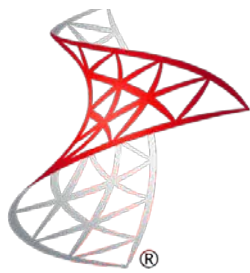


**Microsoft®**



Microsoft®  
**SQL Server® 2012**

## SQL Server 2012 自習書シリーズ No.19

---

データ パーティション入門

---

Published: 2012 年 5 月 25 日

SQL Server 2012 更新版: 2012 年 9 月 30 日

有限会社エスキューエル・クオリティ



この文章に含まれる情報は、公表の日付の時点での Microsoft Corporation の考え方を表しています。市場の変化に応える必要があるため、Microsoft は記載されている内容を約束しているわけではありません。この文書の内容は印刷後も正しいとは保障できません。この文章は情報の提供のみを目的としています。

Microsoft、SQL Server、Visual Studio、Windows、Windows XP、Windows Server、Windows Vista は Microsoft Corporation の米国およびその他の国における登録商標です。

その他、記載されている会社名および製品名は、各社の商標または登録商標です。

© Copyright 2012 Microsoft Corporation. All rights reserved.

## 目次

---

STEP 1. データ パーティションの概要 と自習書を試す環境について.....	4
1.1 データ パーティションの概要 .....	5
1.2 自習書を試す環境について .....	7
STEP 2. データ パーティション の設定手順 .....	8
2.1 データ パーティション作成の流れ.....	9
2.2 手順 2 ファイル グループの作成 .....	10
2.3 手順 3 パーティション関数の作成.....	12
2.4 手順 4 パーティション構成の作成.....	14
2.5 手順 5 テーブルの作成 .....	15
STEP 3. データ パーティション の動作確認 .....	18
3.1 パーティション インデックスの作成.....	19
3.2 インデックスの再構築と再編成 .....	21
3.3 パーティションのデータ圧縮.....	22
STEP 4. スライディング ウィンドウ のシナリオ .....	23
4.1 パーティションの SWITH とスライディング ウィンドウ.....	24
4.2 スライディング ウィンドウの実装手順.....	27
4.3 パーティションのスイッチ： SWITCH.....	34
4.4 古いパーティションの MERGE .....	39
4.5 MERGE の注意点 .....	43
4.6 SWITCH 操作の条件.....	46
STEP 5. 一部のデータ ファイル破損時の 復旧手順.....	48
5.1 一部のデータ ファイル破損時の復旧.....	49
5.2 参考情報： パーティション ウィザード .....	60

## STEP 1. データ パーティションの概要 と自習書を試す環境について

---

この STEP では、データ パーティションの概要と自習書を試す環境について説明します。

この STEP では、次のことを学習します。

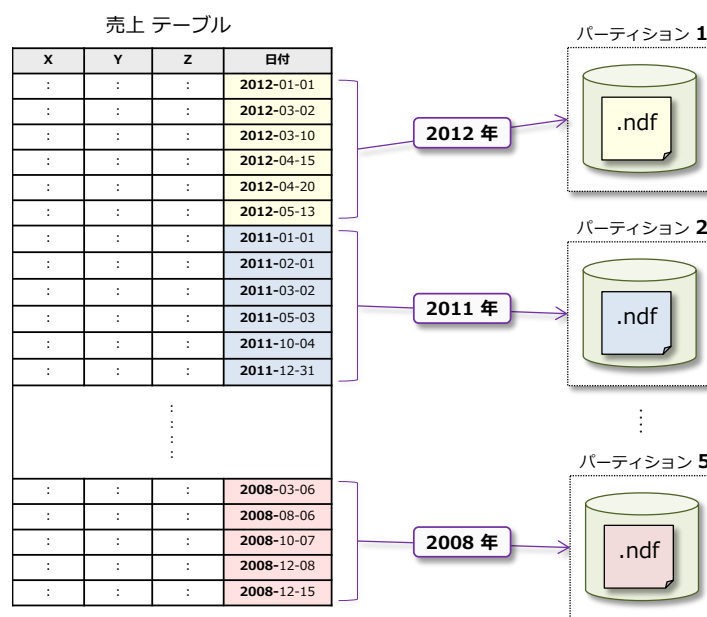
- ✓ データ パーティションの概要
- ✓ 自習書を試す環境について
- ✓ サンプル スクリプトについて

## 1.1 データ パーティションの概要

### ➡ データ パーティションの概要

データ パーティションは、データの範囲によってデータの格納場所（パーティション）を分割することができる、SQL Server 2005 から提供された機能です。データ パーティションは、パフォーマンスとスケーラビリティの向上、運用管理性の向上に大変役立ち、特に大規模データベース環境では、欠かせない機能です。

データ パーティションの典型的な利用方法は、次のように日付の範囲（年ごとや四半期ごとなど）でデータの格納場所（ファイル）を分割するというものです。



このようにデータを日付で分割しておけば、「**不要になった古いパーティションの削除**」や「**新しいパーティションへの大量データ挿入（バッチ挿入）**」などのパフォーマンスを大幅に向上させることができます。

また、データ パーティションは、「**インデックスの再構築と再編成**」や「**バックアップと復元**」、「**データ圧縮**」をパーティション単位で実行できることも大きなメリットです。インデックスの再構築は、再構築の実行中は、ユーザーアクセスがブロックされるので、テーブル全体をまとめて再構築するよりも、パーティション単位で再構築したほうが、ブロックを局所化することができます。また、古いパーティションは、データの追加や更新がされない場合が多いので、そういったパーティションに対しては、インデックスを再構築しなくて済みます。その分、テーブル全体を再構築するよりも、再構築時間を大幅に短縮することができます。

**データ圧縮**は、SQL Server 2008 から提供された機能ですが、これを利用すると、利用頻度の少ない古いパーティションのデータを圧縮しておくことで、ディスク コストを大幅に削減することができます（データ圧縮は、データの特性にもよりますが、4～10 倍程度の圧縮率を実現できます）。

また、パーティションを複数のファイル グループおよびデータ ファイル（.ndf）で構成しておけ

ば、一部のデータ ファイルが破損してもそのまま稼働させることが可能です。さらに、パーティション単位でバックアップしていたファイルを利用して、破損したパーティションのみを復元するだけで、完全復旧させることができます。

以上の機能は、Step 2 以降で、1 つ 1 つ実際に試しながら説明しますので、ぜひチャレンジしてみてください。

なお、データ パーティション機能は、Oracle Database での「レンジ パーティション」に相当する機能です。

## ➡ SQL Server 2008 以降で提供された機能

2 つ前のバージョンの SQL Server 2008 からは、データ パーティションの新機能として、次のものが追加されました。

- パーティションの平行処理によるパフォーマンスの大幅な向上
- パーティション単位でのデータ圧縮
- パーティションに関連付けられたインデックス付きビュー
- パーティション単位でのロック エスカレーション
- パーティション ウィザードによる GUI からの設定

SQL Server 2012 からは、次の新機能が追加されました。

- 15,000 パーティション（作成できるパーティション数の上限が 15,000 に UP）

このように、データ パーティションには、多くの機能があり、SQL Server 2008 以降でさらにパワーアップ（特にパフォーマンスが向上）しています。大規模データベース環境では広く利用されており、不可欠と言っても過言ではない機能です。筆者自身も、データベース サイズが 9TB（テラ バイト）のシステムを設計する際にデータ パーティションを採用して（年ごとにパーティションを分割して）、パフォーマンスを大きく向上させたことがあります。皆さんも、ぜひ活用していただければと思います。

## 1.2 自習書を試す環境について

---

### ➡ 必要な環境

この自習書で実習を行うために必要な環境は次のとおりです。

#### OS

Windows Server 2008 SP2 以降 または  
Windows Server 2008 R2 SP1 以降 または  
Windows Server 2012 または  
Windows Vista SP2 以降 または Windows 7 SP1 以降 または Windows 8

#### ソフトウェア

SQL Server 2012

この自習書内での画面やテキストは、OS に Windows Server 2008 R2 (x64)、ソフトウェアに SQL Server 2012 Enterprise エディション (x64) を利用して記述しています。

## STEP 2. データ パーティション の設定手順

---

この STEP では、データ パーティションの設定手順について説明します。

この STEP では、次のことを学習します。

- ✓ データ パーティション作成の流れ
- ✓ データ パーティションの設定手順



## 2.1 データ パーティション作成の流れ

---

### ➡ データ パーティション作成の流れ

データ パーティションを作成する手順は、次のとおりです。

1. パーティションの事前計画
2. 1 つまたは複数の**ファイル グループ**の作成
3. **パーティション関数** (PARTITION FUNCTION) の作成
4. **パーティション構成** (PARTITION SCHEME) の作成
5. パーティション構成を指定してテーブルの作成

以降では、これらの手順を 1 つずつ説明します。

### ➡ 手順 1 パーティションの事前計画

データ パーティションを作成するには、まず最初に次の項目を計画しておく必要があります。

- パーティション キー列の決定 (どの列でデータを分割するのかを決定)
- 境界値の決定 (年ごとや四半期ごとなど、パーティションを分割する単位の決定)
- パーティション数の計画 (古いデータのアーカイブ計画)
- ストレージ (ファイル グループ) 配置とインデックス配置の計画

この Step では、一番多く利用されている実装の、パーティション キー列へ「**日付**」、境界値を「**年ごと**」へ設定したパーティションを作成する手順を説明します。パーティション数は、操作を試しやすくするために「**2009～2012 年の 4 年分**」、ストレージ (ファイル グループ) は同じドライブへ配置することとします。

## 2.2 手順 2 ファイル グループの作成

### ➡ ファイル グループの作成

次の手順は、ファイル グループの作成です。データ パーティションは、1 つのファイル グループ内（既定の PRIMARY ファイル グループ内）へ作成することも可能ですが、複数のファイル グループへ分散して格納しておくことで、運用管理性を向上させることができます。たとえば、ファイル グループ単位でのバックアップ／リストアが実行できるようになったり、ファイル グループを分けておくことで、一部のファイル グループに破損が発生してもそのまま稼働させられることができ、かつ破損したファイル グループのみを復元するだけで完全復旧が可能になる、などのメリットが得られます（これらは Step 5 で説明します）。

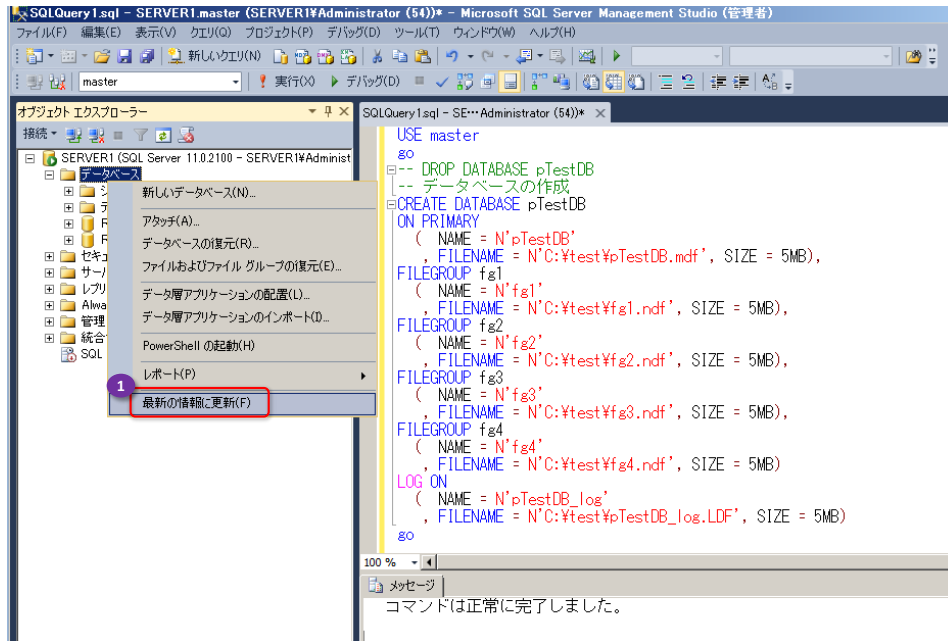
### ➡ Let's Try

それでは、ファイル グループを作成してみましょう。ここでは、「pTestDB」という名前のデータベースを 5 つのファイル グループで作成します。ファイル グループの名前は **PRIMARY**、**fg1**、**fg2**、**fg3**、**fg4** とします。

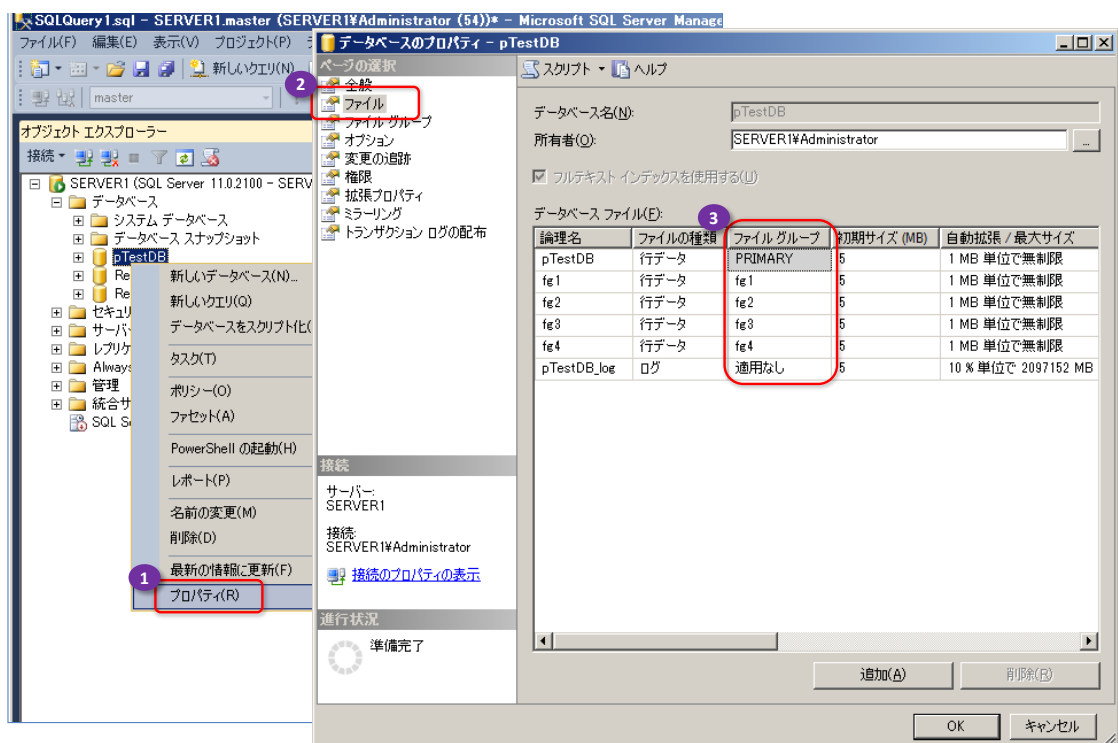
1. まずは、SQL Server Management Studio を起動します。
2. 起動後、クエリ エディターを開き、次のように **CREATE DATABASE** ステートメントを実行して、データベースを作成します（完成版のスクリプトは、サンプル スクリプト フォルダの「**Step2\_CreatePartition.sql**」ファイルへ記載しているので、入力が面倒な方は、それをコピーして実行してください。また、データベースの作成先となるフォルダーのパス「**C:¥test**」は皆さんの環境に合わせて変更してください）。

```
USE master
go
CREATE DATABASE pTestDB
ON PRIMARY
( NAME = N' pTestDB'
, FILENAME = N' C:¥test¥pTestDB.mdf' , SIZE = 5MB),
FILEGROUP fg1
( NAME = N' fg1'
, FILENAME = N' C:¥test¥fg1.ndf' , SIZE = 5MB),
FILEGROUP fg2
( NAME = N' fg2'
, FILENAME = N' C:¥test¥fg2.ndf' , SIZE = 5MB),
FILEGROUP fg3
( NAME = N' fg3'
, FILENAME = N' C:¥test¥fg3.ndf' , SIZE = 5MB),
FILEGROUP fg4
( NAME = N' fg4'
, FILENAME = N' C:¥test¥fg4.ndf' , SIZE = 5MB)
LOG ON
( NAME = N' pTestDB_log'
, FILENAME = N' C:¥test¥pTestDB_log.LDF' , SIZE = 5MB)
```

3. 作成後、オブジェクト エクスプローラーで[データベース]フォルダーを右クリックして[最新の情報に更新]をクリックして、作成したデータベースを確認します。



4. 次に、「pTestDB」データベースを右クリックして、[プロパティ]をクリックし、データベースのプロパティを確認します。



[ファイル] ページを参照すると、5つのファイル グループ (PRIMARY と fg1～fg4) が作成されていることを確認できます。

以降の手順では、fg1～fg4 の 4つのファイル グループへ「2009～2012年の4年分」のデータを格納できるようにパーティション関数とパーティション構成を作成していきます。

## 2.3 手順 3 パーティション関数の作成

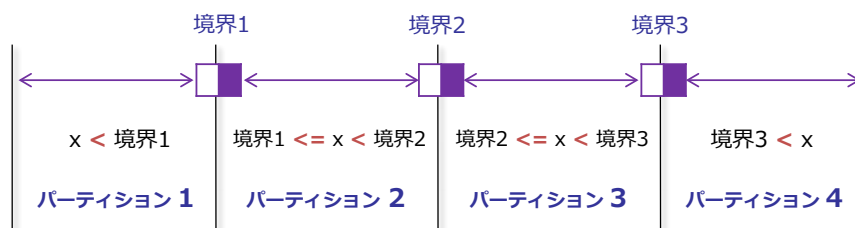
### ➡ パーティション関数の作成

次の手順は、「パーティション関数」(PARTITION FUNCTION) の作成です。パーティション関数は、パーティションを分割する際の境界 (Boundary) となるものです。作成には、「**CREATE PARTITION FUNCTION**」ステートメントを次のように利用します。

```
USE データベース名
go
CREATE PARTITION FUNCTION 名前 (データ型)
AS RANGE [ RIGHT | LEFT ] FOR VALUES (境界1, 境界2, 境界3)
```

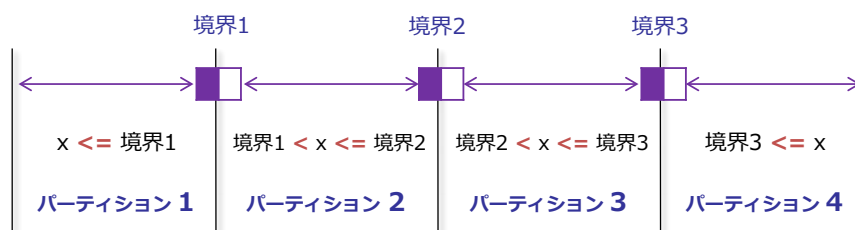
データ型には、パーティションを分割するキーのデータ型を指定し、RANGE (範囲) には、RIGHT または LEFT を指定します。両者の違いは、次のとおりです。

#### RANGE RIGHT (境界1, 境界2, 境界3)



**RIGHT (右) は、右側のパーティションに境界値を含む**

#### RANGE LEFT (境界1, 境界2, 境界3)



**LEFT (左) は、左側のパーティションに境界値を含む**

境界値は、パーティションの区切りになり、RIGHT を指定した場合は、右側のパーティションへ境界値を含み、LEFT を指定した場合は、左側のパーティションへ境界値を含みます。

### ➡ Let's Try

それでは、これを試してみましょう。ここでは、「**2009～2012 年の 4 年分**」のデータを格納するために、3つの境界を **RIGHT** を指定して作成します。

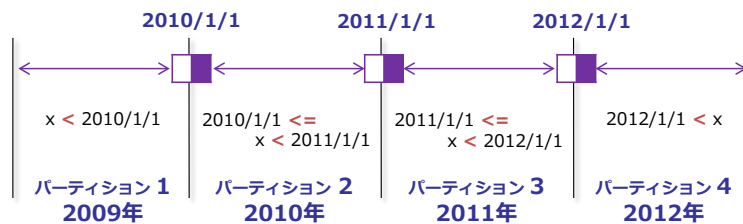
1. パーティション関数の名前は、任意に設定し (**pFunc1** など)、データ型には、日付を格納で

きる「**datetime**」型を指定します。

```
USE pTestDB
go
CREATE PARTITION FUNCTION pFunc1 (datetime)
AS RANGE RIGHT FOR
VALUES (' 2010/01/01', ' 2011/01/01', ' 2012/01/01')
```

RANGE へ RIGHT を指定して、境界値へ「**2010/01/01、2011/01/01、2012/01/01**」を指定しているので、次の 4 つのパーティション（4 年分）が作成できるようになります。

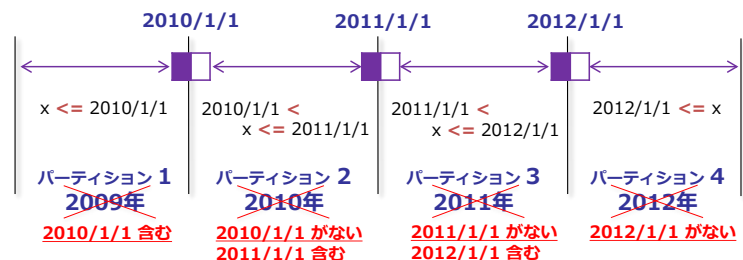
**RANGE RIGHT** ('2010/01/01', '2011/01/01', '2012/01/01')



**Note : LEFT よりも RIGHT がお勧め**

RANGE には、上の手順のように RIGHT を指定することをお勧めします。LEFT と指定した場合は、次のように境界が構成されます。

**RANGE LEFT** ('2010/01/01', '2011/01/01', '2012/01/01')



LEFT の場合は、境界値へ 1 月 1 日を指定すると、1 月 1 日が前の年のパーティションへ含まれてしまいます。これを回避するには、境界値を「**200x 年 12 月 31 日 23:59:59.997**」のように、その年の最後の時刻を指定しなければなりません。

また、Step 4 で説明するスライディング ウィンドウのシナリオでは、LEFT と RIGHT で動作が異なるので（特に MERGE と SPLIT 時のファイル グループの動作が異なるので）、RIGHT を利用することをお勧めします。

## 2.4 手順 4 パーティション構成の作成

### ➡ パーティション構成の作成

パーティション関数の作成後は、「パーティション構成」(PARTITION SCHEME)を作成します。パーティション構成は、パーティション関数によって分割された RANGE (パーティション) とファイル グループをマッピングするためのものです。パーティション構成を作成するには、「**CREATE PARTITION SCHEME**」ステートメントを次のように利用します。

```
USE データベース名
go
CREATE PARTITION SCHEME 名前
AS PARTITION パーティション関数名
TO ( フィルグループ 1, ファイルグループ 2, ... )
```

ファイル グループの指定は、パーティション関数で作成するパーティションの数分を指定する必要があります。たとえば、境界値が 3 つ (パーティションが 4 つ) の場合は、4 つのファイル グループを指定する必要があります。

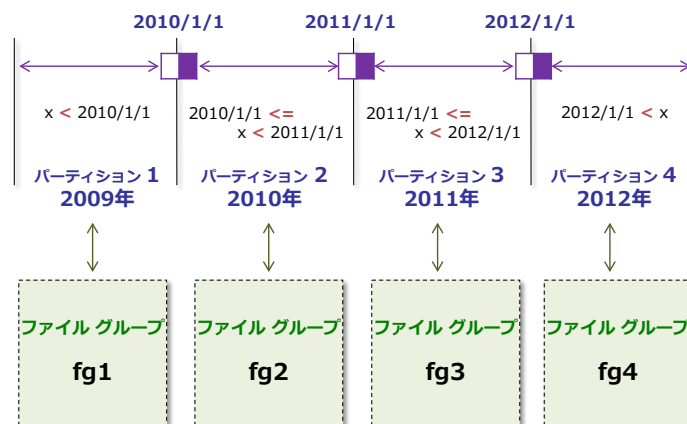
### ➡ Let's Try

それでは、これを試してみましょう。

1. パーティション構成の名前は、任意に設定し (**pScheme1** など)、パーティション関数には、前の手順で作成した「**pFunc1**」を指定します。

```
USE pTestDB
go
CREATE PARTITION SCHEME pScheme1
AS PARTITION pFunc1
TO (fg1, fg2, fg3, fg4)
```

pFunc1 では、境界値を 3 つ (パーティションが 4 つ) へ設定しているので、次のようにファイル グループとマッピングされます。



## 2.5 手順 5 テーブルの作成

### ➡ テーブルの作成

次の作業は、テーブルの作成です。テーブルをデータ パーティション化するには、CREATE TABLE ステートメントの実行時に、ON 句を利用して、利用するパーティション構成を指定します。これは次のように記述します。

```
CREATE TABLE テーブル名
( 列名 データ型
, 列名 データ型 , ... )
ON パーティション構成名 ( パーティションキー列 )
```

### ➡ Let's Try

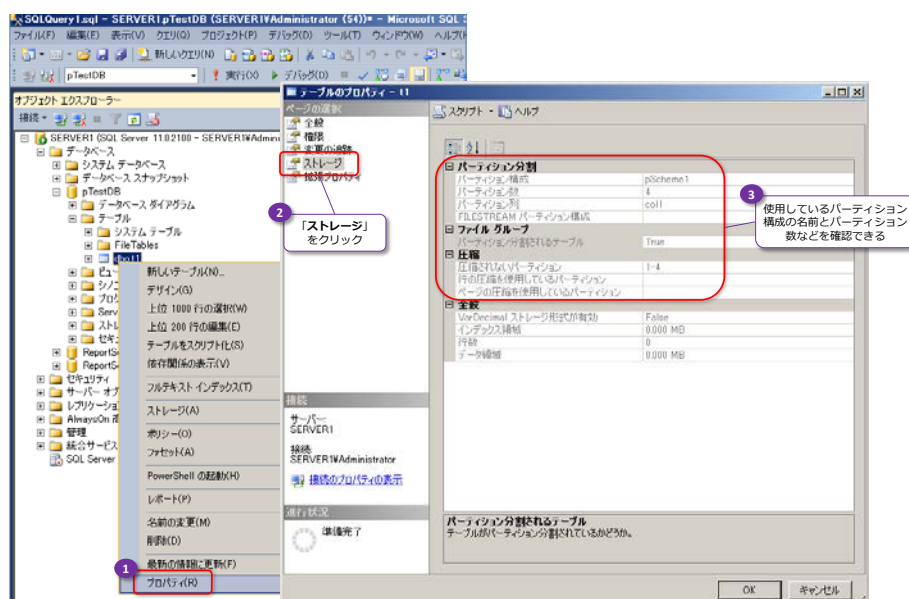
それでは、これを試してみましょう。

1. 前の手順で作成したパーティション構成「pScheme1」を指定して、「t1」という名前のテーブルを作成し、「col1」列をパーティション キー列へ設定してみましょう。

```
USE pTestDB
CREATE TABLE t1
( id int IDENTITY(1,1) NOT NULL
, col1 datetime
) ON pScheme1 (col1)
```

このようにパーティション構成を指定してテーブルを作成することで、パーティション関数で設定された 4 つのパーティションヘデータを格納できるようになります。

2. 次に、オブジェクト エクスプローラーで作成したテーブルのプロパティを開いてみましょう。



## ➡ データの追加

3. 次に、テーブルへデータを追加してみましょう。

```
INSERT INTO t1
VALUES ('2009/10/15')
      , ('2010/01/01')
      , ('2010/05/22')
      , ('2010/12/31')
      , ('2011/01/01')
      , ('2011/12/31')
      , ('2012/01/01')
```

## ➡ \$PARTITION : どのパーティションへ格納されたかを確認

テーブルへ追加したデータがどのパーティションへ格納されたかどうかを確認するには、「\$PARTITION」という特殊のキーワードを利用します。これは、次のように利用します。

**\$PARTITION.** パーティション関数名 (値)

ドット (.) に続けてパーティション関数を指定し、値を与えると、その値がどのパーティションに格納されるかを、パーティション番号 (左のパーティションから 1 から始まる連番) で返すことができます。では、これを試してみましょう。

1. 次のように「\$PARTITION.pFunc1(col1)」と利用して、col1 列へ追加したデータがどのパーティションへ格納されたかを確認してみましょう。

```
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1
```

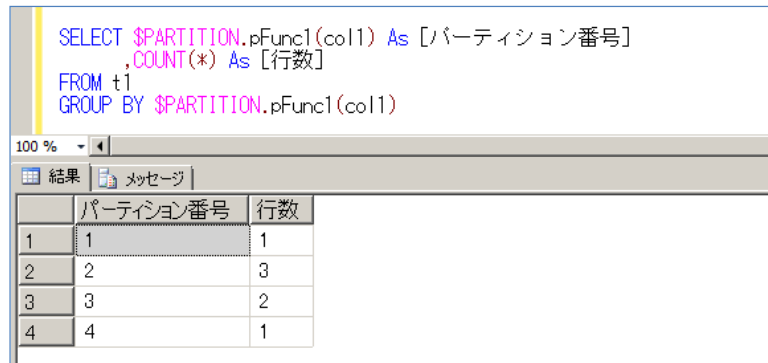
	id	col1	パーティション番号
1	1	2009-10-15 00:00:00.000	1
2	2	2010-01-01 00:00:00.000	2
3	3	2010-05-22 00:00:00.000	2
4	4	2010-12-31 00:00:00.000	2
5	5	2011-01-01 00:00:00.000	3
6	6	2011-12-31 00:00:00.000	3
7	7	2012-01-01 00:00:00.000	4

2009 年のデータがパーティション「1」へ、2010 年が「2」、2011 年が「3」、2012 年が「4」へ格納されていることを確認できます。

2. 次に、\$PARTITION の結果を **GROUP BY** 句によってグループ化して、各パーティション内のデータ件数を確認してみましょう。



```
SELECT $PARTITION.pFunc1(col1) As [パーティション番号]  
      ,COUNT(*) As [行数]  
FROM t1  
GROUP BY $PARTITION.pFunc1(col1)
```



The screenshot shows a SQL query window with the following text:

```
SELECT $PARTITION.pFunc1(col1) As [パーティション番号]  
      ,COUNT(*) As [行数]  
FROM t1  
GROUP BY $PARTITION.pFunc1(col1)
```

Below the query window, the 'Results' pane is visible, showing a table with two columns: 'パーティション番号' (Partition Number) and '行数' (Row Count). The table contains four rows of data.

	パーティション番号	行数
1	1	1
2	2	3
3	3	2
4	4	1

このようにデータ パーティションでは、\$PARTITION キーワードを利用することで、データがどのパーティションへ格納されたかを確認したり、パーティション内のデータ件数を調べられるようになります。

## STEP 3. データ パーティション の動作確認

---

この STEP では、データ パーティションに対してインデックスを作成したり、インデックスの再構築を実行したり、いくつかの動作確認を行っていきましょう。

この STEP では、次のことを学習します。

- ✓ パーティション インデックスの作成
- ✓ インデックスの再構築と再編成
- ✓ パーティションのデータ圧縮

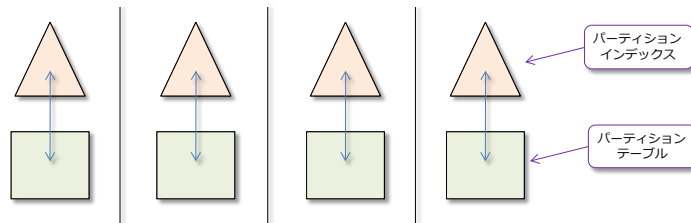
### 3.1 パーティション インデックスの作成

#### ➡ パーティション インデックスの作成

データ パーティションでは、インデックスをパーティション化することもできます。これは、テーブルの作成時と同様、インデックスの作成時（CREATE INDEX ステートメント実行時）に、ON 句でパーティション構成を指定するだけで完了します。

```
CREATE CLUSTERED INDEX インデックス名
ON テーブル名 (列名)
ON パーティション構成名 (パーティションキー列)
```

これにより、テーブル データとインデックスを 1 対 1 で対応させることができます。

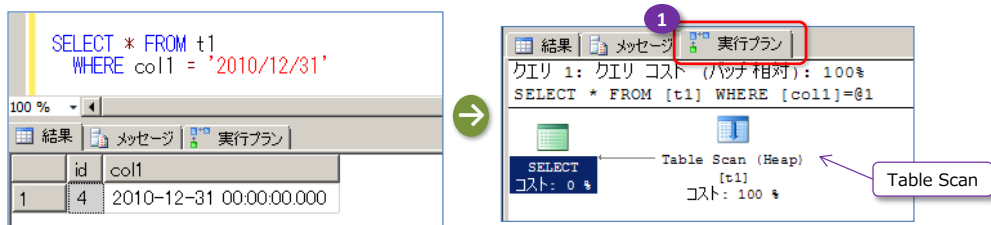


#### ➡ Let's Try

それでは、これを試してみましょう。

1. まずは、パーティション インデックスを作成していない場合のデータ検索を行ってみます。Management Studio の【クエリ】メニューで【**実際の実行プランを含める**】をクリックしてから、次の検索を実行します。

```
SELECT * FROM t1
WHERE col1 = '2010/12/31'
```



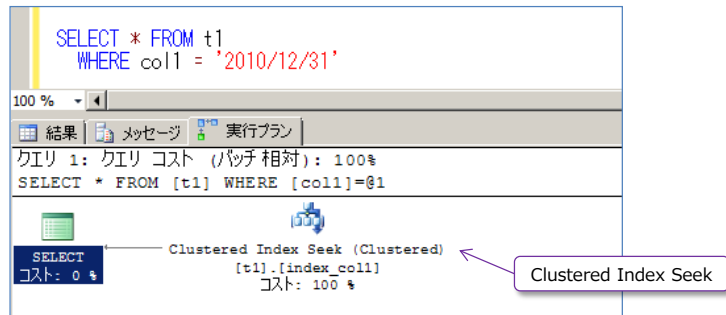
結果は、Table Scan が発生していることを確認できます。

2. 次に、パーティション インデックスを作成します。

```
CREATE CLUSTERED INDEX index_col1
ON t1 (col1)
ON pScheme1 (col1)
```

3. 作成後は、もう一度同じ検索を実行します。

```
SELECT * FROM t1
WHERE col1 = '2010/12/31'
```



今度は、Clustered Index Seek へ変わって、インデックスが利用されたことを確認できます。

## 3.2 インデックスの再構築と再編成

### ➡ インデックスの再構築と再編成

パーティション インデックスは、パーティション単位で「**インデックスの再構築**」(REBUILD)と「**インデックスの再編成**」(REORGANIZE)を実行することができます。これは、**ALTER INDEX** ステートメントを次のように利用します。

```
ALTER INDEX インデックス名
ON テーブル名
{ REBUILD | REORGANIZE } PARTITION = パーティション番号
```

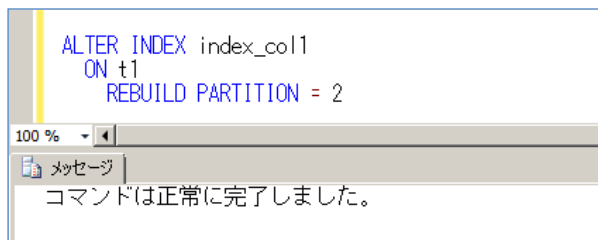
SQL Server 2000 以前のコマンド「**DBCC DBREINDEX**」(再構築)と「**DBCC INDEXDEFRAG**」(再編成)は、パーティション機能に対応していないので、パーティション単位で再構築または再編成を実行したい場合には、必ず **ALTER INDEX** ステートメントを利用しなければなりません。

### ➡ Let's Try

それでは、これを試してみましょう。

1. まずは、パーティション番号「**2**」のインデックスを**再構築 (REBUILD)** してみましょう。

```
ALTER INDEX index_col1
ON t1
REBUILD PARTITION = 2
```



「**コマンドは正常に完了しました**」と表示されれば、再構築が完了しています。

2. 次に、パーティション番号「**3**」のインデックスを**再編成 (REORGANIZE)** してみましょう。

```
ALTER INDEX index_col1
ON t1
REORGANIZE PARTITION = 3
```

このようにデータ パーティションでは、パーティション単位でインデックスの再構築および再編成が行えるのが大きなメリットです。

### 3.3 パーティションのデータ圧縮

#### ➡ パーティションのデータ圧縮

**データ圧縮**（Data Compression）は、SQL Server 2008 から提供された機能で、その名の通り、データを圧縮できる機能です。データ圧縮は、パーティション単位で行うことができます。データ圧縮を行うには、**ALTER TABLE** ステートメントを次のように利用します。

```
ALTER TABLE テーブル名
REBUILD PARTITION = パーティション番号
WITH ( DATA_COMPRESSION = ROW | PAGE | NONE )
```

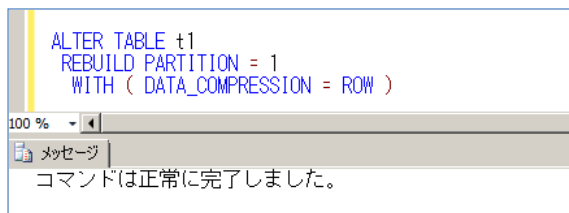
**REBUILD PARTITION** に続けて、パーティション番号を指定し、**DATA\_COMPRESSION=** では、データ圧縮の種類（**ROW** は行単位の圧縮、**PAGE** はページ単位の圧縮、**NONE** は圧縮なし）を指定します。

#### ➡ Let's Try

それでは、これを試してみましょう。

1. パーティション番号「1」に対して、データ圧縮（行圧縮）を実行してみましょう。

```
ALTER TABLE t1
REBUILD PARTITION = 1
WITH ( DATA_COMPRESSION = ROW )
```



「**コマンドは正常に完了しました**」と表示されれば、データ圧縮が完了しています。

このように、データ パーティションでは、パーティション単位でデータ圧縮が行えるのも大きなメリットです。古いパーティションなど、利用頻度の低いデータを圧縮しておけば、ディスク コストを大きく削減することができます（データ圧縮機能は、データの特性にもよりますが、4～10 倍程度の圧縮率が実現できます）。

## STEP 4. スライディング ウィンドウ のシナリオ

---

この STEP では、データ パーティションの一番のメリットである「**SWITCH**」機能を利用したデータの削除と、古いパーティションを定期的にアーカイブ テーブルへ保管するシナリオである「**スライディング ウィンドウ**」を実装する手順について説明します。

この STEP では、次のことを学習します。

- ✓ パーティションの SWITCH
- ✓ スライディング ウィンドウの設定手順
- ✓ SPLIT による新しいパーティションの作成
- ✓ MERGE による古いパーティションのマージ
- ✓ MERGE の注意点
- ✓ SWITCH 操作の条件

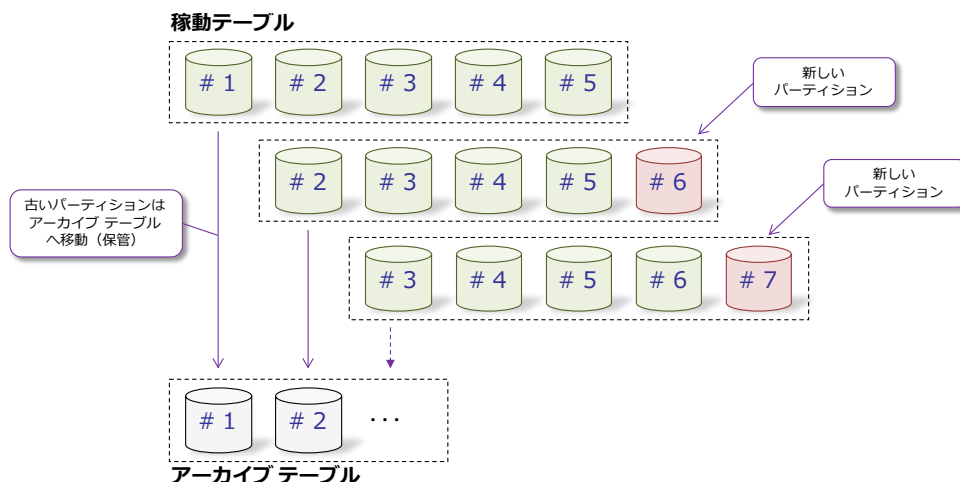
## 4.1 パーティションの SWITCH とスライディング ウィンドウ

### ➡ パーティションの SWITCH

データ パーティションの一番のメリットは、不要になったパーティションを簡単に切り離せる点にあります。この操作は、「**パーティションの SWITCH**」(パーティションの切り替え)と呼ばれます。SWITCH によるデータ移動は、内部的なポインター変更のみで完了するので、SELECT INTO 操作などのように物理的なデータ移動は発生しません。したがって、SWITCH 操作は瞬間的に完了します。

### ➡ 古いデータをアーカイブへ移動する「スライディング ウィンドウ」シナリオ

SWITCH 機能を利用すれば、古いデータ (パーティション) を定期的にアーカイブ テーブルへ移動させることが簡単に行えます。これは、「**スライディング ウィンドウ**」(Sliding Window) と呼ばれ、たとえば、稼働テーブルを 5 つのパーティションとして、古いパーティションはアーカイブ テーブルへ移動 (SWITCH) していくというシナリオであれば、次のようになります。



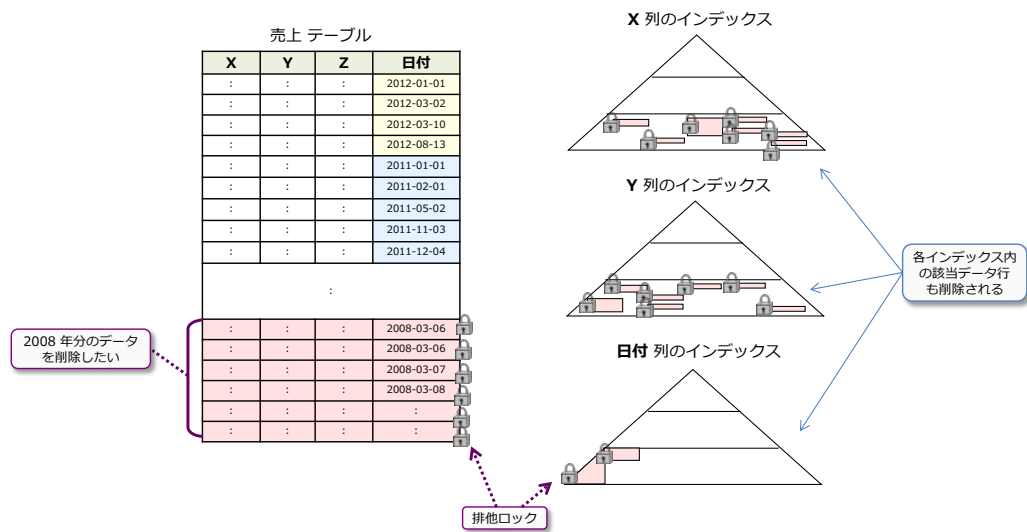
このようにデータ パーティションでは、不要になったパーティションを簡単に切り離せる (SWITCH できる) のが一番のメリットです。データ パーティションは、データ削除時の大幅なパフォーマンスの向上と運用管理性の向上に大きく貢献します。

### ➡ データ削除時のパフォーマンス向上について

データ削除時のパフォーマンス向上については、データ パーティションを利用していない場合の内部動作を考えると、理解しやすいと思います。次の図は、データ パーティションを利用していない場合に、データを削除したときの様子です。



## データ パーティションを利用しない場合のデータの削除時の内部動作



この図は、売上データの過去 4 年分を保存して（稼動テーブルを 4 年分として）、4 年以上経過したデータ（2008 年のデータ）を削除するというシナリオです。また、売上テーブルの「X」列、「Y」列、および「日付」列には、インデックスを作成しているとします。

このような場合に、2008 年のデータを削除するには、内部的には該当データのインデックス内のデータも削除されることになり、また、そのすべてのデータに排他ロックがかかることに注意しなければなりません。たとえば、削除するデータが 10 万件だった場合には、実際のデータから 10 万件削除されるだけでなく、「X」列のインデックスからも 10 万件分、「Y」列のインデックスからも 10 万件分、「日付」列のインデックスからも 10 万件分のデータが削除されることになり、合計 40 万件分のデータの削除が発生します。

このように大量のデータ削除が発生した場合は、トランザクション ログへの大量のログ書き込みが発生し、ログ サイズが大きく膨れ上がります（削除データが 40 万件なら、ログには 40 万件 + a のログ レコードが書き出されます）。これでは、トランザクション ログ（ディスク）への大量書き込みが発生し、トランザクションの完了に時間がかかることになります。

また、データの削除時には、削除対象となるデータ（インデックス内も含む）に排他ロックをかける必要があるため、大量のロックが獲得されることにもなります。削除対象が 40 万件なら 40 万件 + a の排他ロック（行単位のロックの場合）が獲得されます。しかも、この排他ロックは、40 万件分すべてのデータの削除が完了するまでは解放されません。排他ロックの獲得数が多い場合には、ロック エスカレーションが発生して、テーブル全体が排他ロックされる可能性もあります。これでは、同時実行されるトランザクションは、データの削除が完了するまで、アクセスできない（ロック待ちの状態）となってしまいます。

このようにデータ パーティションを利用しない場合の大量のデータ削除は、パフォーマンスが著しく低下する可能性があります。なお、SQL Server 2000 までは、こういった状況を回避する方法として、サポート技術情報（KB: Knowledge Base）の文書番号 **323630**「**SQL Server でロックのエスカレーションが原因で発生するブロッキングの問題を解決する**」(<http://support.microsoft.com/kb/323630/ja>) を提供しています。この KB では、次のように DELETE ステートメントによる削除データ（影響となる行数）を **SET ROWCOUNT** ステートメントによって制限

し、ロック数を減らす方法を紹介しています。

```
-- 5,000件ずつデータを削除
SET ROWCOUNT 5000
    DELETE FROM t1 WHERE 削除条件
WHILE (@@ROWCOUNT > 0)
    DELETE FROM t1 WHERE 削除条件
SET ROWCOUNT 0
```

しかしこの方法では、データを少しずつ削除するので、削除に時間がかかってしまうという問題と、運用管理が複雑になってしまうというデメリットがあります（獲得するロック数を減らすことで同時実行トランザクションのパフォーマンスは向上するので、全体としてのスループットは向上します）。

この問題は、データ パーティションを利用すれば解決できるので、SQL Server 2005 以降のバージョンを利用している場合は、データ パーティション機能を利用することをお勧めします。データ パーティションであれば、削除したいパーティションを切り離す（SWITCH する）だけで、削除は一瞬で完了します。

## 4.2 スライディング ウィンドウの実装手順

### ➡ スライディング ウィンドウの実装手順

次に、「スライディング ウィンドウ」の実装手順を試してみましょう。

スライディング ウィンドウを実装する、おおまかな流れは、次のとおりです。

- 稼働テーブルとまったく**同じ範囲のパーティション関数とパーティション構成**を作成し、アーカイブ テーブルへ適用します（SWTICH 操作は、同じファイル グループ内でのみ有効になるので、これを行っておく必要があります）。

**Note : 1つのファイル グループのみでパーティションを実装する場合**

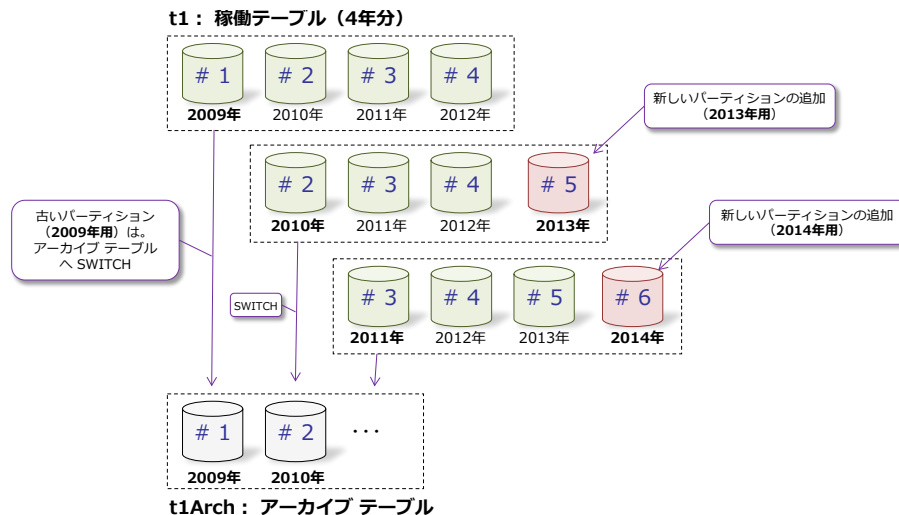
複数のファイル グループを作成せずに、1つのファイル グループのみでパーティションを実装する場合は、同じ範囲のパーティション関数を作成する必要はありません。しかし、1つのファイル グループのみで実装する場合は、パーティション単位でのバックアップや、一部のデータ ファイル破損時のデータ復旧（Step 5 で説明）などを実現できないので、複数のファイル グループでパーティションを実装することをお勧めします。

- 稼働テーブルと**同じパーティション インデックス**を、アーカイブ テーブル側へも作成します。
- 稼働テーブルと**同じデータ圧縮の設定**を、アーカイブ テーブル側へも設定します。
- 稼働テーブルへ新しいパーティションを追加するときは、次の手順で行います（この作業は、アーカイブ テーブルに対しても同様に行います）。
  1. ファイル グループの追加
  2. パーティション構成の変更（**NEXT USED** ファイル グループの指定）
  3. パーティション関数（境界値）の追加（**SPLIT RANGE** によるパーティション分割）
- パーティションの移動には **ALTER TABLE .. SWITCH** ステートメントを利用します。構文は、次のとおりです。

```
ALTER TABLE 稼働テーブル
SWITCH PARTITION n TO アーカイブテーブル PARTITION n
```

### ➡ Let's Try

それでは、これを試してみましょう。ここでは、前の Step で作成した「**t1**」テーブル（4 年分のパーティション）を稼働テーブルとし、次のように古いパーティションをアーカイブ用の「**t1Arch**」テーブルへ SWITCH していくシナリオとして、スライディング ウィンドウの実装手順を試してみましょう。

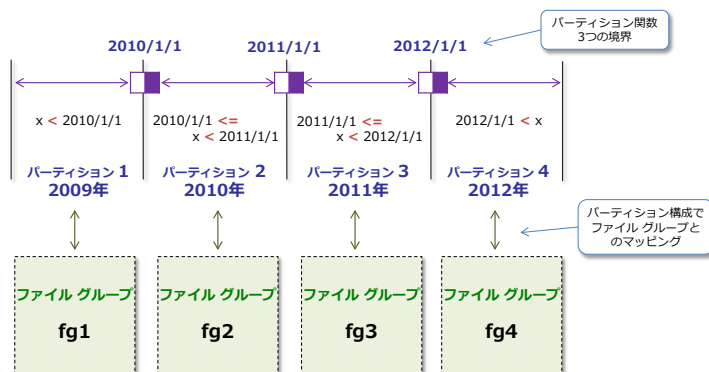


## ➡ 手順 1 アーカイブ テーブル用のパーティション関数とスキームの作成

スライディング ウィンドウを実装する最初の手順は、稼働テーブルとまったく同じ範囲のパーティション関数とパーティション構成を、アーカイブ テーブル用に作成することです。SWITCH 操作は、同じファイル グループ内でのみ有効となるので、これを行っておく必要があります。では、これを試してみましょう。

1. 稼働テーブル用のパーティション関数「pFunc1」と同じ範囲 (3 つの境界値) のアーカイブ用の「pFunc1Arch」を作成し、パーティション構成についても、「pScheme1」と同じように「pScheme1Arch」を作成します。

```
USE pTestDB
go
CREATE PARTITION FUNCTION pFunc1Arch (datetime)
AS RANGE RIGHT FOR
VALUES ('2010/01/01', '2011/01/01', '2012/01/01')
go
CREATE PARTITION SCHEME pScheme1Arch
AS PARTITION pFunc1Arch
TO (fg1, fg2, fg3, fg4)
go
```



## ➡ 手順 2 アーカイブ テーブルの作成

- 次に、前の手順で作成したパーティション構成「pScheme1Arch」を指定して、アーカイブテーブルを「t1Arch」という名前で作成します。

```
CREATE TABLE t1Arch
( id int IDENTITY(1,1) NOT NULL
, col1 datetime
) ON pScheme1Arch(col1)
```

## ➡ 手順 3 アーカイブ テーブルへパーティション インデックスの作成

- 次に、稼働テーブルへ作成しているパーティション インデックスと同じパーティション インデックスをアーカイブ テーブルに対しても作成します。

```
CREATE CLUSTERED INDEX index_col1
ON t1arch (col1)
ON pScheme1Arch (col1)
```

## ➡ 手順 4 アーカイブ テーブル側のパーティションへデータ圧縮

- 稼働テーブル側でデータ圧縮を設定している場合は、アーカイブ側のパーティションに対しても、同じようにデータ圧縮を設定しておく必要があります。Step 3.3 では、パーティション番号「1」に対してデータ圧縮（行圧縮）を設定していたので、同じように設定します。

```
ALTER TABLE t1arch
REBUILD PARTITION = 1
WITH ( DATA_COMPRESSION = ROW )
```

## ➡ 手順 5 新しいファイル グループの追加

- 次に、稼働テーブル用の新しいパーティション（**2013 年**のデータ格納用）のために、新しいファイル グループを「fg5」という名前で追加します。これは、**ALTER DATABASE** ステートメントを次のように実行します。

```
USE master
go
ALTER DATABASE pTestDB
ADD FILEGROUP fg5
go
ALTER DATABASE pTestDB
ADD FILE ( NAME = 'fg5'
, FILENAME = 'C:\test\fg5.ndf', SIZE = 5MB)
```

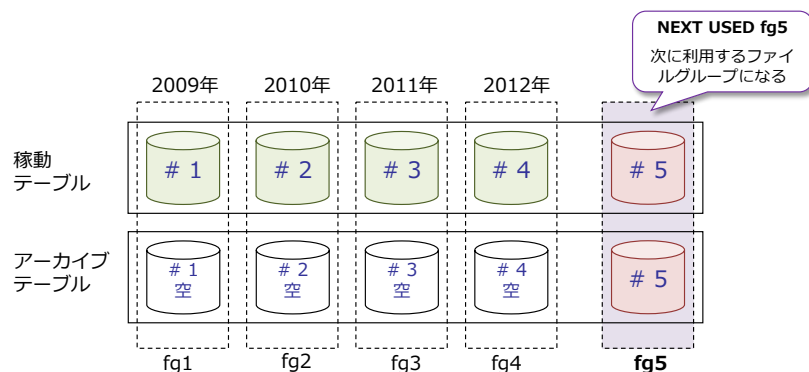
## TO FILEGROUP fg5

## ➡ 手順 6 稼動とアーカイブのパーティション構成の変更 (NEXT USED の指定)

6. 前の手順で追加したファイル グループをパーティションとして使用するには、パーティション構成 (pScheme1 と pScheme1Arch) を変更します。これは、**ALTER PARTITION SCHEME** ステートメントで **NEXT USED** を利用します。

```
USE pTestDB
go
ALTER PARTITION SCHEME pScheme1
NEXT USED fg5
go
ALTER PARTITION SCHEME pScheme1Arch
NEXT USED fg5
go
```

このように NEXT USED を利用すると、パーティションが "次に" 使用するファイル グループを指定することができます。



新しいファイル グループは、のちのアーカイブ テーブルへの移動用としても利用するので、アーカイブ側のパーティション構成についても同様に変更しておく必要があります。

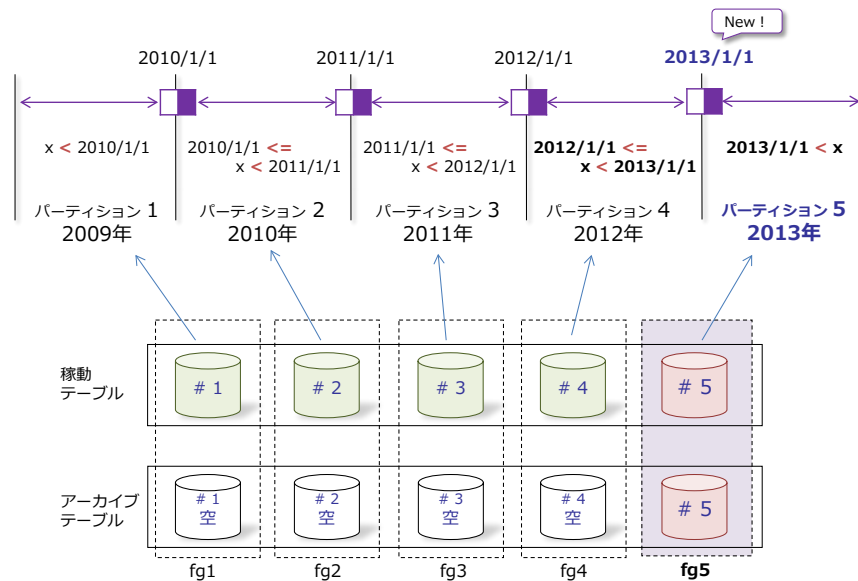
## ➡ 手順 7 稼動テーブルとアーカイブのパーティション関数へ範囲を追加 (SPLIT)

7. 次に、新しいパーティション (2013 年のデータ格納用) を作成するために、パーティション関数へ境界を追加します。境界を追加するには、**ALTER PARTITION FUNCTION** ステートメントで、**SPLIT RANGE** を利用します。

```
USE pTestDB
go
ALTER PARTITION FUNCTION pFunc1 ()
SPLIT RANGE ('2013/01/01')
go
```

```
ALTER PARTITION FUNCTION pFunc1Arch()
  SPLIT RANGE ('2013/01/01')
go
```

このように SPLIT RANGE を利用すると、パーティション関数の範囲を分割できるようになります。'2013/01/01' という境界値を指定することで、パーティションは、次の構成（5つのパーティション）へ変わります。



## ➡ 作成されたパーティション関数の名前と境界の確認

8. SPLIT によってパーティションが分割されたことを確認するには、次のように **partition\_functions** と **partition\_range\_values** カタログ ビューを JOIN します。

```
USE pTestDB
go
SELECT f.name, r.value, *
FROM sys.partition_range_values r
  INNER JOIN sys.partition_functions f
    ON r.function_id = f.function_id
```

```
USE pTestDB
go
SELECT f.name, r.value, *
FROM sys.partition_range_values r
  INNER JOIN sys.partition_functions f
    ON r.function_id = f.function_id
```

	name	value	function_id	boundary_id	parameter_id
1	pFunc1	2010-01-01 00:00:00.000	65536	1	1
2	pFunc1	2011-01-01 00:00:00.000	65536	2	1
3	pFunc1	2012-01-01 00:00:00.000	65536	3	1
4	pFunc1	2013-01-01 00:00:00.000	65536	4	1
5	pFunc1Arch	2010-01-01 00:00:00.000	65537	1	1
6	pFunc1Arch	2011-01-01 00:00:00.000	65537	2	1
7	pFunc1Arch	2012-01-01 00:00:00.000	65537	3	1
8	pFunc1Arch	2013-01-01 00:00:00.000	65537	4	1

パーティション関数の名前と境界値を取得できたことを確認できます。

## ➡ パーティション番号とファイル グループの対応を確認

9. パーティション番号とファイル グループの対応を確認するには、**partition\_schemes** と **data\_spaces**、**destination\_data\_spaces** の 3 つのカatalog ビューを JOIN して、次のように記述します。

```
USE pTestDB
go
SELECT
    ps.name As [パーティション構成名]
    , ds.name As [ファイルグループ名]
    , dds.destination_id As [パーティション番号]
    , *
FROM sys.destination_data_spaces dds
    INNER JOIN sys.partition_schemes ps
        ON dds.partition_scheme_id = ps.data_space_id
    INNER JOIN sys.data_spaces ds
        ON dds.data_space_id = ds.data_space_id
ORDER BY partition_scheme_id
```

```
USE pTestDB
go
SELECT
    ps.name As [パーティション構成名]
    , ds.name As [ファイルグループ名]
    , dds.destination_id As [パーティション番号]
    , *
FROM sys.destination_data_spaces dds
    INNER JOIN sys.partition_schemes ps
        ON dds.partition_scheme_id = ps.data_space_id
    INNER JOIN sys.data_spaces ds
        ON dds.data_space_id = ds.data_space_id
ORDER BY partition_scheme_id
```

	パーティション構成名	ファイルグループ名	パーティション番号	partition_scheme_id	destination_id	data_space_id
1	pScheme1	fg1	1	65601	1	2
2	pScheme1	fg2	2	65601	2	3
3	pScheme1	fg3	3	65601	3	4
4	pScheme1	fg4	4	65601	4	5
5	pScheme1	fg5	5	65601	5	6
6	pScheme1Arch	fg1	1	65602	1	2
7	pScheme1Arch	fg2	2	65602	2	3
8	pScheme1Arch	fg3	3	65602	3	4
9	pScheme1Arch	fg4	4	65602	4	5
10	pScheme1Arch	fg5	5	65602	5	6

## ➡ パーティション 4 とパーティション 5 ヘデータを追加

10. 次に、パーティション分割したパーティション 4 (2012 年用) とパーティション 5 (2013 年用) ヘデータを追加して、パーティションが正しく設定されたことを確認してみましょう。

```
-- データの追加
INSERT INTO t1 VALUES ('2012/12/31')
INSERT INTO t1 VALUES ('2013/01/01')
```



-- どのパーティション内のデータか確認

```
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1
```

-- データの追加

```
INSERT INTO t1 VALUES ('2012/12/31')
```

```
INSERT INTO t1 VALUES ('2013/01/01')
```

-- どのパーティション内のデータか確認

```
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1
```

100 %

結果 メッセージ

	id	col1	パーティション番号
1	1	2009-10-15 00:00:00.000	1
2	2	2010-01-01 00:00:00.000	2
3	3	2010-05-22 00:00:00.000	2
4	4	2010-12-31 00:00:00.000	2
5	5	2011-01-01 00:00:00.000	3
6	6	2011-12-31 00:00:00.000	3
7	7	2012-01-01 00:00:00.000	4
8	8	2012-12-31 00:00:00.000	4
9	9	2013-01-01 00:00:00.000	5

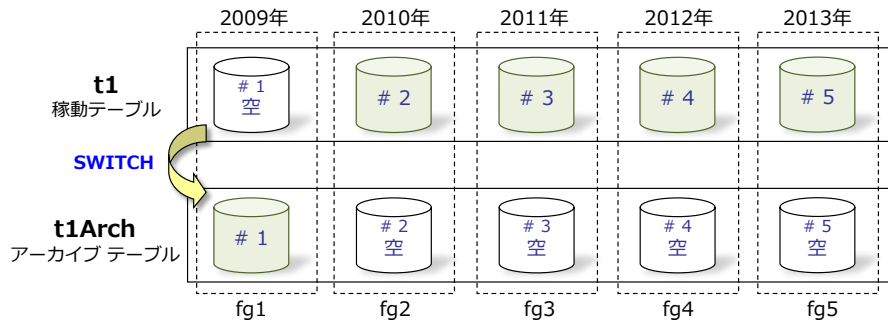
追加したデータ

2012 年のデータがパーティション 4 へ、2013 年のデータがパーティション 5 へ格納され、新しいパーティションが正しく動作していることを確認できます。

## 4.3 パーティションのスイッチ： SWITCH

### ➡ パーティションのスイッチ

次に、稼動テーブル「**t1**」のパーティション「**1**」（2009 年用）を、アーカイブ テーブル「**t1Arch**」のパーティション「**1**」へ SWITCH してみましょう。



1. パーティションをスイッチするには、**ALTER TABLE .. SWITCH** を利用して、次のように記述します。

```
USE pTestDB
go
ALTER TABLE t1
    SWITCH PARTITION 1 TO t1arch PARTITION 1
```

これにより、稼動テーブルのパーティション 1 のデータをすべて、アーカイブ テーブルのパーティション 1 へ移動することができます（内部的には、ポインターの変更のみで完了するので、操作は一瞬で完了します）。

2. スイッチが完了したら、データが移動されたことを確認してみましょう。

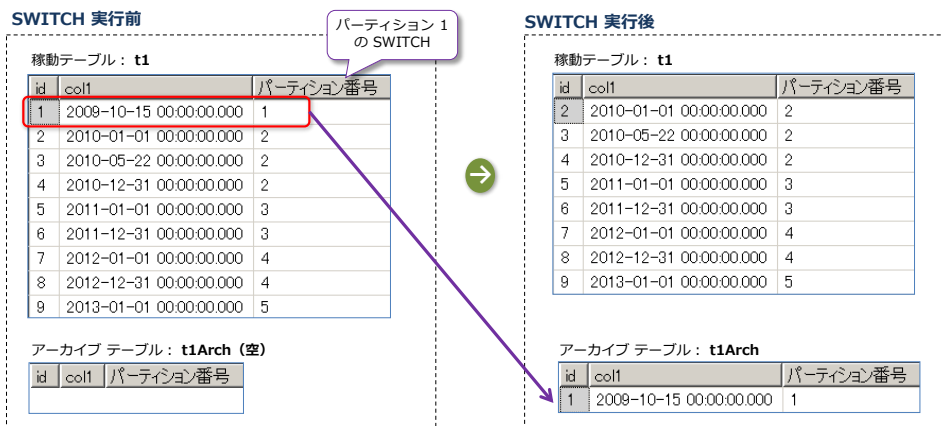
```
SELECT * FROM t1arch
```

	id	col1
1	1	2009-10-15 00:00:00.000

3. 次に、\$PARTITION 関数を利用して、パーティション番号を確認しておきましょう。

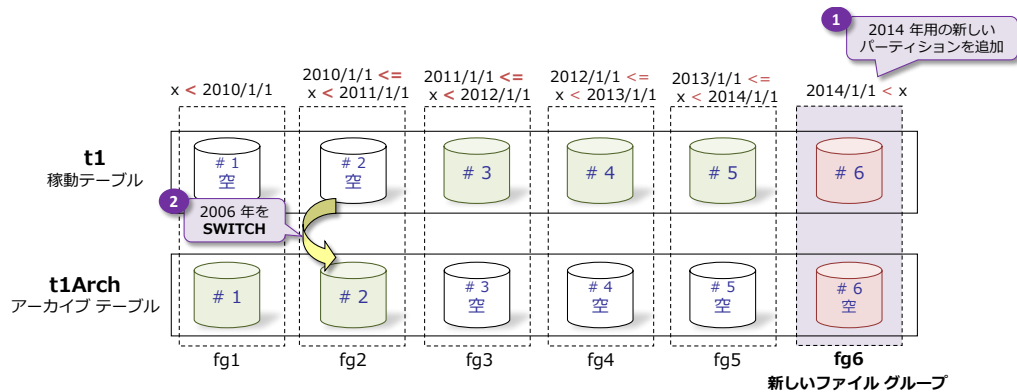
```
-- 稼動テーブル: t1
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1

-- アーカイブテーブル: t1Arch
SELECT *, $PARTITION.pFunc1Arch(col1) As [パーティション番号]
FROM t1Arch
```



## ◆ 新しいパーティションの追加とパーティション 2 のスイッチ

次に、もう一度パーティションのスイッチを試すために、新しいパーティション（**2014 年**）用のファイル グループとして「**fg6**」を追加して、その後、稼動テーブルのパーティション 2 をアーカイブ テーブルのパーティション 2 へスイッチしてみましょう。



1. 操作手順は、パーティション 1 をスイッチしたときと同じで、次のように実行します。

```
-- ファイル グループ「fg6」の追加
USE master
go
ALTER DATABASE pTestDB
ADD FILEGROUP fg6
go
ALTER DATABASE pTestDB
ADD FILE ( NAME = 'fg6'
, FILENAME = 'C:\test\fg6.ndf', SIZE = 5MB)
TO FILEGROUP fg6
go

-- パーティション構成の更新 (NEXT USED fg6)
USE pTestDB
go
ALTER PARTITION SCHEME pScheme1
NEXT USED fg6
```

```

go
ALTER PARTITION SCHEME pScheme1Arch
NEXT USED fg6
go

-- パーティション関数の範囲の変更 (SPLIT RANGE)
ALTER PARTITION FUNCTION pFunc1 ()
SPLIT RANGE ('2014/01/01')
go
ALTER PARTITION FUNCTION pFunc1Arch ()
SPLIT RANGE ('2014/01/01')
go

-- データの追加。2013年と 2014年のデータ
INSERT INTO t1 VALUES ('2013/12/31')
INSERT INTO t1 VALUES ('2014/01/01')

-- どのパーティションに格納されたかを確認
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1

/*-----
パーティション2 のSWITCH
-----*/

ALTER TABLE t1
SWITCH PARTITION 2 TO t1Arch PARTITION 2

-- アーカイブ テーブルヘデータが移動されたことを確認
SELECT * FROM t1Arch

-- 稼動テーブル: どのパーティション内のデータか確認
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1

-- アーカイブ テーブル: どのパーティション内のデータか確認
SELECT *, $PARTITION.pFunc1Arch(col1) As [パーティション番号]
FROM t1Arch

```

SWITCH 実行前

稼動テーブル: t1

id	col1	パーティション番号
2	2010-01-01 00:00:00.000	2
3	2010-05-22 00:00:00.000	2
4	2010-12-31 00:00:00.000	2
5	2011-01-01 00:00:00.000	3
6	2011-12-31 00:00:00.000	3
7	2012-01-01 00:00:00.000	4
8	2012-12-31 00:00:00.000	4
9	2013-01-01 00:00:00.000	5
10	2013-12-31 00:00:00.000	5
11	2014-01-01 00:00:00.000	6

アーカイブ テーブル: t1Arch (空)

id	col1	パーティション番号
1	2009-10-15 00:00:00.000	1

パーティション 2  
の SWITCH

SWITCH 実行後

稼動テーブル: t1

id	col1	パーティション番号
5	2011-01-01 00:00:00.000	3
6	2011-12-31 00:00:00.000	3
7	2012-01-01 00:00:00.000	4
8	2012-12-31 00:00:00.000	4
9	2013-01-01 00:00:00.000	5
10	2013-12-31 00:00:00.000	5
11	2014-01-01 00:00:00.000	6

アーカイブ テーブル: t1Arch

id	col1	パーティション番号
1	2009-10-15 00:00:00.000	1
2	2010-01-01 00:00:00.000	2
3	2010-05-22 00:00:00.000	2
4	2010-12-31 00:00:00.000	2

## ◆ SWITCH 済みのパーティションへ CHECK 制約の追加

ここまでの手順だけでは、稼動テーブルの SWITCH 済みのパーティションへも依然としてデータが追加できてしまうことに注意する必要があります。では、これを試してみましょう。

1. 稼動テーブル「t1」の SWITCH 済みのパーティションである **1** (2009 年) と **2** (2010 年) に対して、データを追加してみましょう。

```
-- パーティション 1 ヘデータ追加
INSERT INTO t1 VALUES ('2009/08/08')

-- パーティション 2 ヘデータ追加
INSERT INTO t1 VALUES ('2010/08/08')

-- 追加されたデータの確認
SELECT * FROM t1
```

```
-- パーティション 1 ヘデータ追加
INSERT INTO t1 VALUES ('2009/08/08')

-- パーティション 2 ヘデータ追加
INSERT INTO t1 VALUES ('2010/08/08')

-- 追加されたデータの確認
SELECT * FROM t1
```

	id	col1
1	12	2009-08-08 00:00:00.000
2	13	2010-08-08 00:00:00.000
3	5	2011-01-01 00:00:00.000
4	6	2011-12-31 00:00:00.000
5	7	2012-01-01 00:00:00.000
6	8	2012-12-31 00:00:00.000
7	9	2013-01-01 00:00:00.000
8	10	2013-12-31 00:00:00.000
9	11	2014-01-01 00:00:00.000

2009 年と 2010 年のデータが追加できてしまったことを確認できます。この状況を回避するには、稼動テーブル側へ CHECK 制約を追加するようにします。

2. CHECK 制約を追加するには、まず、追加したデータを DELETE しておきます。

```
DELETE FROM t1 WHERE col1 IN('2009/08/08', '2010/08/08')
```

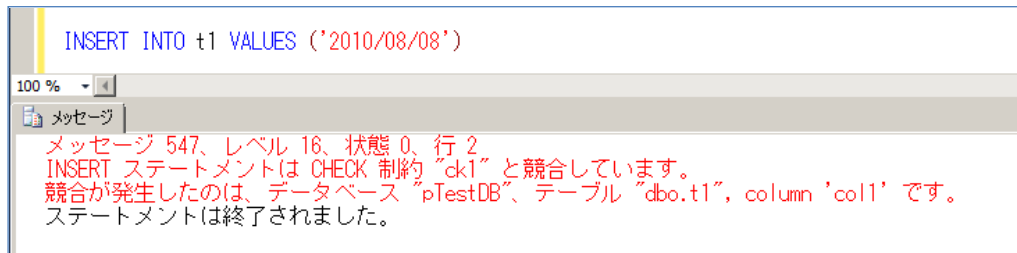
3. 削除後、**ALTER TABLE .. ADD CONSTRAINT** を利用して、CHECK 制約を追加します。

```
ALTER TABLE t1
ADD CONSTRAINT ck1 CHECK(col1 >= '2011/01/01')
```

これで、**t1** テーブルへは、2011/1/1 以上のデータしか追加できないように設定できました。

4. これを確認するために、もう一度 2010 年のデータを追加してみましょう。

```
INSERT INTO t1 VALUES ('2010/08/08')
```

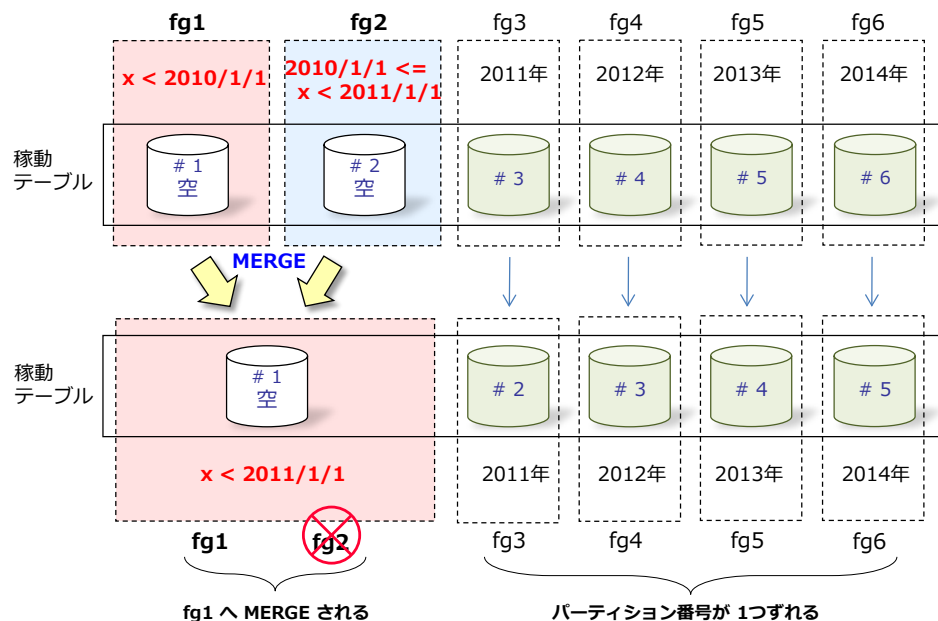


今度は、制約違反エラーが発生して、データを追加できなかったことを確認できます。このように、パーティションの SWITCH 後は、SWITCH 済みの範囲へ、データが追加されないように、CHECK 制約を設定しておくようにしましょう。

## 4.4 古いパーティションの MERGE

### ➡ 古いパーティションの MERGE

古いパーティションは、**MERGE** 操作によってマージ（結合）することができます。たとえば、次のように稼働テーブルの SWITCH 済みのパーティションである 1 と 2 を MERGE して、1 つのパーティションにすることができます



### ➡ Let's Try

それでは、これを試してみましょう。

1. MERGE は、**ALTER PARTITION FUNCTION** ステートメントを利用して、マージしたい境界値を指定することで行えます。パーティション「1」とパーティション「2」をマージするには、次のようにその境界値である「**2010/01/01**」を指定します。

```
ALTER PARTITION FUNCTION pFunc1 ()
MERGE RANGE (' 2010/01/01')
```

2. MERGE 後のパーティション関数の境界値を確認するには、前の Step で利用した **partition\_functions** と **partition\_range\_values** カタログ ビューを JOIN するクエリを利用します。

```
SELECT f.name, r.value, *
FROM sys.partition_range_values r
INNER JOIN sys.partition_functions f
ON r.function_id = f.function_id
```

2011/01/01 になっている

name	value	function_id	boundary_id	parameter_id
pFunc1	2011-01-01 00:00:00.000	65536	1	1
pFunc1	2012-01-01 00:00:00.000	65536	2	1
pFunc1	2013-01-01 00:00:00.000	65536	3	1
pFunc1	2014-01-01 00:00:00.000	65536	4	1
pFunc1Arch	2010-01-01 00:00:00.000	65537	1	1
pFunc1Arch	2011-01-01 00:00:00.000	65537	2	1
pFunc1Arch	2012-01-01 00:00:00.000	65537	3	1
pFunc1Arch	2013-01-01 00:00:00.000	65537	4	1
pFunc1Arch	2014-01-01 00:00:00.000	65537	5	1

稼動テーブル用のパーティション関数（**pFunc1**）から境界値（value）の「**2010/01/01**」が消えていることを確認できます。これにより、パーティション 1 の範囲が「**< 2011/01/01**」へと変更されて、以前のパーティション 2 とマージされています。

3. MERGE 後のパーティションとファイル グループの対応を確認するには、前の Step で利用した **partition\_schemes** と **data\_spaces**、**destination\_data\_spaces** カタログ ビューを JOIN するクエリを利用します。

```
SELECT
    ps.name As [パーティション構成名]
    , ds.name As [ファイルグループ名]
    , dds.destination_id As [パーティション番号]
    , *
FROM sys.destination_data_spaces dds
    INNER JOIN sys.partition_schemes ps
        ON dds.partition_scheme_id = ps.data_space_id
    INNER JOIN sys.data_spaces ds
        ON dds.data_space_id = ds.data_space_id
ORDER BY partition_scheme_id
```

パーティション構成名	ファイルグループ名	パーティション番号	partition_id	data_space_id
pScheme1	fg1	1	65601	2
pScheme1	fg3	2	65601	4
pScheme1	fg4	3	65601	5
pScheme1	fg5	4	65601	6
pScheme1	fg6	5	65601	7
pScheme1Arch	fg1	1	65602	2
pScheme1Arch	fg2	2	65602	3
pScheme1Arch	fg3	3	65602	4
pScheme1Arch	fg4	4	65602	5
pScheme1Arch	fg5	5	65602	6
pScheme1Arch	fg6	6	65602	7

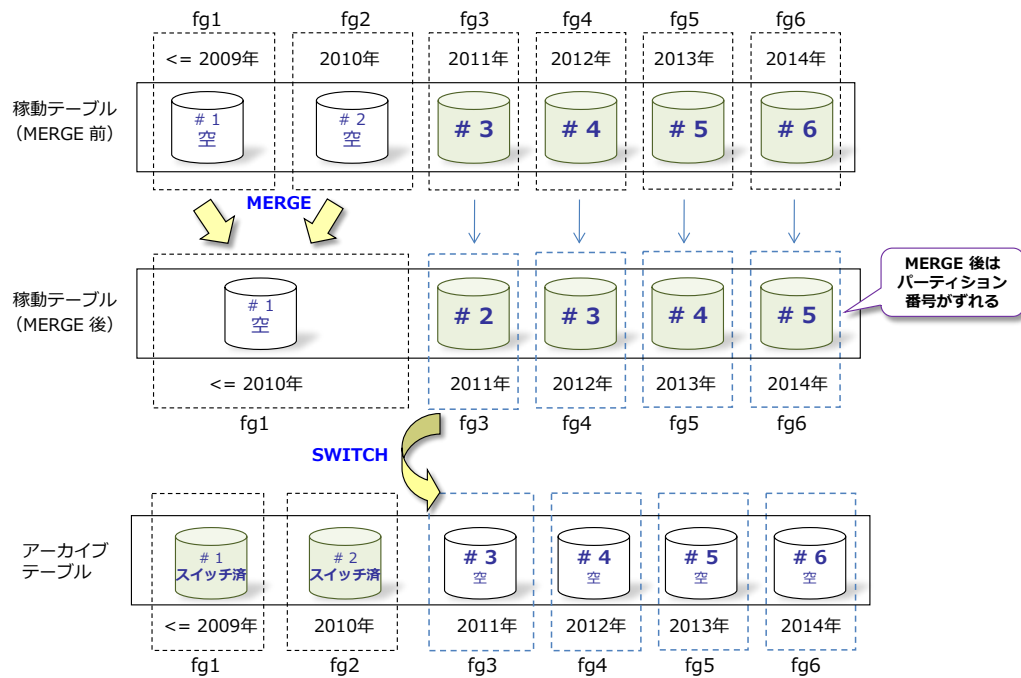
稼動テーブル用のパーティション構成（**pScheme1**）は、パーティション番号が **1～5** となり、パーティション数が 6 から 5 つへ減っていることを確認できます。パーティション 1 は、以前のパーティション 1 と 2 をマージしたもの、パーティション 2～5 は、以前のパーティション 3～6 のパーティション番号が 1 つずつ前へずれたものです。

また、稼動テーブルでは、「**fg2**」ファイル グループが使われなくなっていることにも注目してください。これについては、次の Step 4.5「**MERGE の注意点**」で詳しく説明します。



## MERGE した場合のパーティション スイッチ

稼動テーブルの古いパーティションを MERGE した場合は、パーティション番号がずれていることに注意する必要があります。以前のパーティション 3 以降は、次のようにパーティション番号が 1 つずつずれています。



このように、MERGE 後は、パーティション番号がずれるので、パーティションの SWITCH 時には番号を間違えないように注意する必要があります。では、これを試してみましょう。

1. 以前のパーティション 3 (現在のパーティション 2) を、アーカイブ テーブルへスイッチしてみましょう。

```
ALTER TABLE t1
  SWITCH PARTITION 2 TO t1arch PARTITION 3
```

```
ALTER TABLE t1
  SWITCH PARTITION 2 TO t1arch PARTITION 3
```

100 %

メッセージ

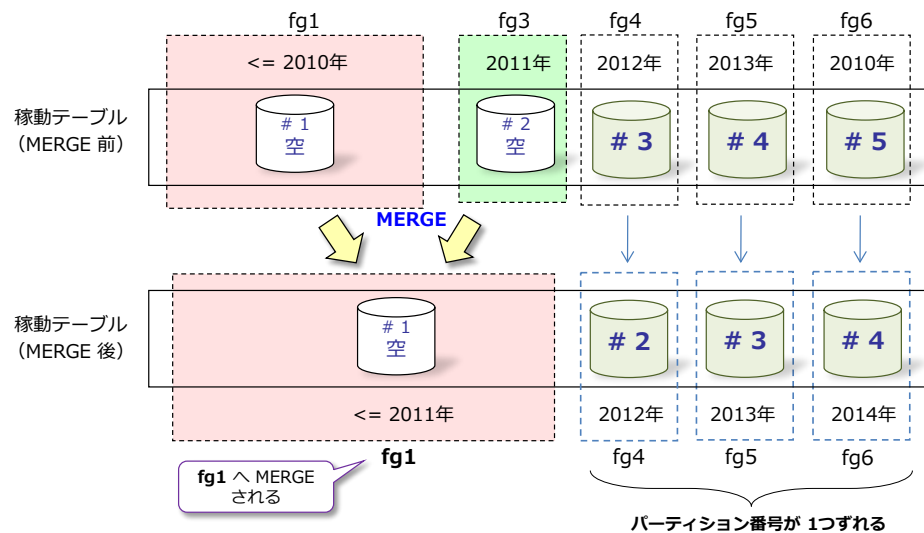
コマンドは正常に完了しました。

パーティションに "2" を指定している点に注意してください。

2. SWITCH 後は、パーティション 2 と 1 を MERGE しておきましょう。

```
ALTER PARTITION FUNCTION pFunc1 ()
  MERGE RANGE ('2011/01/01')
```

MERGE 後のパーティション構成は、次のようになります。



MERGE は、またパーティション番号がずれることに注意する必要があります。このように SWITCH 済みの古いパーティションを常に MERGE している環境では、稼働テーブルの SWITCH 元となるパーティションは常に "2" になります。

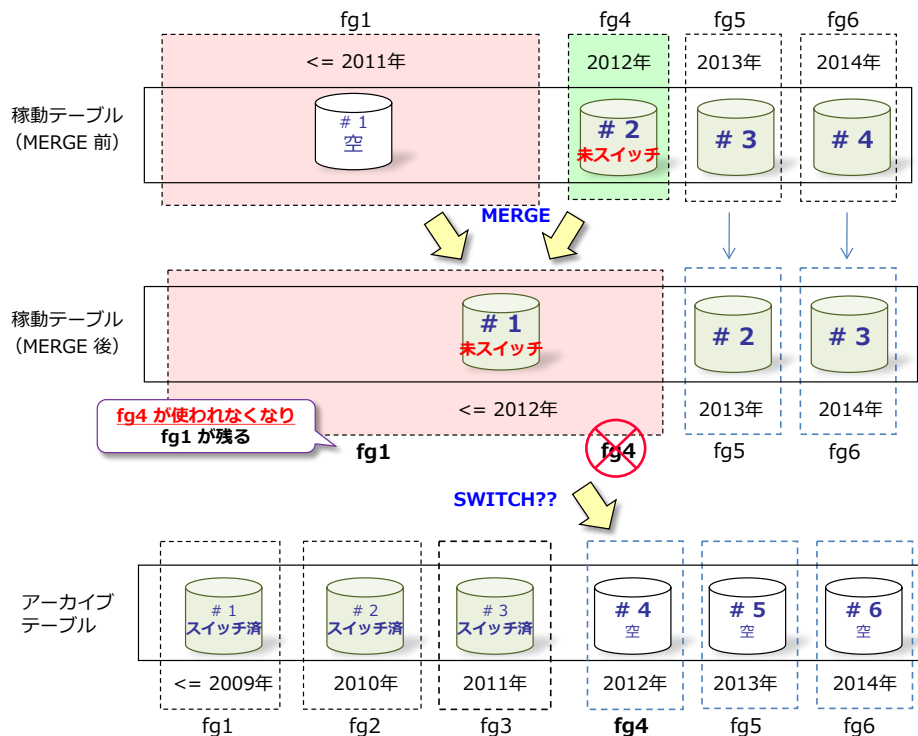
したがって、以降のパーティションの SWITCH は、次のように稼働テーブル側は「2」が固定で、アーカイブ テーブル側の SWITCH 先のパーティションが 1 つずつインクリメントしていくことになります。

```
ALTER TABLE t1
  SWITCH PARTITION 2 TO t1arch PARTITION n
```

## 4.5 MERGE の注意点

### ➡ MERGE の注意点

パーティションの MERGE は、SWITCH 済みのパーティション同士で行う必要があります。SWITCH をしていないパーティションを MERGE することも可能ですが、これをしてしまうと、そのパーティションが SWITCH できなくなってしまう。これは、次のような状況です。



この図では、まだスイッチしていないパーティション 2 (fg4 ファイル グループを利用) を MERGE してから、アーカイブ テーブルへ SWITCH しようとしています。これは、ファイル グループが fg1 と fg4 で異なるので、SWITCH が失敗します。SWITCH 操作は、同じファイル グループ内でのみ実行できないという制限があるためです。

### ➡ Let's Try

それでは、これを試してみましょう。

1. まずは、2012 年のパーティション 2 を、パーティション 1 と MERGE します。

```
ALTER PARTITION FUNCTION pFunc1 ()
MERGE RANGE ('2012/01/01')
```

パーティション 1 と 2 の境界値の 2012/01/01 を指定することで、1 と 2 を MERGE することができます。

2. MERGE 後のパーティションとファイル グループの対応を確認するために、前の Step で利

用した **partition\_schemes** と **data\_spaces**、**destination\_data\_spaces** カタログ ビューを JOIN するクエリを利用します。

```
SELECT
    ps.name As [パーティション構成名]
    , ds.name As [ファイルグループ名]
    , dds.destination_id As [パーティション番号]
    , *
FROM sys.destination_data_spaces dds
    INNER JOIN sys.partition_schemes ps
        ON dds.partition_scheme_id = ps.data_space_id
    INNER JOIN sys.data_spaces ds
        ON dds.data_space_id = ds.data_space_id
ORDER BY partition_scheme_id
```

	パーティション構成名	ファイルグループ名	パーティション番号	ps.partition_id	ps.data_space_id	dds.destination_id	data_space_id
1	pScheme1	fg1	1	65600	2	2	2
2	pScheme1	fg5	2	65601	2	6	6
3	pScheme1	fg6	3	65601	3	7	7
4	pScheme1Arch	fg1	1	65602	1	2	2
5	pScheme1Arch	fg2	2	65602	2	3	3
6	pScheme1Arch	fg3	3	65602	3	4	4
7	pScheme1Arch	fg4	4	65602	4	5	5
8	pScheme1Arch	fg5	5	65602	5	6	6
9	pScheme1Arch	fg6	6	65602	6	7	7

パーティション番号「1」のファイル グループが「**fg4**」になっていることに注目します。MERGE では、範囲（境界値）は右側とマージされますが、ファイル グループに関しては、左側とマージされてしまうのです。なお、この動作は、パーティション関数の RANGE が RIGHT 指定か LEFT 指定かで逆になり、今回は RIGHT 指定の場合の動作です。

- この状態で、MERGE したパーティション 1（末スイッチの 2012 年のデータ）を、アーカイブ側のパーティション 4（2012 年用）へ SWITCH してみましょう。

```
ALTER TABLE t1
    SWITCH PARTITION 1 TO t1arch PARTITION 4
```

```
ALTER TABLE t1
    SWITCH PARTITION 1 TO t1arch PARTITION 4
```

メッセージ 4938、レベル 16、状態 1、行 2  
 ALTER TABLE SWITCH ステートメントが失敗しました。  
 インデックス 'pTestDB.dbo.t1.index\_col1' のパーティション 1 はファイル グループ 'fg1' に所属していますが、  
 インデックス 'pTestDB.dbo.t1arch.index\_col1' のパーティション 4 はファイル グループ 'fg4' に所属しています。

結果は、「ファイル グループが異なるので SWITCH できない」という主旨のエラーが発生します。SWITCH 操作は、同じファイル グループに属したパーティション同士で行わなければならないという制限があるためです。しかし、SWITCH 元のパーティションは、MERGE 操作によって、ファイル グループが **fg4** から **fg1** へ変わってしまっているため、アーカイブ先 (**fg4**) とは異なるファイル グループになってしまっています。これでは、SWITCH 操作を実行することはできません。

**Point : MERGE 操作は SWITCH 済みのパーティション同士でのみ実行すること**

以上のように、まだ SWITCH していないパーティションを MERGE してしまうと、そのパーティションが SWITCH できなくなってしまう。MERGE 操作は、必ず SWITCH 済みのパーティション同士でのみ行うことに注意してください。これにより、稼働テーブルには、一番左側へ空のパーティションが 1 つ残ることになりますが、これは正しい実装です。この空のパーティションへデータが格納されないようにするには、前述したように、テーブルへ CHECK 制約を設定するようにします

## 4.6 SWITCH 操作の条件

### ➡ SWITCH 操作の条件

SWITCH 操作は、内部的にはポインターの付け替えのみで、瞬間的に完了する処理なので、実行には、いくつかの条件があります（もちろん、この Step で説明したとおりに実装しておけば、この条件へ引っかかることはありません）。

SWITCH の条件は、次のとおりです。

- SWITCH 操作を行うには、**同じファイル グループ**に属したパーティション同士でなければなりません。同じファイル グループに属していない場合は、前の Step 4.5 で試したように、次のエラーが通達されます。

メッセージ 4938、レベル 16、状態 1、行 2  
 ALTER TABLE SWITCH ステートメントが失敗しました。  
 インデックス 'pTestDB.dbo.t1.index\_col1' のパーティション1 は  
 ファイルグループ 'fg1' に所属していますが、  
 インデックス 'pTestDB.dbo.t1arch.index\_col1' のパーティション4 は  
 ファイルグループ 'fg4' に所属しています。

- SWITCH 元と SWITCH 先では、**同じデータ圧縮の設定**を行っている必要があります。同じデータ圧縮の設定でない場合は、前の Step 4.5 で試したように、次のエラーが通達されます。

メッセージ 11406、レベル 16、状態 1、行 2  
 ALTER TABLE SWITCH ステートメントが失敗しました。  
 切り替え元のパーティションと切り替え先のパーティションで  
**DATA\_COMPRESSION** オプションの値が異なります。

- SWITCH 先となるパーティションは、「空」である必要があります。パーティションが空でない場合は、次のエラーが通達されます。

メッセージ 4904、レベル 16、状態 1、行 1  
 ALTER TABLE SWITCH ステートメントが失敗しました。  
 切り替え先テーブル 'pTestDB.dbo.t1arch' の指定された**パーティション 1** を空にしてください。

- ターゲット テーブル (SWITCH 先となるテーブル) へ非クラスター化インデックスが作成されている場合は、ソース テーブル (SWITCH 元となるテーブル) にも**同じインデックス**が作成されている必要があります。

同じインデックスが作成されていない場合は、次のエラーが通達されます。

メッセージ 4947、レベル 16、状態 1、行 1

ALTER TABLE SWITCH ステートメントが失敗しました。切り替え元テーブル 'pTestDB.dbo.t1' には、切り替え先テーブル 'pTestDB.dbo.t1arch' のインデックス 'idx\_col1' と**同一のインデックスがありません**。

- ターゲット テーブルへ CHECK 制約を定義する場合は、パーティション関数によって定義された範囲を満たすようにするか、ソース テーブルにも**まったく同じ制約**またはターゲットの制約の**サブセットとなるような制約**（ターゲットの CHECK 制約が col < '2011/01/01' なら、ソースは col < '2010/01/01' とするなど）を作成しておく必要があります。

この条件を満たしていない場合は、次のエラーが通達されます。

**メッセージ 4972、レベル 16、状態 1、行 2**

ALTER TABLE SWITCH ステートメントが失敗しました。切り替え元テーブル 'pTestDB.dbo.t1' のチェック制約またはパーティション関数で許可される値が、切り替え先テーブル 'pTestDB.dbo.t1arch' の**チェック制約またはパーティション関数では許可されません**。

上記のほかにも、SWITCH 操作には、いくつか細かい条件があります。これらについては、オンライン ブックの以下のトピックを参考にしてください。

パーティションの切り替えを使用した効率的なデータの転送

<http://msdn.microsoft.com/ja-jp/library/ms191160.aspx>

## STEP 5. 一部のデータ ファイル破損時の 復旧手順

---

この STEP では、一部のデータ ファイルが破損した場合の復旧手順について説明します。

この STEP では、次のことを学習します。

- ✓ ファイル グループ単位のバックアップ
- ✓ 障害のシミュレート
- ✓ 一部のデータ ファイル破損時の復旧手順



## 5.1 一部のデータ ファイル破損時の復旧

### ➡ 一部のデータ ファイル破損時の復旧

データ パーティションは、複数のファイル グループおよびデータ ファイル（.ndf）で構成している場合は、一部のデータ ファイルが破損してもそのまま稼働させることができます。また、破損したデータ ファイル（ファイル グループ）のバックアップを復元するだけで、破損前と同じ状態へ復旧することができます。

このときの作業の流れは、次のとおりです。

1. 事前に**ファイル グループ単位のバックアップ**またはデータベース全体のフル バックアップを実行しておく
2. 一部のデータ ファイルに障害が発生した場合は、**SQL Server のログを確認**して、障害が発生したデータ ファイル（ファイル グループ）の名前を確認する
3. 手順 2 で確認したファイル グループを **OFFLINE** にする
4. データベースを **ONLINE** にする
5. **ログ末尾**（TAIL ログ）をバックアップする
6. 手順 1 でバックアップしたファイルから、障害が発生したファイル グループのみを復元する
7. 手順 5 でバックアップしたログ末尾を復元する

### ➡ Let's Try

それでは、これを試してみましょう。前の Step で利用した「**t1**」テーブルが格納されている「**pTestDB**」データベースを利用して、一部のデータ ファイルが破損したときの復旧手順を試してみましょう。

#### 手順 1 ファイル グループ単位のバックアップの実行

1. 最初の手順は、データベース「**pTestDB**」を構成する 7 つファイル グループ（**PRIMARY**、**fg1** ～ **fg6**）を、それぞれバックアップします。通常の **BACKUP** ステートメントで、次のように **FILEGROUP** キーワードを指定することで、ファイル グループ単位でのバックアップを実行することができます。

```
-- PRIMARY ファイル グループのバックアップ
BACKUP DATABASE pTestDB
FILEGROUP = 'PRIMARY'
TO DISK = 'C:¥test¥fgPrimary.bak' WITH FORMAT

-- fg1 ファイル グループのバックアップ
BACKUP DATABASE pTestDB
```

```

FILEGROUP = 'fg1'
TO DISK = 'C:¥test¥fg1.bak' WITH FORMAT

-- fg2 ファイル グループのバックアップ
BACKUP DATABASE pTestDB
FILEGROUP = 'fg2'
TO DISK = 'C:¥test¥fg2.bak' WITH FORMAT

-- fg3 ファイル グループのバックアップ
BACKUP DATABASE pTestDB
FILEGROUP = 'fg3'
TO DISK = 'C:¥test¥fg3.bak' WITH FORMAT

-- fg4 ファイル グループのバックアップ
BACKUP DATABASE pTestDB
FILEGROUP = 'fg4'
TO DISK = 'C:¥test¥fg4.bak' WITH FORMAT

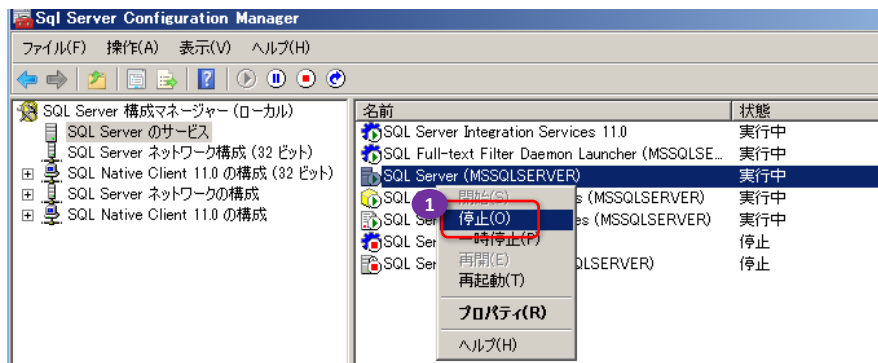
-- fg5 ファイル グループのバックアップ
BACKUP DATABASE pTestDB
FILEGROUP = 'fg5'
TO DISK = 'C:¥test¥fg5.bak' WITH FORMAT

-- fg6 ファイル グループのバックアップ
BACKUP DATABASE pTestDB
FILEGROUP = 'fg6'
TO DISK = 'C:¥test¥fg6.bak' WITH FORMAT

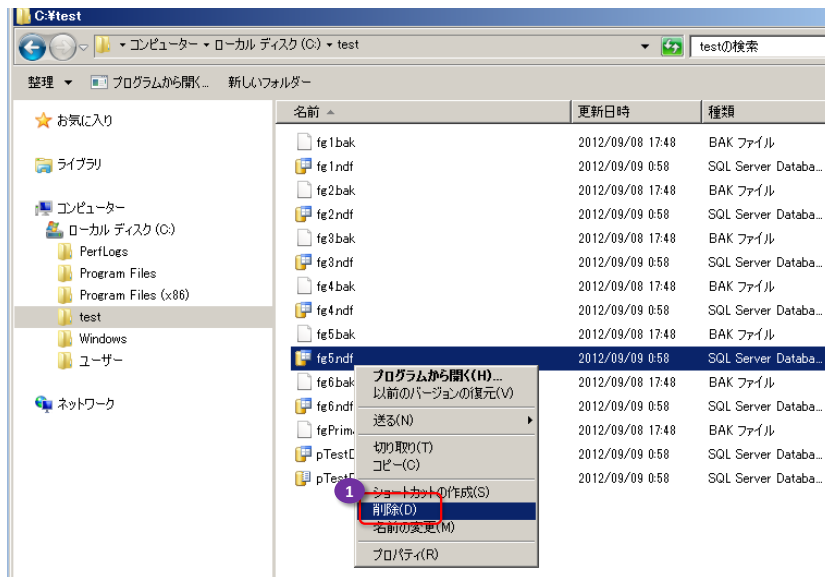
```

## ➡ データ ファイル破損のシミュレーション

- 次に、データ ファイルの破損をシミュレーションするために「**fg5.ndf**」ファイル（**fg5** ファイル グループ）を削除します。ファイルを削除するには、まず、SQL Server サービスを停止しておく必要があります。サービスの停止は、次のように「**SQL Server 構成マネージャー**」ツールから、SQL Server サービスを右クリックして、**[停止]** をクリックします。



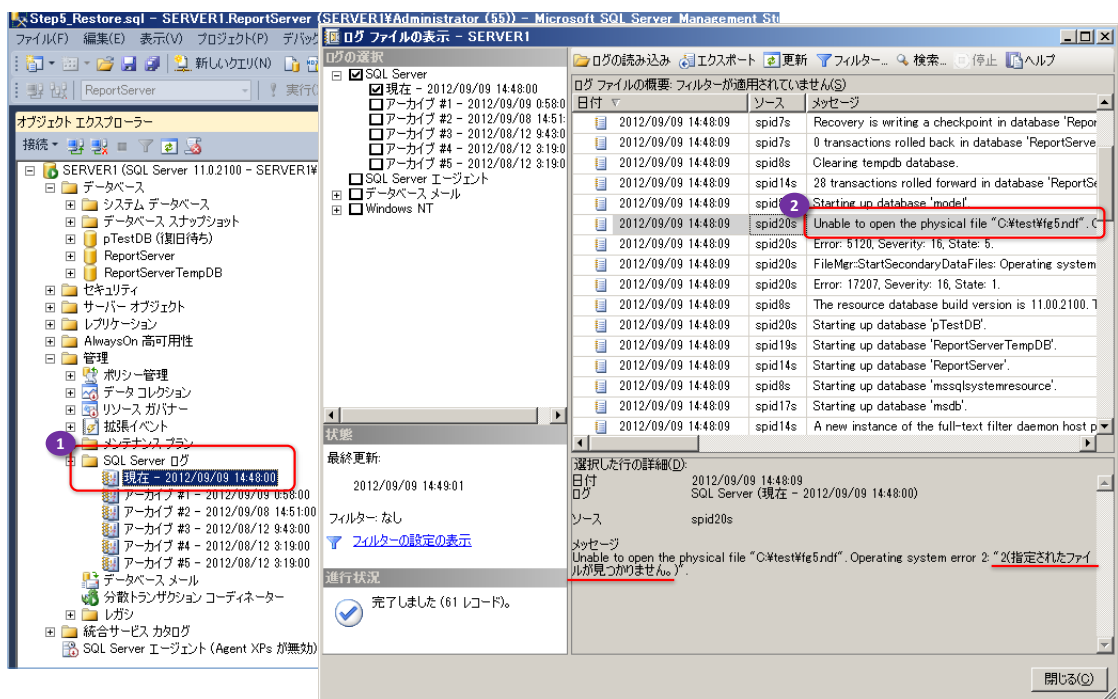
- サービスが停止されたら、**Windows エクスプローラー**を起動して、次のように「**fg5.ndf**」ファイルを右クリックして**[削除]** をクリックし、直接削除します。



4. ファイルの削除が完了したら、「SQL Server 構成マネージャー」ツールから SQL Server サービスを開始します。

## ➡ 手順 2 SQL Server ログから障害が発生したファイルの確認

5. SQL Server サービスが開始されたら、Management Studio を起動して、SQL Server ログを参照します。データ ファイルへ障害が発生している場合は、次のようにエラーが記録されます。



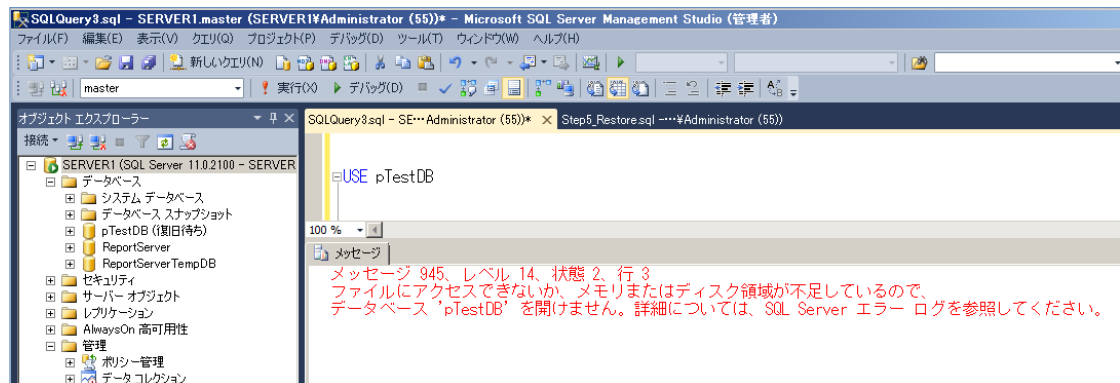
ログへは、次のように記録されます。

Unable to open the physical file "C:\%test%\fg5.ndf".  
Operating system error 2: "2(指定されたファイルが見つかりません)。"

日本語訳だと「物理ファイルが開けません、指定されたファイルが見つかりません」というエラーで、これがファイルに障害が発生した場合に記録されるエラー メッセージです。

6. この状態では、データベースへアクセスすることができないので、これを確認しておきましょう。クエリ エディターを開いて、データベースへ接続してみます。

USE pTestDB



### ➡ 手順 3 破損したファイル グループを OFFLINE へ変更

この状態から復旧するには、まず破損したファイル グループを **OFFLINE** へ設定します。

7. ファイル グループを OFFLINE へ設定するには、ファイル グループの「論理名」を指定する必要がありますので、次のようにバックアップ ファイルに対して **RESTORE FILELISTONLY** ステートメントを実行して、ファイル グループの論理名を取得します。

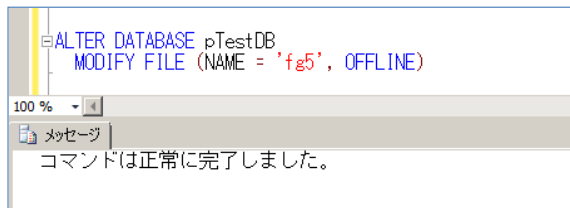
```
RESTORE FILELISTONLY
FROM DISK = 'C:¥test¥fg5.bak'
```

RESTORE FILELISTONLY FROM DISK = 'C:¥test¥fg5.bak'							
100 %							
結果 メッセージ							
	LogicalName	PhysicalName	Type	FileGroupName	Size	MaxSize	FileId
1	pTestDB	C:¥test¥pTestDB.mdf	D	PRIMARY	5242880	35184372080640	1
2	fg1	C:¥test¥fg1.ndf	D	fg1	5242880	35184372080640	3
3	fg2	C:¥test¥fg2.ndf	D	fg2	5242880	35184372080640	4
4	fg3	C:¥test¥fg3.ndf	D	fg3	5242880	35184372080640	5
5	fg4	C:¥test¥fg4.ndf	D	fg4	5242880	35184372080640	6
6	fg5	C:¥test¥fg5.ndf	D	fg5	5242880	35184372080640	7
7	fg6	C:¥test¥fg6.ndf	D	fg6	5242880	35184372080640	8
8	pTestDB_log	C:¥test¥pTestDB_log.LDF	L	NULL	5242880	2199023255552	2

結果の **LogicalName** が論理名です。これにより、破損したファイル (C:¥test¥fg5.ndf) の論理名が「fg5」であることが分かるので、これを OFFLINE へ設定します。

8. OFFLINE へ設定するには、次のように **ALTER DATABASE .. MODIFY FILE** ステートメントで OFFLINE を指定します。

```
ALTER DATABASE pTestDB
  MODIFY FILE (NAME = 'fg5', OFFLINE)
```

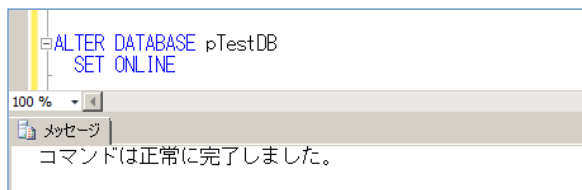


#### ◆ 手順 4 データベースを ONLINE へ変更

破損したファイル グループを OFFLINE にした後は、データベースを **ONLINE** へ設定します。

9. データベースを ONLINE へ設定するには、次のように **ALTER DATABASE** ステートメントを実行します。

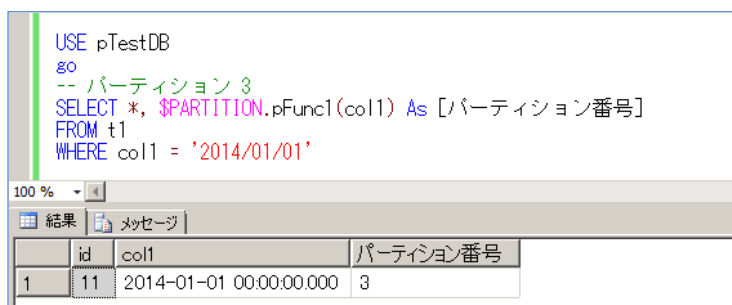
```
ALTER DATABASE pTestDB
  SET ONLINE
```



これにより、データベースが利用できるようになり、破損していないファイル グループへアクセスできるようになります。

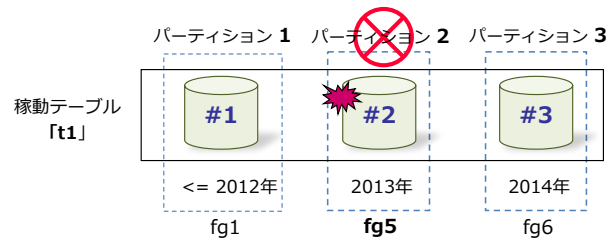
10. データベースが利用できることを確認するために、パーティション 3 (2014 年用、fg6 ファイル グループを利用) のデータを参照してみましょう。

```
USE pTestDB
go
-- パーティション 3
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1
WHERE col1 = '2014/01/01'
```



結果は、問題なくデータを参照することができ、破損したファイルがあるにも関わらず、デー

データベースを利用できたことを確認できます。現在のファイル グループとパーティションの構成は、次のとおりです。



11. 次に、パーティション 3 に対して、更新系のステートメントを実行してみましょう。

```
USE pTestDB
go
-- パーティション 3 ヘーダーの追加、更新、削除
INSERT INTO t1 VALUES('2014/01/02')
UPDATE t1 SET col1='2014/05/02' WHERE col1 = '2014/01/02'
DELETE FROM t1 WHERE col1='2014/01/01'
```

```
USE pTestDB
go
-- パーティション 3 ヘーダーの追加、更新、削除
INSERT INTO t1 VALUES('2014/01/02')
UPDATE t1 SET col1='2014/05/02' WHERE col1 = '2014/01/02'
DELETE FROM t1 WHERE col1='2014/01/01'
```

100 %

メッセージ

(1 行処理されました)

(1 行処理されました)

(1 行処理されました)

これも問題なく実行できたことを確認できます。更新系のステートメントについても、破損したファイル グループを除いて、実行することができます（更新操作は、トランザクション ログへ記録されています）。

12. 次に、破損したファイル グループ（パーティション 2）へアクセスしてみましょう。

```
-- パーティション 2 へのアクセス
SELECT * FROM t1 WHERE col1 = '2013/01/01'
INSERT INTO t1 VALUES('2013/01/02')
```

```
-- パーティション 2 (破損したパーティション) へのアクセス
SELECT * FROM t1 WHERE col1 = '2013/01/01'
INSERT INTO t1 VALUES('2013/01/02')
```

100 %

メッセージ

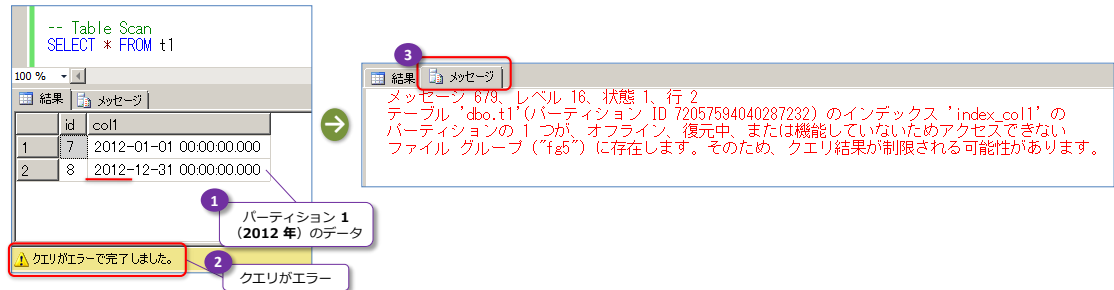
メッセージ 679、レベル 16、状態 1、行 1  
 テーブル 'dbo.t1' (パーティション ID 72057594040287232) のインデックス 'index\_col1' の  
 パーティションの 1 つか、オフライン、復元中、または機能していないためアクセスできない  
 ファイル グループ (fg5) に存在します。そのため、クエリ結果が制限される可能性があります。

結果は、どちらもエラーになります。エラーには、破損したファイル グループの名前「fg5」

があり、これにアクセスできないというメッセージになっていることを確認できます。

13. 次に、テーブル スキャンを実行してみましょう。

```
SELECT * FROM t1
```



結果は、2012 年のデータ（パーティション 1）のデータが表示されて、クエリがエラーで完了します。[メッセージ] タブを開くと、同じエラーが発生していることを確認できます。このように、破損したファイル グループを含んだデータ範囲検索の場合は、そのファイル グループへアクセスするまでの間に取得できたデータを参照することができます。

14. 続いて、データベースの状態を確認するために、次のように **database\_files** カタログ ビューを参照してみましょう。

```
SELECT state_desc, name, *
FROM sys.database_files
```

	state_desc	name	file_id	file_guid	type	type_desc	data_space_id
1	ONLINE	pTestDB	1	627DA986-5055-4821-8AF3-3B66F88AA83A	0	ROWS	1
2	ONLINE	pTestDB_log	2	906D3604-2661-40BB-8D42-F85CE57DD01C	1	LOG	0
3	ONLINE	fg1	3	650048D8-7747-4CD6-90A4-80BF9429E2E7	0	ROWS	2
4	ONLINE	fg2	4	CB253E2E-3002-477C-81B7-1C03C174E569	0	ROWS	3
5	ONLINE	fg3	5	C6D906E6-911C-4721-994A-52628D619954	0	ROWS	4
6	ONLINE	fg4	6	9BE3B26A-D646-41FD-AC3C-DE92B99A3208	0	ROWS	5
7	OFFLINE	fg5	7	C703EFAD-3FE8-4121-B362-B7DF289C5A3D	0	ROWS	6
8	ONLINE	fg6	8	D1AEB38-864D-462A-B387-20242EAB8E5B	0	ROWS	7

結果は、**fg5** のみが OFFLINE であることを確認できます。

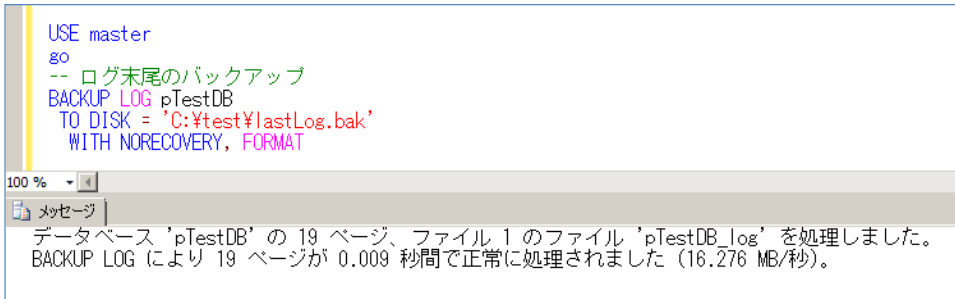
## ➡ 手順 5 ログ末尾（TAIL ログ）のバックアップ

ここからは、破損したファイル グループを復旧するための手順です。

前の手順では、オンラインとなっているファイル グループに対して更新操作を行いました。これらの操作は、**トランザクション ログの末尾**（まだバックアップされていないアクティブ ログで、「TAIL ログ」とも呼ばれます）へ記録されています。

15. ログの末尾をバックアップするには、次のように **BACKUP LOG** ステートメントで **WITH NORECOVERY** を指定して実行します。

```
USE master
go
BACKUP LOG pTestDB
TO DISK = 'C:¥test¥lastLog.bak'
WITH NORECOVERY, FORMAT
```



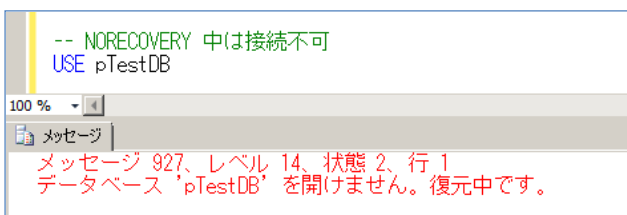
「正常に処理されました」と表示されれば、バックアップが完了です。**NORECOVERY** オプションを指定することで、ログ末尾をバックアップして、かつ「オンライン復元シーケンス」（SQL Server をオンラインにしたままでの復元）を開始することができます。

なお、実行時に「使用中で排他アクセスを獲得できませんでした」エラーが表示される場合には、ログ末尾のバックアップが失敗しています。この場合は、ほかに接続しているユーザーがいないことを確認してから、次のように **ALTER DATABASE** ステートメントを実行して、データベースを **SINGLE\_USER** モードへの切り替えを行ってから、ログ末尾のバックアップを再度実行してみてください。

```
USE master
go
ALTER DATABASE pTestDB
SET SINGLE_USER WITH ROLLBACK IMMEDIATE
go
```

16. オンライン復元シーケンスの開始後は、データベースが **NORECOVERY** 状態（復元中）となり、データベースへのアクセスはできなくなります。

```
USE pTestDB
```



このようにデータベースの復元中は、復旧作業を正しく行うために、データベースへは一切ア



クセスさせないようにして、ユーザーによる更新操作を行わせないように（トランザクションログへ更新操作が記録されないように）しています。

## ➡ 手順 6 障害の発生したファイル グループのみを復元

17. 次に、破損したファイル グループ（fg5）のみをバックアップから復元します。

```
RESTORE DATABASE pTestDB
FILEGROUP = 'fg5'
FROM DISK = 'C:\test\fg5.bak'
WITH NORECOVERY
```

```
-- 破損したファイル グループのみをリストア
RESTORE DATABASE pTestDB
FILEGROUP = 'fg5'
FROM DISK = 'C:\test\fg5.bak'
WITH NORECOVERY
```

100 %

メッセージ

データベース 'pTestDB' の 16 ページ、ファイル 1 のファイル 'fg5' を処理しました。  
データベース 'pTestDB' の 2 ページ、ファイル 1 のファイル 'pTestDB\_log' を処理しました。  
RESTORE DATABASE ... FILE=<name> により 18 ページが 0.017 秒間で正常に処理されました (7.898 MB/秒)。

## ➡ 手順 7 ログ末尾（TAIL ログ）のリストア

18. 最後に、手順 5 で取得したログ末尾のバックアップをリストアします。

```
RESTORE LOG pTestDB
FROM DISK = 'C:\test\lastLog.bak'
WITH RECOVERY
```

```
-- ログ末尾のリストア
RESTORE LOG pTestDB
FROM DISK = 'C:\test\lastLog.bak'
WITH RECOVERY
```

100 %

メッセージ

データベース 'pTestDB' の 0 ページ、ファイル 1 のファイル 'pTestDB' を処理しました。  
データベース 'pTestDB' の 0 ページ、ファイル 1 のファイル 'fg1' を処理しました。  
データベース 'pTestDB' の 0 ページ、ファイル 1 のファイル 'fg2' を処理しました。  
データベース 'pTestDB' の 0 ページ、ファイル 1 のファイル 'fg3' を処理しました。  
データベース 'pTestDB' の 0 ページ、ファイル 1 のファイル 'fg4' を処理しました。  
データベース 'pTestDB' の 0 ページ、ファイル 1 のファイル 'fg5' を処理しました。  
データベース 'pTestDB' の 0 ページ、ファイル 1 のファイル 'fg6' を処理しました。  
データベース 'pTestDB' の 19 ページ、ファイル 1 のファイル 'pTestDB\_log' を処理しました。  
RESTORE LOG により 19 ページが 0.006 秒間で正常に処理されました (24.414 MB/秒)。

19. リストアが完了したら、リストアしたデータを確認してみましょう。

```
USE pTestDB
go
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1
```

```
USE pTestDB
go
SELECT *, $PARTITION.pFunc1(col1) As [パーティション番号]
FROM t1
```

	id	col1	パーティション番号
1	7	2012-01-01 00:00:00.000	1
2	8	2012-12-31 00:00:00.000	1
3	9	2013-01-01 00:00:00.000	2
4	10	2013-12-31 00:00:00.000	2
5	15	2014-05-02 00:00:00.000	3

障害が発生したパーティションのデータが復旧

障害発生後に追加・更新したデータ

障害発生後に、パーティション **3** へ追加したデータ（**2014-05-02**）が正しく格納されて、破損したファイル グループ（パーティション **2**）のデータも復元されていることを確認できます。

20. すべてのファイル グループが ONLINE になっていることも確認しておきましょう。

```
SELECT state_desc, name, *
FROM sys.database_files
```

```
SELECT state_desc, name, *
FROM sys.database_files
```

	state_desc	name	file_id	file_guid	type	type_desc	data_space_id
1	ONLINE	pTestDB	1	627DA986-5055-4821-8AF3-3B66F88AA83A	0	ROWS	1
2	ONLINE	pTestDB_log	2	906D3604-2661-40BB-8D42-F85CE57DD01C	1	LOG	0
3	ONLINE	fg1	3	650048D8-7747-4CD6-90A4-80BF9429E2E7	0	ROWS	2
4	ONLINE	fg2	4	CB253E2E-3002-477C-81B7-1C03C174E569	0	ROWS	3
5	ONLINE	fg3	5	C6D906E8-911C-4721-994A-52628D619954	0	ROWS	4
6	ONLINE	fg4	6	9BE3B26A-D646-41FD-AC3C-DE92B99A3208	0	ROWS	5
7	ONLINE	fg5	7	C703EFAD-3FE8-4121-B362-B7DF289C5A3D	0	ROWS	6
8	ONLINE	fg6	8	D1AEB38-864D-462A-B387-20242EAB8E5B	0	ROWS	7

## ➡ データ パーティションのまとめ

以上のように、データ パーティションは、運用管理性／保守性の利点が非常に多くあります。特に、本自習書で試した次の特徴は、データ パーティションを利用する大きなメリットになります。

- 古いデータを定期的にアーカイブ テーブルへ移動する「**スライディング ウィンドウ**」のシナリオを簡単に実現可能
- 古いデータ削除時のパフォーマンスが大幅アップ
- パーティション単位でデータ圧縮が可能
- パーティション単位でインデックスの再構築と再編成が可能。古いパーティションは、インデックスの再構築が必要ないケースが多いので、テーブル全体を再構築するよりも、大幅に再構築時間を短縮することが可能
- パーティションを複数のファイル グループで構成しておけば、パーティション単位でバック

アップと復元が可能

- パーティションを複数のデータ ファイル (.ndf) で構成しておけば、一部のデータ ファイルが破損しても、そのまま稼働させることが可能
- 一部のデータ ファイルが破損しても、破損したファイル グループのみを復元するだけで完全復旧可能

## ➡ おわりに

最後までこの自習書の内容を試された皆さま、いかがでしたでしょうか？

SQL Server 2012 のデータ パーティション機能には、多くのメリットがあり、大規模データベース環境では非常に重要な機能であることを確認できたのではないのでしょうか。今回は「**入門編**」ということで、データ パーティションの基本的な操作方法のみの紹介になりましたが、データ パーティションのパフォーマンスについては、以下のホワイト ペーパーへ詳しく記載されていますので、ぜひ参考に見てみてください（SQL Server 2005 ベースの情報ですが、SQL Server 2012 でも、多くのことが当てはまります）。

**徹底検証シリーズ「SQL Server 2005 データ パーティション パフォーマンス検証」**

<http://www.microsoft.com/ja-jp/sqlserver/2008/r2/technology/cqi.aspx#a03>

## 5.2 参考情報：パーティション ウィザード

### ➡ パーティション ウィザード

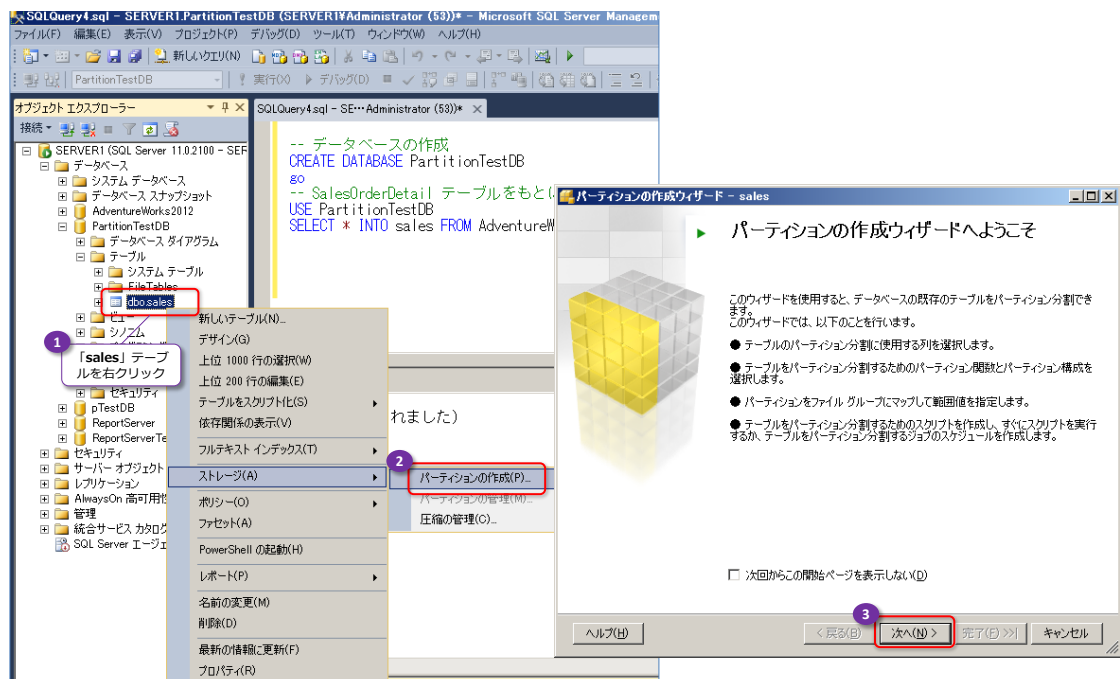
パーティション ウィザードは、その名のとおり、データ パーティションの設定をウィザードで簡単に設定できる機能です（SQL Server 2008 から提供されました）。

それでは、これを試してみましょう。ここでは、**AdventureWorks2012** データベースの「**SalesOrderHeader**」テーブルをもとに新しいテーブルを作成し、そのテーブルをパーティション化してみます（AdventureWorks2012 データベースは、CodePlex サイト <http://msftdbprodsamples.codeplex.com/> からダウンロードすることができます）。

1. まずは、次のように「**PartitionTestDB**」データベースを作成し、**AdventureWorks2012** データベースの「**SalesOrderHeader**」テーブルをもとに「**Sales**」テーブルを作成します。

```
-- データベースの作成
CREATE DATABASE PartitionTestDB
go
-- SalesOrderDetail テーブルをもとに sales テーブルを作成
USE PartitionTestDB
SELECT * INTO sales FROM AdventureWorks2012.Sales.SalesOrderHeader
```

2. 次に、作成した「**PartitionTestDB**」データベースの「**sales**」テーブルを右クリックして、**[ストレージ]** の **[パーティションの作成]** をクリックします。



すると、「パーティションの作成ウィザード」が起動するので、**[次へ]** ボタンをクリックします。

3. 次の「パーティション分割列の選択」ページでは、パーティションを区切る基準となる列を選択します。

パーティションの作成ウィザード - sales

パーティション分割列の選択  
テーブルのパーティション分割に使用する列を選択します。

使用可能なパーティション分割列(A):

列名	データ型	長さ	有効桁数	小数点以下桁数
<input type="radio"/> ModifiedDate	datetime	8	23	3
<input type="radio"/> OnlineOrderFlag	bit	1	1	0
<input checked="" type="radio"/> OrderDate	datetime	8	23	3
<input type="radio"/> PurchaseOrderNu...	nvarchar	25	0	0
<input type="radio"/> RevisionNumber	tinyint	1	3	0
<input type="radio"/> rowguid	uniqueidentifier	16	0	0
<input type="radio"/> SalesOrderID	int	4	10	0
<input type="radio"/> SalesOrderNumber	nvarchar	25	0	0
<input type="radio"/> SalesPersonID	int	4	10	0

☐ このテーブルを選択したパーティション テーブルに併置する(C):

☐ 一意ではないインデックスと一意インデックスをすべて、インデックス付きのパーティション分割列でストレージ固定する(S)

上のグリッドには、選択したテーブルのパーティション分割列が表示されています。このテーブルでパーティション分割列として使用する列を選択してください。

ヘルプ(H) < 戻る(B) 次へ(N) > 完了(F) >> キャンセル

ここでは、「OrderDate」（受注日）を選択して、[次へ] ボタンをクリックします。

4. 次の「パーティション関数の選択」ページでは、[新しいパーティション関数] を選択して、任意の名前（pf1 など）を入力し、[次へ] ボタンをクリックします。

パーティションの作成ウィザード - sales

パーティション関数の選択  
新しいパーティション関数を作成するか、パーティション分割用に既存の関数を選択します。

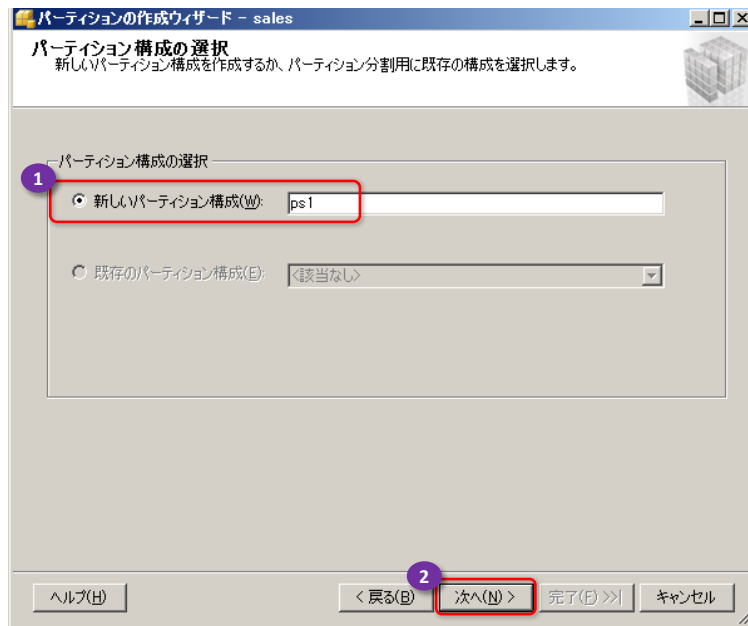
パーティション関数の選択

☒ 新しいパーティション関数(W): pf1

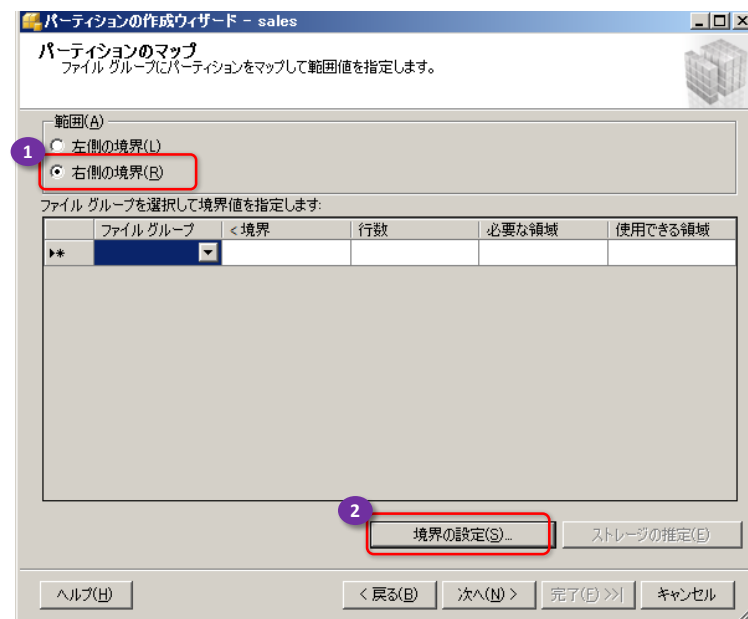
☐ 既存のパーティション関数(E): <使用できるパーティション関数がありません>

ヘルプ(H) < 戻る(B) 次へ(N) > 完了(F) >> キャンセル

5. 次の「パーティション構成の選択」ページでは、[新しいパーティション構成] を選択して、任意の名前（ps1 など）を入力し、[次へ] ボタンをクリックします。



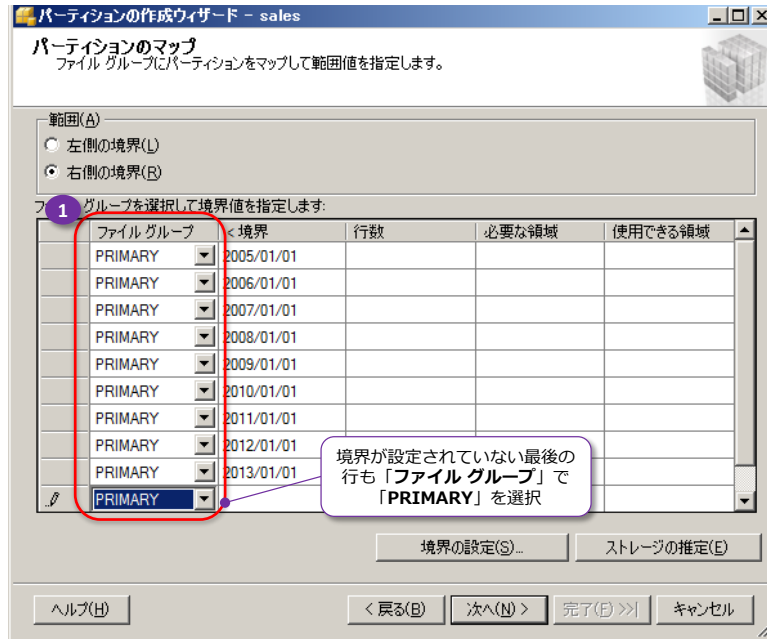
6. 次の「パーティションのマップ」ページでは、「右側の境界」を選択して、「境界の設定」ボタンをクリックします。



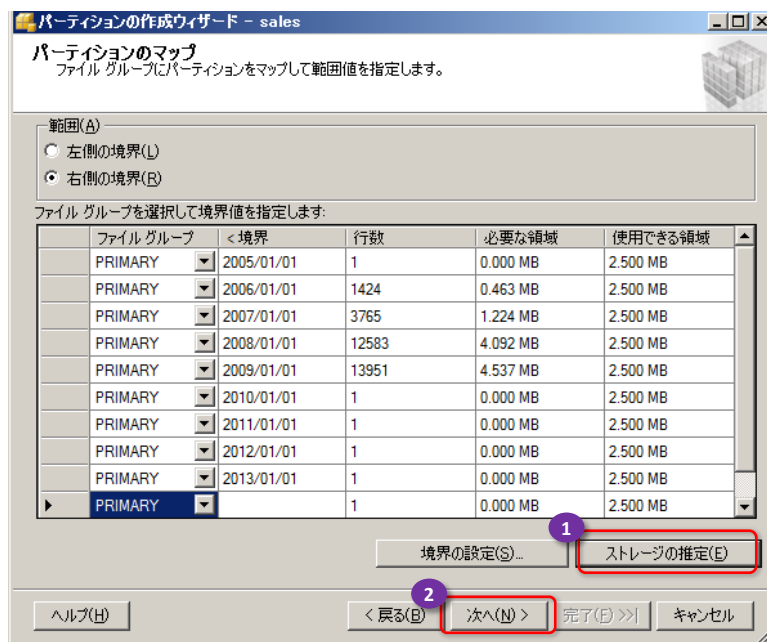
7. 「境界値の設定」ダイアログが表示されたら、「開始日」を「2005/01/01」へ、「終了日」を「2012/12/31」へ、「日付範囲」を「毎年」へ変更し、「OK」ボタンをクリックします。



これにより、2005 年から 2012 年までの 1 年ごとのパーティション範囲を作成することができます。



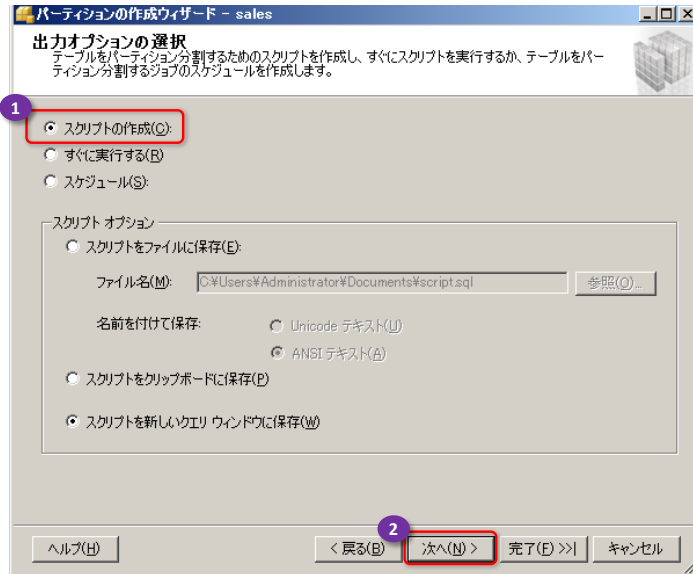
8. 次に、それぞれの境界の「ファイル グループ」で「PRIMARY」を選択します。



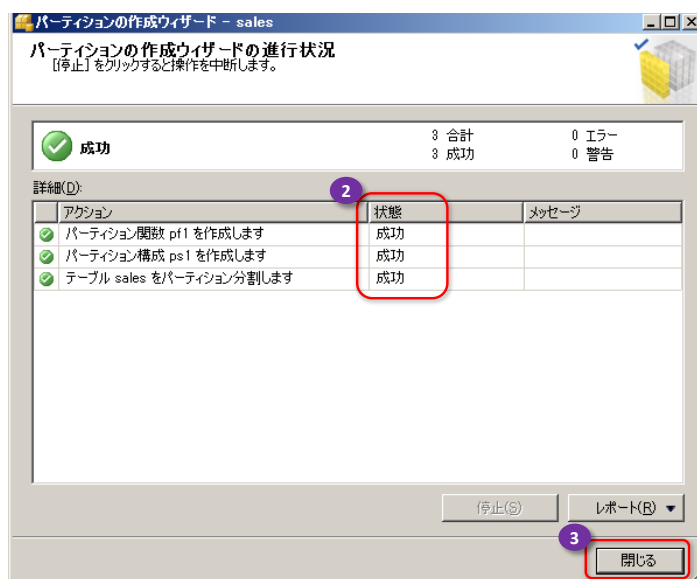
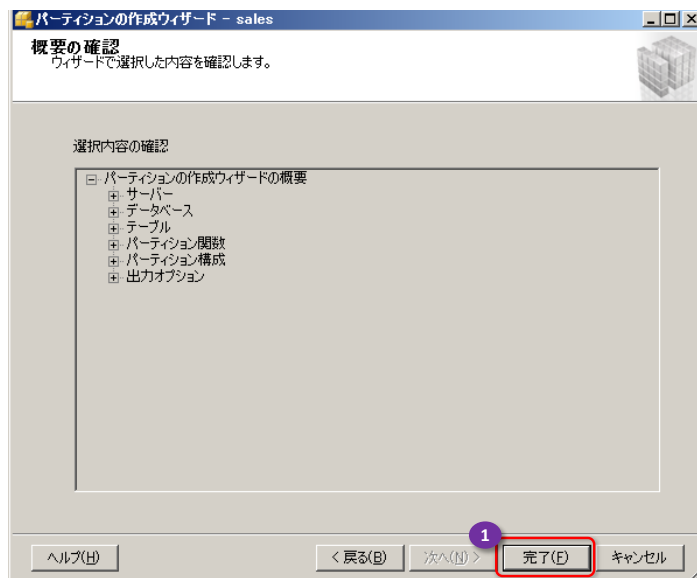
続いて、[ストレージの推定] ボタンをクリックして、各パーティションに含まれる行数や領域の見積もりを表示します。

確認後、[次へ] ボタンをクリックして、次へ進みます。

9. 次の「出力オプションの選択」ページでは、[スクリプトの作成] を選択して、[次へ] ボタンをクリックします。

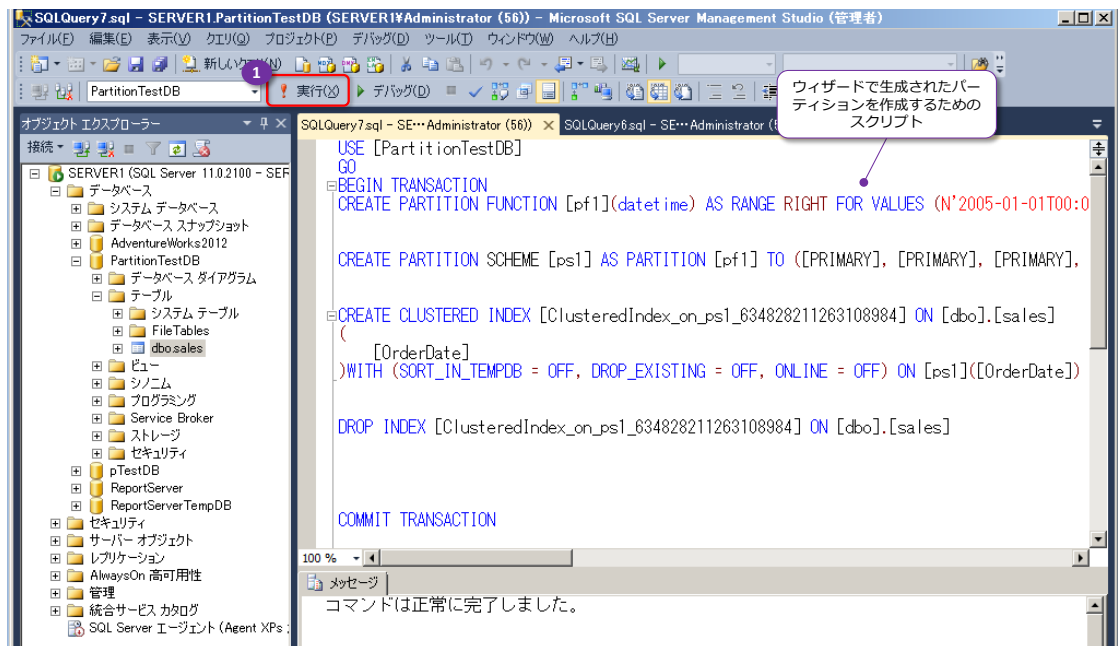


10. 最後の「概要の確認」ページでは、「完了」ボタンをクリックします。





11. ウィザードが完了すると、パーティションを作成するためのスクリプトがクエリ エディターへ自動生成されるので、ツール バーの【実行】ボタンをクリックして、ステートメントを実行します。



12. 次に、データがパーティション分割されたことを確認するために、次のように **\$PARTITION** 関数を利用します。

```
SELECT $PARTITION.pf1(OrderDate), * FROM sales
```

	(列名なし)	SalesOrderID	RevisionNumber	OrderDate	DueDate
1	2	43659	3	2005-07-01 00:00:00.000	2005-07-13 00
2	2	43660	3	2005-07-01 00:00:00.000	2005-07-13 00
3	2	43661	3	2005-07-01 00:00:00.000	2005-07-13 00
4	2	43662	3	2005-07-01 00:00:00.000	2005-07-13 00
5	2	43663	3	2005-07-01 00:00:00.000	2005-07-13 00
6	2	43664	3	2005-07-01 00:00:00.000	2005-07-13 00
7	2	43665	3	2005-07-01 00:00:00.000	2005-07-13 00
8	2	43666	3	2005-07-01 00:00:00.000	2005-07-13 00
⋮					
17511	4	55922	3	2007-10-11 00:00:00.000	2007-10-23 00
17512	4	55923	3	2007-10-11 00:00:00.000	2007-10-23 00
17513	4	55924	3	2007-10-11 00:00:00.000	2007-10-23 00
17514	4	55925	3	2007-10-11 00:00:00.000	2007-10-23 00
17515	5	61173	3	2008-01-01 00:00:00.000	2008-01-13 00
17516	5	61174	3	2008-01-01 00:00:00.000	2008-01-13 00
17517	5	61175	3	2008-01-01 00:00:00.000	2008-01-13 00
17518	5	61176	3	2008-01-01 00:00:00.000	2008-01-13 00
17519	5	61177	3	2008-01-01 00:00:00.000	2008-01-13 00
17520	5	61178	3	2008-01-01 00:00:00.000	2008-01-13 00

このようにパーティション ウィザードを利用すると、既存のテーブルを簡単にパーティション分割できるようになります。

## 執筆者プロフィール

**有限会社エスキューエル・クオリティ (http://www.sqlquality.com/)**

SQLQuality (エスキューエル・クオリティ) は、**日本で唯一の SQL Server 専門の独立系コンサルティング会社**です。過去のバージョンから最新バージョンまでの SQL Server を知りつくし、多数の実績と豊富な経験を持つ、OS や .NET にも詳しい **SQL Server の専門家 (キャリア 17 年以上) がすべての案件に対応します**。人気メニューの「**パフォーマンス チューニング サービス**」は、100%の成果を上げ、過去すべてのお客様環境で驚異的な性能向上を実現。チューニング スキルは**世界トップレベル**を自負、検索エンジンでは (英語情報を含めて) ヒットしないノウハウを多数保持。ここ数年は **BI/DWH システム構築支援**のご依頼が多い。

主なコンサルティング実績

- ▶ 大手映像制作会社の **BI システム構築支援** (会計/業務システムにおける予算管理/原価管理など)
- ▶ 大手流通系の **DWH/BI システム構築支援** (POS データ/在庫データ分析)  
大規模テラバイト級データ ウェアハウスの物理・論理設計支援および運用管理設計支援
- ▶ 大手アミューズメント企業の **BI システム構築支援** (人事システムにおける人材パフォーマンス管理)
- ▶ 外資系医療メーカーの Analysis Services による「**販売分析**」システムの構築支援 (売上/顧客データ分析)
- ▶ **9 TB** データベースの物理・論理設計支援 (パーティショニング対応など)
- ▶ ハードウェア リプレース時の**ハードウェア選定** (最適なサーバー、ストレージの選定)、**高可用性環境**の構築
- ▶ **SQL Server 2000** (32 ビット) から **SQL Server 2008** (x64) への移行/アップグレード支援
- ▶ 複数台の SQL Server の **Hyper-V 仮想環境**への移行支援 (サーバー統合支援)
- ▶ **2 時間**かかっていた日中バッチ実行時間を、わずか **5 分**へ短縮 (**95.8%** の性能向上)
- ▶ ピーク時の CPU 利用率 **100%** のシステムを、わずか **10%** にまで軽減し、大幅性能向上
- ▶ 平均 **185.3ms** かかっていた処理を、わずか **39.2ms** へ短縮 (**78.8%** の性能向上)
- ▶ **Java 環境** (Tomcat, Seasar2, S2Dao) の SQL Server パフォーマンス チューニング etc

コンサルティング時の作業例 (パフォーマンス チューニングの場合)

- ▶ アプリケーション コード (VB, C#, Java, ASP, VBScript, VBA) の解析/改修支援
- ▶ ストアド プロシージャ/ユーザー定義関数/トリガー (Transact-SQL) の解析/改修支援
- ▶ インデックス チューニング/SQL チューニング/ロック処理の見直し
- ▶ 現状のハードウェアで将来のアクセス増にどこまで耐えられるかを測定する高負荷テストの実施
- ▶ IIS ログの解析/アプリケーション ログ (log4net/log4j) の解析
- ▶ ボトルネック ハードウェアの発見/ボトルネック SQL の発見/ボトルネック アプリケーションの発見
- ▶ SQL Server の構成オプション/データベース設定の分析/使用状況 (CPU, メモリ, ディスク, Wait) 解析
- ▶ 定期メンテナンス支援 (インデックスの再構築/断片化解消のタイミングや断片化の事前防止策など) etc

**松本美穂 (まつもと・みほ)**

有限会社エスキューエル・クオリティ 代表取締役

Microsoft MVP for SQL Server (2004 年 4 月~)

経産省認定データベース スペシャリスト/MCDBA/MCSD for .NET/MCITP Database Administrator

SQL Server の日本における最初のバージョンである「SQL Server 4.21a」から SQL Server に携わり、現在、SQL Server を中心とするコンサルティングを行っている。得意分野はパフォーマンス チューニングと Reporting Services。コンサルティング業務の傍ら、講演や執筆も行い、マイクロソフト主催の最大イベント Tech・Ed などでスピーカーとしても活躍中。SE や ITPro としての経験はもちろん、記名/無記名含めて多くの執筆実績も持ち、様々な角度から SQL Server に携わってきている。著書の『SQL Server 2000 でいってみよう』と『ASP.NET でいってみよう』(いずれも翔泳社刊)は、トップ セラー (前者は 28,500 部、後者は 16,500 部発行)。近刊に『SQL Server 2012 の教科書』(ソシム刊)がある。

**松本崇博 (まつもと・たかひろ)**

有限会社エスキューエル・クオリティ 取締役

Microsoft MVP for SQL Server (2004 年 4 月~)

経産省認定データベース スペシャリスト/MCDBA/MCSD for .NET/MCITP Database Administrator

SQL Server の BI システムとパフォーマンス チューニングを得意とするコンサルタント。過去には、約 3,000 本のストアド プロシージャのチューニングや、テラバイト級データベースの論理・物理設計、運用管理設計、高可用性設計、BI・DWH システム設計支援などを行う。アプリケーション開発 (ASP/ASP.NET, C#, VB 6.0, Java, Access VBA など) やシステム管理者 (IT Pro) 経験もあり、SQL Server だけでなく、アプリケーションや OS、Web サーバーを絡めた、総合的なコンサルティングが行えるのが強み。Analysis Services と Excel による BI システムも得意とする。マイクロソフト認定トレーナー時代の 1998 年度には、Microsoft CPLS トレーナー アワード (Trainer of the Year) を受賞。