
- To configure table multithreading, set the DBInternalThreads value greater than one in Minion.CheckDBSettingsServer (or in Minion.CheckDBSettingsDB). **Note:** If you specify DBInternalThreads in Minion.CheckDBSettingsServer, that value takes precedence over the DBInternalThreads setting in Minion.CheckDBSettingsDB.

Warning: You can max out server resources very quickly if you use too many concurrent operations. If for example you're running 5 databases simultaneously, and each of those operations runs 10 tables simultaneously, that can add up very quickly!

IMPORTANT: Custom dynamic snapshots for CheckTable are only available for **single-threaded** operations. This means that you must set DBInternalThreads in Minion.CheckDBSettingsDB, and DBInternalThreads in Minion.CheckDBSettingsServer, to 1 for custom dynamic snapshots. For more information, see the Custom Dynamic Snapshots section in "About: Custom Snapshots"; and, see "About: Feature Compatibility".

Multithreading information is logged in Minion.CheckDBLogDetails. In a multithreaded run, the ProcessingThread column records number of the thread assigned to this operation. You can use this to query with GROUP BY to see the distribution of threads (e.g., did one thread handle most of the work, or was there a reasonably good distribution of work?)

On DisableDOP and "parallel checking"

In SQL Server Enterprise, by default a DBCC CheckDB operation runs with multiple parallel threads under the covers. If you set DisableDOP=1, you force it to use a single thread, instead of multiple threads. In Minion CheckDB, we have a completely separate (but compatible) concept called database multithreading; this is where we spawn two or more DBCC CheckDB operations to run simultaneously.

	DisableDOP = 0	**DisableDOP = 1**
Database Multithreading on	Multiple DBs process simultaneously; each may have multiple parallel threads.	Multiple DBs process simultaneously; each may have only one thread.
Database Multithreading off	Each DB is processed serially; each may have multiple parallel threads.	Each DB is processed serially; each may have only one thread.

> Checking Objects in Parallel – from https://msdn.microsoft.com/en-us/library/ms176064.aspx
>
> "By default, DBCC CHECKDB performs parallel checking of objects. The degree of parallelism is automatically determined by the query processor. The maximum degree of parallelism is configured just like parallel queries. To restrict the maximum number of processors available for DBCC checking, use sp_configure. For more information, see Configure the max degree of parallelism Server Configuration Option. Parallel checking can be disabled by using trace flag 2528. For more information, see Trace Flags (Transact-SQL)."

About: Rotational Scheduling

Minion CheckDB allows you to define a rotation scenario for your operations. For example, a nightly round of 10 databases would perform integrity checks on 10 databases the first night, another 10 databases the second night, and so on. You can choose to set up a CheckDB rotation for databases, CheckTable rotation for tables, or a combination of both.

You can also use the rotational scheduling to limit operations by time; for example, you could configure MC to cycle through DBCC CheckDB operations for 90 minutes each night.

The table Minion.CheckDBSettingsRotation holds the rotation scenario for your operations (e.g., "run CheckDB on 10 databases every night; the next night, process the next 10; and so on"). This table applies to both CheckDB and CheckTable operations.

For more information, see "How to: Configure Rotational Scheduling".

Example 1: DBCount rotation

Let's say we enable one of the default settings in Minion.CheckDBSettingsRotation, so we have a rotational schedule of 10 databases per run:

DBName	OpName	RotationLimiter	RotationLimiterMetric	RotationMetricValue
MinionDefault	CHECKDB	DBCount	count	10

Note that not all columns are shown here.

If our Minion CheckDB schedule is set to run CheckDB nightly, and we have 13 databases (DB1 through DB13), then:

- The first night would perform CheckDB on 10 databases: DB1 through DB10.
- The second night would include DB11, DB12, DB13, and DB1 through DB7.
- The third night would include DB8 through DB13, and DB1 through DB4.
- And, so on.

Example 2: Time rotation

DBName	OpName	RotationLimiter	RotationLimiterMetric	RotationMetricValue
MinionDefault	CHECKDB	Time	Mins	60

Note that not all columns are shown here.

If our Minion CheckDB schedule is set to run CheckDB nightly, and we have 13 databases (DB1 through DB13), then it might go like this:

- The first night, MC estimates that it can perform CheckDB on 4 databases in 60 minutes: DB1 through DB4.
- The second night, MC estimates that it can process the next 5 databases in 60 minutes: DB5 through DB9.
- The third night, MC estimates it can process 3 databases: DB10 through DB12.
- The fourth night, MC estimates it can process 5 databases: DB13 and DB1 through DB4.
- And, so on.

Rotational Scheduling Internals

The procedure Minion.CheckDBRotationLimiter (internal use only) sets up the list of objects that should run in the current batch. Here’s how:

1. **What’s been processed:** The SP pulls the list of objects that ran in the last batch from the Minion.CheckDBLogDetails table, and inserts that list to the Minion.CheckDBRotationDBs table. Now, the procedure knows which objects have been processed.
2. **Keep the latest run:** The SP then deletes all but the latest ExecutionDateTime for each object from the Minion.CheckDBRotationDBs table. It only keeps the latest run because it's tracking the last time an object was processed.
3. **Remove processed objects:** After that, the stored procedure deletes any objects from the work table Minion.CheckDBThreadQueue that exist in the Minion.CheckDBRotationDBs table. This means that objects which have already been processed for this period won't be included in the current run.

4. **Limit the list:** Finally, it deletes any objects from Minion.CheckDBThreadQueue that are over the metric value. For example, if you're only going to run 10 databases per run, this will only keep the first 10 databases in the list.

"How To" Topics

How To: View the results of an operation

The whole point of CheckDB and CheckTable operations is to determine whether there is any corruption. So of course, Minion CheckDB records the results of these operations, in the tables Minion.CheckDBResult and Minion.CheckDBCheckTableResult.

An easier way to determine if there were any errors, though, is to check the Status column in Minion.CheckDBLogDetails:

- **Complete** - operation completed without errors.
- **Complete (*N <opname>* Errors found)** - the integrity check operation completed with errors. Check the Consistency and AllocationErrors columns, and the Minion.CheckDBResults, table for full details.
- **Complete with Warnings** - operation completed, but there was an error with the process somewhere along the way. This is usually seen on remote CheckDB operations when the process has a problem getting the results back to the primary server. There are other circumstances that can complete with warning. There could be problems deleting the snapshot, or something else. The point is that the integrity check finished, but something else failed and it's impossible to say what the state of the error reporting will be.
- **Complete with Errors and Warnings** - a combination of the above two.
- **Complete with No Status** - This means the integrity check operation completed, but we specifically couldn't parse the error results. Again, this usually happens on remote runs when we can't figure out how many allocation or consistency errors there are, but it could happen on a local run if Microsoft sneaks in a new column into the result table. To get a "Complete" status, we rely on being able to parse the output; so when you get this message, it usually means that you don't have that return data from CheckDB/CheckTable/etc.
- **Fatal error: <error message>** - There was an error in the Minion CheckDB process itself, or CheckDB/CheckTable itself was unable to run on a database.

How To: Configure settings for a single database

Default settings for the whole system are stored in the Minion.CheckDBSettingsDB table (in the two rows marked DBName=MinionDefault). To specify settings that override those defaults for a specific database, insert two rows for that database to the Minion.CheckDBSettingsDB table – one row for CHECKDB, and one row for CHECKTABLE.

For example, we want to fine tune settings for DB1, so we use the following statement to insert two rows for DB1:

```
INSERT  INTO Minion.CheckDBSettingsDB
  ( DBName,
        OpLevel,
              OpName,
              Exclude,
              RepairOption,
              RepairOptionAgree,
              AllErrorMsgs,
              IncludeRemoteInTimeLimit,
              ResultMode,
              HistRetDays,
              DBInternalThreads,
              DefaultTimeEstimateMins,
              BeginTime,
              EndTime,
              DayOfWeek,
              IsActive,
              Comment )
VALUES  ( 'DB1'            -- DBName
        , 'DB'             -- OpLevel
        , 'CHECKDB'        -- OpName
        , 0                -- Exclude
        , 'NONE'           -- RepairOption
        , 1                -- RepairOptionAgree
        , 1                -- AllErrorMsgs
        , 1                -- IncludeRemoteInTimeLimit
        , 'Full'           -- ResultMode
        , 60               -- HistRetDays
        , 1                -- DBInternalThreads
        , 1                -- DefaultTimeEstimateMins
        , '0:00:00'        -- BeginTime
        , '23:59:00'       -- EndTime
        , 'Daily'          -- DayOfWeek
        , 1                -- IsActive
        , 'DB1 CheckDB'    -- Comment
        ),
       ( 'DB1'             -- DBName
        , 'DB'             -- OpLevel
        , 'CHECKTABLE'     -- OpName
        , 0                -- Exclude
        , 'NONE'           -- RepairOption
        , 1                -- RepairOptionAgree
        , 1                -- AllErrorMsgs
        , 1                -- IncludeRemoteInTimeLimit
        , 'Full'           -- ResultMode
        , 60               -- HistRetDays
        , 1                -- DBInternalThreads
        , 1                -- DefaultTimeEstimateMins
        , '0:00:00'        -- BeginTime
        , '23:59:00'       -- EndTime
        , 'Daily'          -- DayOfWeek
        , 1                -- IsActive
        , 'DB1 CheckTable' -- Comment
```

```
);
```

Minion CheckDB comes with a utility stored procedure, named Minion.CloneSettings, for easily creating insert statements like the example above. For more information, see the "Minion.CloneSettings" section.

IMPORTANT: If you enter database-specific rows, those rows completely override the settings for that particular database. For example, the rows inserted above will be the source of all settings – even if a setting is NULL – for all DB1 integrity check operations. For more information, see the "Configuration Settings Hierarchy" section in "Architecture Overview".

Follow the Configuration Settings Hierarchy Rule: If you provide a database-specific row, **be sure that both integrity check operations are represented in the table for that database**. For example, if you insert a row for DBName='DB1', OpName='CHECKDB', then also insert a row for DBName='DB1', OpName='CHECKTABLE'. Once you configure the settings context at the database level, the context *stays* at the database level (and does not return to the default 'MinionDefault' level for that database).

How To: Configure settings for all databases

When you first install an instance of Minion CheckDB, default settings for the whole system are stored in the Minion.CheckDBSettingsDB table rows where DBName='MinionDefault'. To change settings for all databases on the server, update the values for either or both of the two default rows.

For example, you might want to change the result mode from Full to Summary for CheckDB operations:

```
UPDATE Minion.CheckDBSettingsDB
SET    ResultMode='Summary'
WHERE  DBName = 'MinionDefault'
       AND OpName = 'CHECKDB';
```

Over time, you may have entered one or more database-specific rows for individual databases. In this case, the settings in the default "MinionDefault" rows do not apply to those databases types. You can of course update the entire table – both the default rows, and any database-specific rows – with new settings, to be sure that the change is universal for that instance. So for example, if you want the history retention days to be 90 (instead of the default, 60 days), run the following:

```
UPDATE Minion.CheckDBSettingsDB
SET    HistRetDays = 90;
```

How To: Process databases in a specific order

You can choose the order in which databases will be processed. For example, let's say that you want Minion CheckDB to check databases in this order:

1. [YourDatabase] (it's the most important database on your system)
2. [Semi]

3. [Lame]
4. [Unused]

In this case, we would insert a CheckDB row and a CheckTable row into the Minion.CheckDBSettingsDB table for each of the databases, specifying either GroupDBOrder, GroupOrder, or both, as needed. In the following example, we have inserted CheckDB and CheckTable rows for each database, and specified the GroupOrder.

DBName	Port	OpLevel	OpName	Exclude	GroupOrder	GroupDBOrder
MinionDefault	*NULL*	DB	CHECKDB	0	0	0
MinionDefault	1433	DB	CHECKTABLE	0	0	0
YourDatabase	*NULL*	DB	CHECKDB	0	100	0
YourDatabase	*NULL*	DB	CHECKTABLE	0	100	0
Semi	*NULL*	DB	CHECKDB	0	50	0
Semi	*NULL*	DB	CHECKTABLE	0	50	0
Lame	*NULL*	DB	CHECKDB	0	25	0
Lame	*NULL*	DB	CHECKTABLE	0	25	0
Unused	*NULL*	DB	CHECKDB	0	0	0
Unused	*NULL*	DB	CHECKTABLE	0	0	0

NOTE: For GroupDBOrder and GroupOrder, higher numbers have a greater "weight" - they have a higher priority - and will be backed up earlier than lower numbers. Note also that these columns are TINYINT, so weighted values must fall between 0 and 255.

NOTE: When you insert a row for a database, the settings in that row override **all** of the default operational settings for that database. So, inserting a row for [YourDatabase] means that ONLY CheckDB settings from that row will be used for [YourDatabase]; none of the default settings will apply to [YourDatabase].

NOTE: Any databases that rely on the default system-wide settings (represented by the row where DBName='MinionDefault') will be backed up according to the values in the MinionDefault columns *GroupDBOrder* and *GroupOrder*. By default, these are both 0 (lowest priority), and so non-specified databases would be backed up last.

Because we have so few databases in this example, the simplest method is to assign the heaviest "weight" to YourDatabase, and lesser weights to the other databases, in decreasing order. In our example, we would insert four rows. Note that, for brevity, we use far fewer columns in our examples than you would need in an actual environment:

```
INSERT INTO Minion.CheckDBSettingsDB
(DBName,
 OpLevel,
 OpName,
 Exclude,
 GroupOrder,
 GroupDBOrder,
 NoIndex,
 RepairOption,
 IsActive,
 Comment)
VALUES
(N'YourDatabase', 'DB', 'CHECKDB', 0, 100, 0, 0, 'NONE', 1, 'YourDatabase' ),
(N'YourDatabase', 'DB', 'CHECKDB', 0, 100, 0, 0, 'NONE', 1, 'YourDatabase' ),
(N'Semi', 'DB', 'CHECKDB', 0, 50, 0, 0, 'NONE', 1, 'Semi' ),
(N'Semi', 'DB', 'CHECKTABLE', 0, 50, 0, 0, 'NONE', 1, 'Semi' ),
(N'Lame', 'DB', 'CHECKDB', 0, 25, 0, 0, 'NONE', 1, 'Lame' ),
(N'Lame', 'DB', 'CHECKTABLE', 0, 25, 0, 0, 'NONE', 1, 'Lame' ),
(N'Unused', 'DB', 'CHECKDB', 0, 5, 0, 0, 'NONE', 1, 'Unused' ),
(N'Unused', 'DB', 'CHECKTABLE', 0, 5, 0, 0, 'NONE', 1, 'Unused' );
```

For a more complex ordering scheme, we could divide databases up into groups, and then order the CheckDBs both by group, and within each group. The pseudocode for this example might be:

- Insert rows for databases YourDatabase and Semi, both with GroupOrder = 200
  - Row YourDatabase: GroupDBOrder = 255
  - Row Semi: GroupDBOrder = 100
- Insert rows for databases Lame and Unused, both with GroupOrder = 100
  - Row YourDatabase: Lame = 255
  - Row Semi: Unused = 100

The resulting checkdb order would be as follows:

1. YourDatabase
2. Semi
3. Lame
4. Unused

How To: Change schedules

Minion CheckDB offers you a choice of scheduling options:

- You can use the Minion.CheckDBSettingsServer table to configure flexible scheduling scenarios;
- Or, you can use the traditional approach of one job per integrity check schedule;
- Or, you can use a hybrid approach that employs a bit of both options.

For more information about CheckDB schedules, see "About: CheckDB Schedules".

Table based scheduling

When Minion CheckDB is installed, it uses a single job (MinionCheckDB-AUTO) to run the stored procedure Minion.CheckDBMaster with no parameters, every hour. When the Minion.CheckDBMaster procedure runs without parameters, it uses the Minion.CheckDBSettingsServerDB table (among others) to determine its runtime parameters – including the schedule of operations per integrity check type. This is how MC operates by default, to allow for the most flexible scheduling with as few jobs as possible.

This document explains table based scheduling in the Quick Start section "Table based scheduling".

Parameter based scheduling (traditional approach)

Other SQL Server maintenance solutions traditionally use one job per schedule. To use the traditional approach of one job per schedule:

1. Disable or delete the *MinionCheckDB-Auto* job.
2. Configure new jobs for each integrity check schedule scenario you need.

Note: We *highly* recommend always using the Minion.CheckDBMaster stored procedure to run CheckDB and CheckTable operations. While it is possible to use the procedure Minion.CheckDB to perform integrity checks, doing so will bypass much of the configuration and logging benefits that Minion CheckDB was designed to provide.

Run Minion.CheckDBMaster with parameters: The procedure takes a number of parameters that are specific to the current maintenance run. (For full documentation of Minion.CheckDBMaster parameters, see the "Minion.CheckDBMaster" section.)

To configure traditional, one-job-per-schedule operations, you might configure three new jobs:

MinionCheckDB-SystemCheckDB, to run DBCC CheckDB for each system database nightly at 9pm. The job step should be something similar to:

```
EXEC Minion.CheckDBMaster @DBType = 'System'
        , @OpName = 'CHECKDB'
        , @StmtOnly = 0
        , @ReadOnly = 1;
```

MinionCheckDB-UserCheckDB, to run DBCC CheckDB for all but two user databases nightly at 10pm. The job step should be something similar to:

```
EXEC Minion.CheckDBMaster @DBType = 'User'
        , @OpName = 'CHECKDB'
        , @StmtOnly = 0
        , @ReadOnly = 1
        , @Exclude = 'DB4,DB5';
```

MinionCheckDB-UserCheckTable, to run DBCC CheckTable for certain user databases nightly at 11:00pm. The job step should be something similar to:

```
EXEC Minion.CheckDBMaster @DBType = 'User'
    , @OpName = 'CHECKDB'
    , @StmtOnly = 0
    , @ReadOnly = 1
    , @Include = 'DB4,DB5';
```

Hybrid scheduling

It is possible to use both methods – table based scheduling, and traditional scheduling – by one job that runs Minion.CheckDBMaster with no parameters, and one or more jobs that run Minion.CheckDBMaster with parameters.

We recommend against this, as hybrid scheduling has little advantage over either method, and increases the complexity of your scenario. However, it may be that there are as yet unforeseen situations where hybrid scheduling might be very useful.

How To: Configure timed settings

The "How To: Change Schedules" section described how to set up operational schedules. Timed settings are different from schedules: they are settings that only apply during certain windows of time.

For example, we could configure a CheckDB schedule to run all databases at noon on Saturday, and a second schedule to run "physical only" checks on DB1 nightly. We set up the schedule itself in Minion.CheckDBSettingsServer:

```
SELECT ID
    , DBType
    , OpName
    , Day
    , ReadOnly
    , BeginTime
    , EndTime
    , MaxForTimeframe
    , IsActive
FROM Minion.CheckDBSettingsServer;
```

DBType	**OpName**	**Day**	**ReadOnly**	**BeginTime**	**EndTime**	**MaxForTimeframe**
System	CHECKDB	Daily	1	20:00:00	21:30:00	1
User	CHECKDB	Weekday	1	22:00:00	23:30:00	1
User	CHECKDB	Saturday	1	12:00:00	14:00:00	1

But, notice that our schedule doesn't actually cover the "physical only" aspect of what we want. So, we must configure PHYSICAL_ONLY in Minion.CheckDBSettingsDB, with the proper time window (weekdays):

```
SELECT  DBName
     , OpLevel
     , OpName
     , IntegrityCheckLevel
     , BeginTime
     , EndTime
     , DayOfWeek
FROM    Minion.CheckDBSettingsDB;
```

DBName	**OpLevel**	**OpName**	**IntegrityCheckLevel**	**BeginTime**	**EndTime**	**DayOfWeek**
MinionDefault	DB	CHECKDB	**PHYSICAL_ONLY**	00:00:00	23:59:00	**Weekday**
MinionDefault	DB	CHECKTABLE	**PHYSICAL_ONLY**	00:00:00	23:59:00	**Weekday**
MinionDefault	DB	CHECKDB	*NULL*	00:00:00	23:59:00	Weekend
MinionDefault	DB	CHECKTABLE	*NULL*	00:00:00	23:59:00	Weekend

If we put this all together on paper, here is what a week of operations looks like:

Day	**DBType**	**Operation Begin Time**	**Integrity Check Level**
Monday through Friday	System	20:00:00	**PHYSICAL_ONLY**
Monday through Friday	User	22:00:00	**PHYSICAL_ONLY**
Saturday	System	20:00:00	*NULL*
Saturday	User	12:00:00	*NULL*
Sunday	System	20:00:00	*NULL*
Sunday	User	**(none)**	

How To: Generate statements only

Sometimes it is useful to generate integrity check statements and run them by hand, either individually or in small groups. To generate statements without running the statements, run the procedure Minion.CheckDBMaster with the parameter @StmtOnly set to 1.

Example code - The following code will generate full CheckDB statements for all system databases:

```
EXEC Minion.CheckDBMaster @DBType = 'User'
        , @OpName = 'CHECKDB'
        , @StmtOnly = 1
        , @ReadOnly = 1;
```

Running Minion.CheckDBMaster with @StmtOnly=1 will generate a list of Minion.CheckDB procedure execution statements, all set to @StmtOnly=1. Running these Minion.CheckDBDB statements will generate the DBCC CheckDB statements.

This is an excellent way to discover what settings Minion CheckDB will use for a particular database (or set of databases).

How To: Run code before or after integrity checks

You can schedule code to run before or after integrity checks, using precode and postcode. Pre- and postcode can be configured:

- Run code before or after the entire batch of operations
- Run code before or after a single database
- Run code before or after several, or each and every database
- Run code before or after a single table
- Run code before or after several, or each and every table in a database
- Run code before or after reindex statements (within the same statement batch)

IMPORTANT: Unless otherwise specified, pre- and postcode will run in the context of the Minion CheckDB database (wherever the Minion CheckDB objects are stored); it was a design decision not to limit the code that can be run to a specific database. Therefore, always use “USE” statements – or, for stored procedures, three-part naming convention – for pre- and postcode.

Batch precode and postcode

Batch precode and postcode run before and after an entire integrity check operation.

To run code before or after the integrity check batch, update (or insert) the appropriate row in Minion.CheckDBSettingsServer. In that row, populate the BatchPreCode column to run code before the integrity check operation; and populate the column BatchPostCode to run code after the integrity check operation. For example:

```
UPDATE  Minion.CheckDBSettingsServer
SET     BatchPreCode = 'EXEC master.dbo.IntegrityCheckPrep;'
      , BatchPostCode = 'EXEC master.dbo.IntegrityCheckCleanup;'
WHERE   DBType = 'User'
        AND OpName = 'CHECKDB'
        AND Day = 'Saturday';
```

IMPORTANT: The Minion.CheckDBSettingServer columns BatchPreCode and BatchPostCode are *only* in effect for table based scheduling – that is, running Minion.CheckDBMaster without parameters. If you use parameter based scheduling, the only way to enact batch precode or batch postcode is with additional job steps.

Database precode and postcode

Database precode and postcode run before and after an individual database; or, if there are multiple databases in the batch, before and after each database integrity check operation.

To run code before or after a single database, insert a row for the database into Minion.CheckDBSettingsDB. Populate the column DBPreCode to run code before the operations for that database; populate the column DBPostCode to run code after the operations for that database. Note that this table requires two rows for each database you enter: one CHECKDB and one CHECKTABLE. In our example, we want the precode and postcode to run whether the database is running a CHECKDB operation or a CHECKTABLE, so we populate the PreCode for both rows.

For example:

```
INSERT  INTO [Minion].CheckDBSettingsDB
        ( [DBName], [OpLevel], [OpName], [Exclude], [GroupOrder],
          [GroupDBOrder], [NoIndex], [RepairOption], [RepairOptionAgree],
          [AllErrorMsgs], [ExtendedLogicalChecks], [NoInfoMsgs], [IsTabLock],
          [IntegrityCheckLevel], [IsRemote], [ResultMode], [HistRetDays],
          [DefaultSchema], [DBPreCode], [DBPostCode], [DBInternalThreads],
          [LogSkips], [BeginTime], [EndTime], [DayOfWeek], [IsActive],
          [Comment] )
VALUES  ( 'DB1'             -- DBName
          , 'DB'                        -- OpLevel
```

```
 , 'CHECKDB'                    -- OpName
 , 0          -- Exclude
 , 0          -- GroupOrder
 , 0          -- GroupDBOrder
 , 0          -- NoIndex
 , 'NONE'     -- RepairOption
 , 0          -- RepairOptionAgree
 , 1          -- AllErrorMsgs
 , 0          -- ExtendedLogicalChecks
 , 0          -- NoInfoMsgs
 , 0          -- IsTabLock
 , 'PHYSICAL_ONLY'              -- IntegrityCheckLevel
 , 0          -- IsRemote
 , 'Full'     -- ResultMode
 , 60                  -- HistRetDays
 , 'dbo'      -- DefaultSchema
 , 'EXEC master.dbo.GenericSP1;' -- DBPreCode
 , 'EXEC master.dbo.GenericSP2;' -- DBPostCode
 , 1          -- DBInternalThreads
 , 1          -- LogSkips
 , '00:00:00'-- BeginTime
 , '23:59:00'-- EndTime
 , 'Weekday'           -- DayOfWeek
 , 1          -- IsActive
 , 'DB1 CHECKDB on weekdays.' )
                     ,
( 'DB1'                -- DBName
 , 'DB'                         -- OpLevel
 , 'CHECKTABLE'                 -- OpName
 , 0          -- Exclude
 , 0          -- GroupOrder
 , 0          -- GroupDBOrder
 , 0          -- NoIndex
 , 'NONE'     -- RepairOption
 , 0          -- RepairOptionAgree
 , 1          -- AllErrorMsgs
 , 0          -- ExtendedLogicalChecks
 , 0          -- NoInfoMsgs
 , 0          -- IsTabLock
 , 'PHYSICAL_ONLY'              -- IntegrityCheckLevel
 , 0          -- IsRemote
 , 'Full'     -- ResultMode
 , 60                  -- HistRetDays
 , 'dbo'      -- DefaultSchema
 , 'EXEC master.dbo.GenericSP1;' -- DBPreCode
 , 'EXEC master.dbo.GenericSP2;' -- DBPostCode
 , 1          -- DBInternalThreads
 , 1          -- LogSkips
 , '00:00:00'-- BeginTime
 , '23:59:00'-- EndTime
```

```
        , 'Weekday'             -- DayOfWeek
        , 1             -- IsActive
        , 'DB1 CHECKTABLE on weekdays.' );
```

To run code before or after each and every database, update the MinionDefault row AND every database-specific rows (if any) in Minion.CheckDBSettingsDB, populating the column DBPreCode or DBPostCode. For example:

```
UPDATE [Minion].[CheckDBSettingsDB]
SET                 DBPreCode = 'EXEC master.dbo.GenericSP1;' ,
                    DBPostCode = 'EXEC master.dbo.GenericSP1;'
WHERE  DBName = 'MinionDefault'
                    AND OpName IN ('CHECKDB', 'CHECKTABLE');
```

To run code before or after each of a few databases, insert one row for each of the databases into Minion.CheckDBSettingsDB, populating the DBPreCode column and/or DBPostCode column as appropriate.

To run code before or after all but a few databases, update the MinionDefault row in Minion.CheckDBSettingsDB, populating the DBPreCode column and/or the DBPostCode column as appropriate. This will set up the execution code for all databases. Then, to prevent that code from running on a handful of databases, insert a row for each of those databases to Minion.CheckDBSettingsDB, and keep the DBPreCode and DBPostCode columns set to *NULL*.

For example, if we want to run the stored procedure dbo.SomeSP before each database *except* databases DB1, DB2, and DB3, we would:

1. Update row in Minion.CheckDBSettingsDB for “MinionDefault”, setting PreCode to ‘EXEC dbo.SomeSP;’
2. Insert a row to Minion.CheckDBSettingsDB for [DB1], establishing all appropriate settings, and setting DBPreCode to *NULL*.
3. Insert a row to Minion.CheckDBSettingsDB for [DB2], establishing all appropriate settings, and setting DBPreCode to *NULL*.
4. Insert a row to Minion.CheckDBSettingsDB for [DB3], establishing all appropriate settings, and setting DBPreCode to *NULL*.

Note: The Minion.CheckDBSettingsDB columns DBPreCode and DBPostCode are in effect whether you are using table based scheduling – that is, running Minion.CheckDBMaster without parameters – or using parameter based scheduling. (This is not the case for batch precode and postcode, which the previous section covers.)

Table precode and postcode

Table precode and postcode run before and after an individual table; or, if there are multiple table in the batch, before and after each DBCC CheckTable operation.

To run code before or after a single table, insert a row for the table into Minion.CheckDBSettingsTable. Populate the column TablePreCode to run code before the operations for that table; populate the column TablePostCode to run code after the operations after that table.

For example:

```
INSERT  INTO Minion.CheckDBSettingsTable
        ( DBName , SchemaName , TableName , Exclude , DefaultTimeEstimateMins , NoIndex
          , AllErrorMsgs , ExtendedLogicalChecks , NoInfoMsgs , IsTabLock , ResultMode , HistRetDays
          , TablePreCode , TablePostCode , BeginTime , EndTime , DayOfWeek , IsActive
          , Comment
        )
VALUES  ( N'DB1'            -- DBName
        , N'dbo'       -- SchemaName
        , N'MyTable'  -- TableName
        , 0            -- Exclude
        , 10                        -- DefaultTimeEstimateMins
        , 0            -- NoIndex
        , 1            -- AllErrorMsgs
        , 1            -- ExtendedLogicalChecks
        , 0            -- NoInfoMsgs
        , 0            -- IsTabLock
        , 'FULL'       -- ResultMode
        , 30                        -- HistRetDays
        , N''                       -- TablePreCode
        , N''                       -- TablePostCode
        , '00:00:00'  -- BeginTime
        , '23:59:59'  -- EndTime
        , 'Daily'      -- DayOfWeek
        , 1            -- IsActive
        , 'DB1.dbo.MyTable daily CheckTable.'
        );
```

To run code before or after each and every table, update the MinionDefault CHECKTABLE row AND every database-specific CHECKTABLE rows (if any) in Minion.CheckDBSettingsDB, populating the column TablePreCode or TablePostCode. For example:

```
UPDATE [Minion].[CheckDBSettingsDB]
SET             TablePreCode = 'EXEC master.dbo.GenericSP1;' ,
                TablePostCode = 'EXEC master.dbo.GenericSP1;'
WHERE  DBName = 'MinionDefault'
                AND OpName = 'CHECKTABLE';
```

To run code before or after each of a few tables, insert one row for each of the tables into Minion.CheckDBSettingsTable, populating the TablePreCode column and/or TablePostCode column as appropriate.

To run code before or after all but a few tables, update the MinionDefault row in Minion.CheckDBSettingsDB, populating the TablePreCode column and/or the TablePostCode column as appropriate. This will set up the execution code for all databases. Then, to prevent that code from running on a handful of tables, insert a row for each of those databases to Minion.CheckDBSettingsTable, and keep the TablePreCode and TablePostCode columns set to *NULL*.

For example, if we want to run the stored procedure dbo.SomeSP before each table *except* the DB1 tables T1 and T2, we would:

1. Update row in Minion.CheckDBSettingsDB for "MinionDefault", setting TablePreCode to 'EXEC dbo.SomeSP;'.
2. Insert a row to Minion.CheckDBSettingsTable for [DB1].dbo.T1, establishing all appropriate settings, and setting TablePreCode to *NULL*.
3. Insert a row to Minion.CheckDBSettingsTable for [DB1].dbo.T2, establishing all appropriate settings, and setting TablePreCode to *NULL*.

Note: The columns TablePreCode and TablePostCode (in both Minion.CheckDBSettingsDB and Minion.CheckDBSettingsTable) are in effect whether you are using table based scheduling – that is, running Minion.CheckDBMaster without parameters – or using parameter based scheduling. (This is not the case for batch precode and postcode, which an earlier section covers.)

Statement prefix and suffix

Statement prefix and suffix allow you to begin or end *every* integrity check statement with a statement of your own. This is different from the precode and postcode, because it is run within the same *batch*. Whereas, precode and postcode are run as completely separate statements, in different contexts.

You can set statement prefix and suffix (StmtPrefix, StmtSuffix) in Minion.CheckDBSettingsDB, or Minion.CheckDBSettingsTable, or both. The best use case for this is turning a trace flag on and off before/after your operations.

To set statement prefix and suffix at the database level, follow the same procedure in "Database precode and postcode" above (substituting StmtPrefix/StmtSuffix for DBPrecode/DBPostcode).

To set statement prefix and suffix at the table level, follow the same procedure in "Table precode and postcode" above (substituting StmtPrefix/StmtSuffix for TablePrecode/TablePostcode).

How To: Include or exclude READ_ONLY databases from integrity checks

You can control the inclusion of READ_ONLY databases in one of two ways: in the Minion.CheckDBSettingsServer table, or using the Minion.CheckDBMaster stored procedure. For either method, the ReadOnly values are:

- 1 – Include READ_ONLY databases in the CheckDB routine. This is the default option.
- 2 – Do NOT include READ_ONLY databases in the CheckDB routine.

- 3 – ONLY include READ_ONLY databases in the CheckDB routine.

To exclude READ_ONLY databases using table based scheduling, update the ReadOnly field for the appropriate rows in Minion.CheckDBSettingsServer:

```
UPDATE  Minion.CheckDBSettingsServer
SET     [ReadOnly] = 2
WHERE   DBType = 'User'
        AND [Day] = 'Saturday';
```

To exclude READ_ONLY databases in the CheckDB routine, run the procedure Minion.CheckDBMaster with the parameter @ReadOnly set to 2. For example, to perform CheckDB only on the read/write user databases, use the following call:

```
EXEC [Minion].[CheckDBMaster]
                @DBType = 'User' ,
                @OpName = 'CHECKDB',
                @ReadOnly = 2;
```

To include READ_ONLY databases and read/write databases, set @ReadOnly=1. And to perform maintenance only on read only databases, set @ReadOnly=3.

How To: Include databases in operations

By default, Minion CheckDB is configured to check all databases. As you fine tune your scenarios and schedules, you may want to configure specific subsets of databases to be checked with different options, or at different times.

You can limit the set of databases to be checked in a single operation via an explicit list, LIKE expressions, or regular expressions. In the following two sections, we will work through the way to do this first via table based scheduling, and then in traditional scheduling.

NOTE: The use of the regular expressions include and exclude features are not supported in SQL Server 2005.

Include databases in table based scheduling

Table based scheduling pulls CheckDB schedules and other options from the Minion.CheckDBSettingsServer table. In this table, you have the following options for configuring which databases to include in CheckDB operations:

- **To include all databases in an operation, set Include = 'All' (or NULL)** for the relevant row(s).
- **To include a specific list of databases, set Include = a comma delimited list of those database names**, and/or LIKE expressions. (For example: 'YourDatabase, DB1, DB2', or 'YourDatabase, DB%'.)
- **To include databases based on regular expressions, set Include = 'Regex'**. Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

We will use the following sample data as we demonstrate each of these options. This is a subset of Minion.CheckDBSettingsServer columns:

ID	DBType	OpName	Day	BeginTime	EndTime	MaxForTimeframe	Include	Exclude
1	System	CHECKDB	Daily	22:00:00	22:30:00	1	NULL	NULL
2	User	CHECKDB	Saturday	23:00:00	23:30:00	1	DB1,DB2	NULL
3	User	CHECKDB	Sunday	23:00:00	23:30:00	1	Regex	NULL
4	User	AUTO	Weekday	23:00:00	23:30:00	1	NULL	NULL

And, here are the contents of the Minion.DBMaintRegexLookup table:

Action	MaintType	Regex
Include	CheckDB	DB[3-5](?!\d)

Based on this data, Minion CheckDB would perform CheckDBs as follows:

- DBCC CheckDB for all system databases, daily at 10:00 pm.
- DBCC CheckDB for DB1 and DB2, Saturdays at 11:00 pm.
- DBCC CheckDB for databases included in the regular expressions table (Minion.DBMaintRegexLookup), run Sundays at 11:00 pm. (This particular regular expression includes DB3, DB4, and DB5, but does not include any database with a 2 digit number at the end, such as DB35.)
- Operations for user databases every weekday at 11:00 pm. (The AUTO option allows Minion CheckDB to choose the appropriate operation per database. For more information, see "How to: Configure Minion CheckDB Dynamic Thresholds".)

Note that you can create more than one regular expression in Minion.DBMaintRegexLookup. For example:

- **To use Regex to include DB3, DB4, and DB5**: insert a row like the example above, where Regex = 'DB[3-5](?!\d)'.
- **To use Regex to include any database beginning with the word "Market" followed by a number**: insert a row where Regex='Market[0-9]'.
- **With these two rows**, a CheckDB operation with @Include='Regex' will CheckDB *both* the DB3-DB5 databases, and the databases Marketing4 and Marketing308 (and similar others, if they exist).

Include databases in traditional scheduling

We refer the common practice of configuring integrity checks in separate jobs (to allow for multiple schedules) as "traditional scheduling". Shops that use traditional scheduling will run Minion.CheckDBMaster with parameters configured for each particular run.

You have the following options for configuring which databases to include in integrity check operations:

- **To include all databases in a CheckDB operation, set @Include = 'All' (or NULL)**.

- **To include a specific list of databases, set @Include = a comma delimited list of those database names**, and/or LIKE expressions. (For example: 'YourDatabase, DB1, DB2', or 'YourDatabase, DB%'.)
- **To include databases based on regular expressions, set @Include = 'Regex'**. Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

The following example executions will demonstrate each of these options.

First, to run DBCC CheckDB on all user databases, we would execute Minion.CheckDBMaster with these (or similar) parameters:

```
-- @Include = NULL for all databases
EXEC Minion.CheckDBMaster
            @DBType = 'User',
            @OpName= 'CHECKDB',
            @StmtOnly = 1,
            @Include = NULL,
            @Exclude=NULL,
            @ReadOnly=1;
```

To include a specific list of databases:

```
-- @Include = a specific database list (YourDatabase, all DB1% DBs, and DB2)
EXEC Minion.CheckDBMaster
            @DBType = 'User',
            @OpName = 'CHECKDB',
            @StmtOnly = 1,
            @Include = 'YourDatabase,DB1%,DB2',
            @Exclude=NULL,
            @ReadOnly=1;
```

To include databases based on regular expressions, first insert the regular expression into the Minion.DBMaintRegexLookup table, and then execute Minion.CheckDBMaster with @Include='Regex':

```
INSERT  INTO Minion.DBMaintRegexLookup
        ( [Action] ,
          [MaintType] ,
          [Regex]
        )
SELECT  'Include' AS [Action] ,
        'CheckDB' AS [MaintType] ,
        'DB[3-5](?!\d)' AS [Regex];

-- @Include = 'Regex' for regular expressions
EXEC Minion.CheckDBMaster
            @DBType = 'User',
            @OpName = 'CHECKDB',
            @StmtOnly = 1,
   @Include = 'Regex',
            @Exclude=NULL,
            @ReadOnly=1;
```

For information on Include/Exclude precedence (that applies to both the Minion.CheckDBSettingsServer columns, and to the parameters), see "Include and Exclude Precedence".

How To: Exclude databases from operations

By default, Minion CheckDB is configured to perform integrity checks on all databases. As you fine tune your scenarios and schedules, you may want to exclude certain databases from scheduled integrity check operations, or even from *all* integrity check operations.

You can exclude databases from all integrity check operations via the Exclude column in Minion.CheckDBSettingsDB. Or, you can exclude databases from integrity check operations via an explicit list, LIKE expressions, or regular expressions. In the following three sections, we will work through Exclude=1, then excluding databases from table based scheduling, and finally excluding from traditional scheduling.

NOTE: The use of the regular expressions include and exclude features are not supported in SQL Server 2005.

Exclude a database from all integrity checks

To exclude a database – for example, DB13 – from all integrity checks, just insert database-specific rows for that database into Minion.CheckDBSettingsDB, one with CheckDBType=CHECKDB and one with CheckDBType=CHECKTABLE; and Exclude=1:

```
        INSERT  INTO [Minion].CheckDBSettingsDB
                ( DBName
                , OpLevel
                , OpName
                , Exclude
                , IsActive
                , Comment
                )
        VALUES
                ( 'DB13' -- DBName
                , 'DB'      -- OpLevel
                , 'CHECKDB' -- OpName
                , 1          -- Exclude
                , 1          -- IsActive
                , 'Exclude DB13' -- Comment
                ),
                ( 'DB13' -- DBName
                , 'DB'      -- OpLevel
                , 'CHECKTABLE' -- OpName
                , 1          -- Exclude
                , 1          -- IsActive
                , 'Exclude DB13' -- Comment
                );
```

IMPORTANT: This insert has a bare minimum of options, as the row is only intended to exclude DB13 from the CheckDB routine. We recommend configuring individual database rows with the full complement of settings if there is a chance that integrity checks may be re-enabled for that database in the future.

IMPORTANT: Exclude=1 can be overridden by an explicit Include. For more information, see "Include and Exclude Precedence".

Exclude databases in table based scheduling

Table based scheduling pulls operational schedules (and other options) from the Minion.CheckDBSettingsServer table. In this table, you have the following options for configuring which databases to exclude from CheckDB operations:

- **To exclude a specific list of databases, set Exclude = a comma delimited list of those database names**, and/or LIKE expressions. (For example: 'YourDatabase, DB1, DB2', or 'YourDatabase, DB%'.)
- **To exclude databases based on regular expressions, set Exclude = 'Regex'**. Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

We will use the following sample data as we demonstrate each of these options. This is a subset of Minion.CheckDBSettingsDBServer columns:

ID	DBType	OpName	Day	BeginTime	EndTime	Include	Exclude
1	System	AUTO	Daily	21:00:00	23:59:00	*NULL*	*NULL*
2	User	AUTO	Saturday	22:00:00	23:59:00	*NULL*	RegEx
3	User	AUTO	Sunday	22:00:00	23:59:00	DB1,DB2	*NULL*

And, here are the contents of the Minion.DBMaintRegexLookup table:

Action	MaintType	Regex
Exclude	CheckDB	DB[3-5](?!\d)

Based on this data, Minion CheckDB would perform operations as follows:

System databases would get CheckDB or CheckTable operations (based on settings in the Minion.CheckDBSettingsAutoThresholds table) **daily at 9pm**.
User databases – except for those excluded via the regular expressiosn table – would get CheckDB or CheckTable operations (based on settings in the Minion.CheckDBSettingsAutoThresholds table) **Saturday at 9pm**.
Full user database CheckDBs for all databases – except for those excluded via the regular expressions table (Minion.DBMaintRegexLookup) – **run Saturdays at 10pm**. This particular regular expression excludes DB3, DB4, and DB5 from CheckDBs, but does not exclude any database with a 2 digit number at the end, such as DB35.
Full user database CheckDBs for databases DB1 and DB2 run **Sundays at 10pm**.

Note that you can create more than one regular expression in Minion.DBMaintRegexLookup. For example:

To use Regex to exclude DB3, DB4, and DB5: insert a row like the example above, where Regex = ’DB[3-5](?!\d)’.
To use Regex to exclude any database beginning with the word “Market” followed by a number: insert a row where Regex=’Market[0-9]’.
With these two rows, a CheckDB operation with @Exclude=’Regex’ will exclude *both* the DB3-DB5 databases, and the databases Marketing4 and Marketing308 (and similar others, if they exist) from integrity checks.

```
  _
 /   \
|***|
 \__/
```

Exclude databases in traditional scheduling

We refer the common practice of configuring maintenance in separate jobs (to allow for multiple schedules) as “traditional scheduling”. Shops that use traditional scheduling will run Minion.CheckDBMaster with parameters configured for each particular CheckDB run.

You have the following options for configuring which databases to exclude from integrity check operations:

To exclude a specific list of databases, set @Exclude = a comma delimited list of those database names, and/or LIKE expressions. (For example: ‘YourDatabase, DB1, DB2’, or ‘YourDatabase, DB%’.)
To exclude databases based on regular expressions, set @ Exclude = ‘Regex’. Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

The following example executions will demonstrate each of these options.

First, to exclude a specific list of databases:

```
-- @Exclude = a specific database list (YourDatabase, all DB1% DBs, and DB2)
EXEC Minion.CheckDBMaster
        @DBType = 'User',
        @OpName = 'CHECKDB',
        @StmtOnly = 1,  -- Only generate the statements for now!
        @Include = NULL,
        @Exclude='YourDatabase,DB1%,DB2',
        @ReadOnly=1;
```

To exclude databases based on regular expressions, first insert the regular expression into the Minion.DBMaintRegexLookup table, and then execute Minion.CheckDBMaster with @Exclude=’Regex’:

```
INSERT  INTO Minion.DBMaintRegexLookup
        ( [Action] ,
          [MaintType] ,
          [Regex]
        )
SELECT  'Exclude' AS [Action] ,
        'CheckDB' AS [MaintType] ,
        'DB[3-5](?!\d)' AS [Regex]
-- @Exclude = 'Regex' for regular expressions
EXEC Minion.CheckDBMaster
            @DBType = 'User',
            @OpName = 'CHECKDB',
            @StmtOnly = 1,  -- Only generate the statements for now!
            @Include = NULL,
            @Exclude='Regex',
            @ReadOnly=1;
```

For information on Include/Exclude precedence (that applies to both the Minion.CheckDBSettingsDBServer columns, and to the parameters), see "Include and Exclude Precedence".

How To: Include or exclude tables from operations

By default, Minion CheckDB is configured to check all databases and all tables. As you fine tune your scenarios and schedules, you may want to configure specific subsets of tables to be checked with different options, or at different times.

You can limit the set of tables to be checked in a single operation via an explicit list, and/or LIKE expressions. In the following two sections, we will work through the way to do this first via table based scheduling, and then in traditional scheduling.

Include tables in table based scheduling

Table based scheduling pulls schedules and other options from the Minion.CheckDBSettingsServer table. In this table, you have the following options for configuring which tables to include in CheckDB operations:

- **To include all tables in CheckTable operations, set Tables = NULL** for the relevant row(s).
- **To include all tables in one or more schemas in CheckTable operations, set Tables = NULL and Schemas = a comma delimited list of those schema names**.
- **To include a specific list of tables, set Tables = a comma delimited list of those table names**, and/or LIKE expressions. (For example: 'YourTable, T1, T2', or 'YourTable, T%'.)

We will use the following sample data as we demonstrate each of these options. This is a subset of Minion.CheckDBSettingsServer columns:

ID	DBType	OpName	Day	BeginTime	EndTime	MaxForTimeframe	Schemas	Tables
1	System	CHECKDB	Daily	22:00:00	22:30:00	1	*NULL*	NULL
2	User	CHECKDB	Saturday	23:00:00	23:30:00	1	*NULL*	NULL
3	User	CHECKTABLE	Daily	21:00:00	22:30:00	1	dbo	T1,T2
4	User	CHECKTABLE	Daily	20:00:00	21:30:00	1	M1	*NULL*

Based on this data, Minion CheckDB would perform operations as follows:

- DBCC CheckDB for all system databases, daily at 10:00 pm.
- DBCC CheckDB for user databases, Saturdays at 11:00 pm.
- DBCC CheckTable for tables dbo.T1 and dbo.T2, daily at 9:00 pm. Note that as Include = NULL (not shown), MC will perform a CheckTable on ALL tables named dbo.T1 and dbo.T2, in any database.
- DBCC CheckTable for all tables in schema "M1", daily at 8:00 pm. Note that as Include = NULL (not shown), MC will perform a CheckTable on ALL tables in the M1 schema, in any database.

Include tables in traditional scheduling

We refer the common practice of configuring integrity checks in separate jobs (to allow for multiple schedules) as "traditional scheduling". Shops that use traditional scheduling will run Minion.CheckDBMaster with parameters configured for each particular run.

You have the following options for configuring which tables to include in integrity check operations:

- **To include all tables in a DBCC CheckTable operation, set @Tables = NULL.**
- **To include all tables in one or more schemas, set @Tables = NULL and @Schemas = a comma delimited list of those schema names**.
- **To include a specific list of databases, set @Tables = a comma delimited list of those table names**, and/or LIKE expressions. (For example: 'YourTable, T1, T2', or 'YourTable, T%'.)

IMPORTANT: @Schemas does not limit @Tables; if you set @Schemas = 'A' and @Tables = 'T1', MC will attempt to process all tables within schema 'A', PLUS all tables named T1 (MC will look for dbo.T1 unless otherwise specified in DefaultSchema). However, @DBName limits both @Schemas and @Tables.

The following example executions will demonstrate each of these options.

First, to run DBCC CheckTables on all user databases, we would execute Minion.CheckDBMaster with these (or similar) parameters:

```
EXEC Minion.CheckDBMaster
        @DBType = 'User',
        @OpName= 'CHECKTABLE',
        @StmtOnly = 1,
        @Include = NULL,
        @Exclude=NULL,
        @Schemas=NULL,
        @Tables=NULL,
        @ReadOnly=1;
```

To run DBCC CheckTables on all tables in a database:

```
EXEC Minion.CheckDBMaster
        @DBType = 'User',
        @OpName = 'CHECKTABLE',
        @StmtOnly = 1,
        @Include = 'DB1',
        @Exclude=NULL,
        @Schemas= NULL,
        @Tables=NULL,
        @ReadOnly=1;
```

To run DBCC CheckTables on a specific list of schemas in a database:

```
    EXEC Minion.CheckDBMaster
            @DBType = 'User',
            @OpName = 'CHECKTABLE',
            @StmtOnly = 1,
            @Include = 'DB1',
            @Exclude=NULL,
            @Schemas='A,B,dbo,M%',
            @Tables=NULL,
            @ReadOnly=1;
```

To run DBCC CheckTables on a specific list of tables in a database:

```
    EXEC Minion.CheckDBMaster
            @DBType = 'User',
            @OpName = 'CHECKTABLE',
            @StmtOnly = 1,
            @Include = 'DB1',
            @Exclude=NULL,
            @Schemas= NULL,
            @Tables='dbo.T1,A.tab,B.tab',
            @ReadOnly=1;
```

This is not a comprehensive set of the things you can do with traditional scheduling, but only a small sample. For more information, see "Minion.CheckDBMaster".

How to: Configure Dynamic Thresholds

Minion CheckDB allows you to automate whether databases get a DBCC CheckDB operation, or a DBCC CheckTable operation. Configure dynamic integrity check thresholds in the Minion.CheckDBSettingsAutoThresholds table. These settings only apply to runs of the stored procedure Minion.CheckDBMaster where OpName = 'Auto' in Minion.CheckDBSettingsDB (or, for a manual run, where @OpName = 'Auto').

The default entry that comes installed with Minion CheckDB sets a threshold by size, at 100 GB. What this means is that by default – for Minion.CheckDBMaster runs with @OpName = 'Auto', **any database under 100 GB gets a CheckDB operation instead of a CheckTable operation**.

Note: As outlined in the "Configuration Settings Hierarchy" section, more specific settings in a table take precedence over less specific settings. So if you insert a database-specific row for DB1 to this table, that row will be used for DB1 (instead of the "MinionDefault" row).

Let's take the example where the Minion.CheckDBSettingsAutoThresholds "MinionDefault" row is set at 100 GB, but we need DB1 to have a CHECKDB operation if it's under 50 GB. Insert a row for DB1 to override MinionDefault (for that database):

```
INSERT  INTO Minion.CheckDBSettingsAutoThresholds
        ( [DBName]
        , [ThresholdMethod]
        , [ThresholdType]
        , [ThresholdMeasure]
        , [ThresholdValue]
        , [IsActive]
        , [Comment]
        )
SELECT  'DB1' AS [DBName]
        , 'Size' AS [ThresholdMethod]
        , 'DataAndIndex' AS [ThresholdType]
        , 'GB' AS [ThresholdMeasure]
        , 50 AS [ThresholdValue]
        , 1 AS [IsActive]
        , 'DB1' AS [Comment];
```

The setting applies to any run of Minion.CheckDBMaster where OpName = 'AUTO' in Minion.CheckDBSettingsDB (or, for a manual run, where @OpName = 'Auto'). So, let's insert a row to the Minion.CheckDBSettingsServer table for a Sunday AUTO run:

```
INSERT  INTO Minion.CheckDBSettingsServer
        ( DBType
        , OpName
        , Day
        , ReadOnly
        , BeginTime
        , EndTime
        , MaxForTimeframe
        , FrequencyMins
        , Schemas
        , Debug
        , FailJobOnError
        , FailJobOnWarning
        , IsActive
        , Comment
        )
VALUES  ( 'User'            -- DBType
        , 'AUTO'            -- OpName
        , 'Sunday'          -- Day
```

```
                , 1                     -- ReadOnly
                , '14:00:00'            -- BeginTime
                , '18:00:00'            -- EndTime
                , 1                     -- MaxForTimeframe
                , 0             -- FrequencyMins
                , NULL          -- Schemas
                , 0                     -- Debug
                , 0                     -- FailJobOnError
                , 0                     -- FailJobOnWarning
                , 1                     -- IsActive
                , 'Sunday AUTO op'  -- Comment
        );
```

That's it!

How to: Configure Rotational Scheduling

Minion CheckDB allows you to define a rotation scenario for your operations. For example, a nightly round of 10 databases would perform integrity checks on 10 databases the first night, another 10 databases the second night, and so on. You can schedule rotations for CheckTable operations, CheckDB operations, or both.

You can also use the rotational scheduling to limit operations by time; for example, you could configure MC to cycle through DBCC CheckDB operations for 90 minutes each night. Note that the timed rotations are an experimental feature; test first and use with caution!

The table Minion.CheckDBSettingsRotation holds the rotation scenario for your operations (e.g., "run CheckDB on 10 databases every night; the next night, process the next 10; and so on"). This table applies to both CheckDB and CheckTable operations.

Scenario 1: Run CheckDB on 10 databases each night.

The Minion.CheckDBSettingsRotation comes installed with inactive, default rows for different rotation scenarios. To run DBCC CheckDB on 10 databases each night, enable the "CheckDB/DBCount" row and make sure that RotationMetricValue is set to 10:

```
UPDATE Minion.CheckDBSettingsRotation
SET IsActive = 1,
   RotationMetricValue = 10
WHERE DBName = 'MinionDefault'
     AND OpName = 'CHECKDB'
     AND RotationLimiter = 'DBCount';
```

Scenario 2: Run CheckTable on 50 tables each night.

The Minion.CheckDBSettingsRotation comes installed with inactive, default rows for different rotation scenarios. To run DBCC CheckTable on 50 tables each night, enable the "CheckTable/DBCount" row and make sure that RotationMetricValue is set to 50:

```
UPDATE Minion.CheckDBSettingsRotation
SET IsActive = 1,
   RotationMetricValue = 50
WHERE DBName = 'MinionDefault'
     AND OpName = 'CHECKTABLE'
     AND RotationLimiter = 'DBCount';
```

How to: Set up CheckDB on a Remote Server

Minion CheckDB allows you to run DBCC CheckDB remotely for any database. The "Dynamic Remote CheckDB" feature additionally allows you to set a tuning threshold, so the CheckDB will run remotely only if it is above that threshold.

Note: See "About: Remote CheckDB" for remote CheckDB requirements and information.

We can configure one of many remote CheckDB scenarios. Starting with the simplest scenario:

- Remote CheckDB for all databases
- Remote CheckDB for a single database
- Remote CheckDB for any database above a certain size (remote thresholds)

Scenario 1: Remote CheckDB for all databases

Here we will configure remote CheckDB operations for all databases. If the remote CheckDB requirements are met, the only step is to enable and configure remote CheckDB.

To enable and configure remote CheckDB for all databases, update Minion.CheckDBSettingsDB:

```
UPDATE  Minion.CheckDBSettingsDB
SET     IsRemote = 1
     , IncludeRemoteInTimeLimit = 0
     , PreferredServer = 'YourRemoteSvr1'
     , PreferredServerPort = NULL
     , PreferredDBName = '%DBName%'
     , RemoteJobName = 'MinionCheckDB-%Server%_%DBName%'
     , RemoteCheckDBMode = 'Disconnected'
     , RemoteRestoreMode = 'LastMinionBackup'
     , DropRemoteDB = 1
     , DropRemoteJob = 1
WHERE OpName = 'CHECKDB';
```

Because we are updating every CHECKDB row in the table, *all* CheckDB operations will be conducted on the remote server.

You can look up the meaning of each of these fields in the Minion.CheckDBSettingsDB section. But this update statement does need some immediate discussion:

- **IsRemote** enables remote CheckDB.
- Edit **PreferredServer** to reflect the name of *your* remote server.
- The definition of **PreferredDBName** and **RemoteJobName** are entirely up to you. Notice that in the statement above, we use default Inline Tokens "Server" and "DBName". Get more information about that in "About: Inline Tokens".
- The choice between Connected and Disconnected **RemoteCheckDBMode** is entirely yours. Connected mode has fewer moving parts internally; but Disconnected mode has higher tolerance for things like network fluctuations.
- You can learn more about **RemoteRestoreMode** more in the "About: Remote CheckDB" section. In brief, LastMinionBackup (and NewMinionBackup) requires Minion Backup 1.3 running on the local server.
- **DropRemoteDB** set to 0 will retain the remote database after the operation is complete. If you set it to 1, then at the end of the DBCC operation, Minion CheckDB will drop the remote database. For RemoteRestoreMode = NewMinionBackup or LastMinionBackup, it usually makes sense to enable DropRemoteDB.
- The only reason to set **DropRemoteJob** = 0 is for troubleshooting purposes. Otherwise, we highly recommend enabling this.

From here on, any CheckDB operation will be completed on the remote server, and the information will be logged locally (in the source server).

Scenario 2: Remote CheckDB for a single database

The process for setting up remote CheckDB for a single database is remarkably similar to Scenario 1, above. The difference is, of course, we must configure the individual database settings. So the steps are:

1. Insert rows for CHECKDB and CHECKTABLE, for the single database. (That is, of course, if rows do not already exist for that database.)
2. Enable and configure remote CheckDB in Minion.CheckDBSettingsDB.

Note: Each level of settings (that is, the default level, and each database level) should have one row for CHECKTABLE and one row for CHECKDB. For more information, see "Configuration Settings Hierarchy".

First, insert rows for CHECKDB and CHECKTABLE for the single database. For our example, we'll use DB1:

```
INSERT  INTO Minion.CheckDBSettingsDB
        ( DBName, OpLevel, OpName, Exclude, GroupOrder, GroupDBOrder, NoIndex,
          RepairOption, RepairOptionAgree, AllErrorMsgs, ExtendedLogicalChecks,
          NoInfoMsgs, IsTabLock, IsRemote, ResultMode, HistRetDays, LogSkips,
          BeginTime, EndTime, DayOfWeek, IsActive, Comment )
VALUES  ( N'DB1', 'DB', 'CHECKDB', 0, 0, 0, 0, 'LastMinionBackup', 1, 1, 0, 0, 0, 1,
          'Full', 60, 1, '00:00:00', '23:59:00', 'Daily', 1, 'DB1' ),
        ( N'DB1', 'DB', 'CHECKTABLE', 0, 0, 0, 0, 'LastMinionBackup', 1, 1, 0, 0, 0, 1,
            'Full', 60, 1, '00:00:00', '23:59:00', 'Daily', 1, 'DB1 CheckTable' );
```

You can use the stored procedure "Minion.CloneSettings" to easily generate a template insert statement.

Next, to enable and configure remote CheckDB for all databases, update the DB1 CHECKDB row in Minion.CheckDBSettingsDB:

```
UPDATE  Minion.CheckDBSettingsDB
SET     IsRemote = 1
     , IncludeRemoteInTimeLimit = 0
     , PreferredServer = 'YourRemoteSvr1'
     , PreferredServerPort = NULL
     , PreferredDBName = '%DBName%'
     , RemoteJobName = 'MinionCheckDB-%Server%_%DBName%'
     , RemoteCheckDBMode = 'Disconnected'
     , RemoteRestoreMode = 'LastMinionBackup'
     , DropRemoteDB = 1
     , DropRemoteJob = 1
WHERE   DBName = 'DB1'
        AND OpName = 'CHECKDB';
```

From now on, all CheckDB operations for database DB1 will be conducted on the remote server.

Scenario 3: Remote CheckDB for any database above a certain size

Minion CheckDB allows you to define thresholds to prevent smaller databases from taking part in remote CheckDB operations.

Here we will configure remote CheckDB operations for any database above 10 GB. The steps are:

1. Set IsRemote = 0, and configure remote CheckDB in Minion.CheckDBSettingsDB.
2. Configure the threshold in Minion.CheckDBSettingsRemoteThresholds.

Note: It may seem counterintuitive to turn IsRemote off, but it makes sense if you understand what that field is for. "IsRemote" turns on remote CheckDB for *all* databases (that the given row applies to). What we want is to handle remote operations dynamically, based on database size. So, we set IsRemote = 0 – meaning, "I want operations to be local *unless* a database crosses the threshold".

First, set IsRemote = 0, and configure remote CheckDB in Minion.CheckDBSettingsDB:

```
UPDATE  Minion.CheckDBSettingsDB
SET     IsRemote = 0  -- Important!
     , IncludeRemoteInTimeLimit = 0
     , PreferredServer = 'YourRemoteSvr1'
     , PreferredServerPort = NULL
     , PreferredDBName = '%DBName%'
     , RemoteJobName = 'MinionCheckDB-%Server%_%DBName%'
     , RemoteCheckDBMode = 'Disconnected'
     , RemoteRestoreMode = 'LastMinionBackup'
```

```
    , DropRemoteDB = 1
    , DropRemoteJob = 1
WHERE OpName = 'CHECKDB';
```

Last, configure the threshold in Minion.CheckDBSettingsRemoteThresholds. This table comes with a "MinionDefault" default row configured, so we can simply update and activate that:

```
UPDATE  Minion.CheckDBSettingsRemoteThresholds
SET     ThresholdValue = 10
    , IsActive = 1
WHERE   DBName = 'MinionDefault';
```

Scenario 4: Remote CheckDB for all databases, connected mode

Any of the above scenarios can use Connected mode or Disconnected mode. The difference is simply settings RemoteCheckDBMode = Connected:

```
UPDATE  Minion.CheckDBSettingsDB
SET     IsRemote = 1
    , IncludeRemoteInTimeLimit = 0
    , PreferredServer = 'YourRemoteSvr1'
    , PreferredServerPort = NULL
    , PreferredDBName = '%DBName%'
    , RemoteJobName = 'MinionCheckDB-%Server%_%DBName%'
    , RemoteCheckDBMode = 'Connected'
    , RemoteRestoreMode = 'LastMinionBackup'
    , DropRemoteDB = 1
    , DropRemoteJob = 1
WHERE OpName = 'CHECKDB';
```

For more on Disconnected and Connected modes, see:

- "About: Remote CheckDB"
- the discussion in "About: Minion CheckDB Operations"
- "Minion.CheckDBSettingsDB"

Scenario 5: Remote CheckDB for all databases, using third party restores

Any of the above scenarios can use third party restores instead of Minion Backup 1.3 restores. The difference is twofold: set RemoteRestoreMode = NONE, and be sure that an external process provides the database on the remote server (via restore, detatch/attach, etc.).

```
UPDATE  Minion.CheckDBSettingsDB
SET     IsRemote = 1
    , IncludeRemoteInTimeLimit = 0
    , PreferredServer = 'YourRemoteSvr1'
    , PreferredServerPort = NULL
    , PreferredDBName = '%DBName%'
    , RemoteJobName = 'MinionCheckDB-%Server%_%DBName%'
    , RemoteCheckDBMode = 'Connected'
    , RemoteRestoreMode = 'NONE'
    , DropRemoteDB = 0
    , DropRemoteJob = 1
WHERE OpName = 'CHECKDB';
```

Note: In this scenario, we have chosen RemoteRestoreMode = NONE. This does not require Minion Backup 1.3, but does require some outside process to restore the desired database to the remote server.

In this example we set DropRemoteDB = 0. In most situations where an external process is managing restores, that process also manages database retention. Of course, you should judge for your own situation whether it makes sense to keep or remove the database from the remote server after CheckDB.

For more on remote CheckDB modes, see:

- "About: Remote CheckDB"
- the discussion in "About: Minion CheckDB Operations"
- "Minion.CheckDBSettingsDB"

Scenario 6: Remote CheckDB for all databases, using new Minion Backup

Any of the above scenarios can use a new Minion Backup instead of an existing MB backup, or an external process restore. The process is:

1. Configure the remote CheckDB in Minion.CheckDBSettingsDB with RemoteRestoreMode = NewMinionBackup.
2. Configure a new row in Minion.BackupSettings with BackupType = CheckDB.
3. Modify the restore settings in Minion.BackupRestoreSettingsPath as needed.
4. Modify the thresholds in Minion.BackupRestoreTuningThresholds as needed.

First, configure the remote CheckDB:

```
UPDATE  Minion.CheckDBSettingsDB
SET     IsRemote = 1
    , IncludeRemoteInTimeLimit = 0
    , PreferredServer = 'YourRemoteSvr1'
    , PreferredServerPort = NULL
    , PreferredDBName = '%DBName%'
    , RemoteJobName = 'MinionCheckDB-%Server%_%DBName%'
    , RemoteCheckDBMode = 'Disconnected'
    , RemoteRestoreMode = 'NewMinionBackup'
    , DropRemoteDB = 1
    , DropRemoteJob = 1
WHERE OpName = 'CHECKDB';
```

Next, configure a new row in Minion.BackupSettings with BackupType = CheckDB:

```
INSERT  INTO Minion.BackupSettings
        ( [DBName]
        , [BackupType]
        , [Exclude]
        , [GroupOrder]
        , [GroupDBOrder]
        , [Mirror]
        , [DelFileBefore]
        , [DelFileBeforeAgree]
        , [PushToMinion]
        , [HistRetDays]
        , [DynamicTuning]
        , [Verify]
        , [ShrinkLogOnLogBackup]
        , [Encrypt]
        , [Checksum]
        , [Init]
        , [Format]
        , [CopyOnly]
        , [IsActive]
        , [Comment]
                    )
SELECT  'MinionDefault' AS [DBName]
        , 'CheckDB' AS [BackupType]  -- Important!
        , 0 AS [Exclude]
        , 50 AS [GroupOrder]
        , 0 AS [GroupDBOrder]
        , 0 AS [Mirror]
        , 0 AS [DelFileBefore]
        , 0 AS [DelFileBeforeAgree]
        , 'Local' AS [PushToMinion]
        , 30 AS [HistRetDays]
        , 1 AS [DynamicTuning]
        , '0' AS [Verify]
        , 0 AS [ShrinkLogOnLogBackup]
        , 0 AS [Encrypt]
        , 0 AS [Checksum]
        , 1 AS [Init]
        , 1 AS [Format]
        , 1 AS [CopyOnly] -- Optional
        , 1 AS [IsActive]
        , 'Settings for Minion CheckDB remote operations.' AS [Comment];
```

Modify the restore settings as needed:

```
UPDATE  Minion.BackupRestoreSettingsPath
SET     RestoreDrive = 'F:\'
```

```
        , RestorePath = 'SQLData\'
        , RestoreDBName = '%DBName%.%Date%'
WHERE   DBName = 'MinionDefault';
```

Finally, insert thresholds into Minion.BackupRestoreTuningThresholds as needed:

```
INSERT  INTO [Minion].BackupRestoreTuningThresholds
        ( [ServerName]
        , [DBName]
        , [RestoreType]
        , [SpaceType]
        , [ThresholdMeasure]
        , [ThresholdValue]
        , [Buffercount]
        , [MaxTransferSize]
        , [BlockSize]
        , [Replace]
        , [WithFlags]
        , [BeginTime]
        , [EndTime]
        , [DayOfWeek]
        , [IsActive]
        , [Comment]
        )
SELECT  'MinionDefault' AS [ServerName]
        , 'MinionDefault' AS [DBName]
        , 'All' AS [RestoreType]
        , 'DataAndIndex' AS [SpaceType]
        , 'GB' AS [ThresholdMeasure]
        , 0 AS [ThresholdValue]
        , 0 AS [Buffercount]
        , 0 AS [MaxTransferSize]
        , 0 AS [BlockSize]
        , 0 AS [Replace]
        , 0 AS [WithFlags]
        , '00:00:00' AS [BeginTime]
        , '23:59:59' AS [EndTime]
        , 'Daily' AS [DayOfWeek]
        , 1 AS [IsActive]
        , 'Zero level thresholds for all servers, all DBs.' AS [Comment]
UNION
SELECT  'MinionDefault' AS [ServerName]
        , 'MinionDefault' AS [DBName]
        , 'All' AS [RestoreType]
        , 'DataAndIndex' AS [SpaceType]
        , 'GB' AS [ThresholdMeasure]
        , 10 AS [ThresholdValue]
        , 30 AS [Buffercount]
        , 1048576 AS [MaxTransferSize]
        , 0 AS [BlockSize]
        , 0 AS [Replace]
        , 0 AS [WithFlags]
        , '00:00:00' AS [BeginTime]
        , '23:59:59' AS [EndTime]
        , 'Daily' AS [DayOfWeek]
        , 1 AS [IsActive]
        , '10GB thresholds for all servers, all DBs.' AS [Comment];
```

Note: You can set your CheckDB backups as Copy Only backups, so you don't interfere with the normal run of backups.

How to: Limit operations by time

You can limit integrity check operations by time in one of two ways: by passing in the time limit as a parameter to Minion.CheckDBMaster, or by using timed rotations.

Operation run time estimates are calculated based on past operations, per database. If a database has never had an integrity check through Minion CheckDB, the system uses the **DefaultTimeEstimateMins** field in the Minion.CheckDBSettingsDB table.

Limit time by parameter

Minion.CheckDBMaster has a @TimeLimitInMins parameter that applies to both CHECKDB and CHECKTABLE operations.

IMPORTANT: If you run the procedure with the @TimeLimitInMins parameter set, it trumps any other time limit setting, including timed rotations.

To run DBCC CheckDB for all user databases, and limit the run to 120 minutes, execute Minion.CheckDBMaster with @TimeLimitInMins = 120:

```
EXEC Minion.CheckDBMaster @DBType = 'User'
        , @OpName = 'CHECKDB'
        , @StmtOnly = 0
        , @ReadOnly = 1
        , @TimeLimitInMins = 120;
```

Limit time using timed rotations

Enter in a timed CheckDB row for the time limitation you want.

If you want a time rotation, you not only need the value in the Minion.CheckDBSettingsRotation table, but you also need to set the TimeLimit param to 0 or NULL.

IMPORTANT: This is an experimental feature; test first and use with caution.

How to: Configure Custom Snapshots

When you run DBCC CheckDB or DBCC CheckTable, behind the scenes SQL Server creates a snapshot of the database to run the operation against. SQL Server decides where to place the files for these snapshots, and deletes the snapshot after the operation is complete.

If your version of SQL Server supports it, you can also choose to create a custom snapshot. You might want to do this if your operation takes long enough that the internal snapshot would grow too large (and risk filling up the drive), which would stop the operation.

Note: SQL Server 2016 and earlier versions only allow custom snapshots for Enterprise edition. SQL Server 2016 SP1 allow custom snapshots in any edition.

For CheckDB, custom snapshots allow you to determine where the snapshot file(s) will be located. For CheckTable, custom snapshots allow you both to set the file locations, *and* to drop and recreate the snapshot every few minutes (which we call "custom dynamic snapshots"). What follows are a few scenarios that cover both custom snapshots, and custom dynamic snapshots.

For more information, see the section "About: Custom Snapshots", and the video "Custom Snapshot Basics" on YouTube: https://youtu.be/0PVFXm6KDr0

Scenario 1: Custom snapshots for all operations

To configure custom snapshots for all databases:

1. **Enable custom snapshots:** Update the two MinionDefault rows in Minion.CheckDBSettingsSnapshot, with CustomSnapshot=1.
2. **Configure paths:** Configure the snapshot file location(s) in Minion.CheckDBSnapshotPath.

First, we update Minion.ChekDBSettingsSnapshot. Minion CheckDB comes with two "MinionDefault" rows in this table – one for CHECKDB and one for CHECKTABLE – both with CustomSnapshot = 0. These are example rows so you can easily enable custom snapshots:

```
UPDATE  Minion.CheckDBSettingsSnapshot
SET     CustomSnapshot = 1
      , DeleteFinalSnapshot = 1
      , IsActive = 1
WHERE   DBName = 'MinionDefault';
```

Note: We strongly recommend you review the settings available in the Minion.CheckDBSettingsSnapshot table and configure them as needed. In the example above, we have simply enabled custom snapshots and configured the system to delete the custom snapshot after operations are complete.

Then, we update the MinionDefault rows in Minion.CheckDBSnapshotPath:

```
UPDATE  Minion.CheckDBSnapshotPath
SET     SnapshotDrive = 'D:\'
```

```
        , SnapshotPath = 'SQLSnapshots\'
        , IsActive = 1
WHERE   DBName = 'MinionDefault';
```

Note that the rows with DBName = 'MinionDefault' also have FileName = 'MinionDefault', meaning that the settings in these rows apply to all databases, and to all files within a database. See the section "Scenario 4: Multi file custom snapshots" below for more on multi file custom snapshots.

From this point on, custom snapshots will be created on the D: drive for all databases, and you can see a record of local snapshot files in Minion.CheckDBSnapshotLog.

Scenario 2: Custom snapshots for a single database

To configure a custom snapshot for a database:

1. **Enable custom snapshots:** Enter a row into Minion.CheckDBSettingsSnapshot for that database, with CustomSnapshot=1. (Actually, you need to enter two such rows: one for CheckDB, and one for CheckTable.)
2. **Configure paths:** Configure the snapshot file location(s) in Minion.CheckDBSnapshotPath.

For example, to configure custom snapshots for the DB1 database, we first insert rows to the Minion.CheckDBSettingsSnapshot table:

```
INSERT  INTO Minion.CheckDBSettingsSnapshot
        ( DBName, OpName, CustomSnapshot, SnapshotRetMins,
          SnapshotRetDeviation, DeleteFinalSnapshot, IsActive, Comment )
VALUES  ( 'DB1'  -- DBName
          , 'CHECKTABLE'  -- OpName
          , 1  -- CustomSnapshot
          , 1  -- SnapshotRetMins: This will drop/recreate the snapshot every 1 minute.
          , 1  -- SnapshotRetDeviation
          , 1  -- DeleteFinalSnapshot
          , 1  -- IsActive
          , 'DB1 custom snapshot'  -- Comment
          ),
        ( 'DB1'  -- DBName
          , 'CHECKDB'  -- OpName
          , 1  -- CustomSnapshot
          , 0 -- SnapshotRetMins
          , 1  -- SnapshotRetDeviation
          , 1  -- DeleteFinalSnapshot
          , 1  -- IsActive
          , 'DB1 custom snapshot'  -- Comment
          );
```

From here, we can either rely on the MinionDefault rows in Minion.CheckDBSnapshotPath, or we can insert custom rows for DB1:

```
INSERT  INTO Minion.CheckDBSnapshotPath
```

```
        ( DBName, OpName, FileName, SnapshotDrive, SnapshotPath, ServerLabel,
         PathOrder, IsActive, Comment )
VALUES  ( 'DB1'  -- DBName
         , 'CHECKTABLE'  -- OpName
         , 'DB1Snapshot'  -- FileName
         , '\\share1\'  -- SnapshotDrive
         , 'SnapshotCheckDB\'  -- SnapshotPath
         , NULL  -- ServerLabel
         , 0  -- PathOrder
         , 1  -- IsActive
         , 'DB1 snapshot path'  -- Comment
         ),
        ( 'DB1'  -- DBName
         , 'CHECKDB'  -- OpName
         , 'DB1Snapshot'  -- FileName
         , '\\share1\'  -- SnapshotDrive
         , 'SnapshotCheckDB\'  -- SnapshotPath
         , NULL  -- ServerLabel
         , 0  -- PathOrder
         , 1  -- IsActive
         , 'DB1 snapshot path'  -- Comment
         );
```

Scenario 3: Custom dynamic snapshots for a single database

The only difference between custom snapshots for CheckTable, and "rotating" custom dynamic snapshots for CheckTable – those that drop and recreate every few minutes – is that a rotating snapshot has "SnapshotRetMins" set to a value greater than zero.

To configure this, follow the directions from Scenario 1 or Scenario 2, above, adding "SnapshotRetMins = 60" to the Minion.CheckDBSettingsSnapshot insert statement.

Discussion - features:

- You can have a drive for each file, or put them all onto a single drive.
- You can override just one file location if you need. Just put that filename into the Path table and leave the rest at 'MinionDefault'.
- If you have several database files, and only one override for a specific filename, *and* no MinionDefault row then you'll be in trouble.

For more information, see the section "About: Custom Snapshots", and the video "Custom Snapshot for CheckTable" on YouTube: https://youtu.be/1wda8fYBVk4

Scenario 4: Multi file custom snapshots

To configure custom snapshots for all databases:

1. **Enable custom snapshots:** Update the two MinionDefault rows in Minion.CheckDBSettingsSnapshot, with CustomSnapshot=1.

2. **Configure paths:** Configure multiple snapshot file location(s) in Minion.CheckDBSnapshotPath.

First, update Minion.ChekDBSettingsSnapshot to enable custom snapshots:

```
UPDATE  Minion.CheckDBSettingsSnapshot
SET     CustomSnapshot = 1
      , DeleteFinalSnapshot = 1
      , IsActive = 1
WHERE   DBName = 'MinionDefault';
```

Note: We strongly recommend you review the settings available in the Minion.CheckDBSettingsSnapshot table and configure them as needed. In the example above, we have simply enabled custom snapshots and configured the system to delete the custom snapshot after operations are complete.

Then, insert rows to Minion.CheckDBSnapshotPath for the specific database and files. We want to configure DB1, and so we insert a row for file DB1_1, file DB1_2, and all other files (FileName=MinionDefault):

```
INSERT  INTO Minion.CheckDBSnapshotPath
        ( [DBName]
        , [OpName]
        , [FileName]
        , [SnapshotDrive]
        , [SnapshotPath]
        , [PathOrder]
        , [IsActive]
        , [Comment]
        )
SELECT  'DB1' AS [DBName]
        , 'CHECKTABLE' AS [OpName]
        , 'MinionDefault' AS [FileName]
        , 'D:\' AS [SnapshotDrive]
        , 'SnapshotFiles\' AS [SnapshotPath]
        , 0 AS [PathOrder]
        , 1 AS [IsActive]
        , 'DB1 default' AS [Comment]
UNION
SELECT  'DB1' AS [DBName]
        , 'CHECKTABLE' AS [OpName]
        , 'DB1_1' AS [FileName]
        , 'F:\' AS [SnapshotDrive]
        , 'SnapshotFilesDB1\' AS [SnapshotPath]
        , 0 AS [PathOrder]
        , 1 AS [IsActive]
        , 'DB1 file1' AS [Comment]
UNION
SELECT  'DB1' AS [DBName]
        , 'CHECKTABLE' AS [OpName]
        , 'DB1_2' AS [FileName]
        , 'G:\' AS [SnapshotDrive]
        , 'SnapshotFilesDB1\' AS [SnapshotPath]
        , 0 AS [PathOrder]
        , 1 AS [IsActive]
        , 'DB1 file2' AS [Comment];
```

For more information, see the video "Custom Snapshot for Multiple Files" on YouTube: https://youtu.be/Le43dzFBOVM

How to: Test schedules

If you use Minion.CheckDBMaster with the @TestDataTime parameter, it should return the ID of the SettingsServer row that's applicable. This allows you to make sure your schedules are set up correctly.

If Minion.CheckDBMaster does not return the ID of the row, it either means there is not a row that applies to that date and time, or that the CurrentNumOps = MaxForTimeFrame for the applicable row (meaning, Minion CheckDB thinks that nothing should happen, because the maximum number of operations for that row has been completed already for the given timeframe).

IMPORTANT: To ONLY run the test, and not the actual operations, run with @StmtOnly = 1. For example:

```
EXEC Minion.CheckDBMaster
   @StmtOnly = 1,
   @TestDateTime = '2017-09-28 18:00';
```

Troubleshooting

Databases without tables

In Minion CheckDB, only actual work done is logged. If you run CheckTable on a database that doesn't have any tables, it will run and nothing will error out, but nothing will be logged.

If you want the operation to be logged for that database, switch to CheckDB instead. Note that this is what dynamic tuning is for. (MC will detect that the database is not big enough for CheckTable.)

For more information, see "How to: Configure Minion CheckDB Dynamic Thresholds".

Database that does not exist

If you have an issue with it trying to run MC against a database that does not exist, it's probably because a database was dropped and was somehow still left over in the Minion.CheckDBThreadQueue table. The process tries to clean up after itself, but if the routine is stopped midway for some reason then it won't have the chance to do that. If the database had already been processed, then it may show up again in the list and generate an error. This can cause the jobs to fail, and need to be restarted.

If this happens, empty the Minion.CheckDBThreadQueue work table (**if** you're not running any other CheckDB runs at the moment).

Processing some databases but not others

If you are only processing some of the databases, check the following:

- Are there database exclusions in Minion.CheckDBSettingsDB (Exclude = 1)?
- Are there databases that are offline?
- Are all the databases covered with active (IsActive=1) settings in Minion.CheckDBSettingsDB? Make sure to check the day, BeginTime, and EndTime fields, too.
- Is there an active rotation setting in Minion.CheckDBSettingsRotation table? Set IsActive = 0, and see if that fixes it.
- Is it possible that some operations are maxing out the server resources? Check the Minion.CheckDBSettingsDB column **DBInternalThreads**. If you're running multiple databases simultaneously with a high number of threads each, it *could* under take up too many resources to complete. Try lowering the number of databases run at the same time, or the number of threads used, or both.
- Have you attempted to mix incompatible features for some databases? Check out the "About: Feature Compatibility" section for more information.

Not processing the include/exclude as expected

Minion CheckDB has enough options for including and excluding databases and tables to/from operations, it can get complicated. If you're working on including or excluding objects from operations, and it's not going the way you expect, this is the section for you.

This would be a *very* big troubleshooting section, so we will keep it to a summary for now.

Subjects to review

- Configuration Settings Hierarchy - Configuration for integrity check operations is stored in tables. A default row (DBName='MinionDefault') in the main settings table provides settings for any database that doesn't have its own specific settings. This is a hierarchy of granularity, where more specific configuration levels completely override the less specific levels.
- Database Include and Exclude Precedence – Minion CheckDB allows you to specify lists of databases to include in a CheckDB/CheckTable routine, in a couple of different ways.
- Table Include and Exclude Precedence – Minion CheckDB allows you to specify lists of tables to include in a DBCC CheckTable routine.
- About: Feature Compatibility – It's possible that incompatible features are interfering with which objcts are processed.

More notes

If you're running Minion.CheckDBMaster with some combination of Include, Exclude, Schema, and Tables, you may not get the behavior you're expecting. Let's take an example:

```
EXEC Minion.CheckDBMaster @DBType = 'User',
   @OpName = 'CHECKTABLE',
   @StmtOnly = 0,
   @ReadOnly = 1,
   @Schemas = N'A,B',
   @Tables = N'T1,T2',
   @Include = N'DB100';
```

What you might expect from this is for MC to check A.T1, A.T2, B.T1, and B.T2 in database DB100. What you need to know is that @Schemas and @Tables are *complimentary*, not co-limiting. In other words, this statement tells MC to run CheckTables on:

- All tables in schema A, in database DB100.
- All tables in schema B, in database DB100.
- Table T1 (dbo.T1), in database DB100.
- Table T2 (dbo.T2), in database DB100.

So, if you want MC to check just the tables A.T1, A.T2, B.T1, and B.T2 in database DB100, you should run this (or the table-based equivalent):

```
EXEC Minion.CheckDBMaster @DBType = 'User',
   @OpName = 'CHECKTABLE',
   @StmtOnly = 0,
   @ReadOnly = 1,
   @Schemas = NULL,
   @Tables = N'A.T1, A.T2, B.T1, B.T2',
   @Include = N'DB100';
```

Time limit is not respected

There are a few things that could make a job run over its time limit. While MC tries to calculate timing as well as possible, it's still just an estimate. There are factors that can make it go over:

- **Database size** – If the database is much bigger than it was before, calculating the estimated time accurately can be difficult.
- **Resources** – If the box is far busier than it was during the last operation, it may not have the resources it did before. That can make it take longer than expected.
- **Configuration changes** – For example, if you move the snapshot to a slower disk, or if more databases are running on the same disk then that could slow things down.
- **Lots of errors** – The more errors that CheckDB finds, the slower it goes. It could take considerably longer than expected.
- **Different threading model (single or multi-threaded)** – Even if the resources on the box don't change, the most recent operation could be running with a different threading model than the time before.

We've documented how we calculate the time estimate so you can see that it's not a perfunctory number.

For more information about time limits, see "How to: limit operations by time".

Estimated time differs from actual time

Why is the estimated time so different from the actual time an operation takes? A lot of it has to do with the answer in the "Time limit is not respected" section.

Additionally, if the database has never had an integrity check operation in Minion CheckDB before, then there's a default time limit you can use to estimate the time. You can configure this in Minion.CheckDBSettingsDB in the **DefaultTimeEstimateMins** column. Use it to get a better initial estimate based on what you know about your environment.

Remote CheckDB isn't working

If remote CheckDB is not working, check the following:

- Check the requirements list in "About: Remote CheckDB".

- Are you attempting Remote CheckDB for CHECKTABLE operations? Remote CheckDB does not support CheckTable. See "About: Feature Compatibility".
- Is the remote SQL Agent on?
- Try setting MC to keep the remote job (DropRemoteJob = 0) and rerun, so you can look at any errors.
- Check the restore statement in the remote job, to see if there is something wrong with the statement itself.
- Do you have an encrypted backup and you have not restored the certificate?
- Is DropRemoteJob = 0? If you have configured MC to keep jobs after the operation is complete, and the job has a static name, the remote CheckDB will fail (because the job is already there). Delete MC jobs on the remote server, set DropRemoteJob = 1, and try again.
- Are you attempting to restore over an existing database without Replace = 1 (Minion.BackupRestoreTuningThresholds)?
- Did the last remote CheckDB fail? Again, the next attempt may be trying to create a job name that already exists. Delete MC jobs on the remote server and try again.

Database snapshots aren't being deleted

If custom snapshots aren't being deleted, check the following:

- Are the operations completing? If not, you'll have to delete the snapshots manually (and figure out why the operations are erroring out).
- Have you configured the operations to delete snapshots? If you want snapshots to be deleted automatically, check the appropriate row in Minion.CheckDBSettingsSnapshot; the column **DeleteFinalSnapshot** should be set to 1.

Custom snapshots fail

If you have enabled custom snapshots in Minion.CheckDBSettingsSnapshot (by setting CustomSnapshot = 1), but suspect they might not be working properly, check the following:

- **Custom snapshots enabled** – Make sure the applicable row(s) in Minion.CheckDBSettingsSnapshot actually have CustomSnapshot = 1, and are active (IsActive=1).
- **Paths configured** – Make sure that rows are configured in Minion.CheckDBSnapshotPath, and that the rows are active.
- **SQL version** – It's possible that your version of SQL Server doesn't support it. In this case, if everything is configured correctly, the "custom snapshot" integrity check operations will complete using the default internal snapshot.
- **Incompatible custom snapshots** – Are you attempting custom *dynamic* snapshots for DBCC CheckDB Operations? (SnapshotRetMins > 0.) This is an incompatible feature for CheckDB. For more information, see "About: Feature Compatibility".
- **Trying multithreading with custom dynamic** – Custom dynamic snapshots for CheckTable are only available for *single-threaded operations*. This means that you must set DBInternalThreads in Minion.CheckDBSettingsDB, and DBInternalThreads in Minion.CheckDBSettingsServer, to 1 for custom dynamic snapshots.

This last "failure" will show up in the log (Minion.CheckDBLogDetails) as follows: DBName and CheckDBName will be the same, and CustomSnapshot = 1.

```
SELECT  ExecutionDateTime
      , DBName
      , CheckDBName
      , CustomSnapshot
FROM    Minion.CheckDBLogDetailsCurrent;
```

ExecutionDateTime	DBName	CheckDBName	CustomSnapshot
2016-12-16 10:12:49.227	DB1	DB1	1
2016-12-16 10:12:49.227	DB2	DB2	1
2016-12-16 10:12:49.227	DB3	DB3	1

If the custom snapshot had worked properly, we would have seen a different name for CheckDBName – the name of the snapshot database created.

Minion.CheckDBMaster @TestDateTime does not work

If you use Minion.CheckDBMaster with the @TestDataTime parameter, it should return the ID of the SettingsServer row that's applicable. Make sure your schedule is right.

If Minion.CheckDBMaster does not return the ID of the row, it either means there is not a row that applies to that date and time, or that the CurrentNumOps = MaxForTimeFrame for the applicable row (meaning, Minion CheckDB thinks that nothing should happen, because the maximum number of operations for that row has been completed already for the given timeframe).

IMPORTANT: To ONLY run the test, and not the actual operations, run with @StmtOnly = 1. For example:
EXEC Minion.CheckDBMaster @StmtOnly = 1, @TestDateTime = '2016-09-28 18:00';

Check the table Minion.CheckDBSettingsServer:

- **Are there active duplicate rows?** If you have defined the same Day, BeginTime, and EndTime for an operation, it's possible you won't get the schedule you expect.

For the settings tables **Minion.CheckDBSettingsDB**, **Minion.CheckDBSettingsTable**, and **Minion.CheckDBSettingsServer**, check that these fields have applicable values:

- **BeginTime, EndTime** – Perhaps your test time falls outside the defined window of time
- **Day** – Does your test time fall on a day that's not defined?
- **Include** – Do you have settings that apply to the given database?

For the settings table **Minion.CheckDBSettingsServer**, check that these fields have applicable values:

- **Exclude** – Is your given database excluded?
- **IsActive** – Is the appropriate row active?

For the settings table **Minion.CheckDBSettingsDB**, check that these fields have applicable values:

- **DBName** – There should be an active row with DBName = 'MinionDefault' and OpName = 'CHECKDB'; and an active row with DBName = 'MinionDefault' and OpName = 'CHECKTABLE'.
- **Exclude** – Is the row marked Exclude=1?

For the settings table **Minion.CheckDBSettingsTable**, check that these fields have applicable values:

- **Exclude** – Is the row marked Exclude=1?

Inline Token is not recognized

If you're trying to use an inline token and it doesn't work, try these steps:

- Check the table "Minion.DBMaintInlineTokens" for spelling and IsActive=1.
- For default tokens, check that IsCustom in "Minion.DBMaintInlineTokens" is 0.
- For default tokens, check that you're using percent sign delimiters, e.g. '%ServerName%'.
- For custom tokens, check that IsCustom in "Minion.DBMaintInlineTokens" is 1.
- For custom tokens, check that you're using pipe delimiters, e.g. '|MyCustomToken|'.
- Test the token definition code to be sure it's usable.
- Note that custom inline tokens can't use internal variables (such as @ExecutionDateTime) like the built-in tokens can. Custom inline tokens can only use SQL functions and @@variables.

Revisions

Version	Release Date	Changes
1.0	February 2017	Initial release.

FAQ

Why isn't the Data Waiter part of Minion CheckDB?

Each database in a high availability scenario (like Availability Groups, or replication, etc.) is a separate entity, as far as integrity checks are concerned. Corruption could occur on one node of an AG, and it may not translate to corruption on other nodes of the AG.

Running CheckDB on a secondary node for DB1 does not directly equate to running CheckDB on the primary node for DB1.

In short, right now we don't see a huge need for sharing MC settings across nodes. I love you. However, if enough people need it, we've been known to change our minds.

About Us

Minion by MidnightDBA is a creation of Jen and Sean McCown, owners of MinionWare, LLC and MidnightSQL Consulting, LLC.

We formed **MinionWare**, LLC to create **Minion Enterprise**: an enterprise management solution for centralized SQL Server management and alerting. This solution allows your database administrator to manage an enterprise of one, hundreds, or even thousands of SQL Servers from one central location. Minion Enterprise provides not just alerting and reporting, but backups, maintenance, configuration, and enforcement. **Go to www.MinionWare.net for details and to request a free 90 day trial.**

In our "**MidnightSQL"** consulting work, we perform a full range of databases services that revolve around SQL Server. We've got over 30 years of experience between us and we've seen and done almost everything there is to do. We have two decades of experience managing large enterprises, and we bring that straight to you. Take a look at www.MidnightSQL.com for more information on what we can do for you and your databases.

Under the "**MidnightDBA"** banner, we make free technology tutorials, blogs, and a live weekly webshow (DBAs@Midnight). We cover various aspects of SQL Server and PowerShell, technology news, and whatever else strikes our fancy. You'll also find recordings of our classes – we speak at user groups and conferences internationally – and of our webshow. Check all of that out at www.MidnightDBA.com

We are both "MidnightDBA" and "MidnightSQL"…the terms are nearly interchangeable, but we tend to keep all of our free stuff under the MidnightDBA banner, and paid services under MidnightSQL Consulting, LLC. Feel free to call us the MidnightDBAs, those MidnightSQL guys, or just "Sean" and "Jen". We're all good.

Table of Contents

RESOURCES

Home	http://MinionWare.net
Tutorials	http://youtube.com/MidnightDBA
Support	https://minionware.desk.com/
Scripts by users	http://MinionWare.net/CommunityZone/
Sales	MinionWareSales@MidnightDBA.com
Twitter	https://Twitter.com/HeyMinionWare
Facebook	https://www.Facebook.com/MinionWare