sqmDataTransfer — Admin-Handbuch

Tabellendaten zwischen SQL-Server-Instanzen verschieben, mit Metadaten-Scripting, sicherer FK-/Index-Behandlung, Zeilenzahl-Abgleich und vollstaendigem HTML-Report — per PowerShell-Aufruf oder ueber die WinForms-GUI. Aufgebaut auf dbatools.

Zielgruppe dieses Handbuchs: SQL-Server-DBAs, die das Modul operativ einsetzen (nicht die Entwicklung des Moduls selbst). Fuer die Versionshistorie siehe CHANGELOG.md, fuer eine Kurzuebersicht README.md.

Inhalt

---

1. Ueberblick

Das Modul deckt vier unterschiedliche, unabhaengig voneinander nutzbare Szenarien ab. Die Entscheidung, welches passt, haengt davon ab, **wie oft** transferiert wird und **ob** die Tabelle einen brauchbaren Schluessel hat:

Szenario	Funktion	Wann sinnvoll
A — Normaler Transfer	`Invoke-sqmTableTransfer`	Einmaliger oder wiederholbarer All-or-nothing-Copy einer oder mehrerer Tabellen, inklusive optionalem Anlegen der Zieltabelle
B — Chunk-Transfer	`Invoke-sqmChunkedTableTransfer`	Eine einzelne, sehr grosse Tabelle (zig bis hunderte Millionen Zeilen), insbesondere ohne Primary Key, resumable nach einem Abbruch

C — Inkrementeller Abgleich	`Sync-sqmTableData`	Ziel existiert bereits vollstaendig befuellt, nur die seit dem letzten Lauf tatsaechlich geaenderten Zeilen sollen nachgezogen werden
Reine Kontrolle	`Compare-sqmTableRowCount` / `Compare-sqmDatabaseRowCount`	Nur pruefen, ob Quelle und Ziel (noch) uebereinstimmen, ohne selbst zu kopieren

Faustregel: Erster Transfer einer Tabelle oder eines Tabellensets -> A. Eine einzelne Tabelle sprengt A (Stunden statt Minuten, kein Primary Key fuer eine saubere Resume-Logik) -> B. Die Zieltabelle ist bereits vollstaendig und es sollen nur noch laufende Aenderungen nachgezogen werden (z.B. waehrend eine Testumgebung parallel gegen das Ziel arbeitet) -> C.

---

2. Voraussetzungen und Installation

- PowerShell 5.1 oder hoeher (die GUI benoetigt WinForms/Desktop-CLR — unter PowerShell 7 auf Windows weiterhin verfuegbar, nicht aber auf PowerShell 7 unter Linux/macOS).
- Modul `dbatools` muss installiert sein (`RequiredModules` im Manifest, gedeckelt auf Versionen unterhalb 3.0 — dbatools 3.0 ist als binaeres Modul mit C#-Cmdlets angekuendigt und bislang nicht gegen dieses Modul getestet).
- Ein SQL-Server-Login mit ausreichenden Rechten auf Quelle **und** Ziel: `SELECT` auf der Quelltabelle, `INSERT`/`ALTER` auf der Zieltabelle, sowie bei `-ScriptMetadata` Rechte zum Anlegen von Tabellen/Typen/Sequenzen auf dem Ziel.
- Windows- oder SQL-Server-Authentifizierung, wahlweise pro Instanz unterschiedlich (`-SourceCredential`/`-DestinationCredential`) oder gemeinsam (`-SqlCredential`).

```
Import-Module "C:\CCM\SQL-Tools\sqmDataTransfer\sqmDataTransfer.psd1"
```

Installation ueber `Install.cmd` im Projektverzeichnis (erkennt automatisch, ob mit Administratorrechten ausgefuehrt wird: `AllUsers`, sonst `CurrentUser`).

Konfiguration wird getrennt von sqmSQLTool persistiert (`%APPDATA%\SQLDataTransfer\config.json`), sodass beide Module nebeneinander importiert werden koennen, ohne sich gegenseitig zu beeinflussen — `LogPath`/`OutputPath` zeigen standardmaessig trotzdem auf denselben Ort, den sqmSQLTool verwendet (`C:\System\WinSrvLog\MSSQL`).

---

3. Ablaufplan A: Normaler Tabellentransfer

Ziel: eine oder mehrere Tabellen von einer Quell- auf eine Zielinstanz kopieren, optional inklusive Anlegen der Zieltabelle.

Schritt 1 — Transfer ausfuehren:

```
Invoke-sqmTableTransfer -Source SQL01 -SourceDatabase Sales -Destination SQL02 `
   -DestinationDatabase Sales -Table 'dbo.Orders', 'dbo.Customers' `
   -ScriptMetadata -Truncate -Confirm:$false
```

Fuenf Schritte laufen automatisch und einzeln geloggt ab:

- **Metadaten scripten** (nur mit `-ScriptMetadata`) — scriptet Spalten, PK, Indizes, Fremdschluessel, Defaults, Checks von der Quelle und legt die Tabelle auf dem Ziel an, falls sie dort noch nicht existiert. Bestehende Zieltabellen werden nie geloescht oder neu angelegt. `Export-sqmTableSchema` folgt dabei der echten Abhaengigkeitskette (SMO `WithDependencies`): benutzerdefinierte Typen, Sequenzen und per FK referenzierte Tabellen werden automatisch mitgescriptet. Partitionierte Tabellen werden vollstaendig transferiert, aber die physische Partitionierung selbst (Partition Function/Scheme, Filegroups je Partition) wird entfernt, da nicht bekannt ist, ob auf dem Ziel ein passendes Schema existiert — landet stattdessen als normale Tabelle auf `PRIMARY`, mit Warnung im Report. CLR-Typen werden gescriptet, aber als Warnung markiert — die Assembly selbst muss manuell auf dem Ziel deployed werden. Die Zielversion wird automatisch erkannt und als `TargetServerVersion` an SMO uebergeben, damit das Scripting von einer neueren Quelle (z.B. SQL 2022) auf ein aelteres Ziel (z.B. SQL 2019) syntaktisch kompatibel bleibt.
- **Deaktivieren** — Fremdschluessel, nicht-geclusterte Indizes und DML-Trigger auf der Zieltabelle werden deaktiviert, damit der Bulk-Load nicht gegen Constraint-Pruefung, Index-Pflege oder einen pro Zeile feuernden Trigger ankaempft. Geclusterte Indizes werden nie deaktiviert (macht die Tabelle unzugaenglich).
- **Daten kopieren** — Bulk-Copy per `Copy-sqmTableData` (siehe Abschnitt 7 fuer Details zur namens-basierten Spaltenzuordnung und zur automatischen Columnstore-Batchgroessen-Deckelung).
- **Zeilenzahlen vergleichen** — `SELECT COUNT_BIG(*)` auf beiden Seiten, Ergebnis landet im Report.
- **Wieder aktivieren** — laeuft in einem `finally`-Block, also auch dann, wenn ein vorheriger Schritt fehlschlaegt. Fremdschluessel und Indizes werden neu aufgebaut (`REBUILD`), nicht nur reaktiviert.

Schritt 2 — Ergebnis pruefen: jeder Aufruf gibt strukturierte Ergebnisobjekte zurueck (`Table`, `Step`, `Status`, `Message`, `Timestamp`) und schreibt zusaetzlich einen eigenstaendigen HTML-Report (`Export-sqmTransferReport`), der am Ende automatisch im Browser geoeffnet wird (ausser `-NoOpen`).

Schritt 3 — Unterbrochenen Lauf fortsetzen: ein Lauf ueber hunderte Tabellen kann mittendrin abbrechen (Absturz, Neustart ueber Nacht). `-SkipCompleted` prueft vorab per `Compare-sqmTableRowCount` jede angeforderte Tabelle und ueberspringt jede, bei der Quelle und Ziel bereits uebereinstimmen — ein erneuter Aufruf mit derselben vollstaendigen Tabellenliste transferiert dann nur noch das tatsaechlich Fehlende, ohne separate Buchfuehrung darueber, was schon fertig war.

```
Invoke-sqmTableTransfer -Source SQL01 -SourceDatabase Sales -Destination SQL02 `
   -DestinationDatabase Sales -Table $allTables -ScriptMetadata -SkipCompleted -Confirm:$false
```

Ab einer konfigurierbaren Quell-Zeilenzahl (Standard 10 Millionen, `Set-sqmTransferConfig -LargeTableRowThreshold`) warnt `Invoke-sqmTableTransfer` statt stillschweigend einen All-or-nothing-Copy durchzufuehren, inklusive fertigem `Invoke-sqmChunkedTableTransfer`-Befehl (siehe Abschnitt 4). Die Warnung greift dabei nur, wenn das Ziel bereits einen nennenswerten Anteil der Quellzeilen enthaelt (Standard 30%, `ChunkAdviceMinExistingPercent`) — auf einem leeren Ziel ist der normale Copy schlicht schneller, da kein Chunking-Overhead anfaellt.

---

4. Ablaufplan B: Grosse Tabellen ohne Primary Key

Ziel: eine einzelne, sehr grosse Tabelle wird spaltenweise in unabhaengig wiederholbaren Portionen ("Chunks") transferiert — ohne Primary Key auf der Zieltabelle resumable, weil die Vollstaendigkeit ueber Zeilenzahlen pro Chunk-Wert festgestellt wird, nicht ueber einen Zeilenschluessel.

Schritt 1 — Chunk-Spalte bestimmen (optional manuell, sonst automatisch):

```
Get-sqmChunkColumnCandidate -SqlInstance SQL01 -Database Sales -Table dbo.Ergebnis_agg
```

Bewertet Datums- und Perioden-Spalten nach Namenskonvention (`Stichtag`, `ReportingDate`, `Dat_`-/`dtm`-Praefixe zuerst) und schaetzt die Chunk-Anzahl je Kandidat aus dem Statistik-Histogramm der Spalte (`sys.dm_db_stats_histogram`) — eine reine Metadatenlektuere, die auf einer 344-Millionen-Zeilen-Tabelle dasselbe kostet wie auf einer leeren. `-Exact` ersetzt die Schaetzung durch ein echtes `COUNT_BIG(DISTINCT ...)` (Vollscan), nur sinnvoll, wenn die Schaetzung nahe an einer Entscheidungsgrenze liegt.

Schritt 2 — Transfer starten, mit oder ohne explizite Chunk-Spalte:

```
Invoke-sqmChunkedTableTransfer -Source SQL01 -SourceDatabase Sales -Destination SQL02 `
   -DestinationDatabase Sales -Table dbo.Ergebnis_agg -ChunkColumn Dat_ReportingDate `
   -Confirm:$false

# oder: Chunk-Spalte automatisch ermitteln lassen (ohne -ChunkColumn)
Invoke-sqmChunkedTableTransfer -Source SQL01 -SourceDatabase Sales -Destination SQL02 `
   -DestinationDatabase Sales -Table dbo.Ergebnis_agg -Confirm:$false
```

Ohne `-ChunkColumn` waehlt die Funktion selbst per `Get-sqmChunkColumnCandidate` und protokolliert die Wahl, die geschaetzte Chunk-Anzahl und die Begruendung; findet sich keine geeignete Spalte, bricht der Lauf ab und nennt, was verworfen wurde und warum. Ohne explizites `-MaxChunkValues` gilt die Konfigurationsgrenze `MaxChunkValueCeiling` (Standard 2000) — die tatsaechlich gefundene Anzahl wird bis dahin akzeptiert, statt an einem festen Default zu scheitern.

Schritt 3 — Ablauf im Detail:

- Quell- und Ziel-Zeilenzahl je Chunk-Wert werden je einmal vorab per `GROUP BY` snapshotted, nicht pro Chunk neu abgefragt. Ein Chunk, dessen Zaehler bereits uebereinstimmen, wird uebersprungen — ein erneuter Lauf nach einem Abbruch transferiert nur noch das tatsaechlich Fehlende.
- Ein mitten in einem Chunk abgebrochener Lauf hinterlaesst ggf. teilweise kopierte Zeilen; vor einem Retry werden diese fuer genau diesen einen Chunk geloescht, sodass ein erneuter Versuch nie Zeilen dupliziert.
- Fremdschluessel, Indizes und Trigger werden einmal fuer den gesamten Chunk-Lauf deaktiviert und am Ende einmal wieder aktiviert/neu aufgebaut — nicht pro Chunk, da ein Index-Rebuild unabhaengig von der Chunk-Groesse immer eine Ganztabellen-Operation ist.
- **Columnstore-Ziele:** hat die Zieltabelle einen Columnstore-Index, wird die tatsaechlich an `SqlBulkCopy` uebergebene Batchgroesse automatisch auf `ColumnstoreBatchSizeCeiling` (Standard 100.000) gedeckelt. SQL Server komprimiert einen Bulk-Insert-Batch ab 102.400 Zeilen sofort in ein komprimiertes Rowgroup statt ueber den Delta-Store zu gehen — teuer, bei `COLUMNSTORE_ARCHIVE` zusaetzlich synchron im Ladepfad, und es entstehen viele kleine statt weniger, voll ausgewachsener Rowgroups. Die Erkennung laeuft automatisch (eine `sys.indexes`-Abfrage, kein Tabellenscan), keine Konfiguration noetig.
- **`-DestinationTable`** transferiert in eine anders benannte Zieltabelle, z.B. eine frisch neu partitionierte `_New`-Kopie vor einem Rename-Swap-Cutover. Nicht kombinierbar mit `-ScriptMetadata` (das legt die Zieltabelle immer unter dem Quellnamen an).
- **`-BulkCopyTimeOut`** (Standard 300 Sekunden, `0` = kein Limit) gilt je Batch, nicht je Tabelle. Ein Checkpoint auf dem Ziel, Autogrowth von Daten-/Logdatei oder IO-Konkurrenz kann einen einzelnen Batch leicht ueber 300 Sekunden halten; bei einem Chunk-Transfer ist das teuer, da ein Chunk ab seiner ersten Zeile neu gestartet

wird. Fuer eine unbeaufsichtigte Migration ist `0` meist die bessere Wahl.

Schritt 4 — Abschluss: ein einziger konsolidierter `GROUP BY`-Scan auf dem Ziel vergleicht alle verarbeiteten Chunks gegen die Ausgangs-Snapshots, statt eines Scans pro Chunk. Ein HTML-Report wird geschrieben (ausser `-NoReport`), inklusive `Compare-sqmTableRowCount -Fast` (liest `sys.dm_db_partition_stats`, kein `COUNT(*)`).

---

5. Ablaufplan C: Inkrementeller Abgleich

Ziel: die Zieltabelle existiert bereits vollstaendig (aus einem vorherigen Ablaufplan A/B), und nur die seit dem letzten Lauf tatsaechlich geaenderten Zeilen sollen nachgezogen werden, ohne die ganze Tabelle neu zu kopieren.

```
Sync-sqmTableData -Source SQL01 -SourceDatabase Sales -Destination SQL02 `
   -DestinationDatabase Sales -Table dbo.Orders, dbo.OrderDetails -Confirm:$false
```

Ablauf pro Tabelle:

1. Liest Primary Key und Spalten-Metadaten von der **Zieltabelle** (muss dort bereits mit derselben Struktur wie auf der Quelle existieren).
2. Berechnet einen SHA2-256-Hash ueber alle Nicht-PK-, Nicht-berechneten, Nicht-Rowversion-Spalten, fuer jede Zeile, auf beiden Seiten — das ist ein Vollscan auf beiden Seiten, der einzige Kostenpunkt, der sich ohne eine vertrauenswuerdige Aenderungsdatums-Spalte nicht vermeiden laesst. `-Table` daher gezielt auf tatsaechlich betroffene Tabellen eingrenzen, nicht auf das gesamte Tabellenset.
3. Vergleicht die beiden (PK -> Hash)-Mengen im Speicher, um eingefuegte, geaenderte und (mit `-IncludeDelete`, Standard `$true`) geloeschte Primary Keys zu finden. Stimmt nichts ueberein, wird die Tabelle komplett uebersprungen.
4. Eingefuegte/geaenderte Zeilen: die vollstaendigen Zeilen fuer genau diese Primary Keys werden von der Quelle gelesen, in eine temporaere Staging-Tabelle auf dem Ziel geladen und per einem einzigen `MERGE` (Upsert) in die echte Zieltabelle uebernommen. IDENTITY-Spalten werden dabei ueber `SET IDENTITY_INSERT` behandelt.
5. Geloeschte Zeilen (`-IncludeDelete`): werden direkt anhand ihres Primary Keys aus der Zieltabelle entfernt.
6. Die Staging-Tabelle wird abschliessend geloescht.

Zeilenwertlisten (Quell-`SELECT`, `DELETE`) werden unabhaengig von `-BatchSize` auf 1000 Schluessel pro Anweisung aufgeteilt (SQL Servers Obergrenze fuer einen `VALUES`-Row-Constructor) — `-BatchSize` steuert ausschliesslich den Bulk-Load in die Staging-Tabelle.

---

6. Konfiguration

`Get-sqmTransferConfig` / `Set-sqmTransferConfig`, persistiert getrennt von sqmSQLTool in `%APPDATA%\SQLDataTransfer\config.json`:

Schluessel	Standard	Bedeutung
`LogPath`	`C:\System\WinSrvLog\MSSQL`	Verzeichnis fuer Logdateien

`OutputPath`	`C:\System\WinSrvLog\MSSQL`	Standard-Zielverzeichnis fuer HTML-Reports und gescriptete Schema-Dateien
`TrustServerCertificate`	`$true`	Ob dbatools-Verbindungen selbstsignierten Zertifikaten vertrauen
`DefaultBatchSize`	`500000`	Standard-Batchgroesse fuer `Copy-sqmTableData`, sofern `-BatchSize` nicht angegeben wird
`LargeTableRowThreshold`	`10000000`	Ab dieser Quell-Zeilenzahl warnt `Invoke-sqmTableTransfer` mit einem fertigen Chunk-Transfer-Befehlsvorschlag
`ChunkAdviceMinExistingPercent`	`30`	Die Chunking-Warnung greift nur, wenn das Ziel bereits mindestens diesen Anteil (%) der Quellzeilen enthaelt
`MaxChunkValueCeiling`	`2000`	Obergrenze der automatisch ermittelten Chunk-Anzahl, wenn `-MaxChunkValues` nicht explizit gesetzt ist
`ColumnstoreBatchSizeCeiling`	`100000`	Obergrenze der tatsaechlich an `SqlBulkCopy` uebergebenen Batchgroesse auf einer Columnstore-Zieltabelle, bewusst unterhalb der 102.400er-Kompressionsschwelle von SQL Server

```
Set-sqmTransferConfig -DefaultBatchSize 250000 -LargeTableRowThreshold 5000000
```

---

7. Funktionsreferenz

Funktion	Zweck
`Invoke-sqmTableTransfer`	Haupteinstiegspunkt, orchestriert die fuenf Schritte aus Abschnitt 3 fuer eine oder mehrere Tabellen

`Invoke-sqmChunkedTableTransfer`	Fuer sehr grosse Tabellen ohne Primary Key, splittet nach Spalte und transferiert/setzt chunkweise fort
`Get-sqmChunkColumnCandidate`	Bewertet Kandidaten-Chunk-Spalten nach Namenskonvention und geschaetzter Chunk-Anzahl, rein aus Statistik, kein Tabellenscan
`Sync-sqmTableData`	Gleicht den tatsaechlichen Insert/Update/Delete-Delta einer Tabelle per Staging-Tabelle ab, statt eines vollstaendigen Neu-Copys
`Export-sqmTableSchema`	Scriptet Tabellen-DDL von einer Quellinstanz (SMO Scripter)
`New-sqmTableFromScript`	Fuehrt gescriptete DDL-Batches gegen eine Zielinstanz aus
`Copy-sqmTableSchema`	Komfort-Wrapper: Export + Anlegen in einem Aufruf
`Disable-sqmTableConstraints`	Deaktiviert FKs / nicht-geclusterte Indizes / Trigger auf einer Tabelle
`Enable-sqmTableConstraints`	Reaktiviert (rebuilded) zuvor deaktivierte FKs/Indizes/Trigger, erkennt den Zustand selbst, nichts muss uebergeben werden
`Copy-sqmTableData`	Bulk-kopiert Tabellendaten (namensbasierte Spaltenzuordnung, unabhaengig von der installierten dbatools-Version, Batchgroesse auf Columnstore-Zielen automatisch gedeckelt)
`Compare-sqmTableRowCount`	Vergleicht Zeilenzahlen Quelle vs. Ziel fuer eine oder mehrere benannte Tabellen
`Compare-sqmDatabaseRowCount`	Vergleicht Zeilenzahlen fuer alle Tabellen zwischen zwei Datenbanken auf einmal, keine Tabellenliste noetig
`Export-sqmTransferReport`	Erstellt den HTML-Zusammenfassungs-/Zeilenzahl-Report je Lauf
`Export-sqmDatabaseComparisonReport`	Erstellt den konsolidierten datenbankweiten Vergleichsreport
`Show-sqmTableTransferGui`	WinForms-GUI fuer den gesamten Workflow
`Get-sqmTransferConfig` / `Set-sqmTransferConfig`	Modulkonfiguration (siehe Abschnitt 6)

Geclusterte Indizes werden nie deaktiviert (macht die Tabelle unzugaenglich), nur nicht-geclusterte Indizes werden angefasst. Fremdschluessel und Trigger werden einzeln namentlich deaktiviert/aktiviert, CHECK-/DEFAULT-Constraints bleiben unangetastet.

---

8. GUI: Schritt-fuer-Schritt

```
Show-sqmTableTransferGui
```

1. **Verbinden** — Quell- und Zielinstanz eingeben, "Verbinden" testet die Konnektivitaet und befuellt das Datenbank-Dropdown je Seite.

2. **Tabellen laden** — zeigt fuer jede Tabelle, ob sie auf dem Ziel bereits existiert ("Transfer") oder erst angelegt werden muss ("Anlegen"). Auswahl einzeln oder ueber Alle/Keine; ein angehaktes Kaestchen zeigt zusaetzlich die Quell-Zeilenzahl (Metadatenlektuere, kein Scan).

Transfermodus waehlen:

- **Automatisch** (Vorgabe) — entscheidet je Tabelle nach derselben Regel wie die Chunking-Warnung in Abschnitt 3 (`LargeTableRowThreshold` **und** Ziel bereits zu mindestens `ChunkAdviceMinExistingPercent` befuellt), und nur dann, wenn ueberhaupt eine brauchbare Chunk-Spalte existiert. Alle uebrigen Tabellen laufen normal.
- **Normal** — immer der klassische All-or-nothing-Copy (Ablaufplan A), inklusive Hinweisdialog mit fertigem Chunk-Befehl bei einer zu grossen Tabelle.
- **Chunk-Transfer** — alle ausgewaehlten Tabellen laufen chunkweise (Ablaufplan B), eine Runde je Tabelle. Ein Chunk-Spalten-Feld mit Schaltflaeche "Erkennen" steht zur Verfuegung; leer bedeutet automatische Erkennung je Tabelle, eine feste Spalte wird nur uebernommen, wenn genau eine Tabelle ausgewaehlt ist.

4. **Optionen setzen** — Metadaten scripten, FKs/Indizes deaktivieren/aktivieren, Truncate, FKs beim Wiederaktivieren revalidieren, Batchgroesse, Simulieren (`-WhatIf`), bereits vollstaendige Tabellen ueberspringen (`-SkipCompleted`, zum Fortsetzen eines unterbrochenen Laufs).

5. **Ausfuehren** — die Oberflaeche blockiert waehrend des Laufs (synchron, wie die anderen lang laufenden Operationen des Moduls). Das Schritt-fuer-Schritt-Log und die strukturierte Ergebnistabelle je Tabelle/Schritt werden nach Abschluss angezeigt.

6. **Gesamtbericht** — "Overall report" ruft `Compare-sqmDatabaseRowCount` fuer die gesamte Datenbank auf; die genaue Verifikation (`-VerifyMismatches`, echter `COUNT_BIG(*)`-Scan fuer jede abweichende Tabelle) ist ueber eine Checkbox opt-in, Standard ist die schnelle metadatenbasierte Pruefung.

---

9. Troubleshooting und bekannte Einschraenkungen

- **"Fuer konnte keine geeignete Chunk-Spalte ermittelt werden"**: die Tabelle hat weder eine Datums- noch eine Perioden-Spalte, die `Get-sqmChunkColumnCandidate` erkennt. `-ChunkColumn` explizit angeben, oder `-Exact` gegen einen Kandidaten pruefen, dessen Schaetzung knapp unter der Eignungsschwelle liegt.
- **" hat mehr Werte als MaxChunkValueCeiling"**: die gewaehlte Spalte ist zu feingranular (naehert sich einem Zeitstempel an) — eine groebere Spalte waehlen, oder `Set-sqmTransferConfig -MaxChunkValueCeiling` dauerhaft anheben.

- **"Execution Timeout Expired" mitten in einem Chunk-Transfer**: ein einzelner Batch ist ueber `-BulkCopyTimeOut` (Standard 300s) hinaus haengen geblieben (Checkpoint, Autogrowth, IO- Konkurrenz auf dem Ziel). `-BulkCopyTimeOut 0` fuer eine unbeaufsichtigte Migration, ggf. kombiniert mit einer kleineren `-BatchSize`, um zu begrenzen, was ein einzelner Haenger kosten kann.
- **Deutlich langsamerer Chunk-Transfer nach Erhoehen der Batchgroesse auf einer Columnstore-Zieltabelle**: `Copy-sqmTableData` deckelt das seit Version 0.1.20.0 automatisch (siehe Abschnitt 4) — bei einer aelteren Modulversion oder einer Zieltabelle mit Nicht-Standard-Columnstore-Konfiguration `Set-sqmTransferConfig -ColumnstoreBatchSizeCeiling` pruefen bzw. explizit setzen.
- **`-DestinationTable` zusammen mit `-ScriptMetadata`**: nicht unterstuetzt, wirft sofort einen Fehler. `Copy-sqmTableSchema` legt die Zieltabelle immer unter dem Quellnamen an — die Zieltabelle unter dem gewuenschten Namen vorher manuell anlegen und ohne `-ScriptMetadata` aufrufen.
- **`Sync-sqmTableData` laeuft ungewoehnlich lange**: der Hash-Vergleich ist ein Vollscan auf beiden Seiten. Bei einem grossen Tabellenset `-Table` auf die tatsaechlich betroffenen Tabellen eingrenzen, statt das gesamte Set jedes Mal zu pruefen.
- **Partitionierte Quelltabellen (`-ScriptMetadata`)**: die physische Partitionierung wird beim Scripting bewusst entfernt (siehe Abschnitt 3) — die Zieltabelle landet als normale Tabelle auf `PRIMARY`. Das ist erwartetes Verhalten, kein Fehler, im Report als Warnung sichtbar.
- **CLR-Spaltentypen**: werden gescriptet, aber die Assembly wird nicht automatisch auf das Ziel deployed — vor dem Anlegen der Tabelle manuell erledigen, sonst schlaegt `CREATE TABLE` auf dem Ziel fehl.

---

10. Sicherheitshinweise

- `-Truncate` leert die Zieltabelle unwiderruflich, bevor kopiert wird — vor dem produktiven Einsatz an einer Testtabelle mit repraesentativen Daten ausprobieren.
- `Sync-sqmTableData -IncludeDelete` (Standard `$true`) loescht Zeilen aus dem Ziel, deren Primary Key auf der Quelle nicht mehr existiert — bei `$false` bleibt das Ziel ein reines Superset, nie ein exakter Spiegel.
- Alle Kernfunktionen unterstuetzen `SupportsShouldProcess` (`-WhatIf`/`-Confirm`) — vor einem produktiven Lauf mit `-WhatIf` pruefen, was tatsaechlich passieren wuerde.
- SQL-Server-Authentifizierung (`-SqlCredential`/`-SourceCredential`/`-DestinationCredential`) sollte ueber Mixed-Mode-Logins mit minimal notwendigen Rechten erfolgen, nicht ueber `sa`.
- `TrustServerCertificate` (Standard `$true`) vertraut selbstsignierten Zertifikaten auf allen dbatools-Verbindungen dieses Moduls — fuer Verbindungen ueber ein ungesichertes Netzwerk Zertifikate mit echter Kette in Erwaegung ziehen und den Wert auf `$false` setzen.