# MINION REINDEX: QUICK START

**Minion Reindex by MidnightDBA** is a stand-alone index maintenance solution that can be deployed on any number of servers, for free. Minion Reindex 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 Reindex by MidnightDBA ("Minion Reindex"), its uses, features, moving parts, and examples.

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

*Minion Reindex is one module of the Minion suite of products.*


## Quick Start

To install, download Minion Reindex from www.MinionWare.net/Reindex and run it on your target server. For simplicity, this Quick Start guide assumes that you have installed Minion Reindex on one server, named "YourServer".

**Note:** You can also use the PowerShell script provided with the download to install Minion Reindex on dozens or hundreds of servers at once, just as easily as you would install it on a single instance.

System requirements:

- SQL Server 2005 or above.
- The sp_configure setting **xp_cmdshell** must be enabled*.
- PowerShell 2.0 or above; execution policy set to **RemoteSigned**.

Once MinionReindexing.sql has been run, nothing else is required. From here on, Minion Reindex will run nightly to defragment **all** non-TempDB databases. The reindexing 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.*

## Change Schedules

Optionally, you can change the reindexing schedules:

1. **View jobs:** Connect to "YourServer" and expand the SQL Agent node. You'll see two new jobs:
   - *MinionReindexDBs-All-All* – Runs once weekly – Fridays at 3:00 AM - to thoroughly defragment indexes (rebuild).
   - *MinionReindexDBs-All-REORG* – Runs Daily – 3:00 AM except for Friday – to complete lightweight defragmenting (reorganize).
2. **Alter schedules:** Edit the two job schedules to fit your company's needs.

## Change Default Settings

Minion Reindex stores default settings for the entire instance in a single row (where DBName='MinionDefault') in the Minion.IndexSettingsDB table.

**Warning**: Do not delete the MinionDefault row from Minion.IndexSettingsDB!

To change the default settings, run an update statement on the MinionDefault row in Minion.IndexSettingsDB. For example:

```
UPDATE [Minion].[IndexSettingsDB]
  SET [Exclude] = 0
    ,[ReindexGroupOrder] = 0
    ,[ReindexOrder] = 0
    ,[ReorgThreshold] = 10
    ,[RebuildThreshold] = 20
    ,[FILLFACTOROpt] = 85
    ,[PadIndex] = 'ON'
    ,[SortInTempDB] = 'OFF'
    ,[DataCompression] = NULL
    ,[GetRowCT] = 1
    ,[GetPostFragLevel] = 1
    ,[UpdateStatsOnDefrag] = 1
    ,[LogIndexPhysicalStats] = 0
    ,[IndexScanMode] = 'Limited'
    ,[LogProgress] = 1
    ,[LogRetDays] = 60
    ,[LogLoc] = 'Local'
    ,[MinionTriggerPath] = '\\minioncon\c$'
    ,[IncludeUsageDetails] = 1
 WHERE [DBName] = 'MinionDefault';
```

**Warning:** Choose your settings wisely; these settings can have a massive impact on your system. For example, if you have a 500 GB database with fill factor set to 100, changing fill factor to 85 could increase the size of your database massively on the next reindex.

For more information on these settings, see the “Minion.IndexSettingsDB” section.

For instructions on setting database-level or table-level settings, see the section titled “How To: Configure settings for a single database”.

# MINION REINDEX

## Contents in Brief



## Top 10 Features

**Minion Reindex by MidnightDBA** is a stand-alone index maintenance module. Once installed, Minion Reindex automatically maintains all online databases on the SQL Server instance, and will automatically incorporate databases and indexes as they are added or removed.

Ten of the very best features of Minion Reindex are, in a nutshell:

1. **Automated operation** – Run the Minion Reindex installation scripts, and it just goes.
2. **Easy mass installation** – Install Minion Reindex on hundreds of servers as easily as you can on one.
3. **Granular configuration without extra jobs** – Configure extensive settings at the default, database, and/or table levels with ease.
4. **Database and table reindex ordering** – Reindex databases *and tables* in exactly the order you need.
5. **Flexible include and exclude** – Reindex only the databases you want, using specific database names, LIKE expressions, and even regular expressions.
6. **Live Insight** – See what Minion Reindex is doing every step of the way, and how much further it has to go.
7. **Maximized maintenance window** – Spend the whole maintenance window on index maintenance, not on gathering fragmentation stats.
8. **Extensive, useful logging** – Use the Minion Reindex log for estimating the end of the current reindexing run, troubleshooting, planning, and reporting.
9. **Built in manual runs** – Choose to only print reindex statements, and run them individually as needed.

10. **Integrated help** – Get help on any Minion Reindex object without leaving Management Studio.

For more information on these, additional features and settings, and How To topics, see the sections **"How To" Topics**, and **Moving Parts**. For links to downloads, tutorials and articles, see www.MinionWare.net.


### *Minion Enterprise Hint*

Minion Enterprise (ME) is our 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. ME provides not just alerting and reporting, but backups, maintenance, configuration, and enforcement. ME integrates with Minion Reindex.

See www.MinionWare.net for more information, or
email us today at Support@MidnightDBA.com for a demo!

# Architecture Overview

Minion Reindex is made up of SQL Server stored procedures, tables, and jobs. There is an optional PowerShell script for installation. The tables store configuration and log information; stored procedures perform reindex operations; and the jobs execute those index operations on a schedule.

**Note**: Minion 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 change the database that the jobs point to.

## Configuration Settings Hierarchy

As much as possible, configuration for reindex is stored in tables: Minion.IndexSettingsDB and Minion.IndexSettingsTable. A default row in Minion.IndexSettingsDB (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 into Minion.IndexSettingsDB, and that row will override ALL of the default settings for that database.
- Insert a row for a specific table in Minion.IndexSettingsTable, and that row will override ALL of the default (or, if available, database-specific) settings for that 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.

# Run Time Configuration

The main Minion Reindex stored procedure – Minion.IndexMaintMaster – takes a number of parameters that are specific to the current maintenance run. For example:

- Use @IndexOption to run index maintenance on only tables marked for ONLINE index maintenance.
- Use @PrepOnly to only gather index fragmentation stats. These are saved to a table, so that later you can run Minion.IndexMaintMaster using @RunPrepped, and the procedure will used the saved fragmentation stats (instead of gathering them anew).
- Use @Include to run index maintenance 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.

# Logging

As a Minion Reindex routine runs, it keeps logs of all activity in two tables:

- Minion.IndexMaintLog – a log of activity at the database level.
- Minion.IndexMaintLogDetail – a log of activity at the index level.

The Status column for the current run is updated continually in each of these tables. This way, status information (**Live Insight**) is available to you while index maintenance is still running, and historical data is available after the fact for help in planning future operations, reporting, troubleshooting, and more.

# “How To” Topics

## How To: Configure settings for a single database

Default settings for the whole system are stored in the Minion.IndexSettingsDB table. To specify settings for a specific database that override those defaults (for that database), insert a row for that database to the Minion.IndexSettingsDB table. For example:

```
INSERT  INTO [Minion].[IndexSettingsDB]
                ( DBName ,
                  [Exclude] ,
                  [ReindexGroupOrder] ,
                  [ReindexOrder] ,
                  [ReorgThreshold] ,
                  [RebuildThreshold] ,
                  [FILLFACTORopt] ,
                  [PadIndex] ,
                  [SortInTempDB] ,
                  [DataCompression] ,
                  [GetRowCT] ,
                  [GetPostFragLevel] ,
                  [UpdateStatsOnDefrag] ,
                  [LogIndexPhysicalStats] ,
                  [IndexScanMode] ,
                  [LogProgress] ,
                  [LogRetDays] ,
                  [LogLoc] ,
                  [MinionTriggerPath] ,
                  [IncludeUsageDetails]
                )
VALUES ( 'YourDatabase' ,       --      DBName ,
                  0 ,           --      Exclude ,
                  0 ,           --      ReindexGroupOrder ,
                  0 ,           --      ReindexOrder ,
                  10 ,          --      ReorgThreshold ,
                  20 ,          --      RebuildThreshold ,
                  80 ,          --      FILLFACTORopt ,
                  'ON' ,        --      PadIndex ,
                  'OFF' ,       --      SortInTempDB ,
                  NULL ,        --      DataCompression ,
                  1 ,           --      GetRowCT ,
                  1 ,           --      GetPostFragLevel ,
                  1 ,           --      UpdateStatsOnDefrag ,
                  0 ,           --      LogIndexPhysicalStats ,
                  'Limited' ,   --      IndexScanMode ,
                  1 ,           --      LogProgress ,
                  60 ,          --      LogRetDays ,
                  'Local' ,     --      LogLoc ,
                  '\\minioncon\c$' ,    --      MinionTriggerPath ,
```

```
        1               --      IncludeUsageDetails
    );
```

## How To: Configure settings for a single table

Default settings are stored in the Minion.IndexSettingsDB table. To specify settings for a specific table that override those defaults (for that table), insert a row for that table to the Minion.IndexSettingsTable table. For example:

```
INSERT  INTO [Minion].[IndexSettingsTable]
                ( [DBName] ,
                  [SchemaName] ,
                  [TableName] ,
                  [Exclude] ,
                  [ReindexGroupOrder] ,
                  [ReindexOrder] ,
                  [ReorgThreshold] ,
                  [RebuildThreshold] ,
                  [FILLFACTORopt] ,
                  [PadIndex] ,
                  [ONLINEopt] ,
                  [SortInTempDB] ,
                  [DataCompression] ,
                  [GetRowCT] ,
                  [GetPostFragLevel] ,
                  [UpdateStatsOnDefrag] ,
                  [LogIndexPhysicalStats] ,
                  [IndexScanMode] ,
                  [LogProgress] ,
                  [LogRetDays] ,
                  [IncludeUsageDetails]

                )
VALUES ( 'YourDatabase' , -- DBName
         'dbo' ,          -- SchemaName
         'YourTable' ,    -- TableName
         0 ,              -- Exclude
         0 ,              -- ReindexGroupOrder
         0 ,              -- ReindexOrder
         10 ,             -- ReorgThreshold
         20 ,             -- RebuildThreshold
         80 ,             -- FILLFACTORopt
         'ON' ,           -- PadIndex
         NULL ,           -- ONLINEopt
         NULL ,           -- SortInTempDB
         NULL ,           -- DataCompression
         1 ,              -- GetRowCT
         1 ,              -- GetPostFragLevel
         1 ,              -- UpdateStatsOnDefrag
```

```
    0 ,             -- LogIndexPhysicalStats
    'Limited' ,     -- IndexScanMode
    1 ,             -- LogProgress
    60 ,            -- LogRetDays
    1               -- IncludeUsageDetails
);
```

## How To: Reindex databases in a specific order

You can choose the order in which databases will be maintained. For example, let's say that you want your databases to be indexed 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 row into the Minion.IndexSettingsDB for each one of the databases, specifying either ReindexGroupOrder, ReindexOrder, or both, as needed.

> **NOTE**: For ReindexGroupOrder and ReindexOrder, higher numbers have a greater "weight" - they have a higher priority - and will be indexed 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 index maintenance settings for that database. So, inserting a row for [YourDatabase] means that ONLY index 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 indexed according to the values in the MinionDefault columns *ReindexGroupOrder* and *ReindexOrder*. By default, these are both 0 (lowest priority), and so non-specified databases would be maintained 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:

```
-- Insert IndexSettingsDB row for [YourDatabase], ReindexOrder=255 (first)
INSERT  INTO [Minion].[IndexSettingsDB]
                    ( DBName ,
                      [Exclude] ,
                      [ReindexGroupOrder] ,
                      [ReindexOrder] ,
                      [ReorgThreshold] ,
                      [RebuildThreshold] ,
                      [FILLFACTORopt] ,
```

```
                    [PadIndex] ,
                    [SortInTempDB] ,
                    [GetRowCT] ,
                    [GetPostFragLevel] ,
                    [UpdateStatsOnDefrag] ,
                    [LogIndexPhysicalStats] ,
                    [IndexScanMode] ,
                    [LogProgress] ,
                    [LogRetDays] ,
                    [LogLoc] ,
                    [MinionTriggerPath] ,
                    [IncludeUsageDetails]
                  )
VALUES ( 'YourDatabase' ,       --  DBName ,
                    0 ,         --  Exclude ,
                    0 ,         --  ReindexGroupOrder ,
                    255 ,       --  ReindexOrder ,
                    10 ,        --  ReorgThreshold ,
                    20 ,        --  RebuildThreshold ,
                    80 ,        --  FILLFACTORopt ,
                    'ON' ,      --  PadIndex ,
                    'OFF' ,     --  SortInTempDB ,
                    1 ,         --  GetRowCT ,
                    1 ,         --  GetPostFragLevel ,
                    1 ,         --  UpdateStatsOnDefrag ,
                    0 ,         --  LogIndexPhysicalStats ,
                    'Limited' , --  IndexScanMode ,
                    1 ,         --  LogProgress ,
                    60 ,        --  LogRetDays ,
                    'Local' ,   --  LogLoc ,
                    '\\minioncon\c$' ,  --  MinionTriggerPath ,
                    1           --  IncludeUsageDetails
                  );
-- Insert IndexSettingsDB row for “Semi”, ReindexOrder=150 (after [YourDatabase])
INSERT  INTO [Minion].[IndexSettingsDB]
                  ( DBName ,
                    [Exclude] ,
                    [ReindexGroupOrder] ,
                    [ReindexOrder] ,
                    [ReorgThreshold] ,
                    [RebuildThreshold] ,
                    [FILLFACTORopt] ,
                    [PadIndex] ,
                    [SortInTempDB] ,
                    [GetRowCT] ,
                    [GetPostFragLevel] ,
                    [UpdateStatsOnDefrag] ,
                    [LogIndexPhysicalStats] ,
                    [IndexScanMode] ,
                    [LogProgress] ,
```

```
                    [LogRetDays] ,
                    [LogLoc] ,
                    [MinionTriggerPath] ,
                    [IncludeUsageDetails]
                  )
VALUES ( 'Semi' ,           --          DBName ,
                    0 ,             --          Exclude ,
                    0 ,             --          ReindexGroupOrder ,
                    150 ,           --          ReindexOrder ,
                    10 ,            --          ReorgThreshold ,
                    20 ,            --          RebuildThreshold ,
                    80 ,            --          FILLFACTORopt ,
                    'ON' ,          --          PadIndex ,
                    'OFF' ,         --          SortInTempDB ,
                    1 ,             --          GetRowCT ,
                    1 ,             --          GetPostFragLevel ,
                    1 ,             --          UpdateStatsOnDefrag ,
                    0 ,             --          LogIndexPhysicalStats ,
                    'Limited' ,     --          IndexScanMode ,
                    1 ,             --          LogProgress ,
                    60 ,            --          LogRetDays ,
                    'Local' ,       --          LogLoc ,
                    '\\minioncon\c$' ,      --          MinionTriggerPath ,
                    1               --          IncludeUsageDetails
                  );
-- Insert IndexSettingsDB row for “Lame”, ReindexOrder=100 (after “Semi”)
INSERT  INTO [Minion].[IndexSettingsDB]
                  ( DBName ,
                    [Exclude] ,
                    [ReindexGroupOrder] ,
                    [ReindexOrder] ,
                    [ReorgThreshold] ,
                    [RebuildThreshold] ,
                    [FILLFACTORopt] ,
                    [PadIndex] ,
                    [SortInTempDB] ,
                    [GetRowCT] ,
                    [GetPostFragLevel] ,
                    [UpdateStatsOnDefrag] ,
                    [LogIndexPhysicalStats] ,
                    [IndexScanMode] ,
                    [LogProgress] ,
                    [LogRetDays] ,
                    [LogLoc] ,
                    [MinionTriggerPath] ,
                    [IncludeUsageDetails]
                  )
VALUES ( 'Lame' ,           --          DBName ,
                    0 ,             --          Exclude ,
                    0 ,             --          ReindexGroupOrder ,
```

```
                    100 ,           --      ReindexOrder ,
                    10 ,            --      ReorgThreshold ,
                    20 ,            --      RebuildThreshold ,
                    80 ,            --      FILLFACTORopt ,
                    'ON' ,          --      PadIndex ,
                    'OFF' ,         --      SortInTempDB ,
                    1 ,             --      GetRowCT ,
                    1 ,             --      GetPostFragLevel ,
                    1 ,             --      UpdateStatsOnDefrag ,
                    0 ,             --      LogIndexPhysicalStats ,
                    'Limited' ,     --      IndexScanMode ,
                    1 ,             --      LogProgress ,
                    60 ,            --      LogRetDays ,
                    'Local' ,       --      LogLoc ,
                    '\\minioncon\c$' ,      --      MinionTriggerPath ,
                    1               --      IncludeUsageDetails
                  );
-- Insert IndexSettingsDB row for “Unused”, ReindexOrder=50 (after [Lame])
INSERT  INTO [Minion].[IndexSettingsDB]
                  ( DBName ,
                    [Exclude] ,
                    [ReindexGroupOrder] ,
                    [ReindexOrder] ,
                    [ReorgThreshold] ,
                    [RebuildThreshold] ,
                    [FILLFACTORopt] ,
                    [PadIndex] ,
                    [SortInTempDB] ,
                    [GetRowCT] ,
                    [GetPostFragLevel] ,
                    [UpdateStatsOnDefrag] ,
                    [LogIndexPhysicalStats] ,
                    [IndexScanMode] ,
                    [LogProgress] ,
                    [LogRetDays] ,
                    [LogLoc] ,
                    [MinionTriggerPath] ,
                    [IncludeUsageDetails]
                  )
VALUES ( 'Unused' ,     --      DBName ,
                    0 ,             --      Exclude ,
                    0 ,             --      ReindexGroupOrder ,
                    50 ,            --      ReindexOrder ,
                    10 ,            --      ReorgThreshold ,
                    20 ,            --      RebuildThreshold ,
                    80 ,            --      FILLFACTORopt ,
                    'ON' ,          --      PadIndex ,
                    'OFF' ,         --      SortInTempDB ,
                    1 ,             --      GetRowCT ,
                    1 ,             --      GetPostFragLevel ,
```

```
    1 ,             --      UpdateStatsOnDefrag ,
    0 ,             --      LogIndexPhysicalStats ,
    'Limited' ,     --      IndexScanMode ,
    1 ,             --      LogProgress ,
    60 ,            --      LogRetDays ,
    'Local' ,       --      LogLoc ,
    '\\minioncon\c$' ,      --      MinionTriggerPath ,
    1               --      IncludeUsageDetails
);
```

## How To: Reindex tables in a specific order

You can choose the order in which tables will be maintained. For example, let's say that you want two tables in [YourDatabase] to be indexed before all other tables in that database, in this order:

1. dbo.[Best] (it's the most important or most badly fragmented table)
2. dbo.[Okay]
3. *other tables*

In this case, we would insert a row into the Minion.IndexSettingsTable for each one of the databases, specifying either ReindexGroupOrder, ReindexOrder, or both, as needed.

> **NOTE**: For ReindexGroupOrder and ReindexOrder, higher numbers have a greater "weight" - they have a higher priority - and will be indexed 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 table, the settings in that row override **all** of the default index maintenance settings for that table. So, inserting a row for [YourDatabase].dbo.[Best] means that ONLY those specified index settings will be used for that table; no settings defined in Minion.IndexSettingsDB will apply to those specific tables.
>
> **NOTE:** Any non-specified tables will have a *ReindexGroupOrder* of 0, and a *ReindexOrder* of 0, by default. (Order settings at the database level have no effect on table-level ordering.)

Because we have so few tables in this example, the simplest method is to assign the heaviest "weight" to dbo.[Best], and lesser weights to dbo.[Okay]. In our example, we would insert two rows:

```
-- Insert IndexSettingsDB row for dbo.[Best], ReindexOrder=255 (first)
INSERT  INTO [Minion].[IndexSettingsTable]
                ( [DBName] ,
                  [SchemaName] ,
                  [TableName] ,
                  [Exclude] ,
                  [ReindexGroupOrder] ,
                  [ReindexOrder] ,
                  [ReorgThreshold] ,
                  [RebuildThreshold] ,
                  [FILLFACTORopt] ,
```

```
                    [PadIndex] ,
                    [GetRowCT] ,
                    [GetPostFragLevel] ,
                    [UpdateStatsOnDefrag] ,
                    [LogIndexPhysicalStats] ,
                    [IndexScanMode] ,
                    [LogProgress] ,
                    [LogRetDays] ,
                    [IncludeUsageDetails]

                  )
VALUES ( 'YourDatabase' , -- DBName
                    'dbo' ,         -- SchemaName
                    'Best' , -- TableName
                    0 ,             -- Exclude
                    0 ,             -- ReindexGroupOrder
                    255 ,           -- ReindexOrder
                    10 ,            -- ReorgThreshold
                    20 ,            -- RebuildThreshold
                    80 ,            -- FILLFACTORopt
                    'ON' ,          -- PadIndex
                    1 ,             -- GetRowCT
                    1 ,             -- GetPostFragLevel
                    1 ,             -- UpdateStatsOnDefrag
                    0 ,             -- LogIndexPhysicalStats
                    'Limited' ,     -- IndexScanMode
                    1 ,             -- LogProgress
                    60 ,            -- LogRetDays
                    1               -- IncludeUsageDetails
                  );
-- Insert IndexSettingsDB row for dbo.[Okay], ReindexOrder=200 (after [Best])
INSERT  INTO [Minion].[IndexSettingsTable]
                  ( [DBName] ,
                    [SchemaName] ,
                    [TableName] ,
                    [Exclude] ,
                    [ReindexGroupOrder] ,
                    [ReindexOrder] ,
                    [ReorgThreshold] ,
                    [RebuildThreshold] ,
                    [FILLFACTORopt] ,
                    [PadIndex] ,
                    [GetRowCT] ,
                    [GetPostFragLevel] ,
                    [UpdateStatsOnDefrag] ,
                    [LogIndexPhysicalStats] ,
                    [IndexScanMode] ,
                    [LogProgress] ,
                    [LogRetDays] ,
                    [IncludeUsageDetails]
```

```
                    )
VALUES ( 'YourDatabase' , -- DBName
                    'dbo' ,          -- SchemaName
                    'Okay' ,-- TableName
                    0 ,              -- Exclude
                    0 ,              -- ReindexGroupOrder
                    200 ,            -- ReindexOrder
                    10 ,             -- ReorgThreshold
                    20 ,             -- RebuildThreshold
                    80 ,             -- FILLFACTORopt
                    'ON' ,           -- PadIndex
                    1 ,              -- GetRowCT
                    1 ,              -- GetPostFragLevel
                    1 ,              -- UpdateStatsOnDefrag
                    0 ,              -- LogIndexPhysicalStats
                    'Limited' ,      -- IndexScanMode
                    1 ,              -- LogProgress
                    60 ,             -- LogRetDays
                    1                -- IncludeUsageDetails
                  );
```

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

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

GroupOrder GroupDBOrder

- 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 backup order would be as follows:

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

## How To: Generate reindex statements only

Sometimes it is useful to generate index maintenance statements and run them by hand, individually or in small groups. To generate reindex statements without running the statements, run the procedure Minion.IndexMaintMaster with the parameter @StmtOnly set to 1.

Example code - The following code will generate index statements for all tables in the [YourDatabase] database with the ONLINEopt set to "ONLINE" (that is, all tables that are configured to be maintained in online operations only).

```
EXEC [Minion].[IndexMaintMaster]
        @IndexOption = 'ONLINE',
        @ReorgMode = 'All',
        @RunPrepped = 0,
        @PrepOnly = 0,
        @StmtOnly = 1,
        @Include = 'YourDatabase',
        @Exclude = NULL,
        @LogProgress = 1;
```

## How To: Reindex only indexes that are marked ONLINE = ON (or, only ONLINE = OFF)

You can choose the set of indexes to maintain at a time. One of the filters available is to choose to maintain only the indexes that are set to ONLINE mode.

**Note:** The ONLINE=ON option is set in either the Minion.IndexSettingsDB table, or the Minion.IndexSettingsTable table. Alter existing rows, or insert new rows, to set which databases or tables should have the ONLINEopt set to ON. Any index that is not marked for ONLINE=ON is, by default, an OFFLINE index (whether it is marked for ONLINEopt=OFF or ONLINEopt=*NULL*).

**Note:** ONLINE index operations may not be possible for certain editions of SQL Server, and only for indexes that are eligible for ONLINE index operations. If you specify ONLINE when it is not possible, the routine will change it to OFFLINE. (For more information on editions that support Online Reindexing, see the MSDN article "Features Supported by the Editions of SQL Server 2014" at http://msdn.microsoft.com/en-us/library/cc645993.aspx.)

To reindex only indexes marked for ONLINE=ON, run the procedure Minion.IndexMaintMaster with the parameter @IndexOption set to 'ONLINE'. For example, to reindex only the ONLINE=ON indexes for ALL databases on the instance, use the following call:

```
EXEC [Minion].[IndexMaintMaster]
        @IndexOption = 'ONLINE',
        @ReorgMode = 'All',
        @RunPrepped = 0,
        @PrepOnly = 0,
        @StmtOnly = 0,
        @Include = NULL,
```

```
    @Exclude = NULL,
    @LogProgress = 1;
```

To reindex the only the ONLINE=ON indexes for a single database – [YourDatabase] – use the following call:

```
EXEC [Minion].[IndexMaintMaster]
    @IndexOption = 'ONLINE',
    @ReorgMode = 'All',
    @RunPrepped = 0,
    @PrepOnly = 0,
    @StmtOnly = 0,
    @Include = 'YourDatabase',
    @Exclude = NULL,
    @LogProgress = 1;
```

To reindex the only OFFLINE indexes (again, any index which does not have ONLINEopt=ON) for a single database – [YourDatabase] – use the following call:

```
EXEC [Minion].[IndexMaintMaster]
    @IndexOption = 'OFFLINE',
    @ReorgMode = 'All',
    @RunPrepped = 0,
    @PrepOnly = 0,
    @StmtOnly = 0,
    @Include = 'YourDatabase',
    @Exclude = NULL,
    @LogProgress = 1;
```

## How To: Gather index fragmentation statistics on a different schedule from the reindex routine

Maintenance windows are never the wide open space we'd like them to be. So, we made sure you have the option to maximize it: you can schedule the gathering of fragmentation stats at a different time than your reindexing itself. This way, you can use your entire maintenance window for processing indexes instead of finding out the fragmentation levels, which can take a very long time.

Let's take the example of ReallyBigDB:

1. Exclude ReallyBigDB from the job MinionReindexDBs-All-All (using @Exclude='ReallyBigDB').
2. Create the job MinionReindexDBs-ReallyBigDB-FragStats, to run sometime before the reindex job. For the job step, run Minion.IndexMaintMaster with @Include='ReallyBigDB', @PrepOnly=1, @RunPrepped=0, and other options as appropriate.
3. Create the job MinionReindexDBs-ReallyBigDB-All. For the job step, run Minion.IndexMaintMaster with @Include='ReallyBigDB', @PrepOnly=0, @RunPrepped=1

(which tells the SP to use the stats stored by the previous @PrepOnly=1 run), and other options as appropriate.

**Note:** There can only be one prep per database at a time. When you run @PrepOnly = 1, it enters the data into the table Minion.IndexTableFrag, and deletes any previous preparation runs for the database in question. So, while you can have as many databases as you like prepped in this table, each database can only have a single prep run. Even if the previous ones weren't deleted, the reindex SP only looks at the last one.

Example code - The following code will gather the fragmentation stats for ReallyBigDB:

```
EXEC [Minion].[IndexMaintMaster]
        @IndexOption = 'All',
        @ReorgMode = 'All',
        @RunPrepped = 0,
        @PrepOnly = 1,
        @StmtOnly = 0,
        @Include = 'ReallyBigDB',
        @Exclude = NULL,
        @LogProgress = 1;
```

The following execution will reindex the [ReallyBigDB] database, using the fragmentation stats stored by the previous @PrepOnly=1 run (instead of gathering statistics at the same time):

```
EXEC [Minion].[IndexMaintMaster]
        @IndexOption = 'All',
        @ReorgMode = 'All',
        @RunPrepped = 0,
        @PrepOnly = 0,
        @StmtOnly = 1,
        @Include = 'ReallyBigDB',
        @Exclude = NULL,
        @LogProgress = 1;
```

## How To: Exclude databases from index maintenance

You can exclude a database from all index maintenance in any of three ways:

- **Database level settings:** In the **Minion.IndexSettingsDB** table, insert or update the row for that database and set the Exclude column = 1.
- **Run time parameter:** In the appropriate reindex job(s), use the @Exclude parameter of the **Minion.IndexMaintMaster** procedure. This parameter accepts a column-delimited list of database names, and/or LIKE expressions. (E.g., @Exclude = 'DB1, DB3, Archive%'.)
- **Regex exclusion (advanced):** In the **Minion.DBMaintRegexLookup** table, insert a row with Action='Exclude' and the appropriate regular expression to encompass the proper set of database names.

**Database level settings:** To exclude [YourDatabase] from the Minion.IndexSettingsDB table, update the existing row, or insert a row:

```
INSERT INTO [Minion].[IndexSettingsDB]
        ( DBName
        , Exclude
        )
    VALUES
        ('YourDatabase'    -- DBName
        , 1                -- Exclude
        );
```

**Run time parameter:** To exclude [YourDatabase] from just one job running Minion.IndexMaintMaster, set the @Exclude parameter = 'YourDatabase'.

If you wanted to exclude all databases that begin with the string "Archive", set @Exclude to "Archive%".

**Regex exclusion:** This advanced option is controlled by regular expressions in a table, to exclude databases. This is most commonly used in rolling database scenarios, where you have archive or test databases with rolling names.

For example, to exclude all databases beginning with the word "Archive", and ending in a number (e.g. Archive2, Archive3, Archive201410), insert the following row:

```
INSERT  INTO [Minion].[DBMaintRegexLookup]
          ( [Action]
          , MaintType
          , Regex )
VALUES
          ( 'EXCLUDE'       -- Action. EXCLUDE or INCLUDE
          , 'ALL'           -- MaintType. ALL or REINDEX
          , '^Archive\d' );          -- Regex expression
```

**NOTE**: The use of the regular expressions exclude feature is not supported in SQL Server 2005.

## How To: Exclude a table from index maintenance

To exclude a single table from all index maintenance, insert a row to the **Minion.IndexSettingsTable** table and set the Exclude column = 1.

```
INSERT  INTO [Minion].[IndexSettingsTable]
                  ( DBName ,
                    SchemaName ,
                    TableName ,
                    Exclude
                  )
    VALUES
          ( 'YourDatabase' -- DBName
          , 'dbo'          -- SchemaName
```

```
  , 'BigTable'      -- TableName
  , 1               -- Exclude
);
```

## How To: Run code before or after index maintenance

You can schedule code to run before or after index maintenance operations. There are several options available:

- Run code before or after a single database
- Run code before or after each and every table in a database
- Run code before or after a single table
- Run code before or after each of a few tables (code executing before or after each table)
- Run code before or after all but a few tables (code executing before or after each table)
- Run code before or after reindex statements (within the same batch)

**NOTE:** Unless otherwise specified, pre and post code will run in the context of the Minion Reindex's database (wherever the Minion Reindex objects are stored), because 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.

**To run code before or after a single database**, insert a row for the database into Minion.IndexSettingsDB. Populate the column DBPreCode to run code before the index operations for that database; populate the column DBPostCode to run code before the index operations after that database. For example:

```
INSERT INTO [Minion].[IndexSettingsDB]
       ( DBName
       , Exclude
       , ReorgThreshold
       , RebuildThreshold
       , FILLFACTORopt
       , PadIndex
       , DBPreCode
       , DBPostCode)
   VALUES
       ('YourDatabase'   -- DBName
       , 0               -- Exclude
       , 15              -- ReorgThreshold
       , 25              -- RebuildThreshold
       , 90              -- FILLFACTORopt
       , 'ON'            -- DBPreCode
       , 'EXEC YourDatabase.dbo.SomeSP;' -- DBPreCode
       , 'EXEC YourDatabase.dbo.OtherSP;' -- DBPostCode
       );
```

**To run code before or after each and every table in a database,** insert a row for the database into Minion.IndexSettingsDB. Populate the column TablePreCode to run code before the index operations for

each individual table in the database; populate the column TablePostCode to run code after the index operations for each individual table in the database. For example:

```
INSERT INTO [Minion].[IndexSettingsDB]
       ( DBName
       , Exclude
       , ReorgThreshold
       , RebuildThreshold
       , FILLFACTORopt
       , PadIndex
       , TablePreCode
       , TablePostCode)
   VALUES
       ('YourDatabase'   -- DBName
       , 0               -- Exclude
       , 15              -- ReorgThreshold
       , 25              -- RebuildThreshold
       , 90              -- FILLFACTORopt
       , 'ON'            -- DBPreCode
       , 'EXEC YourDatabase.dbo.SomeSP;' -- TablePreCode
       , 'EXEC YourDatabase.dbo.OtherSP;' -- TablePostCode
       );
```

**To run code before or after a single table** (instead of each table), insert a row for the table into Minion.IndexSettingsTable. Populate the column TablePreCode to run code before the index operations for that database; populate the column TablePostCode to run code before the index operations after that database.

**Note:** An entry in Minion.IndexSettingsTable overrides ALL the index maintenance settings for that table; defaults set in Minion.IndexSettingsDB will be ignored for this table.

For example:

```
INSERT  INTO [Minion].[IndexSettingsTable]
                ( [DBName] ,
                  [SchemaName] ,
                  [TableName] ,
                  [ReorgThreshold] ,
                  [RebuildThreshold] ,
                  [FILLFACTORopt] ,
                  [PadIndex] ,
                  [GetRowCT] ,
                  [GetPostFragLevel] ,
                  [UpdateStatsOnDefrag] ,
                  [LogIndexPhysicalStats] ,
                  [IndexScanMode] ,
                  [LogProgress] ,
```

```
                    [LogRetDays] ,
                    [IncludeUsageDetails] ,
                    [TablePreCode] ,
                    [TablePostCode]
                  )
VALUES ( 'YourDatabase' ,       -- DBName
                    'dbo' ,     -- SchemaName
                    'YourTable' ,   -- TableName
                    10 ,        -- ReorgThreshold
                    20 ,        -- RebuildThreshold
                    80 ,        -- FILLFACTORopt
                    'ON' ,      -- PadIndex
                    1 ,         -- GetRowCT
                    1 ,         -- GetPostFragLevel
                    1 ,         -- UpdateStatsOnDefrag
                    0 ,         -- LogIndexPhysicalStats
                    'Limited' ,     -- IndexScanMode
                    1 ,         -- LogProgress
                    60 ,        -- LogRetDays
                    1 ,         -- IncludeUsageDetails
                    'EXEC YourDatabase.dbo.SomeSP;' ,     -- TablePreCode
                    'EXEC YourDatabase.dbo.OtherSP;'      -- TablePostCode
                  );
```

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

**To run code before or after all but a few tables**, insert one row for the database into Minion.IndexSettingsDB, populating the TablePreCode column and/or the TablePostCode column as appropriate. This will set up the execution code for all tables. Then, to prevent that code from running on a handful of tables, insert a row for each of those tables to Minion.IndexSettingsTable, 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 in [YourDatabase] *except* tables T1, T2, and T3, we would:

1. Insert a row to Minion.IndexSettingsDB for [YourDatabase], setting PreCode to 'EXEC dbo.SomeSP;'
2. Insert a row to Minion.IndexSettingsTable for [YourDatabase].dbo.T1, establishing all appropriate settings, and setting PreCode to *NULL*.
3. Insert a row to Minion.IndexSettingsTable for [YourDatabase].dbo.T2, establishing all appropriate settings, and setting PreCode to *NULL*.
4. Insert a row to Minion.IndexSettingsTable for [YourDatabase].dbo.T3, establishing all appropriate settings, and setting PreCode to *NULL*.

**NOTE:** We strongly recommend that you encapsulate any pre- or post-code into a stored procedure, unless the code is extremely simple. You can't pass pre- or post-code parameters into the indexing routine, so pre- and post-code must be self-contained.

**Example** - A real world TablePreCode example: You have a database supplied by a vendor. This database has a table with a non-clustered index with ALLOW_PAGE_LOCKS = OFF set. This option causes the reorganize operation on that index to fail. To resolve this, enter a row for that table into the Minion.IndexSettingsTable table, and include the following TablePreCode and TablePostCode options:

```
INSERT  Minion.IndexSettingsTable
                ( DBName
                  , SchemaName
                  , TableName
                  , Exclude
                  , ReindexGroupOrder
                  , ReindexOrder
                  , ReorgThreshold
                  , RebuildThreshold
                  , AllowPageLocks
                  , TablePreCode
                  , TablePostCode
                )
SELECT  'Demo'          --DBName
                , 'dbo'   --SchemaName
                , 'fragment'  --TableName
                , 0                 --Exclude
                , 0                 --ReindexGroupOrder
                , 0                 --ReindexOrder
                , 10                --ReorgThreshold,
                , 20                --RebuildThreshold
                , 'ON'
                , 'USE [Demo]; ALTER index ix_fragment2 ON dbo.fragment SET
(ALLOW_PAGE_LOCKS = ON);'   -- TablePreCode
                , 'USE [Demo]; ALTER index ix_fragment2 ON dbo.fragment SET
(ALLOW_PAGE_LOCKS = OFF);' --TablePostCode
```

**To run code before or after reindex statements** and within the same batch, you can use the StmtPrefix and/or StmtSuffix columns in Minion.IndexSettingsDB, Minion.IndexSettingsTable, or both.

It is important to understand that this column allows you to prefix *every* reindex statement for a table, or for a database, with a statement of your own. This is different from the table precode and postcode, because it is run in the same batch. Whereas, precode and postcode are run as completely separate statements, in different contexts.

A good example use case for this is the need to ensure that your reindex statement is chosen as the deadlock victim (should a deadlock occur) for DatabaseA. In this case, you would set StmtPrefix to "*SET*

*DEADLOCK_PRIORITY LOW;*" for DatabaseA in the Minion.IndexSettingsDB table. Other uses include setting a lock timeout, or adding a time delay to every reindex statement.

The StmtPrefix you choose will be shown as part of the Cmd column in the Minion.IndexMaintLogDetails table. IMPORTANT: To ensure that your statements run properly, **you must end the code in this column with a semicolon.**

## How To: Reindex databases on different schedules

Create a new job for each different schedule you require for index maintenance. Let us take a simple example:

- Perform index rebuilds on [YourDatabase] Friday night at 11pm
- Perform index rebuilds on all other databases on Saturday night at 10pm
- Perform index reorganization on *all* databases Sunday through Thursdays at 10pm.

To achieve this using the default installed Minion Reindex jobs:

1. Connect to "YourServer" and expand the SQL Agent node. You'll see two new jobs:
   - *MinionReindexDBs-All-All* – Runs once weekly – Fridays at 3:00 AM - to thoroughly defragment indexes (rebuild).
   - *MinionReindexDBs-All-REORG* – Runs Daily – 3:00 AM except for Friday – to complete lightweight defragmenting (reorganize).
2. Edit the "*MinionReindexDBs-All-All*" job:
   a. Edit the "Reindex" step: add 'YourDatabase' to the @Exclude parameter.
   b. Edit the schedule to run Friday night at 11pm.
3. Create a new job "*MinionReindexDBs-YourDatabase-All*".
   a. Create a "Reindex" step similar to that in the "MinionReindexDBs-All-All" job. Set @Include to 'YourDatabase', and set @Exclude to NULL.
   b. Schedule it to run Saturday night at 10pm.
4. Edit the schedules for the job "*MinionReindexDBs-All-REORG*" to run Sunday through Thursday at 10pm.

## How To: Configure how long the reindex logs are kept

Minion Reindex stores the "log retention in days" setting (LogRetDays) in the Minion.IndexSettingsDB table and the Minion.IndexSettingsTable table. You can therefore set the log retention for individual tables, individual databases and/or the system as a whole.

To change the default log retention for the system, run an update statement on the MinionDefault row in Minion.IndexSettingsDB. For example:

```
UPDATE [Minion].[IndexSettingsDB]
SET  [LogRetDays] = 60
WHERE [DBName] = 'MinionDefault';
```

To change the log retention for a specific database, run an update statement on that database's row in Minion.IndexSettingsDB. For example:

```
UPDATE [Minion].[IndexSettingsDB]
SET  [LogRetDays] = 90
WHERE [DBName] = 'YourDatabase';
```

# Moving Parts

## Overview of Tables

**Settings** (like rebuild threshold) are stored in two separate tables, one for database-level defaults, and another for table-level defaults. **Index fragmentation statistics** are stored long term in one table, and in the short term (to aid the index maintenance operations) in another table. **Index maintenance activities are logged** at a high ("master") level, and also at a per-operation level. So, for example, you can see how long the entire maintenance operation took, or how long an individual index rebuild lasted.

Reindex settings:

- **Minion.IndexSettingsDB** – This table holds index maintenance default settings at the database level. You may insert rows to define index maintenance settings per database, or you can rely on the system-wide default settings (defined in the "MinionDefault" row), or a combination of both.
- **Minion.IndexSettingsTable** - This table holds index maintenance default settings at the table level. You may insert rows to override the default index maintenance settings for individual tables. Any table that does not have a value in this table gets its settings from the appropriate entry in the Minion.IndexSettingsDB table.
- **Minion.DBMaintRegexLookup** – Allows you to exclude databases from index maintenance (or all maintenance), based off of regular expressions.

Index fragmentation stats:

- **Minion.IndexPhysicalStats** – This table holds index size and fragmentation information when the @currLogIndexPhysicalStats parameter is enabled. You can use this data after an index maintenance to investigate the raw fragmentation data, to estimate the next time a table will need to be reindexed, and more. Currently, this table must be manually deleted; the large amount of data here means we don't recommend leaving this setting on for long. Only turn it on when you need to diagnose something.
- **Minion.IndexTableFrag** - Holds index fragmentation information during the index maintenance process. If you run the index maintenance process with @PrepOnly = 1, this table stores that data; a subsequent

run of index maintenance with @RunPrepped = 1 will make use of this prepared data, instead of gathering statistics at the same time.

Logs:

- **Minion.IndexMaintLog** – Holds a database-level summary of the whole maintenance operation. Each row contains the database name, operation status, the start and end time of the index maintenance event, and much more. This is updated as each operation occurs, so that you have **Live Insight** into active index operations.
- **Minion.IndexMaintLogDetails** - Keeps a record of individual index maintenance activities. It contains one time-stamped row for each individual index operation (e.g., a single index rebuild). This is updated as each operation occurs, so that you have **Live Insight** into active index operations.

# Tables Detail

## Minion.IndexSettingsDB

This table holds index maintenance default settings at the **default** and **database levels**. You may insert rows for individual databases to override the default index maintenance settings (per database).

Minion.IndexSettingsDB is installed with default settings already in place, via the system-wide default row (identified by DBName = "MinionDefault"). If you do not need to fine tune the reindexing process at all, no action is required, and all maintenance will use this default configuration.

**Important**: Do not delete the MinionDefault row, or rename the DBName column for this row!

To override these default settings for a specific database, insert a new row for the individual database with the desired settings. Note that any database with its own entry in Minion.IndexSettingsDB retrieves ALL its configuration data from that row. For example, if you enter a row for [YourDatabase] and leave the FILLFACTORopt at NULL, Minion Reindex does not retrieve that value from the "MinionDefault" row; in this case, fill factor for YourDatabase would default to the current index setting (viewable for that index in sys.indexes).

| Name | Type | Description |
|---|---|---|
| ID | int | Primary key row identifier. |
| DBName | varchar | Database name. |
| Exclude | Bit | Exclude database from index maintenance.<br><br>For more on this topic, see "How To: Exclude Databases from Index Maintenance". |
| ReindexGroupOrder | Tinyint | Group to which this database belongs. Used solely for determining the order in which databases should be processed for index maintenance. |

| | | |
|---|---|---|
| | | By default, all databases have a value of 0, which means they'll be processed in the order they're queried from sysobjects.<br><br>Higher numbers have a greater "weight" (they have a higher priority), and will be indexed earlier than lower numbers. The range of ReindexGroupOrder weight numbers is 0-255.<br><br>For more information, see "How To: Reindex databases in a specific order". |
| ReindexOrder | Int | The index maintenance order within a group. Used solely for determining the order in which databases should be processed for index maintenance.<br><br>By default, all databases have a value of 0, which means they'll be processed in the order they're queried from sysobjects.<br><br>Higher numbers have a greater "weight" (they have a higher priority), and will be indexed earlier than lower numbers. We recommend leaving some space between assigned reindex order numbers (e.g., 10, 20, 30) so there is room to move or insert rows in the ordering.<br><br>For more information, see "How To: Reindex databases in a specific order". |
| ReorgThreshold | tinyint | The percentage threshold at which Index Maintenance should reorganize an index.<br><br>For example, if ReorgThreshold is set to 10 and the RebuildThreshold is 20, then a reorg will be done for all indexes between 10 and 19. And a rebuild will be done for all indexes 20 and above. |
| RebuildThreshold | tinyint | The percentage threshold at which Index Maintenance should rebuild an index.<br><br>For example, if ReorgThreshold is set to 10 and the RebuildThreshold is 20, then a reorg will be done for all indexes between 10 and 19. And a rebuild will be done for all indexes 20 and above. |
| FILLFACTORopt | Tinyint | Specify how full a reindex maintenance should make each page when it rebuilds an index. For example, a value of 85 would leave each data page 85% full of data. |

| | | |
|---|---|---|
| | | A value of *NULL* indicates that reindexing should use the current index setting (viewable for that index in sys.indexes). |
| PadIndex | varchar | Turn PAD_INDEX on or off. Valid inputs:<br>ON<br>OFF<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the current index setting (viewable for that index in sys.indexes). |
| ONLINEopt | Varchar | Perform ONLINE index maintenance for indexes in this database.<br>Valid inputs:<br>ON<br>OFF<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the system setting (in this case, "OFF", meaning the index maintenance will be done offline).<br><br>Note that ONLINE index operations may not be possible for certain editions of SQL Server, and only for indexes that are eligible for ONLINE index operations. If you specify ONLINE when it is not possible, the routine will change it to OFFLINE. |
| SortInTempDB | Varchar | Direct index maintenance to use TempDB to store the intermediate sort results that are used to build the index. Valid inputs:<br>ON<br>OFF<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the system setting (in this case, "OFF"). |
| MAXDOPopt | Tinyint | Specify the max degree of parallelism ("MAXDOP", the number of CPUs to use) for the index maintenance operations. If specified, this overrides the MAXDOP configuration option for the duration of the index operation. |
| DataCompression | Varchar | The data compression option. The options are as follows:<br><br>Valid inputs: |

| | | |
|---|---|---|
| | | NONE<br>ROW<br>PAGE<br>COLUMNSTORE<br>COLUMNSTORE_ARCHIVE<br><br>A NULL value here would indicate DataCompression='NONE'. |
| GetRowCT | Bit | Get a rowcount for each table. |
| GetPostFragLevel | Bit | Get the fragmentation level for each index, after the index maintenance operations are complete.<br><br>This is done on a per index basis as soon as the reindex operation is complete for each index. |
| UpdateStatsOnDefrag | Bit | Update statistics after defragmenting. This should always be on, but Minion provides the option just in case your stats are handled in some other way. |
| StatScanOption | varchar | Options available for the UPDATE STATISTICS statement (that is, anything that would go in the "WITH" statement). Valid inputs include any of the following options, as a comma-delimited list:<br>FULLSCAN<br>SAMPLE …<br>RESAMPLE<br>ON PARTITIONS ...<br>STATS_STREAM<br>ROWCOUNT<br>PAGECOUNT<br><br>For example, StatScanOption could be set to "SAMPLE 50 PERCENT", or "FULLSCAN, NORECOMPUTE". |
| IgnoreDupKey | varchar | Change the option so that for this index, inserts that add (normally illegal) duplicates generate a warning instead of an error. Applies to inserts that occur any time after the index operation. The default is OFF. Valid inputs:<br>ON<br>OFF |
| StatsNoRecompute | Varchar | Disable the automatic statistics update option, AUTO_UPDATE_STATISTICS. Valid inputs:<br>ON<br>OFF |
| AllowRowLocks | Varchar | Enable or disable the ALLOW_ROW_LOCKS option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx |

| | | |
|---|---|---|
| | | Valid inputs:<br>ON<br>OFF |
| AllowPageLocks | varchar | Enable or disable the ALLOW_PAGE_LOCKS option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx<br>Valid inputs:<br>ON<br>OFF |
| WaitAtLowPriority | Bit | Enable or disable the WAIT_AT_LOW_PRIORITY option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| MaxDurationInMins | int | Set the MAX_DURATION option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| AbortAfterWait | Varchar | Enable or disable the ABORT_AFTER_WAIT option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx<br>Valid inputs:<br>NONE<br>SELF<br>BLOCKERS |
| PushToMinion | Bit | Save these values to the central Minion server, if it exists. Modifies values for this particular database on the central Minion server.<br><br>A value of *NULL* indicates that this feature is off. Functionality not yet supported. |
| LogIndexPhysicalStats | Bit | Save the current index physical stats to a table (Minion.IndexPhysicalStats). |
| IndexScanMode | Varchar | Valid inputs:<br>Detailed<br>Limited<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the default (in this case, "LIMITED"). |
| DBPreCode | Nvarchar | Code to run for a database, before the index maintenance operations begin for that database.<br><br>For more on this topic, see "How To: Run code before or after index maintenance". |
| DBPostCode | nvarchar | Code to run for a database, after the index maintenance operations complete for that database. |

| | | |
|---|---|---|
| | | For more on this topic, see “How To: Run code before or after index maintenance”. |
| TablePreCode | Nvarchar | Code to run for each and every table, before the index maintenance operations begin for that table.<br><br>Note: To run precode just once, before maintenance for the database begins, use the DBPreCode column.<br><br>For more on this topic, see “How To: Run code before or after index maintenance”. |
| TablePostCode | Nvarchar | Code to run for each and every table, after the index maintenance operations end for that table.<br><br>For more on this topic, see “How To: Run code before or after index maintenance”. |
| LogProgress | Bit | Track the progress of index operations for this database.<br><br>The overall status is tracked in the Minion.IndexMaintLog table, while specific operations are tracked in the Status column Minion.IndexMaintLogDetails. |
| LogRetDays | Smallint | Number of days to retain index maintenance log data, for this database.<br><br>Just like any setting, if a table-specific row exists (in Minion.IndexSettingTable), those settings take precedence over database level settings. That is, if DB1.Table1 has an entry for LogRetDays=50, and DB1 has an entry for LogRetDays=40, the log will keep 50 days for DB1.Table1.<br><br>When first implemented, Minion Reindex defaults to 60 days of log retention. |
| LogLoc | Varchar | Determines whether log data is only stored on the local (client) server, or on both the local server and the central Minion (repository) server. Valid inputs:<br><br>Local<br>Repo |
| MinionTriggerPath | varchar | UNC path where the Minion logging trigger file is located.<br><br>Not applicable for a standalone Minion Reindex instance. |

| | | |
|---|---|---|
| RecoveryModel | Varchar | Change the recovery model of the database for the duration of the index maintenance operation. After index maintenance operations, the database will be set back to its original recovery model.<br><br>Valid inputs:<br>FULL<br>BULK_LOGGED<br>SIMPLE<br><br>**WARNING**: While we have done extensive testing and checking for this feature, it may still be possible for the process to fail in such a way that a database changed (for example) from FULL to SIMPLE may not switch back. Therefore, we advise that if you're in FULL you switch to BULK_LOGGED instead. It won't break your log chain and it has the same effect as switching to SIMPLE. |
| IncludeUsageDetails | Bit | Save index usage details from sys.dm_db_index_usage_stats, to Minion.IndexMaintLogDetails.<br><br>This feature is useful for tracking which indexes are being used the most over time. |
| StmtPrefix | nvarchar | This column allows you to prefix *every* reindex statement with a statement of your own. This is different from the table precode and postcode, because it is run in the same batch. Whereas, precode and postcode are run as completely separate statements, in different contexts.<br><br>Code entered in this column MUST end in a semicolon.<br><br>For more information, see "How To: Run code before or after index maintenance". |
| StmtSuffix | nvarchar | This column allows you to suffix *every* reindex statement with a statement of your own. This is different from the table precode and postcode, because it is run in the same batch. Whereas, precode and postcode are run as completely separate statements, in different contexts.<br><br>Code entered in this column MUST end in a semicolon. |

| | | |
|---|---|---|
| | | For more information, see “How To: Run code before or after index maintenance”. |

Discussion:

Insert a new row for [YourDatabase], if you wish to specify different default values for the reorg threshold, rebuild threshold, fill factor, and so on.

Discussion:

The Minion.IndexSettingsDB table comes with a row with “MinionDefault” as the DBName value. This row defines the system-wide defaults.

**Important:** Any row inserted for an individual database overrides only ALL of the values, whether or not they are specified. Refer to the following for an example:

| ID | DBName | Exclude | ReorgThreshold | RebuildThreshold | FillFactorOpt |
|---|---|---|---|---|---|
| 1 | MinionDefault | 0 | 10 | 20 | 90 |
| 2 | YourDatabase | 0 | 15 | 25 | NULL |

The first row, “MinionDefault”, is the set of default values to use for all the databases in the SQL Server instance. These values will be used for index maintenance for all databases that do not have an additional row in this table.

The second row, [YourDatabase], specifies some values for YourDatabase. This row completely overrides the “DefaultMinion” values for YourDatabase.

When index operations are performed for YourDatabase, *only* the values from the YourDatabase row will be used. So, even though the system-wide default (as specified in the MinionDefault row) for Fill Factor is 90%, YourDatabase will not use that default value. Because Fill Factor is NULL for YourDatabase, index maintenance will use the current value specified for the index. You can find the current value for a specific index by running the following query:

```
SELECT * FROM sys.indexes
WHERE name = 'nonMyIndex'
```

Likewise, you can also specify table-level override settings in the Minion.IndexSettingsTable table, which will override any settings for that particular table (and ignore the settings in Minion.IndexSettingsDB).

**NOTE**: While it is possible to exclude a single database from reindexing, by setting both the ReorgThreshold and RebuildThreshold above 100% for that database, we do not recommend this approach. This would cause Minion Reindex to gather fragmentation stats that will never be used. Instead, set the Exclude column to 1 for that database.

Likewise, we do not recommend setting the thresholds at 0%. While this would guarantee that every index in the database would be reorganized at every maintenance execution, it would likely be an unnecessary waste of resources.

Usage examples:

**Example 1**: Set custom thresholds, fill factor, and PadIndex for database 'YourDatabase'.

```
INSERT INTO [Minion].[IndexSettingsDB]
       ( DBName
       , Exclude
       , ReorgThreshold
       , RebuildThreshold
       , FILLFACTORopt
       , PadIndex )
   VALUES
       ('YourDatabase'   -- DBName
       , 0               -- Exclude
       , 15              -- ReorgThreshold
       , 25              -- RebuildThreshold
       , 90              -- FILLFACTORopt
       , 'ON'      -- PadIndex
       );
```

**Example 2**: Set custom reindex settings, and enable additional logging options for database 'YourDatabase'.

```
INSERT INTO [Minion].[IndexSettingsDB]
       ( DBName
       , Exclude
       , ReorgThreshold
       , RebuildThreshold
       , FILLFACTORopt
       , PadIndex
       , SortInTempDB
       , UpdateStatsOnDefrag
         -- Logging options:
       , GetRowCT
       , GetPostFragLevel
       , LogIndexPhysicalStats
       , LogProgress
       , LogRetDays
       , LogLoc)
```

```
VALUES
    ('YourDatabase'    -- DBName
    , 0                -- Exclude
    , 15               -- ReorgThreshold
    , 25               -- RebuildThreshold
    , 90               -- FILLFACTORopt
    , 'ON'             -- PadIndex
    , 'ON'             -- SortInTempDB
    , 1                -- UpdateStatsOnDefrag
    , 1                -- GetRowCT
    , 1                -- GetPostFragLevel
    , 1                -- LogIndexPhysicalStats
    , 1                -- LogProgress
    , 90               -- LogRetDays
    , 'Local'    -- LogLoc
    );
```

## Minion.IndexSettingsTable

This table holds index maintenance default settings at the **table level**. You may insert rows for individual tables to override the default index maintenance settings (per table).

Any table that does not have a value in this table will get all of its index maintenance settings from the Minion.IndexSettingsDB table. For example, if FillFactorOpt is set at 90 in Minion.IndexSettingsDB, but a row for Table1 here has FillFactorOpt at 95, then the 95 value is used. (If FillFactorOpt is left at NULL in the Minion.IndexSettingsTable row, the database level setting is still not used. Instead, the current index setting in sys.indexes will be used.)

Note that many shops will have no values in this table, if there is no need for ordering the tables for reindex, or for setting options for specific tables.

**Use**: Insert a new row for each individual table that requires specific table-level values for index maintenance.

| Name | Type | Description |
|---|---|---|
| ID | int | Primary key row identifier. |
| DBName | Varchar | Database name. |
| SchemaName | varchar | Schema name. |
| TableName | Varchar | Table name. |
| Exclude | bit | Exclude table from index maintenance.<br><br>For more on this topic, see “How To: Exclude Databases from Index Maintenance”. |
| GroupOrder | int | Group to which this table belongs. Used solely for determining the order in which tables should be processed for index maintenance. |

| | | |
|---|---|---|
| | | By default, all tables have a value of 0, which means they'll be processed in the order they're queried from sysobjects.<br><br>Higher numbers have a greater "weight" (they have a higher priority), and will be indexed earlier than lower numbers.<br><br>For more information, see "How To: Reindex databases in a specific order". |
| ReindexOrder | Int | The index maintenance order within a group. Used solely for determining the order in which tables should be processed for index maintenance.<br><br>By default, all tables have a value of 0, which means they'll be processed in the order they're queried from sysobjects.<br><br>Higher numbers have a greater "weight" (they have a higher priority), and will be indexed earlier than lower numbers. We recommend leaving some space between assigned reindex order numbers (e.g., 10, 20, 30) so there is room to move or insert rows in the ordering.<br><br>For more information, see "How To: Reindex databases in a specific order". |
| ReorgThreshold | Tinyint | The percentage threshold at which Index Maintenance should reorganize an index.<br><br>For example, if ReorgThreshold is set to 10 and the RebuildThreshold is 20, then a reorg will be done for all indexes between 10 and 19. And a rebuild will be done for all indexes 20 and above. |
| RebuildThreshold | Tinyint | The percentage threshold at which Index Maintenance should rebuild an index.<br><br>For example, if ReorgThreshold is set to 10 and the RebuildThreshold is 20, then a reorg will be done for all indexes between 10 and 19. And a rebuild will be done for all indexes 20 and above. |
| FILLFACTORopt | Tinyint | Specify how full a reindex maintenance should make each page when it rebuilds an index. For example, a value of 85 would leave each data page 85% full of data. |

| | | |
|---|---|---|
| | | A value of *NULL* indicates that reindexing should use the current index setting (viewable for that index in sys.indexes). |
| PadIndex | varchar | Turn PAD_INDEX on or off. Valid inputs:<br>ON<br>OFF<br><br>A value of *NULL* indicates that reindexing should use the current index setting (viewable for that index in sys.indexes). |
| ONLINEopt | Varchar | Perform ONLINE index maintenance for indexes in this database.<br>Valid inputs:<br>ON<br>OFF<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the system setting (in this case, "OFF", meaning the index maintenance will be done offline).<br><br>Note that ONLINE index operations may not be possible for certain editions of SQL Server, and only for indexes that are eligible for ONLINE index operations. If you specify ONLINE when it is not possible, the routine will change it to OFFLINE. |
| SortInTempDB | Varchar | Direct index maintenance to use TempDB to store the intermediate sort results that are used to build the index. Valid inputs:<br>ON<br>OFF<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the system setting (in this case, "OFF"). |
| MAXDOPopt | tinyint | Specify the max degree of parallelism ("MAXDOP", the number of CPUs to use) for the index maintenance operations. If specified, this overrides the MAXDOP configuration option for the duration of the index operation. |
| DataCompression | Varchar | The data compression option. The options are as follows:<br><br>Valid inputs:<br>NONE<br>ROW<br>PAGE |

| | | |
|---|---|---|
| | | COLUMNSTORE<br>COLUMNSTORE_ARCHIVE |
| GetRowCT | Bit | Get a rowcount for this table. |
| GetPostFragLevel | bit | Get the level of fragmentation for each index, after the index maintenance operations are complete. |
| UpdateStatsOnDefrag | Bit | Update statistics after defragmenting. This should always be on, but Minion provides the option just in case your stats are handled in some other way. |
| StatScanOption | varchar | Options available for the UPDATE STATISTICS statement (that is, anything that would go in the "WITH" statement). Valid inputs include any of the following options, as a comma-delimited list:<br>FULLSCAN<br>SAMPLE …<br>RESAMPLE<br>ON PARTITIONS ...<br>STATS_STREAM<br>ROWCOUNT<br>PAGECOUNT<br><br>For example, StatScanOption could be set to "SAMPLE 50 PERCENT", or "FULLSCAN, NORECOMPUTE". |
| IgnoreDupKey | varchar | Change the option so that for this index, inserts that add (normally illegal) duplicates generate a warning instead of an error. Applies to inserts that occur any time after the index operation. The default is OFF. Valid inputs:<br>ON<br>OFF |
| StatsNoRecompute | Varchar | Disable the automatic statistics update option, AUTO_UPDATE_STATISTICS. Valid inputs:<br>ON<br>OFF |
| AllowRowLocks | varchar | Enable or disable the ALLOW_ROW_LOCKS option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx<br>Valid inputs:<br>ON<br>OFF |
| AllowPageLocks | varchar | Enable or disable the ALLOW_PAGE_LOCKS option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx<br>Valid inputs:<br>ON |

| | | |
|---|---|---|
| | | OFF |
| WaitAtLowPriority | bit | Enable or disable the WAIT_AT_LOW_PRIORITY option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| MaxDurationInMins | int | Set the MAX_DURATION option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| AbortAfterWait | Varchar | Enable or disable the ABORT_AFTER_WAIT option of ALTER INDEX. See http://msdn.microsoft.com/en-us/library/ms188388.aspx<br>Valid inputs:<br>NONE<br>SELF<br>BLOCKERS |
| PushToMinion | Bit | Save these values to the central Minion server, if it exists. Modifies values for this particular table on the central Minion server.<br><br>A value of *NULL* indicates that this feature is off. Functionality not yet supported. |
| LogIndexPhysicalStats | bit | Save the current index physical stats to a table. |
| IndexScanMode | Varchar | Valid inputs:<br>Detailed<br>Limited<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the default (in this case, "LIMITED"). |
| TablePreCode | varchar | Code to run for this table, before the index maintenance operations begin for that table.<br><br>Note: To run precode once before each and every individual table in a database, use the TablePreCode column in Minion.IndexSettingsDB.<br><br>For more on this topic, see "How To: Run code before or after index maintenance". |
| TablePostCode | varchar | Code to run for this table, after the index maintenance operations complete for that table.<br><br>Note: To run postcode once after each and every individual table in a database, use the TablePreCode column in Minion.IndexSettingsDB. |

| | | |
|---|---|---|
| | | For more on this topic, see “How To: Run code before or after index maintenance”. |
| LogProgress | bit | Track the progress of index operations for this table.<br><br>The overall index maintenance status is tracked in the Minion.IndexMaintLog table, while specific operations are tracked in the Status column Minion.IndexMaintLogDetails. |
| LogRetDays | smallint | Number of days to retain index maintenance log data, for this table.<br><br>Just like any setting, if a table-specific row exists (in Minion.IndexSettingTable), those settings take precedence over database level settings. That is, if DB1.Table1 has an entry for LogRetDays=50, and DB1 has an entry for LogRetDays=40, the log will keep 50 days for DB1.Table1.<br><br>When first implemented, Minion Reindex defaults to 60 days of log retention. |
| PartitionReindex | bit | Future use. |
| isLOB | bit | Internal use. |
| TableType | char | Internal use. |
| IncludeUsageDetails | bit | Save index usage details from sys.dm_db_index_usage_stats, to Minion.IndexMaintLogDetails. |
| StmtPrefix | nvarchar | This column allows you to prefix *every* reindex statement with a statement of your own. This is different from the table precode and postcode, because it is run in the same batch. Whereas, precode and postcode are run as completely separate statements, in different contexts.<br><br>Code entered in this column MUST end in a semicolon.<br><br>For more information, see “How To: Run code before or after index maintenance” |
| StmtSuffix | nvarchar | This column allows you to suffix *every* reindex statement with a statement of your own. This is different from the table 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.<br><br>For more information, see “How To: Run code before or after index maintenance” |

Discussion:

Insert a new row for a single table in [YourDatabase], if you wish to specify different default values for the reorg threshold, rebuild threshold, fill factor, and so on.

**Important:** Any row inserted for an individual table overrides only ALL of the values for that table, whether or not they are specified. Refer to the following for an example:

| ID | DBName | SchemaName | TableName | Exclude | ReorgThreshold |
|---|---|---|---|---|---|
| 1 | YourDatabase | dbo | Table1 | 0 | 15 |

The first row specifies values for Table1 in YourDatabase. This row completely overrides all other values for that table.

When index operations are performed for Table1, *only* the values from the Table1 row will be used. Even though the ReorgThreshold value may be specified in Minion.IndexSettingsDB for [YourDatabase] – and there is most definitely a default value specified there - Table1 will not use that database-level value.

Usage examples:

**Example 1**: Set custom thresholds, fill factor, and PadIndex for Table1.

```
INSERT INTO [Minion].[IndexSettingsTable]
        ( DBName
        , SchemaName
        , TableName
        , Exclude
        , ReorgThreshold
        , RebuildThreshold
        , FILLFACTORopt
        , PadIndex )
    VALUES
        ('YourDatabase'   -- DBName
        , 'dbo'           -- SchemaName
        , 'Table1'  -- TableName
        , 0               -- Exclude
```

```
, 15                -- ReorgThreshold
, 25                -- RebuildThreshold
, 90                -- FILLFACTORopt
, 'ON'      -- PadIndex
);
```

**NOTE**: While it is possible to exclude a single table from reindexing, by setting both the ReorgThreshold and RebuildThreshold above 100% for that database, we do not recommend this approach. Instead, set the Exclude column to 1 for that table.

**NOTE**: To ensure a table is reindexed at every run, set the ReorgThreshold at 0%.

## Minion.DBMaintRegexLookup

Allows you to exclude databases from index maintenance (or all maintenance), based off of regular expressions.

| Name | Type | Description |
|---|---|---|
| Action | varchar | Action to perform with this regular expression.<br><br>Valid inputs:<br>EXCLUDE |
| MaintType | varchar | Maintenance type to which this applies.<br><br>Valid inputs:<br>ALL<br>REINDEX |
| Regex | nvarchar | Regular expression to match a database name, or set of database names. |

Discussion

This table is meant to be inclusive for all maintenance operations. (Minion will, in future, be more than just an excellent reindex solution.) Therefore, the MaintType column is important. By specifying 'All' you ensure that all databases that satisfy the regex expression are excluded from all maintenance operations (Reindex, Backup, CheckDB, Update Statistics, etc.). This is an excellent way to shotgun groups of databases and exclude them from all maintenance. However, if you want to only exclude the databases from reindexing, set MaintType to 'Reindex'.

**Example 1**

To exclude any database named "Minion" followed by one or more characters, from ALL database maintenance routines, insert the following row:

```
INSERT  INTO Minion.DBMaintRegexLookup
                ( [Action], MaintType, RegEx )
VALUES ( 'Exclude', 'All', 'Minion\w+' );
```

**Example 2**

To exclude any database named "ADB" followed by one or more decimal digits, from index maintenance, insert the following row:

```
INSERT  INTO Minion.DBMaintRegexLookup
                    ( [Action], MaintType, RegEx )
VALUES ( 'Exclude', 'Reindex', 'ADB\d+' );
```

These databases will still be processed in the backups, CheckDB, and other maintenance operations, if those Minion modules are running on your instance.

## Minion.IndexPhysicalStats

Stores the raw data from sys.dm_db_index_physical_stats. You can optionally save index size and fragmentation information to Minion.IndexPhysicalStats for use in investigating issues, as needed.

To turn on IndexPhysicalStats logging, set the **LogIndexPhysicalStats** field to 1 for a database or table (in Minion.IndexSettingsDB or Minion.IndexSettingsTable, respectively). Data will be saved to Minion.IndexPhysicalStats for each index maintenance run thereafter.

**WARNING**: **LogIndexPhysicalStats** is turned off by default because it can generate large amounts of data, and the table is currently not part of the log retention cleanup process. We recommend you use this feature only as needed.

**NOTE:** Even if LogIndexPhysicalStats is enabled, this table will not store data for any table or database that is excluded from index maintenance, because the index process does not gather fragmentation stats for excluded tables\databases.

| Name | Type | Description |
|---|---|---|
| ExecutionDateTime | Datetime | The execution date and time, common to the entire run of a database index maintenance event. |
| BatchDateTime | datetime | Date and time the index physical stats data was gathered. |
| IndexScanMode | varchar | Scan level that is used to obtain statistics. This is equivalent to the 'mode' input for sys.dm_index_physical_stats. |
| DBName | Varchar | Database name. |
| SchemaName | varchar | Schema name. |
| TableName | varchar | Table name. |
| IndexName | varchar | Index name. |

| database_id | smallint | Database ID. See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
|---|---|---|
| object_id | int | Object ID. See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| index_id | int | Index ID. See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| partition_number | int | Partition ID. See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| index_type_desc | nvarchar | Description of index type, e.g. HEAP, CLUSTERED, NONCLUSTERED, etc. See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| alloc_unit_type_desc | nvarchar | Allocation type unit, e.g. IN_ROW_DATA, LOB_DATA, ROW_OVERFLOW_DATA. * See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| index_depth | Tinyint | Number of index levels. Note that 1 means the table is a HEAP. See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| index_level | tinyint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| avg_fragmentation_in_percent | float | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| fragment_count | bigint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| avg_fragment_size_in_pages | float | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| page_count | bigint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| avg_page_space_used_in_percent | float | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| record_count | bigint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| ghost_record_count | bigint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| version_ghost_record_count | bigint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| min_record_size_in_bytes | int | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| max_record_size_in_bytes | int | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| avg_record_size_in_bytes | float | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| forwarded_record_count | bigint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |

| | | |
|---|---|---|
| compressed_page_count | bigint | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |

## Minion.IndexTableFrag

Holds index fragmentation information on a short-term basis, to be used by the currently-running index maintenance process. Minion.IndexTableFrag also holds fragmentation data for prepped operations (created with Minion.IndexMaintMaster with @PrepOnly = 1). PrepOnly data is marked with Prepped = 1 in this table, so Minion Reindex knows the difference between a current process and a prepped process.

For more information on these columns, see Minion.IndexMaintDB, Minion.IndexMaintTable, and/or the MSDN article on sys.dm_db_index_physical_stats at http://msdn.microsoft.com/en-us/library/ms188917.aspx

| Name | Type | Description |
|---|---|---|
| ExecutionDateTime | Datetime | The execution date and time, common to the entire run of a database index maintenance event. |
| DBName | varchar | Database name. |
| DBID | int | Database ID. |
| TableID | bigint | Table ID. |
| SchemaName | varchar | Schema name. |
| TableName | varchar | Table name. |
| IndexName | varchar | Index name. |
| IndexID | bigint | Index ID. |
| IndexType | tinyint | Index type number, e.g. 0 = HEAP, etc.<br><br>See http://msdn.microsoft.com/en-us/library/ms173760.aspx |
| IndexTypeDesc | nvarchar | Description of index type, e.g. HEAP, CLUSTERED, NONCLUSTERED, etc.<br><br>See http://msdn.microsoft.com/en-us/library/ms173760.aspx |
| IsDisabled | bit | See http://msdn.microsoft.com/en-us/library/ms173760.aspx |
| IsHypothetical | bit | See http://msdn.microsoft.com/en-us/library/ms173760.aspx |
| avg_fragmentation_in_percent | float | See http://msdn.microsoft.com/en-us/library/ms188917.aspx |
| ReorgThreshold | tinyint | The percentage threshold at which Index Maintenance should reorganize an index.<br><br>For example, if ReorgThreshold is set to 10 and the RebuildThreshold is 20, then a reorg will be done for all indexes between 10 and 19. And a |

| | | |
|---|---|---|
| | | rebuild will be done for all indexes 20 and above. |
| RebuildThreshold | Tinyint | The percentage threshold at which Index Maintenance should rebuild an index.<br><br>For example, if ReorgThreshold is set to 10 and the RebuildThreshold is 20, then a reorg will be done for all indexes between 10 and 19. And a rebuild will be done for all indexes 20 and above. |
| FillFactorOpt | tinyint | Specify how full a reindex maintenance should make each page when it rebuilds an index. For example, a value of 85 would leave each data page 85% full of data.<br><br>A value of *NULL* indicates that reindexing should use the current index setting (viewable for that index in sys.indexes). |
| PadIndex | varchar | Turn PAD_INDEX on or off. Valid inputs:<br>ON<br>OFF<br><br>A value of *NULL* indicates that reindexing should use the current index setting (viewable for that index in sys.indexes). |
| OnlineOpt | varchar | Perform ONLINE index maintenance for indexes in this database.<br>Valid inputs:<br>ON<br>OFF<br>*NULL*<br><br>A value of *NULL* indicates that reindexing should use the system setting (in this case, "OFF", meaning the index maintenance will be done offline).<br><br>Note that ONLINE index operations may not be possible for certain editions of SQL Server, and only for indexes that are eligible for ONLINE index operations. If you specify ONLINE when it is not possible, the routine will change it to OFFLINE. |
| SortInTempDB | varchar | Direct index maintenance to use TempDB to store the intermediate sort results that are used to build the index. Valid inputs:<br>ON<br>OFF<br>*NULL* |

| | | |
|---|---|---|
| | | A value of *NULL* indicates that reindexing should use the system setting (in this case, "OFF"). |
| MAXDOPopt | tinyint | Specify the max degree of parallelism ("MAXDOP", the number of CPUs to use) for the index maintenance operations. If specified, this overrides the MAXDOP configuration option for the duration of the index operation. |
| DataCompression | varchar | The data compression option. The options are as follows:<br><br>Valid inputs:<br>NONE<br>ROW<br>PAGE<br>COLUMNSTORE<br>COLUMNSTORE_ARCHIVE<br><br>A NULL value here would indicate DataCompression='NONE'. |
| GetRowCT | bit | Get a rowcount for each table. |
| GetPostFragLevel | bit | Get the fragmentation level for each index, after the index maintenance operations are complete.<br><br>This is done on a per index basis as soon as the reindex operation is complete for each index. |
| UpdateStatsOnDefrag | bit | Update statistics after defragmenting. This should always be on, but Minion provides the option just in case your stats are handled in some other way. |
| StatScanOption | varchar | Options available for the UPDATE STATISTICS statement (that is, anything that would go in the "WITH" statement). Valid inputs include any of the following options, as a comma-delimited list:<br>FULLSCAN<br>SAMPLE …<br>RESAMPLE<br>ON PARTITIONS ...<br>STATS_STREAM<br>ROWCOUNT<br>PAGECOUNT<br><br>For example, StatScanOption could be set to "SAMPLE 50 PERCENT", or "FULLSCAN, NORECOMPUTE". |
| IgnoreDupKey | varchar | Change the option so that for this index, inserts that add (normally illegal) duplicates generate a |

| | | |
|---|---|---|
| | | warning instead of an error. Applies to inserts that occur any time after the index operation. The default is OFF. Valid inputs:<br>ON<br>OFF |
| StatsNoRecompute | Varchar | Disable the automatic statistics update option, AUTO_UPDATE_STATISTICS. Valid inputs:<br>ON<br>OFF |
| AllowRowLocks | varchar | See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| AllowPageLocks | varchar | See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| WaitAtLowPriority | bit | See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| MaxDurationInMins | int | See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| AbortAfterWait | varchar | See http://msdn.microsoft.com/en-us/library/ms188388.aspx |
| LogProgress | Bit | Track the progress of index operations for this database.<br><br>The overall status is tracked in the Minion.IndexMaintLog table, while specific operations are tracked in the Status column Minion.IndexMaintLogDetails. |
| LogRetDays | smallint | Number of days to retain index maintenance log data, for this table.<br><br>Just like any setting, if a table-specific row exists (in Minion.IndexSettingTable), those settings take precedence over database level settings. That is, if DB1.Table1 has an entry for LogRetDays=50, and DB1 has an entry for LogRetDays=40, the log will keep 50 days for DB1.Table1.<br><br>When first implemented, Minion Reindex defaults to 60 days of log retention. |
| PushToMinion | Bit | Save these values to the central Minion server, if it exists. Modifies values for this particular table on the central Minion server.<br><br>A value of *NULL* indicates that this feature is off. Functionality not yet supported. |
| LogIndexPhysicalStats | bit | Save the current index physical stats to a table (Minion.IndexPhysicalStats). |
| IndexScanMode | varchar | Valid inputs:<br>Detailed<br>Limited |

| | | |
|---|---|---|
| | | *NULL*<br><br>A value of *NULL* indicates that reindexing should use the default (in this case, "LIMITED"). |
| TablePreCode | nvarchar | Code to run for a table, before the index maintenance operations begin for that table.<br><br>For more on this topic, see "How To: Run code before or after index maintenance". |
| TablePostCode | nvarchar | Code to run for a table, after the index maintenance operations complete for that table.<br><br>For more on this topic, see "How To: Run code before or after index maintenance". |
| Prepped | bit | If Prepped=1, this data was entered into the table as a result of running the Minion.IndexMaintMaster stored procedure with @PrepOnly = 1.<br><br>It is then necessary to run the reindexing routine with @RunPrepped = 1 to use this data.<br><br>For more on this topic, see "How To: Gather index fragmentation statistics on a different schedule from the reindex routine".<br><br>**NOTE**: There can only be one set of prepared data per database at any given time. When you run @PrepOnly = 1, it enters the data into this table, and deletes any previous prep runs for the database in question. So while you can have as many databases as you like prepped in this table, each database can only have a single prep run. |
| GroupOrder | Int | Group to which this database belongs. Used solely for determining the order in which databases should be processed for index maintenance.<br><br>By default, all tables have a value of 0, which means they'll be processes in the order they're queried from sysobjects.<br><br>Higher numbers have a greater "weight" (they have a higher priority), and will be indexed earlier than lower numbers. The range of ReindexGroupOrder weight numbers is 0-255. |

| | | |
|---|---|---|
| | | For more information, see “How To: Reindex databases in a specific order”. |
| ReindexOrder | Int | The index maintenance order within a group. Used solely for determining the order in which databases should be processed for index maintenance.<br><br>By default, all tables have a value of 0, which means they’ll be processes in the order they’re queried from sysobjects.<br><br>Higher numbers have a greater “weight” (they have a higher priority), and will be indexed earlier than lower numbers. We recommend leaving some space between assigned reindex order numbers (e.g., 10, 20, 30) so there is room to move or insert rows in the ordering.<br><br>For more information, see “How To: Reindex databases in a specific order”. |
| StmtPrefix | nvarchar | The code that will prefix *every* reindex statement with a statement of your own.<br><br>For more information, see “How To: Run code before or after index maintenance” |
| StmtSuffix | nvarchar | The code that will suffix *every* reindex statement with a statement of your own.<br><br>For more information, see “How To: Run code before or after index maintenance” |

## Minion.IndexMaintLog

Holds a database level summary of the maintenance operation. This table stores the parameters and settings that were used during the operation, as well as status and summary information. This information can help with troubleshooting, or just stats gathering when you want to see what has happened between one maintenance run to the next. For example, you can use this to determine why a job has wildly varying run times.

| **Name** | **Type** | **Description** |
|---|---|---|
| ID | Bigint | Primary key row identifier. |
| ExecutionDateTime | datetime | Date and time of the entire run. If several databases are run in the same job then this value will be the same for all of them.<br><br>Join ExecutionDatetime and DBName with the same columns in the |

| | | |
|---|---|---|
| | | Minion.IndexMaintDetails table to see full details. |
| Status | Varchar | Status of the current reindex operation. If the database completes without error this column will be set to 'Complete'. If the database encountered errors you will see 'Complete with errors'.<br><br>This column will also be updated with high level status messages when using the Live Insight feature. To see details of these high level messages check the Status column in the IndexMaintLogDetails table. If the current database is complete and this column doesn't have 'Complete' or 'Complete with errors', then that probably means that the job was stopped either by an unhandled fatal error or manually. Once the job is stopped there is no way to update this column further so it will be stuck in an invalid status. |
| DBName | varchar | Database name. |
| Tables | Varchar | Shows whether Offline, Online, or All indexes were processed. Offline indexes are those that have to be done offline because they contain a legacy data type like text, image, etc. Online tables are the ones that can be processed online. If you choose Online for a table and it has an index that must be done offline, then that index will be excluded from processing. |
| RunPrepped | bit | This shows that the job was called with this option set to 1. RunPrepped means that a PrepOnly run was executed before in order to store the fragmentation stats for the indexes.<br><br>For more information, see "How To: Gather index fragmentation statistics on a different schedule from the reindex routine". |
| PrepOnly | bit | This option is used to prepare a reindexing job for later processing.<br><br>For more information, see "How To: Gather index fragmentation statistics on a different schedule from the reindex routine". |
| ReorgMode | varchar | Shows that the job was called with either REORG, REBUILD, or All. If set to REORG, tables will only be reorged. This includes tables that are past the RebuildThreshold. However, if REBUILD is used, only tables that are past the RebuildThreshold will be |

| | | |
|---|---|---|
| | | processed. Tables between the ReorgThreshold and RebuildThreshold will be ignored. |
| NumTablesProcessed | int | The number of tables processed for the current database. |
| NumIndexesProcessed | int | The number of indexes processed for the current database. |
| NumIndexesRebuilt | int | The number of indexes rebuilt for the current database. |
| NumIndexesReorged | int | The number of indexes reorged for the current database. |
| RecoveryModelChanged | bit | 0 or 1. Was the recovery model for the current database changed? |
| RecoveryModelCurrent | varchar | This is the recovery model of the database before the reindex operation began. |
| RecoveryModelReindex | varchar | This is the recovery model of the database during the operation. The recovery model can be changed in the IndexSettingsDB table. |
| SQLVersion | varchar | The current version of SQL Server. |
| SQLEdition | varchar | The current edition of SQL Server. |
| DBPreCode | nvarchar | Any database-level code that was run before Minion Reindex processed any tables. |
| DBPostCode | nvarchar | Any database-level code that was run after Minion Reindex processed all the tables. |
| DBPreCodeBeginDateTime | datetime | Date and time the precode started. |
| DBPreCodeEndDateTime | datetime | Date and time the precode ended. |
| DBPostCodeBeginDateTime | datetime | Date and time the postcode started. |
| DBPostCodeEndDateTime | datetime | Date and time the postcode ended. |
| DBPreCodeRunTimeInSecs | int | How many seconds the precode took. |
| DBPostCodeRunTimeInSecs | int | How many seconds the postcode took. |
| ExecutionFinishTime | datetime | Date and time the entire database reindex operation finished. |
| ExecutionRunTimeInSecs | int | How many seconds the database reindex operation took. |

Discussion:

Each row contains the database name, the start and end time of the index maintenance event, and much more.

## Minion.IndexMaintLogDetails

Keeps a record of individual index maintenance activities. It contains one time-stamped row for each individual index operation (e.g., a single index rebuild).

| Name | Type | Description |
|---|---|---|
| ID | int | Primary key row identifier. |
| ExecutionDateTime | datetime | Date and time the entire reindex operation took place. If the job were started through IndexMaintMaster then all databases in that |

| | | |
|---|---|---|
| | | run have the same ExecutionDateTime. If the job was run manually from Minion.IndexMaintDB, then this value will only be for this database. It will still have a matching row in the Minion.IndexMaintLog table. |
| Status | varchar | Current status of the index operation. If Live Insight is being used the status updates will appear here. When finished, this column will either read 'Complete' or 'FATAL ERROR: *error message*'. The one exception is when the job has been run with PrepOnly = 1.<br><br>When running with PrepOnly = 1, this column is updated with the index fragmentation gather stats. For example, say that you were pulling fragmentation stats for 7 indexes with PrepOnly = 1. The final status message would look something like this: '*7 of 7: GATHERING FRAG STATS: dbo.fragment.ix_fragment2*'. This shows you that all 7 of the fragmentation stats were collected. |
| DBName | Varchar | Database name. |
| TableID | bigint | The table ID in sysobjects. |
| SchemaName | Varchar | Schema name. |
| TableName | Varchar | Table name. |
| IndexID | Int | The index ID from sys.indexes. |
| IndexName | varchar | The index name from sys.indexes. |
| IndexTypeDesc | varchar | The index type description from sys.indexes. |
| IndexScanMode | varchar | Either NULL, Limited, or Detailed. NULL means that nothing was entered into the column in either Minion.IndexSettingsDB or Minion.IndexSettingsTable and therefore the default (*Limited*) was used. |
| Op | varchar | Operation. Valid inputs are Reorg or Rebuild. This is the type of operation performed in the current index. |
| OnlineOpt | varchar | NULL, On, Off. If NULL, then nothing was entered into either the Minion.IndexSettingsDB or Minion.IndexSettingsTable tables, and the default (OFF) is used. So the operation was either done offline or online. |
| ReorgThreshold | Tinyint | The percentage threshold at which Index Maintenance should reorganize an index. |
| ReindexThreshold | Tinyint | The percentage threshold at which Index Maintenance should rebuild an index. |
| FragLevel | tinyint | The fragmentation level of the current index at the time the fragmentation stats were taken. |

| | | |
|---|---|---|
| | | If they were taken earlier in the day as part of a PrepOnly run, then they may not match current fragmentation stats. |
| Stmt | nvarchar | The reindex statement that was run. |
| GroupOrder | int | Group to which this table belongs. Used solely for determining the order in which tables should be processed for index maintenance.<br><br>Most of the time this will be 0. However, if you choose to take advantage of this feature a row in Minion.IndexSettingsTable will get you there. This is a weighted list so higher numbers are more important and will be processed first.<br><br>For more information, see "How To: Reindex databases in a specific order". |
| ReindexOrder | int | The ordering of the tables within the previous group. Most of the time this will be 0. However, if you choose to take advantage of this feature a row in Minion.IndexSettingsTable will get you there. This is a weighted list so higher numbers are more important and will be processed first.<br><br>For more information, see "How To: Reindex databases in a specific order". |
| PreCode | varchar | Any precode run before the table is processed. If the table has multiple indexes the precode will only be run once. |
| PostCode | varchar | Any postcode run after the table is processed. If the table has multiple indexes the postcode will only be run once. |
| OpBeginDateTime | datetime | Date and time the reindex statement began running. |
| OpEndDateTime | datetime | Date and time the reindex statement finished running. |
| OpRunTimeInSecs | int | How many seconds the reindex statement took. |
| TableRowCTBeginDateTime | datetime | Internal use. |
| TableRowCTEndDateTime | datetime | Internal use. |
| TableRowCTTimeInSecs | int | Internal use. |
| TableRowCT | bigint | The count of rows in the table. Therefore, all indexes for a single table will have the exact same row counts. |
| PostFragBeginDateTime | datetime | Date and time the post fragmentation statement began. The post fragmentation level is explained above in the |

| | | |
|---|---|---|
| | | Minion.IndexSettingsDB and Minion.IndexSettingsTable tables. |
| PostFragEndDateTime | datetime | Date and time the post fragmentation statement finished. The post fragmentation level is explained above in the Minion.IndexSettingsDB and Minion.IndexSettingsTable tables. |
| PostFragTimeInSecs | int | How many seconds the post fragmentation stats collection took. |
| PostFragLevel | tinyint | The fragmentation level of the index immediately after the reindex operation finished. This is an excellent way to see the effectiveness of your routines and whether you need to adjust your threshold levels for individual tables. |
| UpdateStatsBeginDateTime | datetime | Date and time update statistics began. This will only be populated if the operation is a REORG and the UpdateStatsOnDefrag column in either Minion.IndexSettingsDB or Minion.IndexSettingsTable is set to 1. The value should always be set to 1 unless you have a specific reason not to. |
| UpdateStatsEndDateTime | datetime | Date and time update statistics finished. This will only be populated if the operation is a REORG and the UpdateStatsOnDefrag column in either Minion.IndexSettingsDB or Minion.IndexSettingsTable is set to 1. The value should always be set to 1 unless you have a specific reason not to. |
| UpdateStatsTimeInSecs | int | How many seconds the update statistics statement took. |
| UpdateStatsStmt | varchar | The exact update statistics statement that was run. |
| PreCodeBeginDateTime | datetime | Date and time the precode for the table began. |
| PreCodeEndDateTime | datetime | Date and time the precode for the table finished. |
| PreCodeRunTimeInSecs | int | How many seconds the table precode took. |
| PostCodeBeginDateTime | datetime | Date and time the postcode for the table began. |
| PostCodeEndDateTime | datetime | Date and time the postcode for the table finished. |
| PostCodeRunTimeInSecs | bigint | How many seconds the table postcode took. |
| UserSeeks | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| UserScans | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| UserLookups | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |

| | | |
|---|---|---|
| UserUpdates | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastUserSeek | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastUserScan | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastUserLookup | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastUserUpdate | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| SystemSeeks | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| SystemScans | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| SystemLookups | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| SystemUpdates | bigint | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastSystemSeek | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastSystemScan | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastSystemLookup | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| LastSystemUpdate | datetime | See http://msdn.microsoft.com/en-us/library/ms188755.aspx |
| Warnings | Varchar(max) | Reserved for future use. |

Discussion:

The data available in this log includes the status of the operation, the object information, the statement used, operation type, reorg and rebuild thresholds, index usage information, and more.

## Overview of Procedures

Two separate procedures **execute index maintenance operations** for Minion Reindex: one procedure runs per database, and the other is a "Master" procedure that performs run time logic and calls the DB procedure as appropriate.

In addition, Minion Reindex comes with a **Help procedure** to provide information about the system itself.

Index maintenance procedures:

- **Minion.IndexMaintMaster** – This procedure makes all the decisions on which databases to reindex, and what order they should be in.
- **Minion.IndexMaintDB** – This procedure is called by Minion.IndexMaintMaster to perform index maintenance for a single database.
- **Minion.HELP** – Display help on Minion Reindex objects and concepts.

# Procedures Detail

## Minion.IndexMaintMaster

The Minion.IndexMaintMaster procedure makes all the decisions on which databases to reindex, and what order they should be in. This stored procedure calls the Minion.IndexSettingsDB stored procedure once per each database specified in the parameters; or, if "All" is specified, per each eligible database in sys.databases.

Minion Reindex 1.2 supports SQL Server databases that are part of an Availability Group (AG). Reindex will run for databases that are not part of an AG, and for AG primaries, but not for databases that act as a secondary in an A scenario. (AG secondary databases do not require index maintenance.)

| Name | Type | Description |
|---|---|---|
| @IndexOption | varchar | Perform maintenance only for indexes marked for online operations; only for those marked for offline operations; or for all indexes.<br><br>Valid inputs:<br>ONLINE<br>OFFLINE<br>ALL<br><br>For more information, see "How To: Reindex only indexes that are marked ONLINE = ON (or, only ONLINE = OFF)" |
| @ReorgMode | varchar | Perform maintenance only for indexes that meet the REORG threshold; only for those that meet the REBUILD threshold; or for all indexes that meet either threshold (when this is set to "All").<br><br>Note that for REORG mode, only REORG statements will be generated, even for indexes that are over the rebuild threshold. For REBUILD, only REBUILD statements will be generated.<br><br>Valid inputs:<br>All<br>REORG<br>REBUILD |
| @RunPrepped | bit | If you've collected index fragmentation stats ahead of time by running with @PrepOnly = 1, then you can use this option. It causes the index maintenance to use the saved frag stats. |

| | | |
|---|---|---|
| | | For more information, see “How To: Gather index fragmentation statistics on a different schedule from the reindex routine”. |
| @PrepOnly | Bit | Only gets index fragmentation stats, and saves to a table. This prepares the databases to be reindexed.<br><br>If @PrepOnly = 1, then @RunPrepped must be set to 0.<br><br>For more information, see “How To: Gather index fragmentation statistics on a different schedule from the reindex routine”. |
| @StmtOnly | Bit | Only prints reindex statements. This is an excellent choice for running statements manually; it allows you to pick and choose which indexes you want to do, or just see how many are over the thresholds.<br><br>For more information, see “How To: Generate reindex statements only”. |
| @Include | varchar | Use @Include to run index maintenance 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.<br><br>Examples of valid inputs include:<br>All<br>NULL<br>DBname<br>DBName1, DBname2, etc.<br>DBName%, YourDatabase, Archive% |
| @Exclude | varchar | Use @Exclude to skip index maintenance for a specific list of databases, or databases that match a LIKE expression.<br><br>Examples of valid inputs include:<br>DBname<br>DBName1, DBname2, etc.<br>DBName%, YourDatabase, Archive%<br><br>For more information, see “How To: Exclude databases from index maintenance”. |
| @LogProgress | Bit | Track the progress of index operations for this database.<br><br>The overall status is tracked in the Minion.IndexMaintLog table, while specific |

| | | |
|---|---|---|
| | | operations are tracked in the Status column Minion.IndexMaintLogDetails. |

Discussion: Minion.IndexMaintMaster is the heart and brain of Minion Reindex; it decides what needs to be done and pushes out orders to get it done. A few things you can do with Minion.IndexMaintMaster include:

- Maintain only indexes that can be done online, only those that can be done offline, or all.
- Generate and execute only reorganize statements, only rebuild statements, or both.
- Run the procedure to gather index fragmentation stats, and save them to a table. This prepares the database to be reindexed.
- Run the procedure without gathering index fragmentation stats. This requires that the index fragmentation data has already been collected.
- Choose to maintain a specific set of databases, via the @Include parameter. (E.g., @Include='DB1, DB2, DB3'…)
- Choose to maintain all databases
- Choose to maintain all databases, with specific exclusions, via the @Exclude parameter.
- Only print reindex statements, do not run. This is an excellent choice for running statements manually; it allows you to pick and choose which indexes you want to maintain, or just see how many indexes are over the thresholds.
- Have every step of the run printed in the log so you can watch the progress (called Live Insight). This option is on by default.

## Minion.IndexMaintDB

The Minion.IndexMaintDB stored procedure performs index maintenance for a single database. Minion.IndexMaintDB is the procedure that creates and runs the actual reindex statements for tables that meet the criteria stored in the settings tables (Minion.IndexSettingsDB and Minion.IndexSettingsTable).

**IMPORTANT**: We HIGHLY recommend using Minion.IndexMaintMaster for all of your reindex operations, even when reindexing a single database. Do not call Minion.IndexMaintDB to perform index maintenance.

The Minion.IndexMaintMaster procedure makes all the decisions on which databases to reindex, and what order they should be in. It's certainly possible to call Minion.IndexMaintDB manually, to run an individual database, but we instead recommend using the Minion.IndexMaintMaster 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.IndexMaintMaster procedure; if you get used to using Minion.IndexMaintMaster now, then things will always work as intended.

| Name | Type | Description |
|---|---|---|
| @DBName | Varchar | Database name to perform maintenance on. |
| @IndexOption | varchar | Perform maintenance only for indexes marked for online operations; only for those |

| | | |
|---|---|---|
| | | marked for offline operations; or for all indexes.<br><br>Valid inputs:<br>ONLINE<br>OFFLINE<br>ALL<br><br>For more information, see “How To: Reindex only indexes that are marked ONLINE = ON (or, only ONLINE = OFF)” |
| @ReorgMode | varchar | Perform maintenance only for indexes that meet the REORG threshold; only for those that meet the REBUILD threshold; or for all indexes that meet either threshold (when this is set to “All”).<br><br>Note that for REORG mode, only REORG statements will be generated, even for indexes that are over the rebuild threshold. For REBUILD, only REBUILD statements will be generated.<br><br>Valid inputs:<br>All<br>REORG<br>REBUILD |
| @RunPrepped | bit | If you've collected index fragmentation stats ahead of time by running with @PrepOnly = 1, then you can use this option. It causes the index maintenance to use the saved frag stats.<br><br>For more information, see “How To: Gather index fragmentation statistics on a different schedule from the reindex routine”. |
| @PrepOnly | bit | Only gets index fragmentation stats, and saves to a table. This prepares the database to be reindexed.<br><br>If @PrepOnly = 1, then @RunPrepped must be set to 0.<br><br>For more information, see “How To: Gather index fragmentation statistics on a different schedule from the reindex routine”. |
| @StmtOnly | bit | Only prints reindex statements. This is an excellent choice for running statements manually; it allows you to pick and choose |

| | | |
|---|---|---|
| | | which indexes you want to do, or just see how many are over the thresholds.<br><br>For more information, see “How To: Generate reindex statements only”. |
| @LogProgress | Bit | Track the progress of index operations for this database.<br><br>The overall status is tracked in the Minion.IndexMaintLog table, while specific operations are tracked in the Status column Minion.IndexMaintLogDetails. |

## Minion.HELP

Use this stored procedure to get help on any Minion Reindex object without leaving Management Studio.

| Name | Type | Description |
|---|---|---|
| @Module | Varchar | The name of the module to retrieve help for.<br><br>Valid inputs include:<br>NULL<br>Reindex |
| @Name | varchar | The name of the topic for which you would like help.<br><br>If you run Minion.HELP by itself, or with a @Module specified, it will return a list of available topics. |

Examples:

For introductory help, run:

```
EXEC Minion.HELP;
```

For introductory help on Minion Reindex, run:

```
EXEC Minion.HELP 'Reindex';
```

For help on a particular topic – in this case, the Top 10 Features – run:

```
EXEC Minion.HELP 'Reindex', 'Top 10 Features';
```

# Overview of Jobs

When you install Minion Reindex, it creates and schedules a daily reorg job, and a weekly rebuild job:

- *MinionReindexDBs-All-REORG* – Runs Daily – 3:00 AM except for Friday – to complete lightweight defragmenting (reorganize).
- *MinionReindexDBs-All-All* – Runs once weekly – Fridays at 3:00 AM - to thoroughly defragment indexes (rebuild).

For information on changing schedules, see the Quick Start topic "Change Schedules".

# Minion Reindex Troubleshooting

## Why is a certain database not being processed?

There are a few reasons why a database could be skipped.

1. It could be excluded in the @Excluded parameter of the SP.
2. It could be excluded in the Exclude column in the Minion.IndexSettingsDB table.
3. It could be excluded in the Minion. DBMaintRegexLookup table.
4. It could be OFFLINE or some other troubled state.
5. There could be no indexes in the database or none of them have exceeded the threshold.
6. There could be a missing entry for 'MinionDefault' in the Minion.IndexSettingsDB table.

## Not all indexes in the Minion.IndexMaintLogDetails table are marked 'Complete'

This is often do to an unhandled exception or caused by someone manually stopping the routine before it is finished. Unhandled exceptions aren't very common but there are still some errors that can halt the database run.

## Nothing happens when I run a specific database

There are a few reasons you could see this behavior.

1. There could be no indexes in the database or none of them have exceeded the threshold.
2. The settings in the tables could be incorrect or missing. While there are few columns in the settings tables that are mandatory, there are some. ReorgThreshold, RebuildThreshold, Exclude, ReindexGroupOrder, ReindexOrder are the only columns I can think of that need to be populated.
3. The SP was set with @RunPrepped = 1 and there are no rows in the Minion.IndexTableFrag table for that database. This is because the PrepOnly was never run or failed.
4. You're running it with @StmtOnly = 1.

## Some tables aren't reindexing at the proper threshold

The only thing that might cause this would be a table override. Check that the Minion.IndexSettingsTable table doesn't have an entry for the problem tables.

## ONLINE was set as the rebuild option, but all or some of the indexes are being done OFFLINE.

Minion Reindexing strives to run no matter what. If you've got the ONLINEopt column set and some indexes are being done OFFLINE, then you could be on an edition of SQL Server that doesn't support online reindexing. In this case, Minion Reindexing will change it to OFFLINE mode for you.

You could also have a legacy data type in the index itself, and for versions of SQL Server under 2014, this automatically means OFFLINE mode reindexing. These legacy types are varchar(max), nvarchar(max), text, image, and also includes, xml, and the spatial data types. If it's a clustered index in question, it'll be done offline if the table itself has a legacy data type. This is a SQL Server limitation.

# Revisions

| Version | Release Date | Changes |
|---|---|---|
| 1.0 | October 2014 | • Initial release. |
| 1.1 | January 2015 | • Minion Reindex handles all nonstandard naming (e.g., object names with spaces or special characters.)<br>• Added support for Availability Group replicas. Basic AG support has been added by only permitting Minion Reindex to run on an AG Primary DB.<br>• Fixed formatting in Minion.Help stored procedure. |
| 1.2 | September 2015 | Issues resolved:<br>• Fix: MR failed when running on BIN collation.<br>• Fix: Help didn't install if Minion Backup was installed.<br>• Fix: MR didn't handle XML and reorganize properly.<br>• Fix: ONLINE/OFFLINE modes were not being handled properly.<br>• Fix: XML indexes were put into ONLINE mode instead of OFFLINE mode.<br>• Fix: Situation where indexes could be processed more than once.<br>• Update: Increased Status column in log tables to varchar(max).<br>• Fix: Status variable in stored procedures had different sizes.<br>• Fix: Wrong syntax created for Wait_at_low_priority option.<br>• Fix: Reports that offline indexes were failing when it's set to online instead of doing it offline.<br>New features:<br>• Error trapping and logging is improved. MR is able to capture many more error situations now, and they all appear in the log table.<br>• Statement Prefix – All of the Settings tables now have a StmtPrefix column. See document for details. Note: To ensure that your statements run properly, you must end the code in this column with a semicolon.<br>• Statement Suffix – All of the Settings tables now have a StmtSuffix column. See document for details. Note: To ensure that your statements run properly, you must end the code in this column with a semicolon. |

# FAQ

**How do I install Minion Reindex?**

For information on this, see the "Quick Start" on page 1, or use the Minion Reindex help function: EXEC Minion.Help 'Reindex', 'Quick Start';

**Do I really need SQL Server 2005 or above / xp_cmdshell / PowerShell 2.0?**

Yep. Minion Reindex does an awful lot for you. To simplify a great many things, we've decided not to support SQL Server 2000 and previous versions (er, sorry about that), to require xp_cmdshell, and to make use of PowerShell 2.0 or above. There's no such thing as a free lunch, they say, but this particular lunch is very very cheap.

**Why does Minion Backup use xp_cmdshell instead of SQL CLR?**

First, it would be a burden to require users to have CLR installed on every single server on the network. Not only that, but the database setting would have to be set to UNTRUSTWORTHY for the things MR needs to do; or else, we would have a far more complex scenario on hand, and that level of complication just for backups is not a good setup.

Using SQL CLR would also put us in the business of having to support different .NET framework versions, which would also complicate things.

Cmdshell is the best choice because it's simple to lock down to only administrators, and it adds no extra "gotchas". There were times when it would have been easier to use CLR, but we simply can't require that everyone enables CLR.

Just be sure to lock down cmdshell. For instructions on this, see this article by Sean:
http://www.midnightdba.com/DBARant/?p=1243

And for further reading, here is the link to one of Sean's rants on the topic:
http://www.midnightdba.com/DBARant/?p=1204

**Why is Minion Reindex better than [some other index maintenance solution]?**

This is a very big question, and I'm tempted to just point to the entire body of documentation. But briefly: (1) Minion Reindex provides vastly improved logging and insight, including live insight into the active process. (2) It provides ease of management, especially through reducing the number of jobs (or job steps) required. (3) Minion Reindex gives you fine-grained control, in the form of database- and table-level configurations and exclusions. (4) And, Minion Reindex is massively scalable, where other solutions require a "one by one by one" approach to deployment and configuration.

For more on this topic, see the MidnightSQL.com article "How does Minion Reindex differ from Ola Hallengren's IndexOptimize?", at http://www.midnightsql.com/how-does-minion-reindex-differ-from-ola-hallengrens-indexoptimize/

**Does Minion Reindex support Availability Groups?**

Yes, as of version 1.1.

**Does Minion Reindex support clusters?**

Clusters have no impact on Minion Reindex. So, yes.

**I have an old database that has objects named with keywords and spaces. Does Minion handle that?**

Yes, as of version 1.1.

Have a questions? Get in contact at http://www.midnightsql.com/contact-me/ or https://minionware.desk.com/.

# 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, MinionWare, or just "Sean" and "Jen". We're all good.

# Contents