Admin Handbook

sqmPartitionTool

Automatic SQL Server table partitioning, runbooks for daily DBA operations

dtcSoftware · Uwe Janke · www.powershelldba.de
As of: 2026-11-05

sqmPartitionTool - Admin Handbook (en-US)

Automatic SQL Server table partitioning: converting existing tables, sliding-window maintenance, retention/archiving of old partitions and migration of whole tables into a separate archive database, either with the GUI wizard or entirely from PowerShell/CLI.

Audience: SQL Server DBAs who operate the tool (not the development of the module itself). For the version history see CHANGELOG.md, for a short overview see README.md. The German edition of this handbook is AdminHandbuch.md.

As of: 2026-11-05, sqmPartitionTool 1.19.1.0 (with sqmDataTransfer 0.1.23.0).

Contents

- 1. Overview: which workflow fits my situation?
- 2. Requirements and installation
- 3. Runbook A: partition an existing table in place
- 4. Runbook B: set up automatic maintenance (sliding window + retention)
- 4a. AlwaysOn / availability groups
- 5. Runbook C: migrate a table into an archive database (cutover)
- 5a. Runbook D: copy an already partitioned table with a new partitioning
- 5b. How the data is copied
- 5c. Runbook E: remove partitioning again
- 6. BoundaryType/SurrogateDateFormat reference
- 7. GUI wizard: step by step
- 8. Troubleshooting and known limitations
- 9. Security notes

1. Overview

The module covers six different scenarios that can be used independently of each other. Which one fits depends on **where the data should end up** and **whether the table stays active**:

Scenario	Function	Source table afterwards	When it makes sense
A - in-place partitioning	`Invoke-sqmTablePartitionConversion`	Stays in the same DB, now partitioned	The table should be partitioned but stay in the same database
B1 - sliding-window extension	`New-sqmPartitionExtendJob` (SQL Agent job)	Unchanged (only new empty partitions are added)	After A: make sure the last partition never fills up

B2 - retention/archiving	`New-sqmPartitionRetentionJob` (SQL Agent job)	Old partitions are deleted or archived first	After A: remove old data automatically after X months/years
C - archive DB migration + cutover	`Invoke-sqmTableArchiveMigration`	Renamed, replaced by a view onto the archive DB	The whole table should move permanently to another (typically smaller or more slowly attached) database while application code keeps running unchanged
D - newly partitioned copy	`Copy-sqmPartitionedTable`	**Unchanged, stays active** (no rename, no cutover)	An **already partitioned** table should additionally exist as an independent copy with a **different** granularity/filegroup strategy in another database (e.g. a reporting extract with coarser granularity)
E - remove partitioning	`Remove-sqmTablePartitioning`	Stays in the same DB, unpartitioned on one filegroup again	Partitioning should be taken back (e.g. undo the conversion from A)

AlwaysOn: if the database is in an availability group, additional steps apply to the jobs (B1/B2), see section 4a.

Rule of thumb: if the table should **stay in the source database**, use A (plus optional B1/B2). If the table should move **completely into another database** (e.g. an archive instance, a separate database with lower backup/storage requirements), use C. If the source table is **already partitioned** and should **keep existing in parallel** elsewhere with a different partitioning (no cutover, no rename), use D.

---

2. Requirements and installation

• PowerShell 5.1 or higher (the GUI needs the desktop CLR/WinForms; it is still available under PowerShell 7 on Windows, but not under PowerShell 7 on Linux/macOS).

• Modules `dbatools`, `sqmSQLTool` (>= 1.9.2.0) and **`sqmDataTransfer` (>= 0.1.23.0)**. Since 1.15.0.0 sqmDataTransfer provides the copy engine for runbooks C, D and the relocation (see section 5b). sqmSQLTool and sqmDataTransfer are not on the PowerShell Gallery and must be installed **beforehand**.

• A SQL Server login with sufficient rights on the target database(s): `ALTER` on the affected table(s)/database(s), `CREATE`/`ALTER PROCEDURE`, and for the SQL Agent jobs (scenario B) rights on `msdb`.

• With **AlwaysOn**: the modules must be installed on **every** replica on which a maintenance job should run (section 4a).

• For scenarios C and D: the **target database must be created by the admin beforehand**, the tool does not create it automatically (a deliberate decision, because file sizes, paths and recovery model are admin-specific).

Install in this order (as administrator, from the respective repository folders):

```
# per repository folder, as administrator
.\sqmSQLTool\Install.ps1       -Scope AllUsers
.\sqmDataTransfer\Install.ps1  -Scope AllUsers
.\sqmPartitionTool\Install.ps1 -Scope AllUsers
# with a locked execution policy, each time:
# powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\<Module>\Install.ps1 -Scope AllUsers
```

The installer of sqmPartitionTool checks that sqmSQLTool and sqmDataTransfer exist in the same scope and are recent enough, and names missing or outdated modules. `dbatools` is installed from the Gallery if needed. Which version is actually running is shown by the GUI wizard in the window title (version and load path), from the console:

```
Get-Module sqmPartitionTool, sqmDataTransfer -ListAvailable |
   Select-Object Name, Version, ModuleBase
```

Start menu entry: the installation creates the entry `sqmPartitionTool` in the Start menu (AllUsers: for all users, CurrentUser: only for the current user). It starts the GUI without a visible console window. Pinning to Start cannot be automated on Windows 10/11: pin it once manually via right-click on the entry > *Pin to Start*.

The connection uses Windows authentication (default) or SQL Server authentication (`-SqlCredential`, also selectable in step 0 of the GUI wizard).

---

3. Runbook A: partition an existing table in place

Goal: an existing, unpartitioned table is partitioned in the same database.

1. **Check whether the table is suitable.** `powershell Test-sqmPartitionReadiness -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" -Table "OrderHistory" -PartitionColumn "OrderDate"` Reports, among other things, whether `-AllowKeyChange` is needed (the PRIMARY KEY/UNIQUE constraint must be extended by the partition column) and whether incoming foreign keys/triggers exist (relevant for `-Method BatchedSwap`, see below).

2. **Choose granularity and filegroup strategy**: Month/Quarter/Year, and `Single` (one filegroup for all partitions) or `PerPeriod` (one filegroup per period: better I/O isolation, more administration). With **`-FilePath`** the new filegroup files go to a directory **on the SQL Server** of your choice, e.g. on a dedicated drive (`-FilePath 'G:\SQLData\Partitions'`); a missing directory is created, a drive that does not exist on the server aborts before any change. Without it: the default data path of the instance. Applies only to **newly** created filegroups, an existing one is not moved. The same parameter is available for `Invoke-sqmTableArchiveMigration` and `Copy-sqmPartitionedTable`.

3. **Run the conversion:** `powershell Invoke-sqmTablePartitionConversion -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" ` -Table "OrderHistory" -PartitionColumn "OrderDate" -Granularity Month ` -FilegroupStrategy Single -AllowKeyChange` Depending on the starting point the function picks the matching implementation automatically: - Clustered index/PK without the partition column in the key: `ALTER TABLE ... DROP/ADD CONSTRAINT`. - Clustered index already fitting: `CREATE CLUSTERED INDEX ... WITH (DROP_EXISTING=ON)`. - Heap: new clustered index directly on the partition scheme, or `-Method NewTableSwap` for very large heaps. - **`-Method BatchedSwap`** (heap **and** indexed/PK tables): for very large tables on storage with **little free space**. Builds a new, empty partitioned copy and moves the data in segments (per boundary period, further divided by `-BatchSize`), with periodic `DBCC SHRINKFILE` while the old table empties. **Limitation:** aborts with an error if the table has incoming foreign keys or triggers; use `-Method Default`/`NewTableSwap` for those.

4. **Check the result:** `powershell Get-sqmPartitionStatus -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" -Table "OrderHistory"`

5. The table is registered automatically in `master.dbo.sqm_PartitionRegistry` (unless `-NoRegister`), a prerequisite for scenario B.

4. Runbook B: set up automatic maintenance

Assumes the table is already partitioned and registered (runbook A).

1. **Sliding-window extension** (makes sure a few periods ahead always exist as empty partitions, `-FutureBufferPeriods` from runbook A): `powershell New-sqmPartitionExtendJob -SqlInstance "SQL01"` Creates a SQL Agent job that runs `sqm_ExtendPartitionWindow` (T-SQL procedure, reads the registry) periodically. Idempotent: running it several times on the same day changes nothing.
2. **Retention/archiving** (removes partitions older than the configured retention period): `powershell Register-sqmPartitionTable -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" -Table "OrderHistory" ` -PartitionColumn "OrderDate" -PartitionFunctionName "..." -PartitionSchemeName "..." ` -Granularity Month -BoundaryType Date -RetentionValue 36 -RetentionUnit Months ` -ArchiveEnabled -ArchiveDatabaseName "SalesArchive" New-sqmPartitionRetentionJob -SqlInstance "SQL01"` With `-ArchiveEnabled`, expiring partitions are copied **before** removal into the given archive database via `Invoke-sqmPartitionArchive` (SWITCH PARTITION, batch copy, then `MERGE RANGE`) instead of simply being deleted. Without `-ArchiveEnabled`, expired partitions are deleted irrevocably: test first!
3. **Check status/progress:** `powershell Get-sqmPartitionRegistry -SqlInstance "SQL01"`

Important: this path concerns **individual partitions expiring one by one** of a table that stays active and partitioned, not the whole table at once. For an immediate migration of the whole table see runbook C.

Ad hoc instead of a job: immediate retention for a single table

`New-sqmPartitionRetentionJob` covers continuous operation (weekly, all registered tables). For an immediate one-off run (a test before setting up the job, an emergency cleanup "we need space NOW", or a table that is not (yet) registered at all) there is `Invoke-sqmPartitionRetention`:

```
Invoke-sqmPartitionRetention -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" `
   -Table "OrderHistory" -RetentionValue 120 -RetentionUnit Months
```

Immediately removes all partitions older than the given cutoff (here: 120 months = 10 years), using the same SWITCH PARTITION + MERGE RANGE logic as above (`Invoke-sqmPartitionArchive` behind it), optionally with `-ArchiveDatabaseName` for a copy before removal. Needs **no** prior registration: the boundary values of each partition are read and evaluated directly. A single confirmation prompt for the whole run (shows the number of affected partitions up front), not one per partition. `-WhatIf` shows how many partitions would be affected without changing anything.

4a. AlwaysOn / availability groups

The maintenance jobs from runbook B and the functions of the module are AlwaysOn capable. Four things need attention, because AlwaysOn does **not** replicate them or allows them only on the primary:

Topic	Problem if ignored	Solution in the module
SQL Agent jobs	Jobs live in `msdb`, not in the AG. After a failover the new primary has no job.	`-AllReplicas` creates the job on **every** replica.

Registry `master.dbo.sqm_PartitionRegistry`	`master` is not part of the AG. The new primary has no registry or an outdated one, and both jobs run into the void.	`Sync-sqmPartitionRegistry` copies the registry to all replicas (`-AllReplicas` calls it automatically).
DDL (SPLIT/MERGE/SWITCH)	On a secondary the database is not writable, the job would end with an error.	`sqm_ExtendPartitionWindow` and the retention sweep **skip** databases that are not locally `ONLINE` and `READ_WRITE`. The job succeeds on a secondary, only the current primary works.
`ADD FILE` (new filegroups)	The file is created on the secondaries via the log **at the identical path**. If the directory is missing there, the database is suspended on that secondary.	`New-sqmPartitionFilegroupPlan` (and with it conversion and copy) checks the directory on all replicas beforehand and aborts.

1. Set up the jobs on all replicas (against the listener or any node):

```
New-sqmPartitionExtendJob    -SqlInstance "AGLISTENER01" -AllReplicas -WhatIf   # preview
New-sqmPartitionExtendJob    -SqlInstance "AGLISTENER01" -AllReplicas
New-sqmPartitionRetentionJob -SqlInstance "AGLISTENER01" -AllReplicas
```

- The replicas are determined from `sys.availability_replicas` of all availability groups of the instance. On each replica the job is created with its **own** instance name (the retention step calls the sweep script with `-SqlInstance <local node>`).
- Finally the registry is copied from the given `-SqlInstance` (listener = primary) to the other replicas.
- The retention job copies the sweep script via `\\<Host>\C$` into the module folder of the other hosts. `sqmPartitionTool`, `sqmSQLTool` and `dbatools` must be installed there (section 2), and the executing account needs access to the admin share.
- Without AlwaysOn, `-AllReplicas` is ignored with a warning and the job is created only on the given instance.
- **Without `-AllReplicas`** the job is created only on the given instance. With the listener as `-SqlInstance` that is the current primary, after a failover the job is missing on the new one.

2. Sync the registry after every change. `Register-sqmPartitionTable`, `Remove-sqmPartitionRegistration` and the conversion write only to the registry of the connected instance. Afterwards:

```
Sync-sqmPartitionRegistry -SqlInstance "AGLISTENER01" -WhatIf
Sync-sqmPartitionRegistry -SqlInstance "AGLISTENER01"
# explicit targets:  -TargetInstance "SQL02","SQL03"
# keep rows that exist only on the target:  -NoDelete
```

The source is authoritative: missing rows are added, differing rows are aligned, rows that exist only on the target are removed (unless `-NoDelete`). `LastExtendedAt` and `LastRetentionRunAt` stay untouched on each replica, they belong to that replica's own job runs. On an instance without AlwaysOn nothing happens.

3. New filegroups. `-FilePath` (or the default data path of the primary) must exist on all replicas: same path, write permission for the SQL Server service account. If it is missing on a replica, the function aborts **before** the `ADD FILE`, naming the replica. `-SkipReplicaPathCheck` turns the check off when the paths are known to exist.

4. What happens on a secondary. The extend job reports e.g. `... 1 uebersprungen (nicht schreibbar/Secondary), 0 Fehler.` (German output text: skipped, not writable/secondary), the retention job writes `'<DB>' uebersprungen (Status ONLINE, READ_ONLY ...)` per database. Both are normal operation, not errors. A database that does not exist locally at all remains an error. A deliberately write-protected (`READ_ONLY`) database is skipped for the same reason.

5. Manual calls. `Invoke-sqmPartitionRetention`, `Invoke-sqmTablePartitionConversion`, `Copy-sqmPartitionedTable` etc. perform DDL: always call them against the **listener** or the current primary. Against a secondary they fail on the write-protected database.

Limits and notes:

- The **archive database** (`-ArchiveDatabaseName`) must be on the same instance. It should itself be a member of the availability group, or exist under the same name on every node, otherwise archiving fails after a failover.
- The registry sync is **manual** (or happens when the jobs are created with `-AllReplicas`), not continuous. Anyone who registers often calls `Sync-sqmPartitionRegistry` at the end of their own script.
- Verification status: the skip logic (READ_ONLY database in the T-SQL procedure and in the sweep script), the registry sync (insert, update, delete, `-NoDelete`, apostrophes in names) and the behavior on an instance without AlwaysOn were tested in the lab. Replica detection and the path check across real secondaries were verified in the lab only for the non-AG case because there is no availability group: start the first use in a real AG with `-WhatIf`.

---

5. Runbook C: migrate a table into an archive database

Goal: an active (not yet partitioned) table is moved **completely** into a partitioned copy in a separate archive database. At the end the source table is renamed and replaced by a compatibility view, so existing application code keeps accessing the same table name, now transparently against the archive copy.

What happens on the call:

- On the first call an empty structure copy is created in the archive database and partitioned by month over the actual value range of the source (internally the same logic as runbook A). The copy is registered automatically for sliding-window maintenance. `-PrimaryKeyFromUniqueIndex <Index>` turns a unique nonclustered index of the source into the PRIMARY KEY CLUSTERED of the archive copy; without this option the copy has no unique key.
- Then **one month per chunk** is transferred with the copy engine of sqmDataTransfer (`Copy-sqmTableData`, SqlBulkCopy, see section 5b). The console shows rows, duration and rows/s per month.
- **Resuming after an abort:** each call counts the rows per month in source and archive (one GROUP BY each). A month with the same count is skipped, a month with a different count (the remainder of an aborted run) is emptied in the archive and copied again. No key is needed for this. A month only counts as `Completed` in `dbo.sqm_ArchiveMonthLog` (source database) after the row count check has passed.
- By default the migration ends at the **previous month**. With **`-IncludeOpenPeriods`** **everything** is transferred, including the current month and later months with data. Open months are copied completely again on every call, so rows that were written to or changed in the still active source after the first run also arrive. They stay `InProgress` in the log.
- **`-PurgeSourceAfterArchive`** deletes each completed month from the source after the row count check (in batches) and returns the space via `DBCC SHRINKFILE` (`-ShrinkAfterEveryNPeriods`, `-AggressiveShrink`). Open months are **never** deleted: rows arriving between the count and the DELETE would otherwise be lost.
- **`-CutoverToArchiveView`** renames the source table at the end (suffix `_Original`, `-RenamedTableSuffix`) and creates a view onto the archive copy under the old name. With `-IncludeOpenPeriods`, the final reconciliation of the

open months, the rename and the view creation run in **one transaction under an exclusive table lock**: no row can be lost between the last chunk and the rename. For this the final reconciliation needs a unique key (`-PrimaryKeyFromUniqueIndex` or `-KeyColumn`, up to 5 columns). The renamed original table is **never** deleted automatically.

- **Self-healing after earlier attempts:** if the archive table does not exist (any more), stale log entries of this table are reset instead of wrongly skipping months. An unused partition function/scheme of the same name from an earlier run is recreated with the current boundaries instead of being reused.

Procedure without the GUI, step by step. The same function is called several times with the same base parameters, each call continues where the previous one stopped. Only step 5 cannot easily be undone.

```
# 0. Common parameters
Import-Module sqmPartitionTool
$p = @{
   SqlInstance              = 'SQL01'
   Database                 = 'Sales'
   Schema                   = 'dbo'
   Table                    = 'Bookings'
   ArchiveDatabaseName      = 'SalesArchive'
   DateColumn               = 'BOOKDATE'      # INT in the format YYYYMMDD
   PrimaryKeyFromUniqueIndex = 'UX_Bookings'   # becomes the PK of the archive copy
   IncludeOpenPeriods       = $true           # everything, incl. the current month
}

# 1. Check without changing anything
Test-sqmPartitionReadiness -SqlInstance SQL01 -Database Sales -Schema dbo `
   -Table Bookings -PartitionColumn BOOKDATE
Invoke-sqmTableArchiveMigration @p -WhatIf

# 2. Optional: only create the partitioned archive table and look at it
Invoke-sqmTableArchiveMigration @p -CreateArchiveTableOnly

# 3. Transfer the data, repeatable any number of times (resumes, reconciles the open month again)
Invoke-sqmTableArchiveMigration @p
#    in stages:           -StartPeriod 202401 -EndPeriod 202406
#    little disk space:   -PurgeSourceAfterArchive

# 4. Check
Get-sqmPartitionStatus -SqlInstance SQL01 -Database SalesArchive -Schema dbo -Table Bookings

# 5. Finish: final reconciliation + rename + view in one transaction
Invoke-sqmTableArchiveMigration @p -CutoverToArchiveView

# 6. Create future monthly partitions automatically
New-sqmPartitionExtendJob -SqlInstance SQL01
```

Watching progress: besides the console output per month, the state can be queried directly at any time:

```
SELECT YYYYMM, Status, RowsArchived, StartedAt, CompletedAt
FROM dbo.sqm_ArchiveMonthLog              -- in the source database
WHERE TableName = N'Bookings'
ORDER BY YYYYMM;
```

After a successful cutover: remove the renamed original table (`Bookings_Original`) manually after checking. Without `-IncludeOpenPeriods` it still contains the last open month.

When `-Method BatchedSwap` (runbook A) instead of runbook C?

`-Method BatchedSwap` partitions **inside the same database** (no database switch, faster batches since no cross-DB communication is needed). It fits when the table should stay in the source database but there is little disk space for a classic one-step conversion. Runbook C is the right choice when the data should move **permanently to another database** (typically a separate, cheaper or differently backed up archive instance).

5a. Runbook D: copy an already partitioned table with a new partitioning

Goal: an **already partitioned**, still active table should additionally (not instead) exist as an independent, **newly partitioned** copy in another database of the same instance, e.g. with coarser granularity for reporting, or as a test extract before a planned change of the production table. Unlike runbook C there is **no cutover**: the source table is never renamed, never replaced by a view and stays completely unchanged.

1. **Create the target database** (admin task, not automated, same reasoning as for C).
2. **Start the copy** (without `-TargetTableName` the copy gets the name of the source): `powershell Copy-sqmPartitionedTable -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" -Table "OrderHistory" ` -TargetDatabaseName "SalesReporting" -Granularity Year -Confirm:$false` Procedure in detail: - Checks that the source table really is already partitioned (otherwise an error pointing to runbook A/C) and derives the partition column from the existing partition scheme unless `-PartitionColumn` explicitly names another column. - On the first call creates the new partitioning (filegroups, partition function/scheme) in the target database and creates a structurally identical table there (columns, indexes, PK/UNIQUE constraints; foreign keys and triggers are **not** copied). - Copies **one partition of the new target table per chunk** with the copy engine of sqmDataTransfer (section 5b). Counting is done per partition via `$PARTITION` of the new partition function: complete partitions are skipped, an incomplete one is emptied in the target and copied again. **No key needed**; a given `-KeyColumn` is ignored since 1.15.0.0 (warning). - `-MaxDurationMinutes` ends the run cleanly between two partitions, a new call resumes. - **`-CreateTableOnly`** creates only the newly partitioned, empty target table (no data, no registration), e.g. to inspect it before the copy or to put the copy into a later maintenance window. A call without the switch then copies into the existing table. If the target table already exists, nothing happens (status `TargetTableExists`). - **`-FilePath`** puts the new filegroups of the target database on a drive of your choice: `powershell Copy-sqmPartitionedTable -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" -Table "OrderHistory" ` -TargetDatabaseName "SalesReporting" -Granularity Year -FilePath 'G:\SQLData\Reporting' -CreateTableOnly` - Registers the new table in `sqm_PartitionRegistry` (unless `-NoRegister`); runbook B1 can then be set up as for any other partitioned table.
3. **Verification:** row counts of source and copy are reconciled at the end. If they differ (e.g. because new rows arrived in the still active source during the copy), the function reports an error; a new call copies the difference.

5b. How the data is copied

Since 1.15.0.0 all copies **between databases** run through the copy engine of **sqmDataTransfer**, the same path as its chunk transfer, which is used in production with tables of several hundred million rows. Before that, sqmPartitionTool used its own T-SQL batches (MERGE or keyset `INSERT ... SELECT TOP`); without a matching index every batch read the whole table, and the console showed no feedback for hours.

What the engine brings:

- **SqlBulkCopy** with explicit column mapping by name (a computed column in the middle of the table cannot shift columns).
- If a chunk fails, the source query on the server is **cancelled** instead of still reading the rest of the chunk over the network.
- Batch size is capped automatically for columnstore targets.
- Progress per chunk (`Write-Progress`) plus a console line with rows/s.
- **Reading by index seek instead of scan per chunk** (since 1.15.1.0): on a heap or a clustered index that does not start with the date/partition column, SQL Server otherwise reads the whole table per chunk for `WHERE <month>`

on large tables. If an index with the column first exists and there are at least 4 chunks, each chunk is read `WITH (FORCESEEK)`. Without such an index, create one before a large run.

Function	A chunk is	Resume after abort
`Invoke-sqmTableArchiveMigration`	one calendar month	Row count per month, source against archive
`Copy-sqmPartitionedTable`	one partition of the new target table	Row count per target partition via `$PARTITION`
`Invoke-sqmTableRelocation`	automatically detected chunk column (date, period or `YYYYMMDD`, by month), otherwise the whole table	Row count per chunk

In all three cases a chunk with the same row count on both sides is skipped and a chunk with a different count is emptied in the target and copied again. Without `-BatchSize` the default batch size of sqmDataTransfer applies (`Get-sqmTransferConfig DefaultBatchSize`, 500000).

Deliberately unchanged, server side: the retention archiving of individual expired partitions (`Invoke-sqmPartitionArchive`, staging table into the archive DB of the same instance via `DELETE ... OUTPUT INTO`, transactional per batch, runs unattended in the retention job) and the in-place conversion `-Method NewTableSwap` (`INSERT ... WITH (TABLOCK)` in the same database). Going through the client would only add a network hop for both.

In practice with very large tables:

- Start the run **on the SQL Server host itself**. SqlBulkCopy reads through the client; from a workstation every row crosses the network twice.
- `ASYNC_NETWORK_IO` on the **writing** session (`INSERT BULK` in the target) means: the target is waiting for the client, the bottleneck is reading the source or the client. Check with `Docs/Diagnose-ChunkTransfer-Quelle.sql` from sqmDataTransfer.
- `ASYNC_NETWORK_IO` on the **reading** session is normal as long as writing is slower than reading. The real bottleneck shows on the **writing** session (`INSERT BULK`): `WRITELOG` or log growth (with FULL recovery every row is logged, size the log accordingly beforehand), `PAGEIOLATCH` (storage), `LCK_M_*` (blocking).
- During a copy, **no index maintenance, no TRUNCATE and no partition SPLIT** on the source table: the reading session holds a schema lock, and waiting DDL blocks everything queued behind it.

5c. Runbook E: remove partitioning again

Goal: a partitioned table should become a normal, unpartitioned table on one filegroup again, e.g. because the partitioning did not pay off, the table has become much smaller, or a conversion (runbook A) should be taken back. The data stays unchanged, only its storage location changes.

1. **Look without changing anything** (`-WhatIf` returns the planned DDL): `powershell $plan = Remove-sqmTablePartitioning -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" ` -Table "OrderHistory" -RemoveEmptyFilegroups -WhatIf $plan.Statements`

2. **Execute:** `powershell Remove-sqmTablePartitioning -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" ` -Table "OrderHistory" -TargetFilegroup "PRIMARY" -RemoveEmptyFilegroups -Confirm:$false` Procedure in detail: - **Pre-flight without changes:** table exists, not a memory-optimized table, no XML/spatial indexes (remove them first), target filegroup exists. Without `-TargetFilegroup` the default filegroup of the database is used (usually PRIMARY). - **Space check:** the data physically moves to the target filegroup. Beforehand it is checked whether the table with all indexes (used pages) fits there: free space in the files plus possible autogrowth up to the limit of the drive or file. If it does not fit, the function aborts with the numbers. Below 20 % reserve (sort space of the rebuild) or if autogrowth is needed: warning. In FULL recovery an additional note about log volume. `-SkipSpaceCheck` turns the check off. - **Clustered index / PRIMARY KEY / UNIQUE constraint and every nonclustered index** are rebuilt with `CREATE ... WITH (DROP_EXISTING = ON) ON [<Filegroup>]` with unchanged definition: key, INCLUDE, filter, uniqueness, fill factor, lock options and compression are preserved. Constraints stay in place even if foreign keys reference them. LOB data moves along. Disabled indexes are moved too and disabled again afterwards. - **Heap:** SQL Server cannot move a heap directly. A temporary clustered index `sqmUnpartitionTmp` is created on the target filegroup and removed again immediately. The compression of the heap is preserved. - **Columnstore:** on a partitioned table a columnstore index must be partition-aligned, a move via `DROP_EXISTING` is rejected by SQL Server. A nonclustered columnstore index, and with a clustered columnstore index all nonclustered indexes, are therefore removed before the rebuild and recreated afterwards with the same definition (PRIMARY KEY/UNIQUE via `ALTER TABLE ... ADD CONSTRAINT`). A clustered columnstore index itself is removed, the table moved as a heap and the index recreated on the target filegroup (the data is decompressed and recompressed once). If foreign keys reference a constraint that has to be recreated like this, the function aborts before any change. - **Partition scheme and partition function** are dropped as soon as no other object uses them. If several tables share a scheme, it stays with a warning and is cleaned up when the last table is removed. - **`-RemoveEmptyFilegroups`:** the filegroups of the scheme that are empty afterwards are removed together with their files (`DBCC SHRINKFILE ... EMPTYFILE`, `REMOVE FILE`, `REMOVE FILEGROUP`). Never PRIMARY, the default filegroup or the target filegroup. If that fails (in FULL recovery e.g. until the next log backup), there is only a warning. - **Registry:** the entry in `master.dbo.sqm_PartitionRegistry` is deleted, the maintenance jobs (runbook B) no longer consider the table afterwards. With `-KeepRegistration` it stays. - **`-Online`:** on Enterprise/Developer the rowstore index operations run online. If the table has a columnstore index, this is not possible (warning, offline).

3. **Check the result:** the returned object names moved indexes, dropped schemes, functions and filegroups and all warnings. A repeated call reports `NotPartitioned`.

Empty an archived table and unpartition it (`-TruncateData`)

Typical case after runbook C with cutover: the data lies completely in the archive database, the source is now called `<X>_Original`, and under `<X>` a view reads from the archive. The old table still occupies its space though (e.g. 11 TB in the partition filegroups of the OLTP database). Copying it over would be pointless, it is emptied:

```
# 1. Look: archive reconciliation, planned DDL (TRUNCATE comes first)
Remove-sqmTablePartitioning -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" `
   -Table "OrderHistory_Original" -TruncateData -RemoveEmptyFilegroups -WhatIf

# 2. Execute (asks separately before the TRUNCATE)
Remove-sqmTablePartitioning -SqlInstance "SQL01" -Database "Sales" -Schema "dbo" `
   -Table "OrderHistory_Original" -TruncateData -RemoveEmptyFilegroups
```

- **Archive reconciliation, before any change and from metadata only** (no scan, even with terabytes without load): the archive table must exist, must not be the table itself and must have at least as many rows as the table to be emptied (`sys.partitions`). Without `-ArchiveTable 'Database.Schema.Table'` it is derived from the view `<X>` for `<X>_Original`. `-SkipArchiveCheck` turns the reconciliation off (only if the archive lives elsewhere).
- **Blocked** if TRUNCATE would fail: foreign keys from other tables, indexed views, replication, CDC.

- **Double confirmation:** in addition to the normal prompt the function asks separately before the TRUNCATE. `-Confirm:$false` does **not** turn this prompt off, only `-Force` does (e.g. in an Agent job). Without `-Force` in a non-interactive session the function aborts without changes.
- **Registry:** after the cutover the entry sits under the view name `<X>`. Every entry of the database that points to the dropped partition scheme is removed (`UnregisteredTables`).
- **Reclaiming space:** `TRUNCATE` releases large tables in the background (deferred drop), which can take a while with terabytes. If `-RemoveEmptyFilegroups` fails on a file that is not yet empty, or in FULL recovery on the missing log backup, there is only a warning: wait for the log backup and repeat the call without `-TruncateData`, it cleans up the remaining filegroups.

In the GUI: *Remove partitioning...*, option *Delete ALL data first (TRUNCATE)* (never preselected), optionally the archive table, then a separate warning prompt.

Repeatable: if a run aborts, a new call continues with the indexes that are still partitioned (a leftover `sqmUnpartitionTmp` is also cleaned up). If the table is already unpartitioned but the scheme/function still exist, the repeated call finds them via the registry entry. If indexes were removed for the columnstore rebuild and not recreated, their DDL is in the error message.

Not undone is the extension of a PRIMARY KEY/UNIQUE constraint by the partition column (`-AllowKeyChange` in runbook A): which column was appended back then can no longer be determined, and removing it would change uniqueness. Likewise a clustered index `IX_<Table>_<Column>` created when converting a heap stays as a normal clustered index.

Effort: every index is rewritten once completely, duration and transaction log as with an index rebuild. For heaps with nonclustered indexes these are rebuilt twice additionally because of the temporary clustered index. Plan a maintenance window for large tables.

In the GUI: step 1 (*Select Table*), select an already partitioned table, **Remove partitioning...**, choose the target filegroup.

---

6. BoundaryType/SurrogateDateFormat reference

All partitioning/migration functions support three column types for the partition/period column, via `-BoundaryType` (derived automatically from the SQL column type if omitted):

BoundaryType	Matching SQL column type	Example value	Note
`Date`	`date`/`datetime`/`datetime2`/`smallc`	`2024-01-15`	`tetimeoffset` Standard case, no further configuration needed
`Int`	`int`/`bigint`/`smallint`/`tinyint`	`20240115`	Numeric date surrogate key
`Text`	`char`/`varchar`/`nchar`/`nvarchar`	`'20240115'`	String date surrogate key

For `Int`/`Text`, `-SurrogateDateFormat` controls the precision:

- `yyyyMMdd` (default): day precision, e.g. `20240115`.

- `yyyyMM`: month precision without day, e.g. `202401` (typical when the source column itself is kept only at month level).

Example: a table with an `INT` column in the format `YYYYMMDD` (e.g. `BOOKDATE`) needs `-BoundaryType Int`; if omitted it is derived from the column type.

---

7. GUI wizard: step by step

```
Show-sqmPartitionToolGui -SqlInstance "SQL01"
```

Step	Content
0 - Connection	Instance + authentication (Windows or SQL Server login), choose database
1 - Select Table	Candidate tables (with row count/size/heap-or-clustered); for an already partitioned table *Remove partitioning...* removes its partitioning (runbook E, outside the flow)
2 - Select Column	Choose the partition/date column
3 - Min/Max Preview	Actual value range of the source data (or manual values for an empty table)
4 - Granularity & Filegroups	Month/Quarter/Year, Single/PerPeriod, for non-date columns also Surrogate Date Format; *Data file folder (server)*: directory for new filegroups, selection list with the drives of the server and their free space (empty = default data path)
5 - Boundary Preview	Calculated partition boundaries for review before execution
6 - Archive & Retention	Two mutually exclusive modes (see below)
7 - Summary & Execute	Summary, execute button, live log

Step 6 depends on the chosen table:

- **Not yet partitioned:**

- **"Migrate to archive database now"** runs runbook C. Options: *Create the partitioned archive table only* or *Create the archive table AND transfer the data*; *Include the current month* (preselected, transfers everything, see `-IncludeOpenPeriods`); *Delete each archived month from the source*; *rename the source table and replace it by a view*. For a heap with a matching unique index the wizard offers to make it the clustered PRIMARY KEY of the archive table. A "Key Column(s)" field appears only if a key is needed and cannot be derived.

- **"Set up automatic maintenance"** runs runbook A + B (partition in place, sliding window, retention). With "Migrate now" active this area is hidden.
- **Already partitioned:** copy mode (runbook D) with target database and optional target name, without key selection. Optionally *Create the partitioned target table only* (`-CreateTableOnly`) or *Create the target table AND copy the data*.

"Finish" asks for confirmation if nothing has been executed yet. The window title shows module version and load path.

---

8. Troubleshooting and known limitations

- **Module does not load ("required module sqmDataTransfer")**: sqmDataTransfer >= 0.1.23.0 is missing in the same scope. Install sqmDataTransfer first, then sqmPartitionTool (section 2).
- **The GUI behaves like an old version** (e.g. only one key column allowed): an older installed copy is running. The window title shows version and load path; `Get-Module sqmPartitionTool -ListAvailable` lists all installed versions.
- **"'-KeyColumn' is mandatory"** in runbook C: only still needed for the cutover with `-IncludeOpenPeriods` on a heap without a unique key. Give `-PrimaryKeyFromUniqueIndex` or `-KeyColumn 'Column1','Column2',...` (max. 5).
- **"date is incompatible with int"**: the date column is not a real DATE/DATETIME type but a numeric/string surrogate key; give `-BoundaryType Int`/`Text` (+ `-SurrogateDateFormat` if needed), see section 6.
- **All rows end up in the last partition of the archive table:** fixed since 1.13.0.0 (an unused old partition function is recreated). With older versions remove the empty archive table and the old function/scheme manually before the run.
- **Months are missing in the archive although the log shows "Completed":** fixed since 1.13.0.0 (stale log entries are reset when the archive table is recreated).
- `-Method BatchedSwap` **aborts with an error** ("incoming foreign keys/triggers"): this method does not support tables that other tables reference by foreign key or that have triggers. Remove foreign keys/triggers first, or use `-Method Default`/`NewTableSwap`.
- **Performance with very large tables (runbook C):** without an index with the date column as leading column every monthly chunk reads the whole table. The tool warns but does not create an index (admin decision). Further notes on waits and log in section 5b.
- **Renamed original table after cutover** (runbook C) does not grow any more, but without `-IncludeOpenPeriods` still contains the last open month; check before deleting it for good.
- `-Method BatchedSwap` **and runbook C do not run online:** both briefly lock the affected rows/partitions during the respective batches, the atomic cutover the whole source table for the duration of the final reconciliation. For production systems plan a maintenance window or times of low load.
- **After a failover the jobs process no tables** (registry empty or outdated): `master` is not replicated by AlwaysOn. Run `Sync-sqmPartitionRegistry` from the former primary or the listener, and create jobs with `-AllReplicas` in future (section 4a).
- **Job on the secondary "succeeds" but does nothing:** intended, databases that are not locally ONLINE/READ_WRITE are skipped. The line `uebersprungen (nicht schreibbar/Secondary)` in the job output proves it.
- **"Verzeichnis ... existiert nicht auf allen Replikaten der Verfuegbarkeitsgruppe"** (directory does not exist on all replicas of the availability group) when creating filegroups: create the directory on the named replicas (same path, rights for the SQL service account). Otherwise `ADD FILE` would suspend the database there.

• `New-sqmPartitionRetentionJob -AllReplicas`: **script cannot be copied:** access to `\\<Host>\C$` is missing or `sqmPartitionTool` is not installed on the host.

• **Remove-sqmTablePartitioning: "Platzpruefung ... nur X MB verfuegbar"** (space check, only X MB available): the target filegroup is too small. Enlarge the file/allow autogrowth, use another `-TargetFilegroup`, or for archived data `-TruncateData`.

• **Remove-sqmTablePartitioning -TruncateData: "Archiv-Tabelle nicht ableitbar"** (archive table cannot be derived): the table is not called `<X>_Original` or the view `<X>` is missing. Give `-ArchiveTable 'Db.Schema.Table'`.

• **Remove-sqmTablePartitioning: "Nicht unterstuetzte Indextypen"** (unsupported index types): remove XML or spatial indexes first and recreate them afterwards.

• **Remove-sqmTablePartitioning: partition scheme stays**: another object still uses it (the warning names it). It is cleaned up when the partitioning of the last object is removed.

• **Remove-sqmTablePartitioning: filegroup not removed**: in FULL recovery an emptied file often can only be removed after the next log backup. Then finish with `ALTER DATABASE ... REMOVE FILE`/`REMOVE FILEGROUP`.

Note: error and log messages of the module are in German, the messages quoted above are shown in their original wording.

---

9. Security notes

• All destructive operations (retention deletion without `-ArchiveEnabled`, `-PurgeSourceAfterArchive`) delete data from the source irrevocably: before productive use try them on a test table with representative data.

• The renamed original table (runbook C, `_Original` suffix) is **never** deleted automatically: that is intentional, not a bug. The tool generally does not delete tables automatically but relies on the admin for the last, final step.

• For productive migrations a full backup of the source database beforehand and a test run with `-WhatIf` are recommended (all core functions support `SupportsShouldProcess`).

• SQL Server authentication (`-SqlCredential`) should only be used with mixed-mode logins with the minimum necessary rights, not with `sa`.