Partitionierung SQL Server Tabellen

Kategorie
SQL Server

Aaron
Priesterroth

In unserem heutigen Artikel möchten wir uns mit der Partitionierung von Dateigruppen beschäftigen und demonstrieren wie automatische Partitionierung von Tabellen in SQL Server auf unterschiedliche Arten realisieren kann.

Eine Automatisierung kann vor allem dann einen entscheidenden Vorteil bieten, wenn der Bereich der Partitionsfunktion nicht mehr ausreicht, um neu eingefügte Datensätze zu verarbeiten. Eine Partitionsfunktion wird oft für besonders große Tabellen verwendet, wobei die Datensätze anhand einer bestimmten Spalte (z.B. nach Monaten, Jahren, etc.) unterteilt werden.

In den meisten Fällen wird für den Aufbau einer Datenbank ein maximaler Bereich, in dem sich die Daten befinden können, definiert. Dafür wird eine Dateigruppe (engl. **filegroup**) dem Partitionierungsschema hinzugefügt und ein Bereich in der Partitionierungsfunktion definiert.
Oft kommt es nach einer gewissen Betriebszeit dazu, dass sich die ursprüngliche Prognose über den Bereich der Daten verändert oder deutlich wird, dass die Schätzung nicht akkurat genug war. Die Partitionierung muss also nachträglich erweitert werden.
Eine Aufgabe, die vermutlich so gut wie jeder DBA schon einmal übernehmen musste: Die Daten müssen analysiert, der Daten-Bereich gespalten und die Dateigruppen bzw. Partitionierungsfunktion manuell erweitert werden. In den folgenden Abschnitten möchten wir uns anschauen, wie diese manuelle Aufgabe mithilfe des SQL Server Schedulers vollständig automatisiert werden kann.

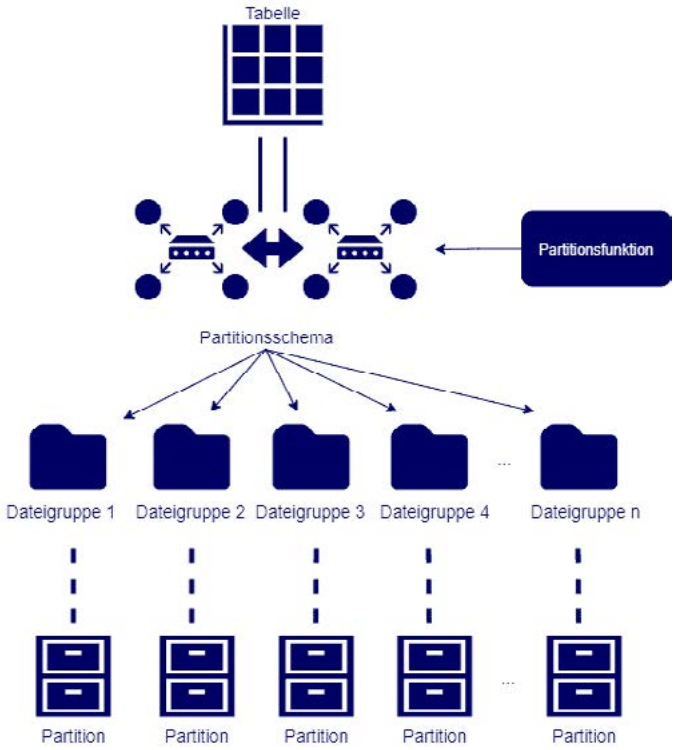
Tabellen partitionieren in SQL Server

Bevor wir uns an die Automatisierung der Partitionierung von Tabellen machen, sollten wir offene Fragen bezüglich der allgemeinen Partitionierung in SQL Server klären. Also: Was ist eine Partitionierung und welchen Mehrwert kann sie mir bieten? Antwort: Die Partitionierung von Tabellen beschreibt die logische Aufteilung der Tabelleninformationen mit der physikalischen Verteilung innerhalb der Dateigruppen. Eine Tabelle kann partitioniert werden, indem ein sogenanntes Partitionsschema auf das Schema der Tabelle angewendet wird. Anhand dieser Schemata kann die Partitionsfunktion die in der Tabelle enthaltenen Informationen in die jeweiligen Bereiche unterteilen. Nachdem also eine Partitionierung auf eine Tabelle angewendet wird, werden die in der Tabelle befindlichen Informationen basierend auf ihrem Bereich auf sekundäre Dateien verteilt.

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

In einer produktiven Umgebung nimmt der Informationshaushalt für gewöhnlich von Tag zu Tag zu. Das hat zur Folge, dass auch die Größe der Datenbanken kontinuierlich wächst. Die korrekte Pflege der Datenbank mit Indexen kann zwar Abhilfe schaffen, mit wachsenden Datenmengen werden jedoch Tabellenleistung und Clientabfragen zwangsläufig kostspieliger und damit langsamer. Was können wir dagegen tun? **Partitionieren!**

Ganz allgemein kann man festhalten, dass Partitionierung als Optimierungswerkzeug für die Abfrage-Performanz auf besonders großen Tabellen verwendet werden kann. Der entscheidende Vorteil entsteht genau dann, wenn eine Abfrage auf die Spalte abzielt, anhand derer die Tabelle partitioniert wurde. Wir sparen uns also die Suche nach Informationen in Dateigruppen die außerhalb unseres (Interesse-) Bereichs liegen.



Partitionierung in SQL Server unterteilt unsere Informationen also in kleinere Speichergruppen. Es geht dabei um Tabellendaten und Indexe. Partitionsfunktionen werden basierend auf den Spalten einer Tabelle ausgedrückt. Sie werden mit einem Namen und Attributen bezüglich des physischen Speicherortes definiert. Um ein besseres Gefühl für die Verwendung von Partitionen zu bekommen, werden wir im folgenden Abschnitt die manuelle Partitionierung einer Tabelle betrachten, bevor wir mit dem automatisieren dieser Aufgabe weitermachen.

Tabellen partitionieren in SQL Server – Schritt für Schritt

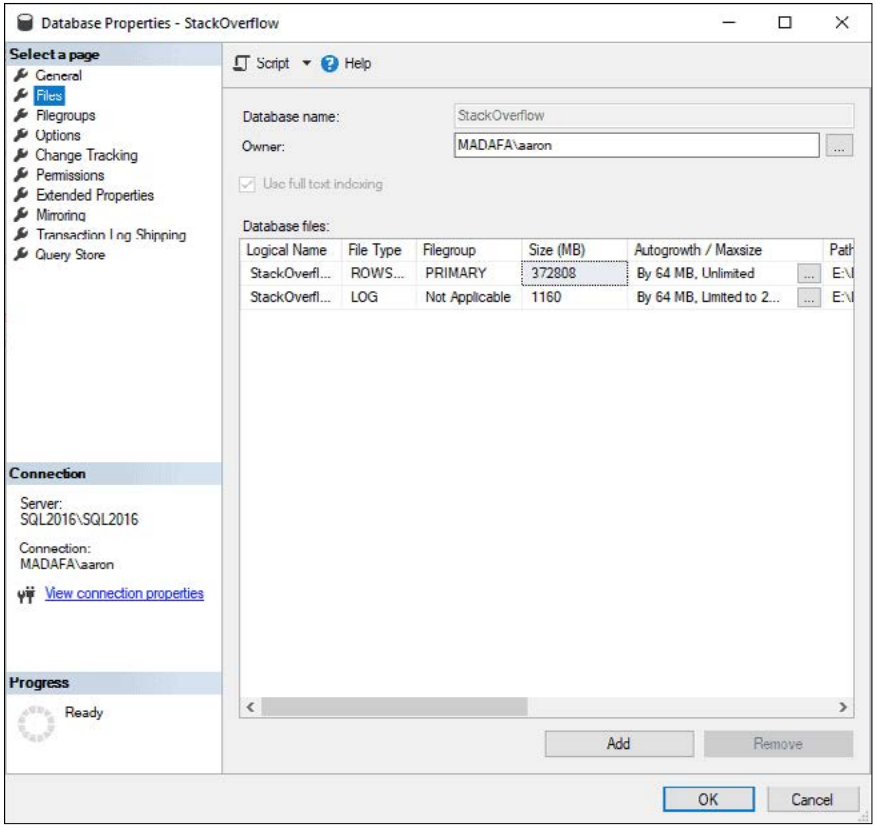
Um eine Tabelle zu partitionieren, müssen wir insgesamt vier Schritte durchführen:

01. Wir müssen eine Dateigruppe erstellen
02. Wir müssen eine neue Datei in die Dateigruppe einfügen
03. Wir müssen eine Partitionsfunktion definieren
04. Wir müssen ein Partitionsschema erstellen und anwenden

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Dateigruppen erstellen
Um ein realistisches Szenario modellieren zu können, werden wir in diesem Beispiel die Datenbank **Stackoverflow** verwenden. Eine Wiederherstellungsdatei dieser Datenbank kann hier gefunden werden.

Nachdem wir die Wiederherstellungsdatei eingespielt haben, überprüfen wir zu allererst die vorhandenen Dateigruppen der Datenbank:



Wie wir sehen können, existiert bis jetzt nur die primäre Dateigruppe.

In diesem Beispiel werden wir die Tabelle **dbo.PostHistory** nach dem Datum (genauer gesagt nach dem Jahr in dem der Post erstellt wurde) partitionieren. Um ein besseres Gefühl für den Bereich, mit dem wir arbeiten wollen zu bekommen, betrachten wir zunächst die obere und untere Schranke unserer Daten:

```
SELECT MIN([CreationDate]) AS mindate,
                MAX([CreationDate]) AS maxdate
FROM [StackOverflow].[dbo].[PostHistory]
```

Führen wir die Abfrage aus, erhalten wir die folgende Ausgabe:

Results | Messages

	mindate	maxdate
1	2008-07-31 21:42:52.667	2019-12-01 07:29:22.000

Für unsere untere Schranke erhalten wir das Jahr 2008, für die Obere das Jahr 2019. Als nächstes werden wir für unsere Jahresgrenzen pro Jahr genau eine Dateigruppe erstellen, beginnend ab dem Jahr 2008. Davor sollten wir uns aber noch einmal verdeutlichen welche Vorteile die Verwendung unterschiedlicher Dateigruppen bietet:

- Der Datenhaushalt ist besser organisiert.
- Große Datenmengen können so leichter aus einer Tabelle entfernt werden.
- Verarbeitungsgeschwindigkeit kann skaliert werden: Oft verwendete Datei sollten auf schnellen Laufwerken abgelegt werden, während die selten verwendeten Dateien auf langsameren Laufwerken platziert werden können.
- Schnellere Verarbeitung von Abfragen die auf Spalten der Partitionierung abzielen.
- Schnellere Verarbeitung von Abfragen auf besonders großen Tabellen.

Nachdem wir diese Frage geklärt haben, können wir mit der Erstellung der Dateigruppen beginnen. Mit dem folgenden Befehl werden die gewünschten 12 Dateigruppen erzeugt:

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
USE [master]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2008]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2009]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2010]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2011]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2012]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2013]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2014]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2015]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2016]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2017]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2018]
GO
ALTER DATABASE [StackOverflow] ADD FILEGROUP [2019]
GO
```

Dateien für Dateigruppen erstellen

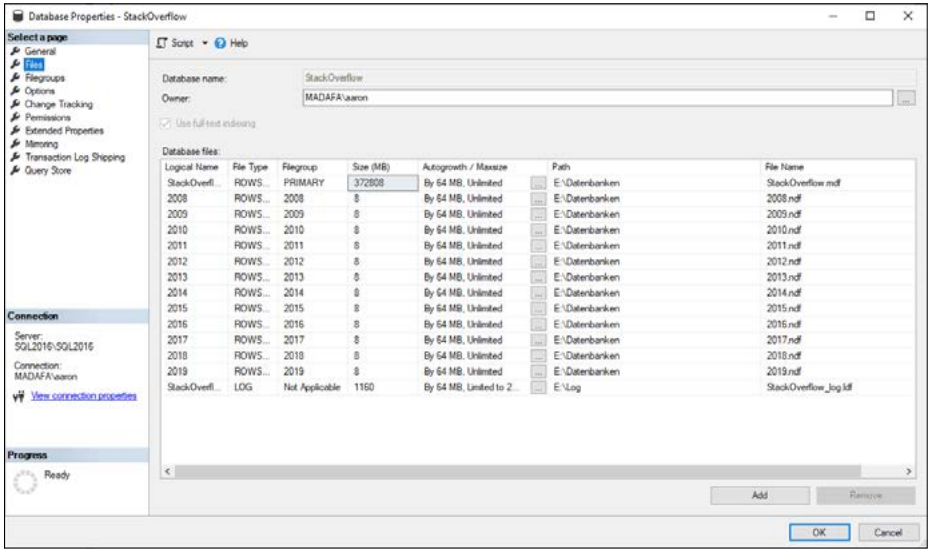
Für jede erstellte Dateigruppe brauchen wir nun eine zusätzlich Datei in der Informationen für die Dateigruppe gespeichert werden können. Mit dem folgenden Befehl fügen wir für jede der zuvor erstellten Dateigruppen eine neue Datei an:

```
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2008', FILENAME = N'E:\
Datenbanken\2008.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2008]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2009', FILENAME = N'E:\
Datenbanken\2009.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2009]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2010', FILENAME = N'E:\
Datenbanken\2010.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2010]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2011', FILENAME = N'E:\
Datenbanken\2011.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2011]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2012', FILENAME = N'E:\
Datenbanken\2012.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2012]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2013', FILENAME = N'E:\
Datenbanken\2013.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2013]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2014', FILENAME = N'E:\
Datenbanken\2014.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2014]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2015', FILENAME = N'E:\
Datenbanken\2015.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2015]
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2016', FILENAME = N'E:\
Datenbanken\2016.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2016]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2017', FILENAME = N'E:\
Datenbanken\2017.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2017]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2018', FILENAME = N'E:\
Datenbanken\2018.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2018]
GO
ALTER DATABASE [StackOverflow] ADD FILE ( NAME = N'2019', FILENAME = N'E:\
Datenbanken\2019.ndf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB
) TO FILEGROUP [2019]
GO
```

Zur Sicherheit können wir das ganze in SSMS noch einmal in den Datenbankeigenschaften überprüfen:



Partitionsfunktion erstellen

Da auf unserer verwendeten Tabelle **dbo.PostHistory** bereits ein Index existiert, müssen wir diesen zunächst entfernen. Das machen wir mit dem folgenden Befehl:

```
ALTER TABLE [dbo].[PostHistory] DROP CONSTRAINT [PK_PostHistory__Id] WITH (
ONLINE = OFF )
```

Anschließend können wir die Partitionsfunktion folgendermaßen bestimmen:

```
CREATE PARTITION FUNCTION [PF_YearPartition] (DATETIME)
AS RANGE RIGHT FOR VALUES
(
    '20080101 00:00:00.000',
    '20090101 00:00:00.000',
    '20100101 00:00:00.000',
    '20110101 00:00:00.000',
    '20120101 00:00:00.000',
    '20130101 00:00:00.000',
    '20140101 00:00:00.000',
    '20150101 00:00:00.000',
    '20160101 00:00:00.000',
    '20170101 00:00:00.000',
    '20180101 00:00:00.000',
    '20190101 00:00:00.000'
);
```

Wir definieren uns also eine Funktion basierend auf **DATETIME**, wobei wir jeweils den 01. Januar jeden Jahres als Schranke wählen.

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Partitionsschema definieren

Als nächstes definieren wir unser Partitionsschema basierend auf der zuvor erstellten Funktion **PF_YearPartition** und bestimmen die einzelnen Dateigruppen, auf die anhand der Partitionsfunktion die Tabelleninformationen verteilt werden sollen:

```
CREATE PARTITION SCHEME PS_YearWise
AS PARTITION PF_YearPartition
TO
(
  '2008', '2009', '2010',
  '2011', '2012', '2013',
  '2014', '2015', '2016',
  '2017', '2018', '2019',
  [PRIMARY]
);
```

Die **PRIMARY** Dateigruppe wird am Ende angehängt, damit arbiträre Werte richtig verarbeitet werden. Kann ein Datensatz also nicht anhand der Partitionsfunktion eingeordnet werden, dient die Primary-Dateigruppe als Aushilfe.

Index erzeugen

Nachdem alle Puzzleteile an ihre korrekte Position gebracht worden sind, können wir nun dafür sorgen, dass die momentan in der Tabelle befindlichen Informationen an ihre korrekte Position (die zugehörige Dateigruppe) gebracht werden.

Dafür erstellen wir für die Tabelle **dbo.PostHistory** einen neuen Index, basierend auf unserer Partitionsfunktion:

```
CREATE CLUSTERED INDEX IX_ID
    ON dbo.PostHistory (ID)
    ON PS_YearWise(CreationDate);
```

Nach Ausführung des Befehls lohnt es sich einen Blick auf den Server zu werfen und zu schauen, wie die Daten nun in die entsprechenden Dateigruppen eingeordnet werden.

Zuordnung überprüfen

Um sicherzustellen, dass die Datensätze korrekt in die Dateigruppen eingeordnet wurden, können wir mit der folgenden Ansicht den Inhalt der Partitionen überprüfen:

```
SELECT DISTINCT o.name as table_name, rv.value as partition_range, fg.name as
file_groupName, p.partition_number, p.rows as number_of_rows
FROM sys.partitions p
INNER JOIN sys.indexes i ON p.object_id = i.object_id AND p.index_id = i.index_id
INNER JOIN sys.objects o ON p.object_id = o.object_id
INNER JOIN sys.system_internals_allocation_units au ON p.partition_id =
au.container_id
INNER JOIN sys.partition_schemes ps ON ps.data_space_id = i.data_space_id
INNER JOIN sys.partition_functions f ON f.function_id = ps.function_id
INNER JOIN sys.destination_data_spaces dds ON dds.partition_scheme_id = ps.data_
space_id AND dds.destination_id = p.partition_number
INNER JOIN sys.filegroups fg ON dds.data_space_id = fg.data_space_id
LEFT OUTER JOIN sys.partition_range_values rv ON f.function_id = rv.function_id
AND p.partition_number = rv.boundary_id
WHERE o.object_id = OBJECT_ID('PostHistory');
```

Alternativ bietet sich auch die Möglichkeit für jede Zeile einer Tabelle die jeweilige Partitionsnummer anzugeben. Dies kann beispielsweise auf die folgende Art umgesetzt werden:

```
SELECT $PARTITION.PF_YearPartition(CreationDate) AS PartitionNumber, *
FROM [dbo].[PostHistory]
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Den Prozess automatisieren

Wie aus dem obigen Beispiel und den beiden Abfragen deutlich geworden sein sollte, wird die PRIMARY-Dateigruppe verwendet, für Datensätze die außerhalb des Partitionsradius liegen. Der Bereich der Partitionen muss also stetig überwacht und angepasst werden. Wie bereits zuvor erwähnt, können wir einen SQL Server Job definieren um die Wartung der Partition automatisch durchführen zu lassen.

SQL Server Jobs werden in einem zuvor definierten Zeitabstand (bspw. einmal pro Woche/Monat) ausgeführt und können uns dabei helfen Partitionsfunktionen zu identifizieren, die angepasst werden müssen. Betrachten wir dazu einmal das folgende Beispiel und die dazugehörigen T-SQL Abfrage:

Wir möchten die Partitionsfunktion finden, deren Bereich bald überschritten wird. Beispielsweise sollten wir gegen Ende des Jahres 2019 bemerken, dass wir eine Partition für das Jahr 2020 benötigen.

```
SELECT o.name as table_name,
  pf.name as PartitionFunction,
  ps.name as PartitionScheme,
  MAX(rv.value) AS LastPartitionRange,
  CASE WHEN MAX(rv.value) <= DATEADD(MONTH, 2, GETDATE()) THEN 1 else 0 END AS
isRequiredMaintenance
--INTO #temp
FROM sys.partitions p
INNER JOIN sys.indexes i ON p.object_id = i.object_id AND p.index_id = i.index_id
INNER JOIN sys.objects o ON p.object_id = o.object_id
INNER JOIN sys.system_internals_allocation_units au ON p.partition_id =
au.container_id
INNER JOIN sys.partition_schemes ps ON ps.data_space_id = i.data_space_id
INNER JOIN sys.partition_functions pf ON pf.function_id = ps.function_id
INNER JOIN sys.partition_range_values rv ON pf.function_id = rv.function_id AND
p.partition_number = rv.boundary_id
GROUP BY o.name, pf.name, ps.name
```

Durch Ausführen der Abfrage erfahren wir, dass unsere Partitionsfunktion **PF_YearPartition** gewartet werden muss. Die folgende Abfrage kann dabei helfen Informationen zu den Partitionsfunktion zu sammeln, die gewartet werden soll:

Hinweis:
Um die Abfrage verwenden zu können, muss der Kommentar ("-") vor "INTO #temp" in der obigen Abfrage entfernt werden!

```
SELECT table_name,
  PartitionFunction,
  PartitionScheme,
  LastPartitionRange,
  CONVERT(VARCHAR, DATEADD(MONTH, 1, LastPartitionRange), 25) AS NewRange,
  'FG_' + CAST(FORMAT(DATEADD(MONTH, 1, LastPartitionRange),'MM') AS VARCHAR(2))
+
    '_' +
    CAST(YEAR(DATEADD(MONTH, 1, LastPartitionRange)) AS VARCHAR(4)) AS
NewFileGroup,
  'File_'+ CAST(FORMAT(DATEADD(MONTH, 1, LastPartitionRange),'MM') AS VARCHAR(2))
+
    CAST(YEAR(DATEADD(MONTH, 1, LastPartitionRange)) AS VARCHAR(4)) AS FileName,
  'E:\Datenbanken\' AS file_path
INTO #generateScript
FROM #temp
WHERE isRequiredMaintenance = 1
```

Zu guter Letzt bleibt nur noch die Erweiterung der Partition. Mit dem folgenden Skript wird eine neue Dateigruppe erzeugt, eine Datei angehängt und der neue Bereich an die Partitionsfunktion angefügt:

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
DECLARE @filegroup NVARCHAR(MAX) = ''
DECLARE @file NVARCHAR(MAX) = ''
DECLARE @PScheme NVARCHAR(MAX) = ''
DECLARE @PFunction NVARCHAR(MAX) = ''

SELECT @filegroup = @filegroup +
    CONCAT('IF NOT EXISTS(SELECT 1 FROM AutoPartition.sys.filegroups WHERE name =
''',NewFileGroup,''')
    BEGIN
      ALTER DATABASE AutoPartition ADD FileGroup ',NewFileGroup,'
    END;'),
    @file = @file + CONCAT('IF NOT EXISTS(SELECT 1 FROM AutoPartition.sys.database_
files WHERE name = ''',FileName,''')
    BEGIN
    ALTER DATABASE AutoPartition ADD FILE
    (NAME = ''',FileName,''',
    FILENAME = ''',File_Path,FileName,'.ndf'',
    SIZE = 5MB, MAXSIZE = UNLIMITED,
    FILEGROWTH = 10MB )
    TO FILEGROUP ',NewFileGroup, '
    END;'),
    @PScheme = @PScheme + CONCAT('ALTER PARTITION SCHEME ', PartitionScheme, '
NEXT USED ',NewFileGroup,';'),
    @PFunction = @PFunction + CONCAT('ALTER PARTITION FUNCTION ',
PartitionFunction, '() SPLIT RANGE (''',NewRange,''');')
FROM #generateScript

EXEC (@filegroup)
EXEC (@file)
EXEC (@PScheme)
EXEC (@PFunction)
```

Werden die hier präsentierten Skripte sequentiell in einer Prozedur realisiert, können wir einen SQL Server Job konfigurieren, der in einem beliebigen Zeitabstand unsere Partitionen erweitert.

Letzte Worte

Gerade die Erweiterung einer Partitionsfunktion ist eine Aufgabe bei der es sich besonders anbietet diese zu automatisieren. Der Overhead in der Pflege von Partitionen wird deutlich verringert und besonders große Tabellen können auf diese Weise effizient genutzt und verwaltet werden.

automatisierung, dateigruppe, filegroup, partition, partitioning, sql, sql server, t-sql

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612