Microsoft

SQL Server 2008 自習書シリーズ

注目の新機能をイチ早く試してみよう！

Published: 2008 年 4 月 30 日
改訂版: 2008 年 11 月 30 日
有限会社エスキューエル・クオリティ

SQLQuality
データベース コンサルティング

目次

STEP 1. SQL Server 2008 新機能の概要とインストール時の注意事項

この STEP では、SQL Server 2008 の新機能とインストールをする際の注意事項などを説明します。

この STEP では、次のことを学習します。

- ✓ SQL Server 2008 新機能の概要
- ✓ SQL Server 2008 のインストール要件
- ✓ インストールの実行方法
- ✓ サンプル データベース（AdventureWorks）のダウンロード
- ✓ サンプル スクリプトについて

1.1　SQL Server 2008 について

SQL Server 2008 について

SQL Server 2008 は、2003 年以降、Windows プラットフォーム上で動作するリレーショナル データベース製品としてシェア第 1 位を獲得（ガートナー社調べ）し続けている Microsoft SQL Server の最新バージョンです。以前のバージョンの SQL Server 2005 は、2006 年 2 月に発売されましたので、約 2 年半ぶりのバージョンアップとなります（SQL Server 2008 は、2008 年 8 月に発売されました）。

この自習書では、SQL Server 2008 の注目の新機能を簡単に試せるようにステップ バイ ステップ形式で画面ショット満載でご紹介しますので、ぜひ、皆さんも実際に操作しながらこの自習書を読み進めていただければと思います。

SQL Server 2008 評価版のダウンロード

SQL Server 2008 の評価版は、次の URL からダウンロードすることができます。
http://www.microsoft.com/japan/sqlserver/2008/downloads/default.mspx

SQL Server 2008 の新機能

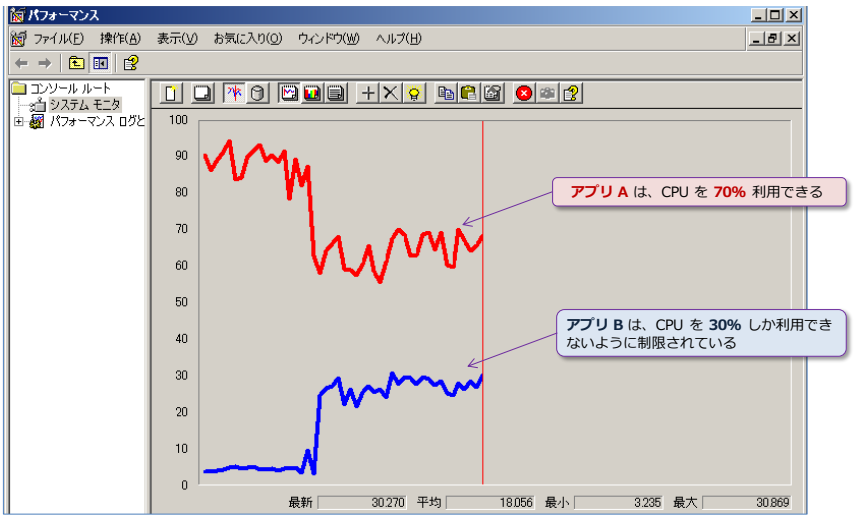
SQL Server 2008 の主な新機能は、次のとおりです。

- **インテリセンス**による入力補完機能
- **新しい日付データ型**（date / time / datetime2 / datetimeoffset）
- **MERGE ステートメント**（データが既に存在する場合は UPDATE を実行し、存在しな

い場合は INSERT を実行できる機能）

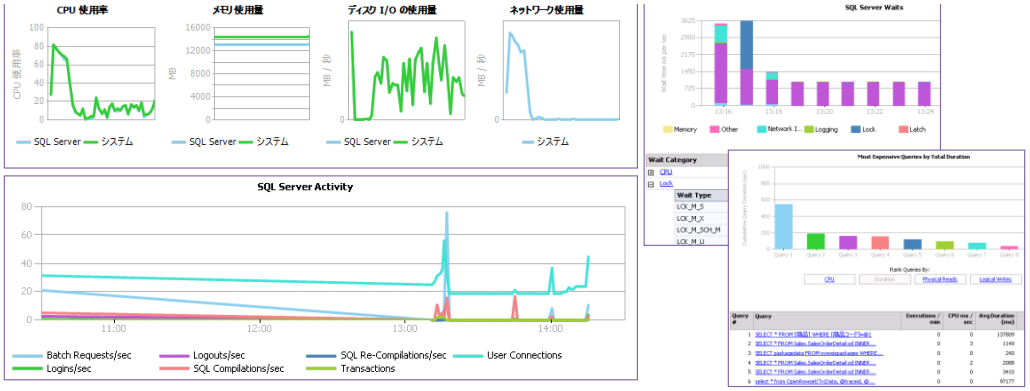
- **オブジェクトの依存関係**を簡単に表示できる機能（ビューやストアド プロシージャのもととなっているテーブルを一覧表示することが可能に）
- **バックアップ圧縮**（バックアップ データを圧縮することで、バックアップとリストアのスピード UP を実現できる機能）
- **データ圧縮**（データを圧縮することで、ディスク Write / Read を減らしてスピード UP を実現できる機能。またディスク コストの削減にも役立つ機能）
- **SQL Server Audit**（SQL Server に対するすべての操作履歴を記録可能に。コンプライアンスの実現や J-SOX 法、内部統制対策として非常に重要となる、SQL Server に対するすべての操作のログを残せる機能。GUI ベースでグラフィカルに設定が可能）
- **透過的なデータ暗号化**（アプリケーションを修正することなく、データの暗号化を行える機能）
- **データ プロファイル タスク**（データの分布状況をグラフィカルに表示することができ、インデックス チューニング時などに大変役立つ機能）

- **リソース ガバナ**（CPU 利用率やメモリ使用量を調整できる機能。CPU を占有してしまうアプリケーションの占有率を下げるなどの利用が可能に）

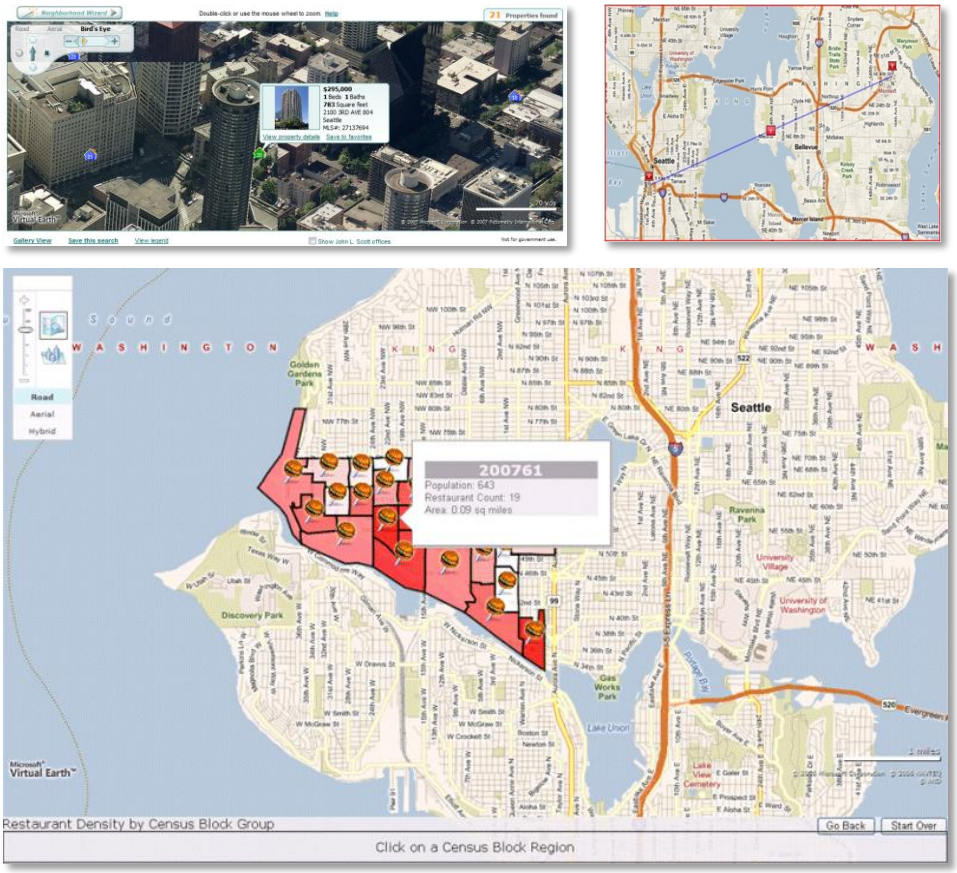


- **ポリシー ベースの管理**（Active Directory のグループ ポリシー機能と同じように、管理者が設定したポリシーを強制適用できる機能。セキュリティ設定の強制や、パフォーマンス関連の設定値を確認する際などに役立つ機能）

- **パフォーマンス データ コレクション** と **データ コレクション**（パフォーマンス状況をグラフィカルに監視することができ、パフォーマンス チューニング時に大変役立つ機能）



- **Spatial データ型**（GIS：地理情報システムにおける地図データを格納できるデータ型で、Virtual Earth と連携することも可能に）



- **パーティション ウィザード**（パーティションの設定をウィザード形式で容易に設定）
- **FileStream データ型**（Word や Excel などの Office ドキュメントや、任意の OS ファイルをデータベース内へ格納することが可能に）

これらの新機能は、この自習書でステップ バイ ステップ形式で試せるように構成していますので、ぜひ試してみてください。

1.2　SQL Server 2008 のインストール

インストール要件

SQL Server 2008 をインストールするためのソフトウェア要件は、SQL Server 2005 の場合とほとんど同じで、次のとおりです。

	要件
OS	Windows Server 2003 SP2 以降　または Windows XP Professional SP2 以降　または Windows Vista　または Windows Server 2008
ソフトウェア	・MDAC 2.8 SP1 以降 ・Internet Explorer 6.0 SP1 以降 ・.NET Framework 3.5 SP1 以降（SQL Server のセットアップ時にインストールが促されます） ・Windows インストーラ 4.5 以降（SQL Server のセットアップ時にインストールが促されます）

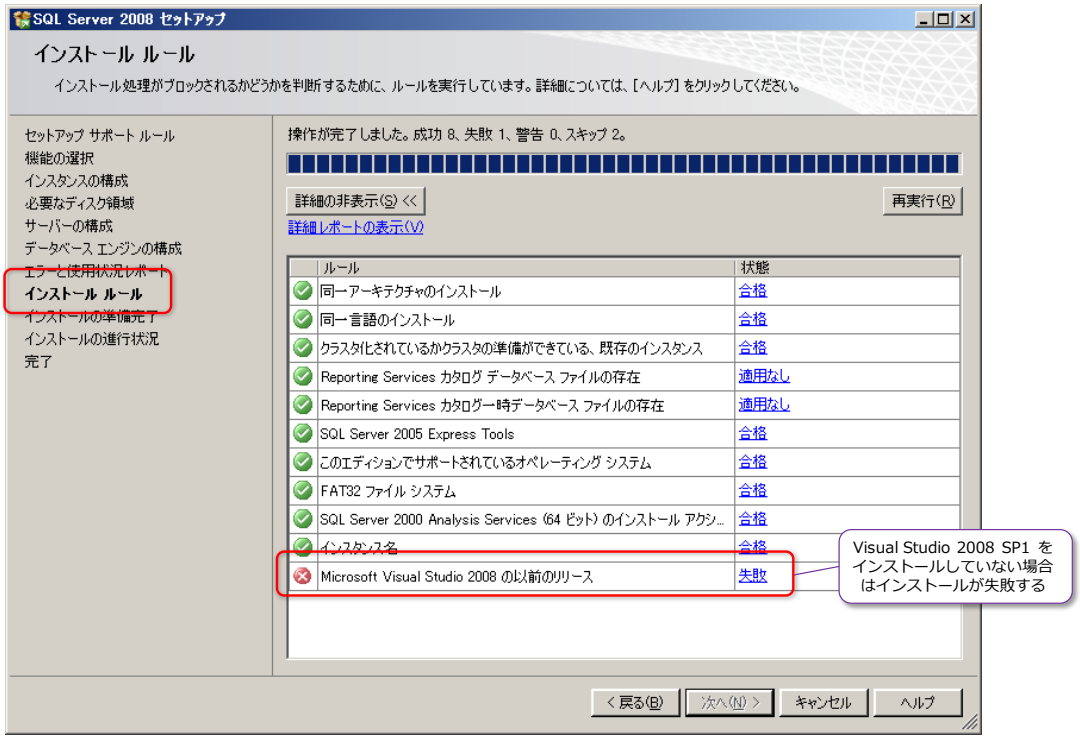
この自習書内での画面やテキストは、OS に Windows Server 2003 SP2、ソフトウェアに SQL Server 2008 Enterprise Edition を利用して記述しています。

Note：　Visual Studio 2008 と共存させる場合の注意点

Visual Studio 2008 と SQL Server 2008 を同じマシンにインストールする場合には、次の 2 点に注意する必要があります。

■ **Visual Studio 2008 をインストール済みのマシンへ SQL Server 2008 をインストールする場合**

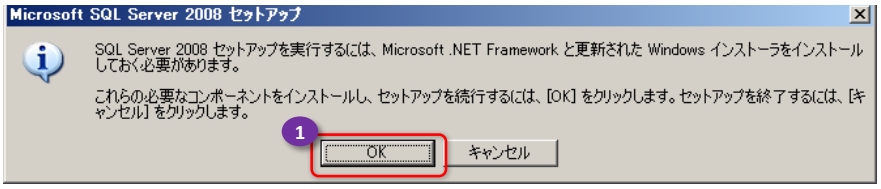
Visual Studio 2008 をインストール済みのマシンへ SQL Server 2008 をインストールする場合には、SQL Server 2008 をインストールする前に、Visual Studio 2008 の Service Pack 1（SP1）を適用しておく必要があります。もし、Visual Studio 2008 SP1 を適用していない場合には、SQL Server 2008 インストール時に、次のように［インストール ルール］画面で「失敗」となり、インストールを続行することができなくなります。



■ SQL Server 2008 のインストール後に、Visual Studio 2008 をインストールする場合
筆者の環境では、SQL Server 2008 をインストール済みのマシンへ、後から Visual Studio 2008 をインストールした場合に、「インストールは成功」と表示されるにも関わらず、Visual Studio 2008 の選択した機能（VB や C#）がインストールされていないことが何度かありました。この場合は、Visual Studio 2008 の修復セットアップを実行することで、正常にインストールすることができました。

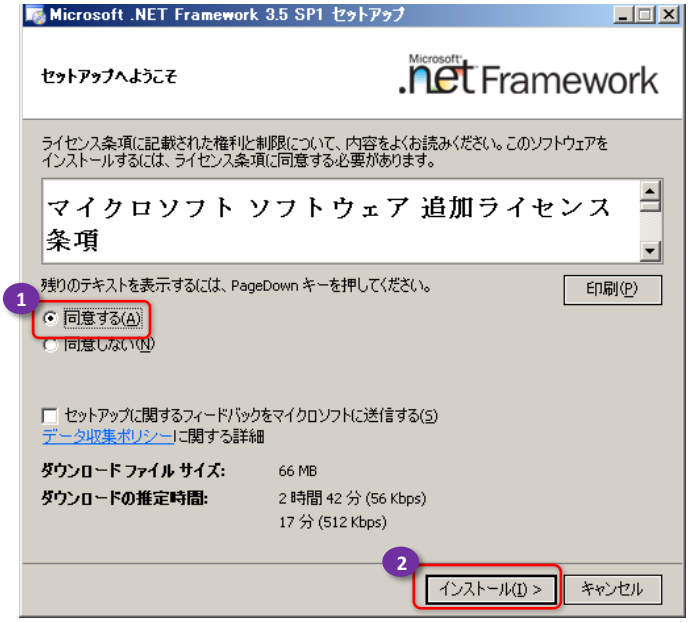
SQL Server 2008 のインストールの開始方法

SQL Server 2008 のインストール メディアをドライブへセットすると、次のように［**Microsoft SQL Server 2008 セットアップ**］ダイアログが起動します。



このダイアログが表示されない場合は、インストール メディアのルート フォルダにある **setup.exe** をダブル クリックします。

続いて、次のように［**Microsoft .NET Framework 3.5 SP1 セットアップ**］画面が表示されるので、ライセンス条項に［**同意する**］をチェックして、［**インストール**］ボタンをクリックします。



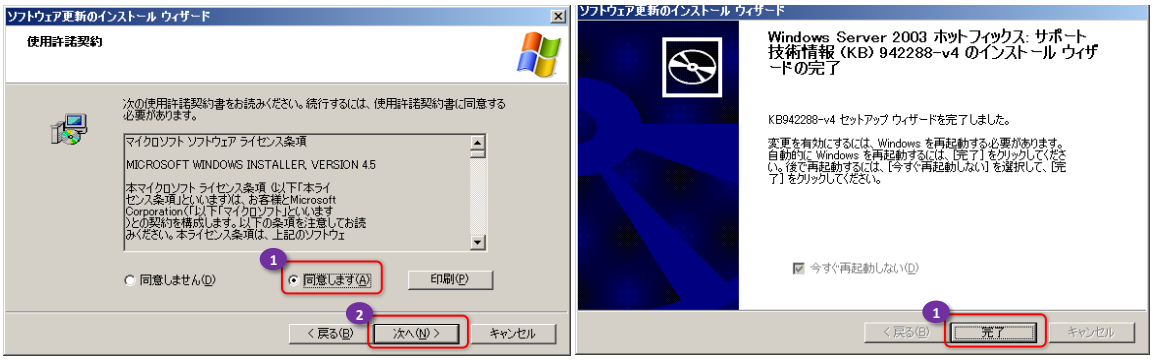
これにより、.NET Framework 3.5 SP1（Service Pack 1）のインストールが開始されます。

Note： .NET Framework 3.5 SP1 は必須コンポーネント

SQL Server 2008 では、.NET Framework 3.5 SP1 が必須コンポーネントになりました。このインストールには、66 MB 分のファイルのダウンロードとインストールの時間がかかるので、10 ～20 分程度の時間がかかります（ネットワーク環境によっては 30 分以上かかる場合もあります）。
なお、Visual Studio 2008 SP1 をインストール済みのマシンへ SQL Server 2008 をインストールする場合には、.NET Framework 3.5 SP1 が既にインストールされているので、この手順はスキップされ、次の［ソフトウェア更新のインストール ウィザード］が表示されます。

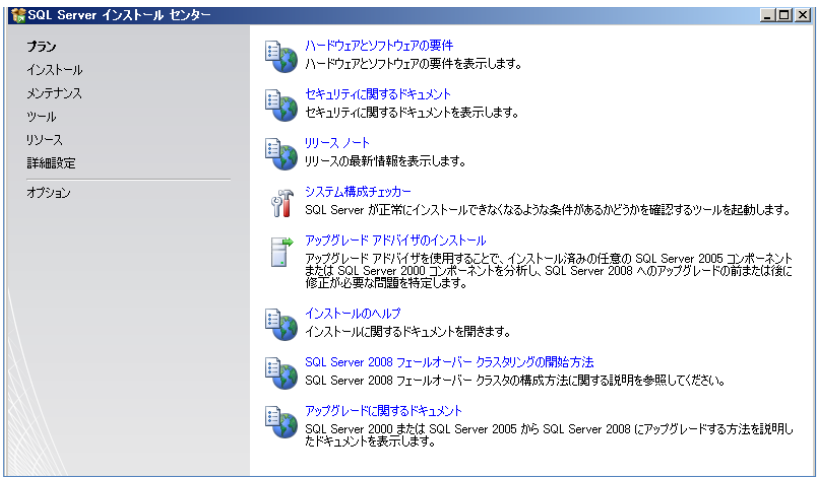
.NET Framework 3.5 SP1 のインストールが完了すると、[**ソフトウェア更新のインストール ウィザード**] が表示されます。

このホット フィックス（KB 942288-v4）は、Windows インストーラ 4.5 のことです。



Windows インストーラ 4.5 のインストールが完了し、[完了] ボタンをクリックすると、"再起動が必要" という旨のメッセージが表示されますが、[OK] ボタンをクリックしても自動では再起動が行われないことに注意してください。OS の再起動は、[スタート] メニューから手動で行う必要があります。

OS の再起動が完了したら、SQL Server のインストール メディアから **setup.exe** をダブル クリックします。これにより、インストール プログラムが起動して、次のように [**SQL Server インストール センター**] 画面が表示されます。



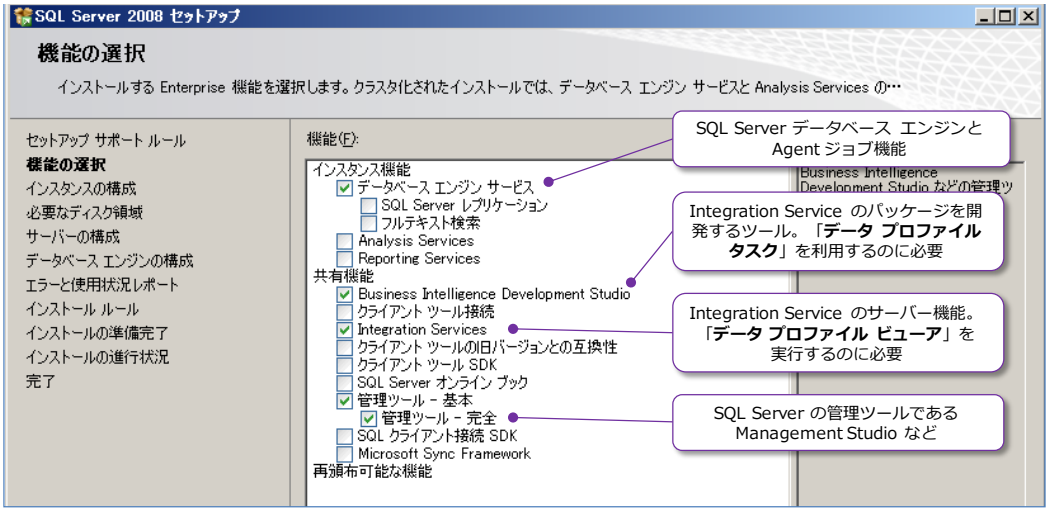
ここでは、［**インストール**］ページをクリックして、「**SQL Server の新規スタンドアロン インストールまたは既存のインストールへの機能の追加**」をクリックれば、インストールを開始することができます。

Note： SQL Server 2008 の詳しいインストール手順について

SQL Server 2008 の詳しいインストール手順については、本自習書シリーズの「ささっと試せる SQL Server 超入門」で説明しています。

本自習書を試すために必要な機能について

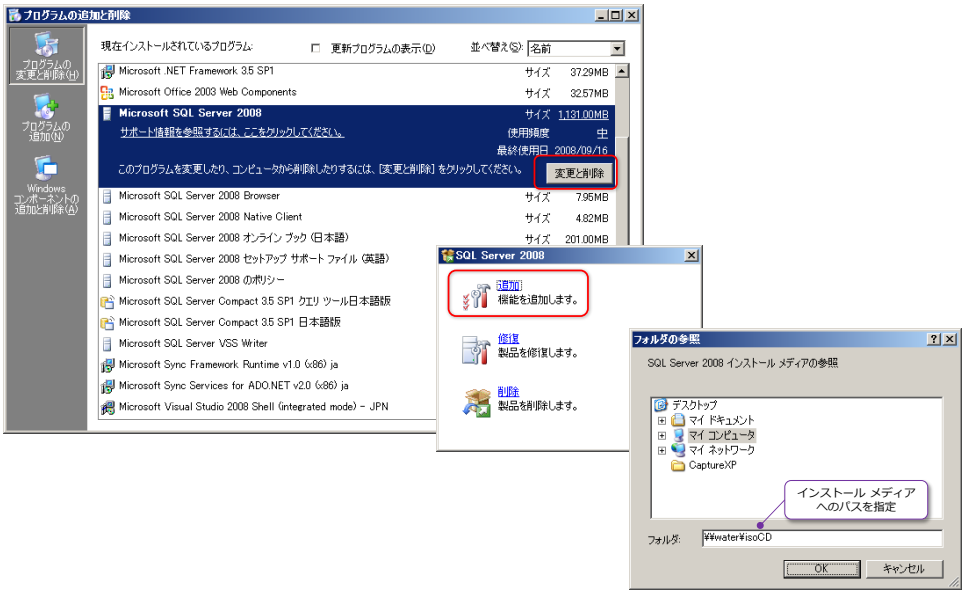
本自習書のすべての機能を試すには、SQL Server 2008 のインストール時の［**機能の選択**］画面で次のコンポーネントを選択しておく必要があります。



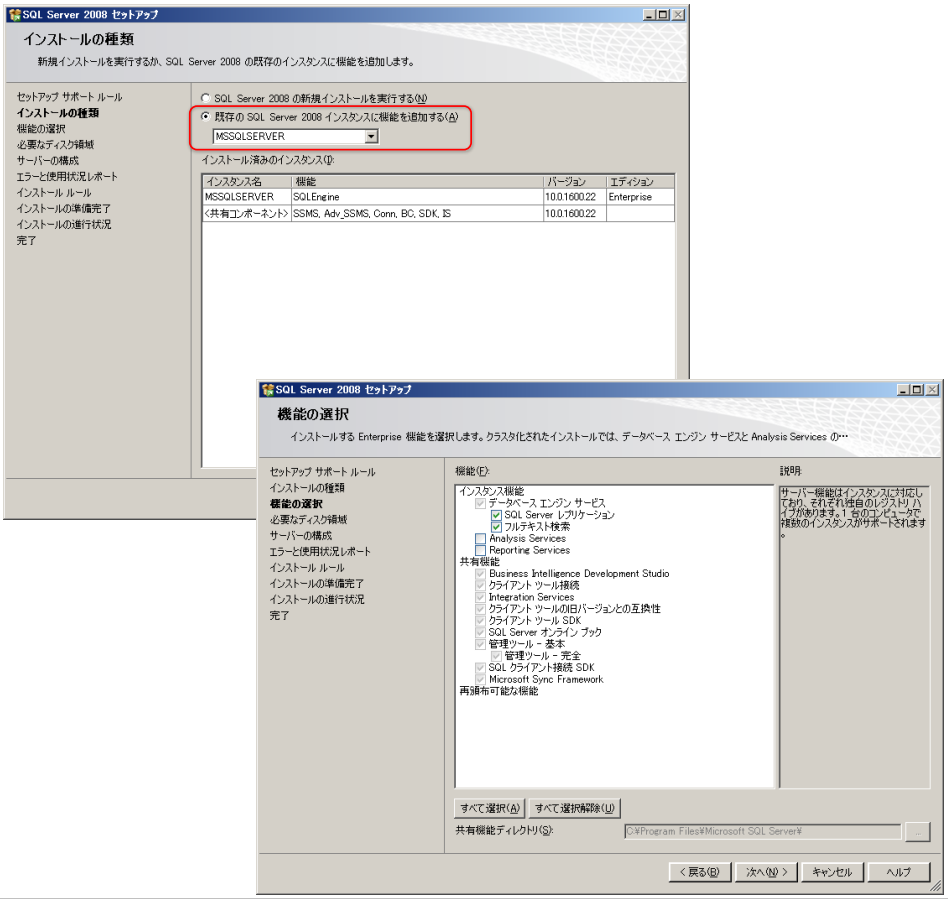
「**データベース エンジン サービス**」と「**Business Intelligence Development Studio**」、「**Integration Services**」、「**管理ツール - 完全**」の 4 つのコンポーネントをチェックしておいてください。

Note：　後から機能を追加インストールする場合

SQL Serer 2008 のインストール後に、後から機能を追加したい場合には、インストールを再実行するか、次のように［コントロール パネル］の［プログラムの追加と削除］から、「Microsoft SQL Server 2008」の［変更と削除］ボタンをクリックして、「追加」をクリックします。

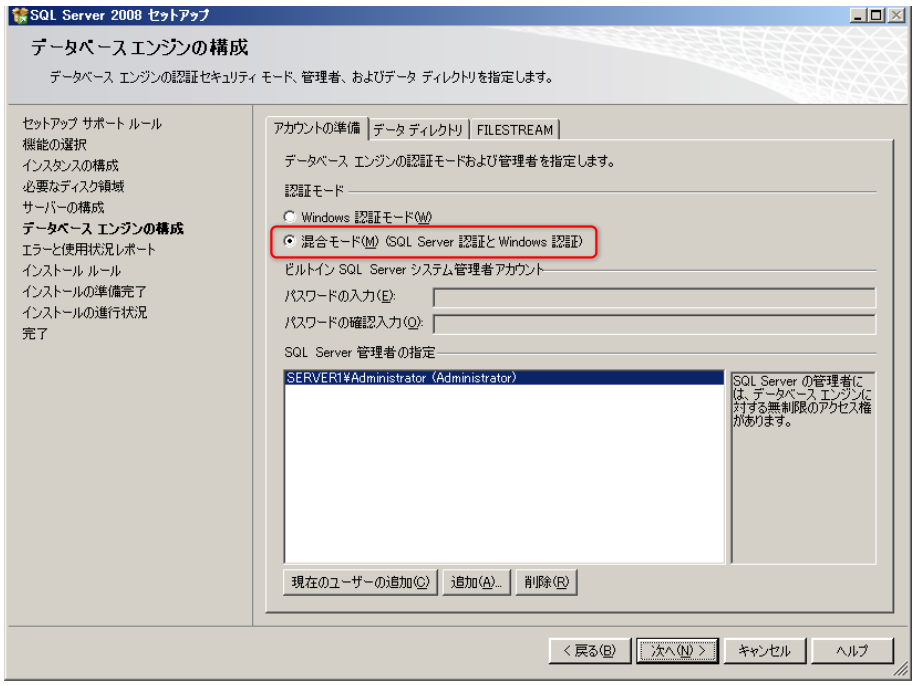


［フォルダの参照］ダイアログでは、インストール メディアへのパスを指定すると、インストールが開始されます。［インストールの種類］画面では、「既存の SQL Server 2008 インスタンスに機能を追加する」をチェックして、［機能の選択］画面で、追加したい機能を選択すれば、後から機能を追加することができます。



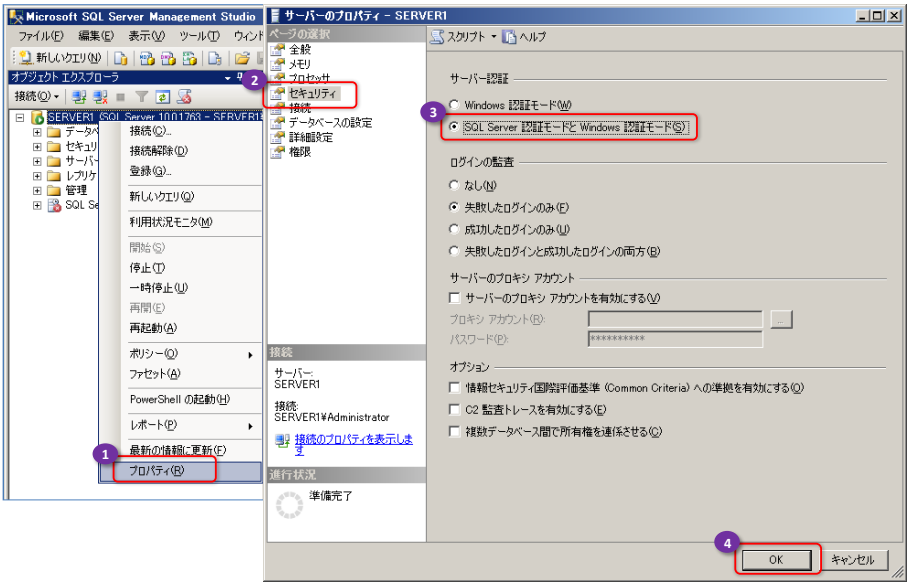
本自習書を試すための認証モードについて

本自習書の一部の手順では、SQL Server 認証（混合モード）を利用しています。認証モードをインストール時に変更するには、次のように［**データベース エンジンの構成**］画面で「混合モード」を選択するようにします。



Note： 後から認証モードを変更する場合

SQL Server のインストール後に、後から認証モードを変更するには、オブジェクト エクスプローラから、次のように SQL Server の名前を右クリックして［**プロパティ**］をクリックします。



［サーバーのプロパティ］ダイアログでは、［**セキュリティ**］ページを開き、「**SQL Server 認証モード**」を選択して、［OK］ボタンをクリックします。クリック後、再起動を促すダイアログが表示されたら、［OK］ボタンをクリックします。

Note： 修正プログラムのインストール

9 月 25 日現在、最新の累積修正パッケージ「**Cumulative update package 1 for SQL Server 2008**」（CU1、KB 956717）が公開されているので、ダウンロードしてインストールしておくことをお勧めします。次の URL から申し込むことができます。

http://support.microsoft.com/kb/956717/en-us

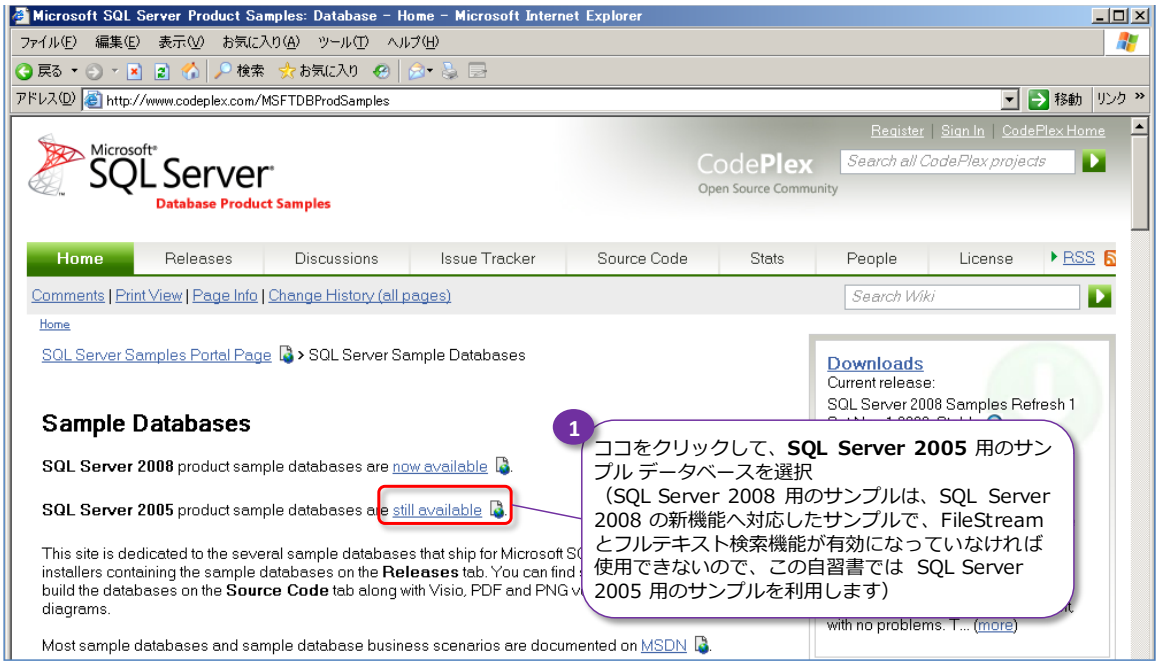
1.3　サンプル データベース（AdventureWorks）のダウンロード

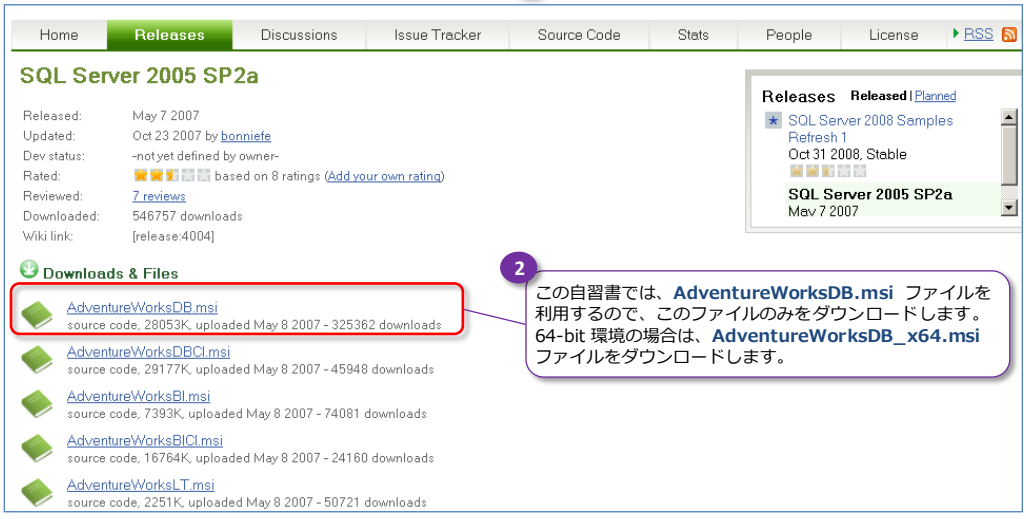
AdventureWorks のダウンロード（この自習書内で利用します）

SQL Server 2008 のインストールでは、サンプル データベースとなる「AdventureWorks」がインストールされません。AdventureWorks データベースは、次の CodePlex サイトからダウンロードすることができます。この自習書内では、いくつかの機能を試す手順で SQL Server 2005 用の AdventureWorks データベース（**AdventureWorksDB.msi**）を利用しているので（64-bit 版の場合は **AdventureWorksDB_x64.msi**）、ダウンロードして、インストールしておいてください（.msi ファイルをダブルクリックするとダウンロードが開始されます）。

CodePlex サイト

http://www.codeplex.com/MSFTDBProdSamples

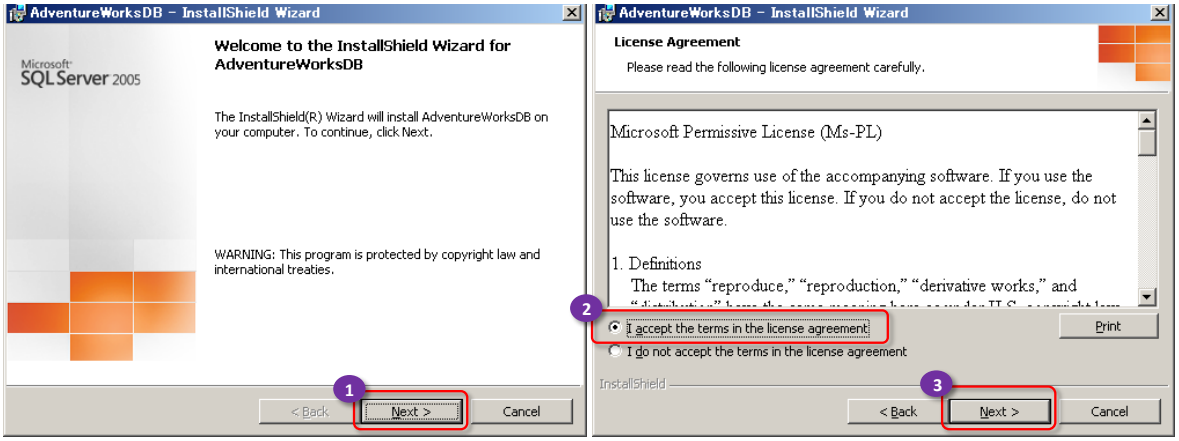




ダウンロードした .msi ファイルをダブル クリックすると、[セキュリティの警告] 画面が表示さ

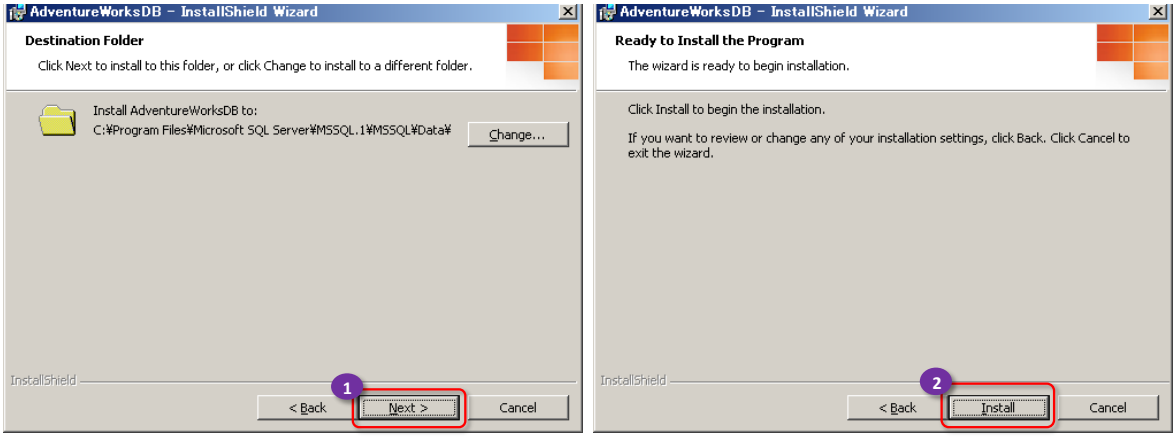
れるので、[実行する] ボタンをクリックします。

これにより、AdventureWorks データベースのセットアップ ウィザードが開始されます。

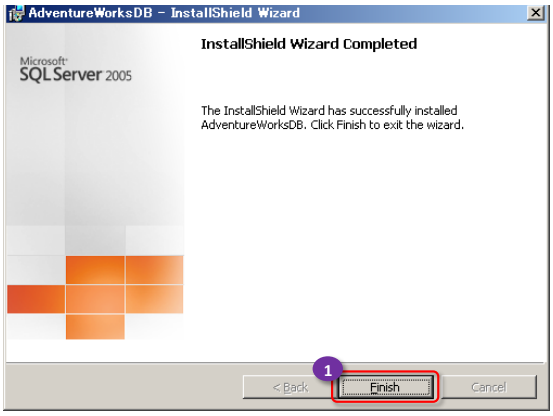


ライセンスに同意して、[Next] ボタンをクリックして、次へ進みます。

次の [**Destination Folder**] 画面では、AdventureWokrs データベースの保存先を指定しますが、ここでは、デフォルトのまま [Next] ボタンをクリックします。



続く、[**Ready to Install the Program**] 画面で、[**install**] をクリックすると、AdventureWorks データベースのインストールが開始されます。



以上で AdventureWorks データベースのインストールが完了です。

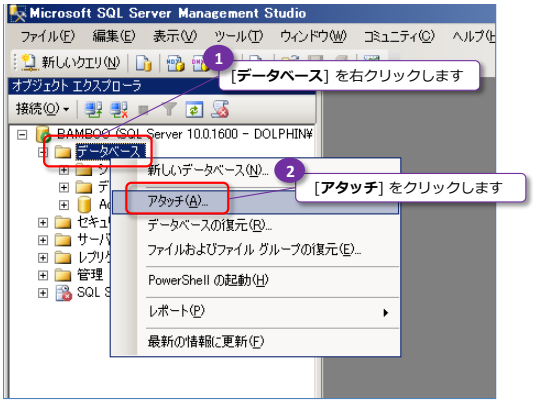
AdventureWorks のアタッチ

続いて、インストールした AdventureWorks データベースをアタッチします。アタッチの手順は次のとおりです。

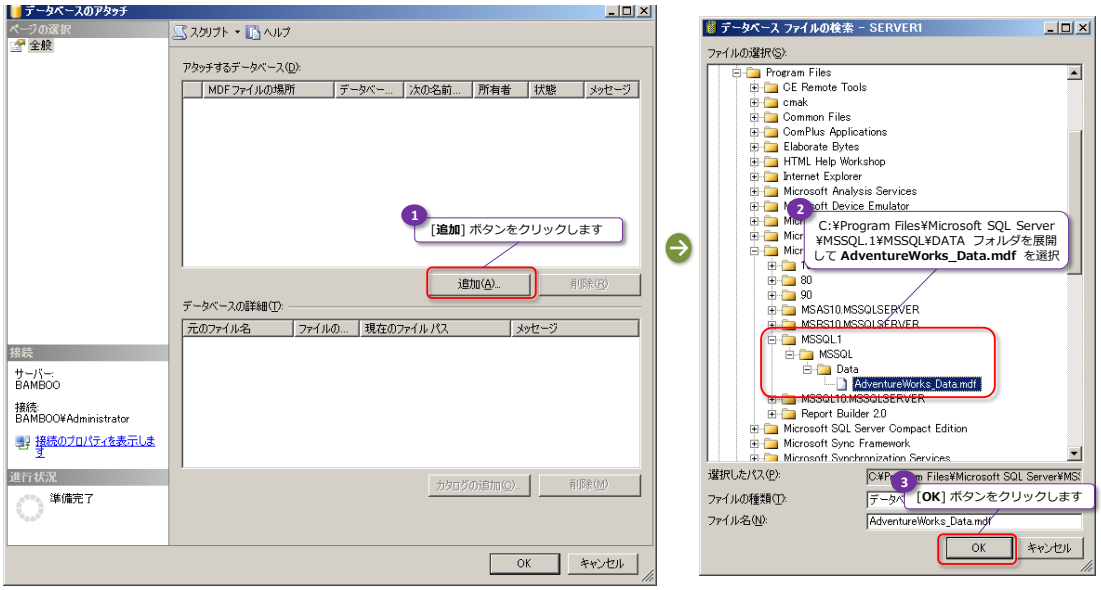
1. ［スタート］メニューの［すべてのプログラム］から、［Microsoft SQL Server 2008］を選択して［SQL Server Management Studio］をクリックし、Management Studio を起動します。

2. 起動後、［**サーバーへの接続**］ダイアログで、［**サーバー名**］へ SQL Server の名前を入力し、［**接続**］ボタンをクリックします。

3. 接続完了後、次のように［**データベース**］フォルダを右クリックして［**アタッチ**］をクリックします。

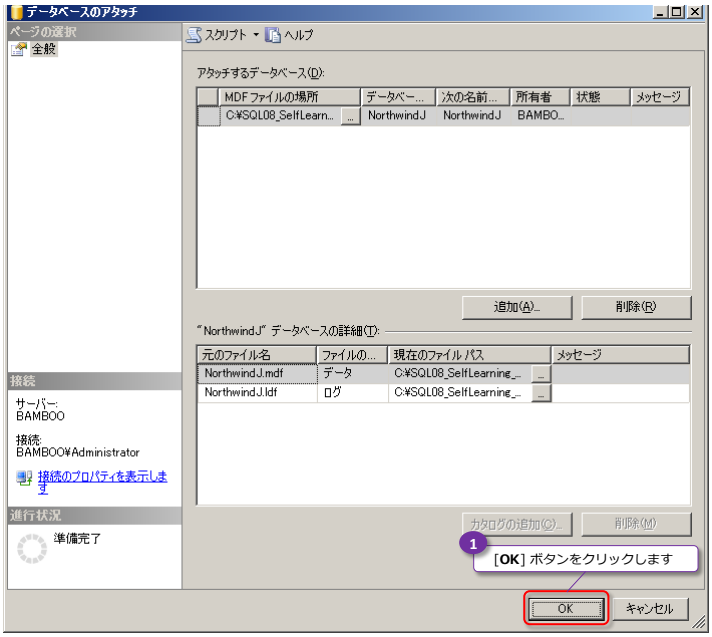


4. すると、次のように［データベースのアタッチ］ダイアログが表示されるので、［**追加**］ボタンをクリックします。



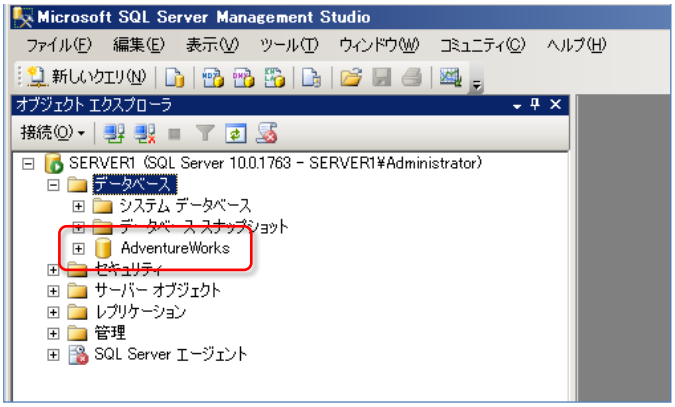
「**C:¥Program Files¥Microsoft SQL Server¥MSSQL.1¥MSSQL¥DATA**」フォルダを展開して、「**AdventureWorks_Data.mdf**」ファイルを選択し、［OK］ボタンをクリックします。

5. 次のように［データベースのアタッチ］ダイアログへ戻ったら、［OK］ボタンをクリックします。



次の警告ダイアログが表示されたら、［OK］ボタンをクリックします。

以上でアタッチが完了です。これにより、次のように［データベース］フォルダへ AdventureWorks データベースが追加されて、利用できるようになります。



1.4　サンプル スクリプトについて

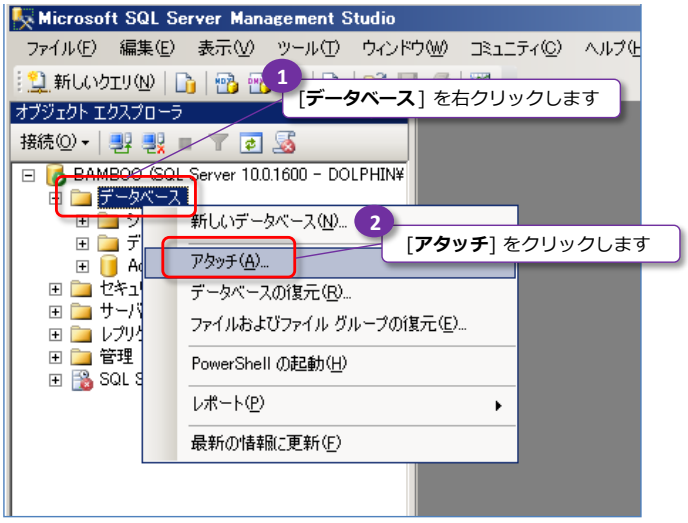
サンプル スクリプト（この自習書内で利用します）

この自習書では、いくつかの機能を試す手順でサンプル スクリプト内に含まれる「**NorthwindJ**」データベース（NorthwindJ.mdf と NorthwindJ.ldf）を利用しますので、STEP2 以降を始める前に、必ずこのデータベースを SQL Server 2008 へアタッチしておいてください。アタッチの手順は、次のとおりです。

1. ［スタート］メニューの［すべてのプログラム］から、［Microsoft SQL Server 2008］を選択して［SQL Server Management Studio］をクリックし、Management Studio を起動します。

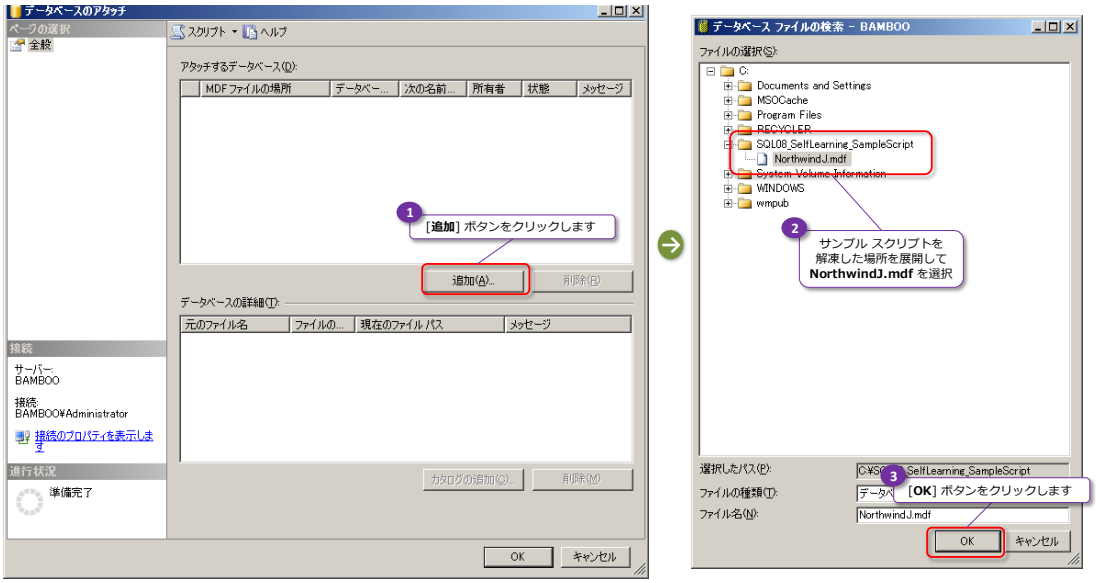
2. 起動後、［**サーバーへの接続**］ダイアログで、［**サーバー名**］へ SQL Server の名前を入力し、［**接続**］ボタンをクリックします。

3. 接続完了後、次のように［**データベース**］フォルダを右クリックして［**アタッチ**］をクリックします。



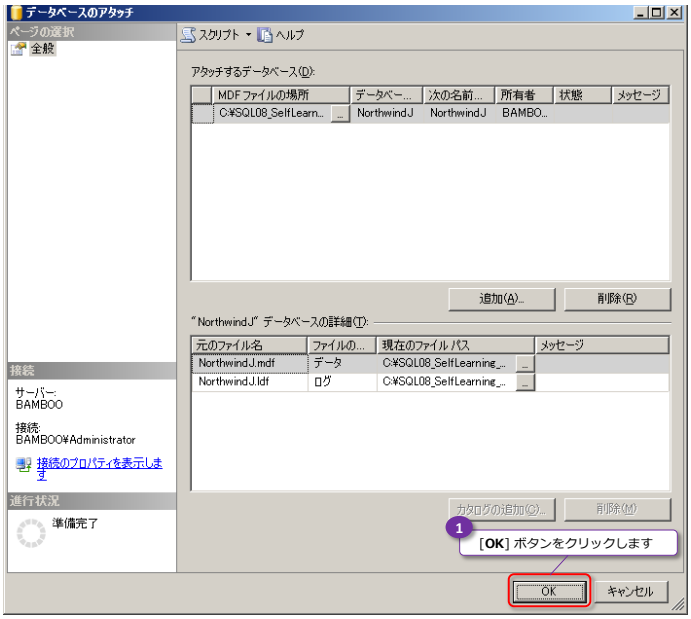
4. すると、次のように［データベースのアタッチ］ダイアログが表示されるので、［**追加**］ボタ

ンをクリックします。



サンプル スクリプトを解凍したフォルダを展開し、「**NorthwindJ.mdf**」ファイルを選択して［OK］ボタンをクリックします。

5. すると、次のように［データベースのアタッチ］ダイアログへ戻るので、［OK］ボタンをクリックします。



6. 以上でアタッチが完了です。もし、エラーが出る場合は、SQL Server のサービス アカウントに対して、「NorthwindJ.mdf」ファイルへの NTFS アクセス権限が付与されているかどうかを確認してみてください。

なお、この NorthwindJ は、Microsoft Access 2003 に付属のサンプル データベース「Northwind」を SQL Server 上へアップサイズしたものを利用していますが、受注年月日など一部のデータを加工しています。

STEP 2. Management Studio の新機能を試してみよう！

この STEP では、SQL Server 2008 で実装された Management Studio の次の新機能を試してみましょう。

- ✓ インテリセンス（IntelliSense）による入力補完
- ✓ 文法エラーの表示機能
- ✓ TOP 1000 クエリ

2.1　インテリセンス（IntelliSense）による入力補完

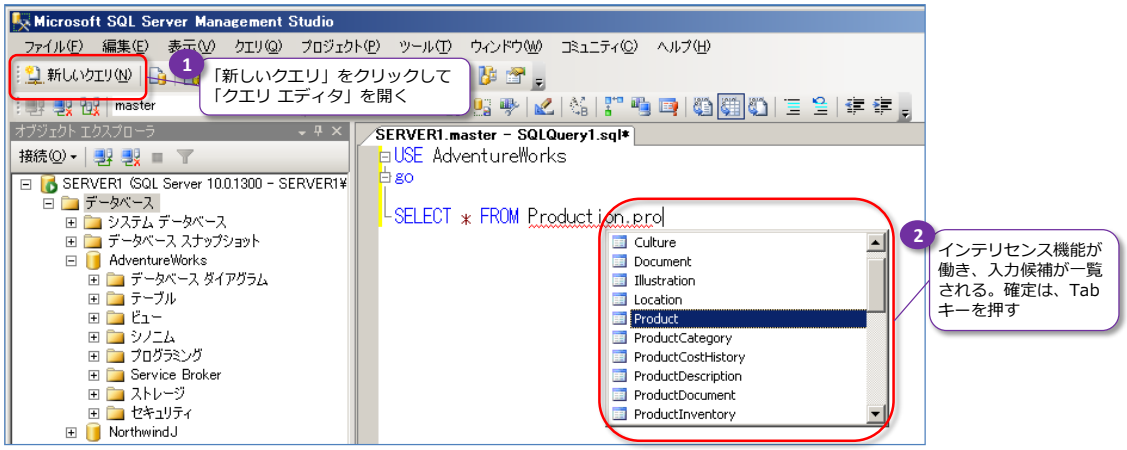
インテリセンスによる入力補完

インテリセンス（IntelliSense）は、Microsoft Visual Studio でおなじみの入力補完機能です。SQL Server 2008 では、Management Studio の「クエリ エディタ」でインテリセンス機能がサポートされるようになり、SQL ステートメントの入力時に、テーブルの一覧を表示したり、列名の一覧を表示したりできるようになりました。

それでは、これを試してみましょう。

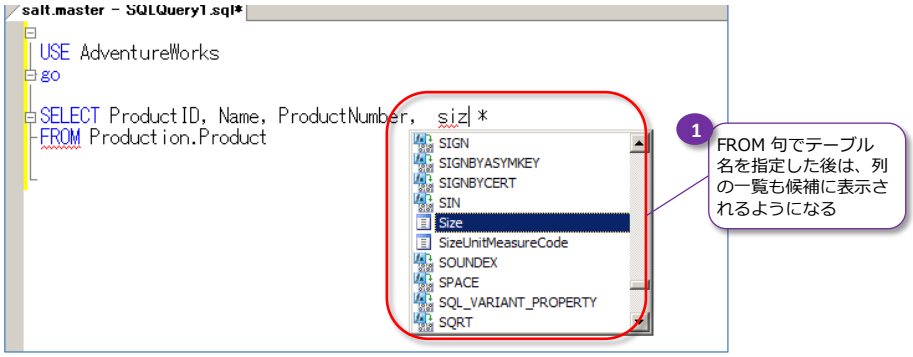
1. Management Studio を起動して、ツールバーの [**新しいクエリ**] をクリックして「クエリ エディタ」を開き、次のように入力してみてください。

```
USE AdventureWorks
go
SELECT * FROM Production.
```



FROM 句でスキーマ名（Production）の後に **.（ドット）** を入力するとインテリセンス機能が働き、入力候補が表示されることを確認できます（上の画面では、AdventureWorks データベース内の Production スキーマが所有するテーブルの一覧が表示されています）。この一覧から任意のテーブル（Product など）を選択して「**Tab**」キーで確定します。

2. FROM 句でテーブルを指定した後は、次のように選択リストでそのテーブルの列名が入力候補として一覧表示されるようになります。



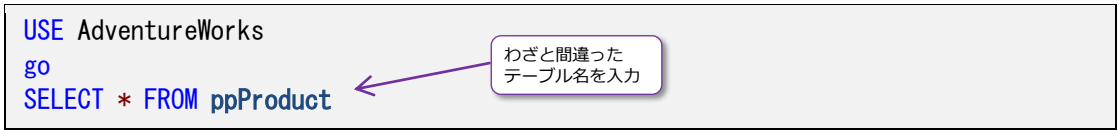
2.2　文法エラーの表示機能

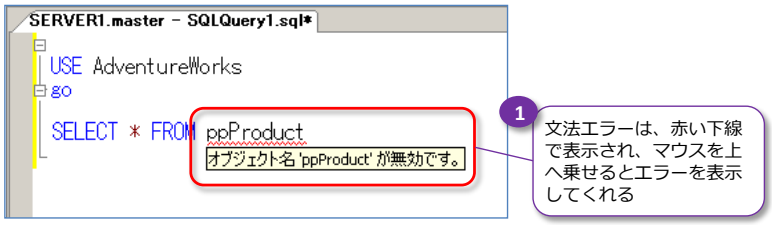
文法エラーの表示機能

この機能も Visual Studio でおなじみの文法チェック機能です。SQL Server 2008 では、SQL ステートメントの入力持に、即時に文法（構文）をチェックし、文法エラーがある場所は赤い下線で表示してくれるようになりました。また、「**エラー一覧**」(Error List) ウィンドウもサポートされ、このウィンドウを利用すると、まとめてエラー情報を確認することもできます。従来のバージョンでは、SQL ステートメントを実行しないと文法エラーを確認できなかったところが、この機能のおかげで、SQL ステートメントを実行しなくても文法エラーを確認できるようになったので大変便利です。

それでは、これを試してみましょう。

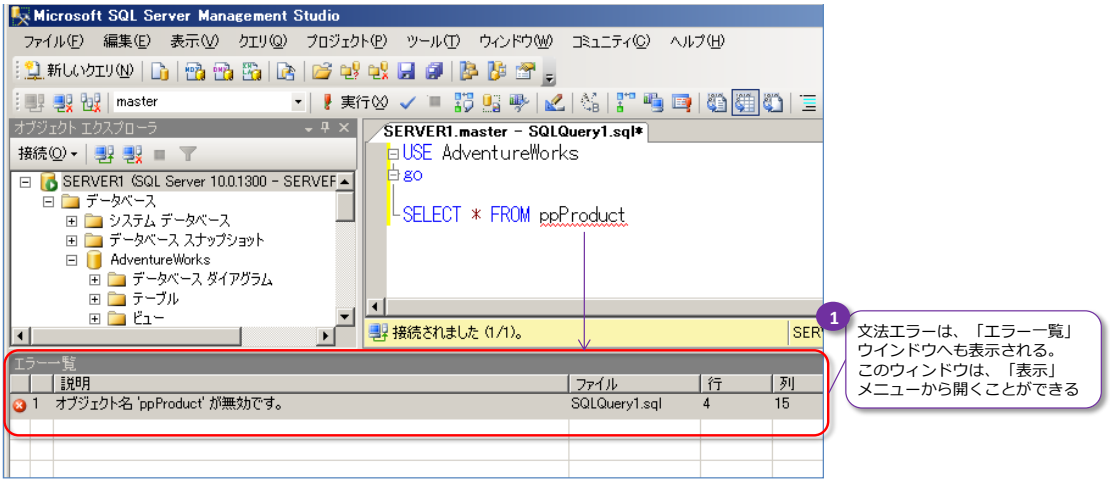
1. クエリ エディタで、次のように記述してみます。





AdventureWorks には「ppProduct」というテーブルは存在しないので、赤い下線が表示されて、文法エラーが発生していることを確認できます。

2. また、[**表示**] メニューから [**エラー一覧**] をクリックすると、次のように「エラー一覧」ウィンドウを表示することができます。



エラー一覧ウィンドウでは、複数の文法エラーをまとめて確認することができます。

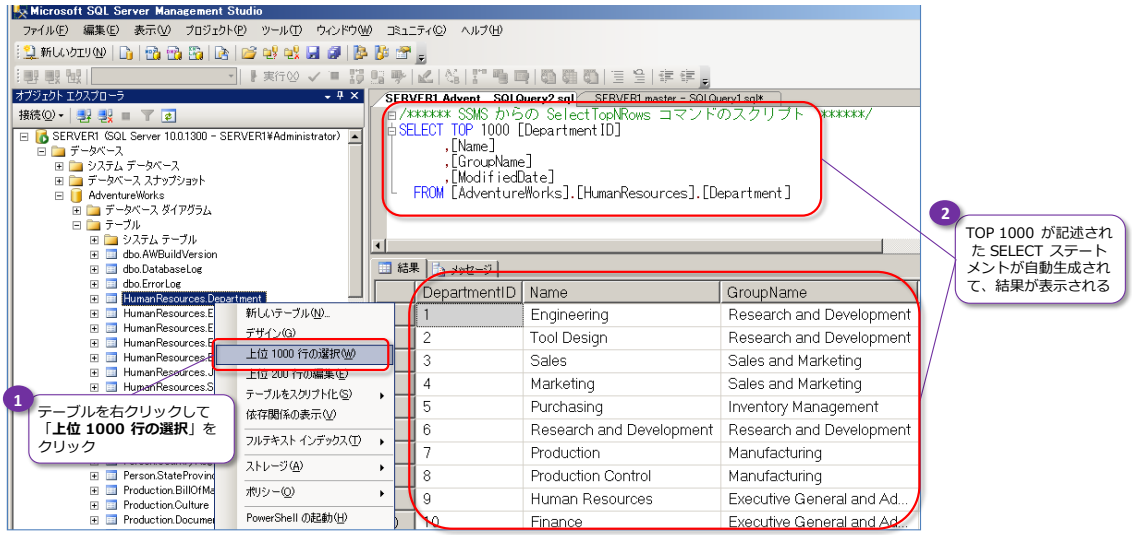
2.3　TOP 1000 クエリ

TOP 1000 クエリ

TOP 1000 クエリは、オブジェクト エクスプローラで選択したテーブルに対して TOP 1000（最初の 1000 件）を指定した SELECT ステートメントを自動生成して、その結果を取得してくれる機能です。テーブルのデータを確認したい場合にとても便利な機能です。

それでは、これを試してみましょう。

1. オブジェクト エクスプローラで、データを確認したいテーブルを右クリックして、［**上位 1000 行の選択**］をクリックします。



このように、結果（データ）を確認できるだけでなく、列名付きの SELECT ステートメントも自動生成されるので、ほかの検索条件を追加したり、特定の列データのみを取得したりする場合にも便利です。

STEP 3. Transact-SQL の新機能を試してみよう！

この STEP では、SQL Server 2008 で実装された Transact-SQL ステートメントの新機能を試してみましょう。

この STEP では、次のことを学習します。

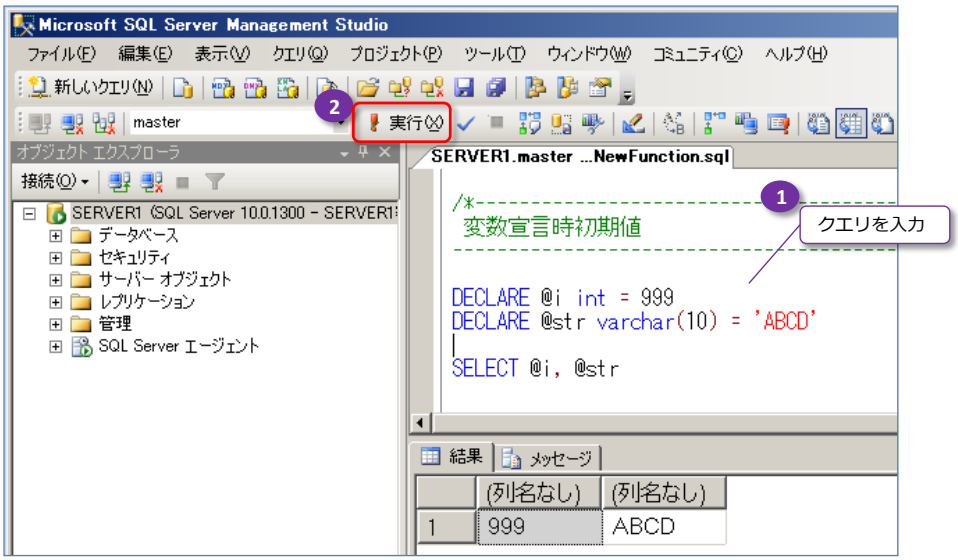
- ✓ DECLARE 変数宣言時の初期値代入
- ✓ インクリメント演算子（+=）のサポート
- ✓ INSERT ステートメントで複数の値を指定
- ✓ 新しい日付データ型（date / time / datetime2 / datetimeoffset）
- ✓ MERGE（UPSERT）
- ✓ GROUPING SETS
- ✓ テーブル値パラメータ（Table-Valued Parameters）
- ✓ ユーザー定義テーブル型（User-Defined Table Type ）
- ✓ HierarchyID データ型
- ✓ オブジェクトの依存関係の表示（dm_sql_referencing_entities）

3.1　DECLARE 変数宣言時の初期値代入

変数宣言時の初期値代入

SQL Server 2008 では、DECLARE ステートメントを使用して変数を宣言する際に、初期値を代入できるようになりました。これは　Management Studio のクエリ エディタから次のように入力して試すことができます。

```
DECLARE @i int = 999
DECLARE @str varchar(10) = 'ABCD'
SELECT @i, @str
```



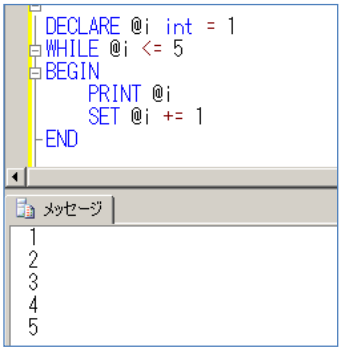
このように変数宣言時に、データ型に続けて「=」を記述できるようになり、変数へ初期値を代入できるようになりました。

3.2　インクリメント演算子（+=）のサポート

インクリメント演算子

SQL Server 2008 では、**+=** を利用したインクリメント演算子がサポートされました。これは、次のように試すことができます。

```
DECLARE @i int = 1
WHILE @i <= 5
BEGIN
        PRINT @i
        SET @i += 1
END
```

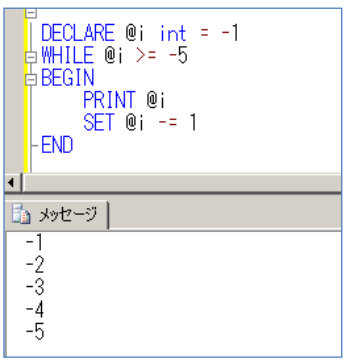


この「SET @i += 1」という記述は、「SET @i = @i + 1」と等価です。

デクリメント演算子（-=）

SQL Server 2008 では、デクリメント演算子の **-=** もサポートされました。これは、次のように利用することができます。

```
DECLARE @i int = -1
WHILE @i >= -5
BEGIN
        PRINT @i
        SET @i -= 1      -- SET @i = @i - 1 と等価
END
```

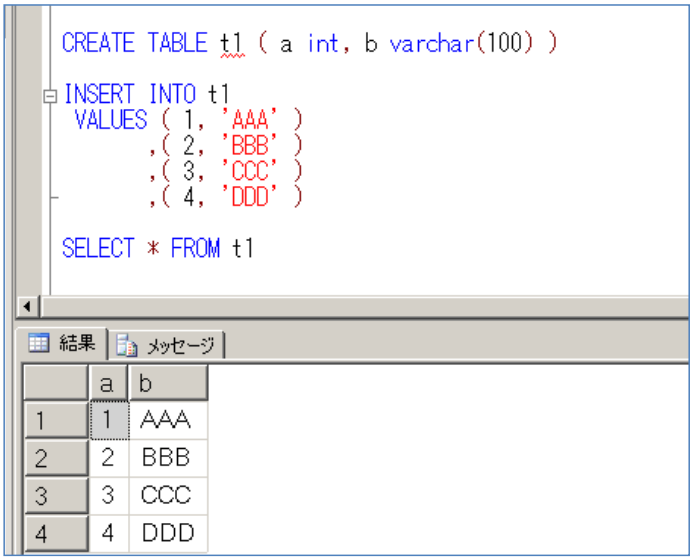


3.3　INSERT ステートメントで複数の値を指定

複数行をまとめて INSERT

SQL Server 2008 では、複数の値（複数の行データ）を 1 つの INSERT ステートメントで記述できるようになりました。これは次のように入力して試すことができます。

```
CREATE TABLE t1 ( a int, b varchar(100) )

INSERT INTO t1
 VALUES ( 1, 'AAA' )
           ,( 2, 'BBB' )
           ,( 3, 'CCC' )
           ,( 4, 'DDD' )

SELECT * FROM t1
```



このように VALUES 句で **,（カンマ）**を入れることで、複数の行データを指定できるようになりました。

3.4　新しい日付データ型（date / time / datetime2 / datetimeoffset）

新しい日付データ型

SQL Server 2008 では、新しい日付データ型として、date / time / datetime2 / datetimeoffset の 4 つが追加されました。それぞれの単位（精度）と内部的に使用するバイト数を、従来のデータ型（datetime / smalldatetime）と比較すると、次の表のようになります。

データ型	単位	使用バイト数	範囲
datetime	3.33 ミリ秒	8 バイト	1753-1-1 ～9999-12-31
smalldatetime	1分	4 バイト	1900-1-1 ～2079-6-6
date	日	3 バイト	1-1-1 ～9999-12-31
time	100 ナノ秒	time(7) は 5 バイト	00:00:00 ～23:59:59.9999999
datetime2	100 ナノ秒	datettime2(7) は 8 バイト	1-1-1 ～9999-12-31
datetimeoffset	100 ナノ秒	datettimeoffset(7) は 10 バイト	1-1-1 ～9999-12-31 +/-14:00

date データ型は、「日」単位でデータを格納できるデータ型で、使用バイト数を 3 バイトに抑えることができるので、時間を格納する必要のないデータの場合には、大変便利なデータ型です。

time データ型は、「時間」のみを格納できるデータ型で、従来の datetime データ型よりも精度が高い時間（100 ナノ秒）まで格納できるのが特徴です。

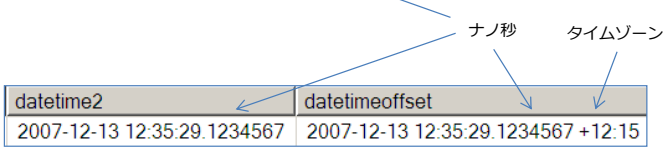
datetime2 データ型は、time データ型と同様、時間を 100 ナノ秒まで格納でき、かつ従来の datetime データ型のように日付も格納できるのが特徴です。

datetimeoffset データ型は、datetime2 データ型の格納範囲に加えて、タイムゾーンのオフセット（グリニッジ標準時からの時間差）を格納できるようになったデータ型です。

では、これらのデータ型を試してみましょう。次のように入力して試すことができます。

```
SELECT
    CAST('2007-12-13 12:35:29.123' AS datetime) AS 'datetime'
   ,CAST('2007-12-13 12:35:29.123' AS smalldatetime) AS 'smalldatetime'
   ,CAST('2007-12-13 12:35:29.1234567 +12:15' AS date) AS 'date'
   ,CAST('2007-12-13 12:35:29.1234567 +12:15' AS time(7)) AS 'time'
   ,CAST('2007-12-13 12:35:29.1234567 +12:15' AS datetime2) AS 'datetime2'
   ,CAST('2007-12-13 12:35:29.1234567 +12:15' AS datetimeoffset(7)) AS 'datetimeoffset'
```

datetime	smalldatetime	date	time
2007-12-13 12:35:29.123	2007-12-13 12:35:00	2007-12-13	12:35:29.1234567



3.5　MERGE（UPSERT）

MERGE ステートメント

MERGE ステートメントは、データが存在する場合には UPDATE を、存在しない場合には INSERT 処理が行えるステートメントです。このため、「UPSERT」とも呼ばれます（UPDATE と INSERT を組み合わせた造語）。Merge は「結合する」「吸収する」という意味の英単語です。また、MERGE ステートメントは、Oracle 9i 以降で搭載されている MERGE ステートメントと同じ構文で利用することができます。

それでは、これを試してみましょう。まずは、次のように t1 テーブルと t2 テーブルを作成します。

t1 テーブル

a	b
1	AAA
2	BBB
3	CCC
4	DDD

t2 テーブル

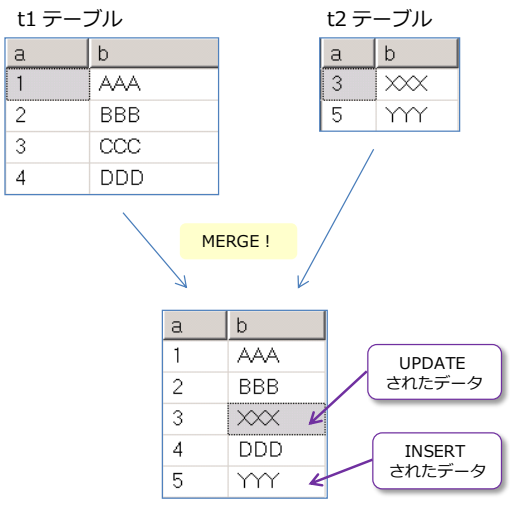
a	b
3	XXX
5	YYY

```
CREATE TABLE t1 ( a int, b varchar(100) )

INSERT INTO t1
  VALUES   ( 1, 'AAA' )
         , ( 2, 'BBB' )
         , ( 3, 'CCC' )
         , ( 4, 'DDD' )
```

```
CREATE TABLE t2 ( a int, b varchar(100) )

INSERT INTO t2
  VALUES  ( 3, 'XXX' )
        , ( 5, 'YYY' )
```

次に、t1 テーブルの「a」列と、t2 テーブルの「a」列をもとに MERGE ステートメントを実行してみましょう。

```
MERGE INTO t1
 USING t2
  ON t1.a = t2.a
 WHEN MATCHED THEN
        UPDATE SET t1.b = t2.b
 WHEN NOT MATCHED THEN
        INSERT VALUES ( t2.a, t2.b );
```



MERGE INTO へマージ(結合)先のテーブル(t1)、**USING** へマージ対象のテーブル（t2）を指定し、ON へマージの条件（ここでは 「a」列が等しいかどうか）を指定します。**WHEN MATCHED**（条件がマッチした場合）には、THEN 以下の UPDATE ステートメント（更新処理）が実行され、**NOT MATCHED**（マッチしなかった場合）には、その下の THEN 以下の INSERT ステートメント（挿入処理）が実行されるようになります。

Note： 文末のセミコロンを忘れずに

MERGE ステートメントでは、ステートメントの末尾に必ず ；（セミコロン）を記述する必要があります。セミコロンを省略した場合には、エラーになるので注意してください。

変数をもとにした MERGE

MERGE ステートメントの USING には、テーブル名だけでなく、任意のクエリを記述することができます。したがって、複数のテーブル同士の MERGE だけでなく、任意の変数の値をもとにして、MERGE ステートメントを実行することもできます。それでは、これを試してみましょう。次のように t1 テーブルと、変数 @a と @b があるとします。

t1 テーブル

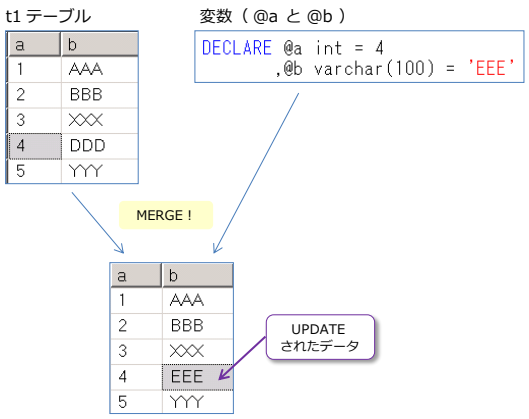
a	b
1	AAA
2	BBB
3	XXX
4	DDD
5	YYY

変数（@a と @b）

```
DECLARE @a int = 4
       ,@b varchar(100) = 'EEE'
```

これを MERGE するには、次のように記述します。

```
DECLARE @a int = 4
       ,@b varchar(100) = 'EEE'

MERGE INTO t1
 USING ( SELECT @a AS a, @b AS b ) var
  ON t1.a = var.a
 WHEN MATCHED THEN
        UPDATE SET t1.b = var.b
 WHEN NOT MATCHED THEN
        INSERT VALUES (var.a, var.b );
```



このように USING には、任意のクエリを記述できるので、変数やパラメータなど特定の値をもとに MERGE を実行することができます。

3.6 GROUPING SETS

GROUPING SETS

GROUPING SETS は、GROUP BY 句とともに利用できる集計関数で、SQL 2003 標準規格で定義されているものです。また、Oracle 10g で搭載されている GROUPING SETS とも同じように利用することができます。

GROUPING SETS は、複数の集計値を結合（UNION ALL）する場合や、WITH ROLLUP または CUBE を利用して複数項目の集計値を取得する場合の置き換えとして利用することができます。たとえば、次の UNION ALL（A 列でグループ化した集計結果と B 列でグループ化した集計結果を結合したもの）は、GROUPING SETS を利用してシンプルに記述できるようになります。

```
SELECT A, NULL AS B, SUM(n) FROM t
GROUP BY A
  UNION ALL
SELECT NULL AS A, B, SUM(n) FROM t
GROUP BY B
```

⇒

```
-- GROUPING SETS の場合
SELECT A, B, SUM(n) FROM t
GROUP BY GROUPING SETS ( (A), (B) )
```

それでは、これを試してみましょう。サンプル スクリプトに含まれる「NorthwindJ」データベース（Access 2003 でおなじみのサンプル データベースを SQL Server 2008 上へアップ サイズしたもので、セットアップ方法は 18 ページの手順を参考にしてください）を利用して、商品区分ごとの受注金額の合計と、受注年ごとの受注金額の合計を結合してみましょう。まずは、UNION ALL を次のように実行して結果を確認します。

```
SELECT 区分名, NULL AS 年, SUM( od.数量 * od.単価) AS 受注金額
FROM 受注明細 od
        INNER JOIN 商品 s ON od.商品コード = s.商品コード
        INNER JOIN 商品区分 c ON s.区分コード = c.区分コード
GROUP BY 区分名
  UNION ALL
SELECT NULL AS 区分名, YEAR(受注日), SUM( od.数量 * od.単価) AS 受注金額
FROM 受注明細 od
        INNER JOIN 受注 o ON o.受注コード = od.受注コード
GROUP BY YEAR(受注日)
```

区分名	年	受注金額
飲料	NULL	4949750.0000
加工食品	NULL	2272300.0000
菓子類	NULL	2862200.0000
魚介類	NULL	5863800.0000
穀類、シリアル	NULL	3556380.0000
調味料	NULL	4340500.0000
肉類	NULL	3522800.0000
乳製品	NULL	2453800.0000
NULL	2007	8682930.0000
NULL	2005	11599700.0000
NULL	2006	9538900.0000

商品区分ごとの受注金額の合計

受注年ごとの受注金額の合計

次に、GROUPING SETS 関数を利用して、これと同じ結果を取得してみましょう。

```
SELECT 区分名, YEAR(受注日) AS 年, SUM( od.数量 * od.単価) AS 受注金額
FROM 受注明細 od
        INNER JOIN 受注 o ON o.受注コード = od.受注コード
        INNER JOIN 商品 s ON od.商品コード = s.商品コード
        INNER JOIN 商品区分 c ON s.区分コード = c.区分コード
GROUP BY GROUPING SETS ( (区分名), (YEAR(受注日)) )
```

このように GROUPING SETS を利用すると、UNION ALL で集計値を結合するクエリをシンプルに記述できるようになります。

全体合計の追加

GROUPING SETS 関数では、次のように () を追加すると、全体合計を結果へ追加できるようになります。

```
SELECT 区分名, YEAR(受注日) AS 年, SUM( od.数量 * od.単価) AS 受注金額
FROM 受注明細 od  ～中略～
GROUP BY GROUPING SETS ( (区分名), (YEAR(受注日)) , () )
```

区分名	年	受注金額
NULL	2005	11599700.0000
NULL	2006	9538900.0000
NULL	2007	8682930.0000
NULL	NULL	29821530.0000
飲料	NULL	4949750.0000
加工食品	NULL	2272300.0000
菓子類	NULL	2862200.0000
魚介類	NULL	5863800.0000

全体合計
NULL、NULL
で表示される

WITH CUBE の置き換え

次に、WITH CUBE を置き換えとして利用可能な GROUPING SETS の使用方法をみていきましょう。たとえば、次のような WITH CUBE (A と B と C 列の組み合わせた集計結果の取得) は、GROUPING SETS を利用して次のように置き換えることができます。

```
SELECT A, B, C, SUM(n) FROM t
GROUP BY A, B, C WITH CUBE
```

⇒

```
-- GROUPING SETS の場合
SELECT A, B, C, SUM(n) FROM t
GROUP BY GROUPING SETS
   ( (A, B, C)
   , (A, B)
   , (A, C)
   , (B, C)
   , (A)
   , (B)
   , (C)
   , () )
```

それでは、これを試してみましょう。まずは、WITH CUBE を次のように実行して、商品区分と

受注年の組み合わせ集計を取得します。

```
SELECT 区分名, YEAR(受注日) AS 年, SUM( od.数量 * od.単価) AS 受注金額
FROM 受注明細 od
        INNER JOIN 受注 o ON o.受注コード = od.受注コード
        INNER JOIN 商品 s ON od.商品コード = s.商品コード
        INNER JOIN 商品区分 c ON s.区分コード = c.区分コード
GROUP BY 区分名, YEAR(受注日)
WITH CUBE
```

区分名	年	受注金額
飲料	2005	1767400.0000
加工食品	2005	1153800.0000
菓子類	2005	904000.0000
魚介類	2005	2100800.0000
穀類、シリアル	2005	1582800.0000
調味料	2005	2015400.0000
肉類	2005	1080200.0000
乳製品	2005	995300.0000
NULL	2005	11599700.0000
飲料	2006	1954600.0000
加工食品	2006	720400.0000
菓子類	2006	743800.0000
魚介類	2006	1807800.0000
穀類、シリアル	2006	1432800.0000
調味料	2006	1160900.0000
肉類	2006	964000.0000
乳製品	2006	754600.0000
NULL	2006	9538900.0000
飲料	2007	1227750.0000
加工食品	2007	398100.0000

商品区分と受注年を組み合わせた受注金額の合計

受注年ごとの受注金額の合計

乳製品	2007	703900.0000
NULL	2007	8682930.0000
NULL	NU...	29821530.0000
飲料	NU...	4949750.0000
加工食品	NU...	2272300.0000
菓子類	NU...	2862200.0000
魚介類	NU...	5863800.0000
穀類、シリアル	NU...	3556380.0000
調味料	NU...	4340500.0000
肉類	NU...	3522800.0000
乳製品	NU...	2453800.0000

全体の合計

商品区分ごとの受注金額の合計

次に、GROUPING SETS 関数を利用して、これと同じ結果を取得してみましょう。

```
SELECT 区分名, YEAR(受注日) AS 年, SUM( od.数量 * od.単価) AS 受注金額
FROM 受注明細 od
        INNER JOIN 受注 o ON o.受注コード = od.受注コード
        INNER JOIN 商品 s ON od.商品コード = s.商品コード
        INNER JOIN 商品区分 c ON s.区分コード = c.区分コード
GROUP BY GROUPING SETS (   ( 区分名, YEAR(受注日) )
                         ,(区分名)
                         ,(YEAR(受注日))
                         ,() )
```

商品区分と受注年を組み合わせた受注金額の合計

全体の合計

このように GROUPING SETS を利用すると、WITH CUBE と同じ結果を取得できるようになります。

3.7　ユーザー定義テーブル型とテーブル値パラメータ

ユーザー定義テーブル型とテーブル値パラメータ

ユーザー定義テーブル型（User-Defined Table Type）は、ユーザー定義の Table データ型に対して、名前を付けて型（Type）として保存できる機能です。また、テーブル値パラメータ（Table-Valued Parameters）は、ユーザー定義テーブル型をパラメータ（ストアド プロシージャなどの引数）として指定できる機能です。

それでは、これを試してみましょう。MERGE で利用した t1 テーブルと同じ構造のユーザー定義の Table データ型を作成し、それをパラメータ（引数）として利用するストアド プロシージャを作成します。

```
-- ユーザー定義テーブル型の作成
CREATE TYPE testTableType
  AS TABLE ( a int, b varchar(100) )
go
-- テーブル値パラメータ
CREATE PROCEDURE testProc1
  @tvp testTableType READONLY
AS
  INSERT INTO t1
    SELECT * FROM @tvp
```

@tvp（テーブル値パラメータ）に格納された値を t1 テーブルへ挿入

次に、作成したユーザー定義テーブル型へ値を代入し、それをストアド プロシージャへ渡して実行してみましょう。

```
-- ユーザー定義テーブル型を利用した変数 @tvp1 の宣言
DECLARE @tvp1 AS testTableType

-- 変数 @tvp1 へ値の格納
INSERT INTO @tvp1 (a, b)
 VALUES ( 7, 'AAAAAA')
       ,( 8, 'BBBBBB')

-- ストアド プロシージャの実行
EXEC testProc1 @tvp1

-- 結果の確認
SELECT * FROM t1
```

@tvp1 変数（testTableType 型）

a	b
7	AAAAAA
8	BBBBBB

t1 テーブル

a	b
1	AAA
2	BBB
3	XXX
4	EEE
5	YYY
7	AAAAAA
8	BBBBBB

追加されたデータの確認

IN 演算子の値リストをパラメータ化

ユーザー定義テーブル型とテーブル値パラメータは、"**配列**" のように利用することもできるので、IN 演算子の値リストをパラメータ化する目的でも利用できます。

それでは、これを試してみましょう。

```
-- ユーザー定義テーブル型
CREATE TYPE valuelist
 AS TABLE ( val int )
go
-- テーブル値パラメータを利用するストアド プロシージャ
CREATE PROCEDURE testProc2
 @v valuelist READONLY
AS
 SELECT * FROM t1
  WHERE a IN ( SELECT val FROM @v )
```

int 型の val 列

@v（テーブル値パラメータ）に格納された値を IN 演算子へ与える

次に、ユーザー定義テーブル型の val 列へ「1」と「5」、「7」を格納して、この値をストアド プロシージャの IN 演算子へ与えてみます。

```
-- ユーザー定義テーブル型を利用した変数 @v の宣言
DECLARE @v AS valuelist

-- 変数 @v へ値の格納（IN 演算子に与える値）
INSERT INTO @v (val)
 VALUES ( 1 )
       ,( 5 )
       ,( 7 )

-- ストアド プロシージャの実行
EXEC testProc2 @v
```

a	b
1	AAA
5	YYY
7	AAAAAA

このように、ユーザー定義テーブル型とテーブル値パラメータを利用すると、配列のように利用することができるので、大変便利です。なお、VB や C# などのアプリケーションからテーブル値パラメータを利用する方法は、本自習書シリーズの「開発者のための Transact-SQL 応用」で説明しています。

3.8 HierarchyID データ型

HierarchyID データ型

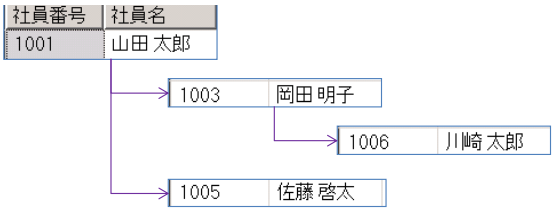
HierarchyID データ型は、階層（Hierarchy）のパスを取得 / 操作が可能なデータ型です。

それでは、これを試してみましょう。次のような親子階層をもった社員テーブル（「上司社員番号」列に上司の社員番号が格納されている）を利用します。

```
CREATE TABLE 社員
( 社員番号          int       NOT NULL,
  社員名            varchar(40) NULL,
  上司社員番号      int           NULL,
  性別              char(4)     NULL )

INSERT INTO 社員 VALUES(1001, '山田 太郎', NULL, '男性')
INSERT INTO 社員 VALUES(1002, '鈴木 一郎', NULL, '男性')
INSERT INTO 社員 VALUES(1003, '岡田 明子', 1001, '女性')
INSERT INTO 社員 VALUES(1004, '若旅 素子', 1002, '女性')
INSERT INTO 社員 VALUES(1005, '佐藤 啓太', 1001, '男性')
INSERT INTO 社員 VALUES(1006, '川崎 太郎', 1003, '男性')
```

社員番号	社員名	上司社員番号	性別
1001	山田 太郎	NULL	男性
1002	鈴木 一郎	NULL	男性
1003	岡田 明子	1001	男性
1004	若旅 素子	1002	女性
1005	佐藤 啓太	1001	男性
1006	川崎 太郎	1003	男性

この社員データには、次の階層（上司と部下）があります。



以前のバージョンの SQL Server 2005 では、このような階層データに対して、CTE（Common Table Expression：共通テーブル式）を利用して、再帰クエリを実行することで、階層レベルを取得することができました。これは次のように記述します。

```
-- SQL Server 2005 の再帰クエリの場合
WITH cte (社員番号, 社員名, 上司社員番号, 階層)
AS
(
 -- 上司
 SELECT 社員番号, 社員名, 上司社員番号, 0
  FROM 社員
   WHERE 社員番号 = 1001

         UNION ALL

 -- 部下（再帰）
 SELECT e.社員番号, e.社員名, e.上司社員番号, M.階層 + 1
  FROM 社員 AS e
   INNER JOIN cte AS M
     ON e.上司社員番号 = M.社員番号
)
SELECT * FROM cte
```

CTE による再帰クエリで、階層のレベルを取得可能

社員番号	社員名	上司社員番号	階層
1001	山田 太郎	NULL	0
1003	岡田 明子	1001	1
1005	佐藤 啓太	1001	1
1006	川崎 太郎	1003	2

SQL Server 2008 では、このクエリに対して HierarchyID データ型を利用することで、階層のパスを取得できるようになります。これは次のように記述します。

```
WITH cte (path, 社員番号, 社員名, 上司社員番号, 階層)
AS
(
 -- 上司
 SELECT HierarchyID::GetRoot() AS root
           ,社員番号, 社員名, 上司社員番号, 0
  FROM 社員
   WHERE 社員番号= 1001

         UNION ALL

 -- 部下（再帰）
 SELECT CAST(M.path.ToString()
                            + CAST(e.社員番号AS varchar(30))
                            + '/' AS HierarchyID )
          , e.社員番号, e.社員名, e.上司社員番号, M.階層+ 1
  FROM 社員AS e
   INNER JOIN cte AS M
     ON e.上司社員番号= M.社員番号
)
SELECT path.GetLevel(), path.ToString(), * FROM cte
```

(列名なし)	(列名なし)	path	社員番号	社員名	上司社員番号	階層
0	/	0x	1001	山田 太郎	NULL	0
1	/1003/	0xEE2DC0	1003	岡田 明子	1001	1
1	/1005/	0xEE2EC0	1005	佐藤 啓太	1001	1
2	/1003/1006/	0xEE2DFB8BD0	1006	川崎 太郎	1003	2

HierarchyID により階層のレベルとパスを取得可能

3.9　オブジェクトの依存関係の表示

オブジェクトの依存関係の表示

SQL Server 2008 では、オブジェクト（ビューやストアド プロシージャ、ユーザー定義関数など）の依存関係を表示するために、次の 3 つのビュー（関数）が追加されました。

- sql_expression_dependencies
- dm_sql_referencing_entities
- dm_sql_referenced_entities

これらのビューにより、ストアド プロシージャが依存しているテーブルや、ユーザー定義関数を利用しているオブジェクトを容易に把握できるようになるので、大変便利です。

それでは、これを試してみましょう。まずは、テーブルを作成し、そのテーブルに依存する（そのテーブルを利用した）ビューやストアド プロシージャ、ユーザー定義関数を作成していきます。

```
-- AdventureWorks サンプル データベースの Product テーブルをもとにテーブルを作成
SELECT * INTO Product
FROM AdventureWorks.Production.Product
go

-- ビュー prodView の作成 (Product テーブルに依存)
CREATE VIEW dbo.prodView
AS
SELECT ProductID, Name FROM dbo.Product
go

-- ユーザー定義関数 prodFunc1 の作成
CREATE FUNCTION dbo.prodFunc1(@p1 int) RETURNS int
BEGIN
 RETURN @p1 * 100
END
go

-- ストアドプロシージャ prodProc1 の作成 (prodFunc1 とprodView に依存)
CREATE PROC dbo.prodProc1
AS
SELECT dbo.prodFunc1(ProductID) FROM dbo.prodView
go

-- ストアドプロシージャ prodProc2 の作成 (prodProc1 に依存)
CREATE PROC dbo.prodProc2
AS
EXEC dbo.prodProc1
go
```

sql_expression_dependencies

次に、sql_expression_dependencies カタログ ビューを利用して、作成したオブジェクトの依存関係を表示してみます。これは次のように記述します。

```
SELECT OBJECT_NAME( referencing_id ), referenced_entity_name AS 依存元, *
 FROM sys.sql_expression_dependencies
```

(列名なし)	依存元	referencing_id	referencing_minor_id
testProc1	t1	5575058	0
testProc1	testTableType	5575058	0
prodView	Product	53575229	0
prodProc1	prodFunc1	85575343	0
prodProc1	prodView	85575343	0
prodProc2	prodProc1	101575400	0

このように sql_expression_dependencies ビューを利用すると、referenced_entity_name 列を参照することで依存元となっているオブジェクトを取得できるようになります。

dm_sql_referenced_entities

次に、dm_sql_referenced_entities 動的管理関数を利用してみます。この関数は、次のように記述することで、依存元の列名まで取得することが可能です。

```
SELECT referenced_entity_name AS 依存元, referenced_minor_name AS 依存元の列名, *
FROM sys.dm_sql_referenced_entities ('dbo.prodView', 'OBJECT')
```

依存元	依存元の列名	referencing_minor_id	referenced_server_name
Product	NULL	0	NULL
Product	ProductID	0	NULL
Product	Name	0	NULL

第 1 引数へ依存関係を調べたいオブジェクトを指定することで、referenced_minor_name 列で依存元の列名を取得することができます（この例では、prodView が Product テーブルの ProductID 列と Name 列に依存していることが分かります）。

依存関係の階層を取得

前述の sql_expression_dependencies ビューは、CTE（共通テーブル式）を利用して再帰クエリとして実行すると、依存関係の階層を取得できるようになります。これは次のように記述します（prodProd2 ストアド プロシージャに依存するオブジェクトの階層を取得するクエリ）。

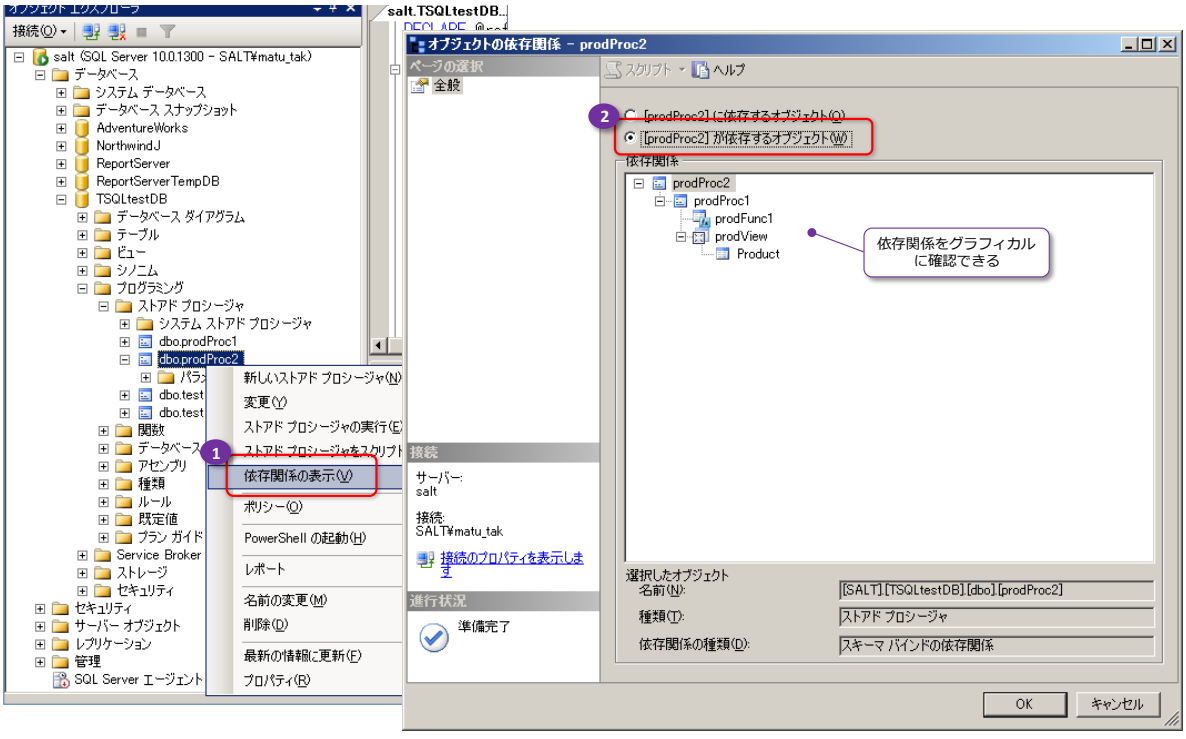
```
DECLARE @referencing_entity AS sysname = N'prodProc2';

WITH ObjectDepends(entity_name, referenced_schema, referenced_entity,
                   referenced_id, level)
AS (
    SELECT entity_name = OBJECT_NAME(referencing_id)
    ,referenced_schema_name
    ,referenced_entity_name
    ,referenced_id
    ,0 AS level
```

```
    FROM sys.sql_expression_dependencies AS sed
    WHERE OBJECT_NAME(referencing_id) = @referencing_entity
UNION ALL
    SELECT entity_name = OBJECT_NAME(sed.referencing_id)
    ,sed.referenced_schema_name
    ,sed.referenced_entity_name
    ,sed.referenced_id
    ,level + 1
    FROM ObjectDepends AS o
                INNER JOIN sys.sql_expression_dependencies AS sed
                 ON sed.referencing_id = o.referenced_id
    )
SELECT entity_name,referenced_schema, referenced_entity, level
FROM ObjectDepends
ORDER BY level;
```

entity_name	referenced_schema	referenced_entity	level
prodProc2	dbo	prodProc1	0
prodProc1	dbo	prodFunc1	1
prodProc1	dbo	prodView	1
prodView	dbo	Product	2

依存関係の階層を GUI から参照

オブジェクト間の依存関係は、Management Studio を利用してグラフィカルに確認することも可能です。これは、次のように対象オブジェクトを右クリックして［**依存関係の表示**］をクリックします。



STEP 4. SQL Server 2008 の注目の新機能を試してみよう！

この STEP では、SQL Server 2008 で実装された注目の新機能を試してみましょう。

この STEP では、次のことを学習します。

- ✓ バックアップ圧縮（Backup Compression）
- ✓ データ圧縮（Data Compression）
- ✓ 変更データ キャプチャ（CDC : Change Data Capture）
- ✓ 透過的なデータ暗号化（TDE : Transparent Data Encryption）
- ✓ SQL Server Audit（SQL Server 監査）
- ✓ データ プロファイル タスク
- ✓ データ コレクション
- ✓ リソース ガバナ
- ✓ パフォーマンス データ コレクション
- ✓ ポリシー ベースの管理
- ✓ Deprecated Features オブジェクト
- ✓ Spatial データ型
- ✓ FileStream データ型
- ✓ データ パーティション ウィザード
- ✓ マルチ サーバー クエリ

4.1　バックアップ圧縮

バックアップ圧縮（Backup Compression）

バックアップ圧縮は、その名のとおり、バックアップ データを圧縮できる機能です。圧縮することによって、バックアップおよびリストア時のデバイス（テープ装置やディスクなどのバックアップ先デバイス）への書き込み / 読み取り量（I/O）を減らすことにより、パフォーマンス向上を実現できます。

それでは、これを試してみましょう。サンプル データベースの AdventureWorks に対して、次のようにバックアップ圧縮を実行してみましょう。事前に、バックアップ先となる任意のフォルダ（**C:¥test** など）を作成してから、次のようにコマンドを実行します。

```
-- バックアップ圧縮
BACKUP DATABASE AdventureWorks
 TO DISK = 'C:¥test¥AdventureWorksComp.bak'
   WITH COMPRESSION
```

Note: バックアップ先のフォルダの権限に注意

バックアップ先のフォルダへは、SQL Server のサービス アカウントに対して NTFS の書き込み権限を付与しておかないと、エラーが発生します（サービス アカウントは、SQL Server のインストール時に設定したアカウントです）。

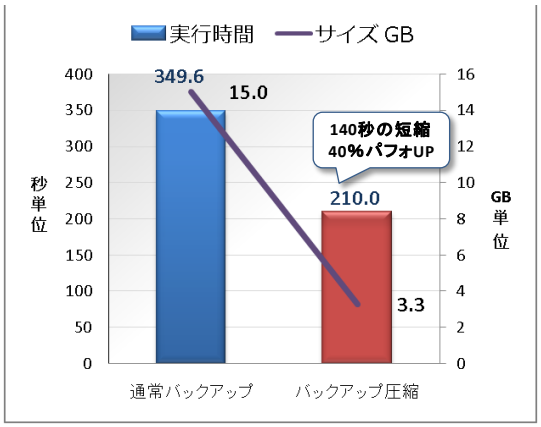
このように、通常の BACKUP ステートメントに WITH COMPRESSION を付けるだけで、バックアップ データを圧縮できるようになります。圧縮後のファイル サイズを、圧縮なしのものを比較すると次のようになります（弊社環境）。

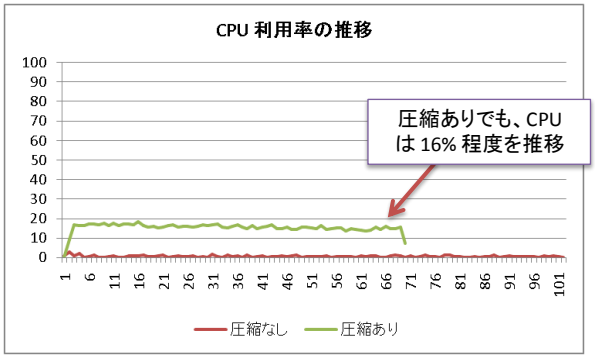
	ファイル サイズ	実行時間
圧縮なし	167 MB	17 秒
圧縮有り	38.7 MB	9秒

ファイル サイズは、4 分の 1 以下になり、実行時間は約半分になりました。このように、Backup 圧縮機能を利用すると、バックアップ パフォーマンスの向上が見込めます。

事例

次のグラフは、筆者のお客様のデータ（45GB のデータベース）を通常バックアップした場合と、Backup 圧縮を利用した場合とで比較したものです。

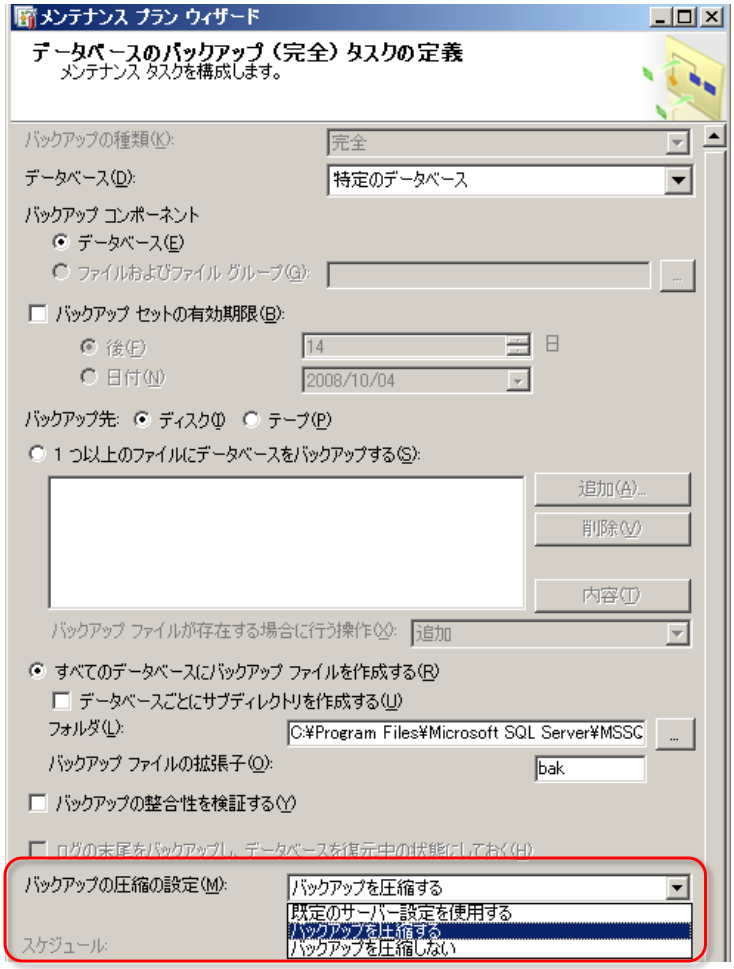




ファイル サイズは、15GB から 3.3 GB へ 4 分の 1 以下になり、実行時間は 350 秒から 210 秒へ短縮することができました。また、このときの CPU 利用率は、圧縮なしではほぼゼロ近辺を推移しているのに対し、圧縮ありでは、16%前後を推移していることを確認しています。

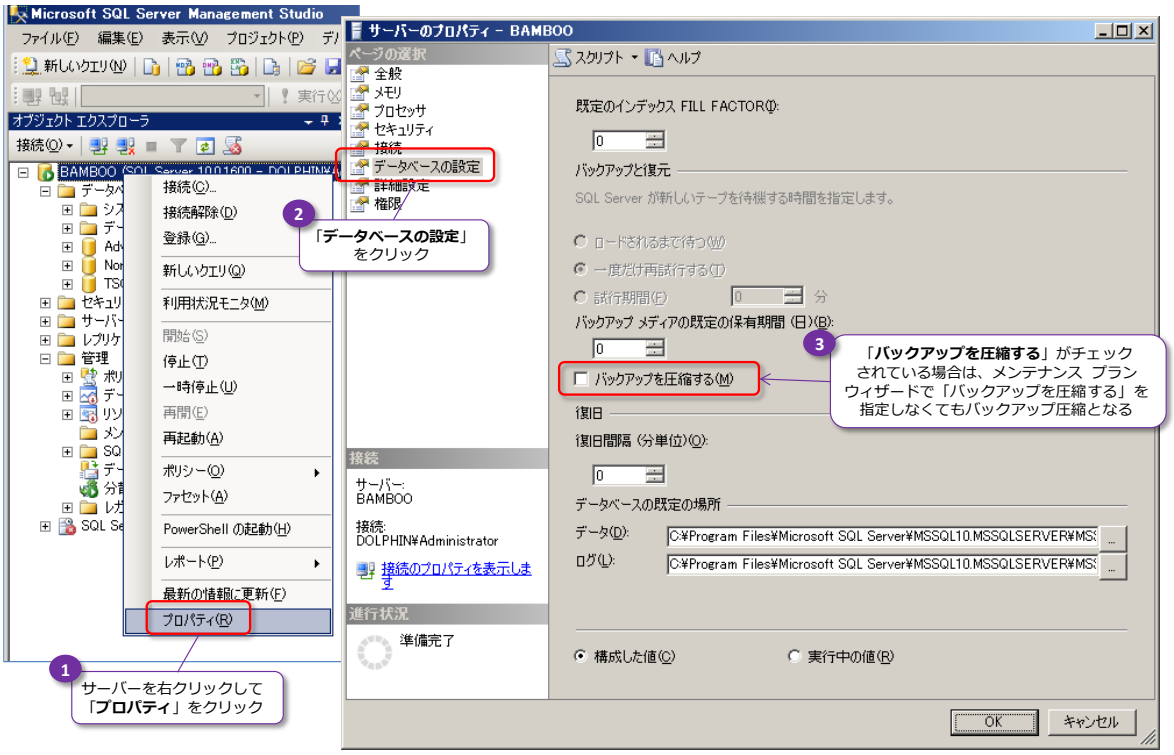
メンテナンス プラン ウィザードを利用する場合

メンテナンス プラン ウィザードを利用して Backup 圧縮を行う場合は、次のように［**バックアップの圧縮の設定**］ドロップダウン リストで「**バックアップを圧縮する**」を選択します。



なお、デフォルトは、「**既定のサーバー設定を使用する**」となっていますが、サーバー設定は、次

のように Management Studio でサーバーのプロパティを開いて、[**データベースの設定**] ページから確認できます。



バックアップ圧縮ファイルからのリストア

バックアップ圧縮したファイルからのリストアは、通常のリストアとまったく同じように実行できます。

```
RESTORE DATABASE AdventureWorks
 FROM DISK = 'C:¥test¥AdventureWorksComp.bak'
```

Note： リストアのパフォーマンスも向上

バックアップ圧縮したファイルからのリストアは、通常のバックアップからリストアするよりもパフォーマンスが向上します。これは、デバイスからの読み取り量（Read I/O）が削減されるためです。

4.2　データ圧縮

データ圧縮（Data Compression）

データ圧縮は、その名のとおり、テーブル内のデータそのものを圧縮できる機能です。圧縮することによって、ディスクへの書き込み / 読み取り量（I/O 数）を減らすことにより、パフォーマンスの向上と、ディスク コストの削減を期待できる機能です。特に、テーブル スキャンや範囲スキャンなど、大量のデータをまとめて読み取るような場合には、効果が大きいので、データ ウェアハウス環境では多いに役立つ機能です。

設定手順

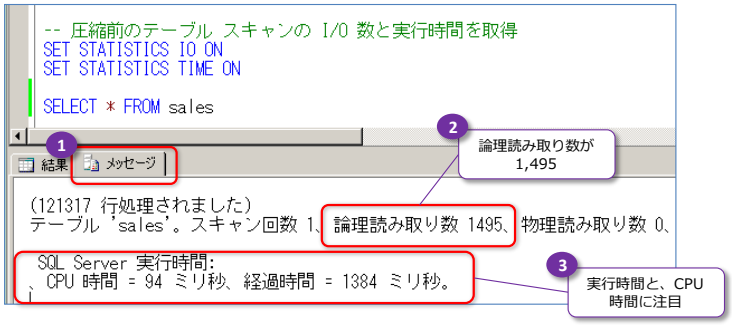
それでは、これを試してみましょう。

1. まずは、「**DataCompTestDB**」という名前のデータベースを作成し、このデータベース内へ AdventureWorks サンプル データベースの SalesOrderDetail テーブルを丸ごとコピーした「**Sales**」テーブルを作成します。

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

2. 次に、データ圧縮を行う前のテーブル スキャン時の読み取り I/O 数(ページ数)と実行時間、CPU 使用時間を調べます。

```
-- 圧縮前のテーブルスキャンの I/O 数と実行時間を取得
SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT * FROM sales
```

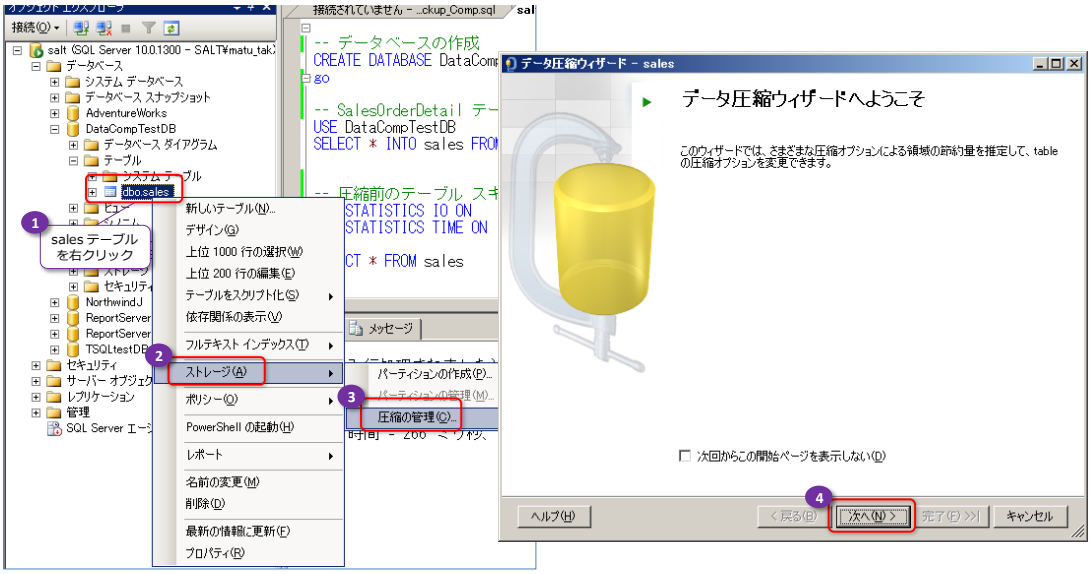


行圧縮（Row Compression）

次に、データ圧縮を行ってみましょう。データ圧縮には、行単位での圧縮が行える「**行圧縮**」（Row Compression）と、ページ単位で圧縮が行える「**ページ圧縮**」（Page Compression）の 2 種類があります。

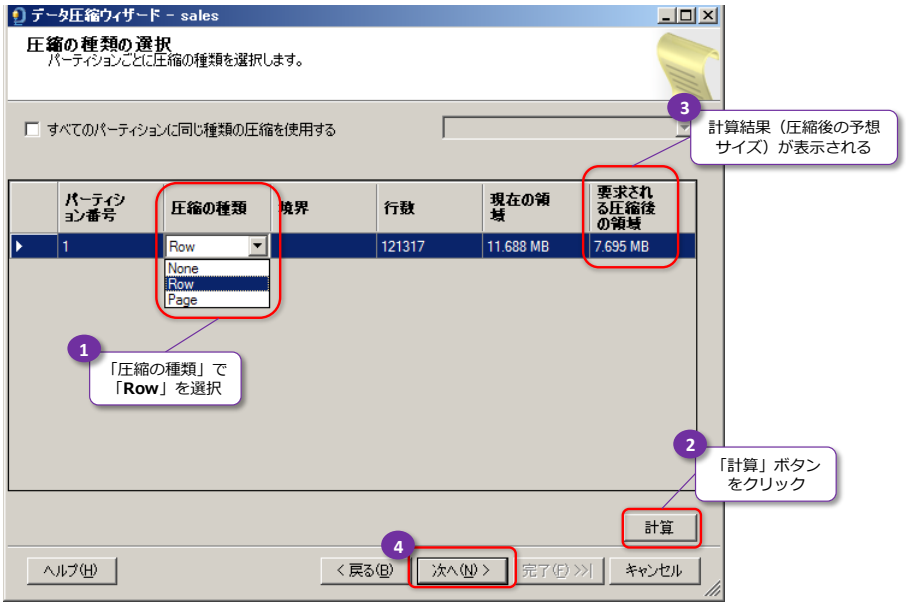
まずは、行圧縮を試してみましょう。

3. 行圧縮を行うには、次のように Management Studio で圧縮対象としたいオブジェクトを右クリックして、［**ストレージ**］メニューの［**圧縮の管理**］をクリックします。



すると、「データ圧縮ウィザード」が起動するので、［次へ］ボタンをクリックします。

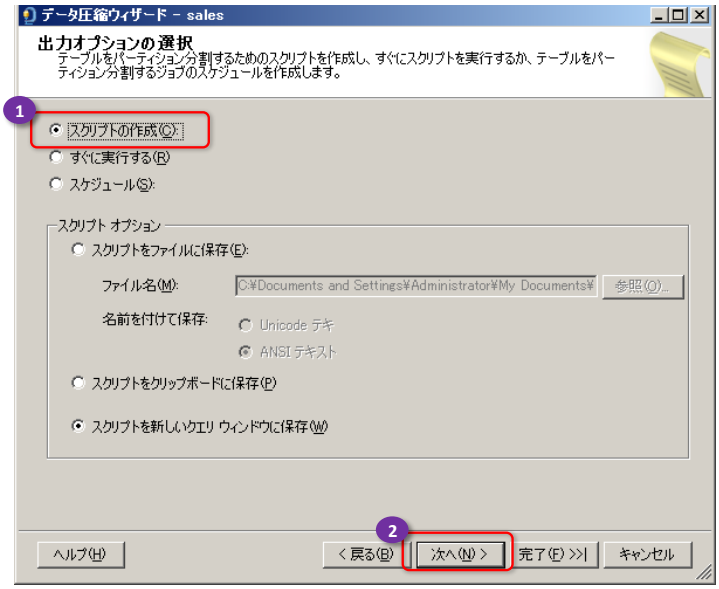
4. 次の「圧縮の種類の選択」画面では、［圧縮の種類］で「**Row**」を選択し、［**計算**］ボタンをクリックします。



これにより、行圧縮を行った場合の予想される圧縮サイズを確認することができます。確認後、

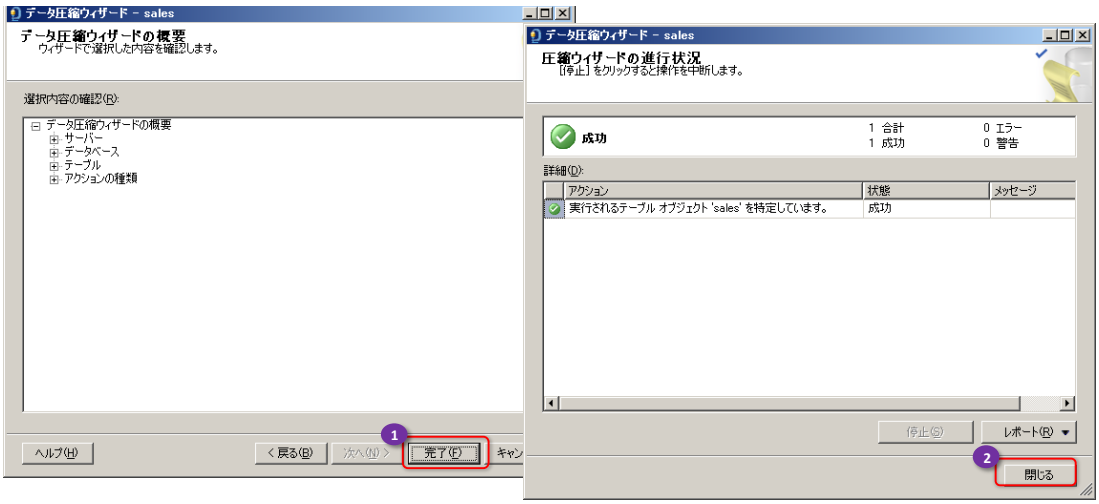
［次へ］ボタンをクリックして、次へ進みます。

5. 次の「出力オプションの選択」画面では、「**スクリプトの生成**」がチェックされていることを確認して、［次へ］ボタンをクリックします。



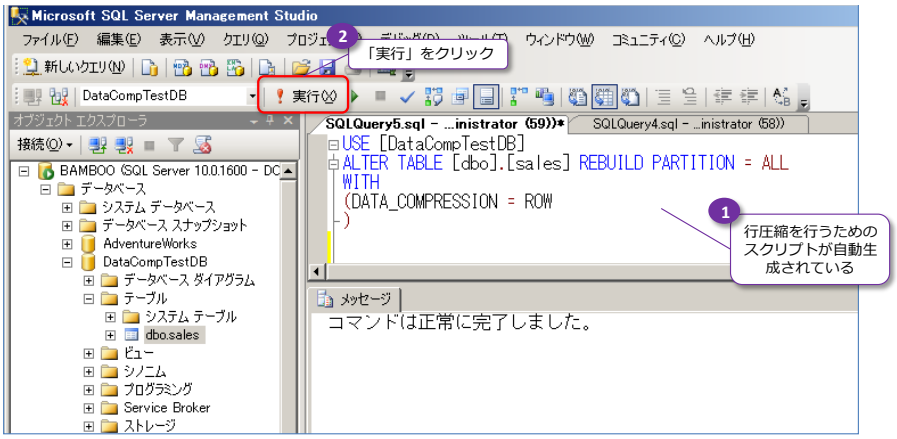
これにより、行圧縮を行うためのスクリプトを生成することができます。

6. 最後の「データ圧縮ウィザードの概要」画面では、「完了」ボタンをクリックします。



すると、進行状況画面が表示されるので、状態が「成功」と表示されることを確認してから、［閉じる］ボタンをクリックします。

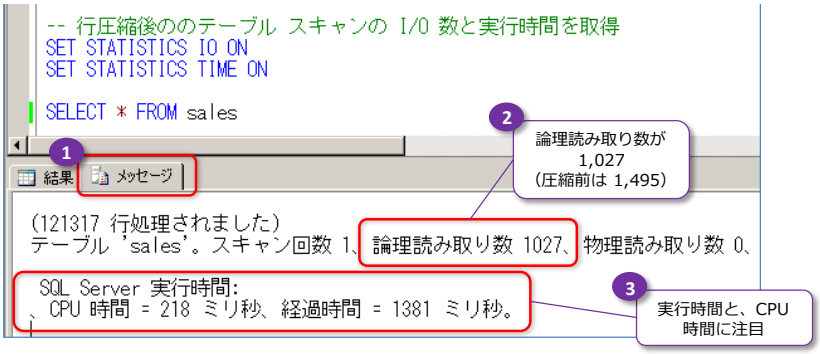
7. 以上で、行圧縮を行うためのスクリプト（ALTER TABLE ～ REBUILD ステートメント）がクエリ エディタへ自動生成されるようになります。



このステートメントは、データ パーティションを構成している場合は、パーティションごとに圧縮 / 非圧縮を設定することができるので、「**PARTITION=**」というキーワードでパーティション番号を指定できるようになっています（ALL の場合は、すべてのパーティションという意味です）。また、WITH 句の「**DATA_COMPRESSION=ROW**」が行圧縮を指定している部分です。

ステートメントを確認後、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（実際の行圧縮を実行します）。

8. 行圧縮が完了したら、手順 2 で実行したクエリと同じクエリを実行し、読み取りページ数と CPU 使用時間に注目します。



圧縮前との結果を比較すると、次のようになります。

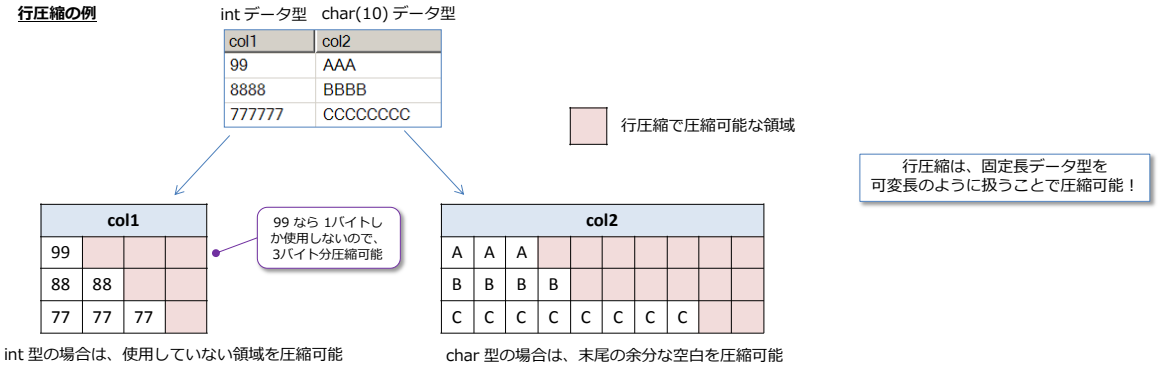
	論理読み取り数	圧縮率	実行時間	CPU 時間
圧縮なし	1495 ページ（11.7 MB）		1,384 ミリ秒	94 ミリ秒
行圧縮後	1027 ページ（8 MB）	31.3%	1,381 ミリ秒	218 ミリ秒

圧縮率は 31.3%（約 2/3 のサイズ）となり、CPU 時間は 100 ミリ秒程度余分にかかっていることを確認できます。実行時間は、今回は対象データが少ないので、差が出ていませんが、データ サイズが大きい場合には、差が出ることになります（後述）。

このように、行圧縮を行うと、読み取り / 書き込みページ数を減らすことができるので、ディスク アクセスの多い環境では、パフォーマンス向上が期待できます。

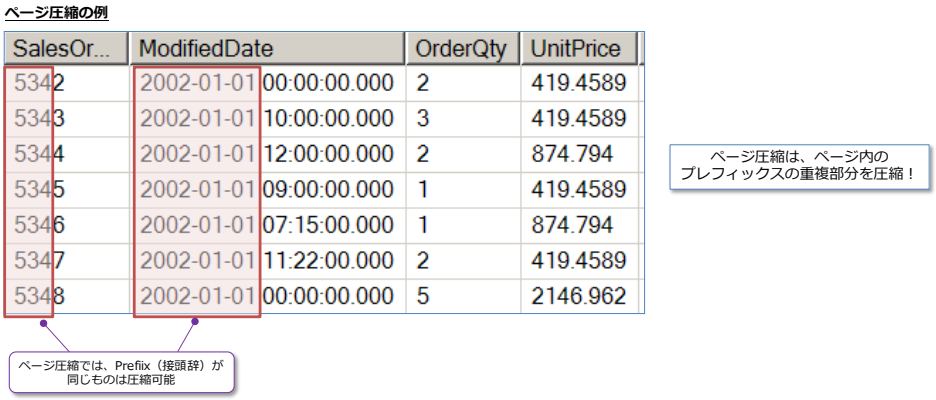
行圧縮の内部動作

行圧縮は、SQL Server 2005 の SP2（Service Pack 2）以降で提供された vardecimal データ型と同じような動作（利用していない領域を削る動作）をすることで、圧縮を行う機能です。具体的には、固定長データ型の利用していない領域を削る（可変長データ型のように扱う）ことで、サイズを小さくしています。たとえば、int データ型は、4 バイトの固定長データ型ですが、0～255 の数値を格納する場合には、1 バイトの使用領域で済むので、残りの 3 バイトを圧縮することができます。



ページ圧縮（Page Compression）

ページ圧縮は、行圧縮に加えて、さらにページ単位での圧縮を行うことで、圧縮率を高くした圧縮が可能な機能です。具体的には、ページ内での重複部分を圧縮（プレフィックスを圧縮）することで、さらにサイズを小さくすることができます。

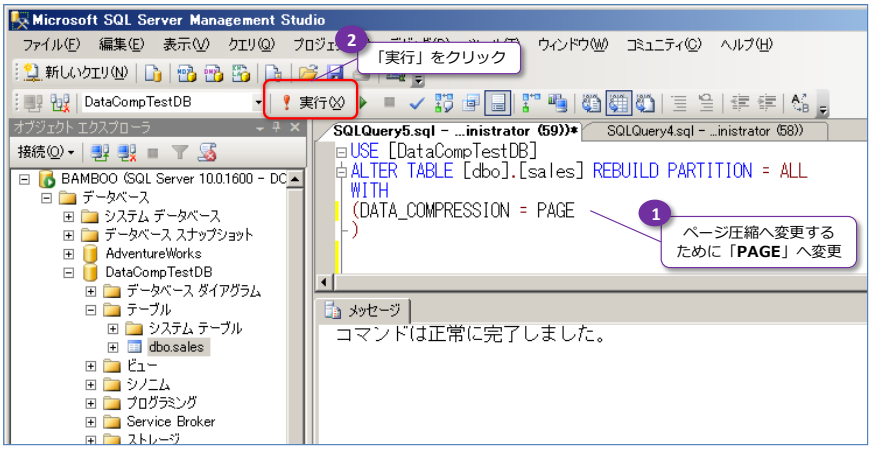


では、ページ圧縮を試してみましょう。

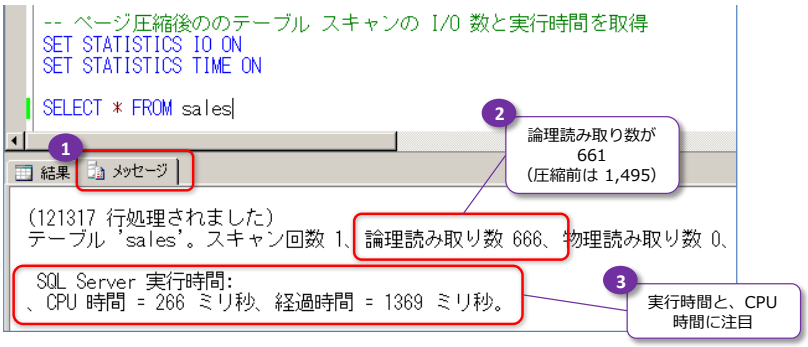
9. まずは、手順 7 で生成した ALTER TABLE ステートメントの DATA_COMPRESSION のところを **ROW** から **PAGE** へ変更して、ステートメントを実行します。

```
USE [DataCompTestDB]
ALTER TABLE [dbo] . [sales] REBUILD PARTITION = ALL
WITH
( DATA_COMPRESSION = PAGE )
```

ROW から PAGE へ変更



10. ページ圧縮が完了したら、手順 2 で実行したクエリと同じクエリを実行し、実行時間と CPU 時間、読み取りページ数に注目します。



圧縮前および行圧縮時との結果を比較すると、次のようになります。

	論理読み取り数	圧縮率	実行時間	CPU 時間
圧縮なし	1495 ページ（11.7 MB）		1,384 ミリ秒	94 ミリ秒
行圧縮後	1027 ページ（8 MB）	31.3%	1,381 ミリ秒	218 ミリ秒
ページ圧縮後	666 ページ（5.2 MB）	55.5%	1,369 ミリ秒	266 ミリ秒

圧縮率は 55.5%（半分以下のサイズ）となり、CPU 時間は、行圧縮よりもさらに余分に時間がかかっていることを確認できます。実行時間は、若干小さくなっていますが、今回は対象データが少ないので、環境によっては、ほとんど差が出なかったり、逆の結果になる場合もあります（データ サイズが大きい場合の圧縮の効果については後述しています）。

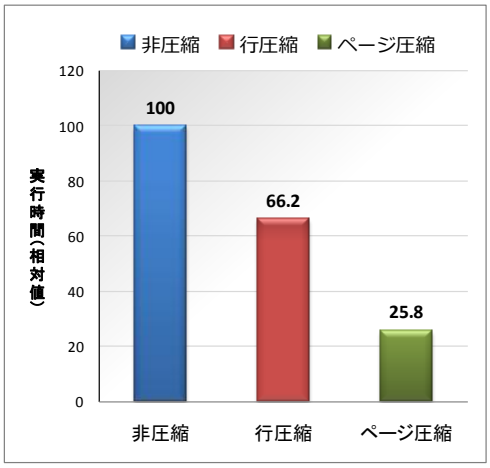
このように、ページ圧縮を行うと、行圧縮よりもさらに圧縮サイズを小さくすることができ、読み取り / 書き込みページ数を減らすことができます（その分、CPU 時間を利用することになります）。

事例（1 億件のデータで 4 倍のスピード UP、1/3 以下へ圧縮）

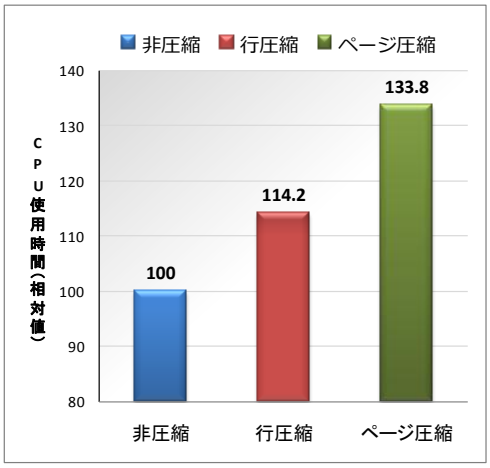
次のグラフは、筆者のお客様のデータ（約 1 億件のデータ）に対して、テーブル スキャン（全件スキャン）をともなう検索を行った場合の非圧縮時と、データ圧縮（行圧縮、ページ圧縮）時の実

行時間と CPU 時間などを比較したものです。

実行時間の比較（非圧縮時を 100 とした場合）



CPU 使用時間の比較（非圧縮時を 100 とした場合）



	論理読み取り数	圧縮率	実行時間（相対値）	CPU 時間（相対値）
圧縮なし	1,451,495 ページ		100	100
行圧縮後	985,152 ページ	32.1%	66.2	114.2
ページ圧縮後	469,984 ページ	67.6%	25.8	133.8

ページ圧縮により、67.6%の圧縮率（1/3 以下の圧縮）にすることができ、実行時間は 100 から 25.8 へ短縮（約 4 倍のスピード UP）にもなっています。また、行圧縮の場合にも 32.1%の圧縮率（約 2/3 に圧縮）、実行時間は 100 から 66.2 へ短縮（約 1.5 倍のスピード UP）という結果を得ることができました。その分、CPU パワーは、余分に使うようになるので、それとのトレード オフになります。したがって、データ ウェアハウス環境のように、ディスク I/O がボトルネックになりやすい環境では、データ圧縮機能によるパフォーマンス向上が多いに期待できます。

4.3　変更データ キャプチャ（CDC： Change Data Capture）

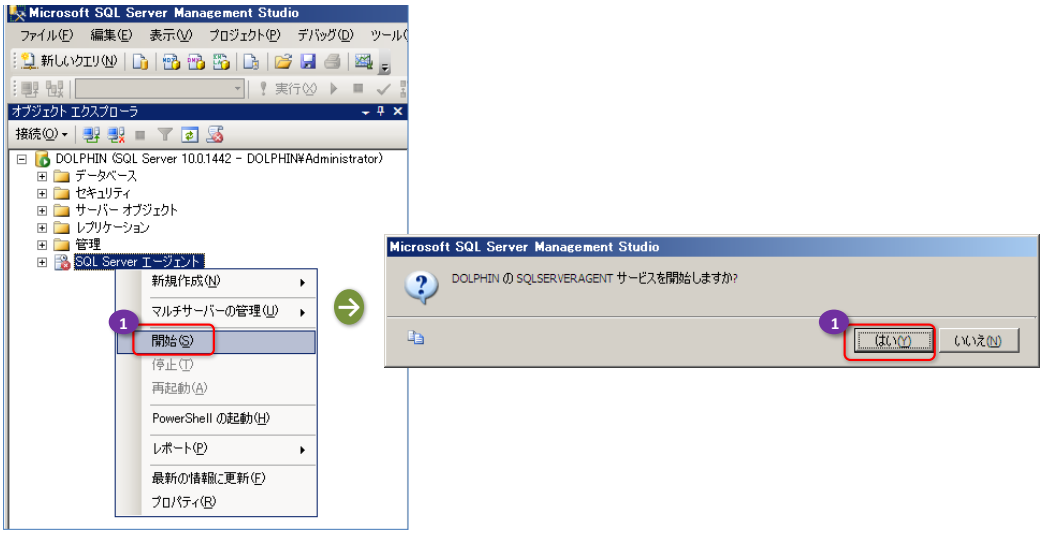
変更データ キャプチャ

変更データ キャプチャ（CDC： Change Data Capture）は、UPDATE や INSERT、DELETE ステートメントによる更新履歴を保管できる（変更データをキャプチャできる）機能です。これにより、Oracle 11g における Total Recall 機能のように、指定した時間の過去のデータを参照したり、オペレーション ミス時のデータ回復などで利用できるようになります。変更データ キャプチャは、悪意のあるユーザーによって、データが改ざんされた場合などに、改ざんされる前のデータを復旧する目的としても利用することができます。

設定手順

それでは、これを試してみましょう。

1. 変更データ キャプチャは、内部的には SQL Server Agent サービス機能を利用しているので、まずは SQL Server Agent サービスを開始しておく必要があります。SQL Server Agent サービスを開始するには、Management Studio から次のように操作します。



2. 次に、変更データ キャプチャ機能を試すために「**CDCtestDB**」という名前のデータベースを作成します。

```
CREATE DATABASE CDCtestDB
```

3. 次に、このデータベースに対して「**sp_cdc_enable_db**」ストアド プロシージャを実行して、変更データ キャプチャ（CDC）を有効に設定します。

```
USE CDCtestDB
EXEC sys.sp_cdc_enable_db
```

```
USE CDCtestDB
EXEC sys.sp_cdc_enable_db
```

メッセージ

コマンドは正常に完了しました。

4. 続いて、NorthwindJ データベースの「**商品**」テーブルをもとに、CDCtestDB データベース内へ「**商品**」テーブルを作成します。

```
USE CDCtestDB

-- 商品テーブルの作成
SELECT * INTO 商品 FROM NorthwindJ..商品

-- データの確認
SELECT * FROM 商品
```

```
-- データの確認
SELECT * FROM 商品
```

結果　メッセージ

	商品コード	フリガナ	商品名	仕入先コード	区分コード	梱包単位	単価
1	1	カジュウ100パーセント　オレンジ	果汁100%オレンジ	2	1	200g×12瓶	200.00
2	2	カジュウ100パーセント　グレープ	果汁100%グレープ	2	1	200g×12瓶	200.00
3	3	カジュウ100パーセント　レモン	果汁100%レモン	2	1	200g×12瓶	200.00
4	4	カジュウ100パーセント　ピーチ	果汁100%ピーチ	2	1	200g×12瓶	200.00
5	5	コーヒーマイルド	コーヒーマイルド	2	1	195g×10缶	190.00
6	6	コーヒービター	コーヒービター	2	1	195g×10缶	190.00
7	7	コーヒーミルク	コーヒーミルク	2	1	195g×10缶	190.00
8	8	ピリピリビール	ピリピリビール	3	1	320ml×24本	280.00
9	9	オタルシロラベル	オタル白ラベル	3	1	250ml×24本	300.00

5. 次に、作成した「商品」テーブルに対して、「**sp_cdc_enable_table**」ストアド プロシージャを実行して、CDC を有効に設定します。

```
USE CDCtestDB
go
EXEC sys.sp_cdc_enable_table
            @source_schema = N'dbo'
           ,@source_name = N'商品'
           ,@role_name = N'cdc_Admin'
```

```
EXEC sys.sp_cdc_enable_table
        @source_schema = N'dbo'
       ,@source_name = N'商品'
       ,@role_name = N'cdc_Admin'
```

メッセージ

コマンドは正常に完了しました。

以上で、変更データ キャプチャの設定が完了です。

Note： SQL Server Agent サービスを開始していない場合のエラー

もし、SQL Server Agent サービスを開始していない場合は、上記のステートメントを実行したときに、次のエラーが返されます。

```
EXEC sys.sp_cdc_enable_table
        @source_schema = N'dbo'
       ,@source_name = N'商品'
       ,@role_name = N'cdc_Admin'
```

メッセージ

```
SQLServerAgent が現在実行されていないので、この操作を通知できません。
SQLServerAgent が現在実行されていないので、この操作を通知できません。
SQLServerAgent が現在実行されていないので、この操作を通知できません。
SQLServerAgent が現在実行されていないので、この操作を通知できません。
```

必ずサービスを開始しておくようにしましょう。

なお、サービス開始後に、再度 sp_cdc_enable_table を実行したときに、次のエラーが出る場合があります。

```
EXEC sys.sp_cdc_enable_table
        @source_schema = N'dbo'
       ,@source_name = N'商品'
       ,@role_name = N'cdc_Admin'
```

メッセージ

```
メッセージ 22926、レベル 16、状態 1、プロシージャ sp_cdc_verify_capture_instance、行 36
キャプチャ インスタンスを作成できませんでした。キャプチャ インスタンス名 'dbo_商品' は
現在のデータベースに既に存在します。パラメータ @capture_instance に明示的な一意名を指定
してください。
```

この場合は、sp_cdc_disable_table を利用して、一度テーブルに対する変更データ キャプチャを無効にした後に、再度 sp_cdc_enable_table を実行してください。

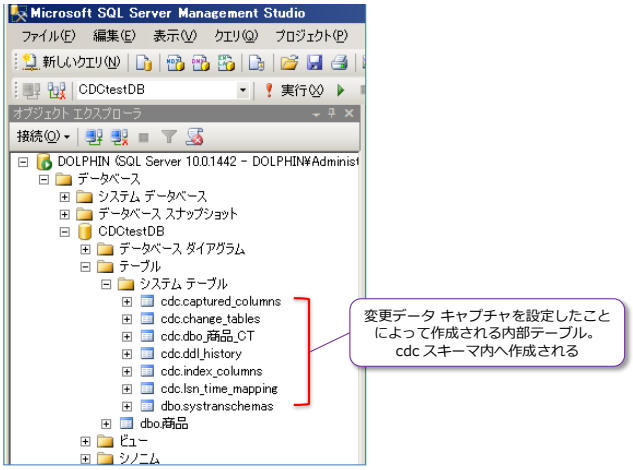
```
EXEC sys.sp_cdc_disable_table
        @source_schema = N'dbo'
       ,@source_name = N'商品'
       ,@capture_instance = 'dbo_商品'
```

メッセージ

```
コマンドは正常に完了しました。
```

更新履歴の参照（スキーマ名_テーブル名_CT テーブル）

変更データ キャプチャを有効にすると、「**dbo_商品_CT**」のように「**スキーマ名_テーブル名_CT**」という名前のテーブルが自動的に作成されて、このテーブルへ更新履歴が記録されていくようになります（スキーマについては、本自習書シリーズの「ログイン認証とオブジェクト権限」で詳しく説明しています）。



それでは、これを確認してみましょう。まずは、**UPDATE / INSERT / DELETE** ステートメントを実行して、商品テーブルのデータを更新します

```
-- データの変更
UPDATE 商品 SET 商品名 = 'XXX' WHERE 商品コード = 1

-- データの追加
INSERT INTO 商品(商品コード, 商品名, 生産中止) VALUES (199, 'AAAAAA', 1)

-- データの削除
DELETE FROM 商品 WHERE 商品コード = 2

-- データが更新されたことを確認
SELECT * FROM 商品
```

商品コード	フリガナ	商品名	仕入先コード	区分コード	梱包
199	NULL	AAAAAA	NULL	NULL	NU
1	ガジュウ100パーセント　オレンジ	XXX	2	1	200g×12瓶
3	ガジュウ100パーセント　レモン	果汁100% レモン	2	1	200g×12瓶
4	ガジュウ100パーセント　ピーチ	果汁100% ピーチ	2	1	200g×12瓶
5	コーヒーマイルド	コーヒーマイルド	2	1	195g×10缶

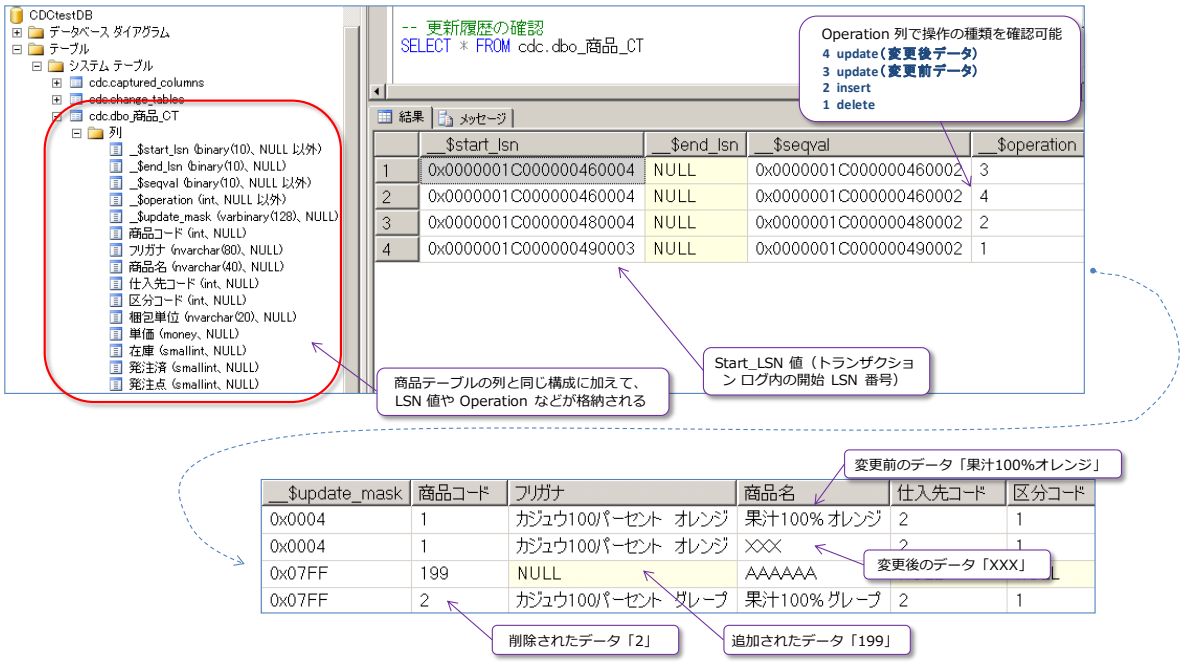
追加された商品 199

更新された商品 果汁100%オレンジ から XXX へ

商品 2 を削除

次に、更新履歴が格納されているテーブル「**dbo_商品_CT**」（**cdc** スキーマ）を参照します。

```
SELECT * FROM cdc.dbo_商品_CT
```

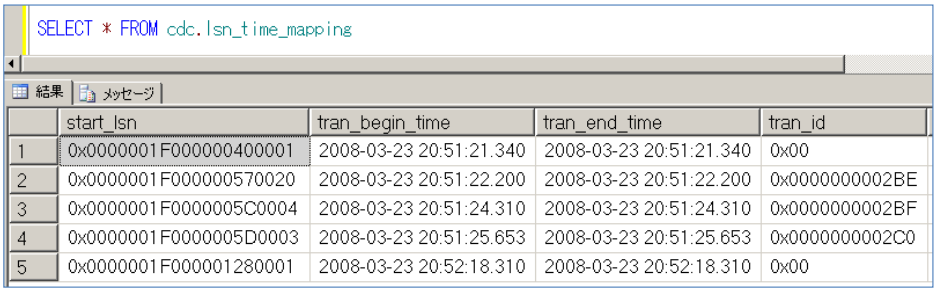


このように「**dbo_商品_CT**」テーブルの **Operation** 列を参照することで、更新前データか更新後データなのか、INSERT / DELETE されたデータなのかを確認することができます。また、変更データ キャプチャは、トランザクション ログへ記録された更新情報をもとにした機能なので、**start_lsn** と **end_lsn** など **LSN**（Log Sequence Number：ログ順序番号）によって、更新時刻（や順序）を管理しています。

時間指定による過去データの参照

変更データ キャプチャでは、特定の時間を指定して、昔のデータの参照することも簡単に行えます。変更データ キャプチャでは、「**lsn_time_mapping**」という名前のテーブルが自動的に作成されて、LSN と時刻とのマッピングが管理されているからです。これは、次のように確認できます。

```
SELECT * FROM cdc.lsn_time_mapping
```



```
SELECT * FROM cdc.lsn_time_mapping
```

結果 | メッセージ

	start_lsn	tran_begin_time	tran_end_time	tran_id
1	0x0000001F000000400001	2008-03-23 20:51:21.340	2008-03-23 20:51:21.340	0x00
2	0x0000001F000000570020	2008-03-23 20:51:22.200	2008-03-23 20:51:22.200	0x0000000002BE
3	0x0000001F0000005C0004	2008-03-23 20:51:24.310	2008-03-23 20:51:24.310	0x0000000002BF
4	0x0000001F0000005D0003	2008-03-23 20:51:25.653	2008-03-23 20:51:25.653	0x0000000002C0
5	0x0000001F000001280001	2008-03-23 20:52:18.310	2008-03-23 20:52:18.310	0x00

このマッピング テーブルは、「**fn_cdc_map_time_to_lsn**」関数を利用すると、時刻をもとに LSN 番号を取得することができるので、この LSN 番号を「**fn_cdc_get_all_changes_テーブル名**」関数へ与えるようにすると、特定の時間の範囲を指定して、昔のデータを参照できるようになります。これは次のように利用できます。

```
/*------------------------------------------------------------
 2008/03/23 20:51:22 以降から現在までの間に更新されたデータの参照
------------------------------------------------------------*/
DECLARE @begin_time datetime = '2008/03/23 20:51:22'
DECLARE @end_time   datetime = GETDATE()
DECLARE @from_lsn binary(10), @to_lsn binary(10)

-- fn_cdc_map_time_to_lsn 関数によりLSN（ログ順序番号）の取得
SELECT @from_lsn  = sys.fn_cdc_map_time_to_lsn('smallest greater than or equal', @begin_time)
SELECT @to_lsn  = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @end_time)

-- fn_cdc_get_all_changes_dbo_商品関数
SELECT * FROM cdc.fn_cdc_get_all_changes_dbo_商品(@from_lsn, @to_lsn, 'all')
```

__$start_lsn	__$seqval	__$operation	__$update_mask	商品コード	フリガナ
0x0000001F000000570020	0x0000001F00000057001E	4	0x0004	1	カジュウ100パーセント　オレンジ
0x0000001F0000005C0004	0x0000001F0000005C0002	2	0x07FF	199	NULL
0x0000001F0000005D0003	0x0000001F0000005D0002	1	0x07FF	2	カジュウ100パーセント　グレープ

4.4　透過的なデータ暗号化（TDE: Transparent Data Encryption）

透過的なデータ暗号化

透過的なデータ暗号化は、データベース内のオブジェクトをすべて暗号化できる機能です。従来のバージョンの SQL Server 2005 では、**EncryptByKey** と **DecryptByKey** を利用した暗号化機能が提供されていましたが、この機能を利用するには、アプリケーションを修正する必要がありました。そこで、SQL Server 2008 からは、アプリケーションを修正しなくてもよい（アプリケーションからは透過的に利用できる）暗号化機能として、透過的なデータ暗号化機能が提供されました。

透過的なデータ暗号化では、データベースに対して暗号化を設定するだけで、データベース内のすべてのオブジェクトを暗号化できるようになります（データ ファイル全体を暗号化することができます）。また、バックアップ ファイルも自動的に暗号化されるようになります。

これにより、データ ファイル（.mdf）やハード ディスクが盗難に遭ったり、バックアップ テープが持ち出されてたとしても、（暗号が解読されない限り）データが読み取られることはありません。

それでは、これを試してみましょう。

設定手順

1. まずは、「**enc**」という名前のデータベースを任意の場所（**C:¥** など）へ作成します（CREATE DATABASE が失敗する場合は、「C:¥」への書き込み権限を与えてください）。

```
USE master
go
CREATE DATABASE enc
ON PRIMARY
 ( NAME = 'enc',
   FILENAME = 'C:¥enc.mdf' )
```

2. 次に、**CREATE MASTER KEY ENCRYPTION** ステートメントを利用して、マスター キーを作成します。

```
USE master
go
CREATE MASTER KEY ENCRYPTION
 BY PASSWORD = '<StrongPassword>'
```

強固なパスワードを設定します

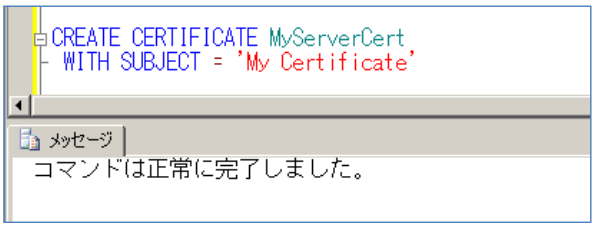
```
CREATE MASTER KEY ENCRYPTION
 BY PASSWORD = '<StrongPassword>'
```

メッセージ

コマンドは正常に完了しました。

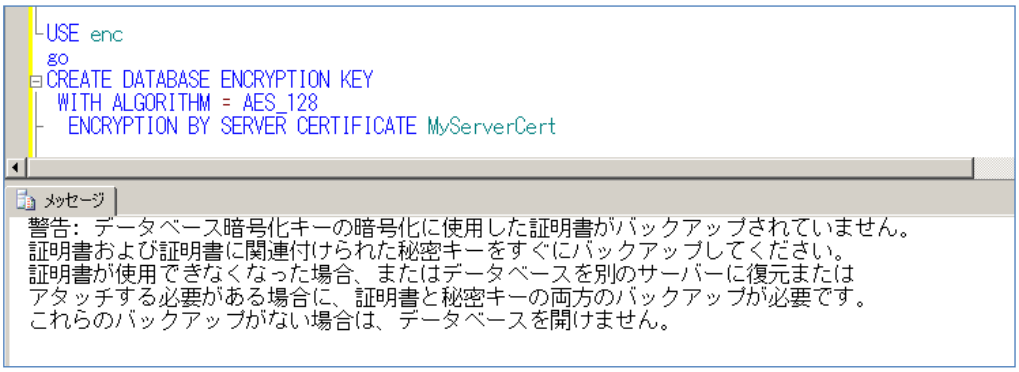
3. 次に、**CREATE CERTIFICATE** ステートメントを利用して、サーバー証明書を「**MyServerCert**」 という名前で作成します。

```
CREATE CERTIFICATE MyServerCert
 WITH SUBJECT = 'My Certificate'
```



4. 続いて、前の手順で作成したサーバー証明書を利用して、データベース暗号化キーを作成します（**CREATE DATABASE ENCRYPTION KEY** ステートメントを利用します）。暗号化アルゴリズムには、**AES_128**、**AES_192**、**AES_256**、**3DES** を選択できますが、ここでは AES_128 を指定しています。

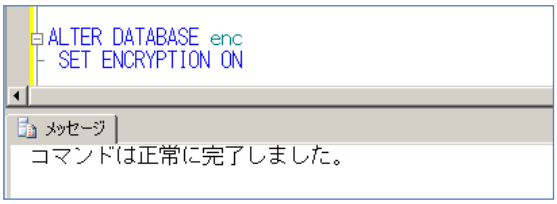
```
USE enc
go
CREATE DATABASE ENCRYPTION KEY
 WITH ALGORITHM = AES_128
  ENCRYPTION BY SERVER CERTIFICATE MyServerCert
```



実行後、証明書のバックアップに関する警告が表示されますが、これについては、後述します。

5. 次に、**ALTER DATABASE** ステートメントを利用して、データベースに対して透過的なデータ暗号化を有効化します。

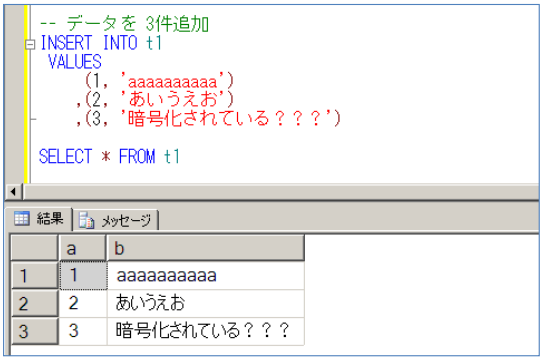
```
ALTER DATABASE enc
 SET ENCRYPTION ON
```



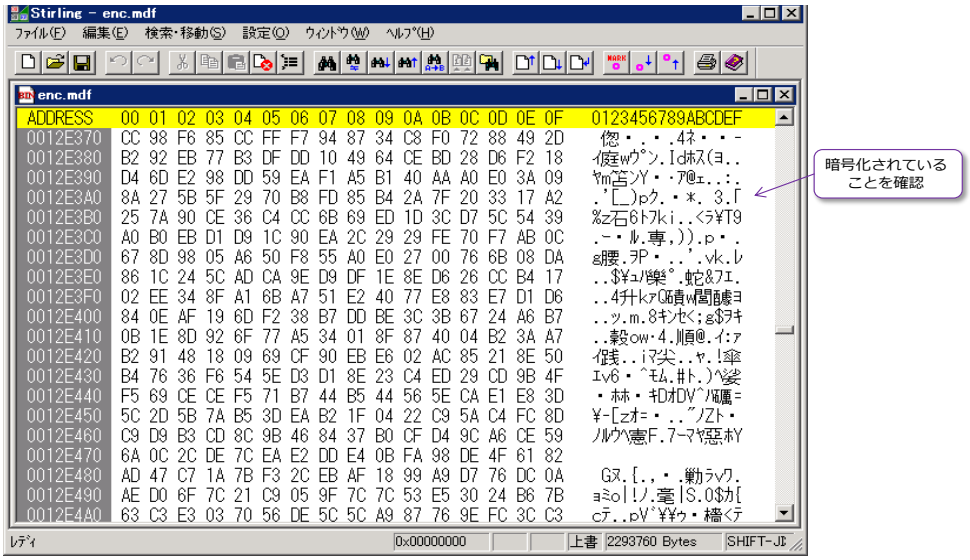
以上で設定が完了です。

次に、この「enc」データベース内へテーブルを作成して、データを追加し、バックアップを実行してみましょう。

```
USE enc
go
-- テーブルの作成
CREATE TABLE t1
( a int , b varchar(100) )

-- データを 3件追加
INSERT INTO t1
 VALUES
          (1, 'aaaaaaaaaa')
        ,(2, 'あいうえお')
        ,(3, '暗号化されている？？？')

SELECT * FROM t1

-- データベースのバックアップ
BACKUP DATABASE enc
 TO DISK = 'C:¥enc.bak'
```

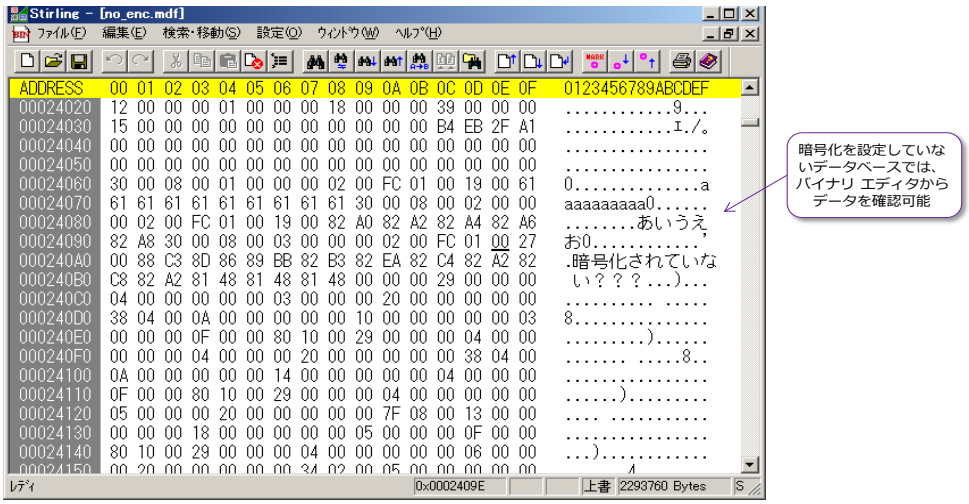


次に、正しく暗号化されたかどうかを確認するために、SQL Server サービスを停止し、停止が完了した後に、データ ファイル「C:¥**enc.mdf**」を任意のバイナリ エディタ（Stirling など）で開いてみましょう。



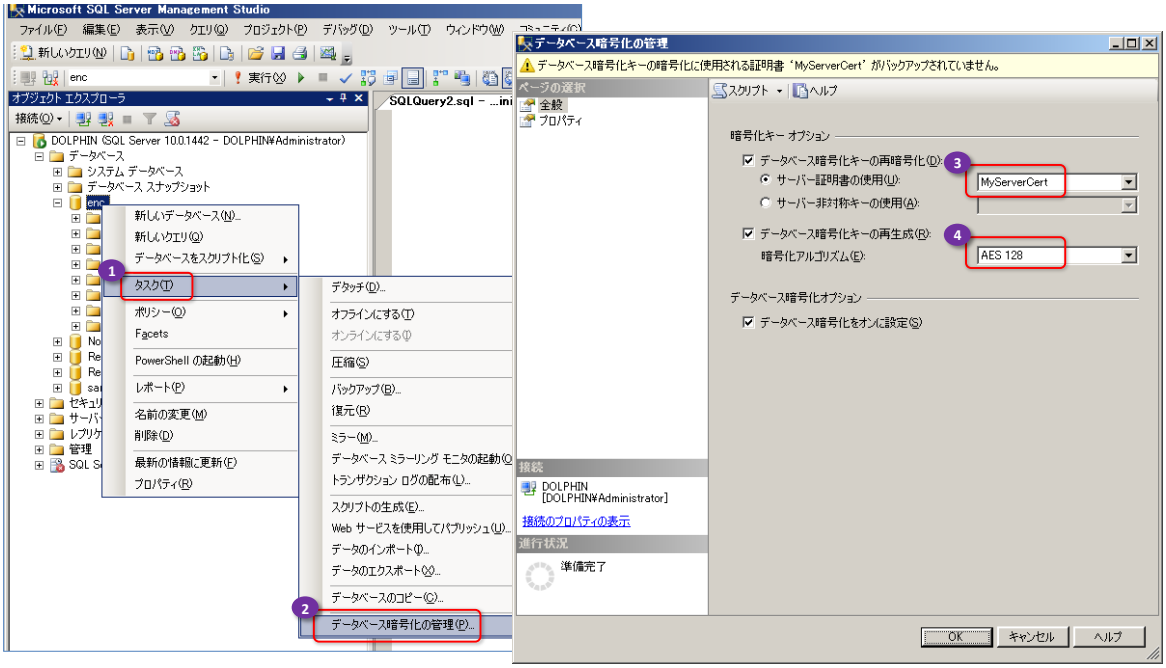
バイナリ エディタで開いても、どういったデータが格納されているかを読み取ることはできず、きちんと暗号化されていることを確認できます。同じように、バックアップ ファイル「C:¥**enc.bak**」についても暗号化がされていることを確認しておきましょう。

なお、暗号化をしていないデータベースをバイナリ エディタで開いた場合は、次のようにデータを読み取ることが可能です。



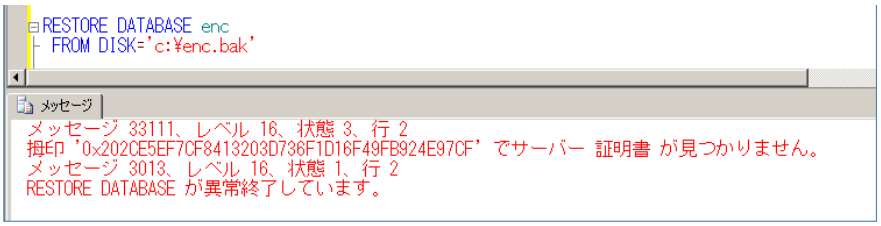
GUI での設定方法

手順 4「データベース暗号化キーの作成」と手順 5「データベース対して暗号化を有効」については、Management Studio から GUI ベースで設定することもできます。これは、次のように対象となるデータベースを右クリックして［**タスク**］メニューの［**データベース暗号化の管理**］をクリックして行えます。



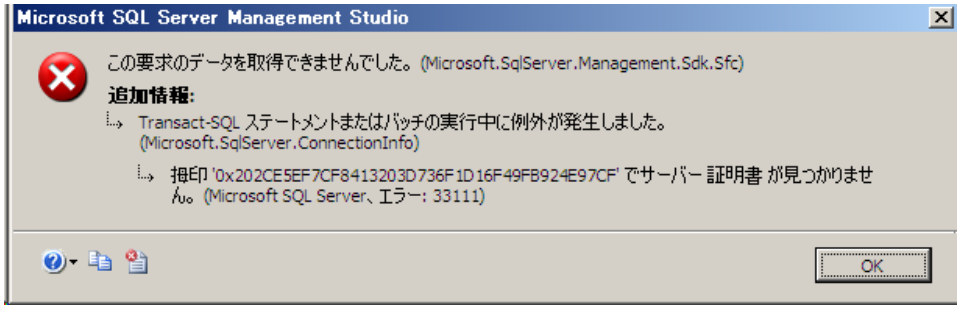
別マシンでのリストアはエラー

透過的なデータ暗号化を設定したデータベースのバックアップは、別のマシンでリストアしようとすると、次のエラーが発生します。



サーバー証明書が存在しないので、リストアができないという主旨です。このように、バックアップ ファイルが持ち出されたとしても、簡単にはリストアできないようになっていて、かつ中身が暗号化されているので、安全性が高いものとなっています。

また、データ ファイル（.mdf）を別マシンでアタッチしようとすると、同様のエラーが発生します。

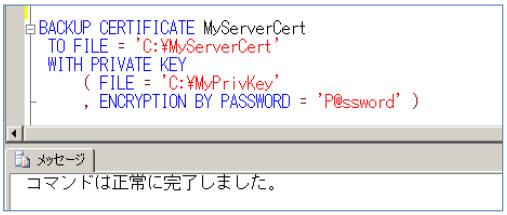


このように、データ ファイルについても、簡単にアタッチできないようになっています。

サーバー証明書のバックアップとリストア

開発機から本番機へ、あるいはその逆へなど、別マシンへデータを移動させたい場合には、サーバー証明書（と秘密キー）のバックアップを次のように実行しておく必要があります。

```
BACKUP CERTIFICATE MyServerCert
 TO FILE = 'C:¥MyServerCert'
 WITH PRIVATE KEY
         ( FILE = 'C:¥MyPrivKey'
         , ENCRYPTION BY PASSWORD = 'P@ssword' )
```

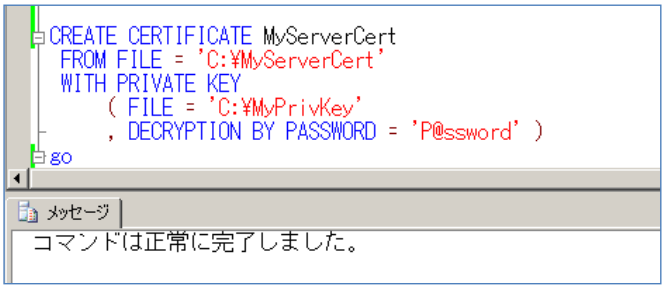


BACKUP CERTIFICATE ステートメントを利用して、ファイルへバックアップを格納し、**WITH**

PRIVATE KEY オプションを付けて、秘密キーもバックアップしておく必要があります。このときに指定するパスワードは、推測されにくい強固なパスワードにし、そのパスワードが簡単に漏れないように注意しておく必要があります。これで証明書と秘密キーをファイルへエクスポートすることができました。

次に、バックアップしたファイル（証明書と秘密キー）を別マシンへ物理的にコピーし、次のように **CREATE CERTIFICATE** ステートメントを利用して、リストア（インポート）します。

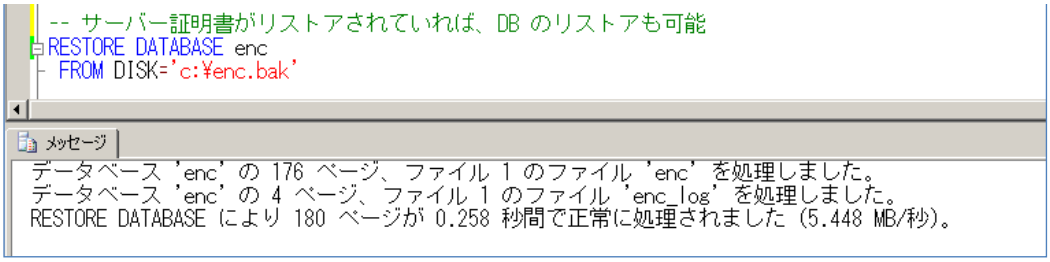
```
-- マスターキーの作成
CREATE MASTER KEY ENCRYPTION
 BY PASSWORD = '<StrongPassword>'
go
-- サーバー証明書のリストア（インポート）
CREATE CERTIFICATE MyServerCert
 FROM FILE = 'C:¥MyServerCert'
 WITH PRIVATE KEY
          ( FILE = 'C:¥MyPrivKey'
          , DECRYPTION BY PASSWORD = 'P@ssword' )
```



サーバー証明書のリストア（ファイルからの作成）には、マスターキーを作成しておく必要があるので、まずマスタ キーを作成しています。

リストア時には、複合化（DECRYPTION）のためのパスワードを、バックアップ時に指定したのと同じものを指定します。これで、サーバー証明書のリストアが完了です。

このように同じサーバー証明書がリストアされている環境であれば、透過的なデータ暗号化を設定したデータベースをリストアして、データを参照することが可能です。



4.5　SQL Server Audit（SQL Server 監査）

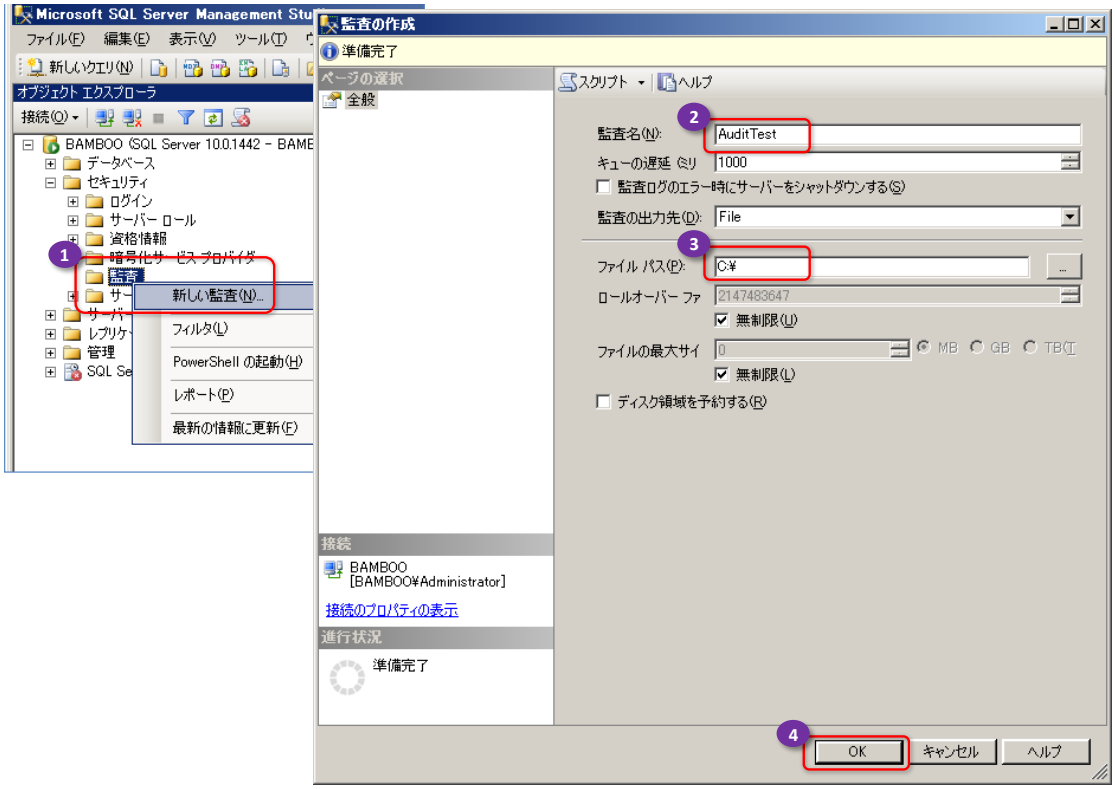
SQL Server Audit

SQL Server Audit（SQL Server 監査）機能は、SQL Server に対して発行されたすべての操作を監査（ログ記録）できる機能で、**J-SOX 法**（日本版 SOX 法）や**内部統制**などの**コンプライアンス**（**法令遵守**）を実現するために欠かせない機能になります。この機能を利用すれば、「いつ」「誰が」「どのデータベースに対して」「どんなステートメント」を実行したのかを記録したり、**特権ユーザー**（管理権限を持ったログイン アカウント）の不正利用を記録したりできるようになるので、J-SOX 法で求められる財務諸表に対する「監査（Audit）」（ログ記録による虚偽表示対策）の実現や、顧客情報の漏えい対策、内部統制の実現など、昨今求められているセキュリティ要件を実現することができます。

操作手順

それでは、SQL Server Audit を試してみましょう。

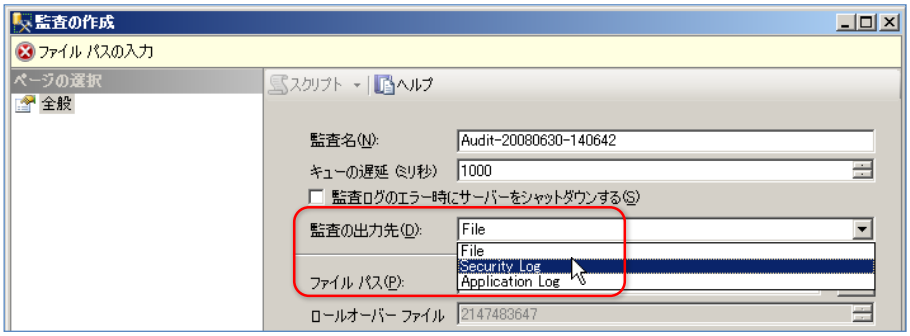
1. まずは、新しく監査を作成して、監査ログの記録場所を指定します。監査を作成するには、次のように Management Studio のオブジェクト エクスプローラで［**セキュリティ**］フォルダを展開して、［**監査**］フォルダの［**新しい監査**］をクリックします。



これにより、「監査の作成」ダイアログが表示されるので、［**監査名**］へ「**AuditTest**」など任意の名前を入力し、［**ファイル パス**］へ「**C:¥**」など任意の場所を入力して、ファイルへ監査ログが記録されるようにします。入力後、[OK]ボタンをクリックしてダイアログを閉じます。

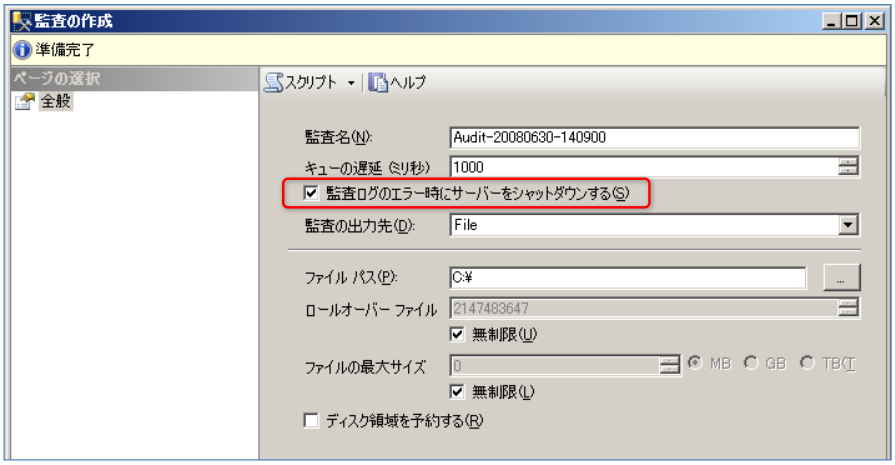
Note： ファイル以外の格納先

監査ログは、ファイルのほかに Windows の**アプリケーション ログ**または**セキュリティ ログ**へ記録することも可能です。これは、次のように［**監査の出力先**］から変更できます。

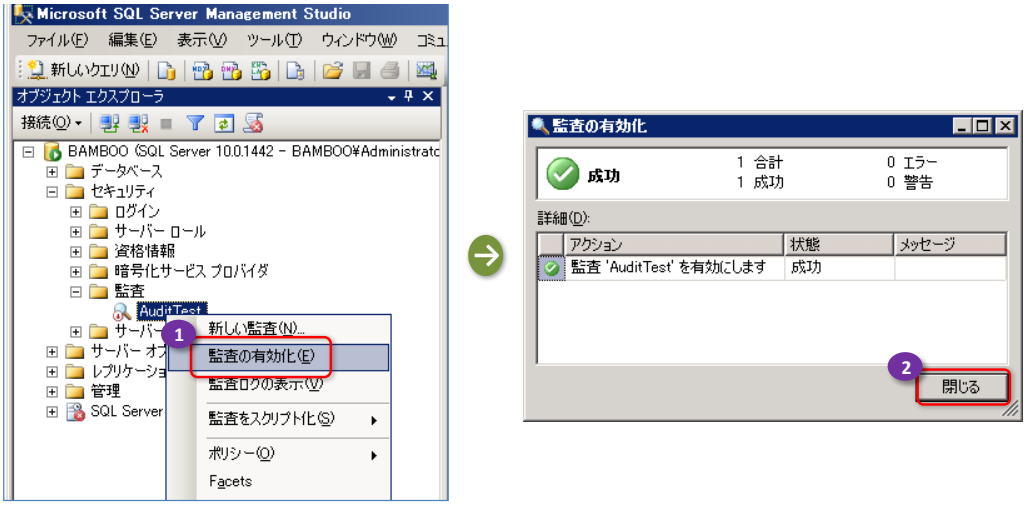


Note： 監査ログのエラー時のシャットダウン

次のように［**監査ログのエラー時にサーバーをシャットダウンする**］をチェックした場合は、監査ログへの書き込みが失敗したときに、SQL Server をシャットダウンすることができます。これにより、ディスク障害時などで、監査ログが記録できない場合に、SQL Server を停止できるようになります。



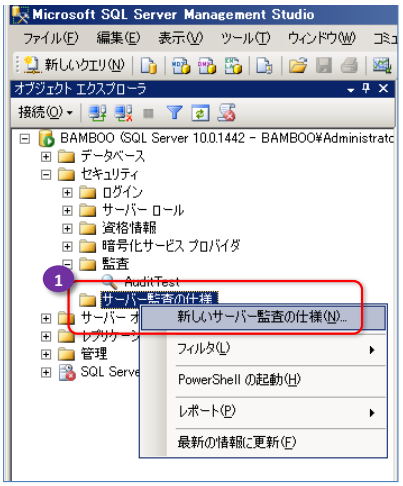
2. 続いて、［**監査**］フォルダを展開して、作成した監査（**AuditTest**）を右クリックし、［**監査の有効化**］をクリックします。



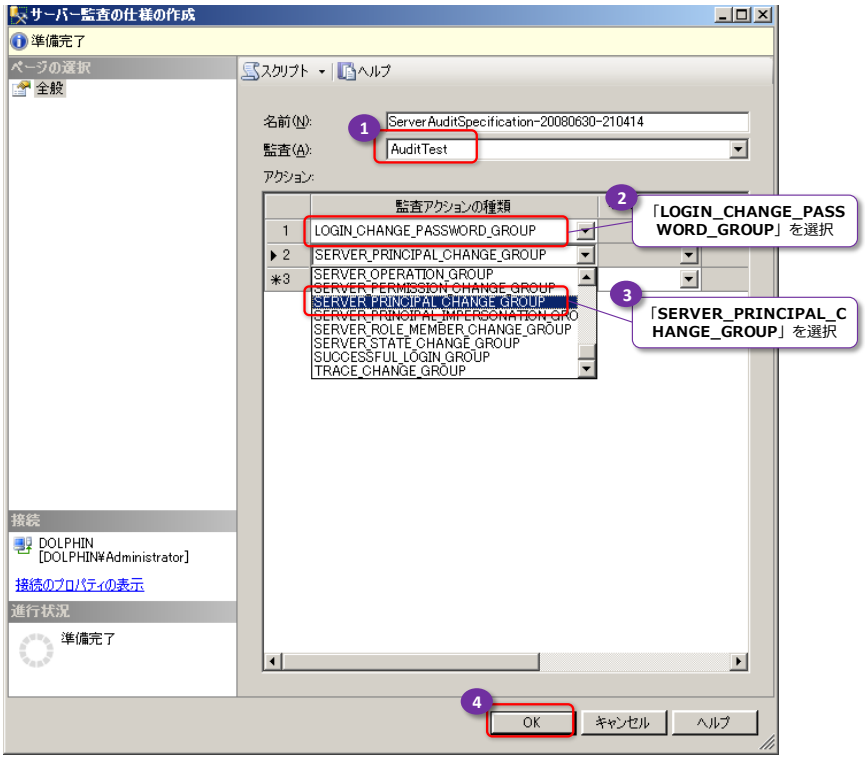
ログイン アカウントの変更履歴の監査

次に、ログイン アカウント（プリンシパル）の作成やパスワード変更など、ログイン アカウントに対する変更操作が行われたことをログへ記録するための監査を作成してみましょう。

3. ログイン アカウントなど、SQL Server 全体に関係するサーバー レベルの監査を作成するには、次のように［**セキュリティ**］フォルダの［**サーバー監査の仕様**］を右クリックして［**新しいサーバー監査の仕様**］をクリックします。

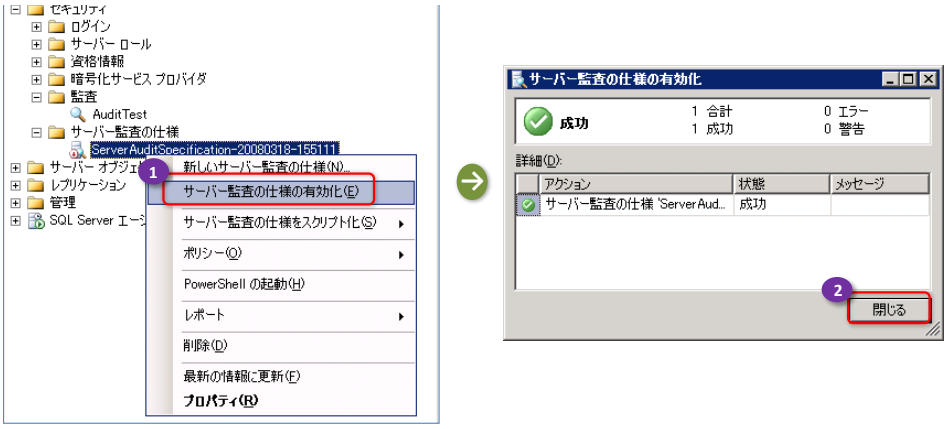


4. これにより、「サーバー監査の仕様の作成」ダイアログが表示されるので、［**監査**］で前の手順で利用した「**AuditTest**」を選択します。



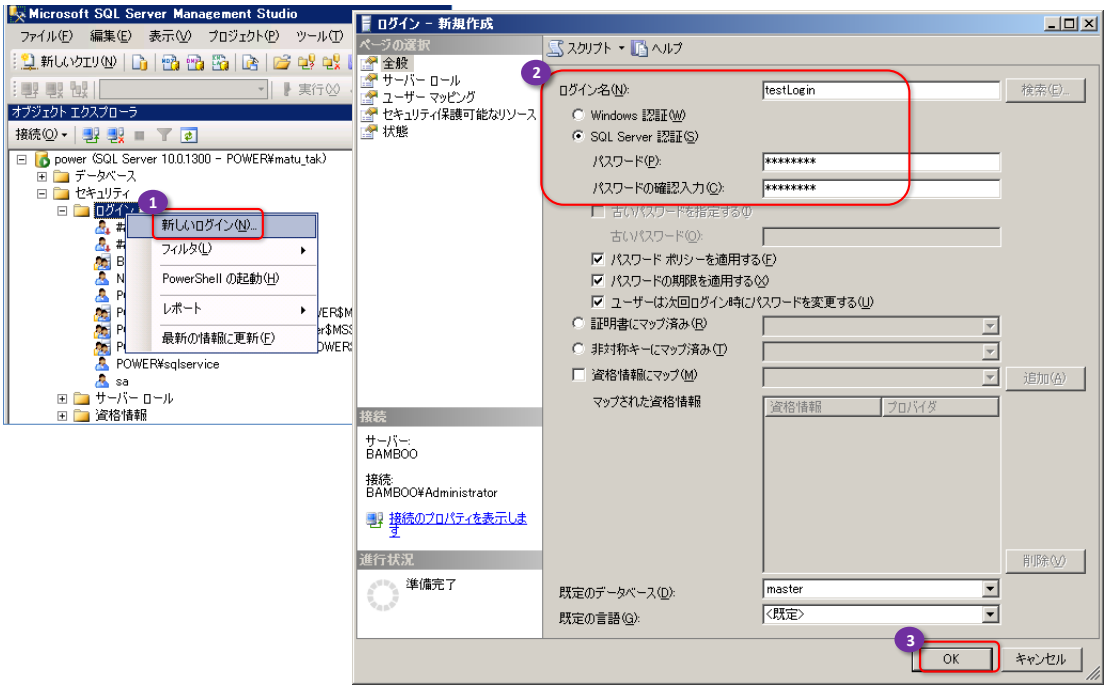
［**監査アクションの種類**］では「**LOGIN_CHANGE_PASSWORD_GROUP**」（パスワードの変更）と「**SERVER_PRINCIPAL_CHANGE_GROUP**」（サーバー プリンシパルの変更）を選択し、［OK］ボタンをクリックします。

5. 次に、作成したサーバー監査の仕様（既定では **ServerAuditSpec ～** という名前）を右クリックして［**サーバー監査の仕様の有効化**］をクリックします。

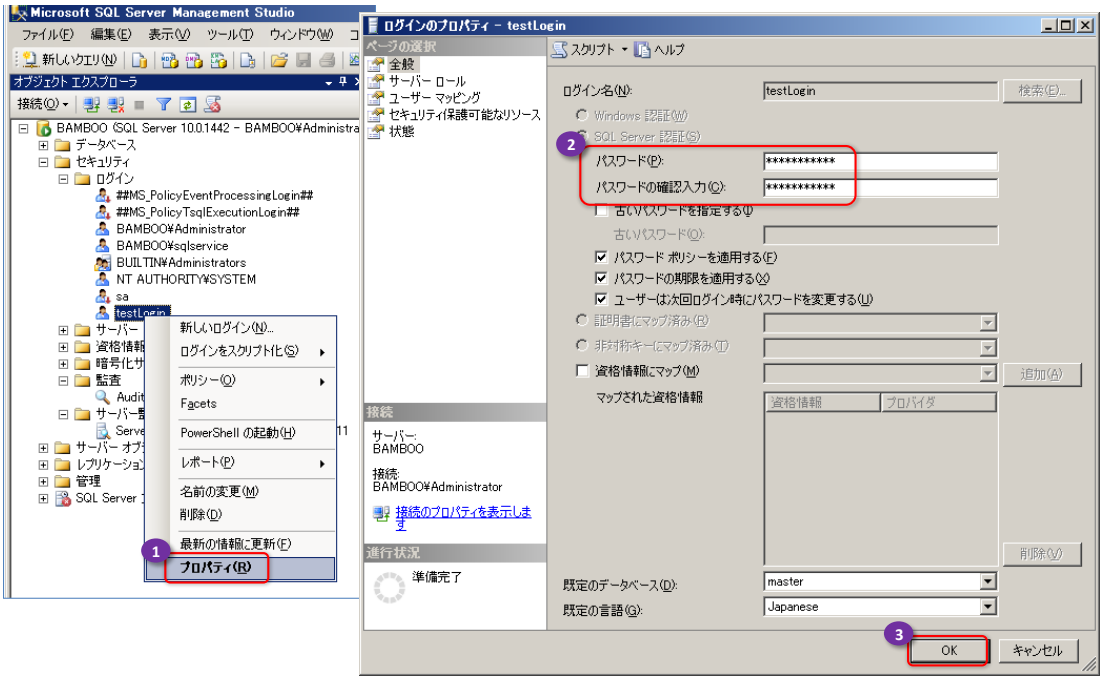


以上で設定が完了です。これで、ログイン アカウント（プリンシパル）の変更（作成 / 変更 / 削除）とパスワードの変更が行われたことをログへ記録できるようになります。

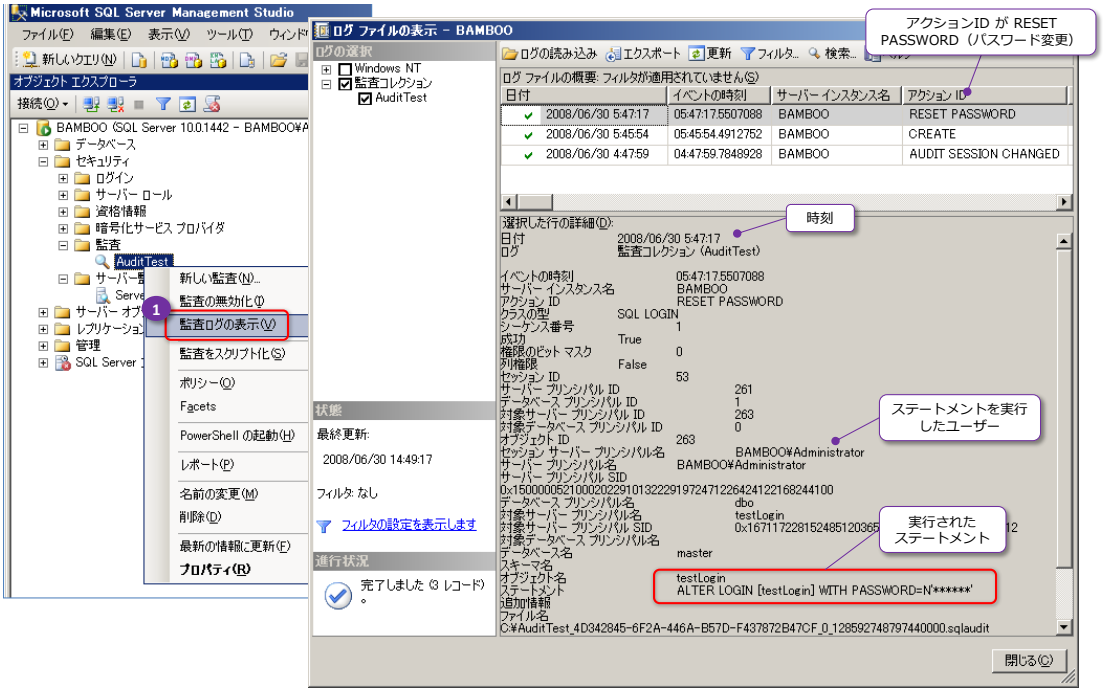
6. 続いて、次のように操作して、任意のログイン アカウントを作成します。



7. 次に、作成したログイン アカウントのプロパティを開いて、パスワードを適当なものへ変更します。



8. 以上の操作が監査ログへ記録されたことを確認するために、次のように［**セキュリティ**］フォルダの［**監査**］フォルダにある監査（**AuditTest**）を右クリックして、［**監査ログの表示**］をクリックします。



パスワードの変更は、［**アクション ID**］が［**RESET PASSWORD**］と表示され、ログイン アカウントの作成は、［**CREATE**］と表示されたログが記録されていることを確認できます。また、ログには、ステートメントが実行された時刻や、実行ユーザー（プリンシパル名）、実際に実行されたステートメントなどが記録されていることも確認できます。

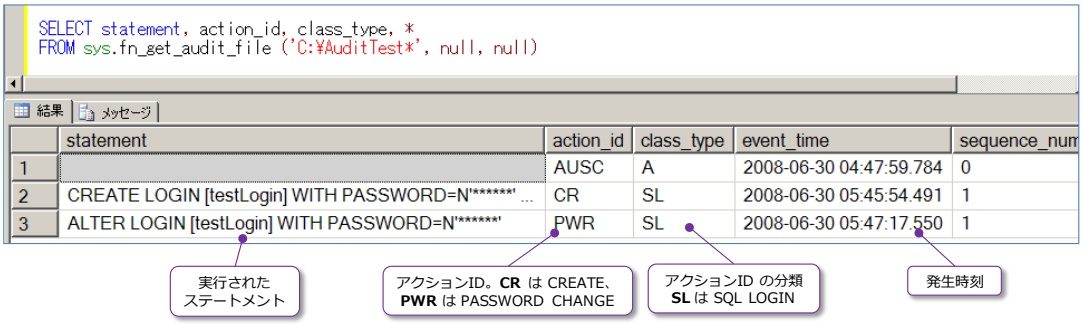
このように、SQL Server Audit 機能を利用すると、ログイン アカウントに対する変更履歴をすべてログへ記録できるようになるので、ログイン アカウントを不正利用（管理者アカウン

トのパスワードを変更されたり、管理者アカウントが勝手に作られるなど）をログへ記録できるようになります。

コマンドでログの参照： fn_get_audit_file

監査ログは、**fn_get_audit_file** システム関数を利用すると、コマンドから参照することもできます。これは、次のように利用します。

```
SELECT statement, action_id, class_type, *
FROM sys.fn_get_audit_file ('C:¥AuditTest*', null, null)
```



この関数は、第１引数へログ ファイルへのパス（* はワイルドカード）を指定して、該当するファイルが複数ある場合には、それらをまとめて読み取ることができます。

このようにステートメントを利用して、ログを参照することもできるので、GROUP BY 演算などと組み合わせて実行すれば、集計レポート（Weekly レポートや Monthly レポートなど）も簡単に作成できるようになります。

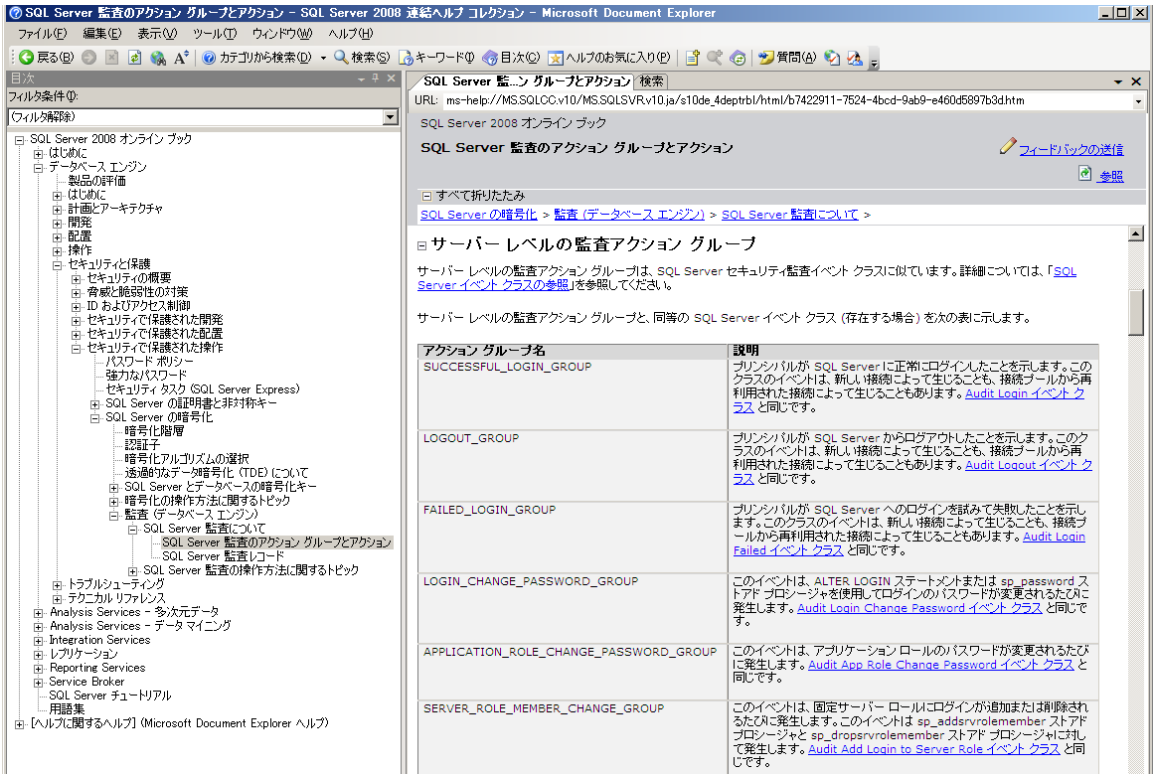
具体的なレポートの作成方法については、SQL Server 2008 徹底検証シリーズの以下のドキュメントに詳しく記載されています。

「コンプライアンス実現のためのガイドライン」
http://www.microsoft.com/japan/sqlserver/2008/bible/cqi.mspx

そのほかのサーバー監査の仕様

サーバー監査の仕様には、「**FAILED_LOGIN_GROUP**」でログインの失敗を記録したり、「**SERVER_PERMISSION_CHANGE_GROUP**」で SQL Server レベルの権限の変更を記録したり、「**AUDIT_ CHANGE_GROUP**」で監査が変更（作成 / 変更 / 削除）されたことを記録できるなど、セキュリティ強化・コンプライアンス実現のための仕様がたくさん用意されています。これらは、オンライン ブックの以下の場所へ詳しく記載されているので、ぜひ参考にしてください。

「データベース エンジン」→「セキュリティと保護」→「セキュリティで保護された操作」
→「SQL Server の暗号化」→「監査（データベース エンジン）」
→「SQL Server Audit について」→「SQL Server 監査のアクション グループとアクション」



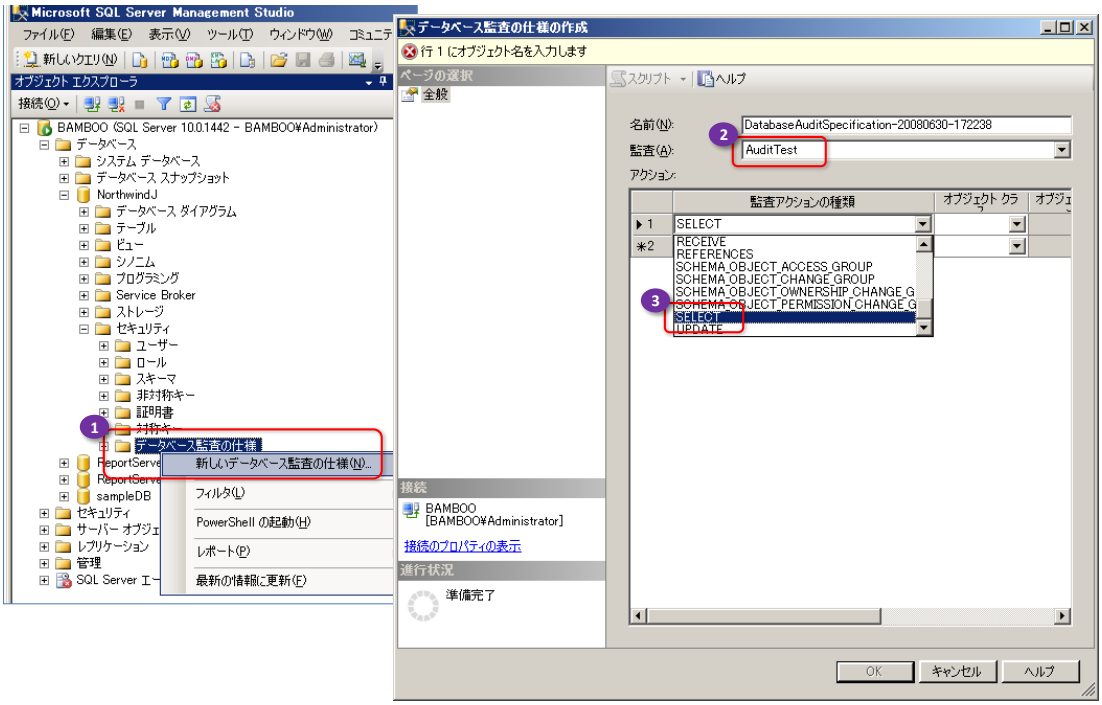
データベースに対する操作履歴の監査

SQL Server Audit では、SQL Server に対するあらゆる操作を監査することができるので、データベースに対する操作（SELECT や INSERT、UPDATE、DELETE など）も、もちろん監査（ログ記録）することができます。

設定手順

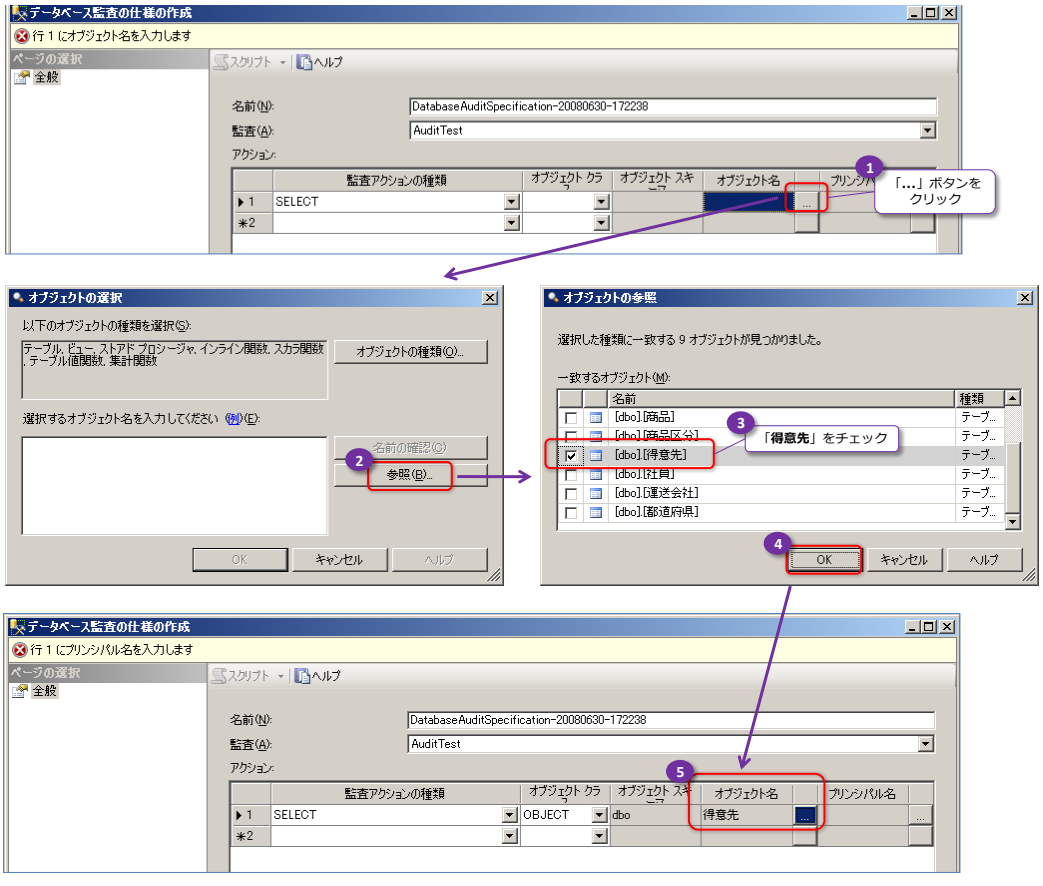
それでは、これを試してみましょう。ここでは、**NorthwindJ** データベース内の「**得意先**」テーブルに対して「**SELECT**」ステートメントが実行されたことをログへ記録する監査（Audit）を作成してみましょう（NorthwindJ データベースのセットアップ方法については、Step 1.3 の手順を参考にしてください）。

1. まずは、[データベース] フォルダの［**NorthwindJ**］データベースを展開して、［**セキュリティ**］フォルダの［**データベース監査の仕様**］を右クリックし、［**新しいデータベース監査の仕様**］をクリックします。

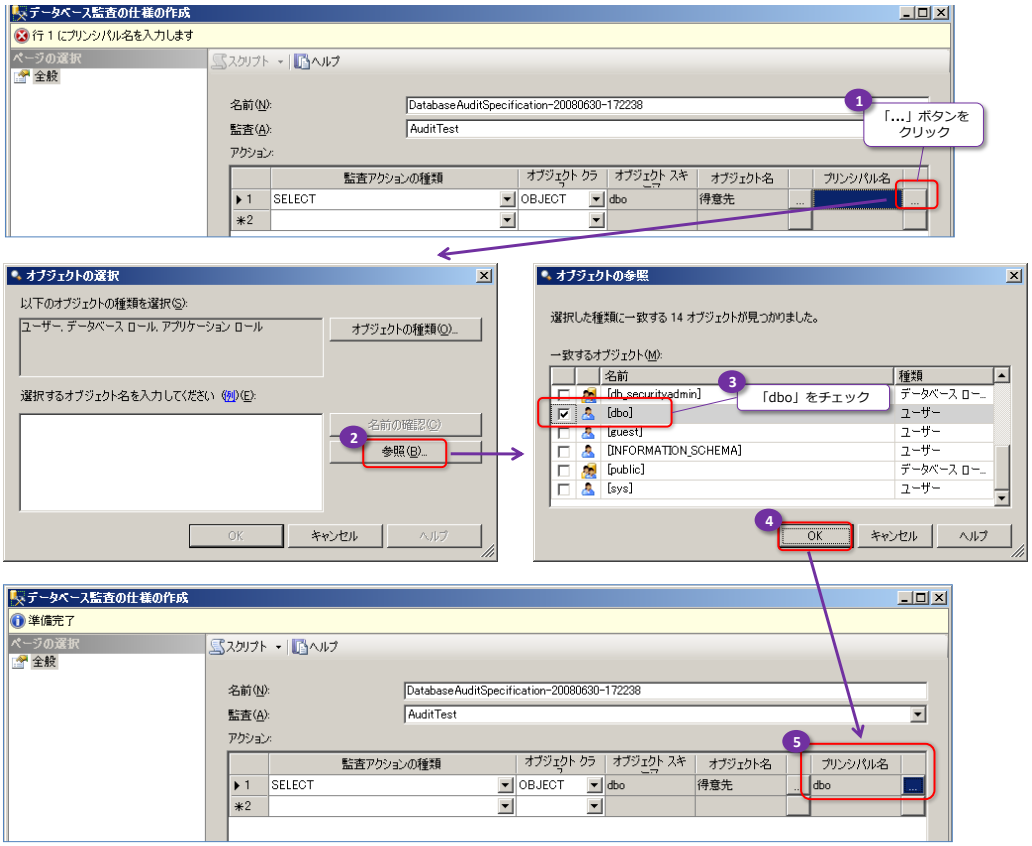


これにより、「データベース監査の仕様の作成」ダイアログが表示されるので、[**監査**]で前の手順で作成した「**AuditTest**」を選択し、[**監査オブジェクトの種類**]で「**SELECT**」を選択します。

2. 次に、[**オブジェクト名**]の[**...**]をクリックして、次のように「オブジェクトの選択」ダイアログで[**参照**]ボタンをクリックして、「**得意先**」テーブルを選択します。

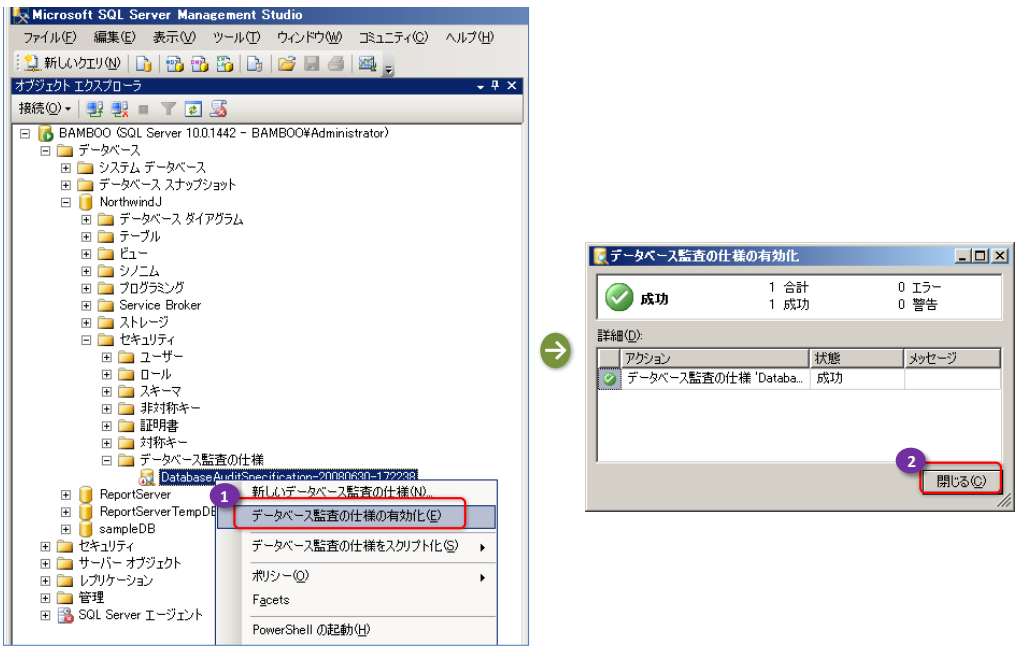


3. 続いて、次のように［**プリンシパル名**］の［**...**］をクリックして、「オブジェクトの選択」ダイアログで［**参照**］ボタンをクリックして、「**dbo**」データベース ユーザーを選択します。



このように設定することで、Northwind」データベース内の「**得意先**」テーブルに対して、「**dbo**」ユーザーから「**SELECT**」ステートメントが実行された場合に、ログへ記録できるようになります。設定を確認後、［**OK**］ボタンをクリックして、ダイアログを閉じます。

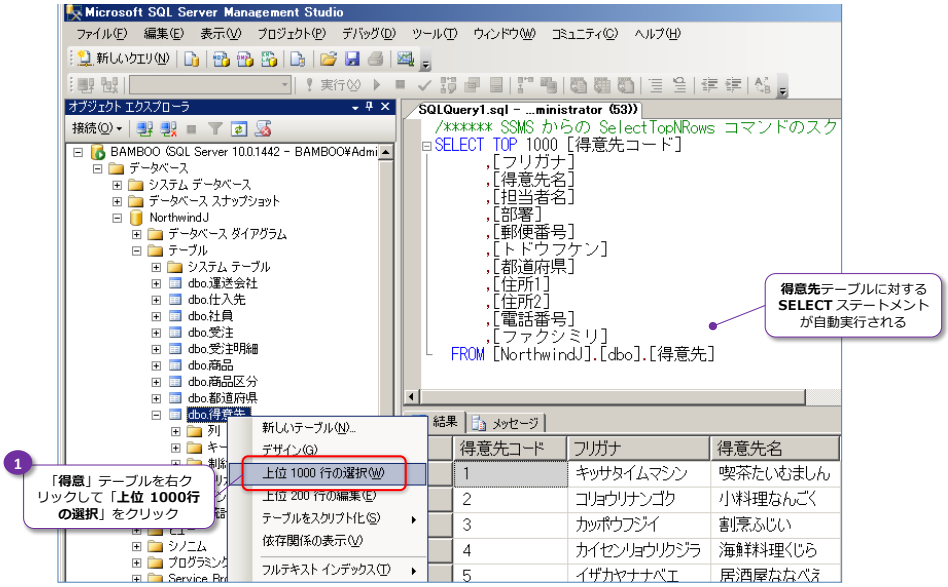
4. 次に、作成したデータベース監査の仕様（既定では **DatabaseAuditSpec ～** という名前）を右クリックして［**データベース監査の仕様の有効化**］をクリックします。



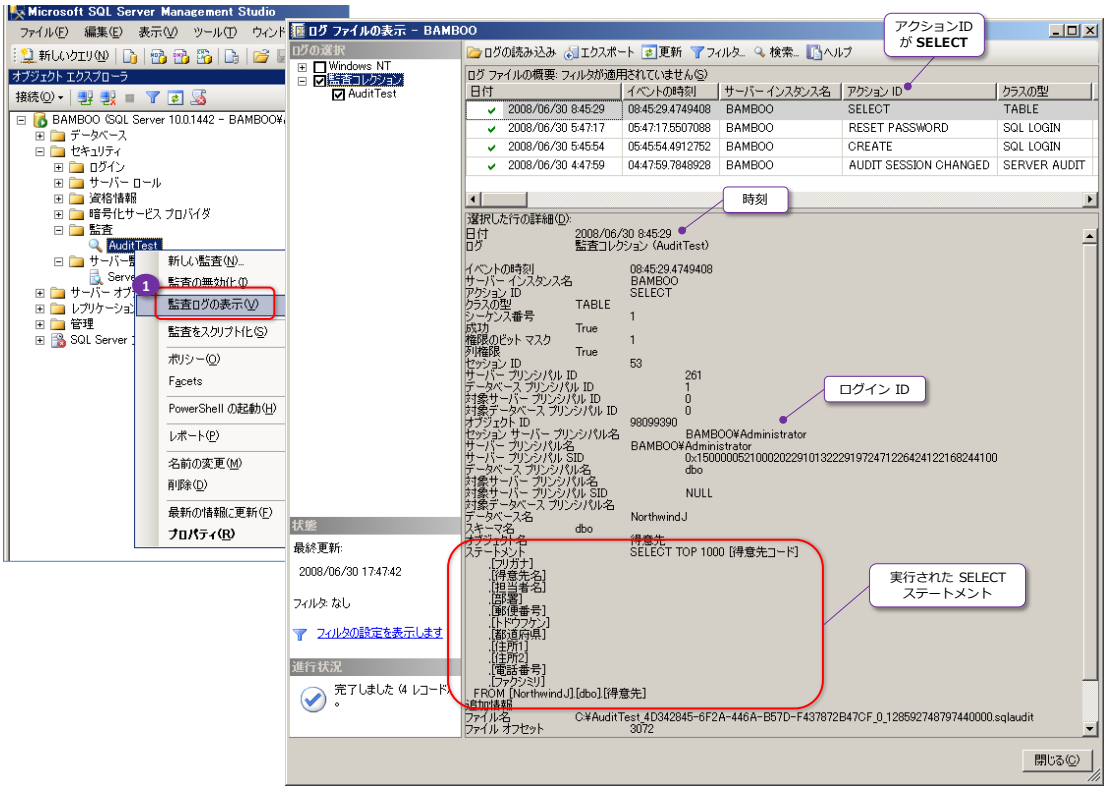
以上で監査の設定が完了です。

監査ログの記録とログの参照

5. 次に、監査対象の「得意先」テーブルに対して、次のように SELECT ステートメントを実行して、監査ログへ記録されるかどうかを確認します（この手順は、dbo ユーザーにマッピングされる SQL Server の管理者アカウントから実行してください）。



6. 続いて、監査ログを参照します。［**セキュリティ**］フォルダの［**監査**］フォルダにある監査（**AuditTest**）を右クリックして、［**監査ログの表示**］をクリックします。



ログの中で、[**アクション ID**] へ「**SELECT**」と表示されているイベントを確認することができます。この中身を参照すると、前の手順で実行した SELECT ステートメントが記録され、そのステートメントを実行したユーザー（ログイン ID）と実行した時刻などを確認することができます。

このように、SQL Server Audit を利用すると、データベースに対するあらゆる操作を監査（ログ記録）できるので、「得意先」や「顧客」テーブルなどを監査すれば、顧客情報（クレジット カードや個人情報など）の漏えいが発生した場合の証跡を残せるようになります。また、特権ユーザー（管理者権限を持ったアカウント）の操作履歴（SQL Server に対する管理操作やデータ操作）をすべて監査しておけば、不正利用（管理者アカウントを悪用したデータの盗難やトロイの木馬の設置など）を監視できるようになります。

以上のように、SQL Server Audit は、**J-SOX 法**（日本版 SOX 法）や**内部統制**などのコンプライアンス（法令遵守）の実現およびセキュリティの強化には、不可欠な機能になります。従来のバージョンのプロファイラ（SQL Server Profiler）でも、同様に監査ログを記録することができますが、プロファイラとの違いは、SQL Server Audit のほうが設定が容易で、かつパフォーマンスが良い（監査のオーバーヘッドがプロファイラよりも小さい）という点です。ぜひ、活用してみてください。

プロファイラと SQL Server のパフォーマンス比較については、SQL Server 2008 徹底検証シリーズの以下のドキュメントに詳しく記載されています。

「コンプライアンス実現のためのガイドライン」
http://www.microsoft.com/japan/sqlserver/2008/bible/cqi.mspx

4.6　データ プロファイル タスクによるデータの分布状況の表示

データ プロファイル タスク

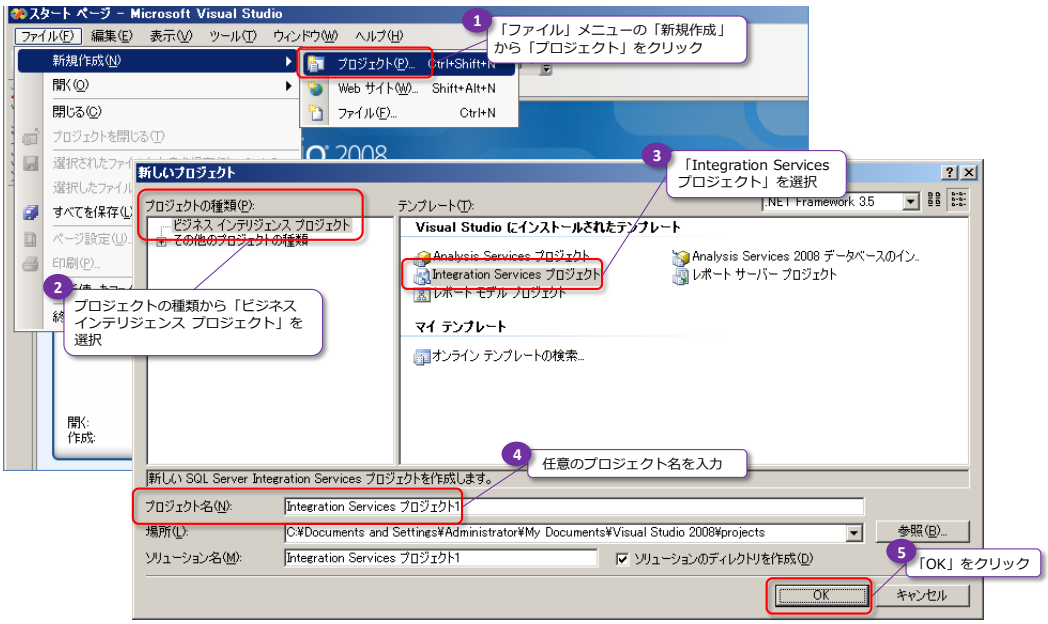
データ プロファイル タスクは、データの分布状況や、データの長さの分布状況、NULL 値の割合などを表示できる便利な機能です。特に、インデックス チューニング時などにデータの分布状況を確認できるのは大変便利です。

データ プロファイル タスクは、SQL Server Integration Services（SSIS）のタスクとして実装され、出力結果を「Data Profile Viewer」ツールを利用してグラフィカルに参照することができます。

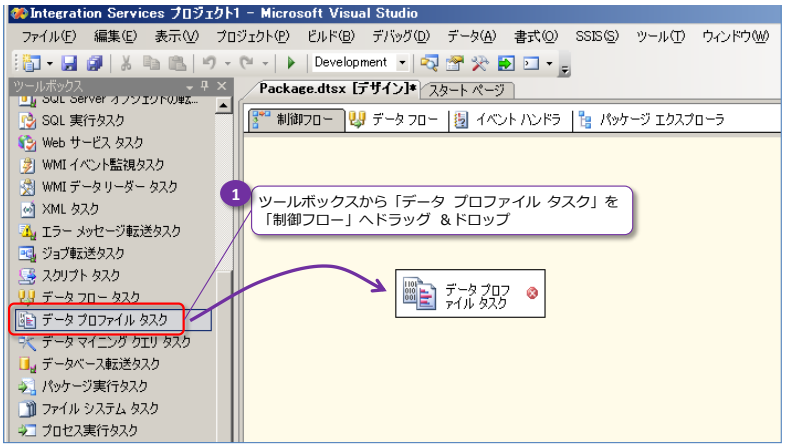
それでは、これを試してみましょう。

設定手順

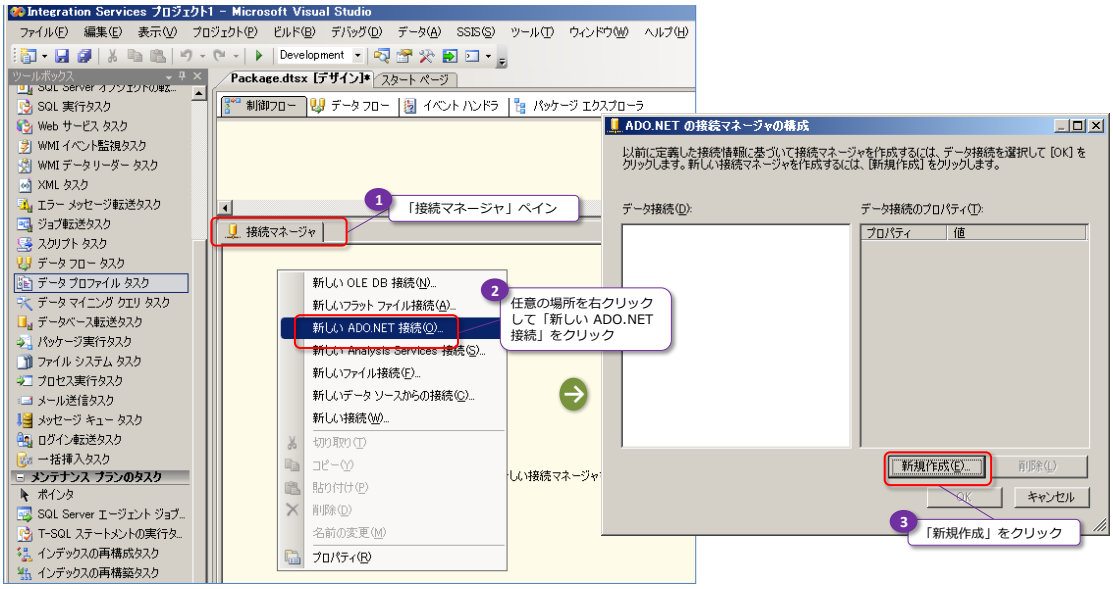
1. まずは、**Business Intelligence Development Studio** を起動して、次のように新しいプロジェクトを作成します。



2. 次に、ツールボックスから［**データ プロファイル タスク**］を［制御フロー］へドラッグ アンド ドロップして配置します。

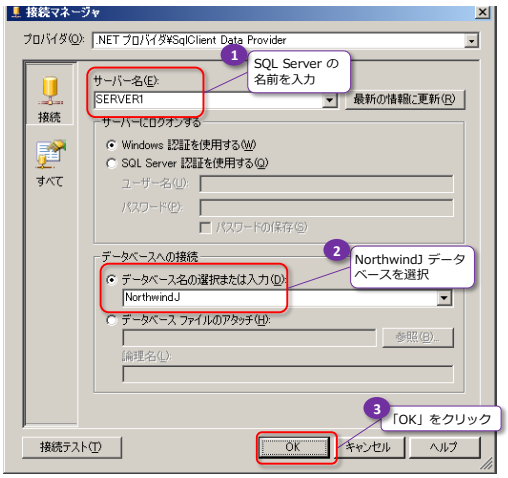


3. 次に、新しい ADO.NET 接続 を作成するために、[**接続マネージャ**] ペイン内の任意の領域で右クリックして、[**新しい ADO.NET 接続**] をクリックします。



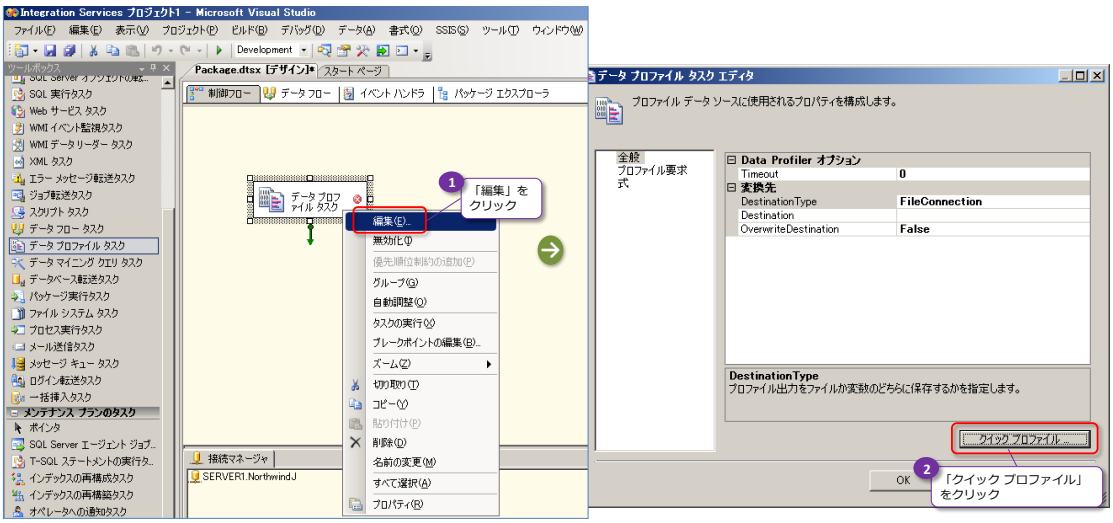
すると、[ADO.NET の接続マネージャの構成] ダイアログが表示されるので、[**新規作成**] ボタンをクリックします。

4. これにより、[接続マネージャ] ダイアログが表示されるので、接続先のサーバー名とデータベースを選択します。



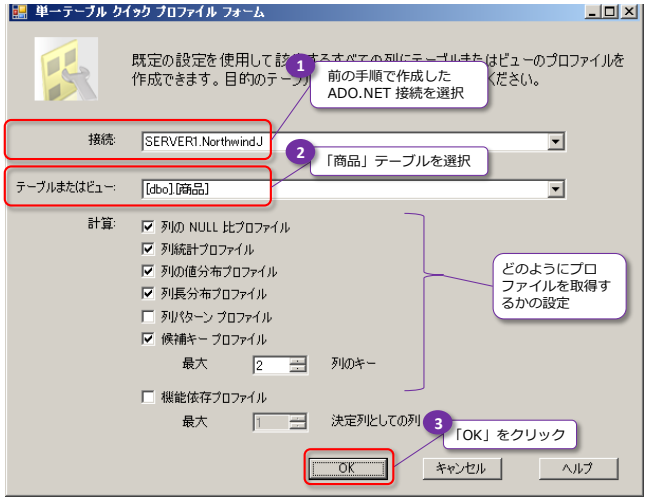
5. 接続先の設定後、[ADO.NET の接続マネージャの構成] ダイアログへ戻ったら、[OK] ボタンをクリックします。

6. 次に、データの分布状況を調査するテーブルを指定するために、[制御フロー] へ配置した [**データ プロファイル タスク**] を右クリックして [**編集**] をクリックします。

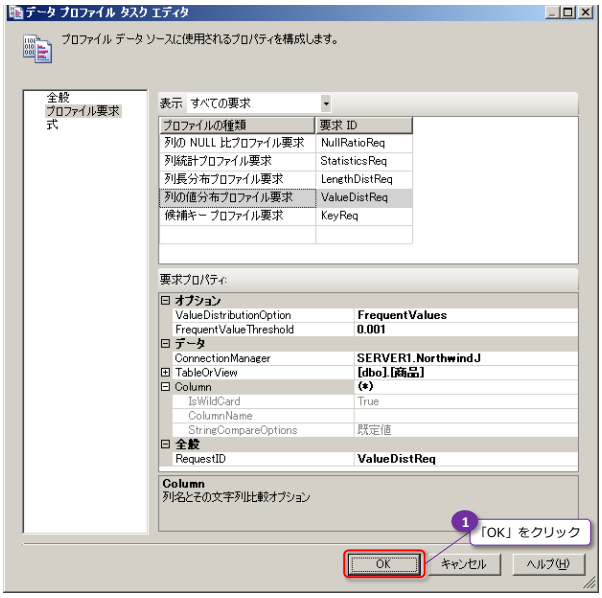


[データ プロファイル タスク エディタ] ダイアログが表示されたら、[**クイック プロファイル**] ボタンをクリックします。

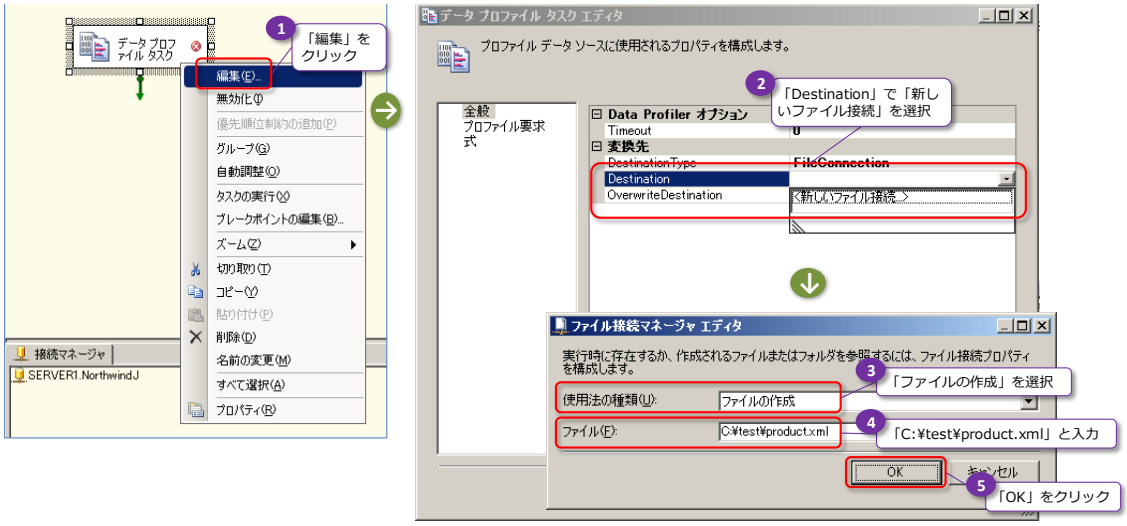
7. これにより、[単一テーブル クイック プロファイル フォーム] ダイアログが表示されるので、次のように [接続] へ前の手順で作成した ADO.NET 接続 を選択して、[**テーブルまたはビュー**] で「**商品**」テーブルを選択し、[OK] ボタンをクリックします。



8. [データ プロファイル タスク エディタ] ダイアログへ戻ったら、[OK] ボタンをクリックします。

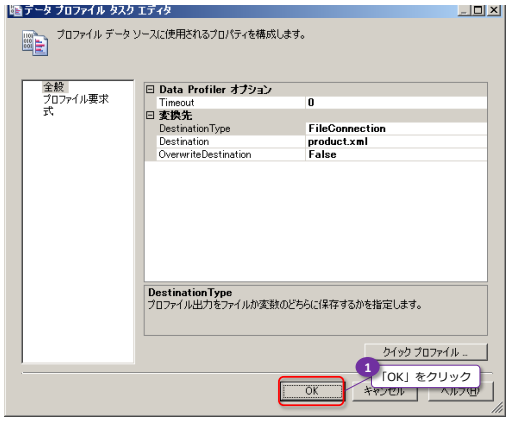


9. 次に、出力結果を書き出すファイルを設定するために、再度［制御フロー］へ配置した［**データ プロファイル タスク**］を右クリックして［**編集**］をクリックし、［データ プロファイル タスク エディタ］ダイアログを表示します。

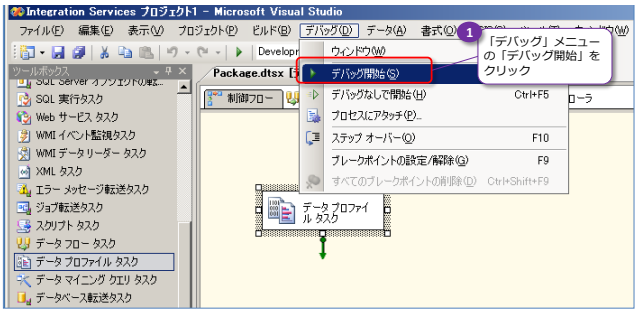


［**Destination**］で「**新しいファイル接続**」を選択し、表示された［ファイル接続マネージャ エディタ］で、［**使用法の種類**］で「**ファイルの作成**」を選択して、［ファイル］に「C:¥test¥**product.xml**」と入力し（フォルダ名は使用している環境に応じて適宜変更してください）、［OK］ボタンをクリックします。

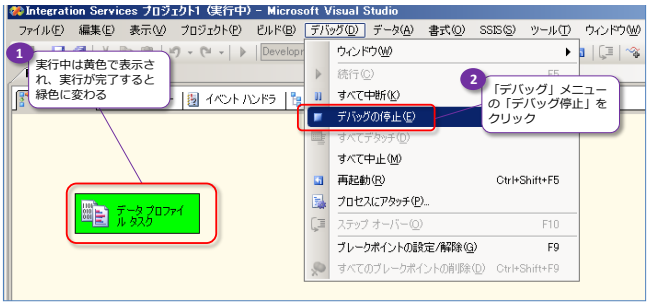
10. ［データ プロファイル タスク エディタ］ダイアログへ戻ったら、［OK］ボタンをクリックします。



11. 次に、データ プロファイル タスク を実行するために、[**デバッグ**] メニューの [**デバッグ開始**] をクリックします。



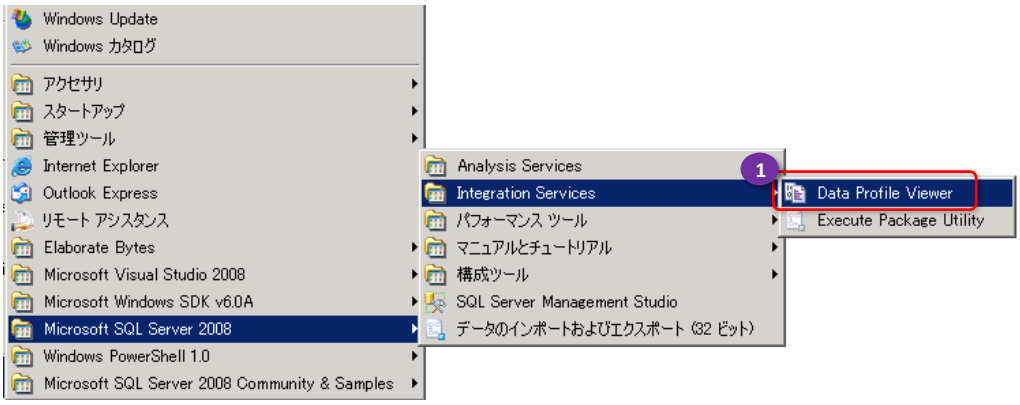
12. デバッグの実行中は [制御フロー] へ配置した [データ プロファイル タスク] が黄色で表示され、実行が完了すると緑色に変わります。デバッグが完了したら、[**デバッグ**] メニューの [**デバッグの停止**] をクリックします。



実行が完了すると、手順 9 で設定した出力結果ファイル（C:¥test¥product.xml）へデータの分布情報が書き込まれています。

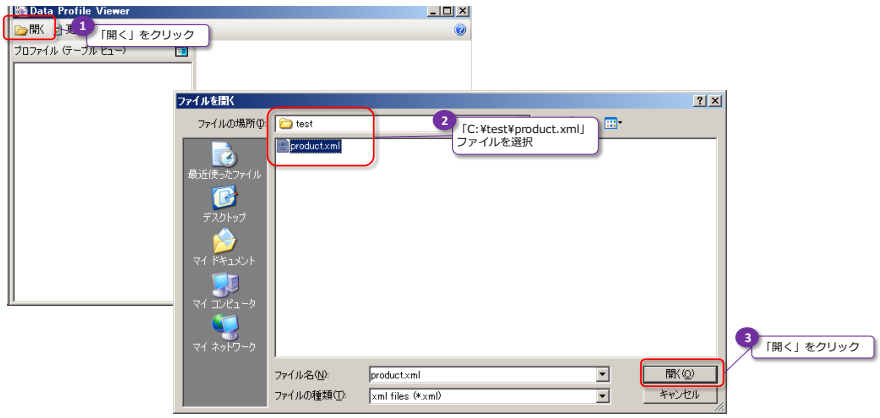
Data Profile Viewer で出力結果の参照

13. 次に、出力結果を参照するために、[スタート] メニューの [すべてのプログラム] から、[Microsoft SQL Server 2008] を選択し、[**Integration Services**] の [**Data Profile Viewer**] をクリックし、「**Data Profile Viewer**」ツールを起動します。

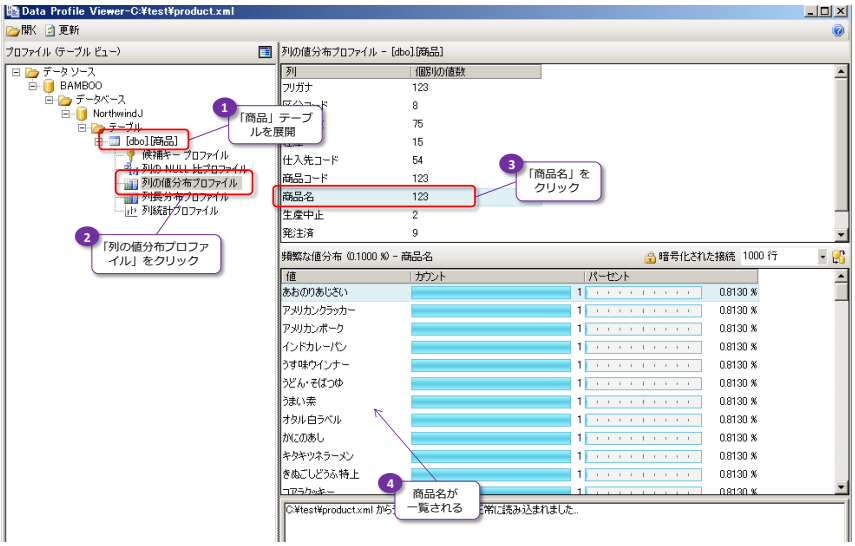


なお、x64 マシン環境では、CU1（累積的な修正プログラム）をインストールしておかないと、Data Profile Viewer が正しく動作しないので、Step1 で説明した CU1 をインストールしておいてください。

14. Data Profile Viewer のツールバーで［**開く**］をクリックして、次のように出力結果ファイル（C:¥test¥**product.xml**）を開きます。



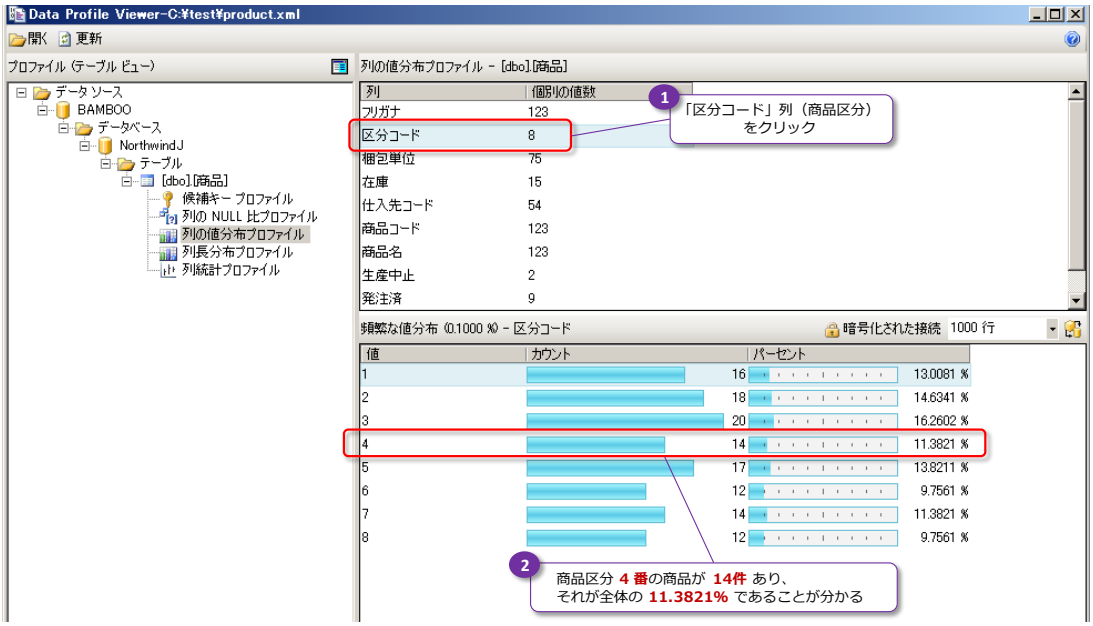
15. Data Profile Viewer で NorthwindJ データベースを展開し、次のように［**列の値分布プロファイル**］をクリックすると、右ペインに、商品テーブルの列名が一覧されることを確認できます。



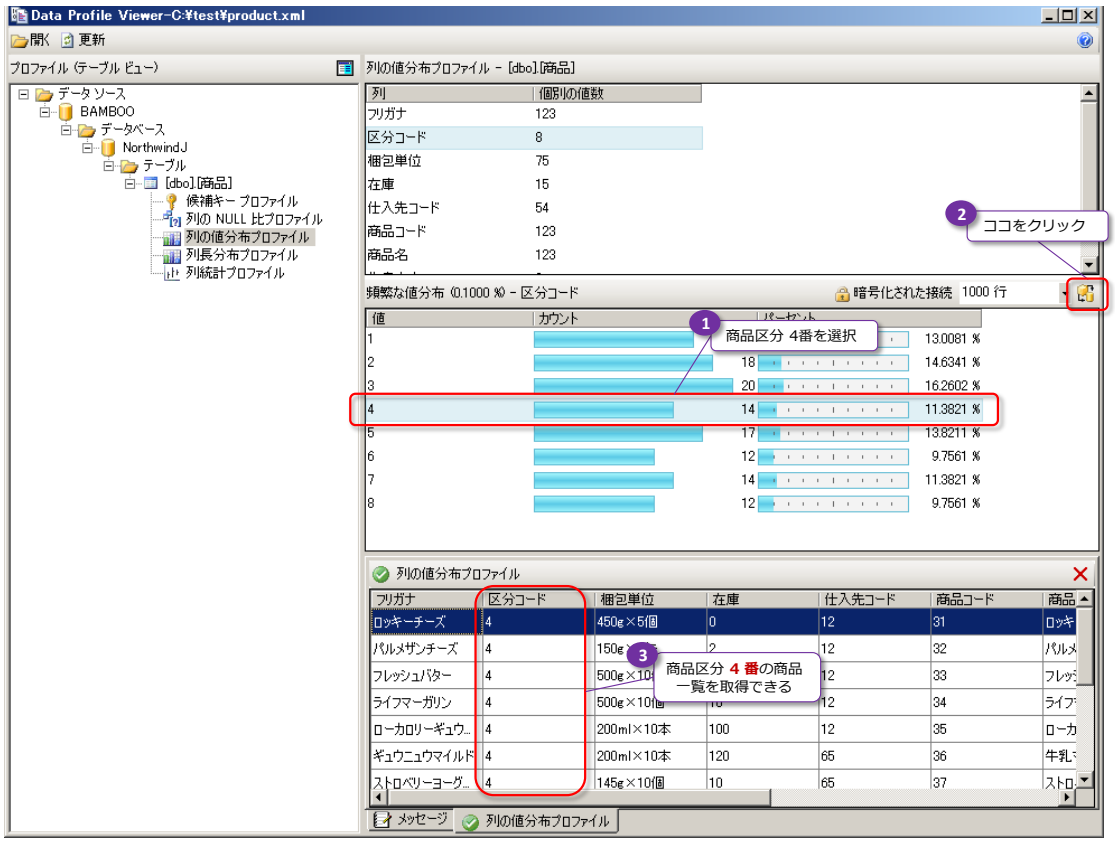
列名の一覧から、「**商品名**」をクリックすると、商品名データが表示されることを確認できま

す。

16. 次に、右ペインの列の一覧から「**区分コード**」をクリックして、区分コードごとのデータの件数を参照します。

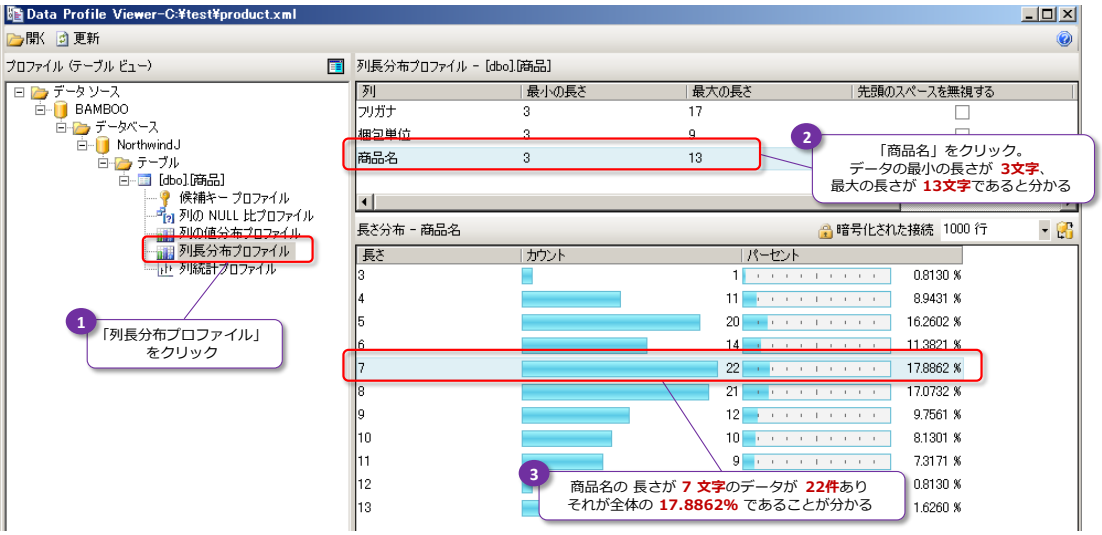


17. 続いて、次のように「**4**」のデータを選択して、[**クエリ**] ボタンをクリックし、区分コードが「４」のデータを参照します。



このように、Data Profile Viewer では、実際のデータも参照することができます。

18. 次に、「商品」テーブルの「**列長分布プロファイル**」をクリックして、右ペインの列の一覧から「**商品名**」列をクリックし、商品名データの長さの分布状況を参照します。



このように データ プロファイル タスク および Data Profile Viewer を利用すると、データの分布状況を簡単に参照することができるので大変便利です。

4.7　データ コレクションによる定期的なデータ収集

データ コレクション（データ収集）

データ コレクションは、定期的なデータ収集が行える機能で、パフォーマンス チューニング時に大変役立つ機能です。データ コレクションでは、任意の Transact-SQL ステートメントと、パフォーマンス カウンタ、トレースからデータを収集することができるので、パフォーマンス状況を監視できる動的管理ビュー（DMV : Dynamic Management View）を定期的にクエリしたり、CPU やメモリなどのパフォーマンス カウンタを定期的に取得するといった使い方ができます。

それでは、これを試してみましょう。

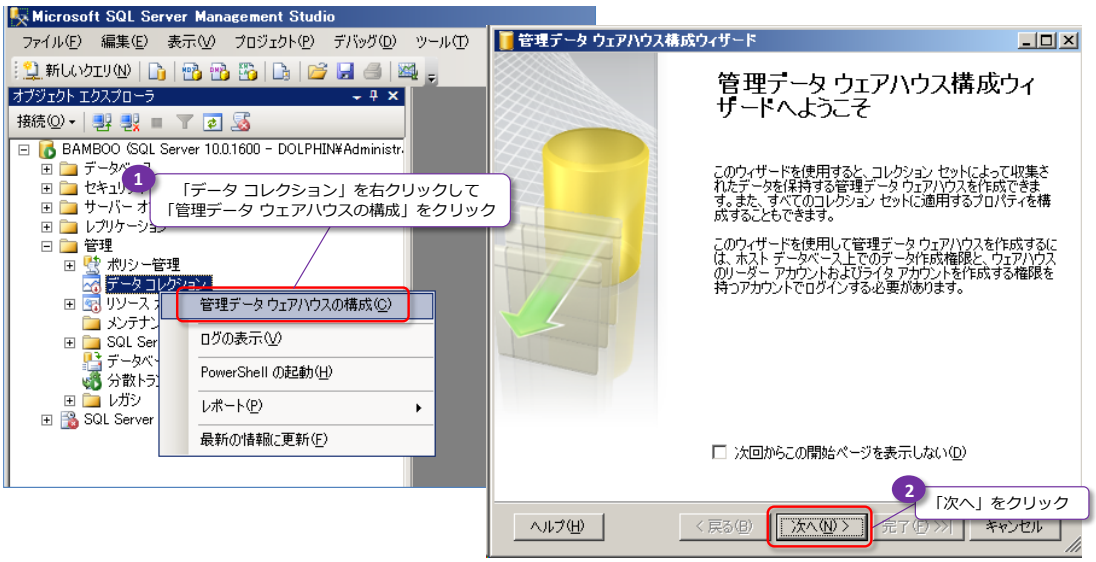
設定手順

ここでは、Transact-SQL ステートメント（GETDATE 関数で現在時刻を取得する SELECT ステートメント）を定期的に実行し、その結果を収集する手順を説明します。

1. データ コレクションは、内部的には SQL Server Agent サービスの機能を利用しているので、まずは SQL Server Agent サービスを開始しておく必要があります。

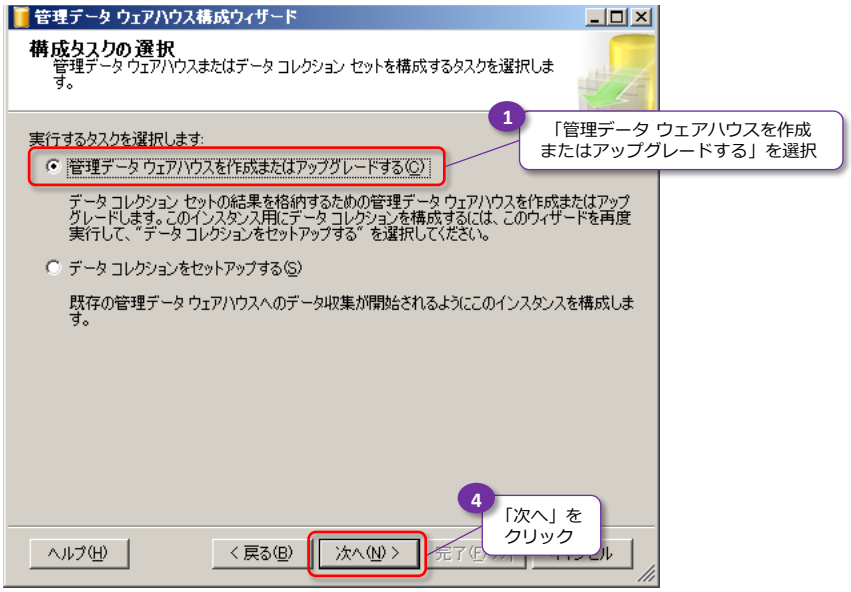
2. データ コレクションでは、収集したデータをデータベースへ格納する前に、一時的にデータを保存する場所として、Windows 上のフォルダを利用するので、その格納先となるフォルダを事前に作成しておく必要があります。ここでは、エクスプローラから、「**C:¥DataCollectionTemp**」という名前のフォルダ（ドライブ名は使用している環境によって適宜変更してください）を作成しておきます。

3. 次に、オブジェクト エクスプローラの［**管理**］フォルダを展開し、［**データ コレクション**］を右クリックして、［**管理データウェアハウスの構成**］をクリックします。

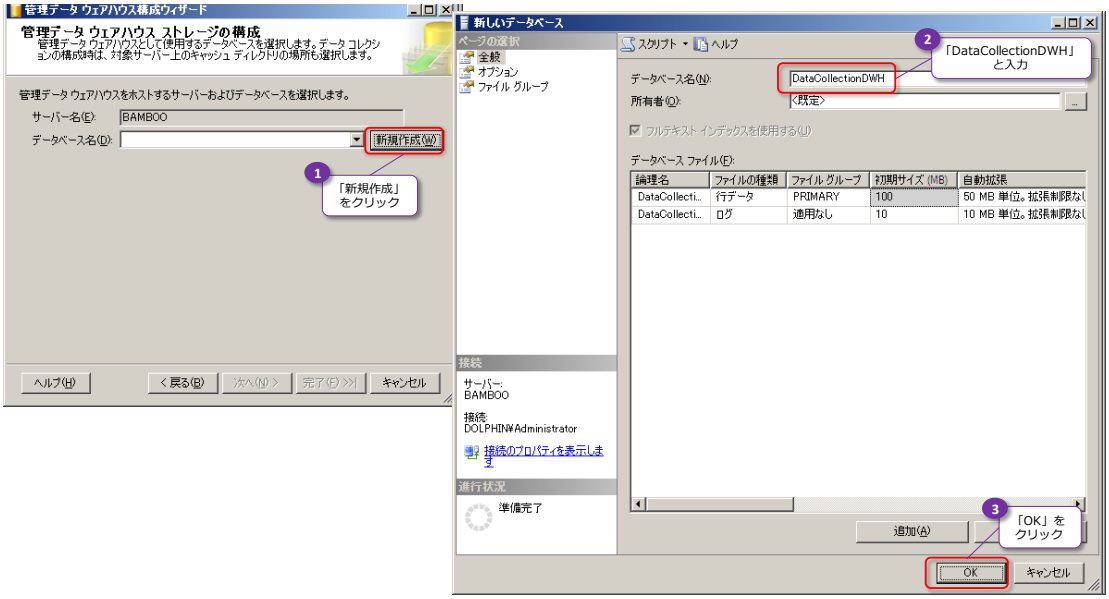


すると、［**管理データ ウェアハウス構成ウィザード**］が起動するので、［次へ］ボタンをクリックして、次のページへ進みます。

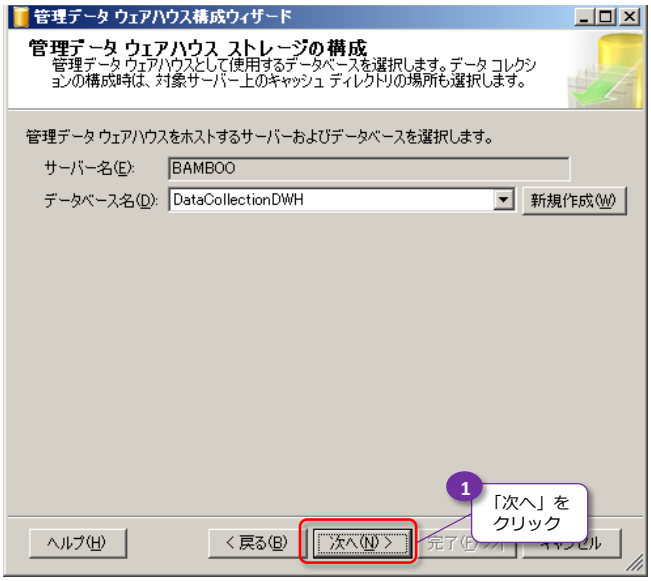
4. 最初のページでは、[**管理データウェアハウスを作成またはアップグレードする**] を選択して、[次へ] ボタンをクリックします。



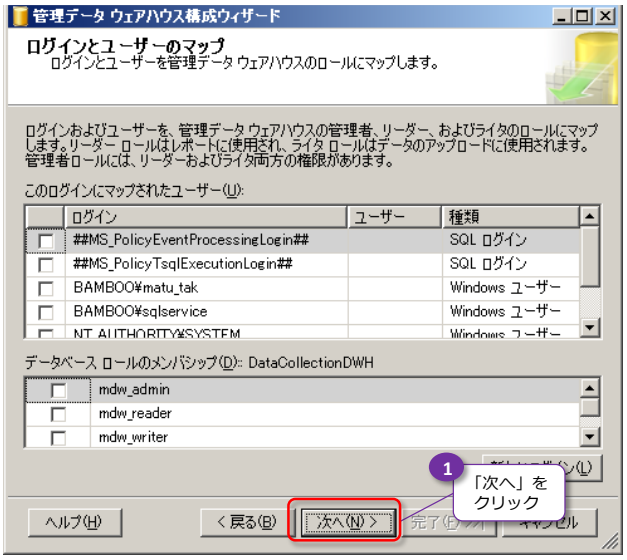
5. データ コレクションでは、収集したデータをデータベースへ格納するので、その格納先となるデータベースを作成しておく必要があります。ここでは、次のように [**新規作成**] ボタンをクリックして、「**DataCollectionDWH**」という名前で作成しておきます。



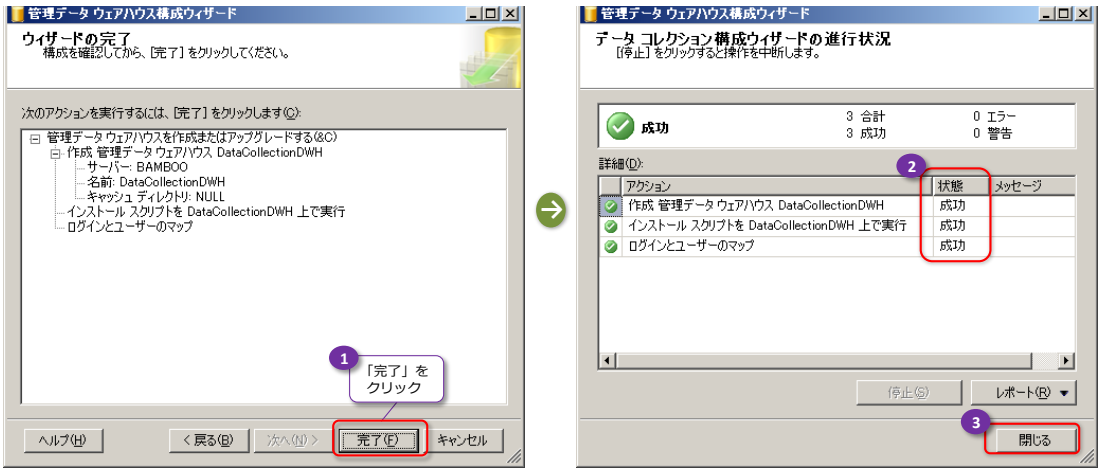
作成後、次のように [次へ] ボタンをクリックします。



6. 続いて、データ コレクションの管理者アカウントとして追加したいログイン アカウントを任意で指定し、［次へ］ボタンをクリックします。



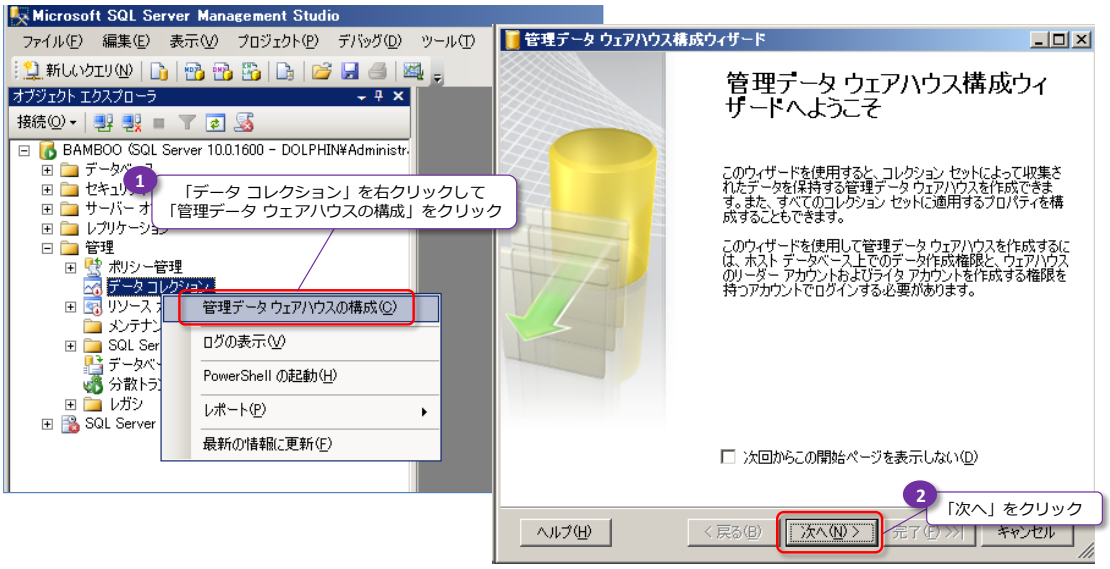
7. 最後のページ［ウィザードの完了］では、［完了］ボタンをクリックします。



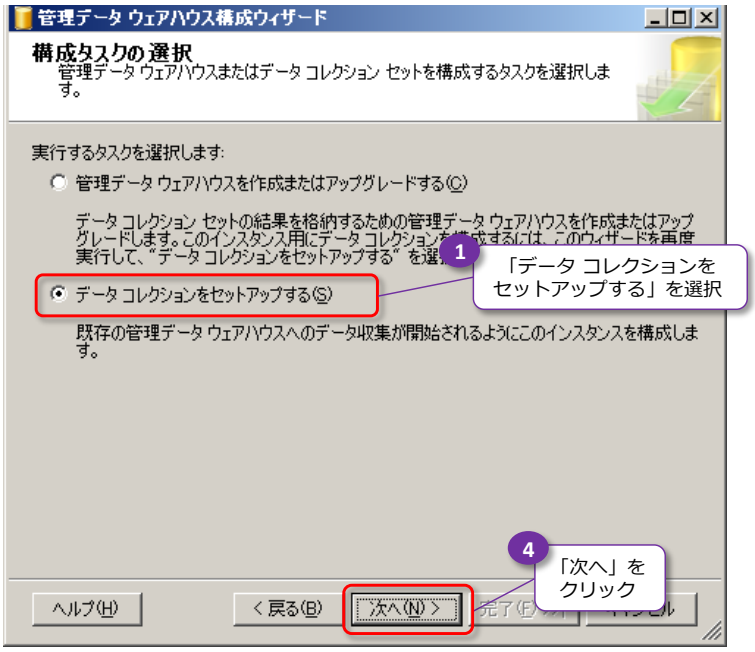
すると、進行状況が表示される［データ コレクション構成ウィザードの進行状況］ページが

表示されるので、構築が完了するのを待ちます（すべての［状態］へ「成功」と表示されれば構築が完了です）。構築完了後、［閉じる］ボタンをクリックします。

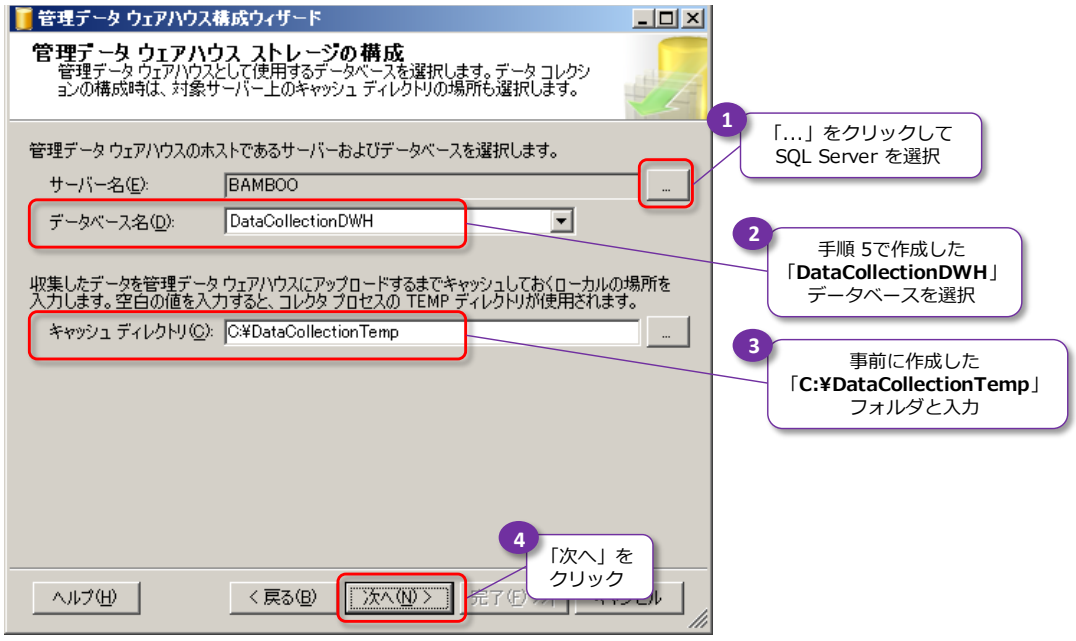
8. 次に、もう一度［**管理データ ウェアハウス構成ウィザード**］を起動します。



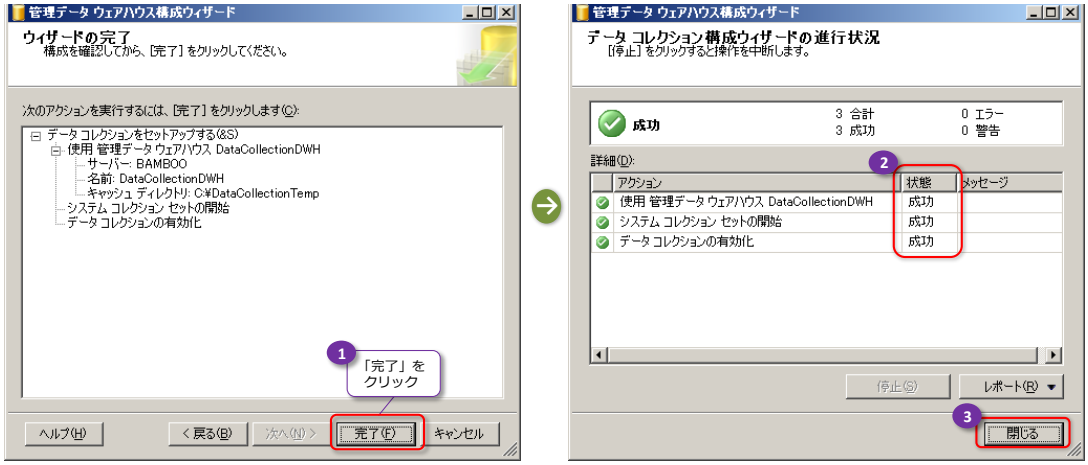
9. ［構成タスクの選択］画面では、今度は、［**データ コレクションをセットアップする**］を選択して、［次へ］ボタンをクリックします。



10. 次の［管理データ ウェアハウス ストレージの構成］画面では、前の手順 5 で作成したデータベースと Windows フォルダを指定して、［次へ］ボタンをクリックします。

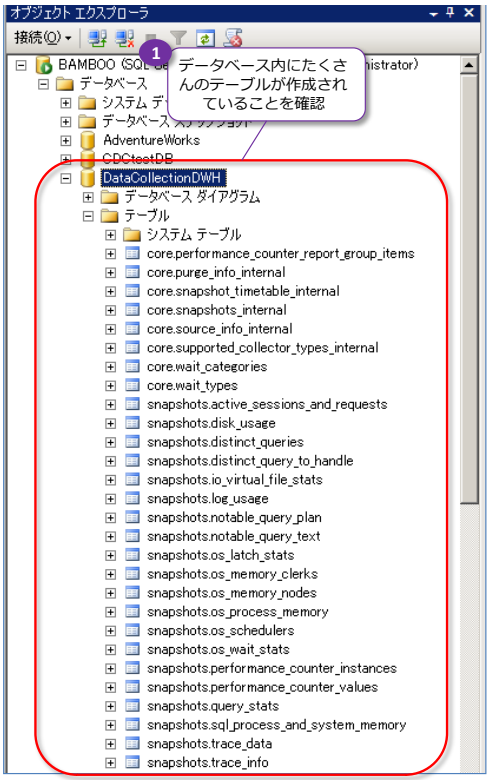


11. 最後の［ウィザードの完了］画面では、［完了］ボタンをクリックします。



進行状況が表示される［データ コレクション構成ウィザードの進行状況］ページが表示されたら、構築が完了するのを待ちます（すべての［状態］へ「成功」と表示されれば構築が完了です）。構築完了後、［閉じる］ボタンをクリックします。

12. オブジェクト エクスプローラで、次のように **DataCollectionDWH** データベース内のテーブルを参照すると、ウィザードによって多くのテーブルが自動作成されていることを確認できます。



13. 次に、Collection Set と Collection Item を作成するために、クエリ エディタで次の SQL ステートメントを記述またはサンプル スクリプト内の「**Step4-07_DataCollection.sql**」ファイルから SQL をコピーして貼り付けて、実行します。

```
-- CollectorSchedule_Every_5min のuid を取得
USE msdb
DECLARE @schedule_uid uniqueidentifier
SELECT @schedule_uid =
                (SELECT schedule_uid
                  FROM sysschedules_localserver_view
                   WHERE name=N'CollectorSchedule_Every_5min')

-- Collection Set の作成
DECLARE @collection_set_id int
EXEC dbo.sp_syscollector_create_collection_set
    @name = N'DCtest1',
    @schedule_uid = @schedule_uid,
    @collection_mode = 1,
    @days_until_expiration = 30,
    @description = N'テスト',
    @collection_set_id = @collection_set_id output

DECLARE @params XML
SELECT @params = CONVERT(XML,
    N'<ns:TSQLQueryCollector xmlns:ns="DataCollectorType">
        <Query>
          <Value>SELECT GETDATE() AS 現在時刻</Value>
          <OutputTable>DCtest_GetDate</OutputTable>
        </Query>
      </ns:TSQLQueryCollector>')
```

5分ごとに T-SQL クエリを実行するために、スケジュール ID を取得

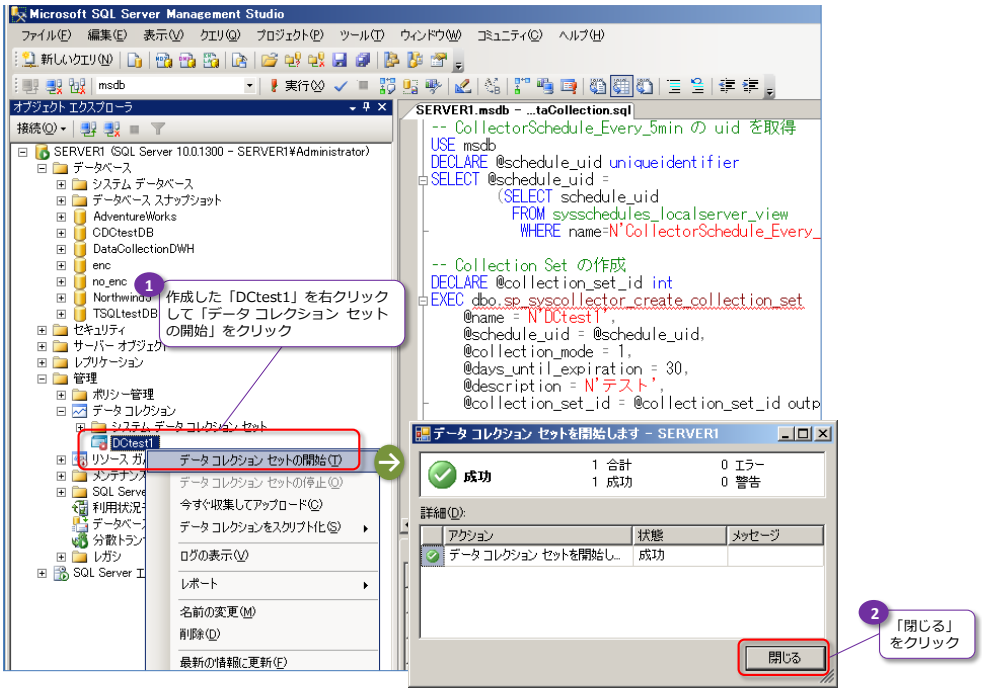
Collection Set を DCtest1 という名前で作成

定期的に実行したい T-SQL を <Query> 要素の <Value> 要素へ記述

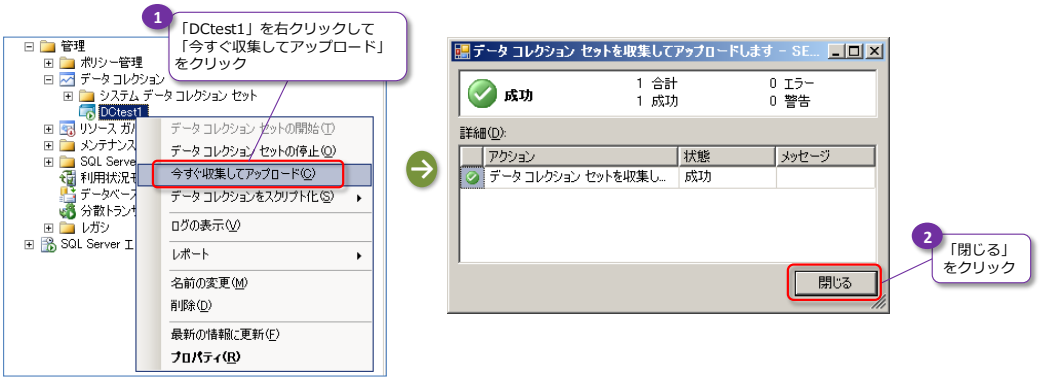
```
-- Collection Item の作成
-- @collector_type_uid の'302E～' は、T-SQL Query Collector Type
DECLARE @collection_item_id int
EXEC dbo.sp_syscollector_create_collection_item
    @collection_set_id = @collection_set_id,
    @collector_type_uid = N'302E93D1-3424-4BE7-AA8E-84813ECF2419',
    @name = 'DMVtest1',
    @frequency = 30,
    @parameters = @params,
    @collection_item_id = @collection_item_id output;
```

XML 形式で記述した T-SQL クエリ

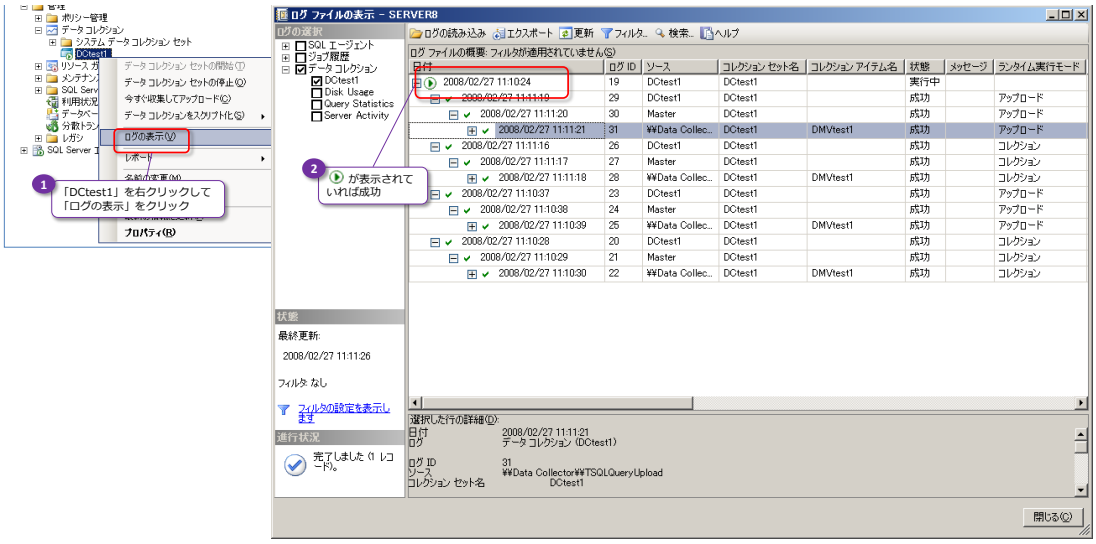
14. 実行後、作成した データ コレクション セットを起動するために、次のように「**DCtest1**」を右クリックして、[**データ コレクション セットの開始**] をクリックします。



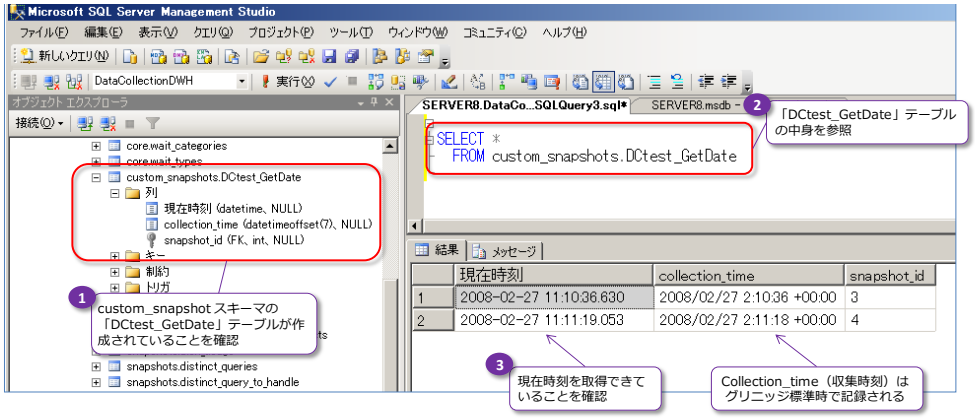
15. 次に、手動でデータを収集するために、DCtest1 を右クリックして［**今すぐ収集してアップロード**］をクリックします。



16. 正常にデータが収集されたかどうかを確認するために、DCtest1 を右クリックして［**ログの表示**］をクリックし、ログを参照します。



17. 収集されたデータを確認するために、DataCollectionDWH データベース内のテーブルを参照します。



custom_snapshot スキーマの「**DCtest_GetDate**」テーブルが作成されていることを確認できます。SELECT ステートメントで DCtest_GetDate テーブルの中身を参照すると、現在時刻を 2 件取得できていることが分かります。それぞれの時刻は、データ コレクション セットの開始時と手順 15 を実行したときの時刻です。また、手順 14 を実行してから 5 分以上経過している場合は、そのタイミングでもデータ収集が行われます（以降、5 分ごとにデータ収集が行われます）。

このように データ コレクション機能を利用すると、任意の Transact-SQL ステートメントを定期的に実行して、データを収集することができるので、パフォーマンス チューニング時に大変役立ちます。また、データ コレクションでは、パフォーマンス カウンタとトレースからもデータを収集することができるので、こちらも便利です。

また、データ コレクションでは、実は「**システム データ コレクション**」と呼ばれる組み込みの データ コレクションが用意されていて、主要な動的管理ビューとパフォーマンス カウンタが自動的に取得されています。これは「**パフォーマンス データ コレクション**」と呼ばれ、グラフィカルなレポートも表示可能になるので、これについては、次の次のトピック「4.9　パフォーマンス データ コレクション」で詳しく説明します。

4.8　リソース ガバナによるリソース調整

リソース ガバナ（Resource Governor）

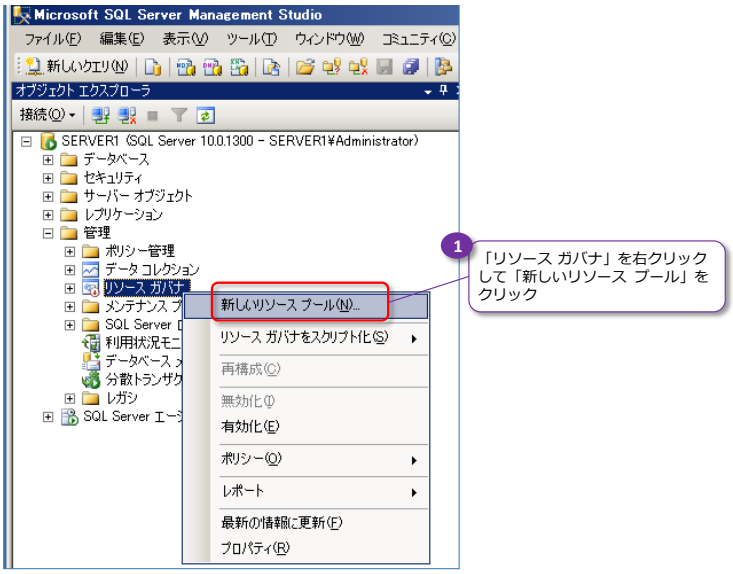
リソース ガバナは、CPU 利用率やメモリ割り当て量といったリソースの調整（governor）が行える大変便利な機能です。これにより、CPU を占有するバッチ アプリケーションの CPU 利用率を制限したり、CPU を占有する SQL の CPU 利用率を制限したりするといったことが可能になります。

それでは、リソース ガバナを試してみましょう。設定のおおまかな流れは次のとおりです。

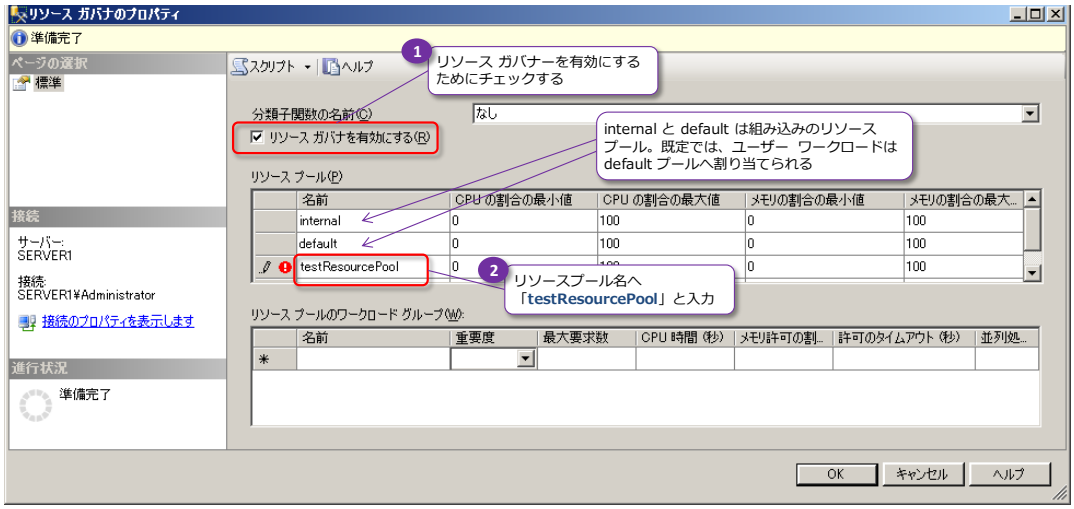
1. リソース プールの作成
2. ワークロード グループの作成
3. ユーザー定義分類子関数の作成と設定、リソース ガバナの有効化

設定手順

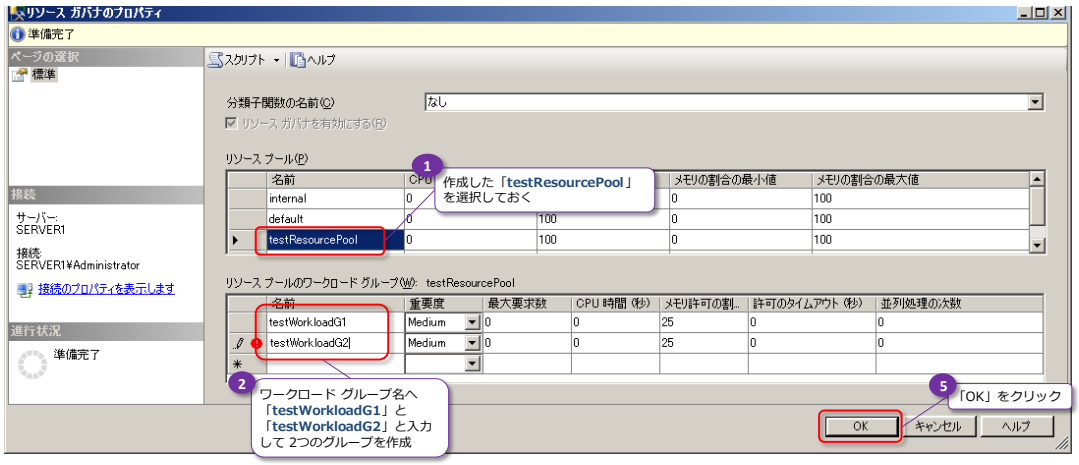
1. まずは、新しいリソース プールを作成するために、次のようにオブジェクト エクスプローラの［**管理**］フォルダで［**リソース ガバナ**］を右クリックして［**新しいリソース プール**］をクリックします。



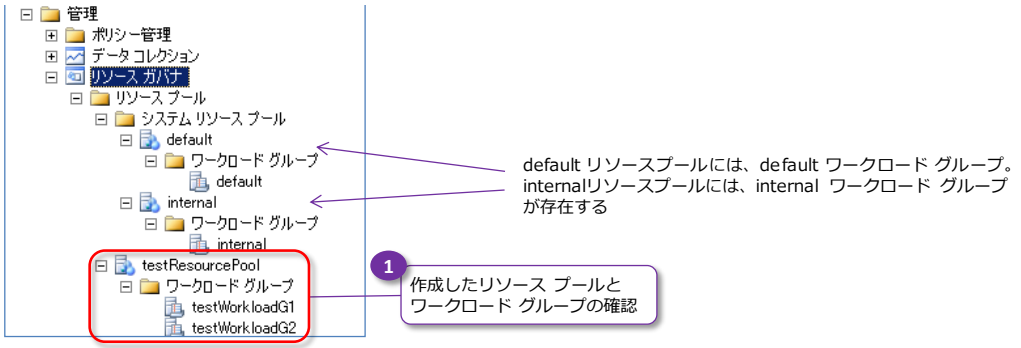
すると、[リソース ガバナのプロパティ] ダイアログが表示されるので、［**リソース ガバナを有効にする**］をチェックし、［リソース プール］の［名前］列に「**testResourcePool**」など、任意の名前を入力します。



2. 次に、リソース プール内へワークロード グループを作成するために、引き続き同じダイアログで、作成した「testResourcePool」を選択して［**リソースのワークロード グループ**］セクションの［名前］列に「**testWorkloadG1**」および「**testWorkloadG2**」と入力して 2 つのワークロード グループを作成します。



3. 作成したリソース プールとワークロード グループを確認するために、次のように［**管理**］フォルダの［**リソース ガバナ**］を展開します。



4. 次に、**ユーザー定義分類子関数**（Classification Function）を作成します。この関数は、通常のユーザー定義関数と同様、**CREATE FUNCTION** ステートメントを利用して次のように作成します（ステートメントは「クエリ エディタ」から実行します）。

```
USE master
go
CREATE FUNCTION [test分類関数] ()
 RETURNS sysname
 WITH SCHEMABINDING
AS
BEGIN
        DECLARE @grp_name AS sysname
        IF (SUSER_NAME() = 'testUser')
                SET @grp_name = 'testWorkloadG1'
        RETURN @grp_name
END
```

必ず master データベースへ接続

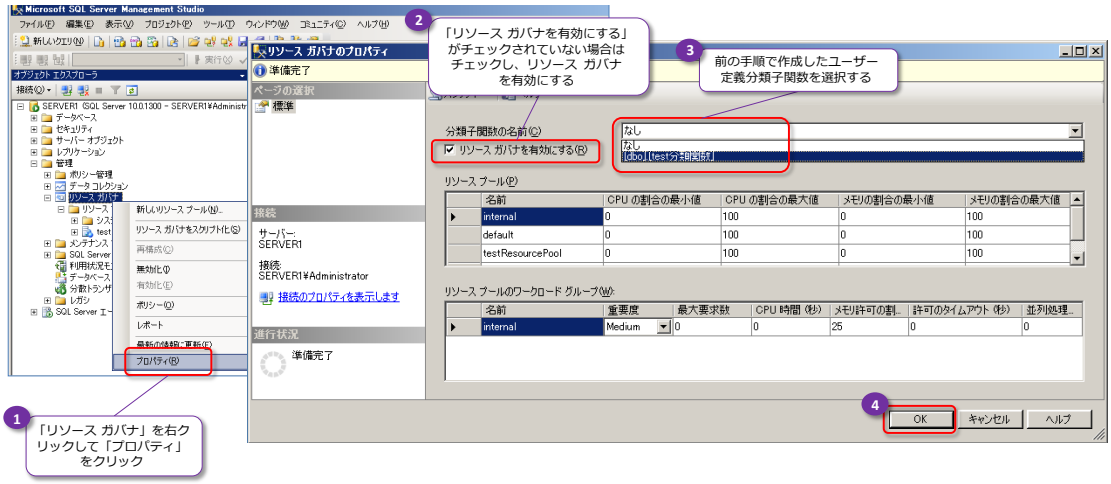
CREATE FUNCATION ステートメントを利用してユーザー定義関数を作成（関数名は任意）。戻り値は sysname で、WITH SCHEMABINDING を指定する

SUSER_NAME 関数で、ユーザー名を取得。**testUser** なら、**testWorkloadG1** を返すようにしている

Note : ユーザー定義分類子関数は master へ

ユーザー定義分類子関数は、master データベース内へ作成する必要があります。

5. 次に、作成した分類子関数をリソース ガバナへ設定するために、次のように［**リソース ガバナ**］を右クリックして［**プロパティ**］をクリックします。



プロパティ画面では、[リソース ガバナを有効にする] をチェックして、[**分類子関数の名前**] で作成した分類子関数（**test 分類関数**）を選択し、［OK］ボタンをクリックします。

以上でリソース ガバナの設定が完了です。

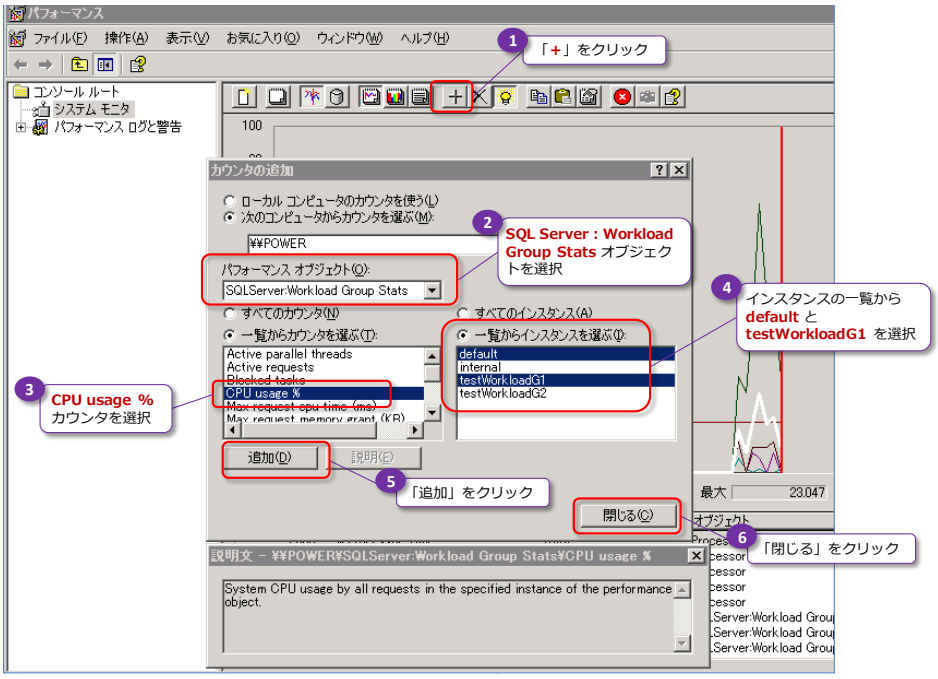
負荷ワークロードの実行と監視

次に、負荷の高いクエリを実行して、リソース ガバナの効果を確認してみましょう（この手順を試すには、SQL Server 2008 が混合認証モードで実行されている必要があります）。

6. まずは、ユーザー定義分類子関数で振り分けている「**testUser**」ログイン アカウントを次のように作成し、AdventureWorks データベースのユーザーとして設定し、db_owner ロールへ追加しておきます。

```
USE master
go

-- ログイン アカウントの作成
CREATE LOGIN testUser
 WITH PASSWORD = 'testUser123'
go

-- データベース ユーザーの作成
USE AdventureWorks
go
CREATE USER testUser
 FOR LOGIN testUser
go

-- db_owner ロールへの追加
EXEC sp_addrolemember 'db_owner', 'testUser'
go
```

7. 次に、ワークロード状況を監視するために、[スタート] メニューの［**管理ツール**］から［**パフォーマンス**］をクリックして、システム モニタを起動します。



システム モニタでは、ツールバーの［**+**］ボタンをクリックして［カウンタの追加］ダイアログを表示し、[パフォーマンス オブジェクト]で「**SQLServer: Workload Group Stats**」を選択、カウンタの一覧から「**CPU Usage %**」（CPU 利用率）を選択し、インスタンスの一覧から「**default**」と「**testWorkloadG1**」を選択して［閉じる］ボタンをクリックします。

8. 続いて、CPU 負荷の高いクエリを、（testUser からではなく）sysadmin ロールのメンバーから（管理者アカウントから）実行します。このクエリは、次のように記述またはサンプル ス

クリプト内の「**Step4-08_ Resource governor.sql**」ファイルからコピーして貼り付けて、実行します。

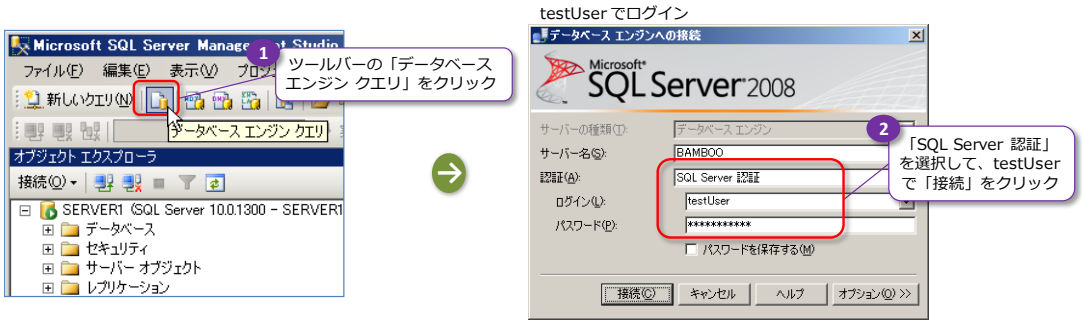
```
USE AdventureWorks
go
DECLARE @i int = 1
WHILE @i <= 100
BEGIN
   DBCC DROPCLEANBUFFERS  --  バッファのクリア
   SELECT * FROM
         Sales.SalesOrderDetail od
         INNER JOIN Sales.SalesOrderHeader oh ON od.SalesOrderID = oh.SalesOrderID
         INNER JOIN Production.Product p ON od.ProductID = p.ProductID
         INNER JOIN Production.ProductDocument pd ON p.ProductID = pd.ProductID

   DBCC DROPCLEANBUFFERS  --  バッファのクリア
   SELECT * FROM
         Sales.SalesOrderDetail od
         INNER JOIN Sales.SalesOrderHeader oh ON od.SalesOrderID = oh.SalesOrderID
         INNER JOIN Production.Product p ON od.ProductID = p.ProductID
         INNER JOIN Production.ProductDocument pd ON p.ProductID = pd.ProductID
         INNER JOIN Production.Document d ON pd.DocumentID = d.DocumentID
         WHERE  d.Title LIKE '%bbb%' OR d.FileName LIKE '%ccc%'
         OR d.DocumentSummary LIKE '%ddd%' OR od.CarrierTrackingNumber LIKE '%sss%'
   SET @i += 1
END
```

同じクエリを 100 回繰り返し実行

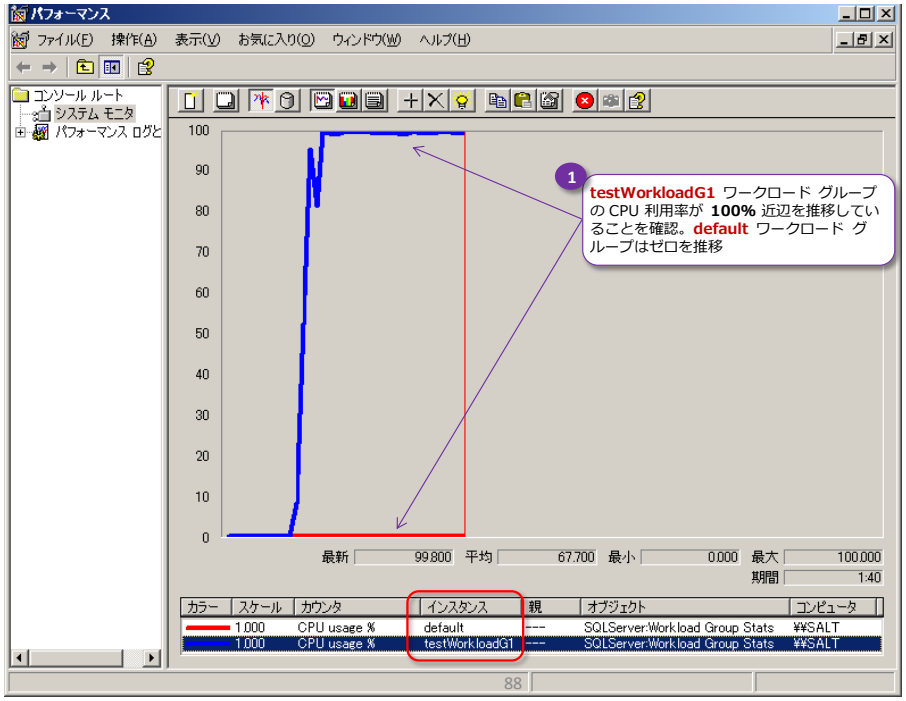
このクエリの実行時間は、環境によって異なりますが、Pentium M 1.2GHz の環境では約 3 分、Core2 Quad 2.4GHz の環境では約 30 秒かかります。クエリの実行が終わる前に、システム モニタへ戻り、CPU 利用率の状況を確認します。

sysadmin メンバーから実行されたクエリは、**default** ワークロード グループとして実行され、CPU 利用率が 100% 近くを推移していることを確認できます。

9. 上記のクエリの実行が完了した後、今度は、testUser ログイン（ユーザー）でクエリ エディタへ接続し、手順 8 で実行したのと同じクエリ（CPU 負荷の高いクエリ）を実行します。

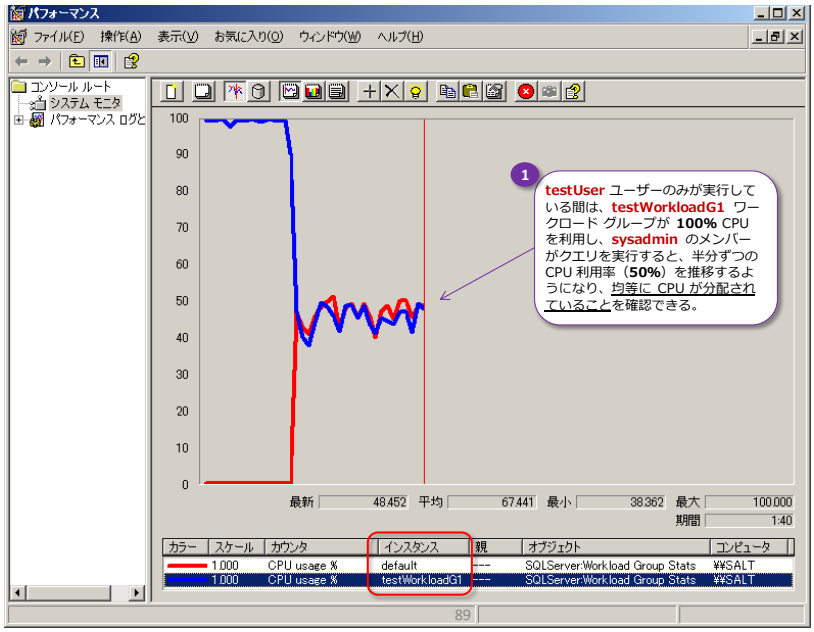


接続後にクエリを実行し、実行中にシステム モニタで CPU 利用率の状況を確認します。



testUser　ユーザーから実行されたクエリは、**testWorkloadG1** ワークロード グループとして実行されていることを確認できます。

10. 次に、上記のクエリ（testUser ユーザーが実行したクエリ）の実行中に、sysadmin ロールのメンバーから同じクエリを実行し、そのときの状況をシステム モニタで確認します。



sysadmin のメンバーがクエリを実行すると、手順 9 で testUser が実行しているクエリと半分ずつの CPU 利用率（50%）を推移するようになり、CPU が均等に分配されていることを確認できます。

11. クエリの実行中に、リソース プールの調整（CPU 利用率の制限）を行います。sysadmin のメンバーでログインした接続をもう 1 つ作成し、次のように **ALTER RESOURCE POOL** ステートメントを実行して **testResourcePool** の **Max CPU Percent** を **5** へ変更します。

```
ALTER RESOURCE POOL testResourcePool
 WITH ( max_cpu_percent = 5 )

ALTER RESOURCE GOVERNOR RECONFIGURE
```

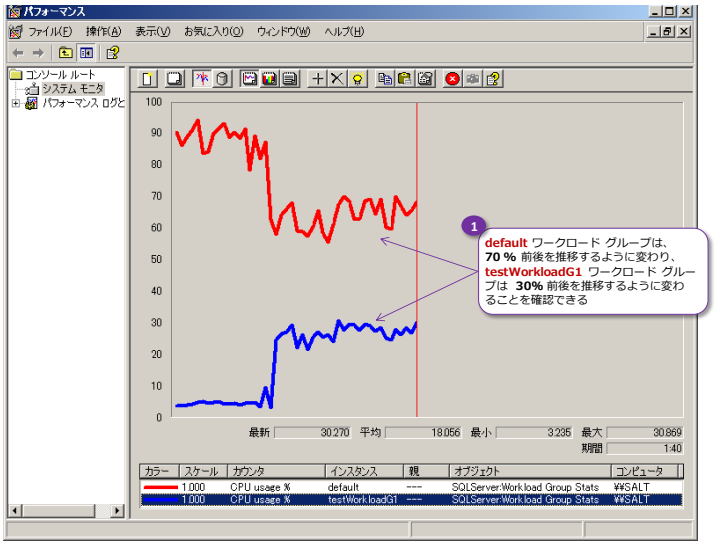
リソース プールの調整後、システム モニタで状況を確認します。

default ワークロード グループは、**95%** 前後を推移するように変わり、**testWorkloadG1** ワークロード グループは **5%** 前後を推移するように変わることを確認できます。

12. さらにリソース プールの調整 （CPU 利用率の制限）を行います。次のように ALTER RESOURCE POOL ステートメントを実行して testResourcePool の Max CPU Percent を **30** へ変更します。

```
ALTER RESOURCE POOL testResourcePool
 WITH ( max_cpu_percent = 30 )

ALTER RESOURCE GOVERNOR RECONFIGURE
```

リソース プールの調整後、システム モニタで状況を確認します。



93

default ワークロード グループは、**70%** 前後を推移するように変わり、**testWorkloadG1** ワークロード グループは **30%** 前後を推移するように変わることを確認できます。

Note： リソース ガバナの注意点

リソース ガバナは、ユーザーが SQL Server への接続を確立する時（ログイン時）に評価されます。そのため、分類子関数（Classification Function）の作成前に既にログインしているユーザーには有効になりません。また、Execute As ステートメントによるユーザー権限の借用を行っている場合にも、ログインが伴わないため、リソース ガバナは有効にはなりません。

分類子関数内で利用できる関数

分類子関数は、ユーザーが SQL Server へ接続を確立する時（ログイン時）に実行される関数なので、利用できる関数が次の種類に限定されています。

・HOST_NAME()
・APP_NAME()

・SUSER_NAME()
・SUSER_SNAME()
・IS_SRVROLEMEMBER()
・IS_MEMBER()

アプリケーション名（APP_NAME 関数）での振り分け

APP_NAME 関数は、アプリケーション名を取得できる便利な関数なので、この関数を利用すればアプリケーションごとにワークロード グループを分けられるようになります。ADO や ADO.NET を利用している場合は、アプリケーション名を次のように接続文字列（Connection String）の Application Name で指定することができます。

```
'' ADO (WSH) の接続文字列の使用例 (アプリケーション名を batch と指定)
Dim cn
cn = WScript.CreateObject("ADODB.Connection")
cn.Open("Provider=SQLOLEDB;" _
         & "Server=salt;" _
         & "Database=AdventureWorks;" _
         & "Application Name=batch;" _
         & "Integrated Security=SSPI;")
```

したがって、アプリケーション（接続文字列）から受け取った「Application Name」を利用して、ワークロード グループを振り分けるように、次のように分類子関数を作成することもできます。

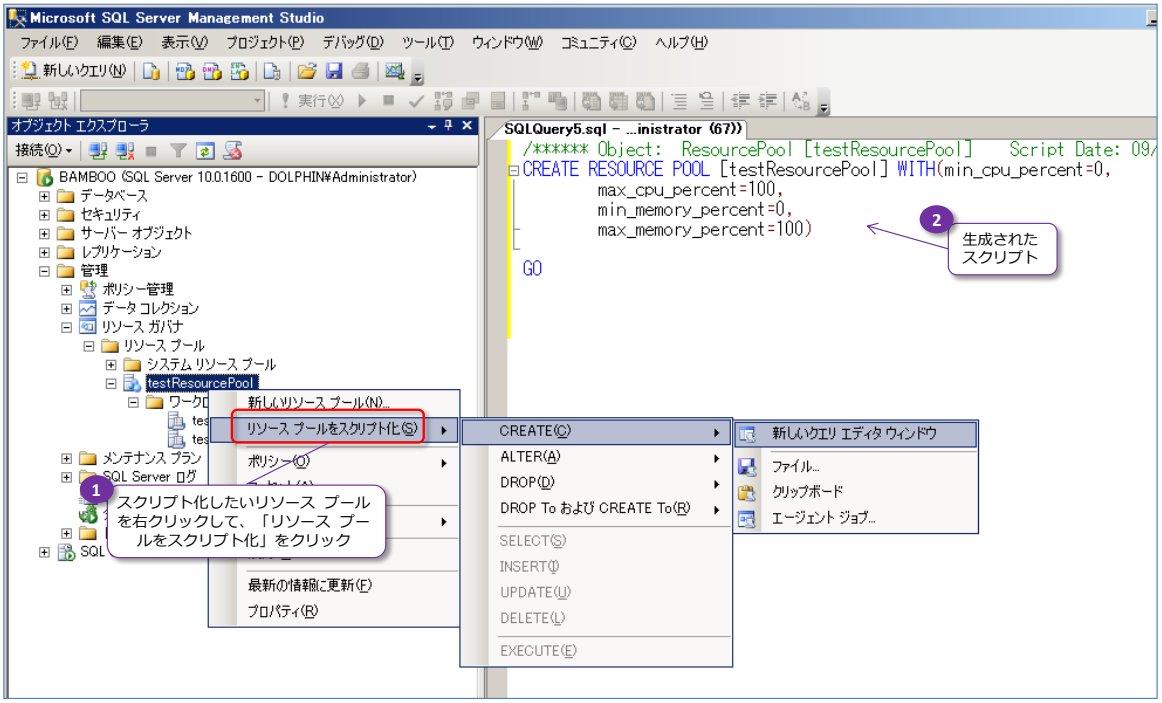
```
-- APP_NAME 関数で振り分けるユーザー定義分類子関数の作成
USE master
go
CREATE FUNCTION dbo.rg_appName()
 RETURNS sysname
 WITH SCHEMABINDING
AS
 BEGIN
        DECLARE @grp_name AS sysname
        IF (APP_NAME() = 'batch')
                SET @grp_name = 'testWorkloadG1'
        RETURN @grp_name
 END
go
```

これらを試すためのスクリプトは、サンプル スクリプト内の「**Step4-08_Resource governor _APP_NAME.sql**」ファイルと「**Step4-08_Resource governor _APP_NAME.vbs**」（WSH ファイル）へ記述してありますので、ぜひ試してみてください。

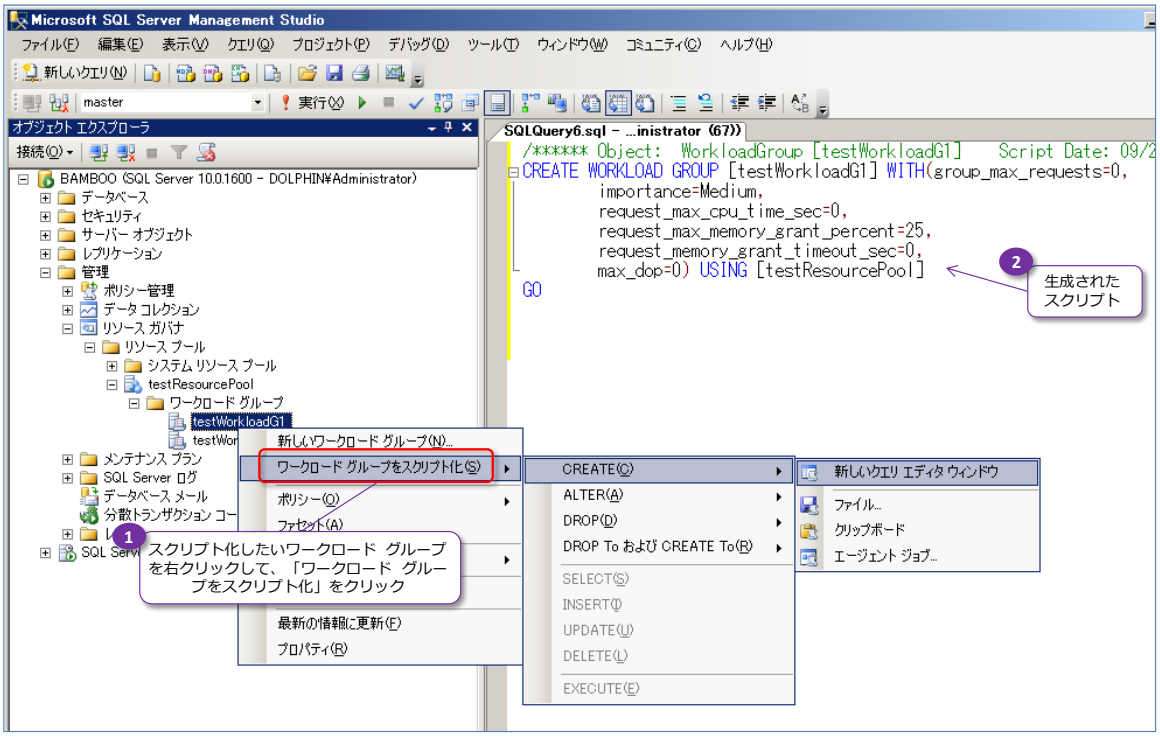
スクリプト生成

Management Studio の GUI 操作で作成したリソース プールやワークロード グループは、次のようにスクリプト生成機能を利用することで、SQL ステートメントを生成することもできます。

リソース プールに対してスクリプト生成



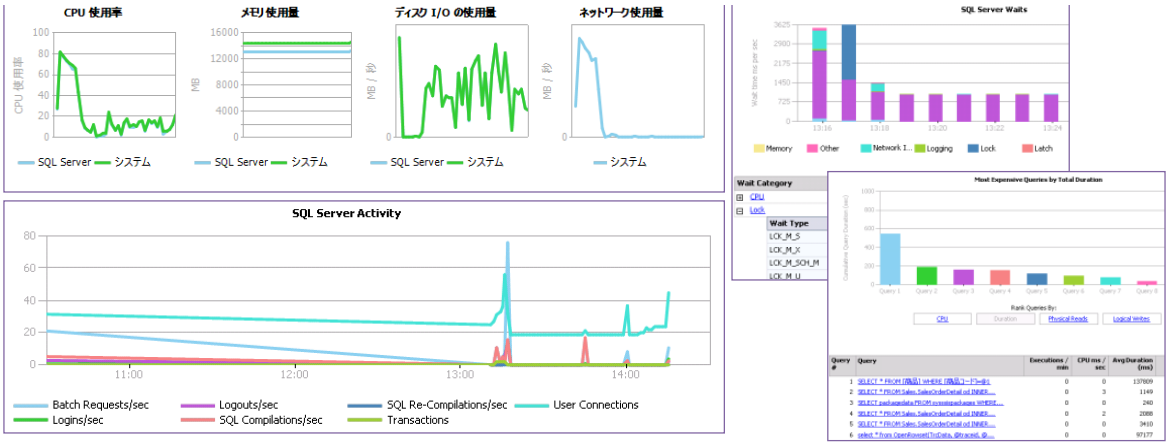
ワークロード グループに対してスクリプト生成



4.9　パフォーマンス データ コレクションによるパフォーマンス監視

パフォーマンス データ コレクション

パフォーマンス データ コレクションは、グラフィカルなパフォーマンス監視が行える機能で、Management Studio のレポートとして実装されています。



パフォーマンス データ コレクションは、SQL Server 2005 SP2（Service Pack 2）のアドインとして提供されていた Performance Dashboard の強化版で、Performance Dashboard が今の状態（リアルタイム）のみしか監視できなかったのに対して、SQL Server 2008 のパフォーマンス データ コレクションでは、データ コレクション機能と連動して、過去のデータまでさかのぼって監視することができます。

パフォーマンス データ コレクションのレポート

パフォーマンス データ コレクションで提供されるレポートは、次の 3 つで、それぞれ組み込みの データ コレクション（**システム データ コレクション**）と連動したレポートとなっています。

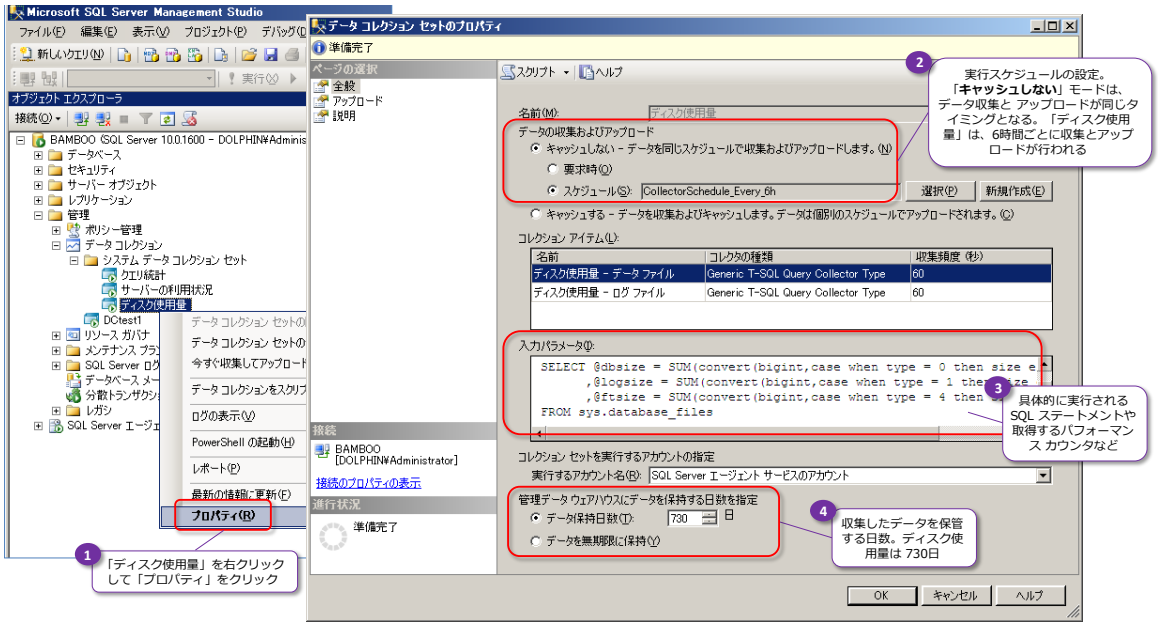
レポート名	対応するシステム データ コレクション	説明
ディスク使用量の概要	ディスク使用量	データとログの使用量の推移を表示
クエリ統計の履歴	クエリ統計	クエリの統計情報（CPU 実行時間の多いクエリや、実行時間の長いクエリ）などを表示
サーバーの利用状況の履歴	サーバーの利用状況	メモリや CPU、ディスク、Wait Stats（各種の待ちに関する情報）などの使用状況を表示

設定手順

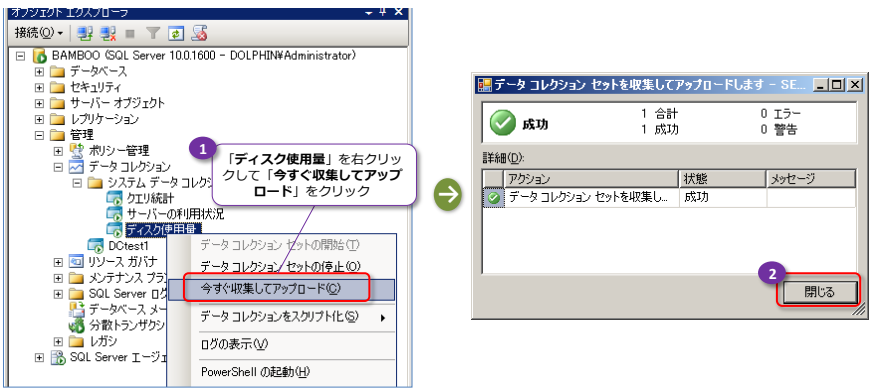
それでは、パフォーマンス データ コレクションを試してみましょう。この機能を試すには、「4.7 データ コレクション」での手順 1 ～手順 11 までを実行しておく必要があるので、まだ実行していない場合は実行しておいてください。この手順を実行することで データ コレクション用のデータ ウェアハウス（データの収集先となるデータベース）が構成されて、システム データ コレクションが自動的に実行される（パフォーマンス データ コレクション レポートのもととなるデータの収集が行われる）ようになります。システム データ コレクションでは、具体的には次の情報が定期的に収集されています。

コレクション セット名	主な収集している内容	収集頻度	保持期間
ディスク使用量	・database_files ・DBCC SQLPERF (LOGSPACE) ・partitions、allocation_units	6時間ごと	730日
サーバーの利用状況	・主要なパフォーマンス カウンタ ・dm_os_wait_stats ・dm_os_latch_stats ・dm_os_schedulers ・dm_os_process_memory ・dm_os_memory_nodes / clerks ・dm_os_sys_memory / info ・dm_io_virtual_file_stats	60秒ごと アップロード は 15分ごと	14日
クエリ統計	・dm_exec_sessions と dm_exec_requests、dm_os_tasks dm_os_waiting_tasks の JOIN	10秒ごと アップロード は 15分ごと	14日
	・dm_exec_query_stats ・dm_exec_text_query_plan	15分ごと (アップロード時)	

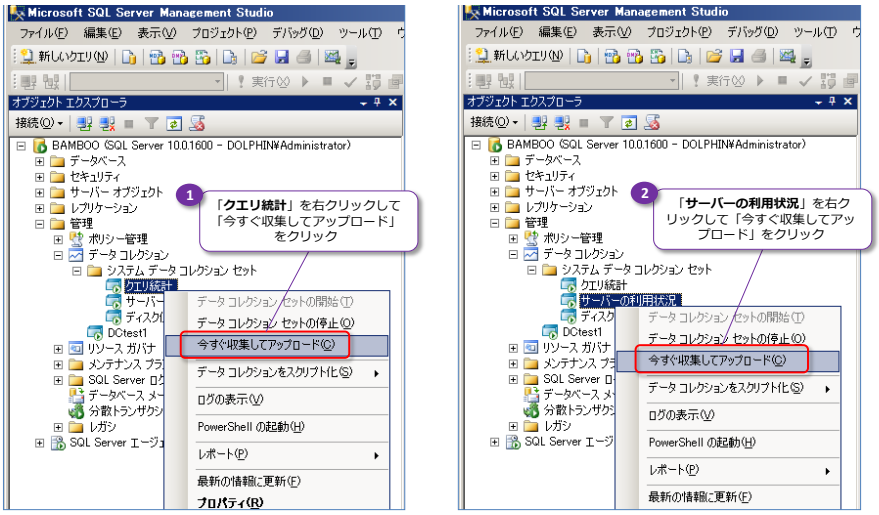
これらの設定は、それぞれのデータ コレクションを右クリックして「**プロパティ**」をクリックし、プロパティを参照することで確認できます。たとえば、「**ディスク使用量**」データ コレクションのプロパティは、次のように確認できます。



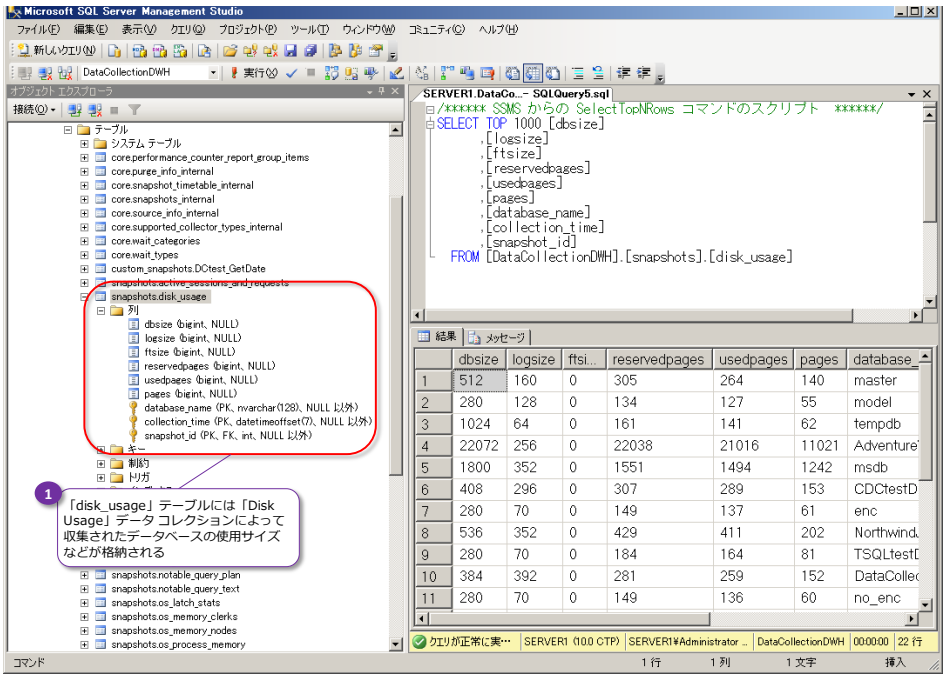
1. 「**ディスク使用量**」データ コレクションは、プロパティで確認したように 6 時間ごとにデータ収集が行われているので、今すぐデータ収集（とアップロード）を実行するために、次のように［**今すぐ収集してアップロード**］をクリックします。



2. 「**クエリ統計**」と「**サーバーの利用状況**」データ コレクションについても、同じように［**今すぐ収集してアップロード**］をクリックして、今すぐデータ収集を実行します。

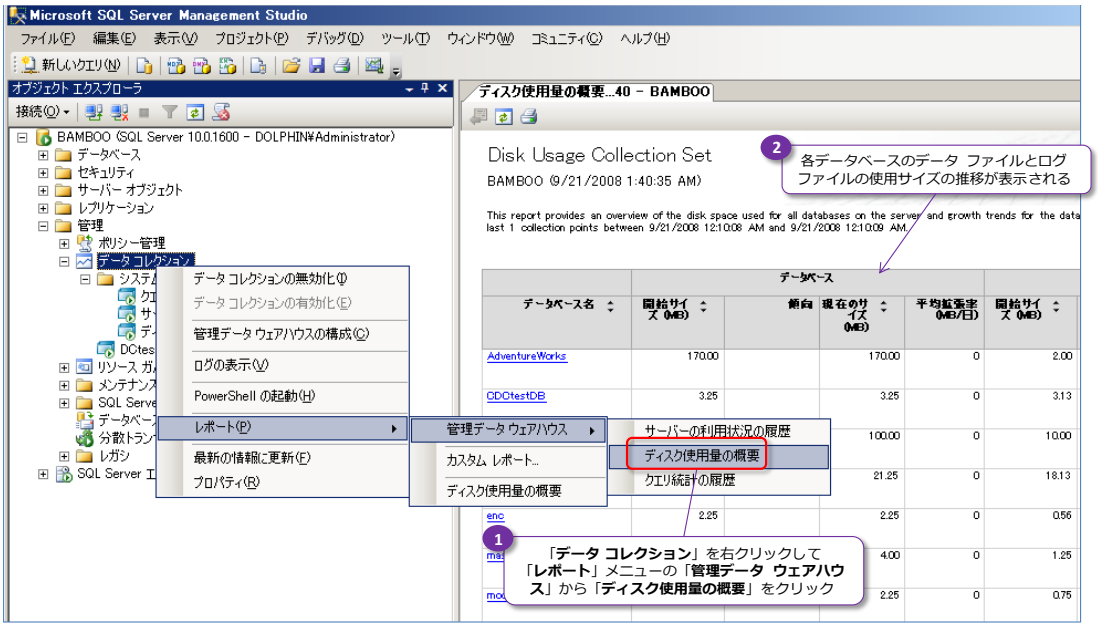


3. これにより、データ コレクション用のデータ ウェアハウス内のテーブルが更新され、たとえば **snapshots** スキーマの「**disk_usage**」テーブルには、「ディスク使用量」データ コレクションによって収集されたデータベースの使用サイズなどが格納されています。

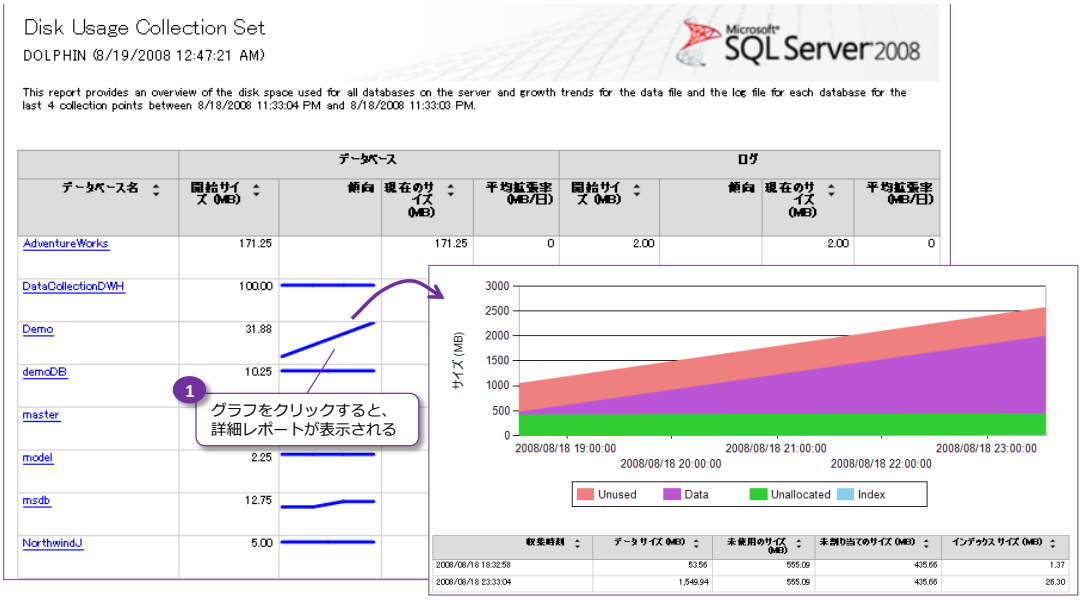


ディスク使用量の概要レポート

4. 次に、［**データ コレクション**］を右クリックして、［**レポート**］メニューの［**管理データ ウェアハウス**］から［**ディスク使用量の概要**］をクリックして、ディスク使用量の概要レポートを表示します。



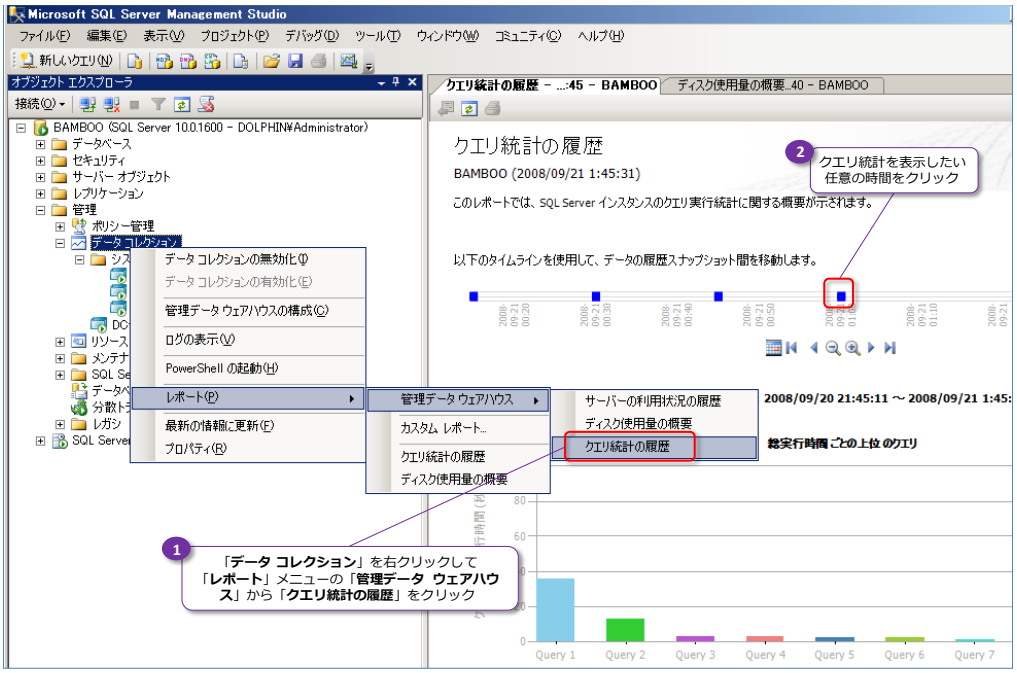
このレポートは、グラフをクリックすると、詳細レポートを表示することも可能です。



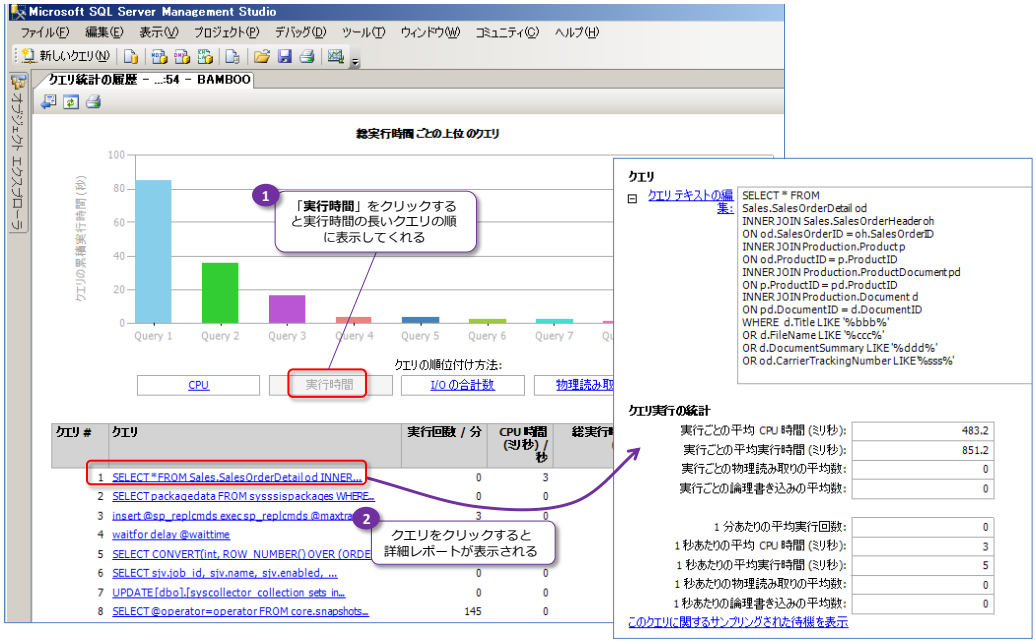
これらのレポートは、「ディスク使用量」データ コレクションによって収集されたデータ（データ ファイルやログ ファイルの使用サイズ）をもとに作成されたものです。

クエリ統計の履歴レポート

5. 次に、[**データ コレクション**] を右クリックして、[**レポート**] メニューの [**管理データ ウェアハウス**] から [**クエリ統計の履歴**] をクリックし、クエリ統計の履歴レポートを表示してみましょう。



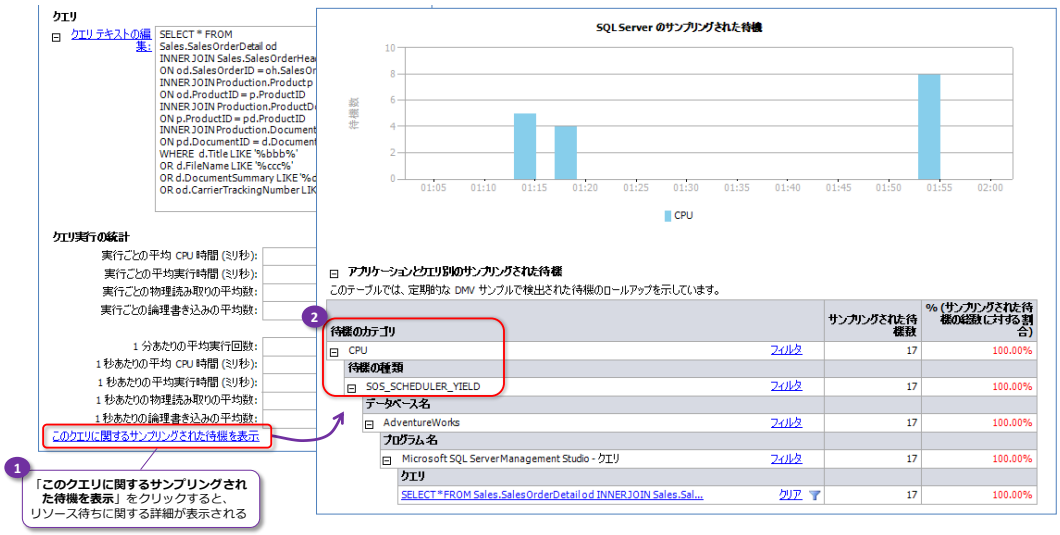
このレポートでは、実行時間（Duration）の長いクエリや、CPU 利用時間の長いクエリの順にクエリを一覧してくれ、そのクエリの詳細を表示できるようになっています。



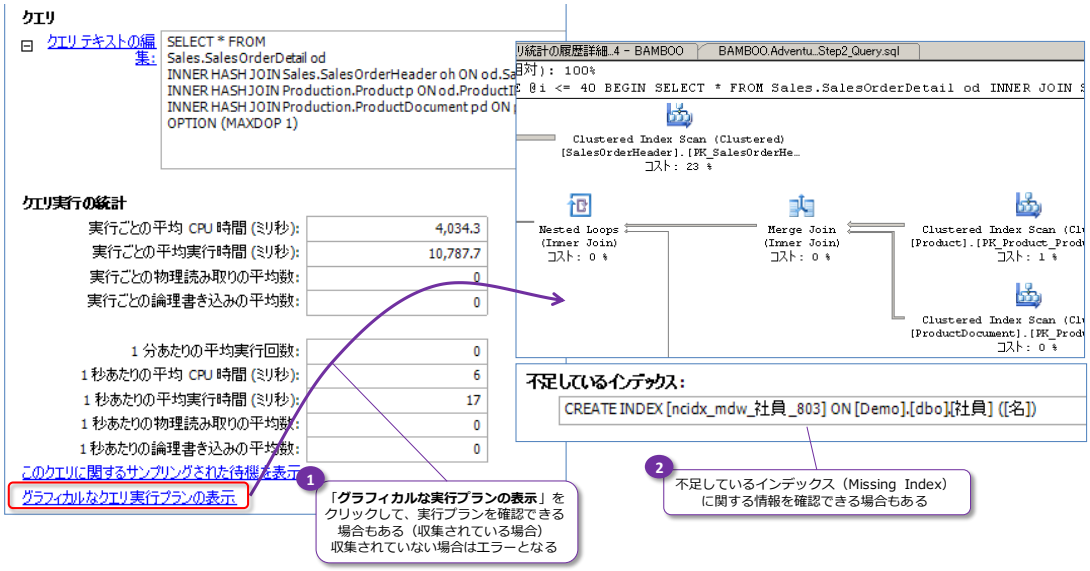
前のトピック「4.8　リソース ガバナ」では、CPU 負荷が高く、実行時間の長いクエリをたくさん実行しましたので、それらが上位にリストされることを確認できると思います。

リソース待ちに関する詳細とグラフィカル実行プランの表示

クエリの詳細画面では、次のように「**このクエリに関するサンプリングされた待機を表示**」ハイパーリンクをクリックすると、リソース待ちに関する詳細を確認することができます。

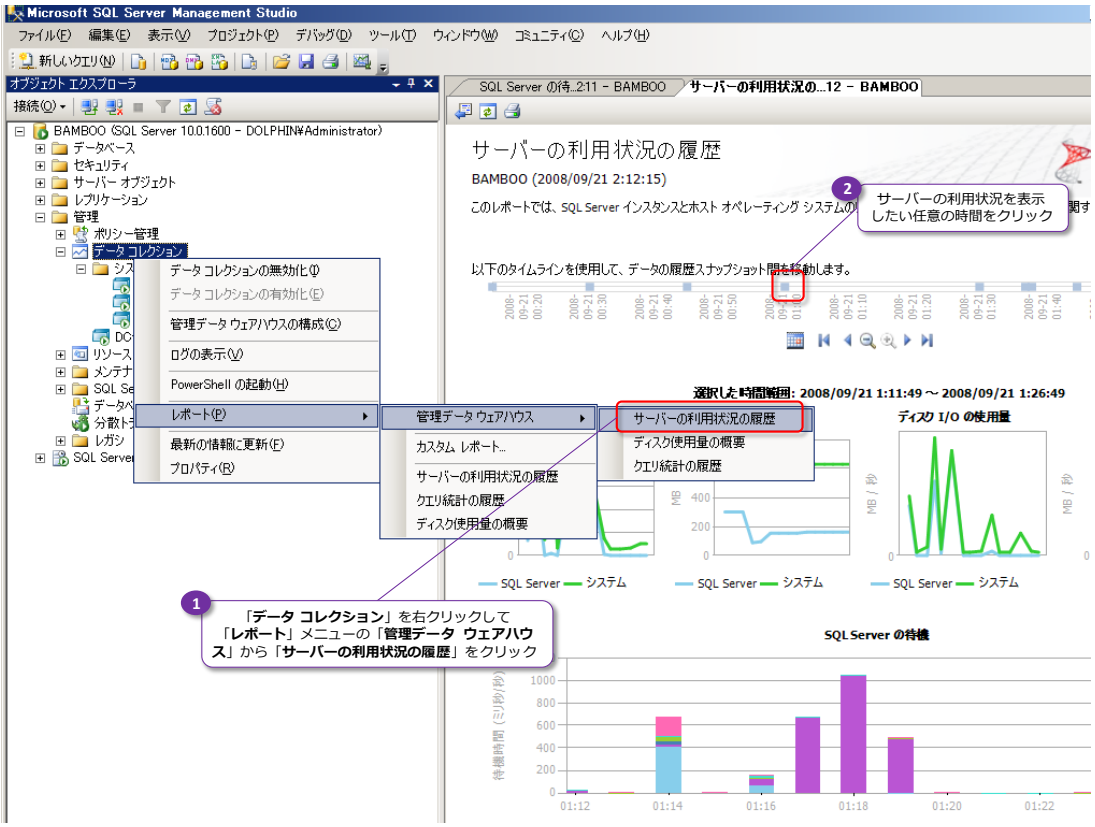


また、クエリの詳細画面では、次のように「グラフィカルなクエリ実行プランの表示」のハイパーリンクをクリックすると、グラフィカル実行プランを確認することもできます（収集されている場合のみ）。

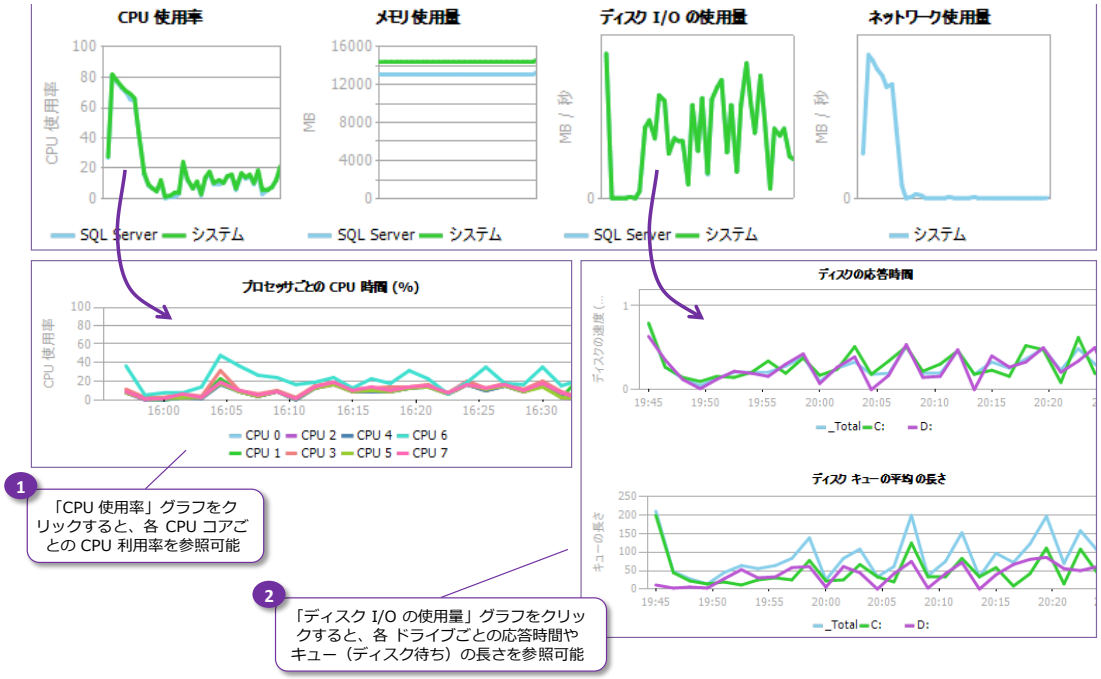


サーバーの利用状況の履歴レポート

6. 次に、[**データ コレクション**] を右クリックして、[**レポート**] メニューの [**管理データ ウェアハウス**] から [**サーバーの利用状況の履歴**] をクリックし、サーバーの利用状況の履歴レポートを表示してみましょう。

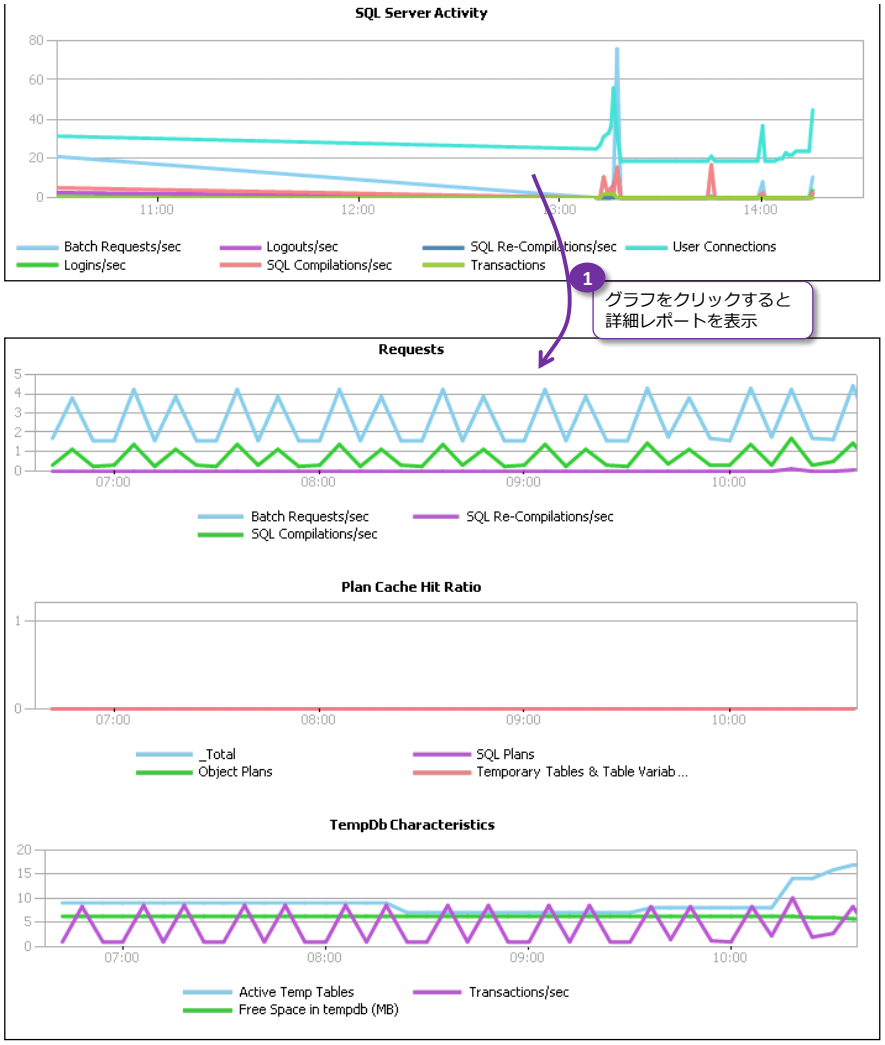


このレポートでは、CPU 利用率やメモリ使用量、ディスク（I/O）の使用量、SQL Server の動作状況（Wait Stats や各種パフォーマンス カウンタの状況）などを表示することができます。また、各グラフをクリックすると、その詳細を表示できるようになっていて、たとえば、次のように CPU 利用率（**%CPU**）のグラフをクリックすると、その詳細（各 CPU コアごとの CPU 利用率など）を確認できるようになります。



SQL Server の動作状況の表示（Wait Stats やロック待ちなど）

サーバーの利用状況の履歴レポートでは、「**User Connections**」や「**Batch Request/sec**」、「**Logins/sec**」、「**Compilation/sec**」といった、SQL Server 関連の各種パフォーマンス カウンタの状況なども確認することができます。



このように、パフォーマンス データ コレクション機能を利用すると、リソースの使用状況を過去にさかのぼって参照することができるので、パフォーマンス チューニング時のボトルネックの発見や、トラブル シューティング時のトラブル原因の究明に大変役立ちます。

4.10 ポリシー ベースの管理

ポリシー ベースの管理（Poliy Based Management）

ポリシー ベースの管理は、Windows（Active Directory）におけるグループ ポリシー機能と同じように、管理者が設定したポリシーを利用して複数の SQL Server を集中管理できる機能です。ポリシー ベースの管理では、SQL Server 2008 だけでなく、SQL Server 2000 / 2005 を対象にすることもできるので、企業内のすべての SQL Server を集中管理する目的として利用することもできます。また、ポリシー ベースの管理で作成したポリシーは、そのポリシーを強制（設定値を強制）したり、定期的なチェックとレポートを作成したりできるようになります。

ポリシー ベースの管理の使用例

ポリシー ベースの管理は、次のようにさまざまな目的で利用することができます。

- セキュリティ強化の徹底
- パフォーマンス関連の環境設定パラメータを一括設定（sp_configure や DB オプションの値をリモートから強制変更可能）
- オブジェクトの名前付けルールの徹底（プレフィックス（接頭辞）の強制：ストアド プロシージャの名前は MyCompany_sp_xxxx とするなど）
- イベント ビューアに対して WMI クエリを実行し、エラーが発生していないかどうかをチェックする

セキュリティ強化としてのポリシーの利用

ポリシー ベースの管理では、Step 4.5 で説明した **SQL Server Audit**（監査）が有効になっているかどうかをチェックしたり、**xp_cmdshell コマンド**や **Ad hoc 分散クエリ**の実行を禁止したりする目的（セキュリティ強化の目的）で利用することができます。以前のバージョンでは、**セキュリティ構成ツール**（Surface Area Configuration： SAC）で設定していたセキュリティ設定などを、ポリシーとして集中管理できるようになります。

なお、SQL Server 2008 では、セキュリティ構成ツールが削除され、ポリシー ベースの管理機能を利用して、同様の設定を行うように変更されています。

設定手順

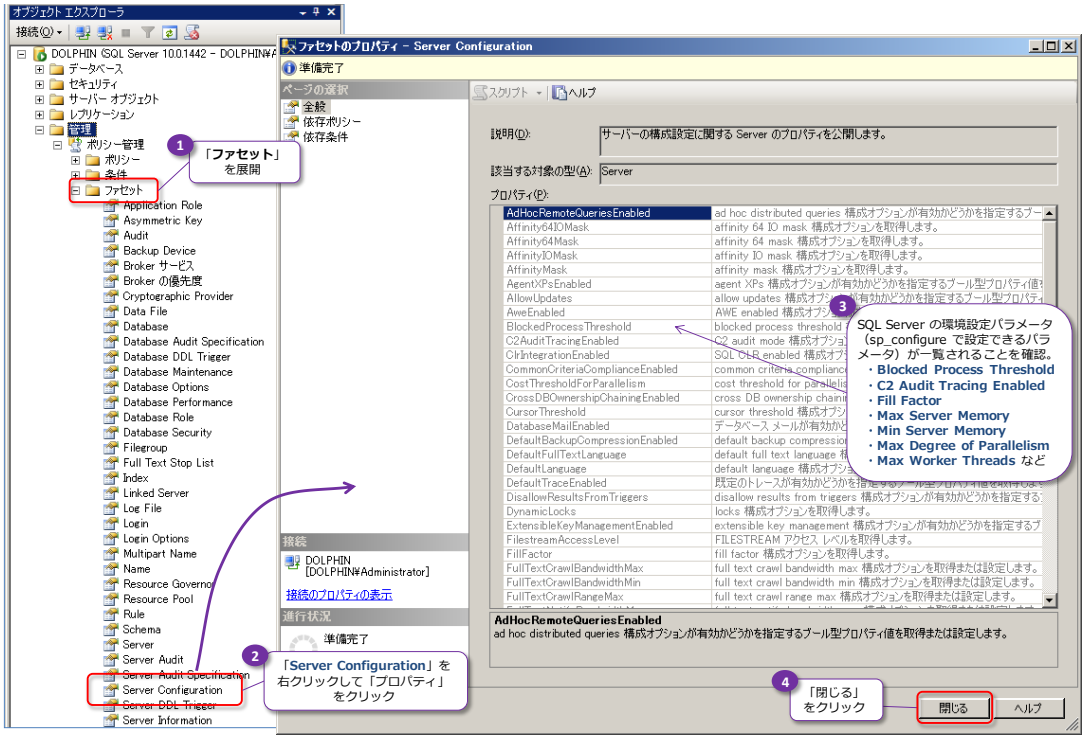
それでは、ポリシー ベースの管理を試してみましょう。設定のおおまかな流れは次のとおりです。

1. **ファセット**（Facet）をもとに、**条件**（Condition）を作成
2. **条件**をもとに、**ポリシー**（Policy）を作成

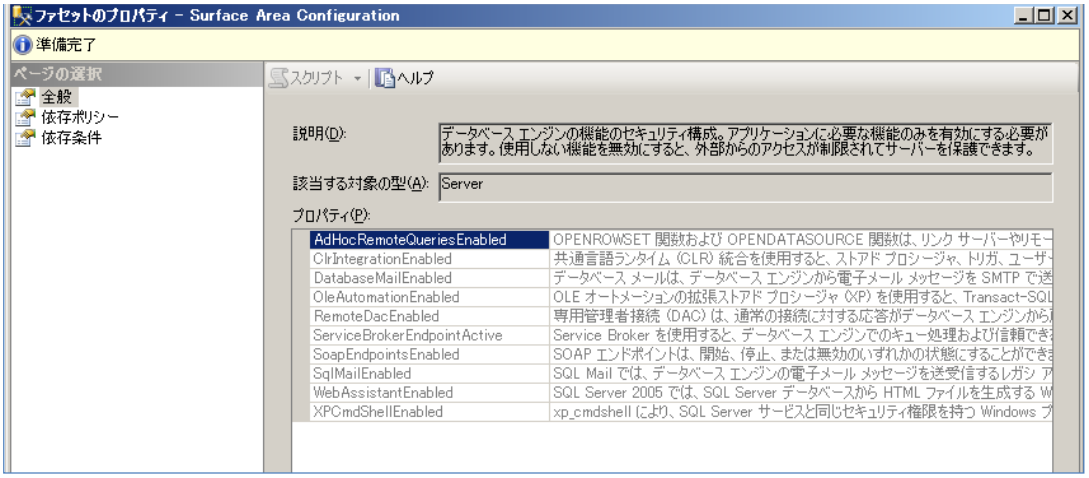
このように ポリシー ベースの管理では、最初にファセット（Facet）から条件（Condition）を作成するので、まずはどういったファセットがあるのかを確認しておく必要があります。

ファセットの確認

1. ファセットを確認するには、次のように［**管理**］フォルダの［**ポリシー管理**］から［**ファセット**］フォルダを展開してファセットの一覧を表示し、中身を確認したいファセットを右クリックして［**プロパティ**］をクリックします。たとえば、**Server Configuration** ファセットの中身には、次のように「**Max Server Memory**」や「**Max Degree of Parallelism**」、「**Max Worker Threads**」などがプロパティ一覧へ表示され、SQL Server の環境設定パラメータ（sp_configure で設定できるパラメータ）が一覧されることを確認できます。

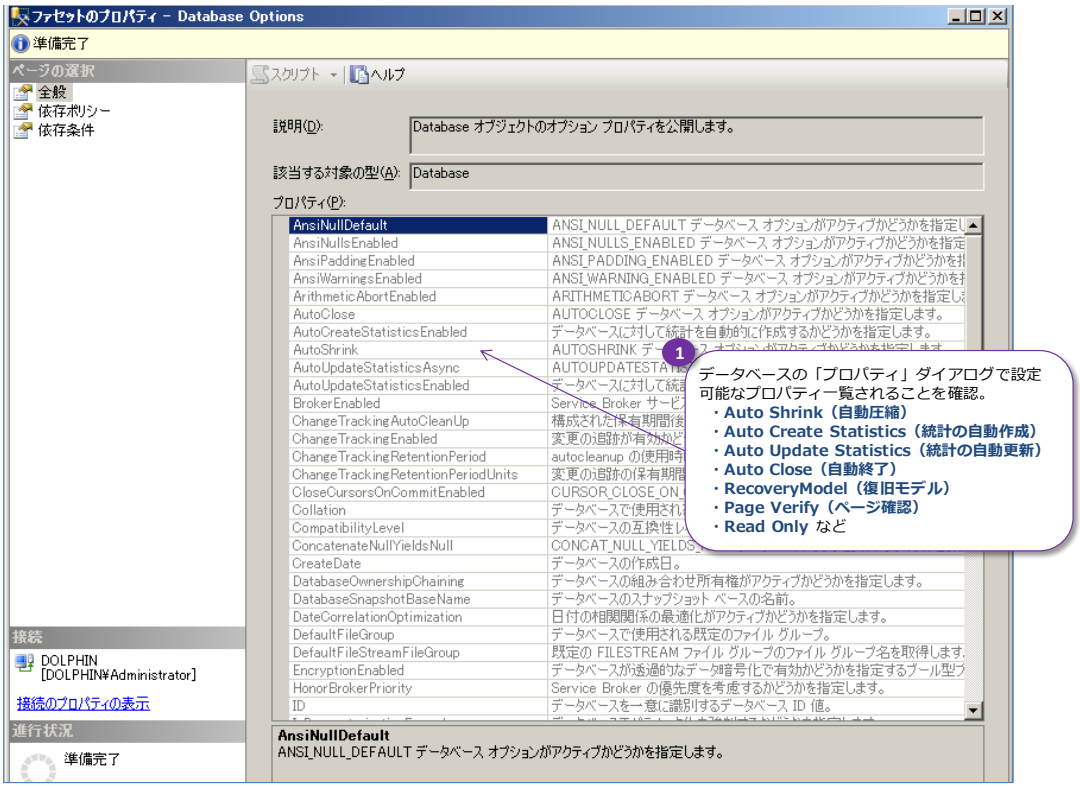


2. 次に、Surface Area Configuration ファセットの中身を確認してみます。



これは、以前のバージョンの**セキュリティ構成ツール**（Surface Area Configuration）で設定していたセキュリティ設定に関する項目が一覧されます。

次に、**Database Options** ファセットの中身を確認してみます。

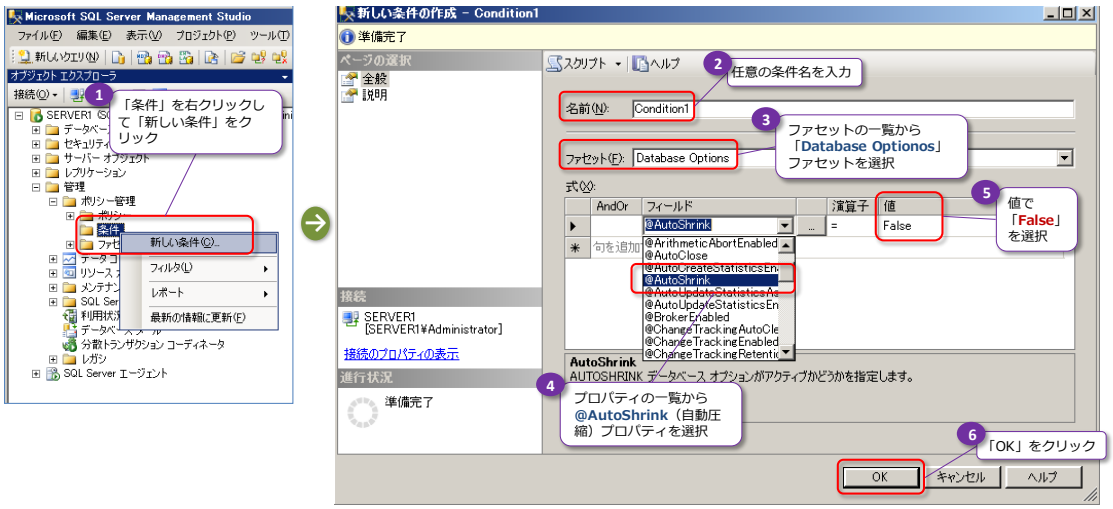


このファセットでは、「**Auto Shrink**」（自動圧縮）や「**Auto Create Statistics**」（統計の自動作成）といった、データベースの「プロパティ」ダイアログで設定可能なプロパティが一覧されることを確認できます。

自動圧縮設定が無効かどうかを確認するポリシー

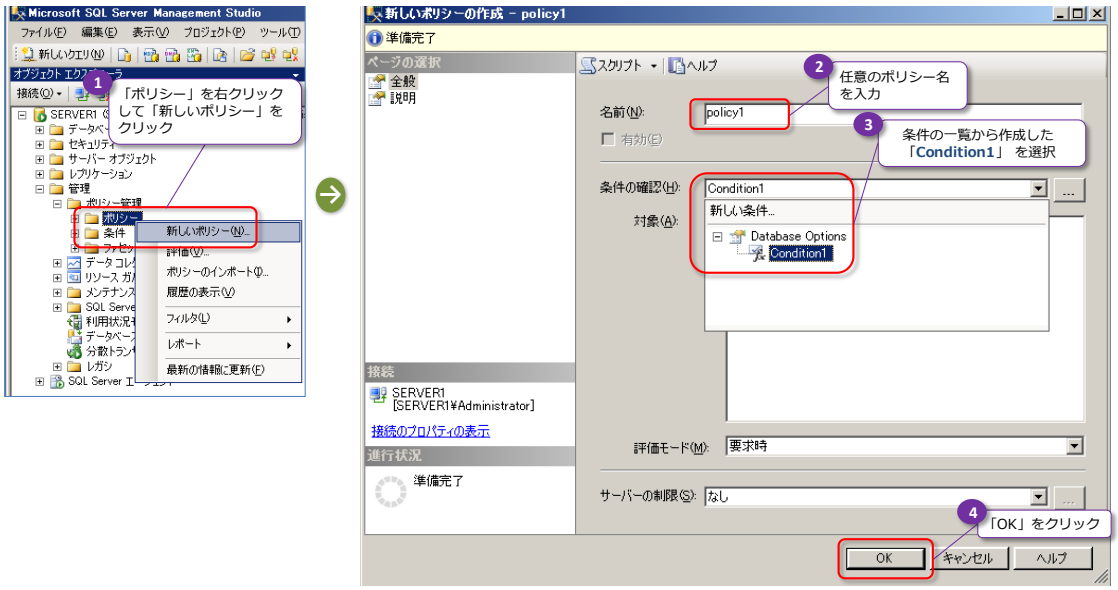
次に、Database Options ファセットを利用して、自動圧縮（Auto Shrink）設定が無効（False）になっていることを確認するためのポリシーを作成してみましょう（パフォーマンスを考慮した場合は、自動圧縮が無効になっていることが望ましいので、それを確認するためのポリシーを作成します）。

1. まずは、**Database Option** ファセットをもとに条件を作成します。条件を作成するには、次のように［**条件**］フォルダを右クリックして［**新しい条件**］をクリックします。



［**名前**］へ任意の名前（conditin1 など）を入力して、［ファセット］で「**Database Option**」ファセットを選択し、［**式**］の［**フィールド**］一覧から「**@autoshrink**」を選択し、［**値**］で「**False**」を選択し、［OK］をクリックします。これにより、自動圧縮が無効（False）になっているかどうかを確認するための条件を作成できます。

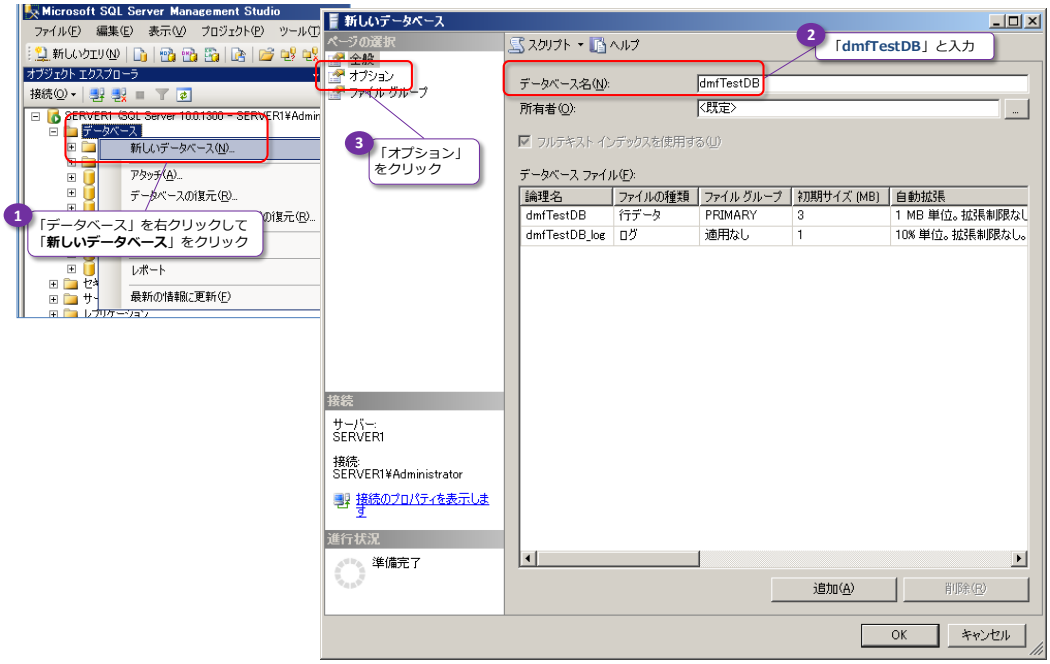
2. 次に、作成した条件をもとにポリシーを作成します。ポリシーを作成するには、次のように［**ポリシー**］フォルダを右クリックして［**新しいポリシー**］をクリックします。



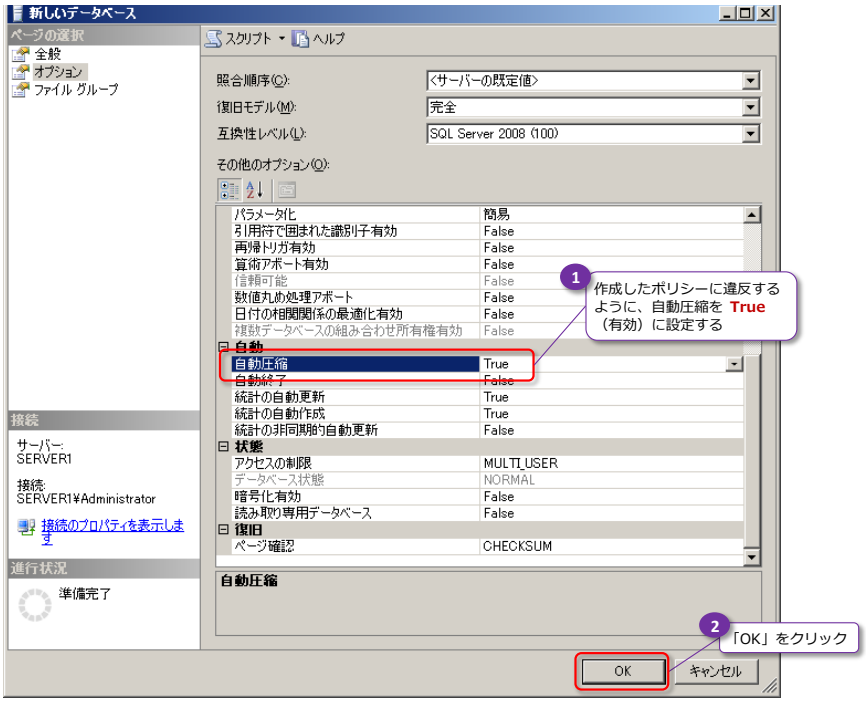
［**名前**］へ任意の名前（policy1 など）と入力して、［**条件の確認**］で作成した条件（Database Option ファセットの下の「**condition1**」）を選択し、［OK］ボタンをクリックします。

以上でポリシーの作成が完了です。

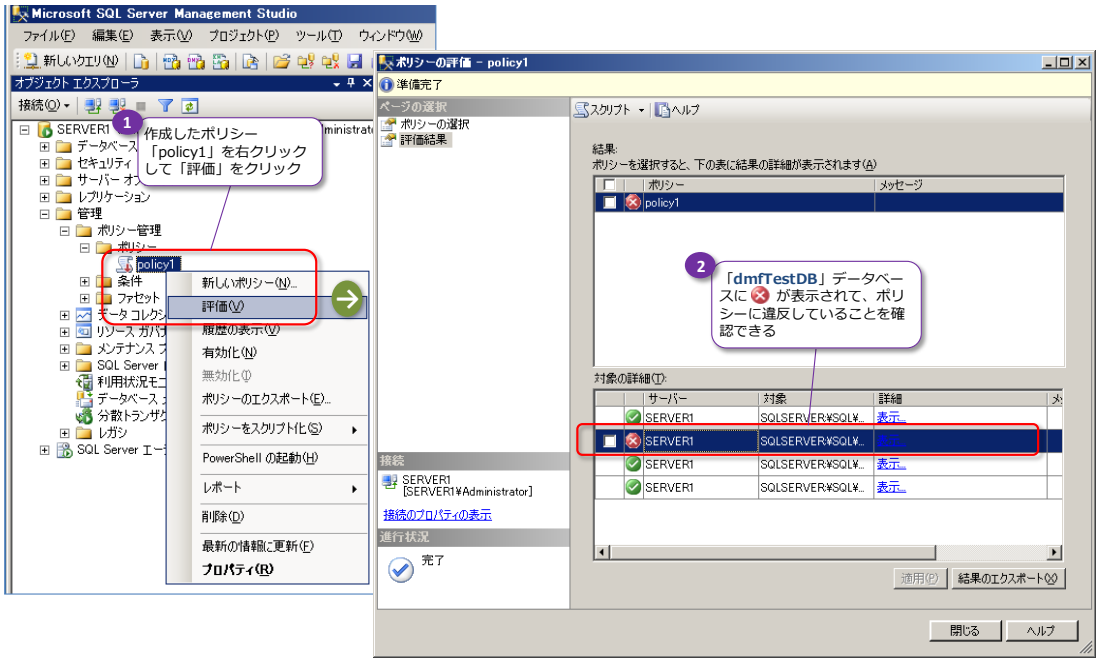
3. 次に、ポリシーの効果を試すために「**dmfTesetDB**」という名前のデータベースを作成します。



［**オプション**］ページでは、［**自動圧縮**］を「**True**」へ設定して、ポリシーへ違反するようにし、［OK］をクリックします。

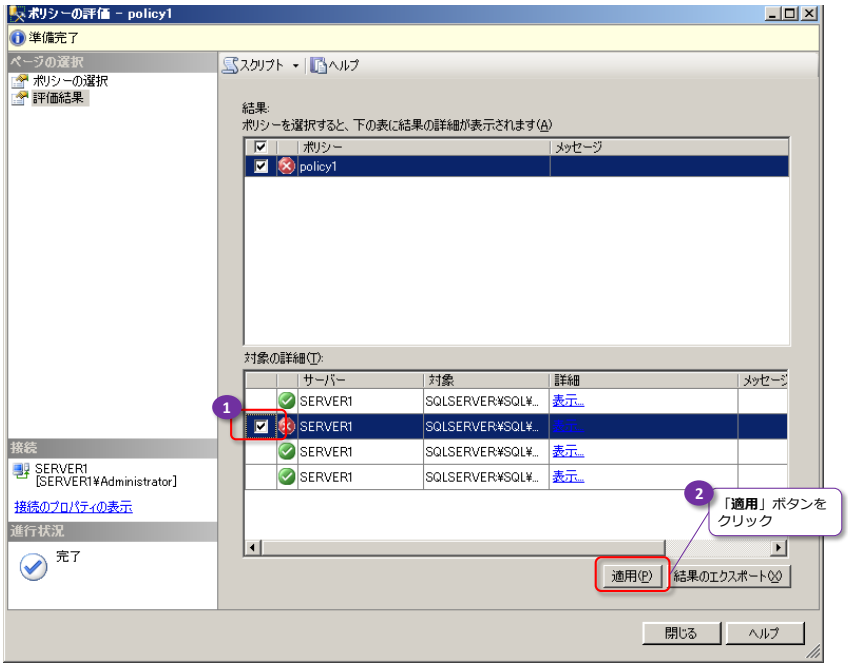


4. データベースの作成が完了したら、次のように作成したポリシー（**policy1**）を右クリックして［**評価**］をクリックし、ポリシーを実行します。

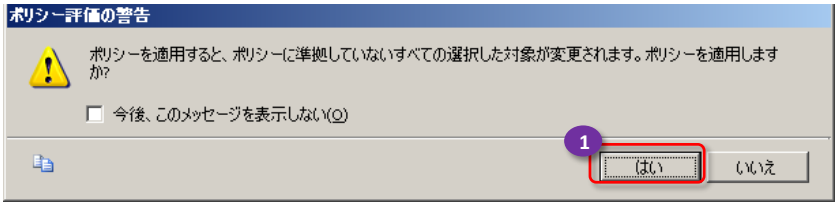


dmfTesetDB データベースがポリシーに違反したデータベースとしてリストアップされることを確認できます。

5. 次に、同じダイアログで、次のようにポリシー違反リストの左横のチェック ボックスをチェックし、［**適用**］ボタンをクリックします。

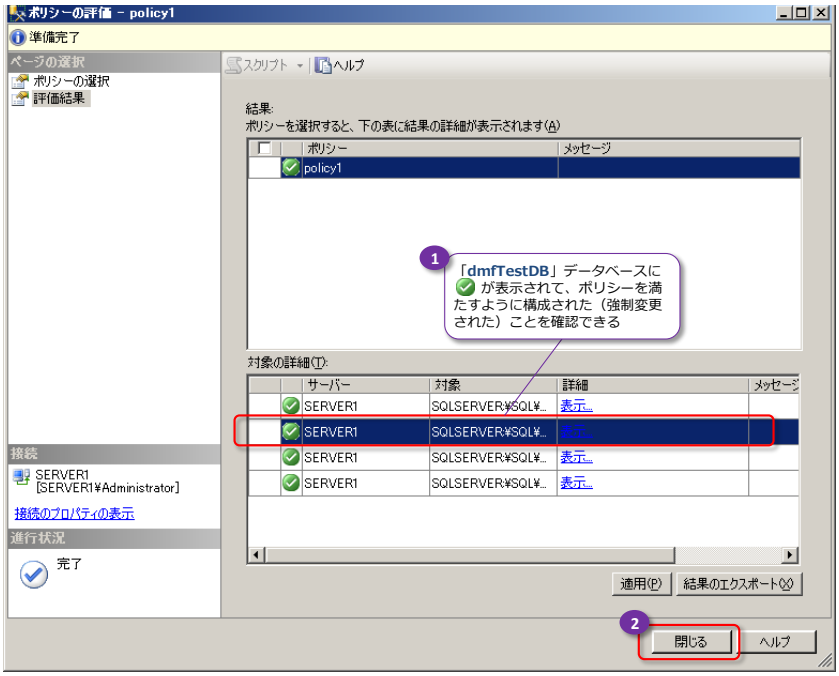


［適用］ボタンをクリックすると、［ポリシー評価の警告］ダイアログが表示されるので、［はい］ボタンをクリックします。

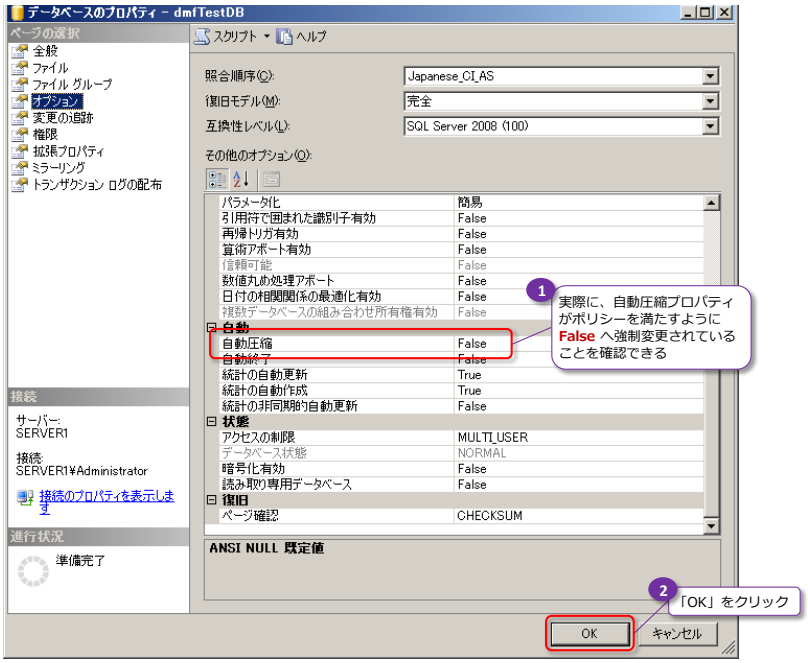


これにより、ポリシーの設定値を強制適用することができます。

適用後はポリシーに違反していないものとしてリストアップされることを確認できます。確認後、［閉じる］をクリックしてダイアログ ボックスを閉じます。



6. 実際に、データベースのオプションが強制適用されたことを確認しておきましょう。



ベスト プラクティス ポリシーのインポート

ポリシーには、**ベスト プラクティス ポリシー**と呼ばれるテンプレートが 50 種類用意されています。これは、ベスト プラクティス（Best Practice： 最優良の事例）という名のとおり、SQL Server を運用する上で、ベスト（と考えられる）設定かどうかをチェックできるポリシーです。これは、セキュリティやパフォーマンス、運用管理などを含めたものになっています。

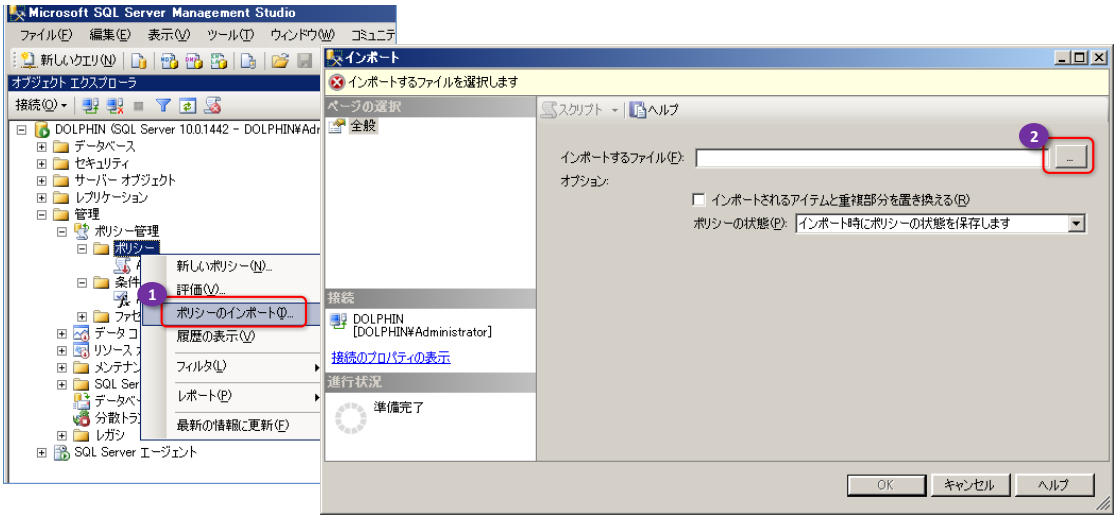
実は、前の手順で作成した自動圧縮が無効かどうかのチェックをするポリシーは、ベスト プラクティス ポリシーとして用意されています（「**Database Auto Shrink**」ポリシー）。そのほかの主なベスト プラクティスには、「**SQL Server Login Mode**」ポリシーで、セキュリティ モード（認証モード）が Windows 認証モードへ設定されているかどうかをチェックできたり、「**Backup and Data File Location**」ポリシーで、バックアップ ファイルとデータ ファイルが異なるドライブへ配置されているかどうかをチェックできたりします。

これらのポリシーは、インポート後に、環境に合わせてカスタマイズして利用することもできるので、雛形（テンプレート）として利用することができます。

設定手順

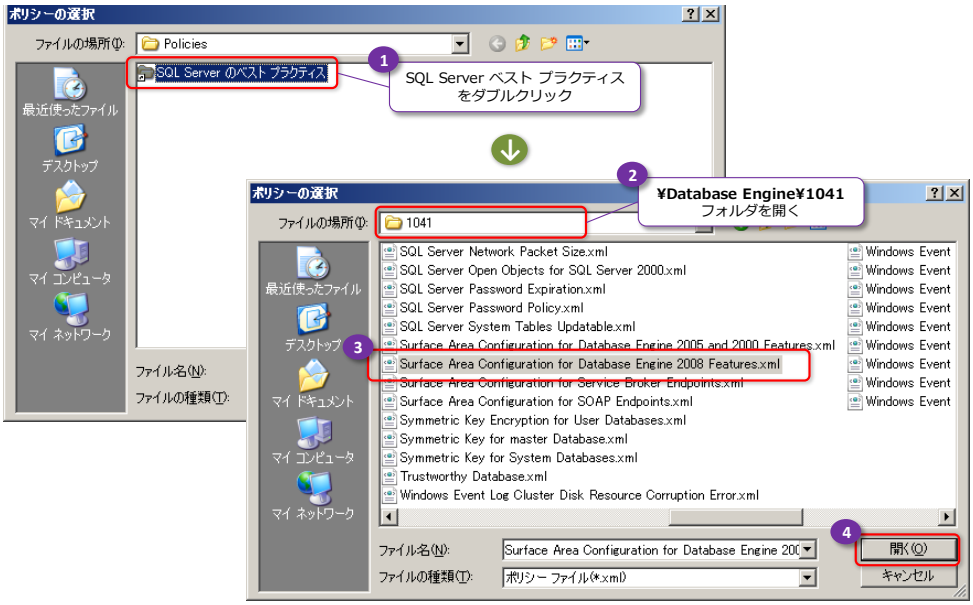
それでは、これを試してみましょう。

1. ベスト プラクティス ポリシーをインポートするには、次のように［**ポリシー**］フォルダを右クリックして、［**ポリシーのインポート**］をクリックします。



［インポート］ダイアログが表示されたら、［**インポートするファイル**］の［**...**］ボタンをクリックします。

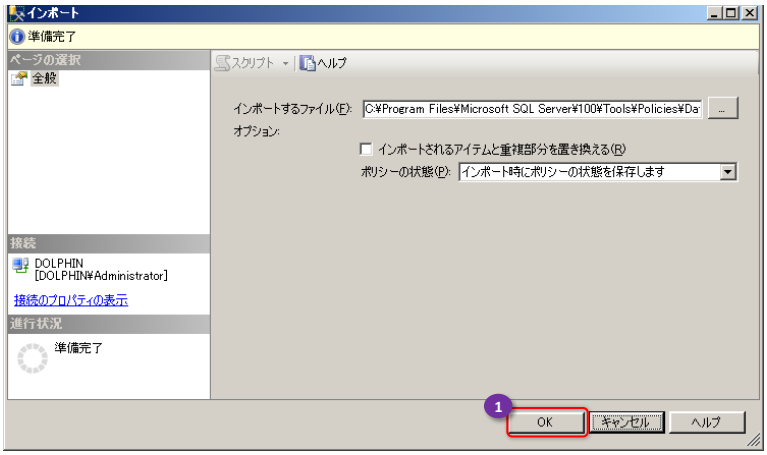
2. これにより、次の［ポリシーの選択］ダイアログが表示されるので、［SQL Server のベスト プラクティス］リンクをダブル クリックします。



続いて、「**Database Engine**」フォルダ →「**1041**」フォルダと展開します。すると、拡張子が「**.xml**」のファイルが 50 種類表示されます。これらがベスト プラクティス ポリシーです。

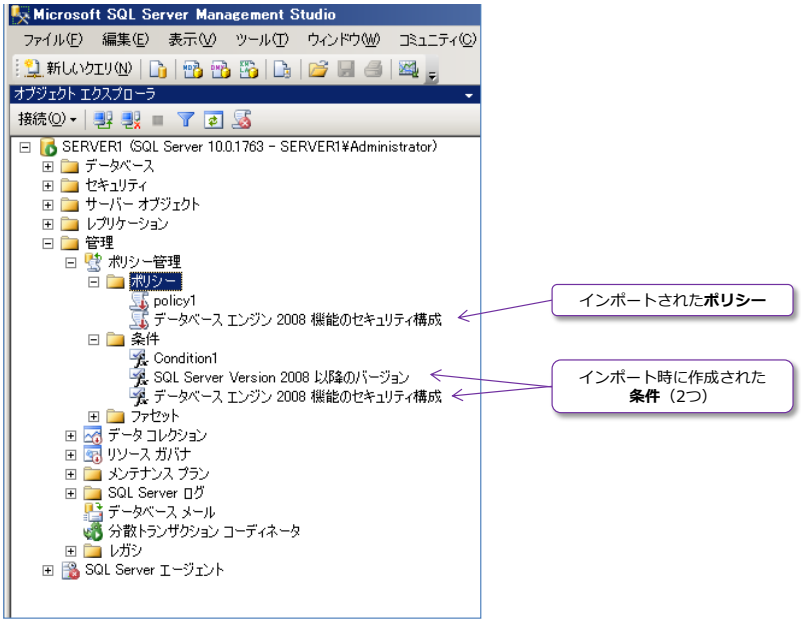
ここでは、「**Surface Area Configuration for Database Engine 2008 Features.xml**」ファイルを選択して、［**開く**］ボタンをクリックします。これは、以前のバージョンの**セキュリティ構成ツール**（Surface Area Configuration）で設定していたセキュリティ設定に関する項目をチェックするためのポリシーです。

3. ［インポート］ダイアログへ戻ったら、［OK］ボタンをクリックします。

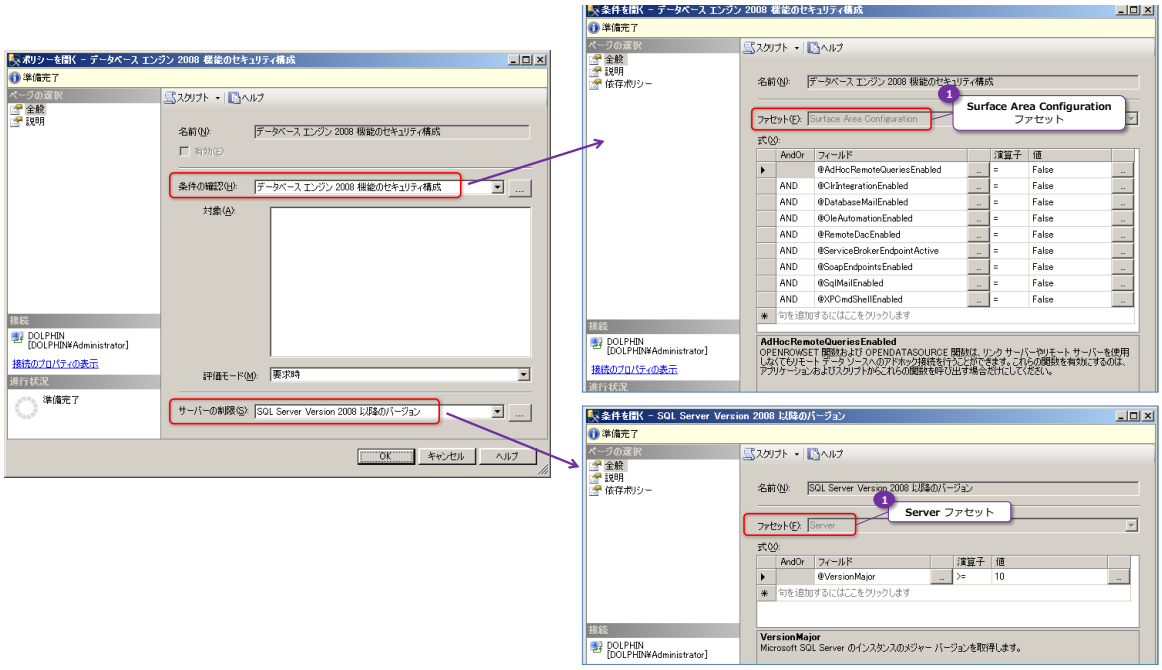


これで、ポリシーのインポートが始まります。

4. ポリシーのインポートが完了すると、次のようにポリシーとして「**データベース エンジン 2008 機能のセキュリティ構成**」、条件として「**データベース エンジン 2008 機能のセキュリティ構成**」、「**SQL Server Version 2008 以降のバージョン**」の 2 つが作成されていることを確認できます。

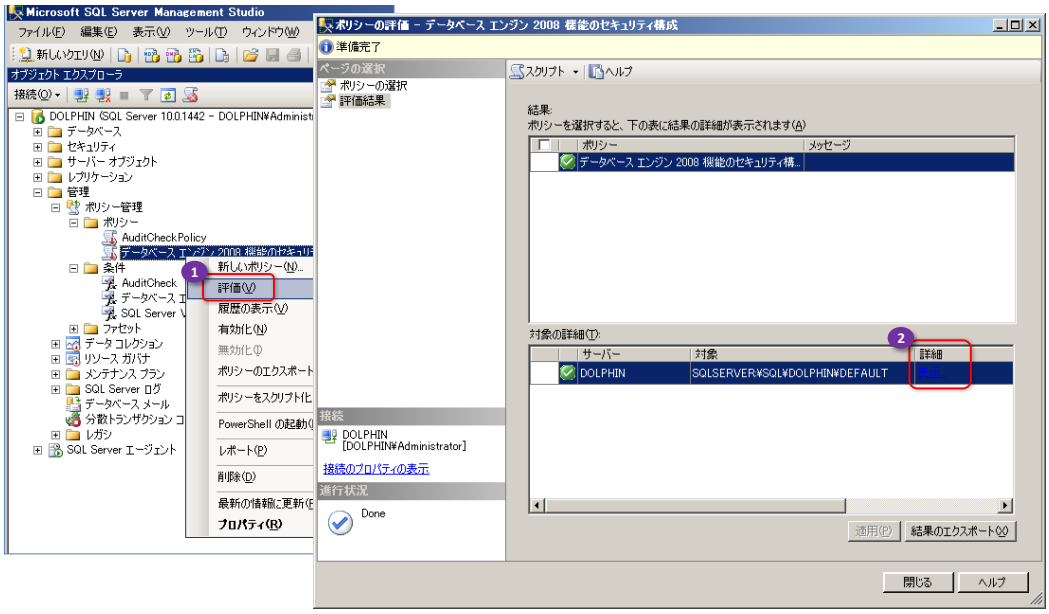


5. これらをダブル クリックして、中身を確認すると次のようになっています。

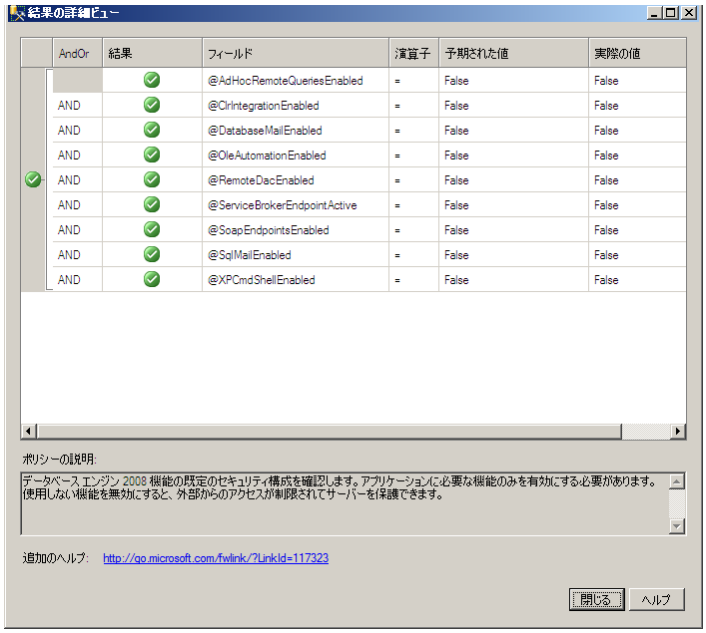


「**データベース エンジン 2008 機能のセキュリティ構成**」条件は、以前のバージョンの**セキュリティ構成ツール**（Surface Area Configuration）でデフォルト設定されていた値（**xp_cmdshell コマンド**の実行が False や、**Ad hoc 分散クエリ**の実行が False など）かどうかをチェックするための条件になっていて、「**SQL Server Version 2008 以降のバージョン**」条件は、SQL Server のバージョンが 2008 以降であるかどうかをチェックするための条件（Server ファセットの VersionMajor プロパティが 100 以上）になっています。

6. 次に、ポリシーの評価を実行してみましょう。



詳細で、［**表示**］をクリックすると、条件内の各プロパティごとの状態を確認することができます。



このように、ベスト プラクティス ポリシーを利用すると、セキュリティ設定などを簡単にチェックできるので便利です。

ポリシー ベースの管理については、本自習書シリーズの「SQL Server 2008 セキュリティ」で、さらに詳しく説明しています。

4.11 Deprecated Features オブジェクト

Deprecated Features オブジェクト

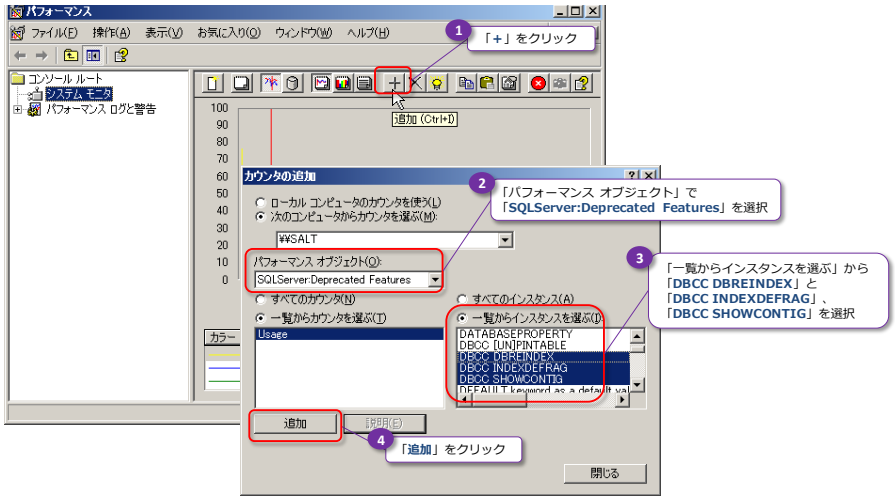
Deprecated Features オブジェクトは、システム モニタのパフォーマンス オブジェクトで、SQL Server の将来のバージョン（SQL Server 2008 の次のバージョン）以降でなくなる予定の機能が利用された回数をカウントしてくれる機能です。Deprecate は、「とがめる」や「反対する」という意味です。

それでは、これを試してみましょう。

設定手順

ここでは、将来のバージョンでなくなる予定となっている「DBCC SHOWCONTIG」と「DBCC DBREINDEX」、「DBCC INDEXDEFRAG」コマンドに対して、これらが使用された回数をカウントしてみましょう。

1. まずは、[**管理ツール**] メニューから [**パフォーマンス**] をクリックして、システム モニタを起動します。

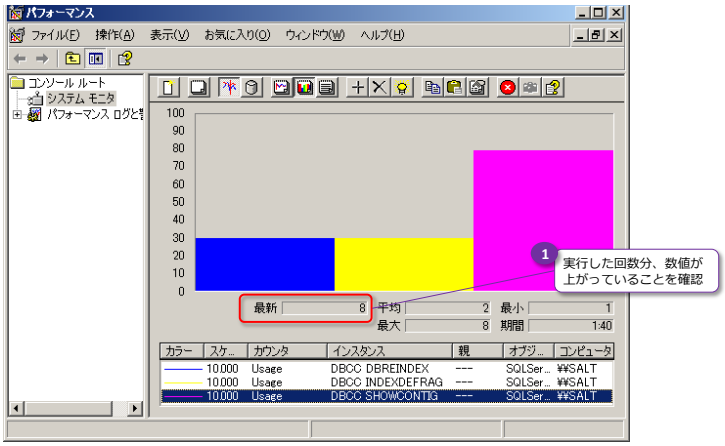


「**+**」ボタンをクリックして、[カウンタの追加] ダイアログ ボックスを表示し、[パフォーマンス オブジェクト] で、「**SQL Server:Deprecated Features**」を選択します。「**Usage**」カウンタの [**一覧からインスタンスを選ぶ**] では、将来のバージョンでなくなる機能がリストアップされるので、「**DBCC SHOWCONIG**」と「**DBCC DBREINDEX**」、「**DBCC INDEXDEFRAG**」を選択して [**追加**] ボタンをクリックします。

2. 続いて、次のように DBCC コマンドを実行します。

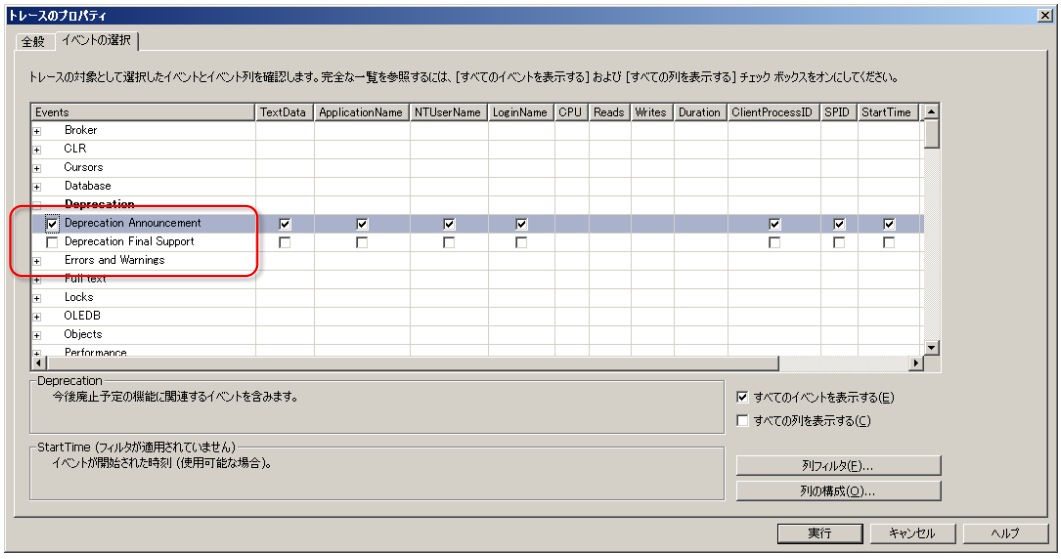
```
USE AdventureWorks
go
DBCC SHOWCONTIG ( 'Production.Product' )
DBCC DBREINDEX ( 'Production.Product' )
DBCC INDEXDEFRAG ( 'AdventureWorks', 'Production.Product')
```

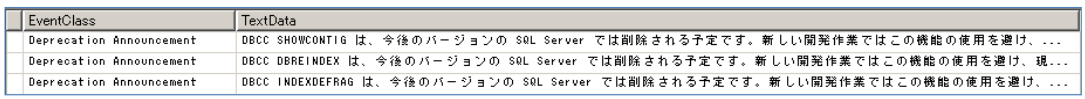
DBCC コマンドを実行後、システム モニタを確認すると、それぞれのインスタンスの数値が、上記のクエリを実行した回数分上がっていることが確認できます。



Note : プロファイラの Deprecation イベント

SQL Server 2005 からの機能であるプロファイラの Deprecation イベントも引き続き SQL Server 2008 で使用することができます。これを利用すると、プロファイラ上で将来のバージョンでなくなる機能をリストアップすることができます。





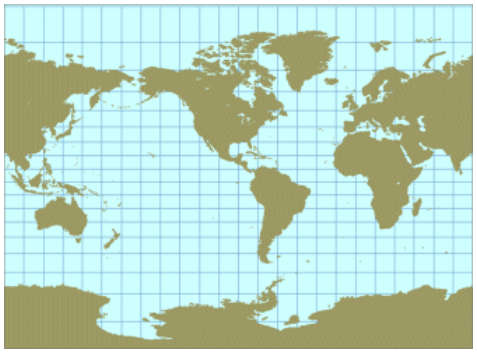
4.12 Spatial データ型による地図データのサポート

Spatial データ型（geometry、geography）

Spatial データ型は、GIS（地理情報システム）における地図データ（緯度と経度）を格納できるデータ型で、Virtual Earth と連携して、次のように地図アプリケーションとして利用することができる機能です。

Spatial データ型には、**geometry データ型**と **geography データ型**の 2 種類があり、次のような違いがあります。

平面モデル（geometry データ型）



測地モデル（geography データ型）

geometry データ型は「平面」（2 次元）として捕え、geography データ型は「球体」として捕えるという違いがあります。

それでは、これらのデータ型を試してみましょう。

geometry データ型

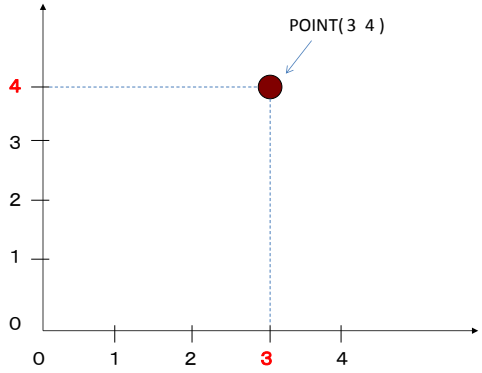
まずは、geometry データ型を利用して、単純なデータの格納の仕方などを試してみましょう。次のように記述して、データを格納、取得してみます。

```
-- データベースの作成
CREATE DATABASE GeoTestDB
go
```

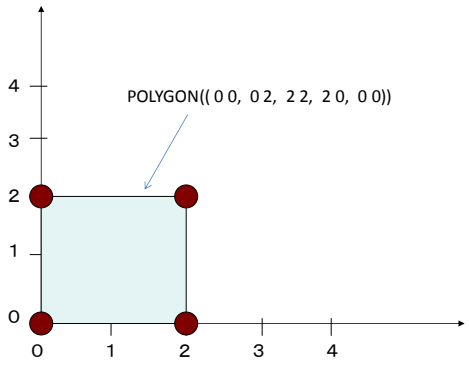
```
-- テーブルの作成。geom 列を geometry データ型へ設定
USE GeoTestDB
CREATE TABLE geomTest
(  a int IDENTITY(1,1) PRIMARY KEY
  ,geom geometry )

-- データの格納は STGeomFromText。POINT と指定することで「点」を追加
INSERT INTO geomTest
        VALUES ( geometry::STGeomFromText('POINT(3 4)', 0) )

-- POLYGON と指定することで「多角形」データを追加
INSERT INTO geomTest
        VALUES ( geometry::STGeomFromText('POLYGON((0 0, 2 0, 2 2, 0 2, 0 0))', 0) )

-- データの確認は STAsText
SELECT geom.STAsText(), * FROM geomTest
```

(列名なし)	a	geom
POINT (3 4)	1	0x00000000010C0000000000000840000000000000001040
POLYGON ((0 0, 2 0, 2 2, 0 2, 0 0))	2	0x000000000104050000000000000000000000000000000000...

POINT は点



POLYGON は多角形



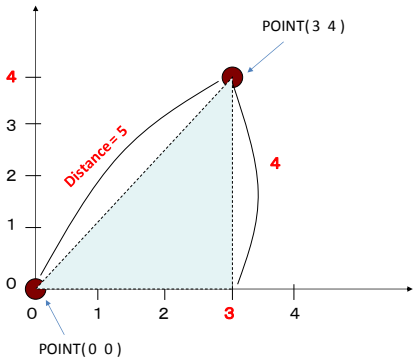
STDistance（2 点間の距離）

次に、STDistance 関数を利用して、２点間（0 0）と（3 4）の距離を取得してみましょう。

```
DECLARE @g1 geometry;
DECLARE @g2 geometry;
SET @g1 = geometry::STGeomFromText('POINT(0 0)', 0);
SET @g2 = geometry::STGeomFromText('POINT(3 4)', 0);
SELECT @g1.STDistance(@g2);
```

	(列名なし)
1	5

結果は、5 が返りますが、次のような三角形を思い浮かべると、イメージが沸くのではないでしょうか。

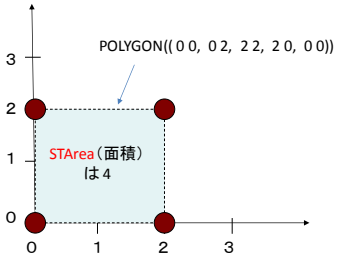


STArea（多角形の面積）

次に、STArea 関数を利用して、面積を取得してみましょう。

```
-- STArea（面積）
DECLARE @g geometry;
SET @g = geometry::STGeomFromText('POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))', 0);
SELECT @g.STArea()
```

	(列名なし)
1	4

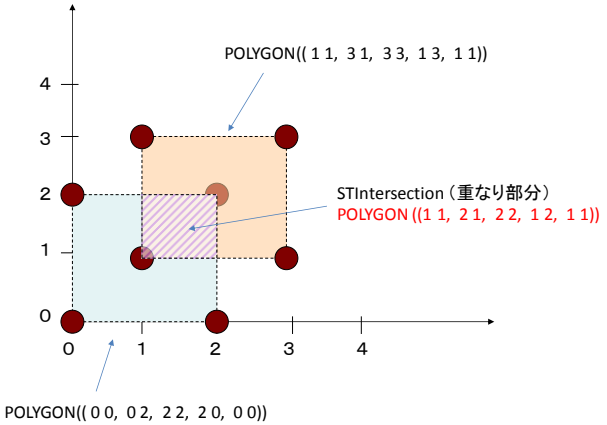


STIntersection（重なり部分の取得）

次に、STIntersection 関数を利用して、2 つの多角形の重なり部分を取得してみましょう。

```
DECLARE @g1 geometry;
DECLARE @g2 geometry;
SET @g1 = geometry::STGeomFromText('POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))', 0);
SET @g2 = geometry::STGeomFromText('POLYGON((1 1, 3 1, 3 3, 1 3, 1 1))', 0);
SELECT @g1.STIntersection(@g2).ToString();
```

(列名なし)
POLYGON ((1 1, 2 1, 2 2, 1 2, 1 1))



geography データ型と Virtual Earth 連携

次に、geography データ型を利用して、Virtual Earth と連動したアプリケーションを作成してみましょう。具体的には、次のように「つくば市」内の 4 つの郵便局を Virtual Earth 上へ表示するようなアプリケーションを作成し、郵便局の緯度と経度を geography データ型の列へ格納し、それを ASP.NET Web フォームから取得するようにします。

まずは、4 つの郵便局のデータと TX（つくばエクスプレス）のつくば駅の緯度と経度を POINT（点）として、次のように「郵便局」テーブルへ格納します（この SQL は、サンプル スクリプト内の「**Spatial_ data**」フォルダの下に「**02_geography_VirtualEarth.sql**」というファイル名で置いてあります）。

```
USE GeoTestDB
go
CREATE TABLE 郵便局
( id int IDENTITY (1,1) PRIMARY KEY,
  郵便局名   varchar(100),
  住所       varchar(200),
  geog    geography )
go

INSERT INTO 郵便局(郵便局名, 住所, geog)
        VALUES ('TXつくば駅', '茨城県つくば市吾妻２丁目１２８',
                geography::STGeomFromText('POINT(140.111561 36.082757)', 4612))

INSERT INTO 郵便局(郵便局名, 住所, geog)
        VALUES ('筑波学園郵便局', '茨城県つくば市吾妻１丁目１３－２',
                geography::STGeomFromText('POINT(140.11575 36.082818)', 4612));

INSERT INTO 郵便局(郵便局名, 住所, geog)
        VALUES ('葛城郵便局', '茨城県つくば市苅間３８８',
```

```
                geography::STGeomFromText('POINT(140.09617 36.077463)', 4612));

INSERT INTO 郵便局(郵便局名, 住所, geog)
        VALUES ('谷田部松代郵便局', '茨城県つくば市松代４丁目２００－１',
                geography::STGeomFromText('POINT(140.103989 36.063796)', 4612));

INSERT INTO 郵便局(郵便局名, 住所, geog)
        VALUES ('小野川郵便局', '茨城県つくば市館野４６４',
                geography::STGeomFromText('POINT(140.113192 36.043797)', 4612));

SELECT geog.STAsText(), * FROM 郵便局
```

(列名なし)	id	郵便局名	住所	geog
POINT (36.082757 140.111561)	1	TXつくば駅	茨城県つくば市吾妻2丁目128	0x04120(
POINT (36.082818 140.11575)	2	筑波学園郵便局	茨城県つくば市吾妻1丁目13－2	0x04120(
POINT (36.077463 140.09617)	3	葛城郵便局	茨城県つくば市苅間388	0x04120(
POINT (36.063796 140.103989)	4	谷田部松代郵便局	茨城県つくば市松代4丁目200－1	0x04120(
POINT (36.043797 140.113192)	5	小野川郵便局	茨城県つくば市館野464	0x04120(

STDistance でつくば駅からの距離を取得

次に、STDistance 関数を利用して、TX つくば駅から 4 つの郵便局までの距離を取得してみましょう。

```
DECLARE @g1 geography
SELECT @g1 = geog FROM 郵便局 WHERE id = 1          -- TX つくば駅
SELECT @g1.STDistance(geog), * FROM 郵便局 WHERE id <> 1
```

(列名なし)	id	郵便局名	住所	geog
377.361240496092	2	筑波学園郵便局	茨城県つくば市吾妻1丁目13－2	0x04120
1505.62544631322	3	葛城郵便局	茨城県つくば市苅間388	0x04120
2211.72329181969	4	谷田部松代郵便局	茨城県つくば市松代4丁目200－1	0x04120
4325.50507004141	5	小野川郵便局	茨城県つくば市館野464	0x04120

筑波学園郵便局は 377 メートル、葛城郵便局は 1.5 km 離れていることを確認できます。

次に、STDistance を WHERE 句で利用して、TX つくば駅から 2km（2,000 メートル）以内の郵便局を検索してみましょう。

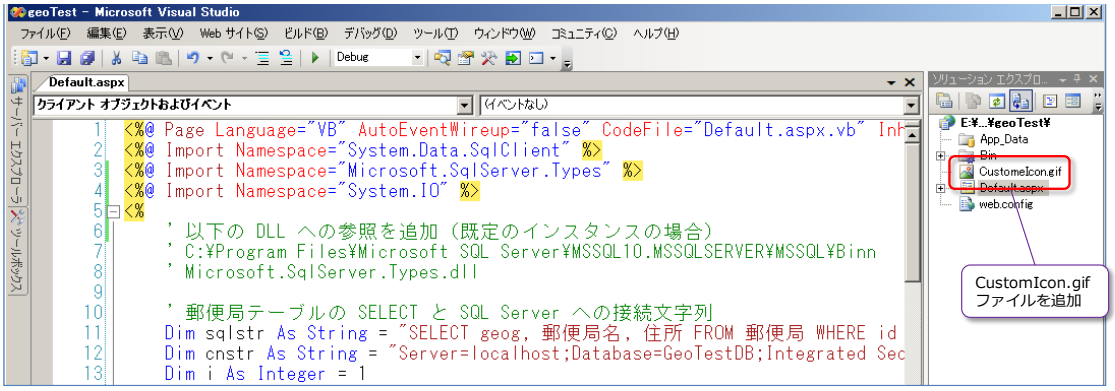
```
-- TX つくば駅から 2km 以内の郵便局
DECLARE @g1 geography
SELECT @g1 = geog FROM 郵便局 WHERE id = 1
SELECT * FROM 郵便局 WHERE @g1.STDistance(geog) < 2000 AND id <> 1
```

id	郵便局名	住所	geog
2	筑波学園郵便局	茨城県つくば市吾妻1丁目13－2	0x041200000
3	葛城郵便局	茨城県つくば市苅間388	0x041200000

筑波学園郵便局と葛城郵便局のみがヒットしていることを確認できます。

Virtual Earth との連携（ASP.NET）

次に、ASP.NET 2.0 と ADO.NET 2.0（Visual Studio 2005 / 2008）を利用して、geography データ型のデータを取得し、それを Virtual Earth 上へ表示してみましょう。Virtual Earth への表示部分は、クライアント サイド スクリプト（JavaScript）を利用して、**Virtual Earth API** を利用します。また、郵便局のアイコンを表示するために、サンプル フォルダ内の「**CustomeIcon.gif**」ファイルを利用するので、プロジェクト内へ追加しておいてください（Visual Studio のソリューション エクスプローラで「既存の項目の追加」から追加できます）。



geography データ型のデータを取得する部分では、ADO.NET を利用しますが、geography データ型のデータは、**SqlGeography** オブジェクトへ格納して操作することができます。この SqlGeography オブジェクトを利用するには、「**Microsoft.SqlServer.Types.dll**」ファイルへの参照を事前に追加しておく必要があります。このファイルは、次のフォルダへ格納されています。

C:¥Program Files¥Microsoft SQL Server¥100¥SDK¥Assemblies

参照の追加が完了したら、次のコード例のように「Microsoft.SqlServer.Types」名前空間をインポートすることで SqlGeography オブジェクトを利用できるようになります。このオブジェクトの「**Lat**」プロパティでは、geography データ型へ格納した緯度（Latitude）を取得することができ、「**Long**」プロパティでは、経度（Longitude）を取得できるようになります。

コード例（サンプル スクリプト内の「**Spatial_data**」フォルダの下の「**geoTest**」にある、「**geoTest.sln**」ファイルをダブル クリックすると確認できます）

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="Microsoft.SqlServer.Types" %>
<%@ Import Namespace="System.IO" %>
<%
  ' 郵便局テーブルの SELECT と SQL Server への接続文字列
  Dim sqlstr As String = "SELECT geog, 郵便局名, 住所 FROM 郵便局 WHERE id <> 1"
  Dim cnstr As String = "Server=localhost;Database=GeoTestDB;Integrated Security=SSPI"
  Dim i As Integer = 1
  Dim scriptStr As New StringBuilder("")
  Dim shapeStr As String = ""

  ' SQL Server へ接続して、郵便局テーブルを取得
  Using cn As New SqlConnection(cnstr)
      cn.Open()
      Using cmd As New SqlCommand(sqlstr, cn)
```

```
        Using dr As SqlDataReader = cmd.ExecuteReader()
            '' 郵便局の数だけ繰り返し処理
            While dr.Read
                ''geography データ型のデータを SqlGeography オブジェクトへ格納
                Dim geog As New SqlGeography
                geog = dr("geog")

                Dim lat As String, lon As String

                '' 緯度。Latitude
                lat = geog.Lat.ToString()
                '' 経度。Longitude
                lon = geog.Long.ToString()

                '' クライアント サイド スクリプト (JavaScript) を StringBuilder で文字列として組み立て。
                '' シェイプへプッシュピンを追加 (Virtual Earth API の VELatLong オブジェクトと)
                '' VEShape オブジェクトを利用。SetCustomIcon メソッドでカスタム画像を指定
                shapeStr = "shape" & i
                scriptStr.Append("var latLon = new VELatLong(" & lat & ", " & lon & ");" & vbCrLf)
                scriptStr.Append("var " & shapeStr & " = new VEShape(VEShapeType.Pushpin, latLon);" & vbCrLf)
                scriptStr.Append(shapeStr & ".SetTitle('" & dr("郵便局名") & "'); " & vbCrLf)
                scriptStr.Append(shapeStr & ".SetDescription('" & dr("住所") & "'); " & vbCrLf)
                scriptStr.Append(shapeStr & ".SetCustomIcon('customeIcon.gif'); " & vbCrLf)
                scriptStr.Append("shapeLayer1.AddShape(" & shapeStr & ");" & vbCrLf)
                i = i + 1
            End While
        End Using
    End Using
End Using
%>
<html>
<head>
    '' Virtual Earth API を利用するための記述
    <script src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6&mkt=ja-jp"></script>
    <script>
      '' クライアント サイド スクリプト (JavaScript)
      var map = null;
      function GetMap()
      {
        ''Virtual Earth API の VEMap オブジェクトの作成
        map = new VEMap('myMap');
        // TX つくば駅を中心として、ズームサイズを 12 に指定
        map.LoadMap(new VELatLong(36.082757, 140.111561), 12, "r", false);
        // シェイプ レイヤーの追加
        shapeLayer1 = new VEShapeLayer();
        // レイヤーへカスタム プッシュピンの追加 (StringBuilder で組み立てた郵便局データ)
        <%=scriptStr %>
        // シェイプレイヤーを地図に追加
        map.AddShapeLayer(shapeLayer1);
      }
    </script>
</head>
<body onload="GetMap();">
    <h1>Virtual Earth 連携テスト</h1>
    <div id='myMap' style="position:relative; width:640px; height:480px;"></div>
</body>
</html>
```

このコードは、単純に ４つの郵便局を表示するだけですが、前述の STDistance 関数などと組み合わせて利用することで、より実践的なアプリケーションを作成できるようになります。

4.13 FileStream データ型

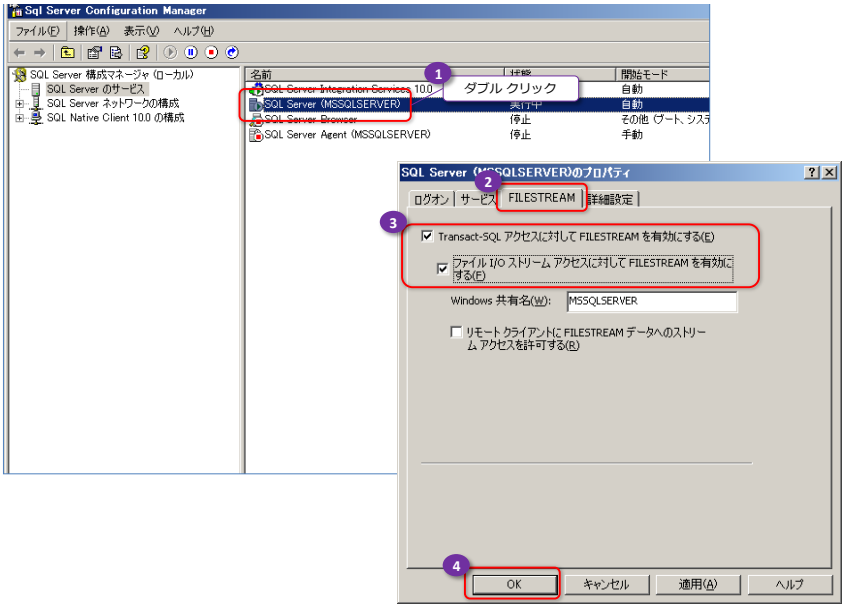
FileStream データ型

FileStream データ型は、Windows のファイルを直接 SQL Server へ格納できるデータ型で、SQL Server 2008 からの新機能です。このデータ型を利用することで、従来の BLOB 型（image データ型や varbinary(max) データ型）へファイル データを格納するよりも、パフォーマンスの良いアプリケーションを作成できるようになります。

Let's Try

それでは、これを試してみましょう。

1. まずは、FileStream の機能を有効化する必要があります。SQL Server 構成マネージャを起動し、SQL Server サービスをダブル クリックして、[SQL Server のプロパティ] ダイアログを開きます。



[**FILESTREAM**] タブを開き、[**Transact-SQL アクセスに対して FILESTEAM を有効にする**] と [**ファイル I/O ストリーム アクセスに対して FILESTREAM を有効にする**] をチェックし、[OK] ボタンをクリックします。

2. 続いて、次のように **sp_configure** システム ストアド プロシージャを実行して、FileStream 機能を有効化します。

```
EXEC sp_configure filestream_access_level, 2
RECONFIGURE
```

3. 次に、FileStream データを格納するためのファイル グループを指定したデータベースを「**fsTestDB**」という名前で作成します（作成先のドライブは、環境にあわせて適宜変更して

ください)。

```
CREATE DATABASE fsTestDB
ON
PRIMARY
         ( NAME = fsTestDB1_mdf
        , FILENAME = 'D:¥fsTestDB1.mdf'),
FILEGROUP FileStreamGroup1 CONTAINS FILESTREAM
        ( NAME = fsTestDB_fs
        , FILENAME = 'D:¥fsTestDB_FileSream')
LOG ON
        ( NAME = fsTestDB_log
        , FILENAME = 'D:¥fsTestDB_Log.ldf')
```

CONTAINS FILESTREAM を指定することで、そのファイル グループを FileStream データの格納用として利用できるようになります。

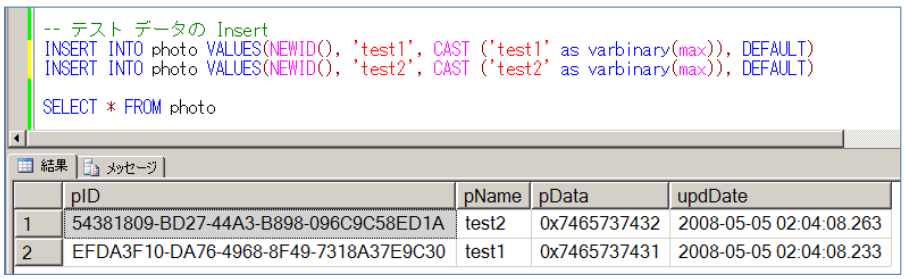
4. 次に、FileStream データを格納するためのテーブルを「**photo**」という名前で作成します。

```
USE fsTestDB
CREATE TABLE photo
(
        pID     uniqueidentifier ROWGUIDCOL PRIMARY KEY,
        pName   varchar(200),
        pData   varbinary(MAX) FILESTREAM NULL,
        updDate datetime DEFAULT GETDATE()
)
```

pData 列を **varbinary(MAX) FILESTREAM** と指定することで、この列へ FileStream データを格納できるようになります（後述の手順で画像データを格納します）。また、FILESTREAM キーワードを指定した場合は、**uniqueidentifier** データ型で **ROWGUIDCOL** キーワードを指定した列が必要になるので、ここでは pID 列をそれにしています。そのほかの pName 列は、ファイル名を格納するための列とし、updDate 列は、データ更新日時を格納するための列とします。

5. 続いて、データを 2 件 INSERT してみましょう。

```
INSERT INTO photo VALUES(NEWID(), 'test1', CAST ('test1' AS varbinary(max)), DEFAULT)
INSERT INTO photo VALUES(NEWID(), 'test2', CAST ('test2' AS varbinary(max)), DEFAULT)

SELECT * FROM photo
```



```
-- テスト データの Insert
INSERT INTO photo VALUES(NEWID(), 'test1', CAST ('test1' as varbinary(max)), DEFAULT)
INSERT INTO photo VALUES(NEWID(), 'test2', CAST ('test2' as varbinary(max)), DEFAULT)

SELECT * FROM photo
```

結果　メッセージ

	pID	pName	pData	updDate
1	54381809-BD27-44A3-B898-096C9C58ED1A	test2	0x7465737432	2008-05-05 02:04:08.263
2	EFDA3F10-DA76-4968-8F49-7318A37E9C30	test1	0x7465737431	2008-05-05 02:04:08.233

6. 次に、PathName と GET_FILESTREAM_TRANSACTION_CONTEXT 関数を利用して、内部的なパスと、トランザクション コンテキストを取得します。

```
BEGIN TRAN
     SELECT pData.PathName(), GET_FILESTREAM_TRANSACTION_CONTEXT() FROM photo
ROLLBACK TRAN
```

```
-- PathName と GET_FILESTREAM_TRANSACTION_CONTEXT() の取得
BEGIN TRAN
    SELECT pData.PathName(), GET_FILESTREAM_TRANSACTION_CONTEXT() FROM photo
ROLLBACK TRAN
```

結果 | メッセージ

	(列名なし)	(列名なし)
1	¥¥POWER¥MSSQLSERVER¥v1¥fsTestDB¥dbo¥photo¥pData¥54381...	0x918EBCF7BE5AEA4B8082C993EE1BCE48
2	¥¥POWER¥MSSQLSERVER¥v1¥fsTestDB¥dbo¥photo¥pData¥EFDA...	0x918EBCF7BE5AEA4B8082C993EE1BCE48

この 2 つの結果は、アプリケーションを作成する際に利用することになります。

7. 結果を確認後、データをすべて削除しておきます。

```
TRUNCATE TABLE photo
```

OpenSqlFilestream API

続いて、Visual Studio 2008 の C# 3.0 を利用して、FileStream データ型へデータを格納するアプリケーションを作成してみましょう。次のように Button1 をクリックすると、ファイルの選択ダイアログが表示されて、選択した画像を photo テーブルへ格納されるようにし、格納したデータを ListView と PictureBox で表示するようにします。

アプリケーションの作成には、**OpenSqlFilestream API** を利用しますが、オンライン ブックの以下の場所へ詳細が記載されています。

データベース エンジン > 開発 > FILESTREAM ストレージの設計と実装
> Win32 を使用した FILESTREAM データの管理 > OpenSqlFilestream API

オンライン ブックより、以下のコードをコピーすると、この API を利用できるようになります。

```
const UInt32 DESIRED_ACCESS_READ = 0x00000000;
const UInt32 DESIRED_ACCESS_WRITE = 0x00000001;
const UInt32 DESIRED_ACCESS_READWRITE = 0x00000002;

const UInt32 SQL_FILESTREAM_OPEN_NO_FLAGS = 0x00000000;
const UInt32 SQL_FILESTREAM_OPEN_FLAG_ASYNC = 0x00000001;
const UInt32 SQL_FILESTREAM_OPEN_FLAG_NO_BUFFERING = 0x00000002;
const UInt32 SQL_FILESTREAM_OPEN_FLAG_NO_WRITE_THROUGH = 0x00000004;
const UInt32 SQL_FILESTREAM_OPEN_FLAG_SEQUENTIAL_SCAN = 0x00000008;
const UInt32 SQL_FILESTREAM_OPEN_FLAG_RANDOM_ACCESS = 0x00000010;

[DllImport("sqlncli10.dll", SetLastError = true, CharSet = CharSet.Unicode)]
static extern SafeFileHandle OpenSqlFilestream(
            string FilestreamPath,
            UInt32 DesiredAccess,
            UInt32 OpenOptions,
            byte[] FilestreamTransactionContext,
            UInt32 FilestreamTransactionContextLength,
            Int64 AllocationSize);

[DllImport("kernel32.dll", SetLastError = true)]
static extern UInt32 GetLastError();
```

コード例

完成版は、サンプル スクリプトの「**FileStreamTest**」フォルダの「**fsTest1.sln**」ソリューション ファイルにあります。

データの Insert 時のコードでポイントとなるのは、次の部分です。

まずは、FileStream データ型の pData 列へ「**0**」を指定した INSERT ステートメントを発行して、データを追加します。

```
private Guid getGuidCol(string p)
{
    Guid g = Guid.NewGuid();

    using (SqlConnection cn = new SqlConnection(cnstr))
    {
        cn.Open();
        using (SqlCommand cmd = new SqlCommand(
            "INSERT INTO photo(pID, pName, pData) " +
            "VALUES(@pID, @pName, 0)", cn))
            {
            cmd.Parameters.AddWithValue("pID", g);
            cmd.Parameters.AddWithValue("pName", p);
            cmd.ExecuteNonQuery();
            }
        cn.Close();
    }
    return (g);
}
```

このときに追加した GUID を WHERE 句の絞り込み条件へ指定して、SELECT ステートメント

を発行し、**PathName** と **GET_FILESTREAM_TRANSACTION_CONTEXT** 関数の結果を取得し、それを OpenSqlFilestream API の引数へ与えます。

```
using (SqlConnection cn = new SqlConnection(cnstr))
{
    // Connection のオープンとトランザクションの開始
    cn.Open();
    SqlTransaction tran = cn.BeginTransaction();

    using (SqlCommand cmd = new SqlCommand(
            "SELECT pData.PathName()," +
            "GET_FILESTREAM_TRANSACTION_CONTEXT() " +
            "FROM photo WHERE pID=@pID", cn, tran))
    {
    cmd.Parameters.AddWithValue("pID", newID);

        using (SqlDataReader rdr = cmd.ExecuteReader())
        {
            if (rdr.Read())
            {
                string path = (string)rdr[0];   // PathName()
                byte[] ctx = (byte[])rdr[1];    // GET_FILESTREAM_TRANSACTION_CONTEXT()
                uint length = (uint)ctx.Length;
                long allocSize = 0;

                // OpenSqlFileStream でハンドルを取得
                // DESIRED_ACCESS_WRITE で書き込み指定
                SafeFileHandle handle =
                    OpenSqlFilestream(path
                    ,DESIRED_ACCESS_WRITE
                    ,SQL_FILESTREAM_OPEN_NO_FLAGS
                    ,ctx
                    ,length
                    ,allocSize);

                // System.IO.FileStream の Write メソッドでバイト配列（画像データ）を書込み
                using (FileStream fs = new FileStream(handle, FileAccess.Write))
                {
                    fs.Write(imageFile, 0, imageFile.Length);
                    fs.Close();
                    handle.Close();
                }
            }
            rdr.Close();
        }
    }
    // コミットとクローズ。これで画像の登録（photo テーブルへの Insert）が完了
    tran.Commit();
    cn.Close();
```

OpenSqlFilestream API の第 2 引数では、**DESIRED_ACCESS_WRITE** 定数を指定することで、書き込み（FileStream データ型への INSERT）ができるようになります。

実際の書き込み（FileStream データ型への INSERT）は、**System.IO.FileStream** の **Write** メソッドで行っています。FileStream クラスのコンストラクタの第 1 引数へ、OpenSqlFilestream API で取得したファイル ハンドルを指定しているところがポイントです。

データの読み取り

データの読み取り時にポイントとなるのは、次の部分です。書き込みのときとほとんど同じで、

GUID を WHERE 句の絞り込み条件へ指定して、SELECT ステートメントを発行し、**PathName** と **GET_FILESTREAM_TRANSACTION_CONTEXT** 関数の結果を取得し、それを OpenSqlFilestream API の引数とへ与えています。違いは、OpenSqlFilestream API の第 2 引数で、**DESIRED_ACCESS_READ** 定数を指定している点です。

```
            using (SqlConnection cn = new SqlConnection(cnstr))
            {
                // Connection のオープンとトランザクションの開始
                cn.Open();
                SqlTransaction tran = cn.BeginTransaction();

                // Guid で絞り込み、
                // PathName() と GET_FILESTREAM_TRANSACTION_CONTEXT() の取得
                using (SqlCommand cmd = new SqlCommand(
                    "SELECT pData.PathName(),"  +
                    "GET_FILESTREAM_TRANSACTION_CONTEXT(), pName " +
                    "FROM photo WHERE pID=@pID", cn, tran))
                {
                    cmd.Parameters.AddWithValue("pID", g);

                    using (SqlDataReader rdr = cmd.ExecuteReader())
                    {
                        if (rdr.Read())
                        {
                            string path = (string)rdr[0];       // PathName()
                            byte[] ctx = (byte[])rdr[1];        // GET_FILESTREAM_TRANSACTION_CONTEXT()
                            uint length = (uint)ctx.Length;
                            long allocSize = 0;

                            pName = (string)rdr[2];

                            // OpenSqlFilestream でハンドルを取得
                            // DESIRED_ACCESS_READ で読み取り指定
                            SafeFileHandle handle =
                                OpenSqlFilestream(path
                                , DESIRED_ACCESS_READ
                                , SQL_FILESTREAM_OPEN_NO_FLAGS
                                , ctx
                                , length
                                , allocSize);

                            // System.IO.FileStream で画像データの読み込み
                            using (FileStream fs = new FileStream(handle, FileAccess.Read))
                            {
                                Image img = Image.FromStream(fs);
                                return (img);
                                fs.Close();
                                handle.Close();
                            }
                        }
                        else
                        {
                            Image img = null;
                            return (img);
                        }
                        rdr.Close();
                    }
                }
                tran.Commit();
                cn.Close();
            }
```

実際の FileStream データ型からのファイルの読み取りは、**System.IO.FileStream** クラスを

利用しています。FileStream クラスのコンストラクタの第 1 引数へ、OpenSqlFilestream API で取得したファイル ハンドルを指定し、第 2 引数へ **FileAccess.Read** を指定しているところがポイントです。

4.14 パーティション ウィザード

パーティション ウィザード

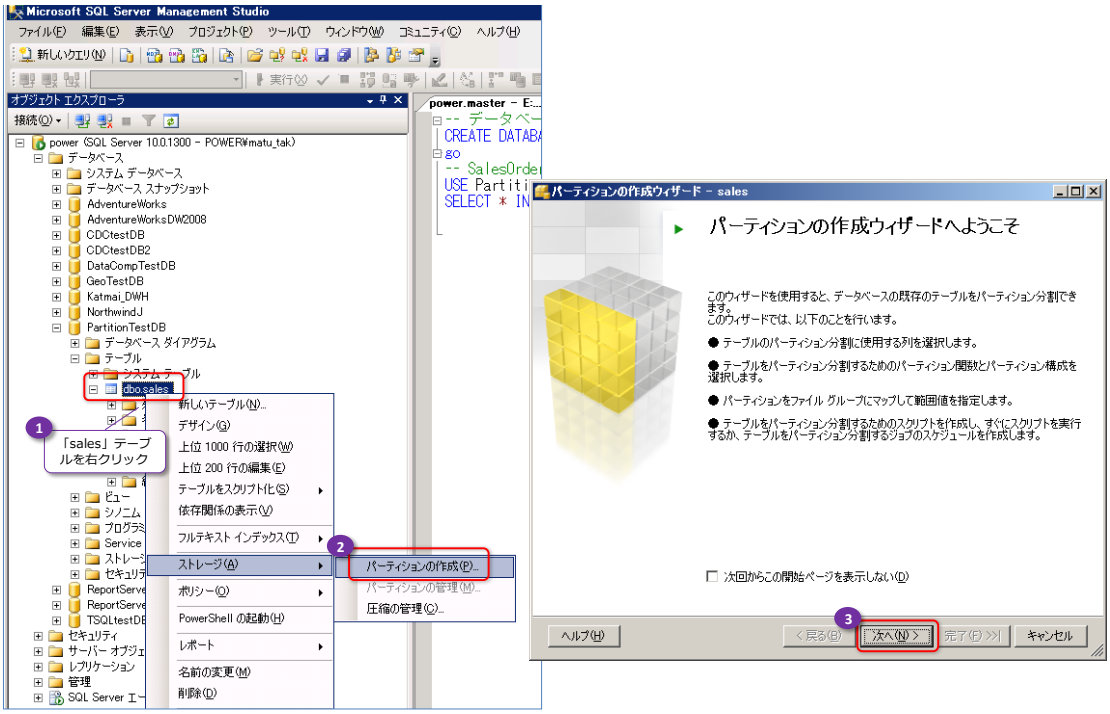
パーティション ウィザードは、その名のとおり、データ パーティション（レンジ パーティション）の設定をウィザードで簡単に設定できる機能です。

それでは、これを試してみましょう。ここでは、**AdventureWorks** 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。

1. まずは、次のように「**PartitionTestDB**」データベースを作成し、AdventureWorks 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。

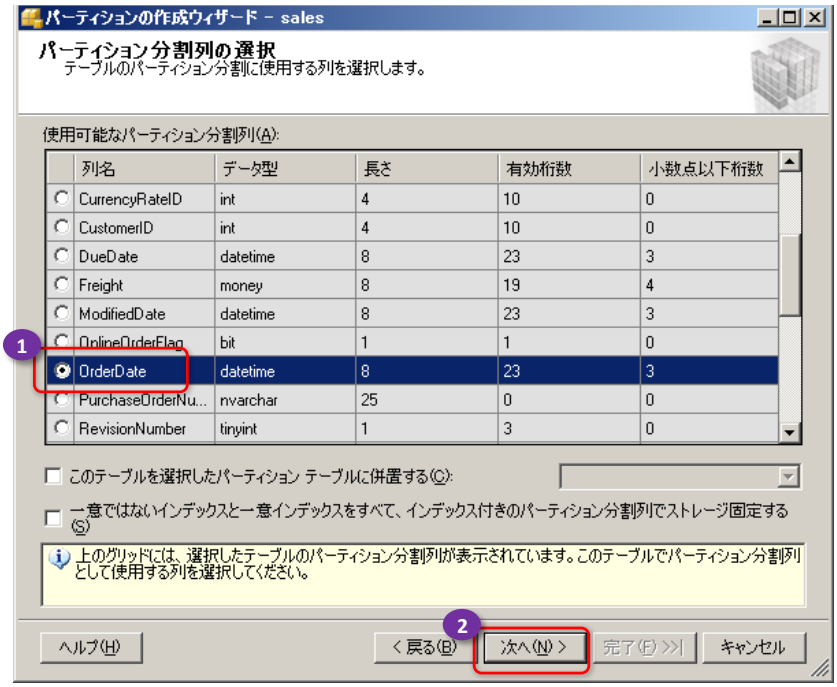
```
-- データベースの作成
CREATE DATABASE PartitionTestDB
go
-- SalesOrderDetail テーブルをもとにsales テーブルを作成
USE PartitionTestDB
SELECT * INTO sales FROM AdventureWorks.Sales.SalesOrderHeader
```

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。



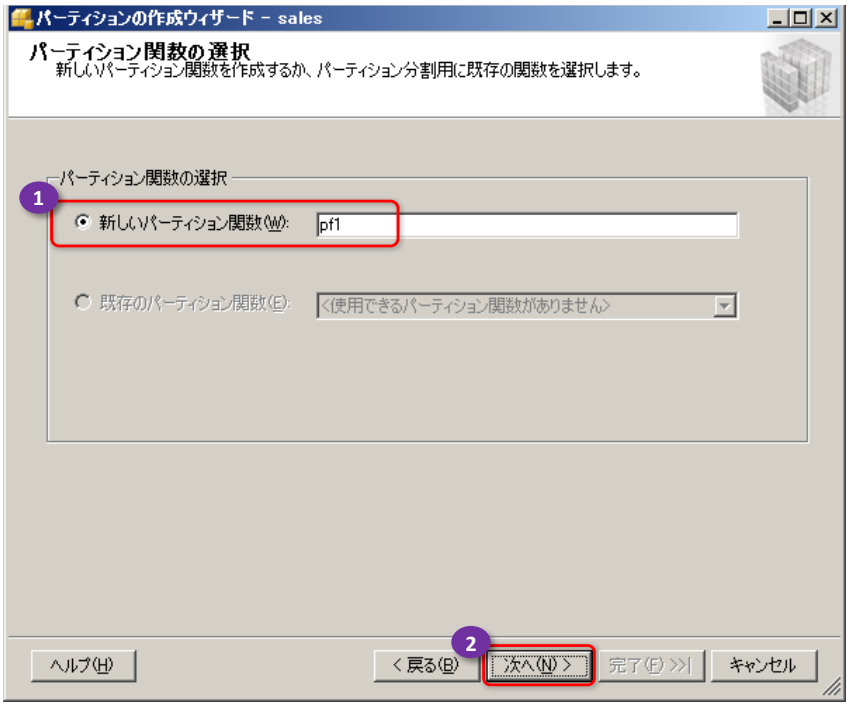
すると、「パーティションの作成ウィザード」が起動するので、[次へ] ボタンをクリックします。

3. 次の［パーティション分割列の選択］画面では、パーティションを区切る基準となる列を選択します。

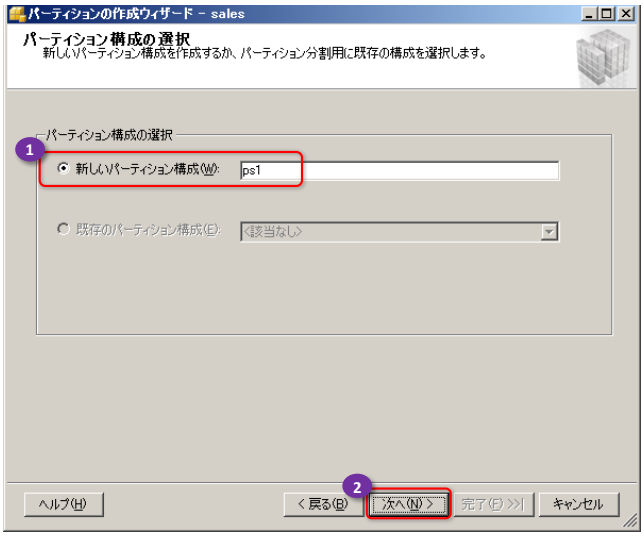


ここでは、「**OrderDate**」（受注日）を選択して、［次へ］ボタンをクリックします。

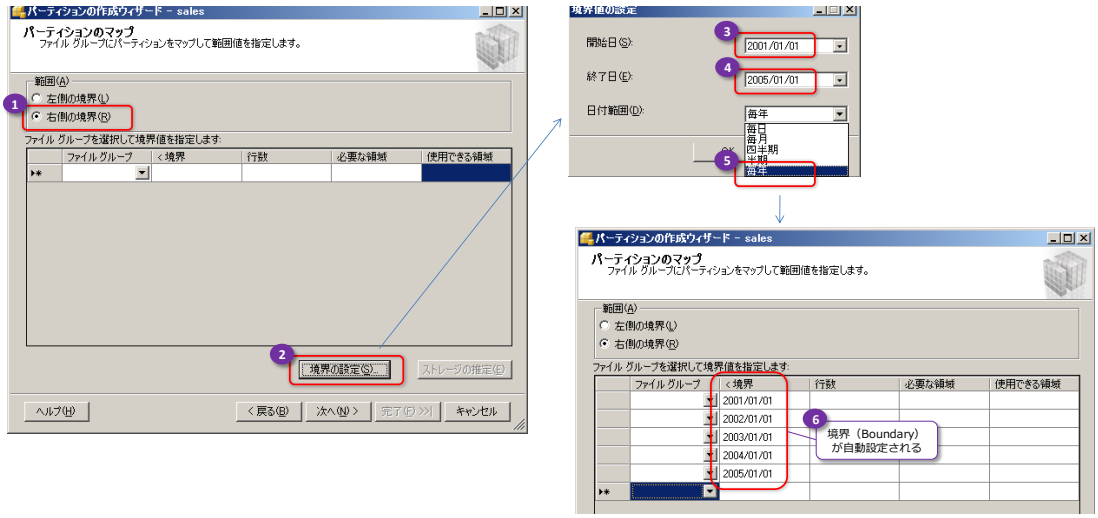
4. 次の［パーティション関数の選択］画面では、［**新しいパーティション関数**］を選択して、任意の名前（**pf1** など）を入力し、［次へ］ボタンをクリックします。



5. 次の［パーティション構成の選択］画面では、［**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。

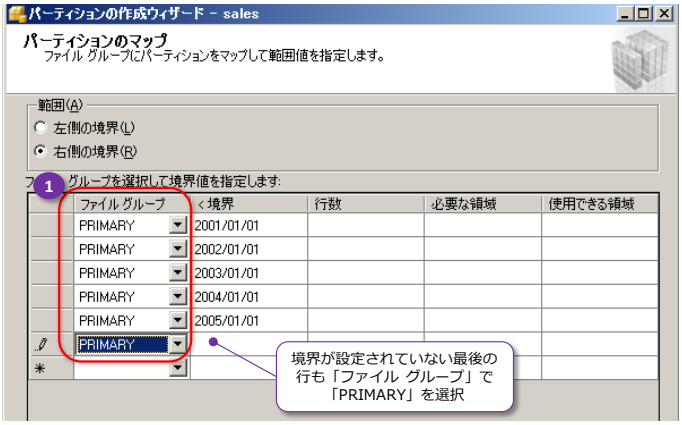


6. 次の［パーティションのマップ］画面では、［**右側の境界**］を選択して、［**境界の設定**］ボタンをクリックします。

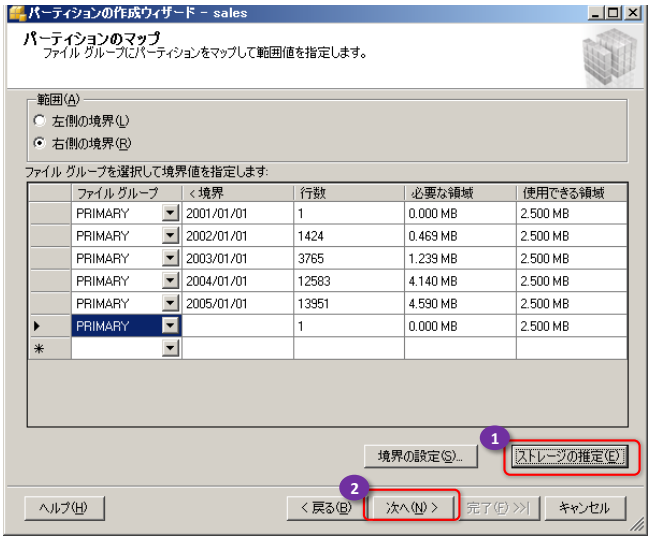


これにより、［境界値の設定］ダイアログが表示されるので、［開始日］を「**2001/01/01**」へ、［終了日］を「**2005/01/01**」へ、［日付範囲］を「**毎年**」へ変更し、［OK］ボタンをクリックします。これにより、2001 年から 2005 年までの 1 年ごとのパーティション範囲を作成することができます。

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。

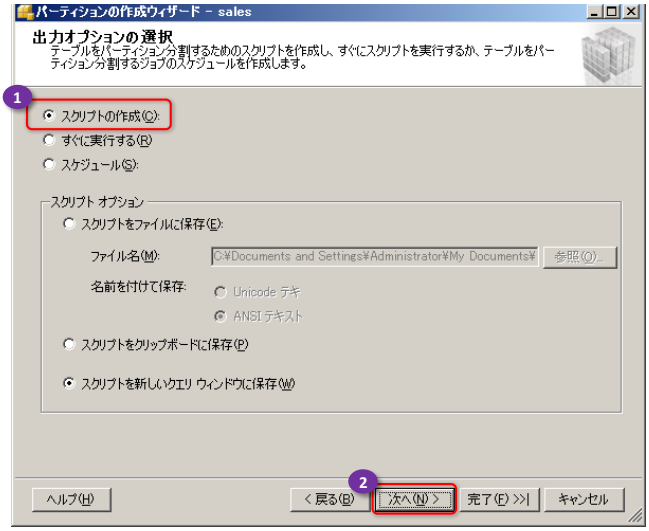


続いて、［**ストレージの推定**］ボタンをクリックして、各パーティションに含まれる行数や領域の見積もりを表示します。

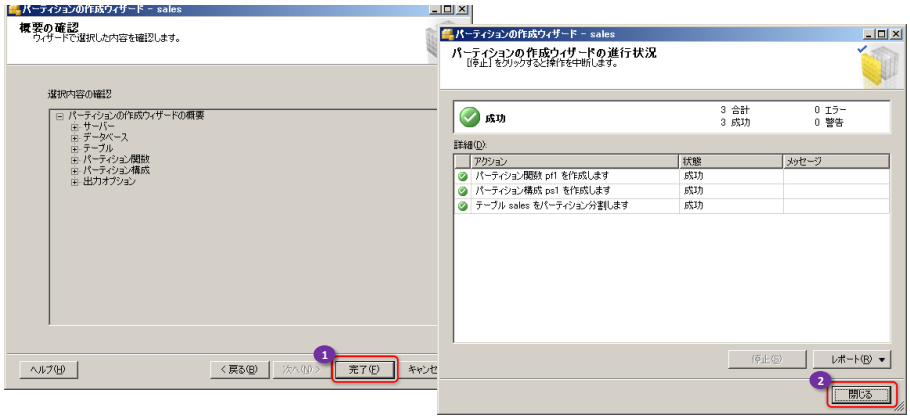


確認後、［次へ］ボタンをクリックして、次へ進みます。

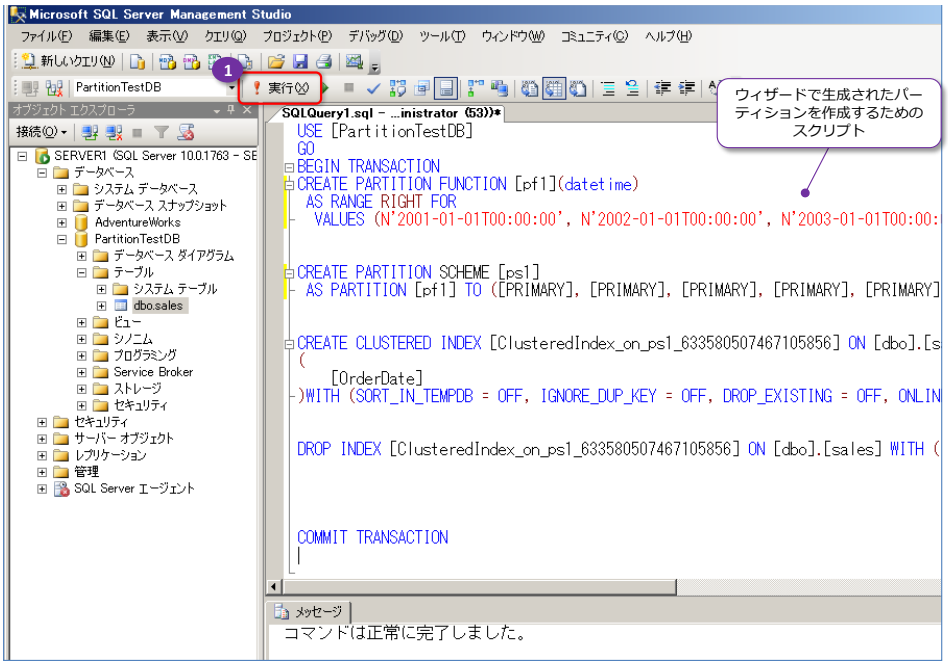
7. 次の［出力オプションの選択］画面では、［**スクリプトの作成**］を選択して、［次へ］ボタンをクリックします。



8. 最後の［概要の確認］画面では、［完了］ボタンをクリックします。

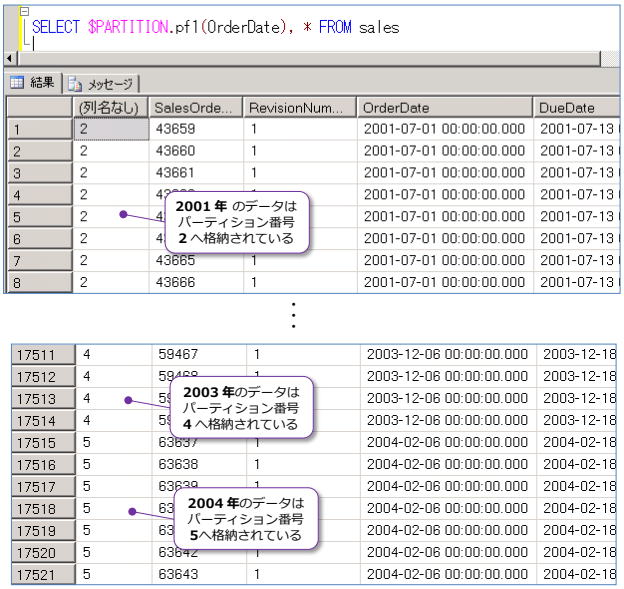


9. ウィザードが完了すると、パーティションを作成するためのスクリプトがクエリ エディタへ自動生成されるので、ツール バーの［**！実行**］ボタンをクリックして、ステートメントを実行します。



10. 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
```



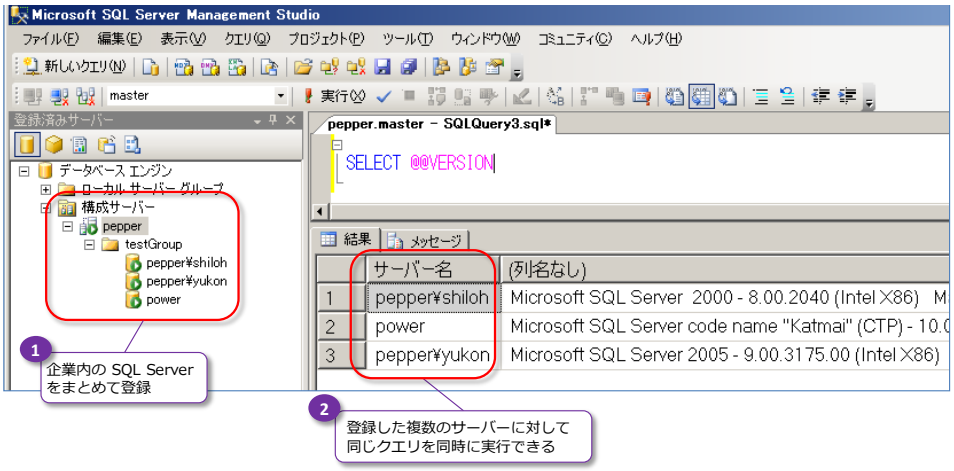
	(列名なし)	SalesOrde...	RevisionNum...	OrderDate	DueDate
1	2	43659	1	2001-07-01 00:00:00.000	2001-07-13
2	2	43660	1	2001-07-01 00:00:00.000	2001-07-13
3	2	43661	1	2001-07-01 00:00:00.000	2001-07-13
4	2	43662	1	2001-07-01 00:00:00.000	2001-07-13
5	2	43663	1	2001-07-01 00:00:00.000	2001-07-13
6	2	43664	1	2001-07-01 00:00:00.000	2001-07-13
7	2	43665	1	2001-07-01 00:00:00.000	2001-07-13
8	2	43666	1	2001-07-01 00:00:00.000	2001-07-13
⋮					
17511	4	59467	1	2003-12-06 00:00:00.000	2003-12-18
17512	4	59468	1	2003-12-06 00:00:00.000	2003-12-18
17513	4	59469	1	2003-12-06 00:00:00.000	2003-12-18
17514	4	59470	1	2003-12-06 00:00:00.000	2003-12-18
17515	5	63637	1	2004-02-06 00:00:00.000	2004-02-18
17516	5	63638	1	2004-02-06 00:00:00.000	2004-02-18
17517	5	63639	1	2004-02-06 00:00:00.000	2004-02-18
17518	5	63640	1	2004-02-06 00:00:00.000	2004-02-18
17519	5	63641	1	2004-02-06 00:00:00.000	2004-02-18
17520	5	63642	1	2004-02-06 00:00:00.000	2004-02-18
17521	5	63643	1	2004-02-06 00:00:00.000	2004-02-18

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。

4.15 マルチ サーバー クエリ

マルチ サーバー クエリ（Multi Server Query）

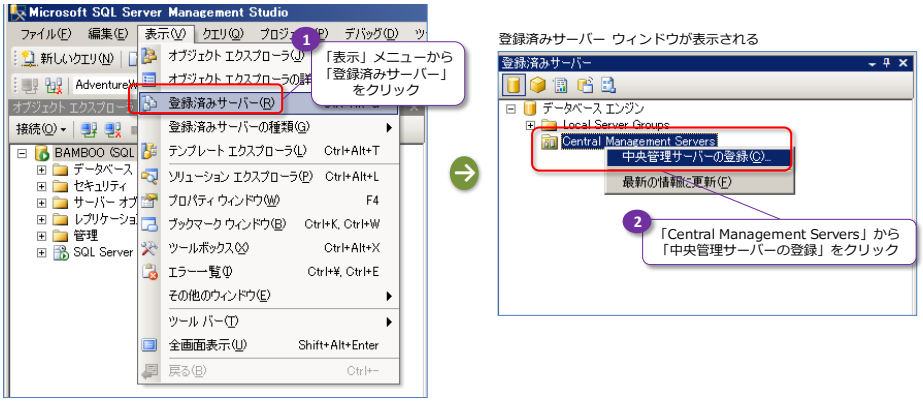
マルチ サーバー クエリ（Multi Server Query）機能は、その名のとおり、複数のサーバーに対して同時に同じクエリを実行できる機能です。マルチ サーバー クエリの対象にできるサーバーは、SQL Server 2008 だけでなく、SQL Server 2005 と SQL Server 2000 も含めることができるので、企業内の SQL Server をまとめて管理する目的として利用することができます。



設定手順

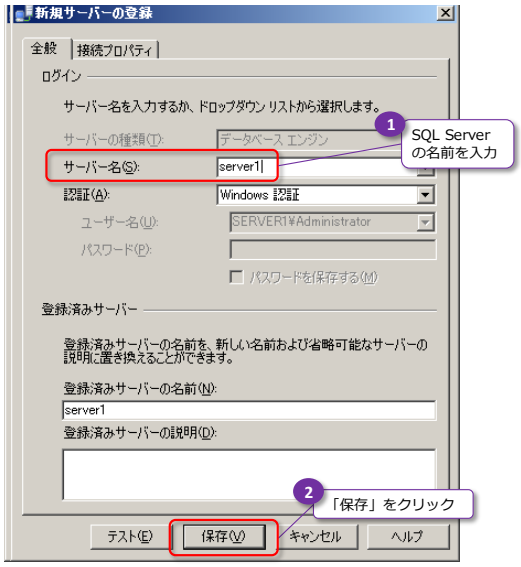
それでは、マルチ サーバー クエリを試してみましょう（この手順を試すには、SQL Server のインスタンスが 3 つ以上必要になるのですが、SQL Server 2005 や SQL Server 2000 を対象とすることもできるので、それらを利用して試してみてください）。

1. マルチ サーバー クエリを利用するには、まず［**表示**］メニューから［**登録済みサーバー**］をクリックして、［登録済みサーバー］ウィンドウを開きます。

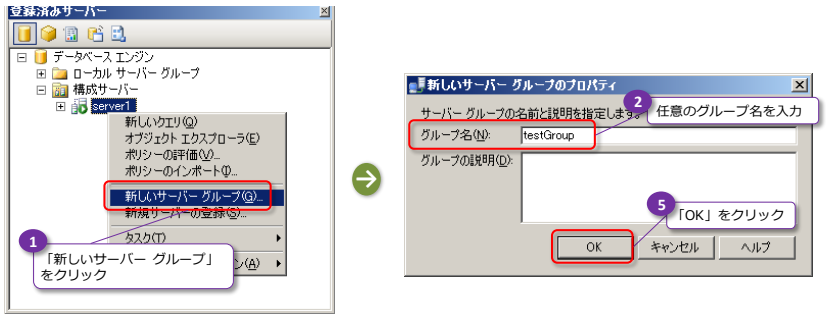


［登録済みサーバー］ウィンドウでは、［**Central Management Servers**］フォルダを右クリックして［**中央管理サーバーの登録**］をクリックします。

2. すると、［新規サーバーの登録］ダイアログが表示されるので、管理サーバーとして設定したい SQL Server の名前を入力して、［保存］ボタンをクリックします。

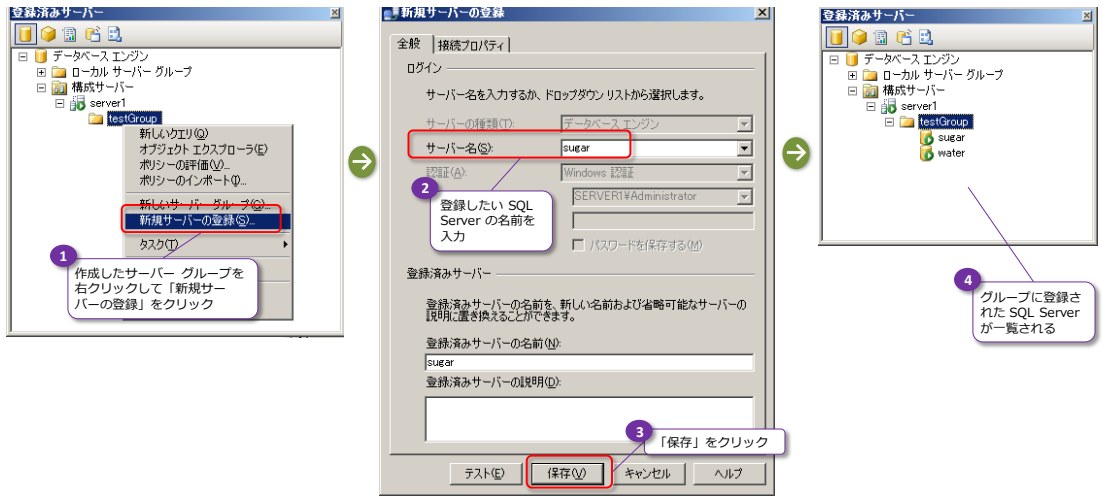


3. 次に、登録した構成サーバーを右クリックして［**新しいサーバー グループ**］をクリックします。



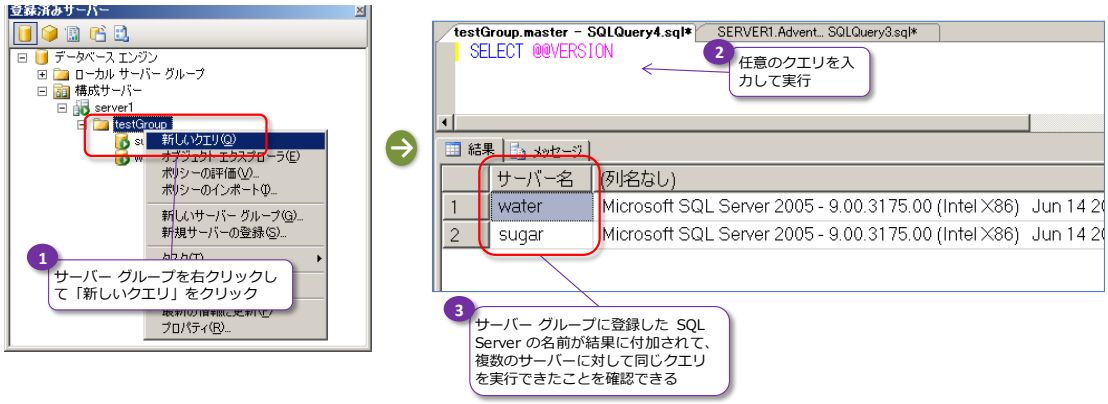
すると、［新しいサーバー グループのプロパティ］ダイアログが表示されるので、［グループ名］に任意のグループ名（testGroup など）を入力して、［OK］ボタンをクリックします。グループは対象となるサーバーをグループ化するために作成するもので、たとえば東京や大阪といった地域の名前を付けたグループを作成して、地域ごとの SQL Server をグループ化するといった使い方ができます。

4. 次に、グループ内へ対象となるサーバーを登録するために、次のようにグループ名を右クリックして［**新規サーバーの登録**］をクリックします。



［新規サーバーの登録］ダイアログでは、［サーバー名］へ対象にしたい SQL Server の名前を入力して、［保存］ボタンをクリックします。

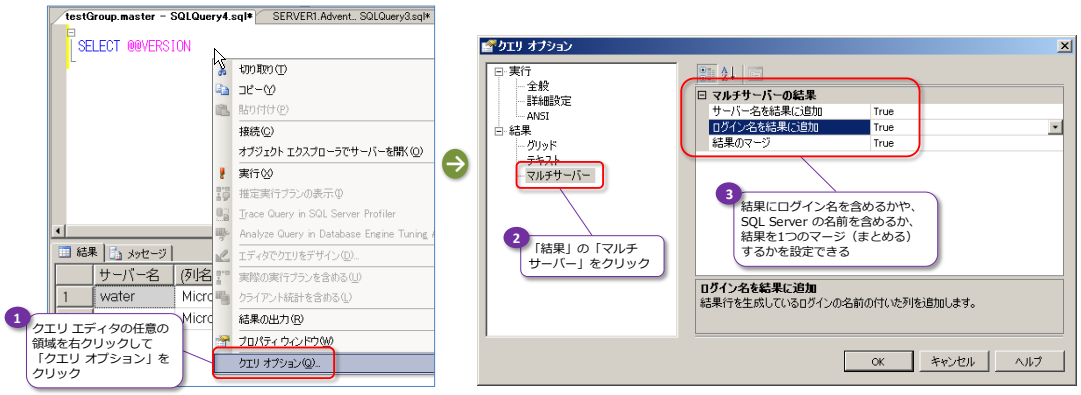
5. 次に、グループ名を右クリックして［**新しいクエリ**］をクリックします。



すると、新しいクエリ エディタが表示されるので、任意のクエリを入力して実行し、実行結果を確認します。グループへ登録した SQL Server の名前が結果に付加されて、複数のサーバーに対して同じクエリを実行できたことを確認できます。

Note：クエリ オプションで結果の表示方法の変更

マルチ サーバー クエリの結果の表示方法は、クエリ オプションで変更することができます。変更するには、次のようにクエリ エディタの任意の領域を右クリックして［クエリ オプション］をクリックします。



おわりに

最後まで試された皆さま、いかがでしたでしょうか。SQL Server 2008 には多くのパワフルな機能が追加されていることを確認できたのではないでしょうか。いろいろな機能がある中で、筆者のイチ押しの機能は、「パフォーマンス データ コレクション & データ コレクション」と「リソース ガバナ」、「データ プロファイル タスク」です。いずれもパフォーマンス チューニング時に大変役立つ機能ですので、ぜひ皆さんも利用してみていただければと思います。

そのほかの機能

SQL Server 2008 には、まだまだたくさんの機能が提供されています。その主なものは次のとおりです。

- **DWH（データ ウェアハウス）系のパフォーマンスの向上**
  - スター ジョインのパフォーマンス向上
  - パーティションのパラレル処理のパフォーマンス向上（1 つのパーティションを複数スレッドが処理可能に）
  - パーティション単位のロックが可能に
  - インデックス付きビューがパーティションへ割り当て可能に
  - 周辺テクノロジーのパフォーマンス向上（Integration Services の Data Flow パイプラインのパフォーマンス向上、Lookup パフォーマンスの向上、Analysis Services と Reporting Services のエンジン改良など）

  SQL Server 2008 は、データ ウェアハウス関連のパフォーマンス向上に目覚ましいものがあります。上述の機能だけでなく、この自習書内でご紹介した、「データ圧縮」や「バックアップ圧縮」、「リソース ガバナ」もデータ ウェアハウス系のシステムで大変役立ちます。

  現在、SQL Server を利用したテラ バイト（TB）サイズのデータ ウェアハウスを構築している企業は多数あり、筆者もそういった企業のコンサルティング（物理設計や論理設計、BI システム設計など）を行った経験があります。SQL Server 2008 のパフォーマンス向上によって、さらにデータ ウェアハウスとしての利用が増えるのではないでしょうか。

- **Lock Escalation オプション**（ALTER TABLE ステートメントを利用して、テーブル単位でロック エスカレーションの禁止が可能に）
- **実行プランの固定**（プラン キャッシュをもとに実行プランの固定が可能に。従来のバージョンでは XML 実行プランをもとに固定が可能）
- **Hot Add CPU**（SQL Server を止めることなく CPU の追加が可能に）
- **Off-Box キー管理**（ハードウェア ベースの鍵管理機構を利用可能に）
- **Sparse カラム**（NULL 値の多いデータ列を効率良く格納し、ストレージ サイズを小さ

くできる機能）

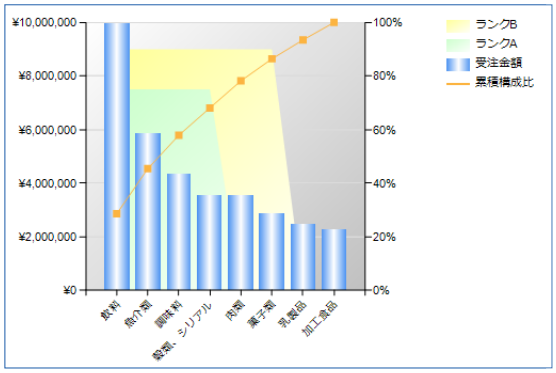
- **Filtered インデックス**（テーブル内の一部の行へインデックスを作成できる機能）
- **フルテキスト インデックス統合**（FullText Search サービスが Database Engine サービスに統合された）
- **Peer To Peer トランザクション レプリケーション**（トランザクション レプリケーションで双方向更新が可能に。グラフィカルな設定インターフェースも搭載）
- **データベース ミラーリング機能の強化**（ログ圧縮によるパフォーマンス向上と破損ページの自動修復機能の追加など）

Reporting Services の新機能

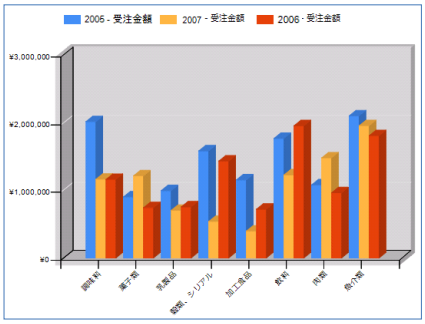
SQL Server 2008 の Reporting Services（標準搭載されるレポート サーバー機能）は、大幅に強化され、次のような実践的なレポートを簡単に作成できるようになりました。

売上目標に対する達成率をゲージで表示

ABC 分析クラフ（パレート図。2Y軸対応）



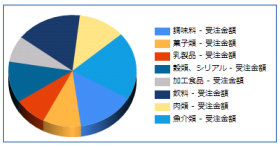
3D 棒クラフ



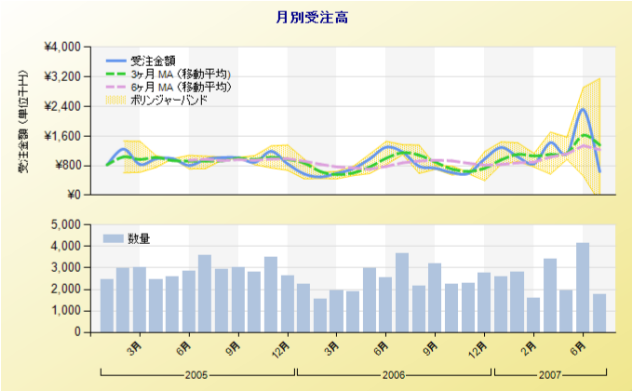
クロス集計レポート

				2005			
	Total	累積構成比率	ABC 評価	Q1	Q2	Q3	Q4
商品区分名	¥29,821,530			¥2,901,900	¥2,778,000	¥3,000,200	¥2,919,600
魚介類	¥5,863,800	19.7%	A	¥485,200	¥506,600	¥671,200	¥437,800
飲料	¥4,949,750	36.3%	A	¥355,900	¥513,600	¥411,900	¥486,000
調味料	¥4,340,500	50.8%	A	¥235,300	¥596,000	¥613,900	¥570,200
穀類、シリアル	¥3,556,380	62.7%	A	¥407,600	¥282,100	¥444,800	¥448,300
肉類	¥3,522,800	74.6%	A	¥545,800	¥247,200	¥163,700	¥123,500
菓子類	¥2,862,200	84.2%	B	¥330,400	¥174,400	¥181,800	¥217,400
乳製品	¥2,453,800	92.4%	C	¥297,000	¥193,800	¥227,900	¥276,600
加工食品	¥2,272,300	100.0%	C	¥244,700	¥264,300	¥285,000	¥359,800

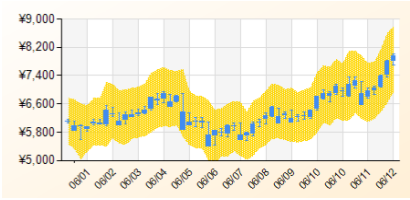
円グラフ



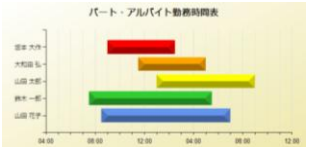
移動平均線、ボリンジャーバンド



箱ひげ、範囲グラフ



ガント チャート



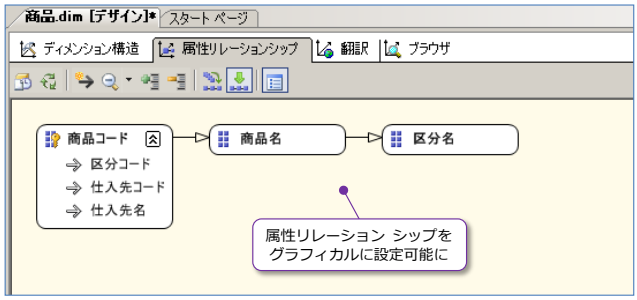
Reporting Services によるレポート作成方法については、本自習書シリーズの「**Reporting Services 入門**」で説明していますので、ぜひご覧いただければと思います。

自習書シリーズ「Reporting Services 入門」のダウンロードはこちらから
http://www.microsoft.com/japan/sqlserver/2008/self-learning/default.mspx

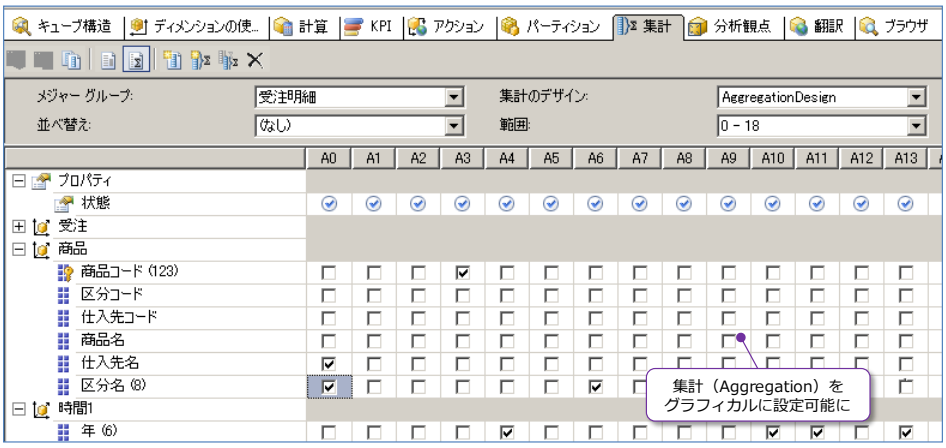
Analysis Services の新機能

SQL Server 2008 の Analysis Services（標準搭載される分析サーバー機能）では、属性リレーションシップ デザイナや集計デザイナの搭載、パフォーマンスの向上、DMV（動的管理ビュー）による監視機能などが追加されています。

属性リレーションシップ デザイナ



集計デザイナ

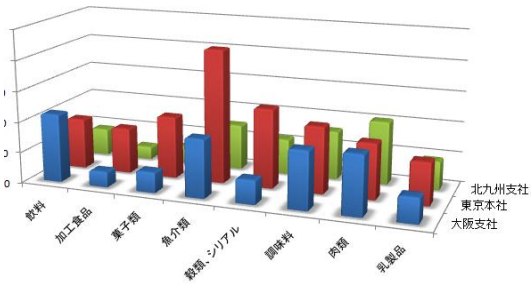


Analysis Services は、"**データ分析**" のためのサーバー機能で、企業の売上分析やクロス集計レポ

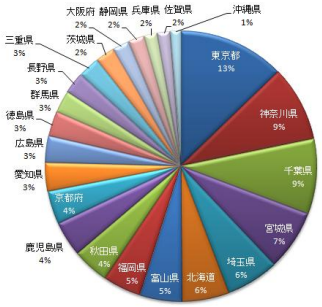
ート、ABC 分析（パレート図）、財務諸表分析、経営ダッシュボード、データ マイニングなどが簡単に行えるようになります（次のようなレポートやグラフを簡単に作成できます）。

商品分類ごとの商品売上トップ 5

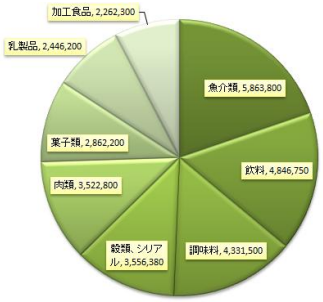
行ラベル	大阪支社	東京本社	北九州支社	総計
飲料	1,118,500	830,950	445,100	2,394,550
バードワイン	172,500	203,750	147,500	523,750
ナイトワイン	365,000	150,000		515,000
ビリビリビール	224,000	128,800	117,600	470,400
オタル白ラベル	243,000	147,000	66,000	456,000
清涼スカッシュ	114,000	201,400	114,000	429,400
加工食品	252,900	744,700	210,100	1,207,700
冷凍コーンクリームコロッケ	19,600	218,400	47,600	285,600
冷凍クリームコロッケ	22,400	196,000	22,400	240,800
冷凍やきおにぎり	57,000	108,000	70,500	235,500
朝日かまぼこ	59,400	127,800	36,000	223,200
特選焼きちくわ	94,500	94,500	33,600	222,600
菓子類	333,800	1,016,800	293,600	1,644,200
バニラクリームアイス	106,400	442,400	14,000	562,800
チョコクリームアイス	162,400	232,400	42,000	436,800
小倉あんぱん	40,000	95,000	95,000	230,000
チーズあんぱん	25,000	115,000	70,000	210,000
抹茶バー		132,000	72,600	204,600



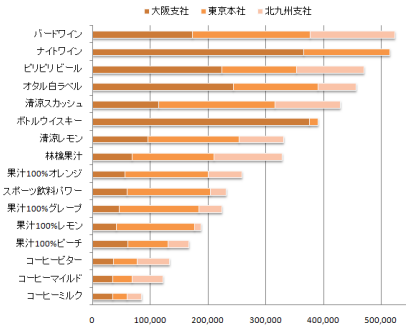
都道府県ごとの商品売上の割合



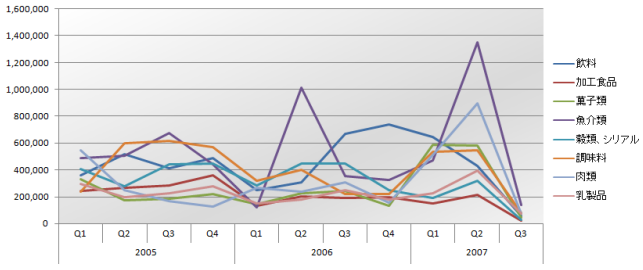
商品区分ごとの商品売上金額



商品ごとの売上金額（飲料区分）



区分ごとの商品売上の推移（四半期単位）



クロス集計、ABC 分析レポート

受注金額	2005	2006	前年比	総計	構成比	累積構成比	ABC評価
総計	1,767,400	1,954,600	187,200	3,722,000			
ナイトワイン	250,000	265,000	15,000	515,000	13.8%	13.8%	
清涼スカッシュ	167,200	262,200	95,000	429,400	11.5%	25.4%	
ボトルウイスキー	135,000	255,000	120,000	390,000	10.5%	35.9%	
ビリビリビール	176,400	162,400	-14,000	338,800	9.1%	45.0%	A
清涼レモン	163,400	167,200	3,800	330,600	8.9%	53.8%	
林檎果汁	112,000	216,000	104,000	328,000	8.8%	62.6%	
オタル白ラベル	144,000	174,000	30,000	318,000	8.5%	71.2%	
バードワイン	162,500	137,500	-25,000	300,000	8.1%	79.3%	
スポーツ飲料パワー	86,400	144,000	57,600	230,400	6.2%	85.4%	B
果汁100% オレンジ	114,000	24,000	-90,000	138,000	3.7%	89.2%	
果汁100% グレープ	98,000	20,000	-78,000	118,000	3.2%	92.3%	
果汁100% レモン	82,000	16,000	-66,000	98,000	2.6%	95.0%	
果汁100% ピーチ	48,000	22,000	-26,000	70,000	1.9%	96.8%	C
コーヒーマイルド	19,000	32,300	13,300	51,300	1.4%	98.2%	
コーヒービター	9,500	32,300	22,800	41,800	1.1%	99.3%	
コーヒーミルク	0	24,700	24,700	24,700	0.7%	100.0%	

Analysis Services の利用する手順については、本自習書シリーズの「**Analysis Services 入門**」で説明していますので、ぜひご覧いただければと思います。

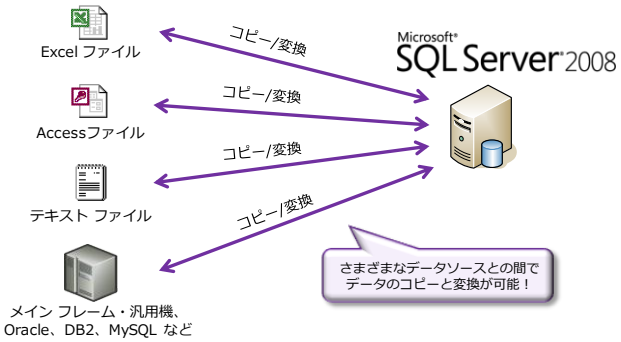
自習書シリーズ「Analysis Services 入門」のダウンロードはこちらから
http://www.microsoft.com/japan/sqlserver/2008/self-learning/default.mspx

Integration Services の新機能

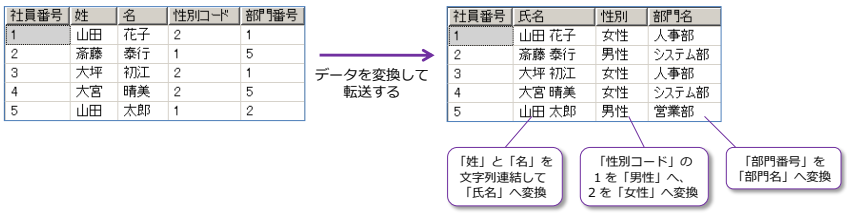
SQL Server 2008 の Integration Services（標準搭載されるデータ転送・変換機能）では、Data Flow パイプライン処理のパフォーマンス向上や Lookup 性能の向上、C# によるスクリプティング、すべての .NET アセンブリが参照可能になるなどの機能が追加されています。

Integration Services は、データの「コピー」や「変換」などが行える "**データ転送ツール**" で、SQL Server 2000 以前のバージョンでは、DTS（Data Transformation Services：データ変換サービス）と呼ばれていたものです。Integration Services を利用すると、SQL Server 同士でのデータ転送はもちろん、Oracle や DB2、Microsoft Office Access、そのほかの ODBC 対応のデータベース、Microsoft Office Excel ファイル、可変長のテキスト ファイル（カンマ区切り、タブ区切り）、固定長のテキスト ファイルなど、さまざまなデータソースから SQL Server へデータを取り込んだり、それとは逆に SQL Server からデータを書き出したりすることができます。

Integration Services は、データ転送・変換ツール



Integration Services のデータを変換（データを加工しながら転送）できる機能は、非常に便利で、データ ウェアハウス（DWH：Data Warehouse）を構築する際には、欠かせないツールとなります（次のようなデータ変換が行えます）。

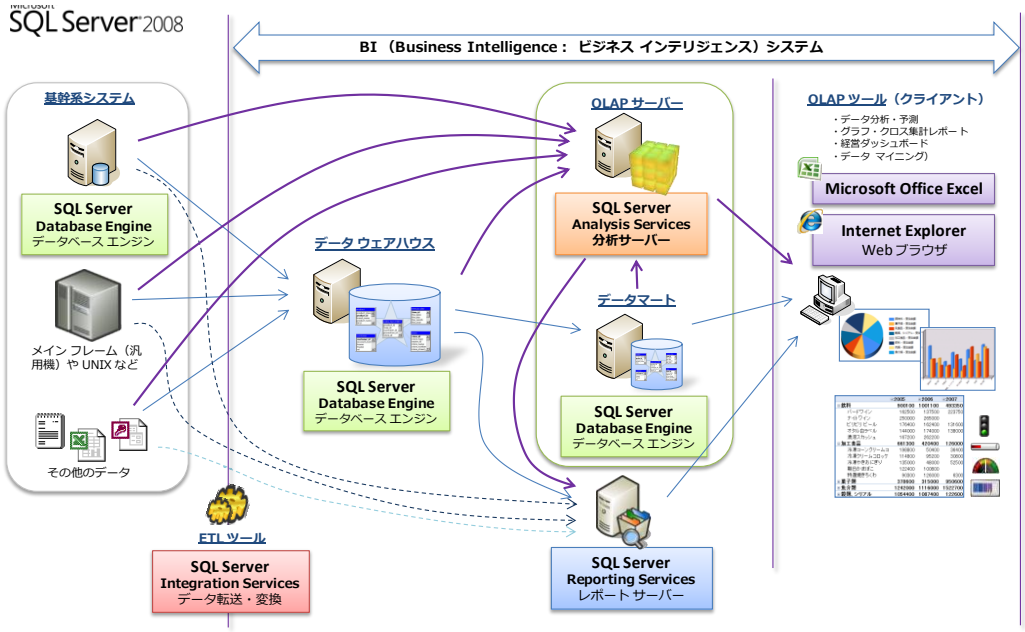


Integration Services を利用する手順については、本自習書シリーズの「**Integration Services 入門**」で説明しているので、ぜひご覧いただければと思います。

自習書シリーズ「Integration Services 入門」のダウンロードはこちらから
http://www.microsoft.com/japan/sqlserver/2008/self-learning/default.mspx

BI システムとしての SQL Server 2008

SQL Server 2008 には、次のように「**BI**」（Business Intelligence：ビジネス インテリジェンス）システムを構築するために必要となる機能がすべて標準で搭載されています。



データの抽出・変換が行える「**Integration Services**」、データ ウェアハウス（データの格納先）としての「**Database Engine**」、データ分析のためのサーバー機能を提供する「**Analysis Services**」、レポート機能を提供する「**Reporting Services**」など、BI システムを実現するためのすべての機能が SQL Server の標準コンポーネントとして提供されているので、手軽に（それでいて高度な）BI システムを構築することができます。したがって、現在は、BI 機能を利用する目的だけで SQL Server を導入する企業も増えてきています。

SQL Server 2008 共同検証プロジェクト（徹底検証ホワイト ペーパー）

徹底検証ホワイト ペーパーは、SQL Server 2008 早期実証プロジェクト「**CQI**」（**Center of Quality Innovation**）の成果物です。CQI は、マイクロソフトと日本電気株式会社、日本ヒューレット・パッカード株式会社、日本ユニシス株式会社の 3 社と共同で実施し、実際のユーザーに対する SI（System Integration）プロジェクトを想定した「**RFP**」（Request for Proposal：提案依頼書）に基づいた「**機能動作検証**」や「**パフォーマンス検証**」を行ったプロジェクトです。

このプロジェクトでは、次の 4 つのシナリオを実施しています。

1. **コンプライアンス シナリオ**

   マイクロソフトによる検証プロジェクト。内部統制や日本版 SOX 法、情報漏えい対策で求められるコンプライアンスに対応するためのガイドライン提示を目的とし、日本版 SOX 法への対応に必要となる機能の実装方法の確立やシステムに与える影響の計測などを実施

2. **移行 / アップグレード シナリオ**

   日本ユニシス株式会社様との共同検証プロジェクト。SQL Server 2000 および 2005 から、SQL Server 2008 への移行や互換性に関する検証を実施

3. **サーバー統合シナリオ**

   日本ヒューレット・パッカード株式会社様との共同検証プロジェクト。複数の SQL

Server を 1 台の 64 ビット サーバーに集約するメリット・運用管理方法などに関する検証を実施

4. **DWH（データ ウェアハウス）シナリオ**

   日本電気株式会社様との共同検証プロジェクト。SQL Server 2008 のデータ ウェアハウス環境における実装方法の検証を実施

これらの検証結果（ホワイト ペーパー）は、次の URL からダウンロードすることができます。

SQL Server 徹底検証シリーズ
http://www.microsoft.com/japan/sqlserver/2008/bible/cqi.mspx

この自習書シリーズの次のステップのハイ レベルな情報として、ぜひご覧いただければと思います。

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/）

SQL Server と .NET を中心とした「コンサルティング サービス」と「メンタリング サービス」を提供。

主なコンサルティング実績

- 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）
- 1 秒あたり 1,000 Batch Request の ASP（アプリケーション サービス プロバイダ）サイトのチューニング（ピーク時の CPU 利用率 100% を 10% まで軽減）
- 高負荷テスト（ラッシュテスト）実施のためのテスト アプリの作成支援
- 大手流通系システムの夜間バッチ実行時間を 4 時間から 1 時間半へ短縮
- 大手インターネット通販システムの夜間バッチ実行時間を 5 時間から 1 時間半へ短縮
- 宅配便トラッキング情報の日中バッチ実行時間を 2 時間から 5 分へ短縮
- 検索系 Web サイトのチューニング（10 倍以上のパフォーマンス UP を実現）
- 10 Server によるレプリケーション環境のチューニング
- 3 TB のセキュリティ監査アプリケーションのチューニング
- 大手家電メーカーの制御系アプリケーション（100GB）のチューニングと運用管理設計
- 約 3,000 本のストアド プロシージャとユーザー定義関数のチューニング
- ASP.NET / ASP（Active Server Pages）アプリケーションのチューニング
- 大手アミューズメント企業の BI システム設計支援
- 外資系医療メーカーの Analysis Services による「販売分析」システムの設計支援
- 大手企業の Analysis Services による「財務諸表分析」システムの設計支援
- Analysis Services OLAP キューブのパフォーマンス チューニング　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 / PASSJ 理事
MCDBA（Microsoft Certified Database Administrator）
MCSD for .NET（Microsoft Certified Solution Developer）

現在、SQL Server を中心とするコンサルティング、企業に対するメンタリング サービスなどを行っている。今までに手がけたコンサルティング案件は、テラバイト クラスの DB から少人数向け小規模 DB までと 幅広く多岐に渡る。得意分野はパフォーマンス チューニング。コンサルティング業務の傍ら、講演や執筆も行い、Microsoft 主催の最大イベント Tech・Ed や、PASSJ が主催するカンファレンスなどでスピーカーとしても活躍中。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 15,500 部発行）。のびのびになっている SQL Server の新書籍は、もうじき刊行予定。

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 / PASSJ 理事
MCDBA（Microsoft Certified Database Administrator）
MCSD for .NET（Microsoft Certified Solution Developer）

SQL Server のパフォーマンス 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、高可用性設計などを行う。また、過去には、実際のアプリケーション開発経験（ASP／ASP.NET、VB 6.0、Java、Access VBA など）と、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。最近は、Analysis Services と Excel 2007 による BI（ビジネス インテリジェンス）システムも得意とする。執筆時のペンネームは「百田昌馬」。月刊 Windows Developer マガジンの 『SQL Server でど～んといってみよう！』、DB マガジンの『SQL Server トラの穴』を連載。マイクロソフト公開のホワイトペーパー（技術文書）のゴースト ライターとして活動することもあり。過去、マイクロソフト認定トレーナー時代には、SMS（Systems Management Server）や、Proxy Server、Commerce Server、BizTalk Server、Application Center、Outlook CDO などの講習会も担当。1998 年度には、Microsoft CTEC（現 CPLS）トレーナー アワード（Trainer of the Year）を受賞。