Microsoft

SQL Server 2017

SQL Server 2017 自習書シリーズ No.1

SQL Server 2017 の新機能の概要

Published: 2017 年 10 月 20 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 2017 の概要

この STEP では、SQL Server の最新バージョンである「**SQL Server 2017**」で提供された主な新機能の概要を説明します。

この STEP では、次のことを学習します。

- ✓ SQL Server 2017 評価版の利用／ダウンロード
- ✓ SQL Server 2017 の主な新機能

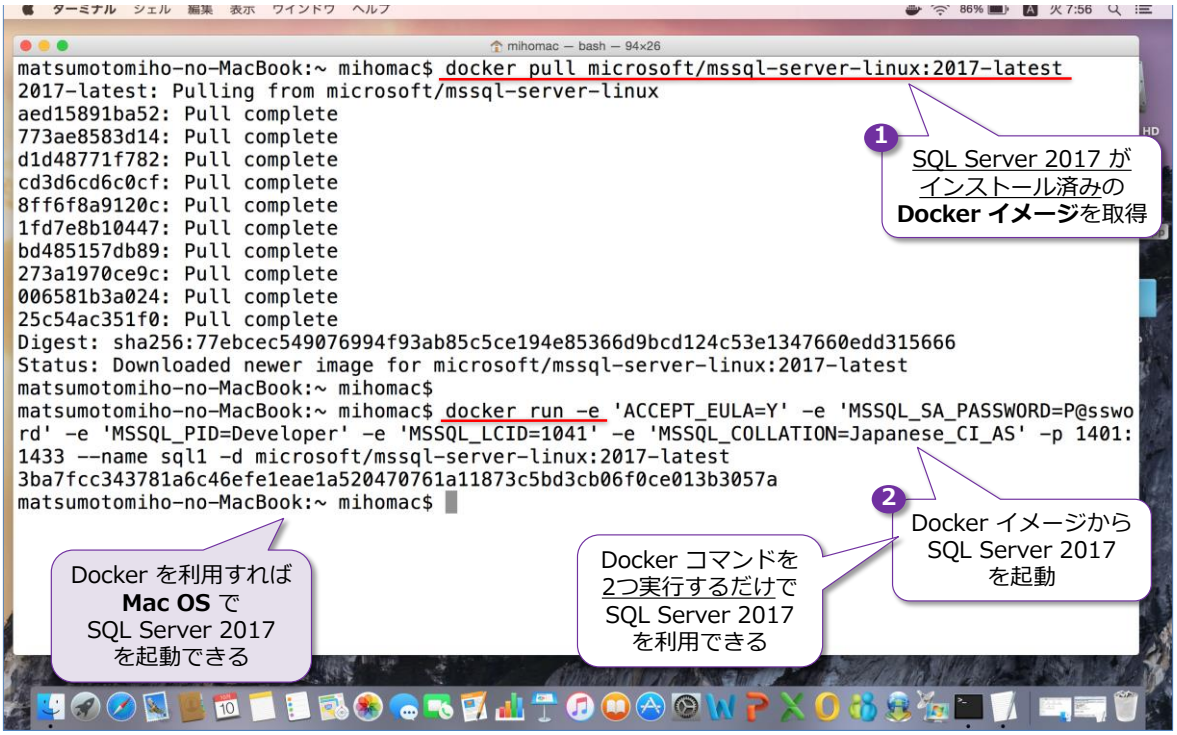
1.1 SQL Server 2017 評価版の利用／ダウンロード

SQL Server の最新バージョンである「**SQL Server 2017**」は、2017 年 10 月に発売されました。1 つ前のバージョンである SQL Server 2016 の発売は 2016 年 6 月でしたので、わずか 1 年半弱でのバージョン アップになりますが、SQL Server 2017 には非常に多くの新機能が提供されています。

目玉の新機能は、やはり**マルチ プラットフォーム**（**Linux** や **Mac OS** に対応！）と、**ビルトイン AI**（AI 機能を SQL Server データベース エンジンに統合）、**自動チューニング**（Automatic Tuning）機能の搭載、**グラフ データベース**対応などです。

SQL Server 2017 では、ついに **Linux**（&**Docker**）をサポートしたので、Windows 環境ではもちろんのこと、**Linux**（Ubuntu や Red Hat Enterprise Linux）でも、**Mac OS**（Docker を利用）でも動作させることができます。

Docker を利用する場合は、既に **SQL Server 2017** がインストールされた状態の Docker イメージが提供されているので、**Docker コマンドを 2 つ実行**するだけで、SQL Server 2017 をインストールする必要なく、簡単に SQL Server 2017 を試すことができます。

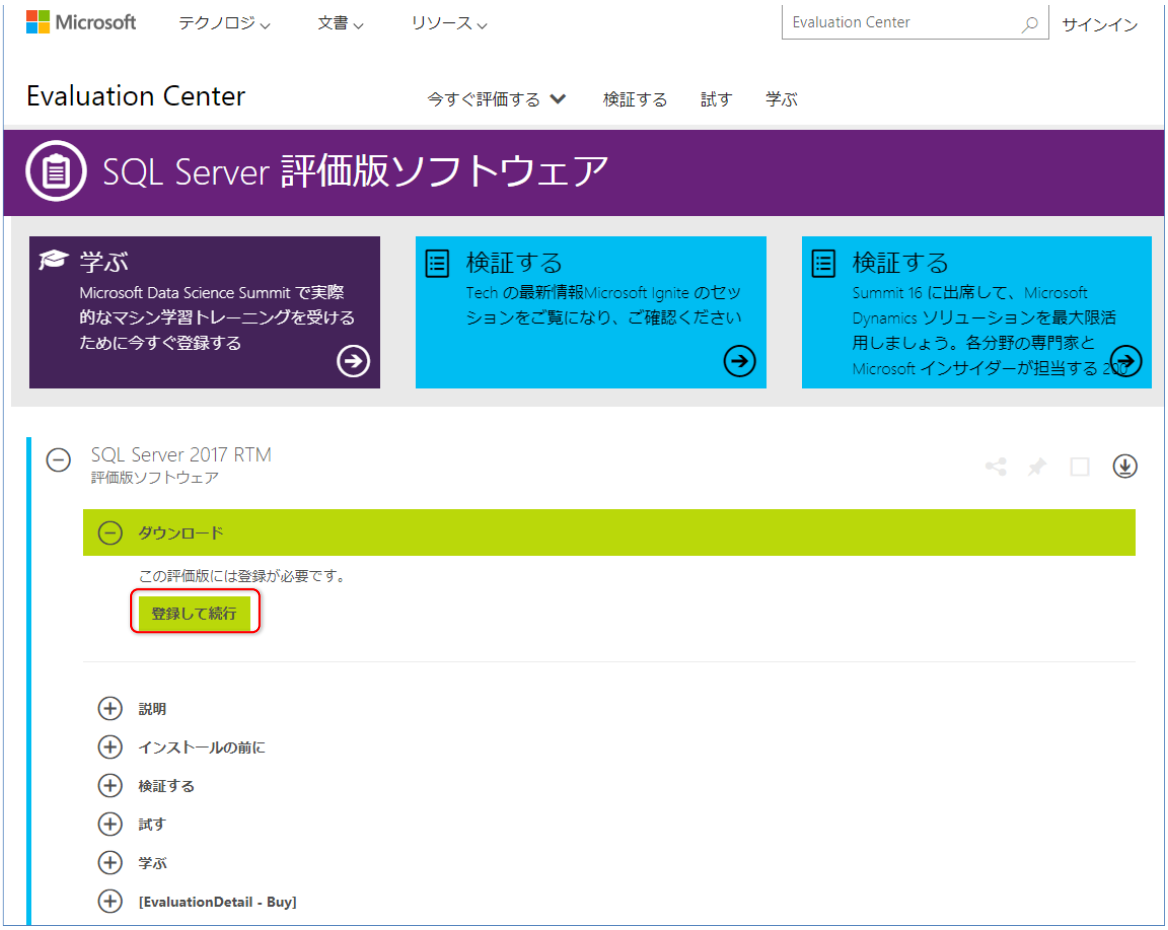


Docker での SQL Server 2017 の利用方法については、第 2 章で詳しく説明しているので、ぜひ試してみてください。**Linux** 環境への SQL Server 2017 のインストール方法についても、第 2 章で詳しく説明していますが、これも **5 個のコマンド**を実行するだけで、簡単にインストールすることができます。こちらもぜひ試してみてください。

Windows 環境での評価版のダウンロード

Windows 環境で SQL Server 2017 を評価したい場合には、次の URL から**評価版**（Evaluation Edition）をダウンロードすることができます。

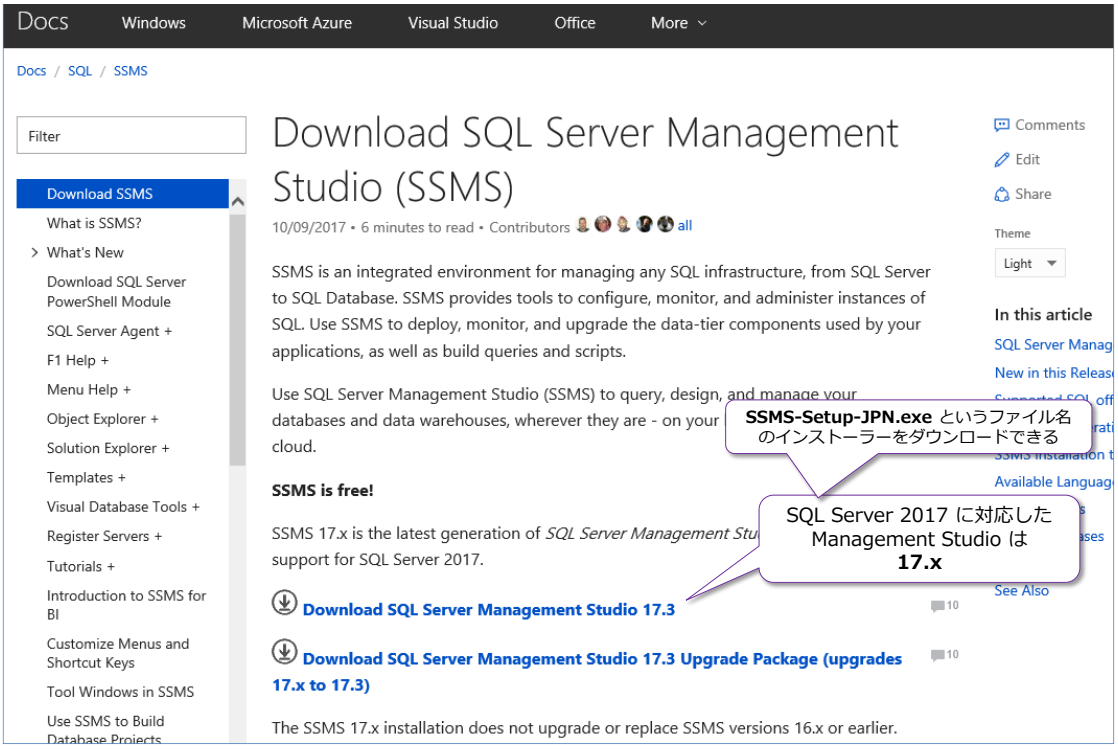
評価版のダウンロード：Microsoft SQL Server 2017
https://www.microsoft.com/ja-jp/evalcenter/evaluate-sql-server-2017-rtm



Windows 環境での Management Studio のダウンロード／インストール

Windows 環境での SQL Server の管理ツールである **Management Studio** は、SQL Server 2016 からはダウンロード版のみの提供に変わりました。最新版の Management Studio は、次の URL からダウンロードすることができます。

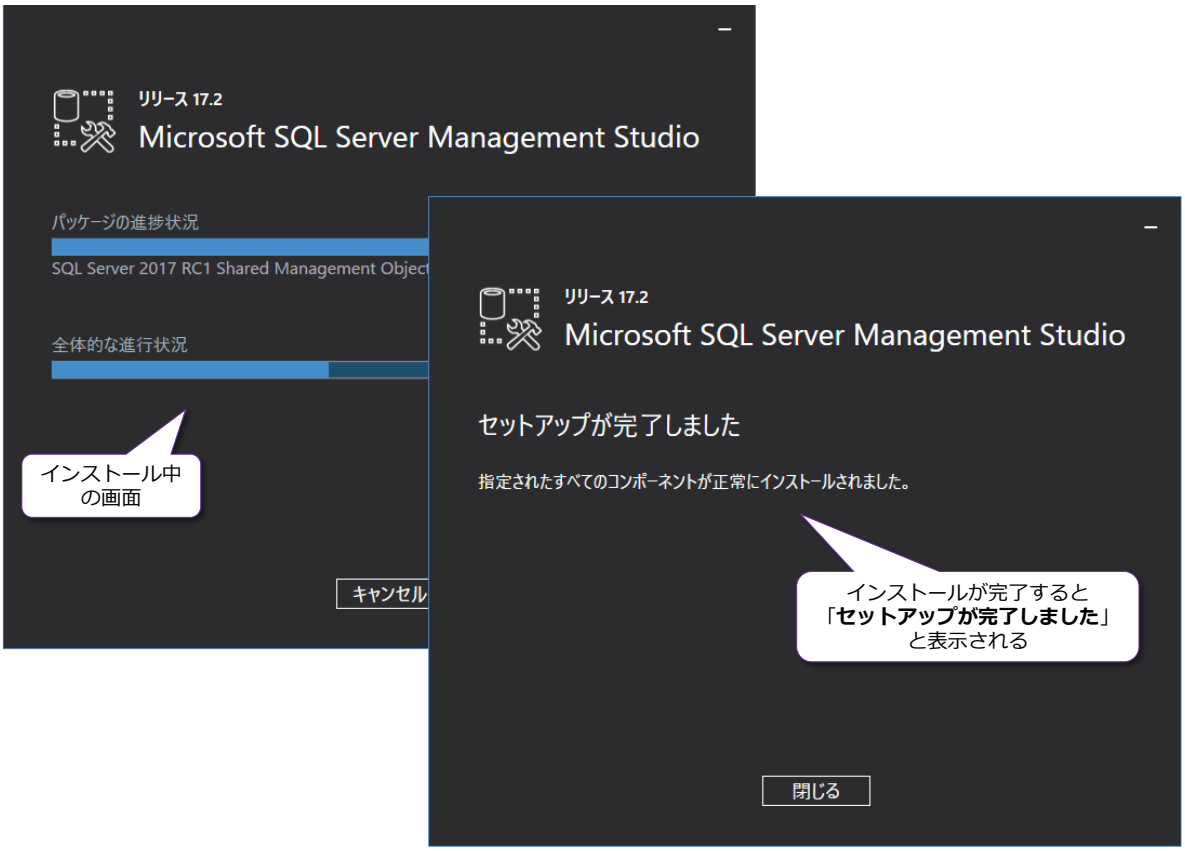
Management Studio の最新版のダウンロード
https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms



この Web サイトは、英語のページですが、日本語環境であれば、日本語版のインストーラーである「**SSMS-Setup-JPN.exe**」ファイルのダウンロードが始まるので、「**-JPN**」が付くことを確認しておいてください（もし、-JPN が付かない場合は、ページを下にスクロールしていくと、「**Available Languages**」というセクションがあって、そこに各国版のインストーラーへのリンクがあるので、そこで **Japanese** を選択すると日本語版をダウンロードすることができます）。

ダウンロードした **SSMS-Setup-JPN.exe** ファイルを実行すると、次のようにインストーラーが起動するので、［**インストール**］ボタンをクリックすれば、インストールを開始できます。

Management Studio のインストール中は、次のように進行状況が表示されます。



「**セットアップが完了しました**」と表示されれば、Management Studio のインストールが完了です（環境によっては、完了後に再起動が促される場合があります）。

1.2 SQL Server 2017 で提供された主な新機能

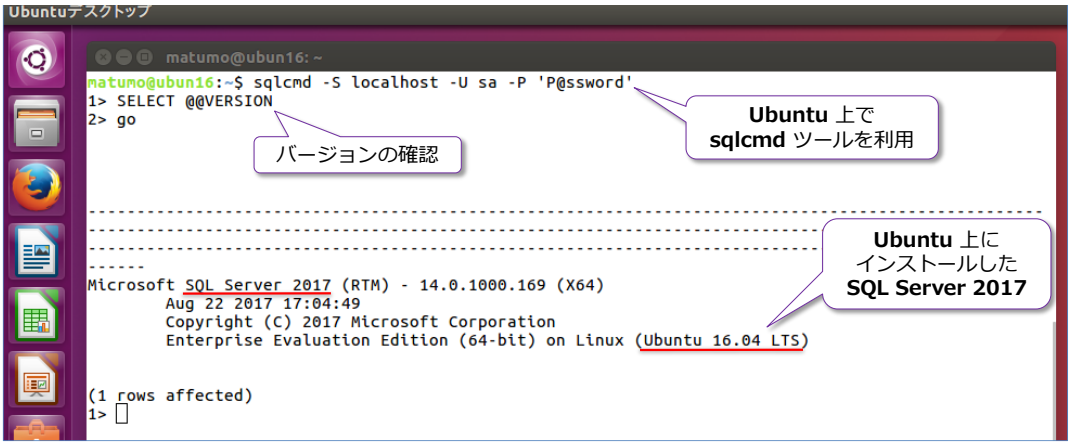
SQL Server 2017 には、非常にたくさんの新機能が提供されています。これらをまとめると、次のようになります。

	SQL Server 2017 からの主な新機能
Linux 対応	・**SQL Server 2017 on Linux** サポート プラットフォーム - Red Hat Enterprise Linux 7.3／7.4 Workstation, Server, and Desktop - SUSE Enterprise Linux Server v12 SP2 - Ubuntu 16.04 LTS - Docker Engine 1.8 以上 ・**on Linux で利用できる機能** - データベース エンジンに関するほぼすべての機能（詳細は 2章参照） - SQL Server Agent ジョブ（Transact-SQL の定期実行） - SSIS パッケージ（.dtsx）の実行
ビルトイン AI 機械学習	・**Python 統合**（Machine Learning Services） Python 言語をデータベース エンジンに統合（ビルトイン） Python における**ディープ ラーニング**（Deep Learning : 深層学習）の定番フレームワークである Microsoft Cognitive Toolkit（CNTK）や Chainer、TensorFlow、Caffe、Theano などを利用可能。GPU にも対応 ・**ネイティブ スコアリング**（PREDICT 関数による予測の実行）
注目の新機能	・**グラフ データベース**（Graph processing） ・**自動チューニング**（Automatic Tuning） ・**Adaptive Query Processing**（適応型クエリ処理） ・**クエリ ストア**の強化、**DTA** の強化、Scan/Read Ahead アルゴリズム改善 ・**列ストア インデックス**の強化（オンライン再構築や LOB 対応、性能向上etc） ・**インメモリ OLTP** の強化（制限緩和の追加や性能向上etc） ・**再開可能**なオンライン インデックス再構築（Resumable Rebuild Index） ・**AlwaysOn 可用性グループ**の強化（DTC 対応、クラスター レス構成など）
その他の新機能	・**Transact-SQL の強化** 新しい関数（TRIM、CONCAT_WS、TRANSLATE、STRING_AGG）、SELECT INTO でのファイル グループ指定、IDENTITY_CACHE オプションのサポート、日本語向けの新しい照合順序の追加など ・**セットアップ時の変更点**（tempdb ファイルの設定） ・**スマート バックアップ**、バックアップ性能の向上、Indirect Chekpoint ・**新しい DMV**（動的管理ビュー） ・**テンポラル テーブルの強化**（保持ポリシーの追加など） ・**XE プロファイラー**の提供、実行プランの検索機能 ・**SQL CLR** の**セキュリティ強化**
BI 関連の強化	・**Analysis Services の強化** Power Query Formula Language（M 言語）対応、ドリルスルー データでの DAX 指定、Ragged 階層対応、Object レベル セキュリティ、DAX 強化、DMV 強化、DISCOVER_CALC_DEPENDENCY など ・**Reporting Services の強化** レポート コメント、DAX エディター for SSDT、REST API 対応、軽量のインストーラー など ・**Integration Services の強化** SSIS パッケージのスケールアウト実行、Linux 対応 など ・**MDS（マスター データ サービス）の強化**

この章では、これらの新機能の概要を説明します。

SQL Server 2017 on Linux ～マルチ プラットフォーム～

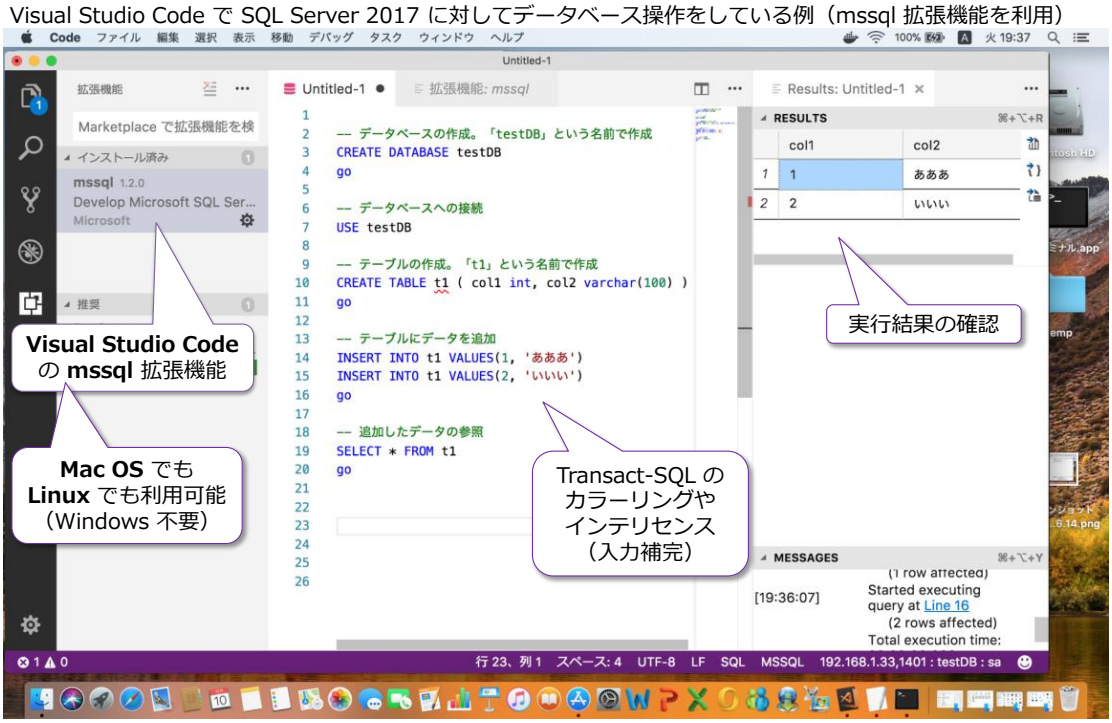
SQL Server 2017 の一番の目玉の新機能は、なんといっても**マルチ プラットフォーム**対応です。SQL Server がついに **Linux** でも、**Mac OS** でも動作するようになりました。



SQL Server 2017 on Linux は、ただ単に Linux 上で SQL Server を動かせるようにしただけではなく、ほとんどの SQL Server の機能（データベース エンジンに関する機能）を、Windows 上の SQL Server とまったく同じように利用できます。データベースに関する**基本操作**（データベースの作成やテーブル作成、データの追加／更新／削除、ビューやストアド プロシージャ、トリガー、インデックスの作成）ができることはもちろん、性能向上を実現できる**列ストア インデックス**や、**インメモリ OLTP**、**データ パーティション**、**データ圧縮**も利用できます。

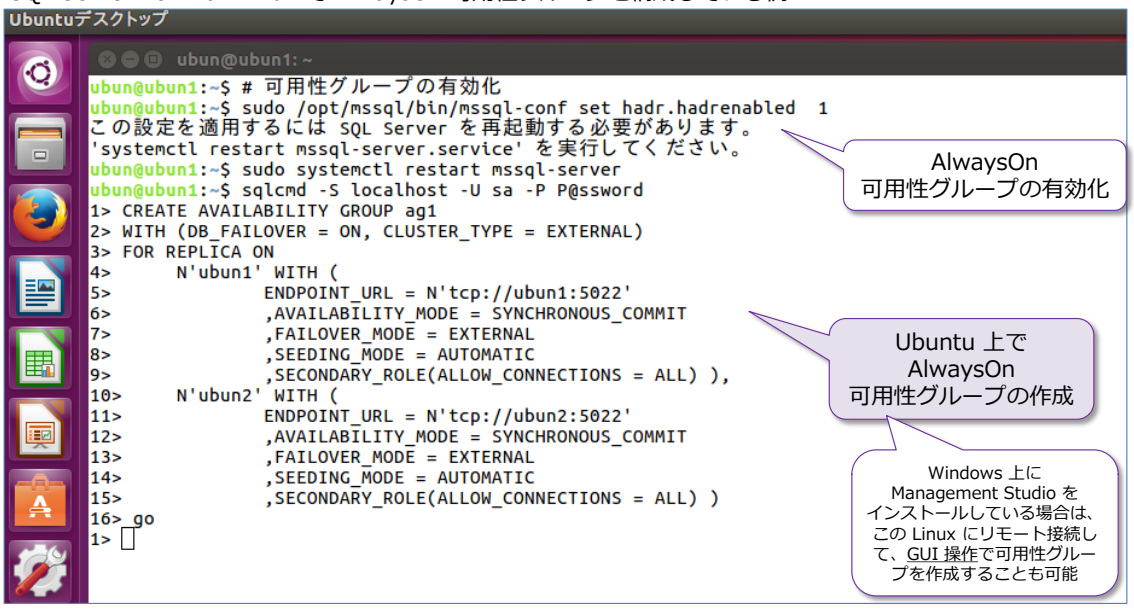
Visual Studio Code などのアプリケーション開発ツールを利用すれば、Windows を利用することなく、Mac OS や Linux だけで、SQL Server 2017 に関する操作をグラフィカルに行うことができ、合わせてアプリケーション開発（Java や PHP、Python、C#、ASP.NET、node.js など）も可能です。

Visual Studio Code で SQL Server 2017 に対してデータベース操作をしている例（mssql 拡張機能を利用）



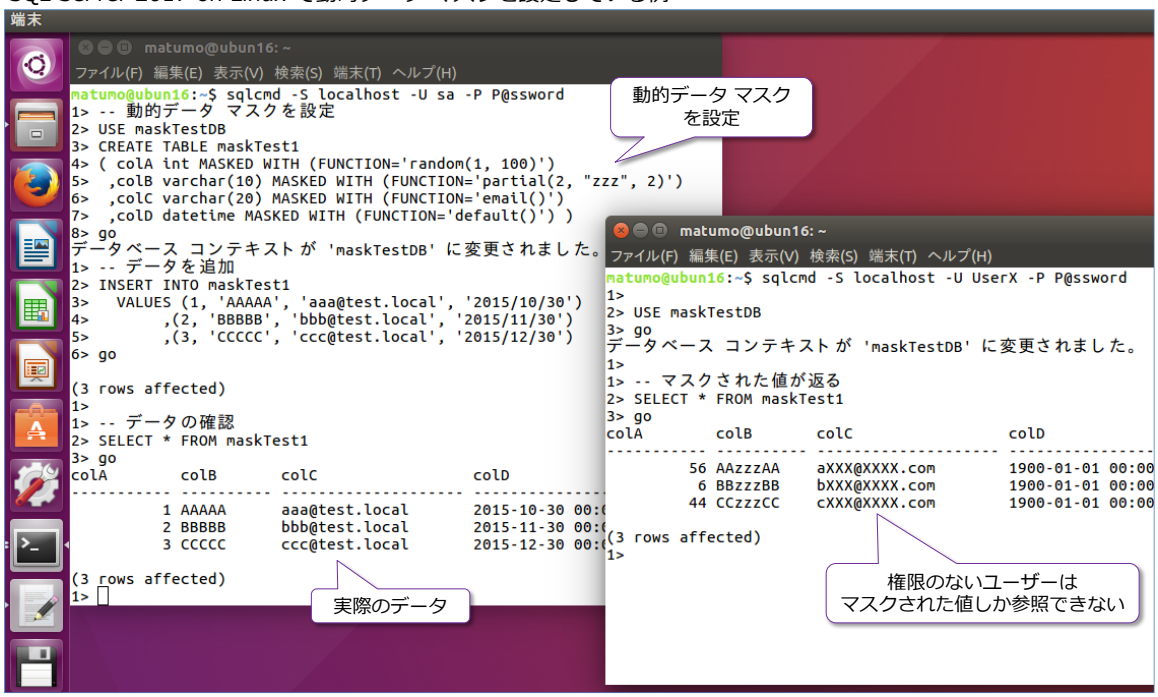
SQL Server 2017 on Linux では、後述の**自動チューニング**機能や、**クエリ ストア**、**グラフ データベース**、**AlwaysOn 可用性グループ**、**スマートバックアップ**などについても、Windows 上の SQL Server を操作するのとまったく同じように利用できます。

SQL Server 2017 on Linux で AlwaysOn 可用性グループを構成している例

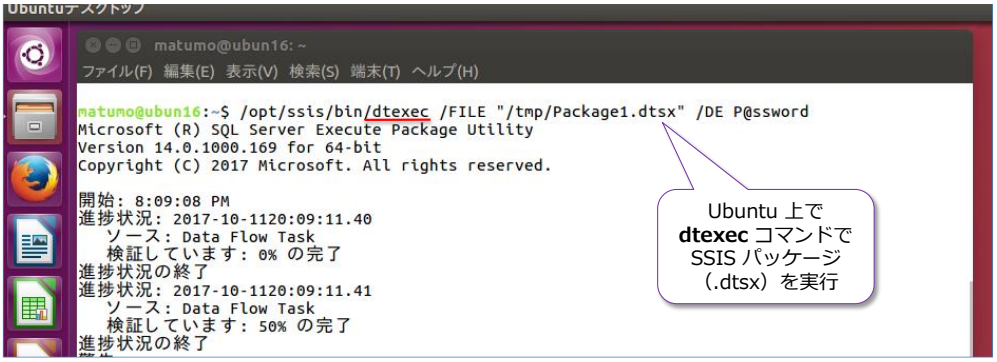


また、SQL Server が標準で搭載している**セキュリティ機能**（ユーザー作成やオブジェクト権限の設定だけでなく、監査や行レベル セキュリティ、動的データ マスク、TDE：透過的なデータ暗号化、バックアップ暗号化、ネットワーク接続の暗号化、Always Encrypted、テンポラル テーブル、包含データベースなど）についても、on Linux で同じように利用することできます。

SQL Server 2017 on Linux で動的データ マスクを設定している例



SQL Server 2017 on Linux では、その他のシステムとの連携に役立つ**リンク サーバー**や **bcp**、**BULK INSERT** なども利用することができ、**Integration Services**（SSIS）のパッケージ（.dtsx）を実行することもできます。



また、SQL Server におけるバックアップのスケジュールなどで定番となっている **SQL Server Agent ジョブ**機能もサポートしているので、バックアップやインデックスの再構築といった各種のメンテナンス系の SQL をスケジュール実行することもできます。**データベース メール**機能もサポートしているので、ジョブの成功や失敗をメールで通達するといったことも行えます。

さらには、次に紹介する **Machine Learning Services** で**機械学習**したモデルを利用した**予測**（**PREDICT** 関数によるスコアリング）を行うこともできます。

Windows 上の SQL Server 環境で取得したバックアップを、Linux 環境にリストアすることも、何の問題もなく行えるので、**移行**（マイグレーション）も簡単です（リストア時の考慮事項は、Windows 環境でのリストアの場合と全く同様です）。

SQL Server 2017 on Linux の性能 ～TPC-H など～

SQL Server 2017 on Linux は、性能に関しても結果が出ています。例えば、次の画面ショットは、執筆時点（2017 年 10 月）での **TPC-H ベンチマーク**での Non-Clustered カテゴリの **1TB**（1,000GB）の結果です。

TPC-H - Top Ten Performance Results - Non-Clustered Version 2 Results As of 15-Oct-2017 at 4:15 PM [GMT]

Note 1: The TPC believes that comparisons of TPC-H results measured against different database sizes are misleading and discourages such comparisons. The TPC-H results shown below are grouped by database size to emphasize that only results within each group are comparable.
Note 2: The TPC believes it is not valid to compare prices or price/performance of results in different currencies.

All Active Results | Active Clustered Results | Active Non-Clustered Results **Currency:** All | Include Historical Results

1,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted
1	Hewlett Packard Enterprise	HPE Proliant DL380 Gen9	717,101	.61 USD	NR	10/19/17	Microsoft SQL Server 2017 Enterprise Edition	Red Hat Enterprise Linux Server 7.3	04/17/17
2	Hewlett Packard Enterprise	HPE Proliant DL380 Gen9	678,492	.64 USD	NR	07/31/16	Microsoft SQL Server 2016 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard Edition	03/24/16
3	CISCO	Cisco UCS C460 M4 Server	588,831	.97 USD	NR	12/16/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard	12/15/14
4	Hewlett Packard Enterprise	HPE Proliant DL380 Gen9	543,102	.69 USD	NR	07/31/16	Microsoft SQL Server 2016 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard Edition	03/09/16

* 執筆時点（2017年 10月）での TPC-H Non-Clustered 1,000GB テストの結果 http://www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster

TPC-H は、データ ウェアハウス／意志決定支援システム向けの**ベンチマーク テスト**として有名なものですが、1 位は **Red Hat Enterprise Linux Server 7.3** 上で動作している **SQL Server 2017** で、2 位は **Windows Server 2012 R2** 上で動作している **SQL Server 2016** です。

2 位の SQL Server 2016 on Windows が **678,492 QphH**（1 時間あたりの DWH クエリ実行数）であるのに対して、1 位の SQL Server 2017 on Linux では **717,101 QphH**、1 クエリあ

たりのコスト（**Price/QphH**）に関しては、**.64** USD（US ドル）が **.61** USD に下がっており、性能向上とともにコスト削減も実現しています。

なお、TPC-H ベンチマークは、SQL Server 2016 のときに **10TB** テストで**世界記録**（ワールドレコード）を更新しましたが、SQL Server 2017 on Windows でさらに記録を更新しています。

10,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted
1	Lenovo	Lenovo ThinkSystem SR950	1,336,109	.92 USD	NR	10/19/17	Microsoft SQL Server 2017 Enterprise Edition	Microsoft Windows Server 2016 Standard Edition	07/09/17
2	Cisco	Cisco UCS C460 M4 Server	1,115,298	.87 USD	NR	11/28/16	Microsoft SQL Server 2016 Enterprise Edition	Microsoft Windows Server 2016 Standard Edition	11/28/16
3	Lenovo	Lenovo System x3850 X6	1,106,832	.89 USD	NR	09/30/16	Microsoft SQL Server 2016 Enterprise Edition	Microsoft Windows Server 2016 Standard Edition	07/11/16
4	Hewlett Packard Enterprise	HPE Proliant DL580 Gen9	1,047,243	1.07 USD	NR	09/30/16	Microsoft SQL Server 2016 Enterprise Edition	Microsoft Windows Server 2016 Standard Edition	06/27/16
5	Hewlett Packard Enterprise	HP Integrity Superdome X	780,346	2.27 USD	NR	02/03/16	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard Edition	02/02/16

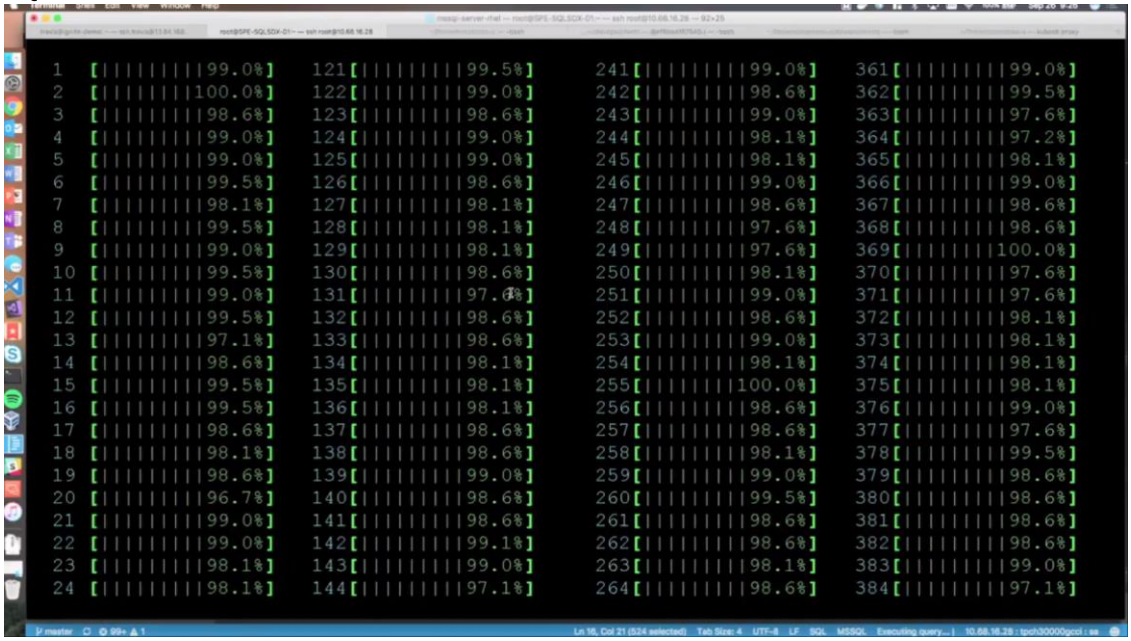
* 執筆時点（2017年 10月）での TPC-H Non-Clustered 10TB テストの結果 http://www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster

SQL Server 2016 で **1,115,298 QphH** だったところを、SQL Server 2017 では **1,336,109 QphH** を達成しました。SQL Server 2017 では、**Adaptive Query Processing**（適応型クエリ処理）や、スキャン／Read Ahead アルゴリズムの改善、インダイレクト CheckPoint、インメモリ OLTP／列ストア インデックスの強化など、**データベース エンジンの性能向上**も図っているので、その成果が出ています。

480 スレッドでもスケール（SQL Server 2017 on Linux）

昨今は、1 個の CPU あたりのコア数が非常に多い "**メニー コア**" 時代になりましたが、コア数が増えてもスケールしないデータベースが存在する中（NUMA に対応していなかったり、64 コアを超える K グループ環境に対応していないデータベースがあったりする中）、SQL Server はメニーコア環境でもスケールします。今年の 9 月に開催された Microsoft Ignite 2017 では、**480 スレッド**分もの **CPU をフル活用**して、スケールしているデモがありました（以下）。

SQL Server 2017 on Linux で 480 スレッドの CPU をフル活用している様子。メモリは 12TB 搭載



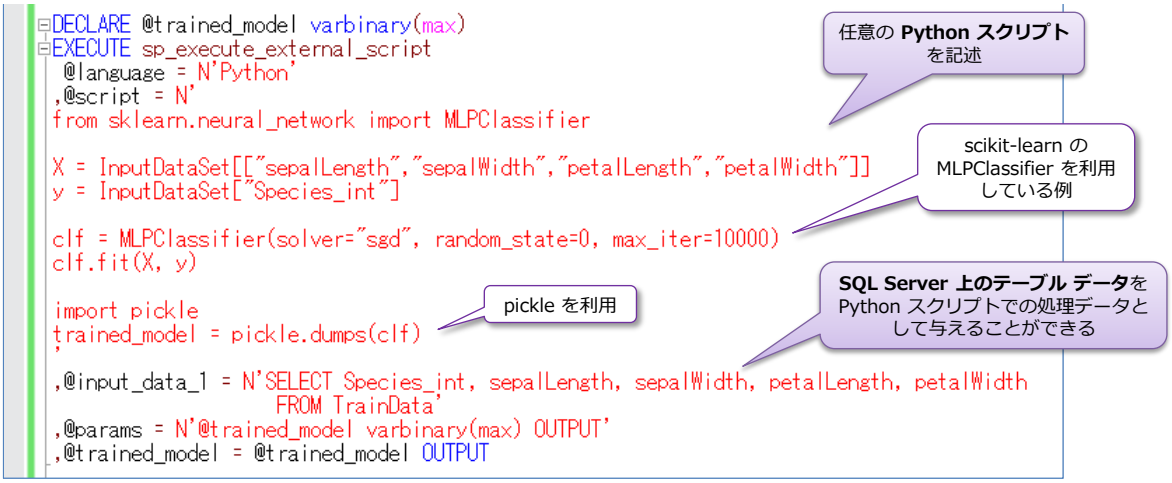
https://myignite.microsoft.com/videos/54946 :Microsoft Ignite 2017 Session「Microsoft SQL Server 2017 deep dive」より引用

こうしたメニー コアへの対応は、Windows 版ではお馴染みでしたが、Linux 版でも見事に対応しています（そうした結果が前述の TPC-H の結果に繋がっています）。

ビルトイン AI（AI 機能を DB エンジンに統合。Python をビルトイン）

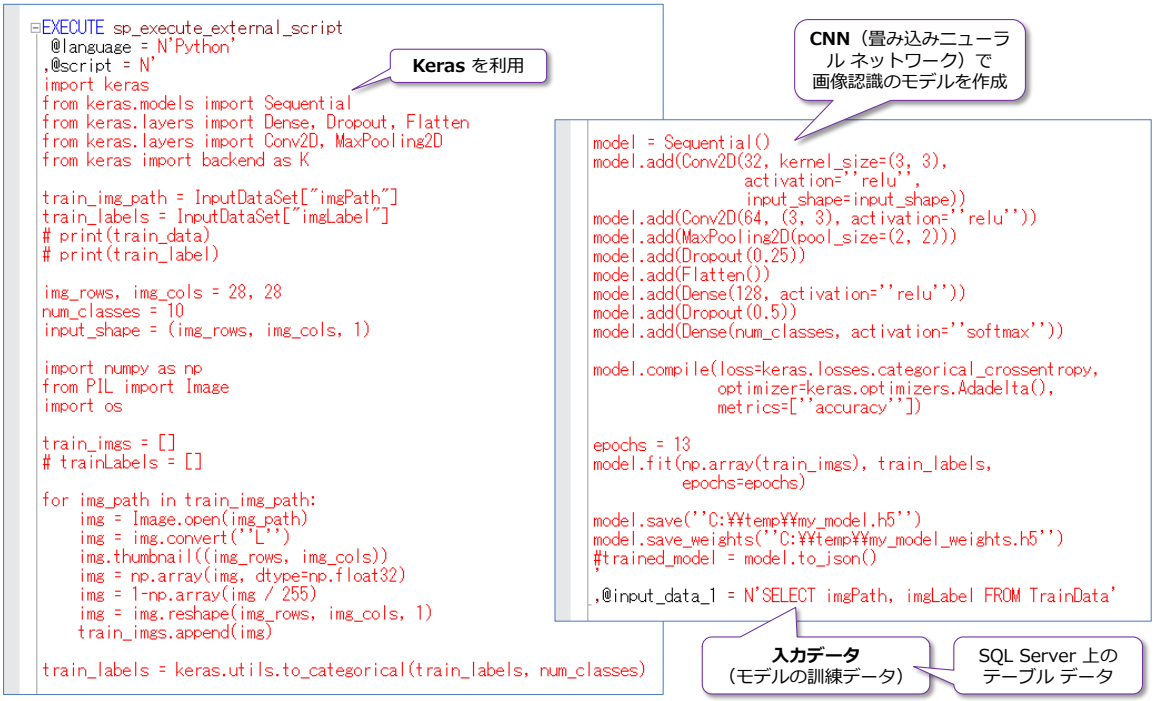
SQL Server 2017 では、**ビルトイン AI** も大きな目玉機能です（SQL Server は、初めて AI 機能をデータベースに統合した製品です）。SQL Server 2016 では、「**SQL Server R Services**」という形で、R 言語を SQL Server に統合（ビルトイン）して、**機械学習**（Machine Learning）によるモデルの作成や予測（Predict）を行うことができましたが、SQL Server 2017 からは、**Python** 言語も SQL Server に統合しました。Python は、現在、機械学習／データ サイエンティストに最も人気があり、最も利用者数が多い言語で、これを SQL Server 上で利用することができます。

この機能は、「**Machine Learning Services**」（以降 **ML Services** と記述）と呼ばれ、次のように SQL Server 上のデータを入力値（機械学習における訓練データやテスト データ）として Python スクリプトを実行することができます。



SQL Server 2017 の Python 統合では、Python での定番ライブラリである「**NumPy**」や「**pandas**」、「**scikit-learn**」、「**pickle**」、「**PIL**」などを利用できることはもちろんのこと、**ディープ ラーニング**（Deep Learning：深層学習）での**定番フレームワーク**である「**Microsoft Cognitive Toolkit**（**CNTK**）」や「**Chainer**」、「**Google TensorFlow**」、「**Caffe**」、「**Theano**」なども利用できます。これによって、昨今のトレンドである**ニューラル ネットワーク**や**ディープ ラーニング**を利用した画像認識や音声認識、自然言語処理、各種の予測（Predict）およびモデル作成といった、いわゆる **AI**（人工知能）を実装することができます。

CNN（Convolutional Neural Network：畳み込みニューラル ネットワーク）で画像認識をしている例




```
EXECUTE sp_execute_external_script
 @language = N'Python'
,@script = N'
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

train_img_path = InputDataSet["imgPath"]
train_labels = InputDataSet["imgLabel"]
# print(train_data)
# print(train_label)

img_rows, img_cols = 28, 28
num_classes = 10
input_shape = (img_rows, img_cols, 1)

import numpy as np
from PIL import Image
import os

train_imgs = []
# trainLabels = []

for img_path in train_img_path:
    img = Image.open(img_path)
    img = img.convert(''L'')
    img.thumbnail((img_rows, img_cols))
    img = np.array(img, dtype=np.float32)
    img = 1-np.array(img / 255)
    img = img.reshape(img_rows, img_cols, 1)
    train_imgs.append(img)

train_labels = keras.utils.to_categorical(train_labels, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation=''relu'',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation=''relu''))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation=''relu''))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation=''softmax''))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=[''accuracy''])

epochs = 13
model.fit(np.array(train_imgs), train_labels,
          epochs=epochs)

model.save(''C:¥¥temp¥¥my_model.h5'')
model.save_weights(''C:¥¥temp¥¥my_model_weights.h5'')
#trained_model = model.to_json()

,@input_data_1 = N'SELECT imgPath, imgLabel FROM TrainData'
```

上の例は、**Keras**（TensorFlow や Microsoft Cognitive Toolkit、Theano をバックエンドに利用するニューラル ネットワークのライブラリ）を利用して、**CNN**（Convolutional Neural Network：畳み込みニューラル ネットワーク）で画像認識のためのモデルを作成しているものです。CNN は、画像認識や音声認識で非常によく利用されているニューラル ネットワークで、現在のディープ ラーニングは、ほとんどが CNN がベースになっています。

このように、ML Services を利用すれば、通常の Python と同様にプログラミングすることができ、また、入力データに関しては SQL Server から直接取得することができるので、その分のコードを記述する必要がなくなります。

また、ML Services は、**GPU**（Graphics Processing Unit）にも対応しているので、GPU を利用すれば CPU よりも高速に演算を行うことができます。

ML Services や on Linux に代表される、最近の SQL Server の進化は、昨今のマイクロソフト社の方向性と同様、オープン化が目覚ましく、今までの Windows や .NET に捕らわれないアプリケーション開発が行えるのは、本当にいろいろな可能性を感じさせてくれます。

SQL Server 2017 の注目の新機能（グラフ、自動チューニングなど）

SQL Server 2017 における注目の新機能は、まだまだあります。ここでは、次の新機能の概要を説明します。

- グラフ データベース（Graph processing）
- 自動チューニング（Automatic Tuning）
- Adaptive Query Processing（適応型クエリ処理）

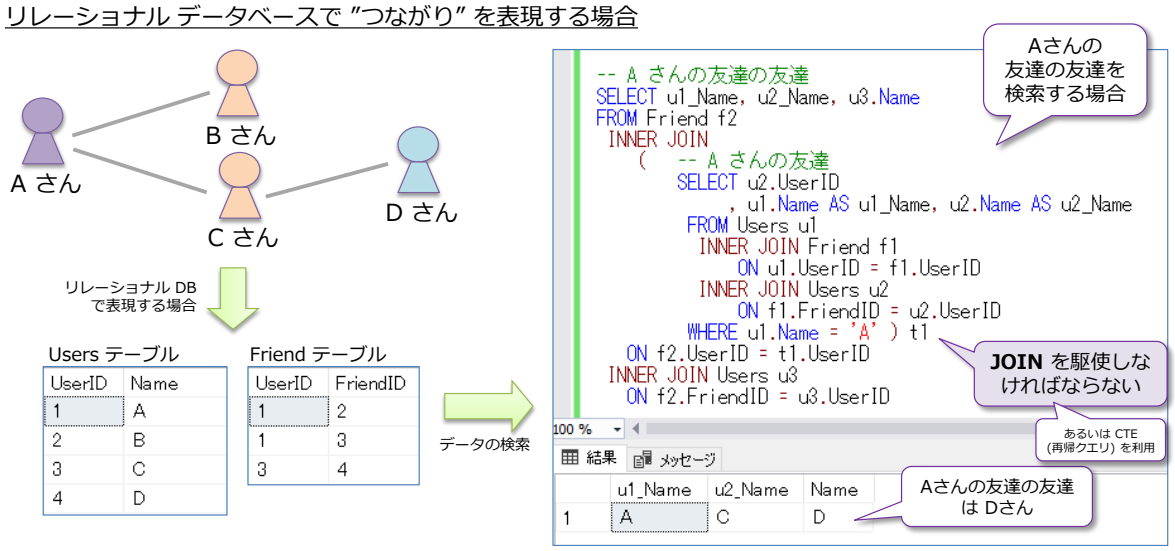
- クエリ ストアの強化、DTA の強化
- 列ストア インデックスの強化
- インメモリ OLTP の強化
- 再開可能なオンライン インデックス再構築
- AlwaysOn 可用性グループの強化

グラフ データベース（Graph processing）

SQL Server 2017 は、**グラフ データベース**に対応しました。グラフ データベースは、Facebook や Twitter といった **SNS**（ソーシャル ネットワーキング サービス）における**データの繋がり**や、**多対多**（Many-To-Many）の関係を表現するのが得意なデータベースです。

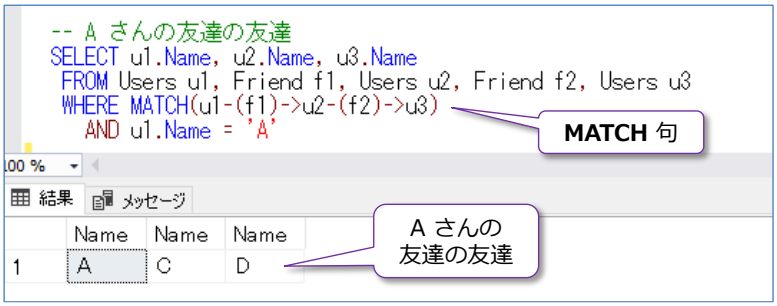
例えば、次のようにデータの繋がり（A さんの友達は B さんと C さんといった関係）を、リレーショナル データベースで表現したとすると、データを検索するときに JOIN を駆使しなければならなくなります。

リレーショナル データベースで "つながり" を表現する場合



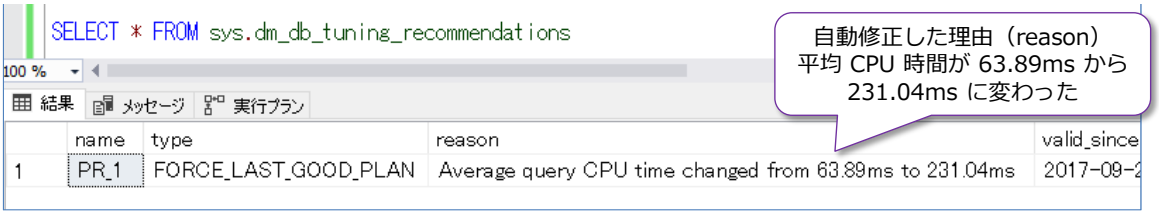
これに対して、グラフ データベース機能を利用すれば、データを検索するときに、次のように **MATCH** 句を利用して、非常にシンプルに記述できるようになります。

グラフ データベースでは MATCH でデータのつながりを簡単に取得できる

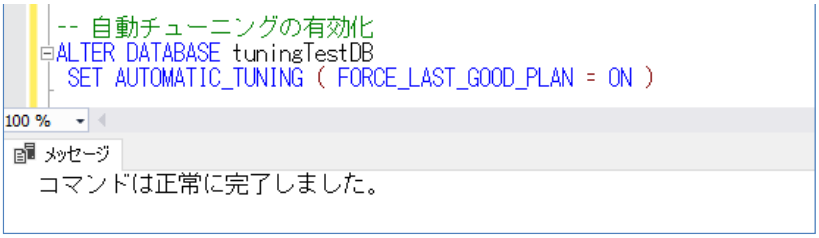


自動チューニング機能（Automatic Tuning）

SQL Server 2017 では、ついに**自動チューニング**機能が提供されました。これを利用すれば、クエリの過去の実行履歴をもとに、遅くなったクエリを自動判別して、遅くなる前の正常に実行できていた状態に自動的に戻すことができます。自動修正されたクエリは、次のように **dm_db_tuning_recommendations** 動的管理ビューで確認することができます（自動修正された理由も確認できます）。



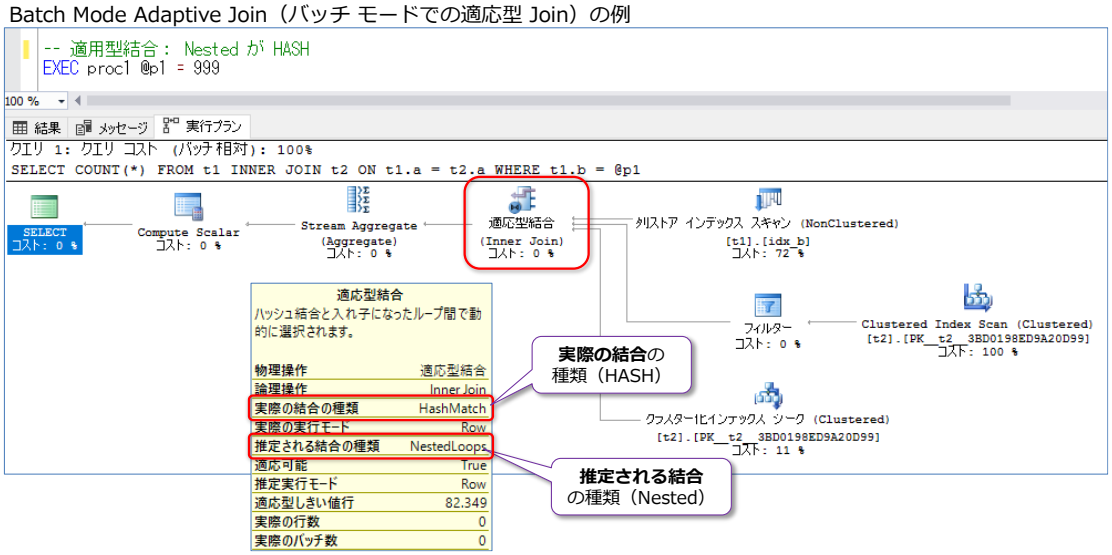
自動チューニングの有効化は、次のように ALTER DATABASE ステートメントを実行するだけで完了です。



Adaptive Query Processing（適応型クエリ処理）

Adaptive Query Processing（適応型クエリ処理）は、性能向上のためのインテリジェントなクエリ処理機能です。**Batch Mode Adaptive Join**（バッチ モードでの適応型 Join）や、**Batch Mode Memory Grant Feedback**（バッチモードでのメモリ割当てフィードバック）、**Interleaved Execution for MSTVF**（Multi-Statement Table Valued Function）の 3 種類の機能が提供されています。

Batch Mode Adaptive Join（バッチ モードでの適応型 Join）の例



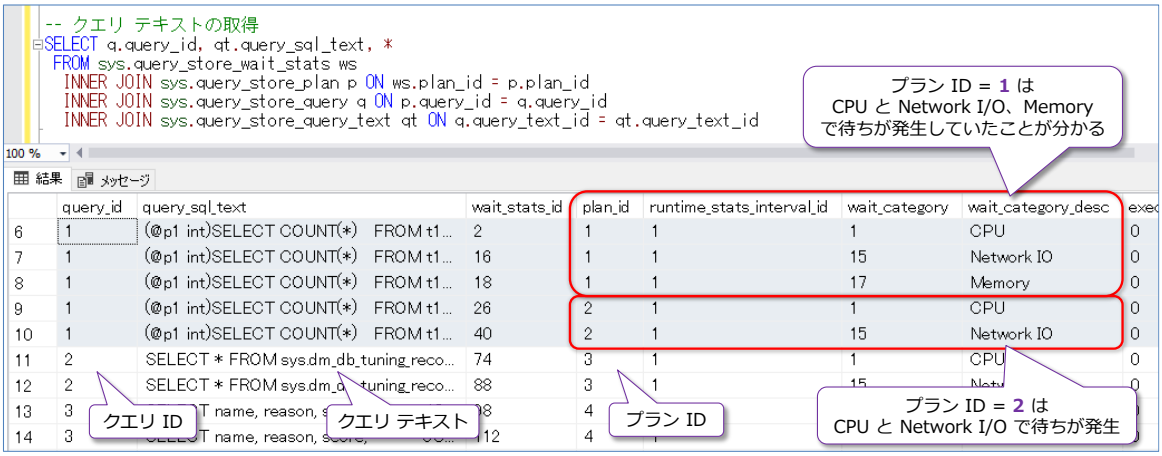
通常の Join では、Nested Loop や Hash Join などが利用されますが、Nested Loop は片方のテーブルのデータ量が小さい場合、Hash Join は両方のテーブルのデータ量が多い場合に効率の良い Join 方式です。しかし、クエリに与える入力パラメーターによっては、片方のテーブルのデータが小さくなったり、大きくなったりするのは、実際の現場のデータでは良く起こりえます。

そういった場合に、データ量が少ない場合には Nested Join、データ量が多い場合には Hash Join を**自動的に選択**してくれる（実行プランとしては、**適応型 Join**（**Adaptive Join**）という形で登録されて、どちらの Join にも対応／自動変換できるようになっている）機能が、**Batch Mode Adaptive Join**（**Adaptive Query Processing** の 1 つの機能）です。

クエリ ストアの強化

クエリ ストアは、**クエリの実行履歴**や**実行プラン**を保存できる機能として SQL Server 2016 から提供されましたが、SQL Server 2017 では**クエリの待機**（**Wait**）に関する情報も記録できるようになりました。**待機**（Wait）は、クエリを実行するときに発生した内部的なリソースに関する待ち（待機）のことで、CPU 待ちや、ディスクへの書き込み待ち、ロック待ち、ラッチ待ち、並列クエリの場合の集約待ちなど、さまざまな種類の待機があります。

クエリ ストアでクエリの待機を参照している様子



待機（Wait）を利用すれば、どのリソースが原因でクエリの実行時間がかかっているのかを特定することができるので、遅くなったクエリや、高負荷になっているクエリの原因（ボトルネック）を調べるのに役立ちます。クエリ ストアには、過去に実行したクエリの実行履歴が格納されているので、後から過去に振り返って、待機を調査できるので大変便利です。

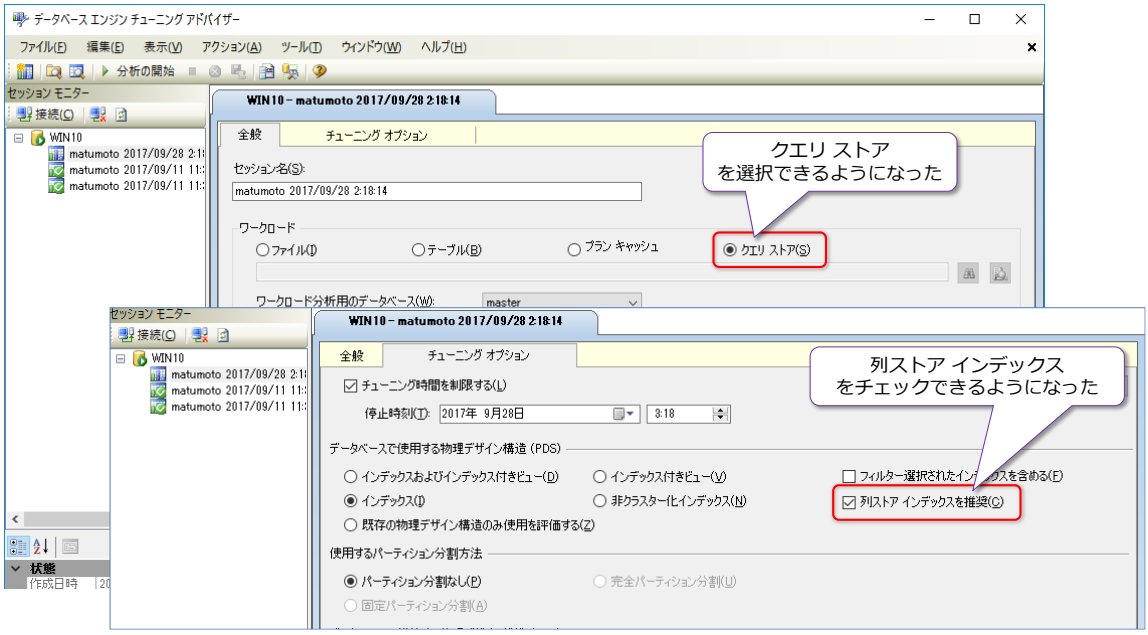
DTA（チューニング アドバイザー）の強化

DTA（**データベース エンジン チューニング アドバイザー**）は、データベースに対するクエリを自動分析して、作成した方が良いインデックスや、削除した方が良いインデックス、お勧めのデータパーティション構造などを提案してくれる機能です。このツールの歴史は非常に古く、SQL Server 7.0 のとき（18 年前！）から提供されています（SQL Server 7.0 のときはインデックス チューニング ウィザードと呼ばれていて、インデックスに関する提案のみが可能でした）。このツールは、

その後 DTA と呼ばれるようになって、インデックス付きビューや、データパーティション構造へのお勧めに対応したり、データソースとしてプラン キャッシュを利用できりょうになったりと、バージョンが上がるたびに進化してきました。

今回の DTA は、Management Studio のバージョン 16.4 以降で強化されているので、SQL Server 2017 だけでなく、SQL Server 2016 でも利用することができますが、**クエリ ストア**をデータソース（チューニング対象のクエリが格納されている場所）に利用できるようになったり、**列ストア インデックス**に対応（列ストア インデックスを付けた方が良いのかどうかの推奨を）できるようになりました。

DTA（データベース エンジン チューニング アドバイザー）でチューニングをしている様子



列ストア インデックスの強化

SQL Server 2017 では、**列ストア インデックス**（カラムストア インデックス）も強化されています。強化ポイントは、次のとおりです。

- バッチ モードにおける**性能向上**（前掲の Adaptive Query Processing）
- 非クラスター化列ストア インデックスでの "**オンライン**" 再構築のサポート
- クラスター化列ストア インデックスで **LOB** データのサポート
  (varchar(max) や nvarchar(max)、varbinary(max) 列のサポート)

列ストア インデックスは、性能を維持するためには、定期的なインデックスの再構築が欠かせませんが、SQL Server 2016 までは "**オフライン**" での再構築しかできませんでした。SQL Server 2017 からは、非クラスター化列ストア インデックス (Non Clustered Columnstore Index) で、オンライン再構築がサポートされるようになりました。

インメモリ OLTP の強化

SQL Server 2017 では、**インメモリ OLTP** も強化されています。強化ポイントは、次のとおりです。

- メモリ最適化テーブルでの 8 個のインデックス上限の廃止
- メモリ最適化テーブルでの**計算列**のサポート（計算列に対するインデックスの作成も可能）
- **sp_rename** のサポート（メモリ最適化テーブルやネイティブ コンパイル ストアド プロシージャに対して、名称変更が可能に）
- ネイティブ コンパイル モジュールで以下のサポートを追加
  CASE 式、**JSON** 関数、**CROSS APPLY** 演算子、**TOP (N) WITH TIES** のサポート
- メモリ最適化テーブルにおけるトランザクション ログの Redo 処理が**並列対応**（これによって、復旧時間が短縮するので、AlwaysOn 可用性グループ構成でのスループットの向上に繋がる）
- メモリ最適化テーブルでの **bw-tree** 非クラスター化インデックスのビルド性能の向上
- **sp_spaceused** のサポート
- メモリ最適化ファイル グループを Azure Storage 上に保存可能に

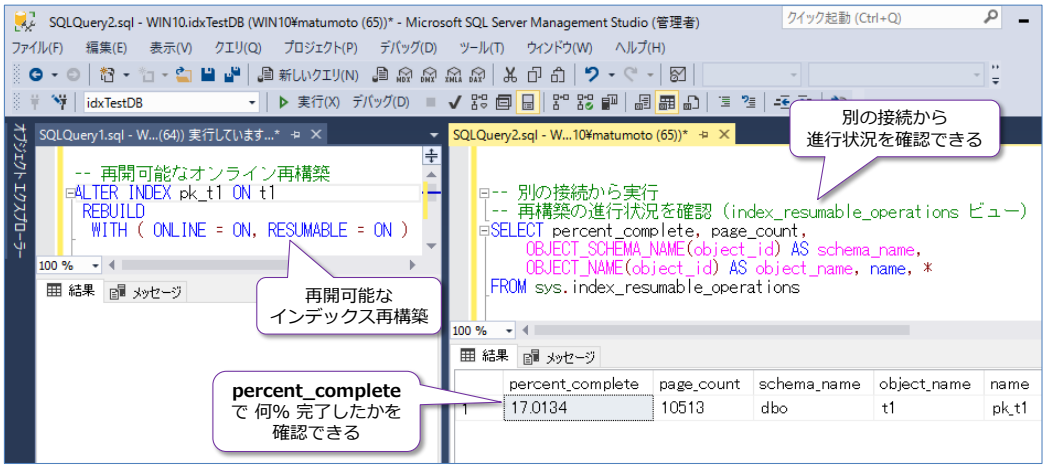
インメモリ OLTP は、完全にインメモリで動作させることで、**OLTP の性能**を向上させることができる機能として SQL Server 2014 から提供されましたが、SQL Server 2016 では OLTP だけでなく、**分析ワークロード**にも対応できるようにクラスター化列ストア インデックスをサポートしました（**Operational Analytics** の実現）。また、SQL Server 2016 では、各種の制限の緩和（ALTER のサポートや、インデックスでの BIN 以外の照合順序のサポートなど）によって、非常に使いやすくなりました（既存のディスク ベースのテーブルを、インメモリ OLTP に移行しやすくなりました）。

そして、SQL Server 2017 からは、さらに制限が緩和されて、インメモリ OLTP におけるテーブルである「**メモリ最適化テーブル**」に作成できるインデックスの上限が 8 個だったものを撤廃、メモリ最適化テーブルに対して**計算列**を作成することもできるようになりました。このように インメモリ OLTP は、SQL Server 2017 でさらに使いやすくなって、ディスク ベースのテーブルと遜色なく使えるようになってきているので、**既存のテーブルからの移行**が非常にしやすくなりました。

再開可能なオンライン インデックス再構築（Resumable Index Rebuild）

SQL Server 2017 では、インデックスの**オンライン再構築**の際に、途中で停止をしたり、一時停止をして、操作を停止時から再開するといったことができるようになりました。また、途中でログの切り捨てもできるので、ディスク容量が足りなくなった場合などでも、再構築を継続できるようになりました。

再開可能なインデックスの再構築を利用している様子



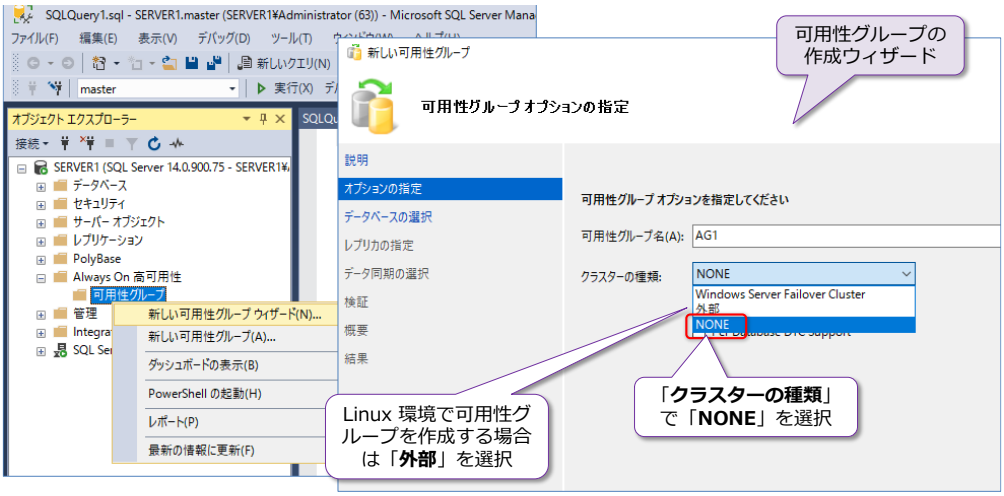
ALTER INDEX ステートメントの **WITH** オプションで **RESUMABLE=ON** を指定することで、再開（**RESUME**）が可能なインデックスの再構築を行えるようになって、別の接続からは、再構築の一時停止（**PAUSE**）や、進行状況の確認、**ログの切り捨て**（ディスク容量不足への対策）などを行えるようになりました。

AlwaysOn 可用性グループの強化

SQL Server 2017 では、**AlwaysOn 可用性グループ**も強化されています。強化ポイントは、次のとおりです。

- DTC（分散トランザクション）のサポート
- Linux 環境での可用性グループのサポート
  （Windows と Linux をまたがった可用性グループも構成可能）
- クラスター レスの可用性グループのサポート
  （分析／レポーティング用途などの読み取り専用ワークロードをスケール アウト）

クラスター レス（クラスター不要）の可用性グループを構成すれば、分析（Analytics）用途や、レポーティングのための読み取り専用のワークロードを**スケール アウト**するために利用することができます。



その他の新機能

SQL Server 2017 には、その他にも新機能がたくさん提供されています。その主なものは、次のとおりです。

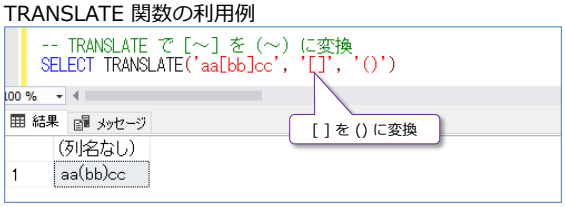
- Transact-SQL の強化
- セットアップ時の変更点（tempdb ファイルの設定）
- スマート バックアップ
- 新しい DMV（動的管理ビュー）
- テンポラル テーブルの強化
- XE プロファイラー／グラフィカル実行プランの検索機能の提供
- SQL CLR のセキュリティ強化

Transact-SQL の強化

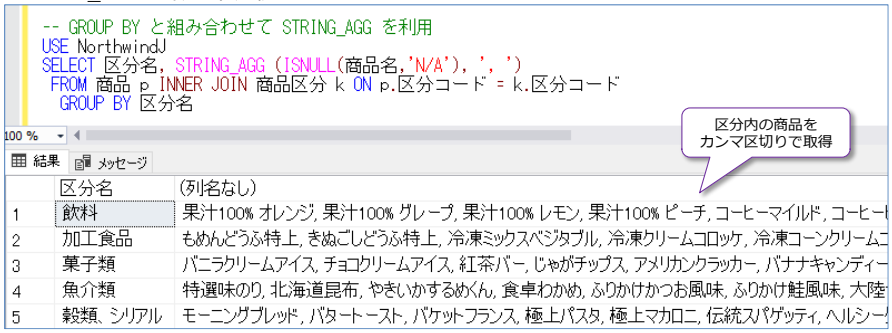
SQL Server 2017 では、Transact-SQL ステートメントも強化されています（以下）。

- 新しい関数の追加
  TRIM、**CONCAT_WS**、**TRANSLATE**、**STRING_AGG（WITHIN GROUP）**

TRANSLATE 関数の利用例



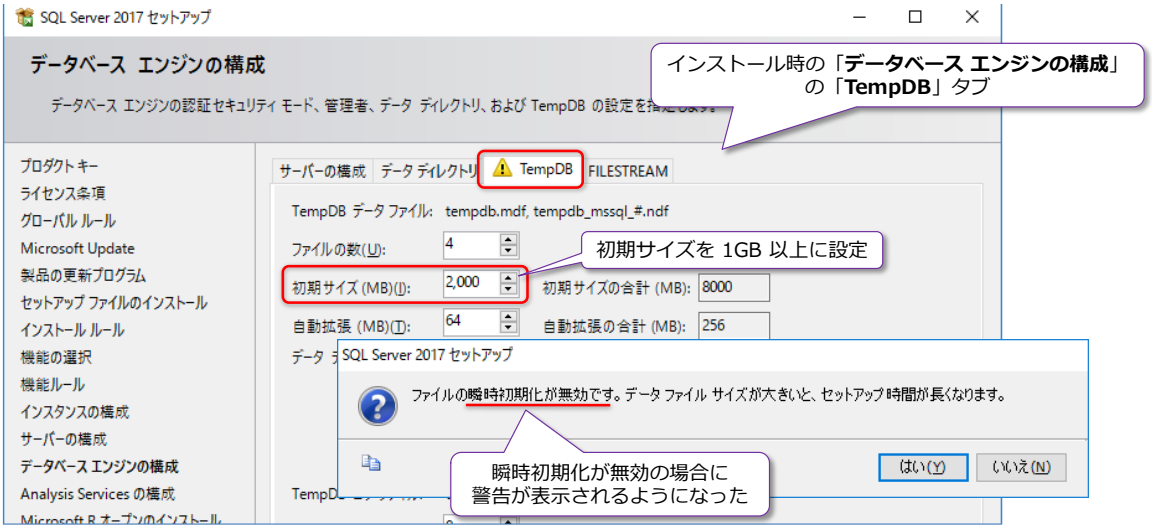
STRING_AGG 関数の利用例



- **SELECT INTO** での**ファイル グループの指定**
- 一括操作での **FORMAT = CSV** による CSV ファイルの取り込みのサポート（BULK INSERT や、OPENROWSET(BULK...) で利用可能）
- **IDENTITY_CACHE** オプションのサポート（IDENTITY 値のキャッシュをオフにすることがデータベース単位で可能に）
- 日本語向けの**新しい照合順序**の追加（Japanese_Bushu_Kakusu_140 と Japanese_XJIS_140 照合順序の追加、それぞれ _VSS（Variation-selector-sensitive）対応

セットアップ時の変更点（tempdb ファイルの設定）

SQL Server 2017 では、インストール時の **tempdb** データベースのファイルに関する設定が変更されました。ファイルあたり **256 GB** (262,144 MB) までの初期ファイル サイズを指定できるようになったり、ファイル サイズが **1 GB** より大きい値に設定されている場合に、**瞬時初期化**が有効になっていないと、警告が表示されるようになったりしました。



スマート バックアップ

SQL Server 2017 では、バックアップ機能も強化されて、効率的にバックアップを取得（適切な頻度でログ バックアップを実行したり、差分バックアップと完全バックアップを適切に使い分けたり）するなど、スマートなバックアップを支援する機能が追加されています。

- 差分バックアップを効率良く実施するために、dm_db_file_space_usage ビューに **modified_extent_page_count** 列が追加（変更があったページ数を確認可能）

スマート バックアップの利用例（差分ページ数に応じて差分／完全バックアップを変更）

```
-- 差分ページ数の割合を取得
DECLARE @p float
SELECT @p = modified_extent_page_count * 1.0 / allocated_extent_page_count
  FROM sys.dm_db_file_space_usage

-- 割合が 80% 以上なら完全バックアップ、それ以下なら差分を実施
IF @p >= 0.8
  BEGIN
    -- 完全バックアップ
    BACKUP DATABASE idxTestDB
     TO DISK = 'NUL'
  END
ELSE
  BEGIN
    -- 差分バックアップ
    BACKUP DATABASE idxTestDB
     TO DISK = 'NUL'
      WITH DIFFERENTIAL
  END
```

差分ページ数の割合が 80% 以上なら完全バックアップを実行、そうでないなら差分バックアップを実行

- ログ バックアップを効率良く実施するために、**dm_db_log_stats** ビューの提供（VLF: 仮想ログ ファイルの統計情報を確認可能）

新しい DMV（動的管理ビュー）

SQL Server 2017 では、**DMV**（動的管理ビュー：Dynamic Management View）も強化されています。その主なものは,次のとおりです。

- sys.**dm_os_host_info** の追加
  Windows／Linux の両方について、OS（オペレーティング システム）情報を参照

Ubuntu の場合

```
SELECT * FROM sys.dm_os_host_info
```

100 %

結果 メッセージ

	host_platform	host_distribution	host_release	host_service_pack_level	host_sku	os_language_version
1	Linux	Ubuntu	16.04		NULL	1041

- sys.**dm_os_sys_info** に 3 つの新しい列が追加
  socket_count、cores_per_socket、numa_node_count

CPU のスケット数や NUMA ノード数の取得

```
-- CPU のソケット／NUMA に関する情報を取得
SELECT socket_count, cores_per_socket, numa_node_count, *
FROM sys.dm_os_sys_info
```

100 %

結果 メッセージ

	socket_count	cores_per_socket	numa_node_count	cpu_ticks	ms_ticks	cpu_count	hyperthread_ratio
1	2	1	1	2677546108806964	723461173	2	1

- sys.**dm_db_stats_histogram** の追加（SQL Server 2016 SP1 CU2～）
  統計情報の確認が可能。従来の DBCC SHOWSTATISTICS に相当

- sys.**dm_tran_version_store_space_usage** の追加
  データベースごとのバージョン ストア使用量を確認可能

- sys.**dm_exec_query_statistics_xml** の追加（SQL Server 2016 SP1 CU2～）
  実行中のクエリの XML 形式の実行プランを確認可能

現在実行中のクエリの XML 形式の実行プランを取得

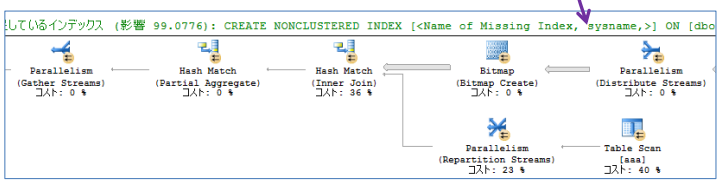
```
-- 現在実行中のクエリの XML プランを取得
SELECT x.* FROM sys.dm_exec_requests
 CROSS APPLY sys.dm_exec_query_statistics_xml(session_id) x
```

実行中のクエリの XML 形式の実行プランを取得

100 %

結果 メッセージ

	session_id	request_id	sql_handle	plan_handle	query_plan
1	56	0	0x02000000EB210F0D5DDC...	0x06001000EB210F0D001C1D...	<ShowPlanXML xmlns="http://scher
2	61	0	0x0200000070CD6228255F5...	0x0600100070CD622830F9F1A...	<ShowPlanXML xmlns="http://scher
3	67	0	0x03000900F01AB84AF130A...	0x05000900F01AB84A30BC62...	<ShowPlanXML xmlns="http://scher



テンポラル テーブルの強化

テンポラル テーブル（Temporal Tables）は、テーブルに対する過去の更新履歴を、自動的に履歴テーブルに保存できる機能として SQL Server 2016 から提供されました。SQL Server 2017 では、**テンポラル テーブル**に、**保持ポリシー**（HISTORY_RETENTION_PERIOD）がサポートされたり、**CASCADE DELETE／CASCADE UPDATE** のサポートが追加されました。

テンポラル テーブルに保持ポリシーを設定している例

```
-- テンポラル テーブルに保持ポリシーを設定
CREATE TABLE Products
(  商品コード int NOT NULL PRIMARY KEY CLUSTERED
  ,商品名 nvarchar(80)
  ,sysstart datetime2 (0) GENERATED ALWAYS AS ROW START
  ,sysend datetime2 (0) GENERATED ALWAYS AS ROW END
  ,PERIOD FOR SYSTEM_TIME (sysstart, sysend)
) WITH
 ( SYSTEM_VERSIONING = ON
     (  HISTORY_TABLE = dbo.ProductsHistory,
        HISTORY_RETENTION_PERIOD = 1 MONTHS ) )
```

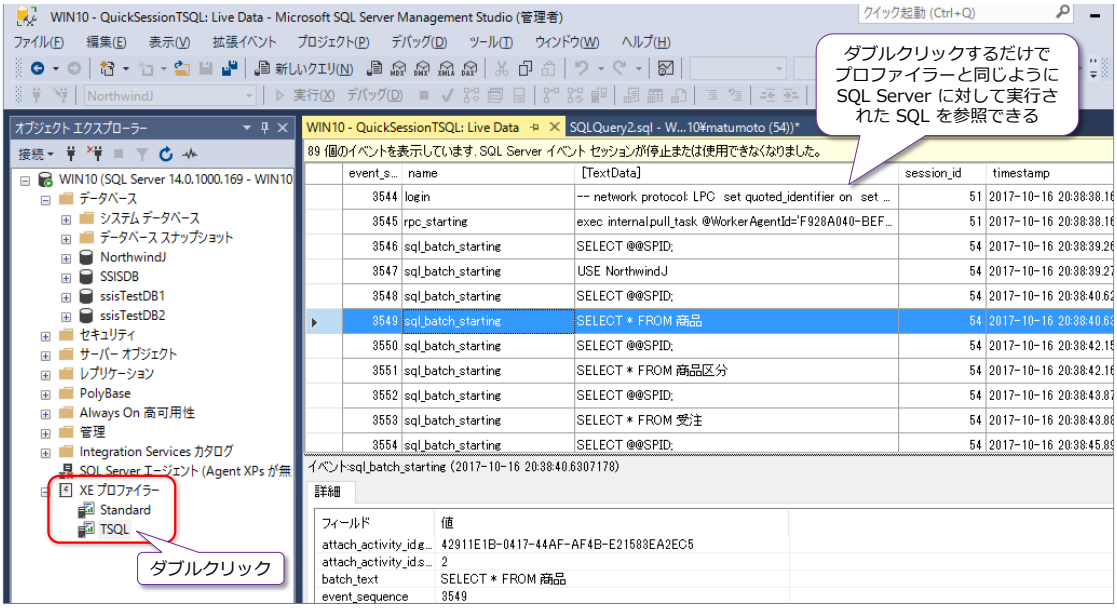
履歴の保持期間を1ヶ月に設定

SQL CLR のセキュリティ強化

SQL Server 2017 では、**SQL CLR**（CLR 統合機能）のセキュリティも強化されています。**clr strict security** が既定でオンになったり、信頼できる CLR を登録するためのホワイト リスト機能の追されたりしました。**sp_add_trusted_assembly** でホワイト リストに追加、**sp_drop_trusted_assembly** でホワイト リストから削除、**sys.trusted_assemblies** ビューでホワイト リストの確認をすることができます。

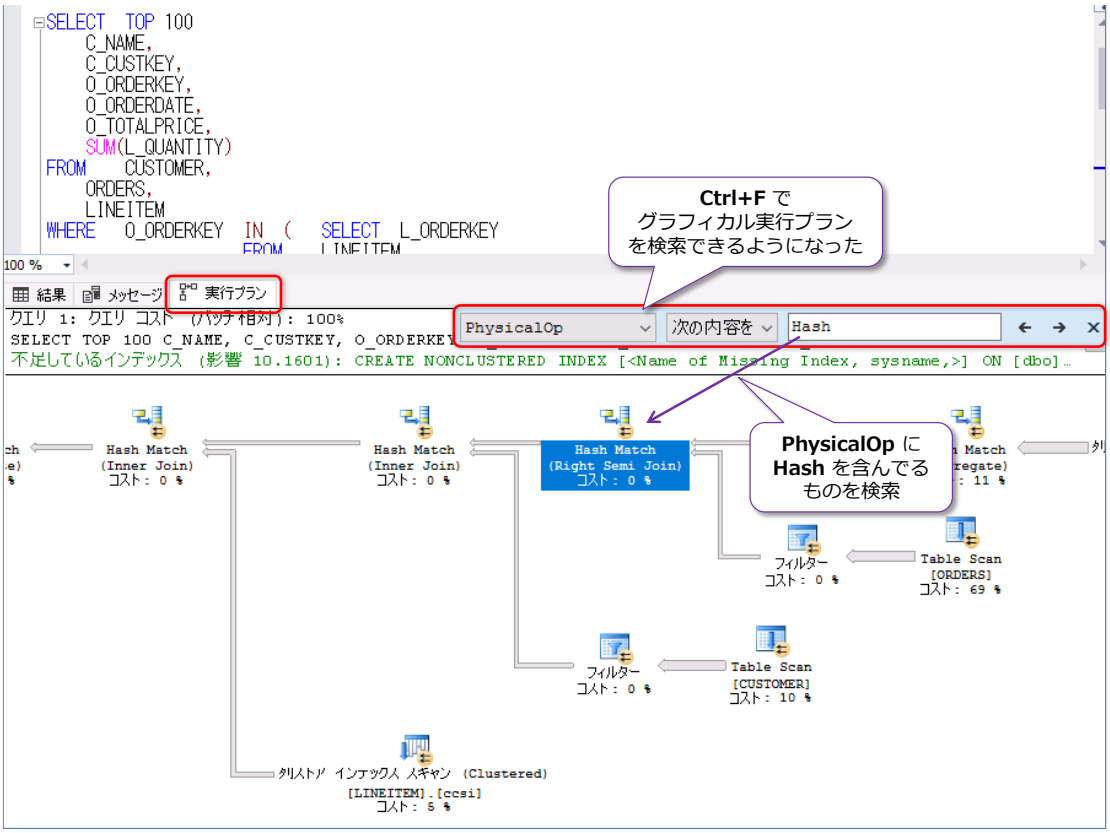
XE プロファイラーの提供

Management Studio のバージョン 17.3 からは、**XE プロファイラー**（XE: 拡張イベントをプロファイラーのように利用できる機能）が追加されました。



グラフィカル実行プランの検索機能の提供

Management Studio のバージョン 17.3 からは、**グラフィカル実行プラン**を**検索**できる機能が追加されました。



これは筆者的には待望の機能で、巨大な実行プランの場合に、オブジェクト名や物理操作（PhysicalOp）で簡単に検索できるようになったので、パフォーマンス チューニングの際に大変役立ちます。

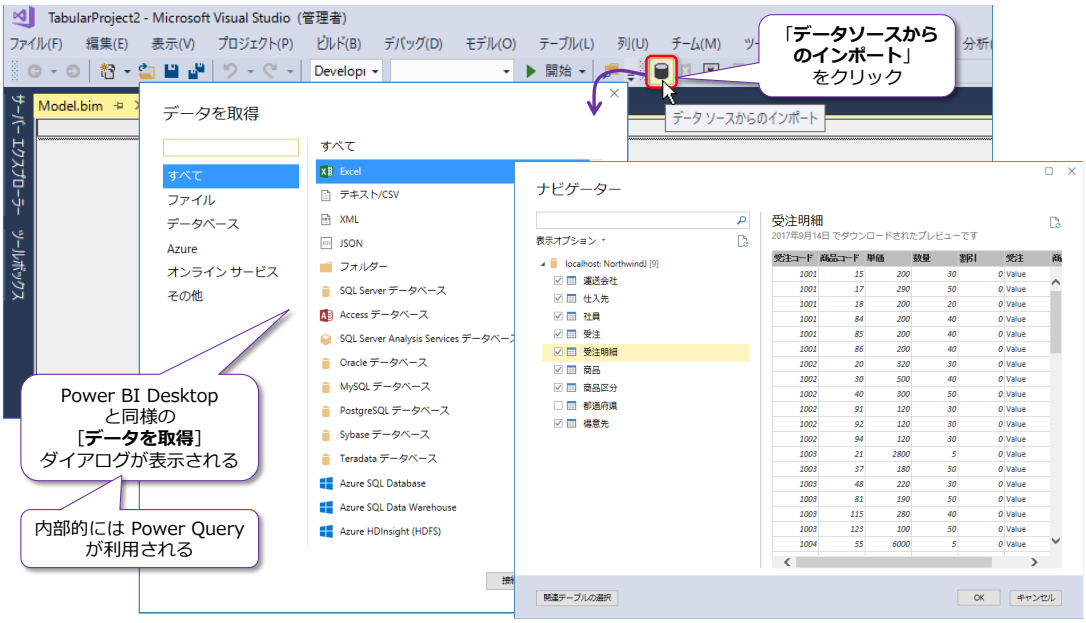
このように、SQL Server 2017 では、既存機能の強化も怠っていません。

BI 機能の強化

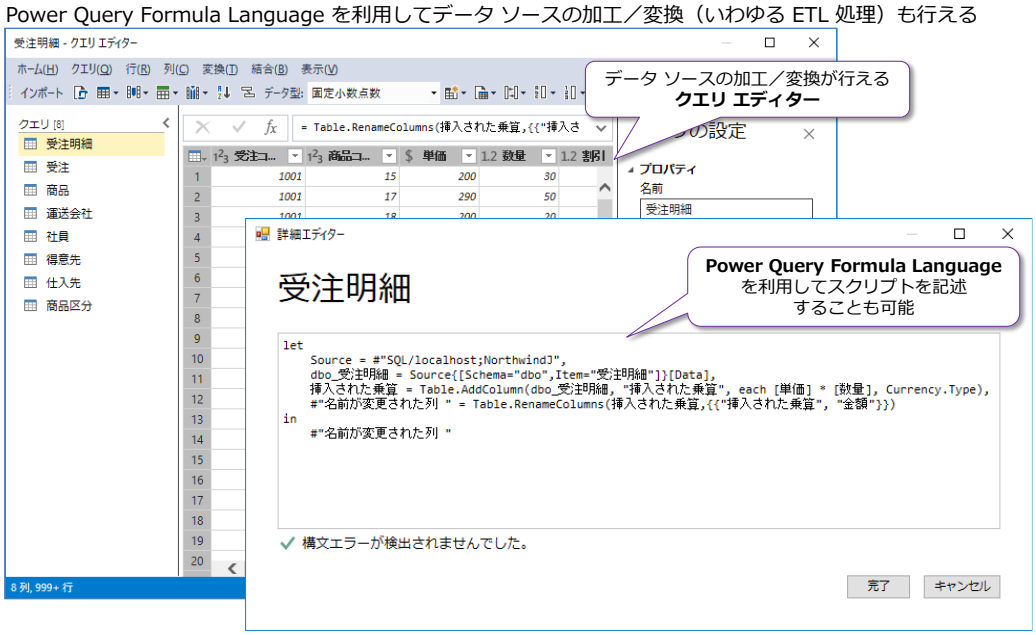
SQL Server 2017 では、BI 機能も進化しています。Analysis Services に Power BI（データ取得部分での Power Query）の統合や、Reporting Services でのレポート コメント、Integration Services のスケールアウト実行など、現場で役立つ機能が提供されています。

Analysis Services の強化（Power Query Formula Language 対応など）

SQL Server 2017 の Analysis Services は、データ ソースの取得が **Power Query** に変更されました（SQL Server 2017 からの新しい互換性レベル **1400** を利用する場合）。これは、**Power BI Desktop** ツールで利用されていたデータ ソースの取得と同様のインターフェースになり、**Power Query Formula Language**（通称 M 言語）にも対応したことになります

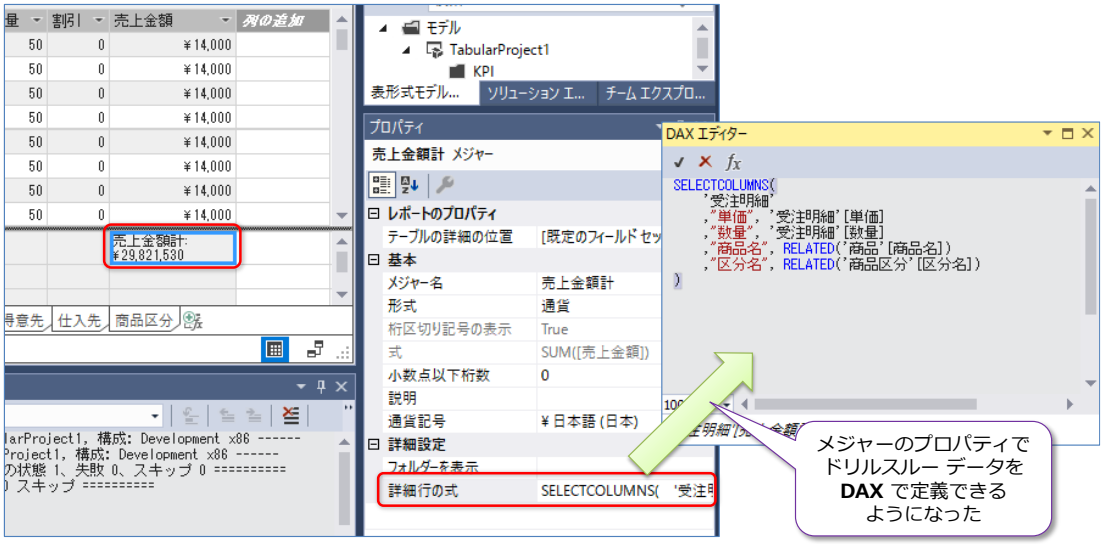


Power Query Formula Language を利用してデータ ソースの加工／変換（いわゆる ETL 処理）も行える



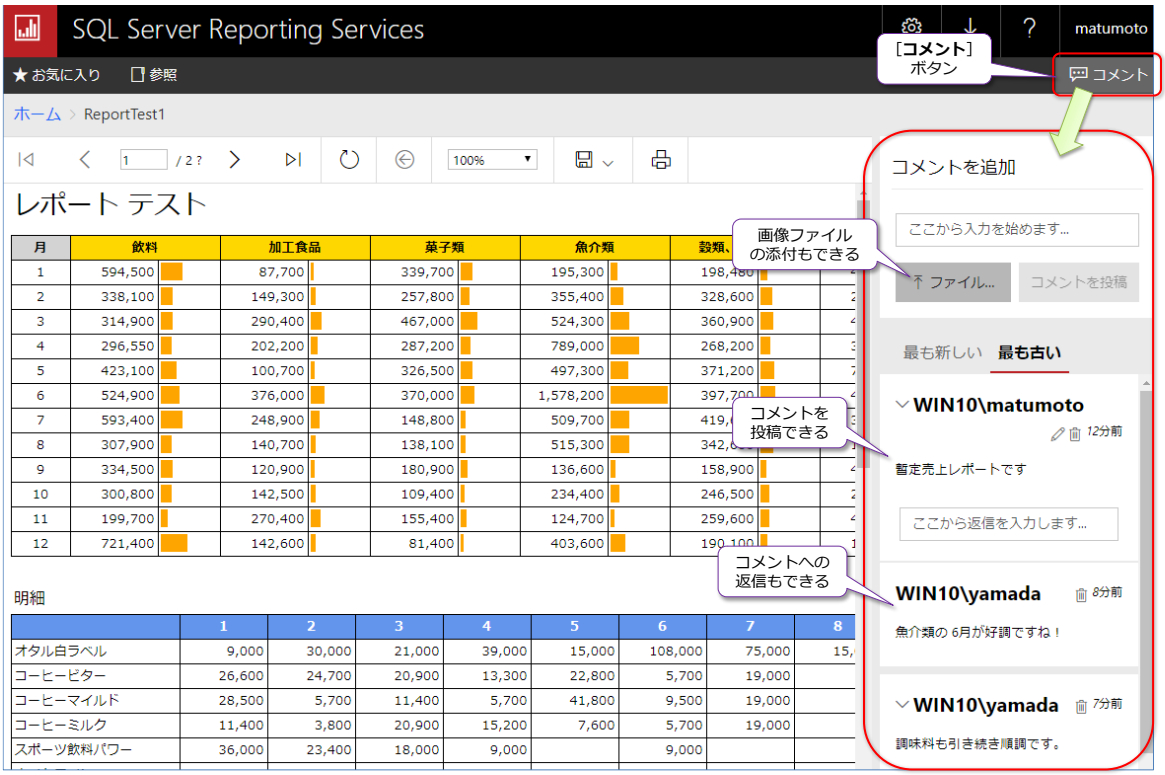
SQL Server 2017 の Analysis Services は、**ドリルスルー データ**に対して、**DAX 式**を記述できるようにもなりました。

ドリルスルー データを DAX 式で指定可能に



Reporting Services の強化（レポート コメントなど）

SQL Server 2017 の Reporting Services では、**レポート コメント**機能の追加や、**DAX クエリ デザイナー**の提供（レポート ビルダーで提供されていた DAX クエリ デザイナーを SSDT でも利用可能に）、**REST API** のサポートなどが強化されています。



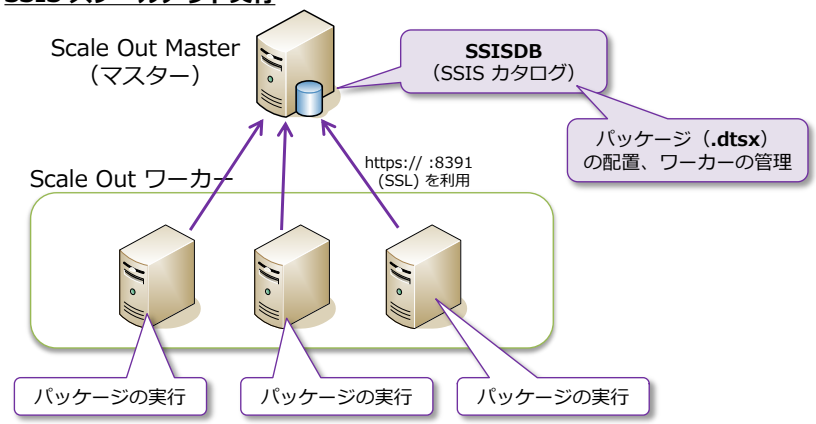
月	飲料	加工食品	菓子類	魚介類
1	594,500	87,700	339,700	195,300
2	338,100	149,300	257,800	355,400
3	314,900	290,400	467,000	524,300
4	296,550	202,200	287,200	789,000
5	423,100	100,700	326,500	497,300
6	524,900	376,000	370,000	1,578,200
7	593,400	248,900	148,800	509,700
8	307,900	140,700	138,100	515,300
9	334,500	120,900	180,900	136,600
10	300,800	142,500	109,400	234,400
11	199,700	270,400	155,400	124,700
12	721,400	142,600	81,400	403,600

	1	2	3	4	5	6	7	8
オタル白ラベル	9,000	30,000	21,000	39,000	15,000	108,000	75,000	15,
コーヒービター	26,600	24,700	20,900	13,300	22,800	5,700	19,000	
コーヒーマイルド	28,500	5,700	11,400	5,700	41,800	9,500	19,000	
コーヒーミルク	11,400	3,800	20,900	15,200	7,600	5,700	19,000	
スポーツ飲料パワー	36,000	23,400	18,000	9,000		9,000		

Integration Services の強化（スケールアウト実行など）

SQL Server 2017 の Integration Services では、パッケージの**スケールアウト実行**や、**Linux** 対応などが強化されています。Integration Services のスケールアウトは、次のように**マスター**と**ワーカー**で構成されます。

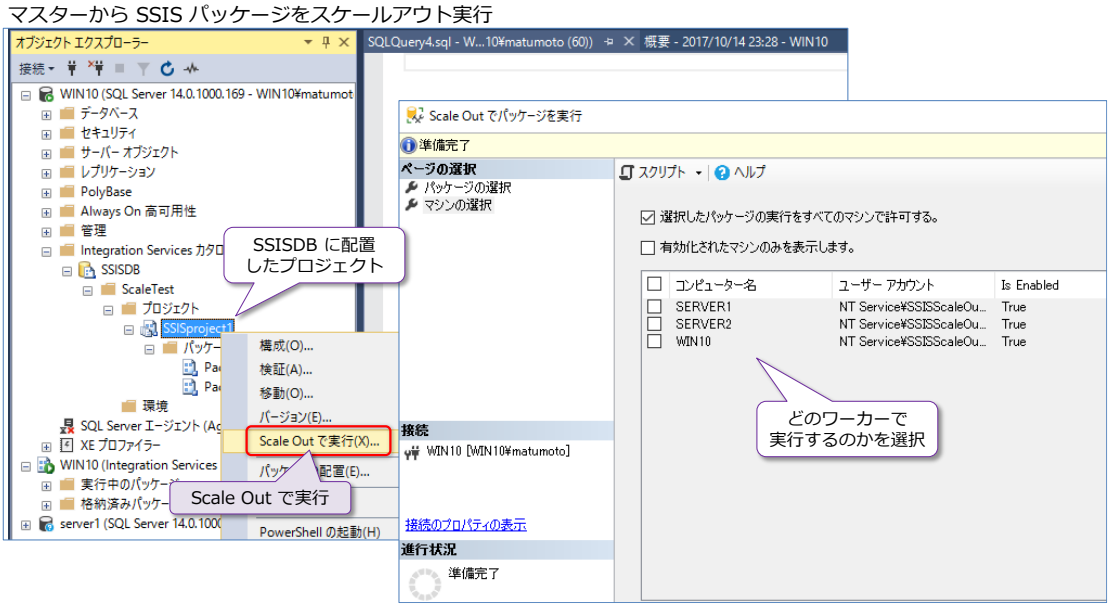
SSIS スケールアウト実行



マスターは、**SSISDB**（SSIS カタログ データベース）を保持して、ここにパッケージ（.dtsx）を配置（Deploy）します。また、パッケージの実行指示（どのワーカーで実行するのか）や、実行の進行状況の確認といったワーカーの管理も行います。

マスターからは、次のようにスケールアウト実行を行うことができます。

マスターから SSIS パッケージをスケールアウト実行



以降では、これらの新機能について、ステップ バイ ステップ形式で画面ショット満載で紹介しているので、ぜひ、皆さんも実際に試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。

STEP 2. SQL Server 2017 on Linux

この STEP では、Linux 環境での SQL Server 2017 に関して、「**Docker での利用方法**」や「**Linux へのインストール方法**」、「**Linux 版で利用できる機能／利用できない機能**」などを説明します。

この STEP では、次のことを学習します。

- ✓ Docker を利用した SQL Server 2017 on Linux
- ✓ Linux 環境への SQL Server 2017 のインストール
- ✓ SQL Server 2017 on Linux でのセキュリティ
- ✓ SQL Server 2017 on Linux で利用できる機能／利用できない機能

2.1 Docker を利用した SQL Server on Linux

SQL Server 2017 は、**Docker イメージ**としても提供されているので、Docker を利用することで、Windows 環境ではもちろんのこと、**Linux** でも、**Mac OS** でも簡単に SQL Server 2017 を動作させることができます。

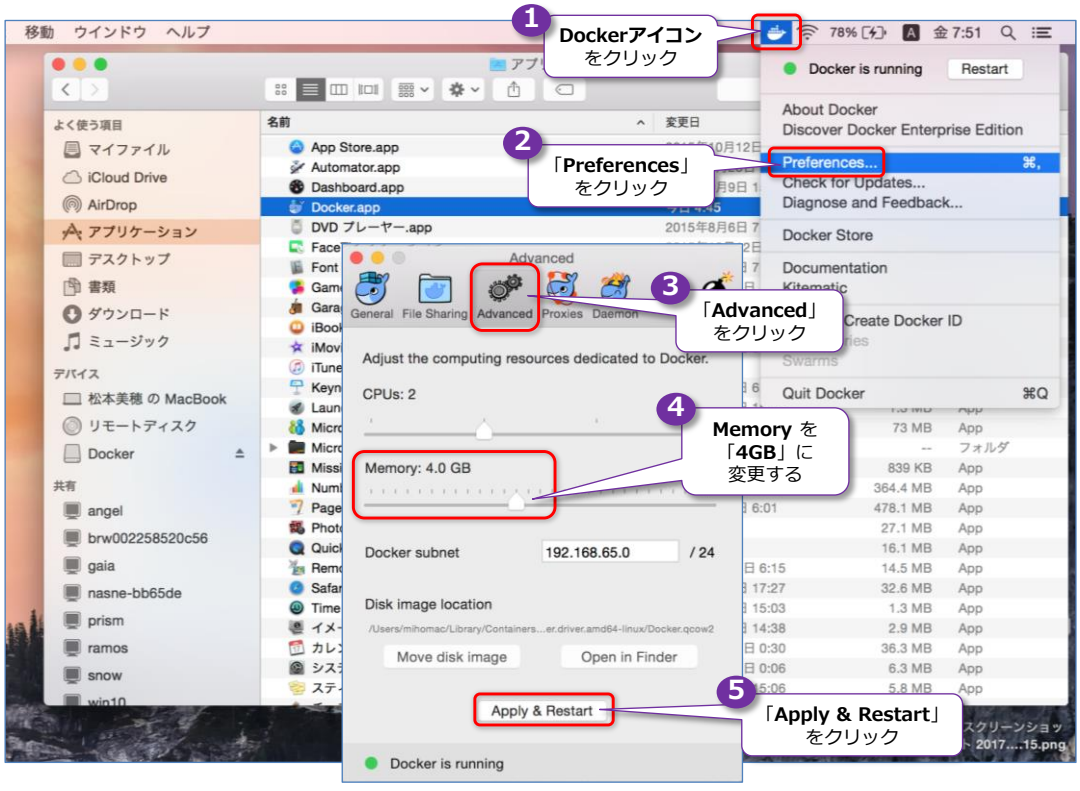
Docker 上で SQL Server 2017 を動作させるための要件は、次のとおりです。

- **Docker Engine** のバージョンに **1.8** 以上を利用する
- 最低 **4GB** のディスク領域
- Docker 用の**メモリを 4GB** 以上に設定する（Mac OS と Windows 版の Docker の場合は、既定値が **2GB** に設定されているので **4GB** に変更する）

Let's Try

それでは、これを試してみましょう。ここでは、**Mac OS X**（**Docker CE for Mac**）を利用した場合の手順を説明しますが、**Linux** 環境でもほとんど同じように試すことができます。Mac OS でも、Linux でも、**Docker コマンドを 2 つ実行**するだけで、SQL Server 2017 を起動することができ、非常に簡単に試すことができるので、ぜひ試してみてください。

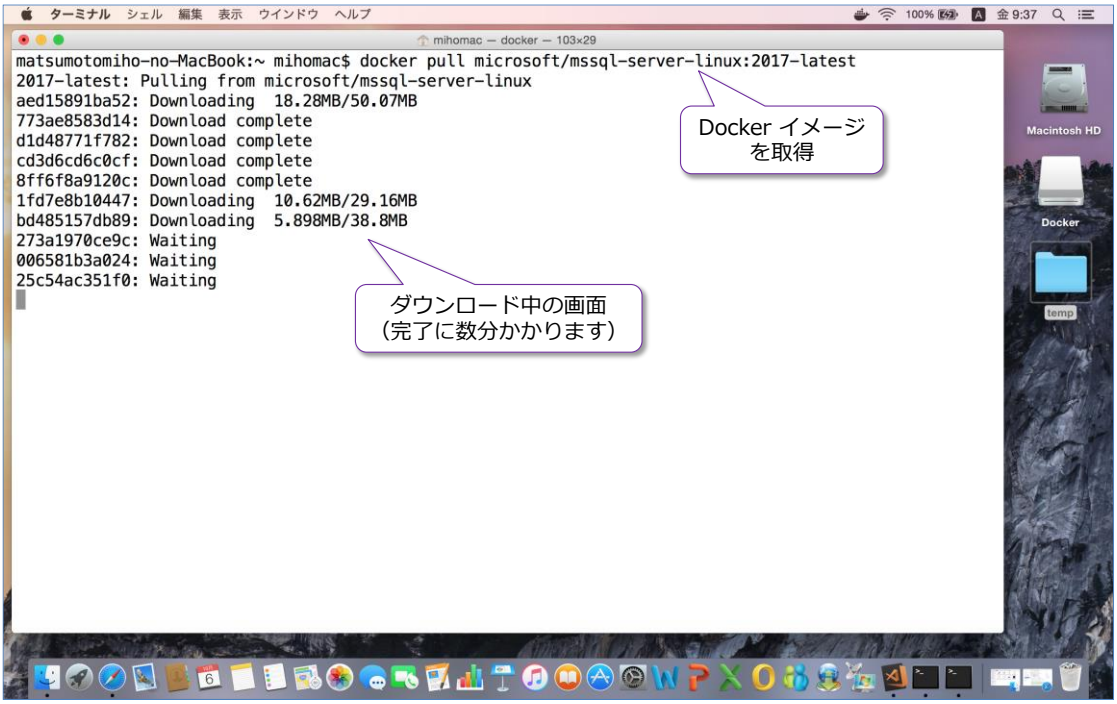
1. **Mac OS X** では、「**Docker CE for Mac**」をインストールすることで、Docker を利用することができますが、既定では Docker 用のメモリが **2GB** に設定されています。まずは、これを **4GB** 以上に変更する必要があります。メモリを変更するには、次のように「**Docker アイコン**」をクリックして、［**References...**］をクリックします。



［**References**］ダイアログが表示されたら、［**Advanced**］ページを開いて、［**Memory**］セクションのスライダーを動かして「**4GB**」に変更します。変更後、［**Apply & Restar**t］ボタンをクリックします。これで Docker が再起動されて、4GB の設定が有効になります。

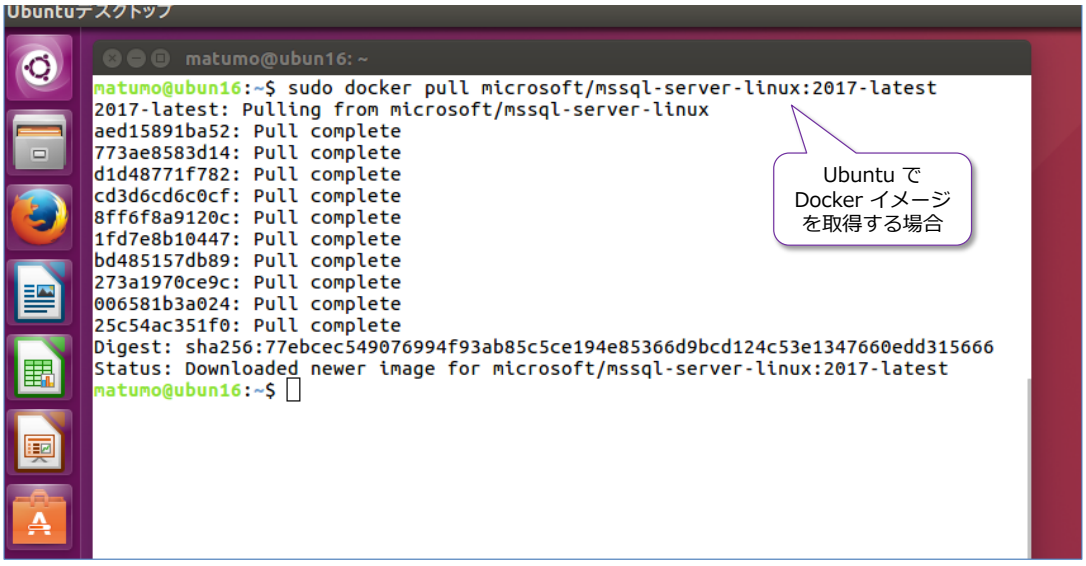
2. Docker の再起動が完了したら、次は、**SQL Server 2017** の **Docker イメージを取得**（pull）します。これを行うには、**ターミナル**を起動して、次のように **Docker** コマンドの **pull** で、イメージ名に「**microsoft/mssql-server-linux:2017-latest**」を指定します。

```
docker pull microsoft/mssql-server-linux:2017-latest
```



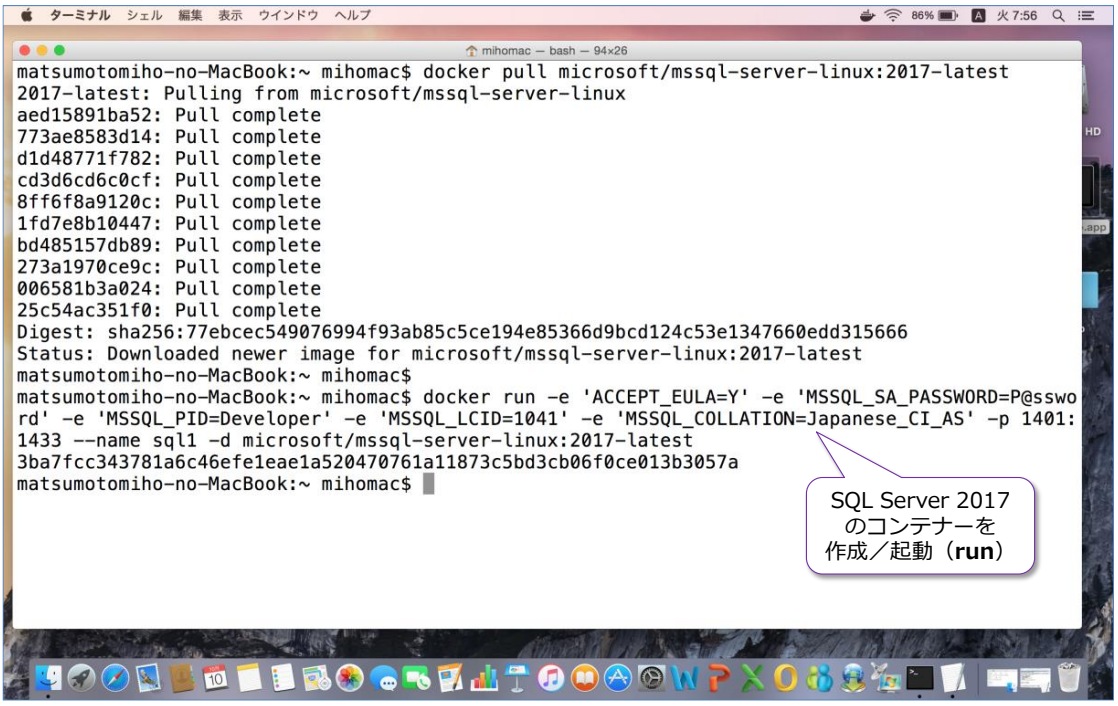
イメージのダウンロードには数分かかるので、完了するまで少し待ちます。

なお、**Linux** を利用して、イメージを取得する場合には、必要に応じて、先頭に「**sudo**」を付けて実行するようにします。



3. SQL Server 2017 のイメージの取得が完了したら、次は**イメージからコンテナーを作成／起動（run）**します。これを行うには、次のように **Docker** コマンドを実行します。以下のコマンドには改行が入っていますが、実際にコマンドを入力するときは、改行せずに 1 行で記述するようにしてください（あるいは、改行を入れる場合には、行末に**バック スラッシュ**を入れるようにしてください）。

```
docker run -e 'ACCEPT_EULA=Y'
 -e 'MSSQL_SA_PASSWORD=任意の複雑なパスワード'
 -e 'MSSQL_PID=Developer'
 -e 'MSSQL_LCID=1041' -e 'MSSQL_COLLATION=Japanese_CI_AS'
 -p 1401:1433 --name sql1
 -d microsoft/mssql-server-linux:2017-latest
```



-e オプションでは、SQL Server 2017 に関する**環境変数**を設定（どのように動作させるのかを指定）しますが、「**-e '変数名=値'**」という形で、**単一引用符**で囲む必要があります（なお、Windows 版の Docker を利用している場合には、単一引用符ではなく、二重引用符で囲むようにします）。

環境変数の「**ACCEPT_EULA=Y**」は、使用許諾契約書（ライセンス条項）の確認になりますが、同意する場合には **Y** を指定します。

「**MSSQL_SA_PASSWORD=**」では、SQL Server の管理者アカウントである「**sa**」に対する任意のパスワードを設定しますが、8 文字以上の複雑なパスワード（大文字、小文字、数字、記号の 4 種類のうち、いずれかの 3 種類の文字を含めるもの）を指定する必要があります。

「**MSSQL_PID=Developer**」では、利用する SQL Server のエディションとして、**Developer**（開発者向けのエディション）を指定しています。有償のエディションを購入している場合には、**Enterprise** や **Standard**、**Web** などを指定することができます。

「**MSSQL_LCID=1041**」では、利用する言語（ロケール ID）を「**日本語**」（**1041**）に設定しています（これは省略可能な環境変数ですが、省略した場合は、英語：**1033** に設定されます）。

「**MSSQL_COLLATION=Japanese_CI_AS**」では、SQL Server の**照合順序**（**Collation**）を設定しますが、「**Japanese_CI_AS**」は日本語版の SQL Server を利用する場合の既定値になっています（これも省略可能な環境変数ですが、省略した場合には、英語版の SQL Server の既定値である **SQL_Latin1_General_CP1_CI_AS** に設定されます）。

「**-p 1401:1433**」では、SQL Server のデータベース エンジンで利用する**ポート番号**を設定しますが、**1401:1433** に設定する場合は、コンテナー内の SQL Server は、**1433** ポートでリッスンして、ホスト側のポートは **1401** を利用します。したがって、別のマシンから、コンテナー内の SQL Server にアクセスするには、ホスト側のポート番号を利用するので、**1401** を指定することになります。なお、「**1433:1433**」と指定した場合には、ホスト上の **1433** ポートを利用して、アクセスさせることもできます（なお、**1433** は、SQL Server における既定のポート番号になります）。

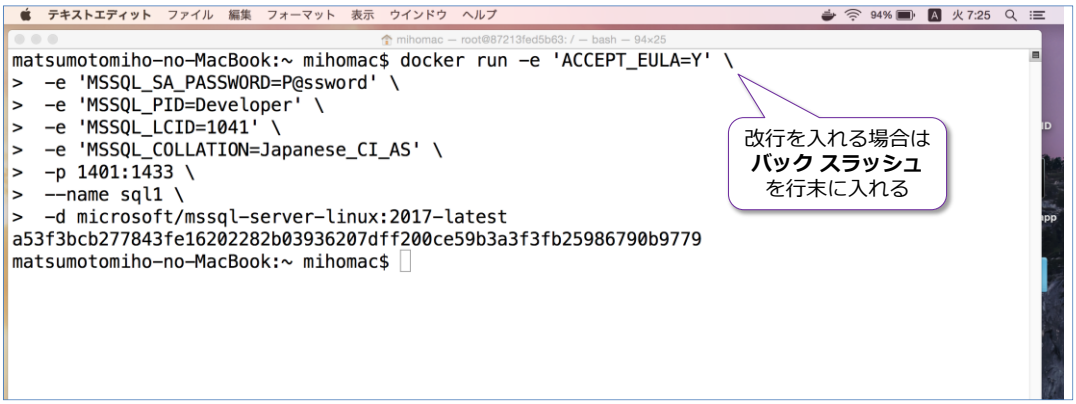
「**--name sql1**」（**name** の前はハイフンが 2 つであることに注意）では、作成するコンテナーに対して任意の名前を付けることができます（ここでは **sql1** という名前を指定していますが、別の名前を指定しても大丈夫です）。

「**-d microsoft/mssql-server-linux:2017-latest**」では、前の手順で取得した SQL Server 2017 の Docker イメージの名前を指定します。

以上で、SQL Server 2017 の起動が完了（コンテナーの作成と起動が完了）です。

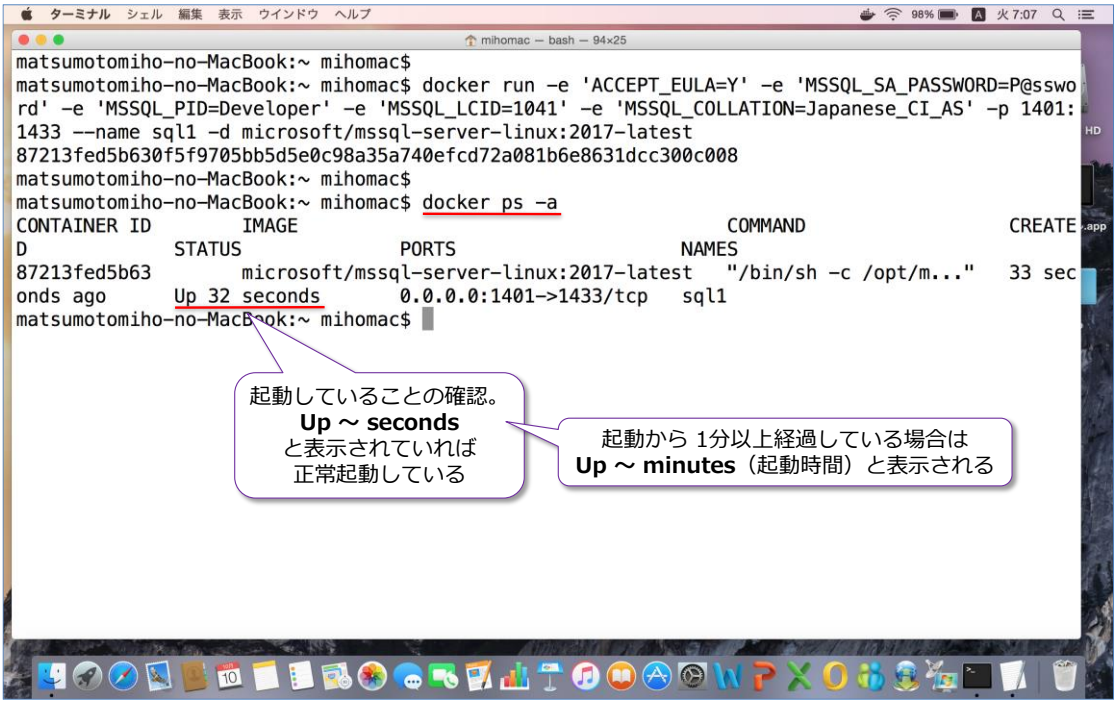
Note： Docker コマンドを複数行で記述する場合

Docker コマンドを、1 行ではなく、改行を入れて複数行で記述したい場合には、次のように**バック スラッシュ**を行末に入れるようします。

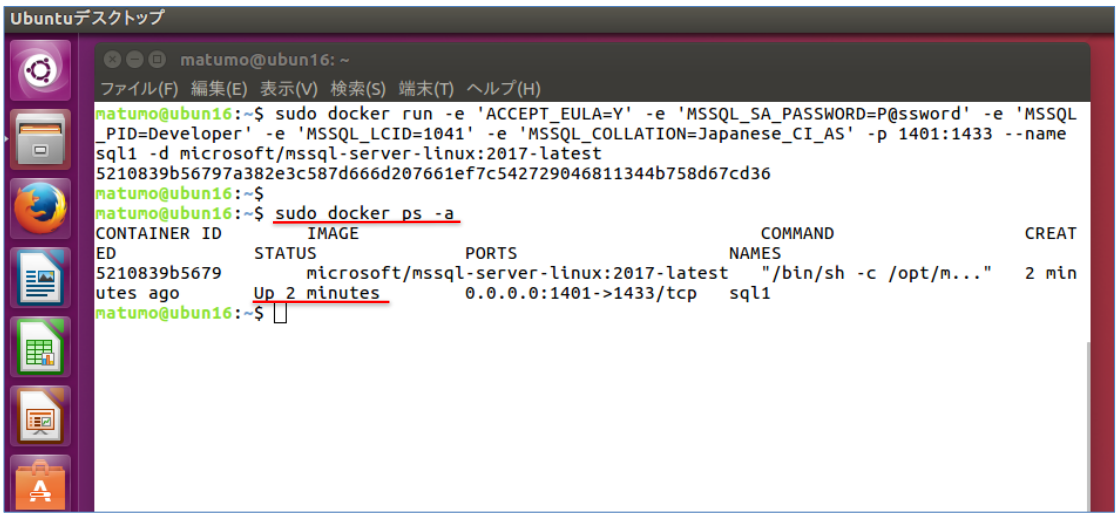


4. 次に、コンテナーが起動していることを確認するために、次のように **Docker** コマンドを実行します。

```
docker ps -a
```



Ubuntu の場合の画面

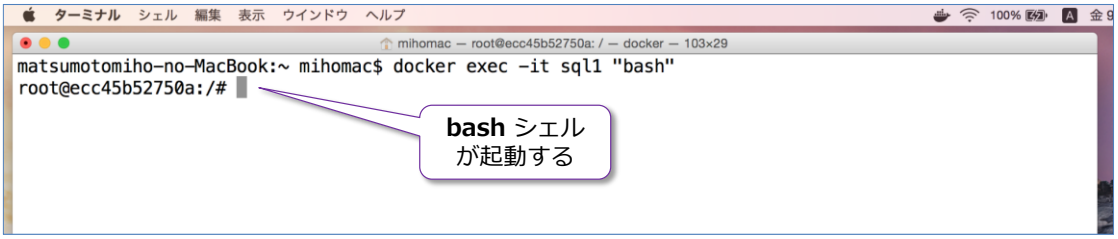


STATUS（状態）に **Up** ～ **seconds** や **Up** ～ **minutes** などと表示されれば、コンテナーが正常に起動しています（**Up** は起動していること、seconds や minutes で起動時間を確認できます）。

SQL Server への接続（sqlcmd ツール）

次に、Docker コンテナー内の **SQL Server 2017 に接続**してみましょう。SQL Server に接続するには、**sqlcmd** ツールを利用しますが、このツールを利用するには、まず、次のように **Docker** コマンドを実行（**exec**）して、コンテナー内の **bash** シェルを起動するようにします。

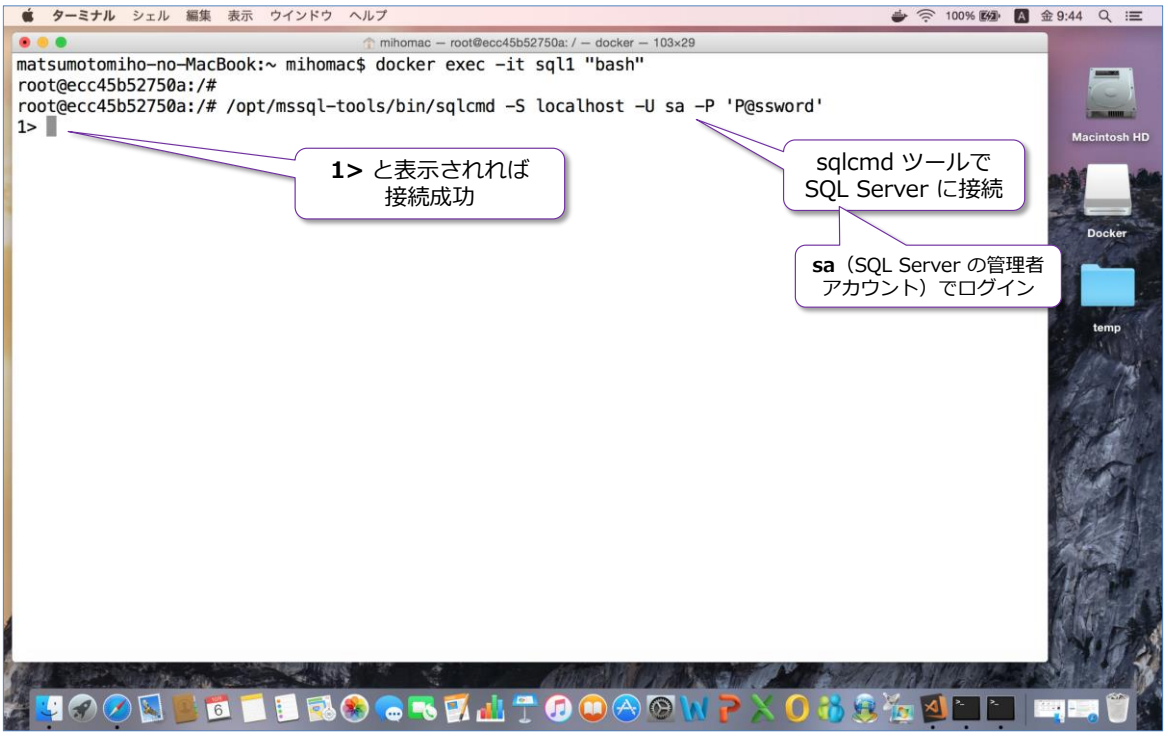
```
docker exec -it sql1 "bash"
```



「**-it**」には、**run** をしたときに「**--name**」で指定したコンテナー名の「**sql1**」を与えて、「"**bash**"」と記述することで、コンテナーの **bash** シェルを起動することができます。

bash シェルが起動したら、次に **sqlcmd** ツール（SQL Server を操作するためのコマンドライン ツール）を利用して、SQL Server に接続してみます。このツールは、コンテナー内の「**/opt/mssql-tools/bin**」ディレクトリに格納されているので、次のように実行できます。

```
/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P 'saに設定したパスワード'
```

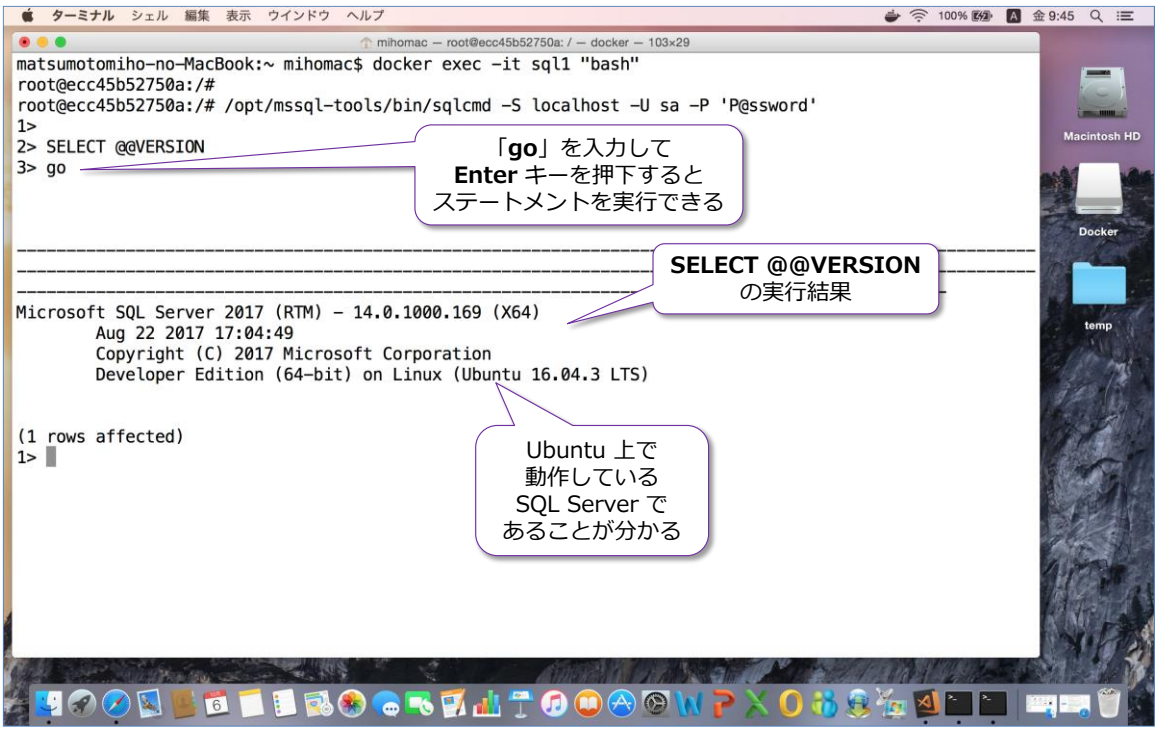


-S オプションでは、接続先となる SQL Server を指定しますが、コンテナー内（自分自身）の SQL Server に接続するので「**localhost**」と指定できます。**-U** オプションでは、接続ユーザーを指定しますが、SQL Server の管理者アカウントである「**sa**」を指定して、**-P** オプションでは、前の手順で Docker run したときに設定した **sa** のパスワードを入力します。

接続が完了すると「**>1**」と表示されて、任意の Transact-SQL ステートメントが実行できるようになります。

次に、SQL Server のバージョンを取得することができる「**@@VERSION**」関数を利用してみましょう。次のように **SELECT** ステートメントを記述して、**Enter** キーで改行し、次に「**go**」を付けて **Enter** キーを押下します。

```
SELECT @@VERSION
go
```



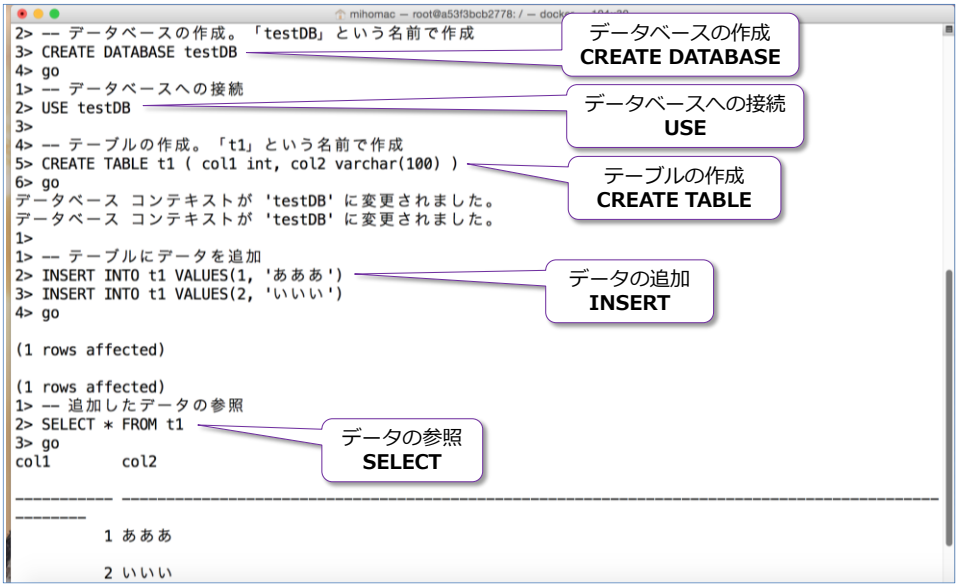
sqlcmd ツールでは、「**go**」を記述することで、そこまでに記述した Transact-SQL ステートメントを実行することができます。

データベースやテーブルの作成例

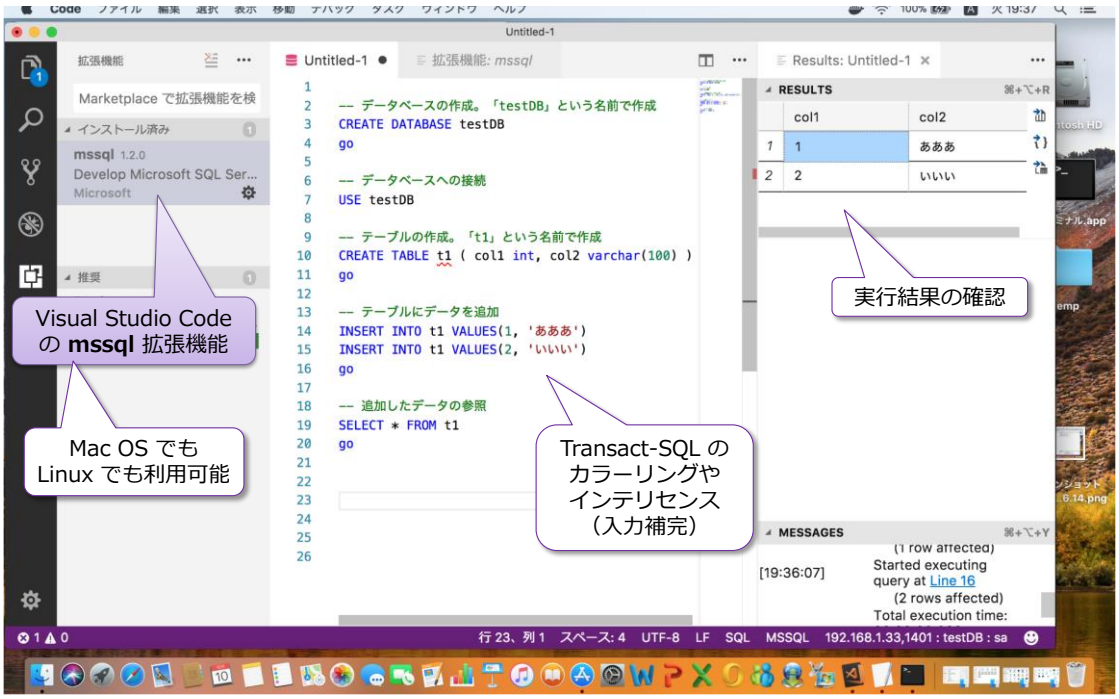
SQL Server 上にデータベースを作成（**CREATE DATABASE**）したり、その中にテーブルを作成（**CREATE TABLE**）するには、次のように記述します。

```
-- データベースの作成。「testDB」という名前で作成
CREATE DATABASE testDB
go
-- データベースへの接続。SQL Server では USE でデータベースに接続する
USE testDB

-- テーブルの作成。「t1」という名前で作成
CREATE TABLE t1 ( col1 int, col2 varchar(100) )
go
```

```
-- テーブルにデータを追加
INSERT INTO t1 VALUES(1, 'あああ')
INSERT INTO t1 VALUES(2, 'いいい')
go

-- 追加したデータの参照
SELECT * FROM t1
go
```



このように **sqlcmd** ツールを利用すれば、SQL Server 2017 に対して Transact-SQL ステートメントを実行できるようになります。ただ、このツールは、コマンドライン ツールなので使いづらい部分もあります。そこで、お勧めになるツールが、**Mac OS** や **Linux** でも利用することができる **Visual Studio Code** の **mssql** 拡張機能です。

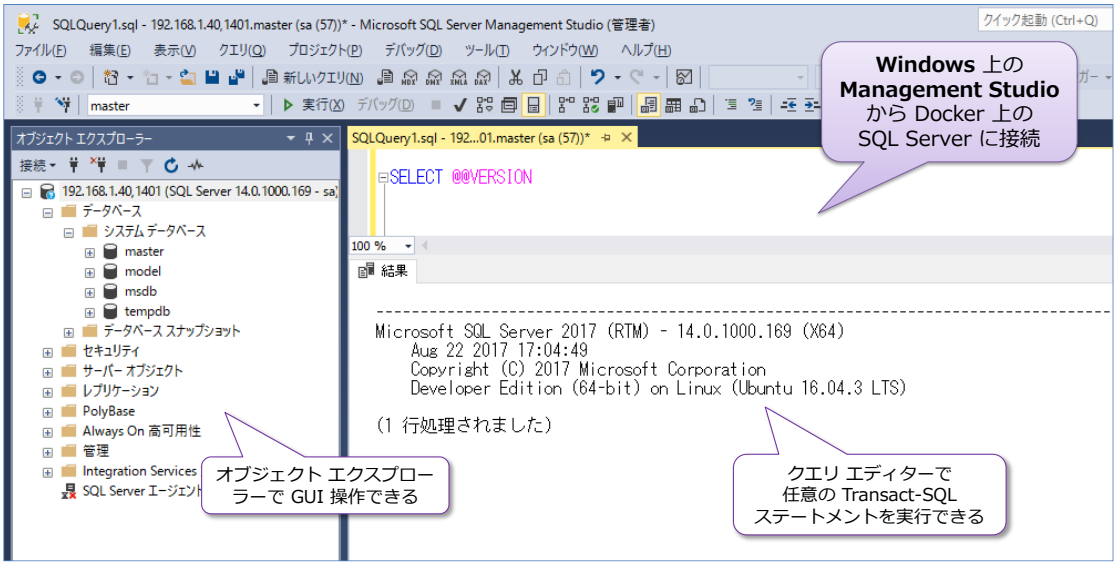


Visual Studio Code は、無償で利用できる開発ツールで、Windows はもちろんのこと、Mac OS でも Linux でも利用することができます。これを利用すれば、さまざまな言語（Java や Python、C#、node.js、ASP.NET など）を利用して、アプリケーション開発ができるので、SQL Server にアクセスするアプリケーションを開発する際にもお勧めのツールになります。

Visual Studio Code には、**mssql** 拡張機能をインストールすることで、上の画面のように、Transact-SQL ステートメントをグラフィカルに実行できるようになります（カラーリングやインテリセンスが効きます）。mssql 拡張機能の利用方法や、Visual Studio Code を利用したアプリケーション開発については、本自習書シリーズの No.2「**SQL Server 2017 on Linux**」編で詳しく説明しているので、こちらもぜひご覧いただければと思います。

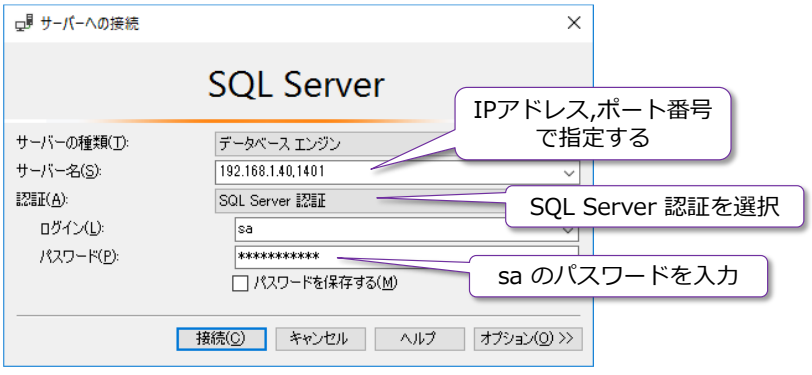
Note： Windows の Management Studio から Docker 上の SQL Server に接続

Docker 上の SQL Server コンテナーには、Windows 上の **Management Studio** ツール（SQL Server の管理ツール）から接続することも可能です。



これを利用すれば、従来ながらの SQL Server の操作と同様、オブジェクト エクスプローラーを利用して GUI で SQL Server を操作することができます。

Management Studio で Docker 上の SQL Server に接続する際には、次のようにポート番号に **1401** を指定するようにします（ホスト側でファイアウォールを有効化している場合には、ファイアウォールで **1401** ポートを解放しておく必要があります）。



「**サーバー名**」に「**IP アドレス,ポート番号**」と入力して、ホストの IP アドレスに続けて**カンマ**を記述し、それに続けてポート番号「**1401**」を指定します（これは、コンテナーを **run** したときに「**-p 1401:1433**」という形で指定したポート番号です）。

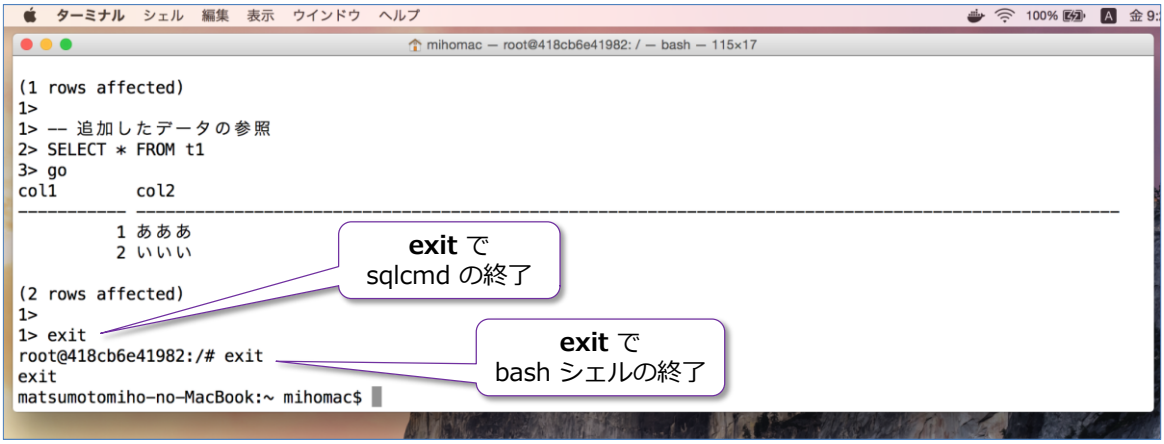
「**認証**」では「**SQL Server 認証**」を選択して、「**ログイン**」に「**sa**」、「**パスワード**」には sa のパスワードを入力すれば、SQL Server に接続することができます。

あとは、Windows 上の SQL Server を操作するのと同じように Docker 上の SQL Server を操作することができます。

sqlcmd の終了、bash シェルの終了 ～exit～

sqlcmd ツールを終了して、bash シェルに戻るには、**exit** と記述します。

```
exit
```



また、コンテナー内の **bash** シェルを終了する場合にも、**exit** と記述します。

コンテナーの停止、削除

コンテナーを停止（**stop**）したい場合には、次のように **Docker** コマンドを実行します。

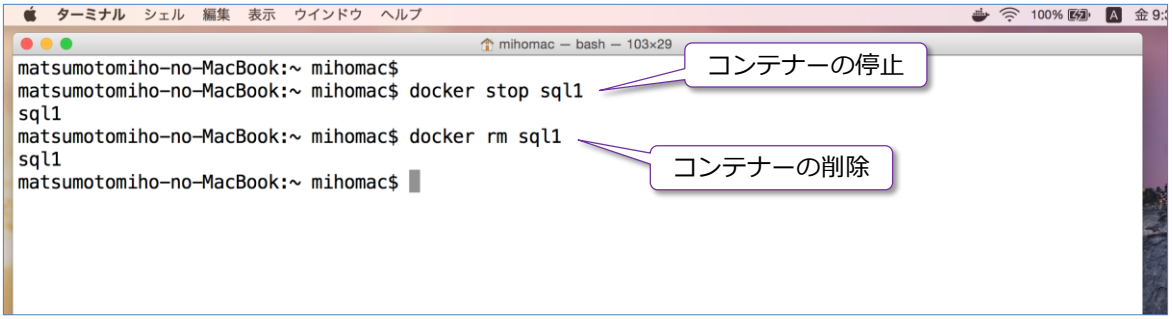
```
docker stop sql1
```

なお、停止したコンテナーを再開（開始）するには、次のように **start** を指定します。

```
docker start sql1
```

コンテナーを削除したい場合には、次のように **rm** を指定しますが、これを実行するには、事前にコンテナーが停止（stop）済みである必要があります。

```
docker rm sql1
```



以上のように、SQL Server 2017 は、**Docker** イメージが提供されるようになったので、**Mac OS** や **Linux** 上でも簡単に試せるようになりました。SQL Server 2017 をインストールすることなく、SQL Server を利用することができるので、開発環境を作成する場合などに大変便利です。

Docker イメージ（コンテナー）に対して行った修正（データベースの作成や、既存のデータベースのリストアなど）は、**commit**（Docker コマンドの commit を実行）をすることで**新しいイメージ**として作成（SQL Server 2017 もデータベースも全て構成された状態のイメージを作成）することができるので、これを Docker Hub や Docker Registory、Azure Container Registory などのイメージの共有が可能なレジストリに配置しておけば、社内の開発メンバーに、同じデータベース環境を簡単に共有することができます（レジストリから pull するだけで、開発に必要なデータベースをすぐに利用することができます）。

Note： イメージを削除したい場合

ダウンロードした SQL Server 2017 イメージを削除したい場合には、次のようにイメージの一覧を取得して、イメージ ID を確認します。

```
docker images
```

「**microsoft/mssql-server-linux:2017-latest**」のイメージ ID を確認したら、次のように **rmi** を指定することで、イメージを削除することができます。

```
docker rmi イメージ ID
```

2.2 Linux 環境への SQL Server 2017 のインストール

次に、Docker ではなく、Linux 環境に直接 SQL Server 2017 をインストールする方法を説明します。SQL Server 2017 がサポートしているプラットフォームは、次のとおりです。

- **Red Hat Enterprise Linux 7.3** または **7.4** Workstation, Server, and Desktop (ファイル システム: XFS または EXT4)
- **SUSE Enterprise Linux Server v12 SP2**（ファイル システム: EXT4）
- **Ubuntu 16.04 LTS**（ファイル システム: EXT4）

インストールのためのハードウェア要件は、次のとおりです。

- メモリ： **3.25GB**
- ディスク領域： **6GB**
- CPU： **x64** と互換性のあるもののみで、**2GHz**、**2 コア**以上

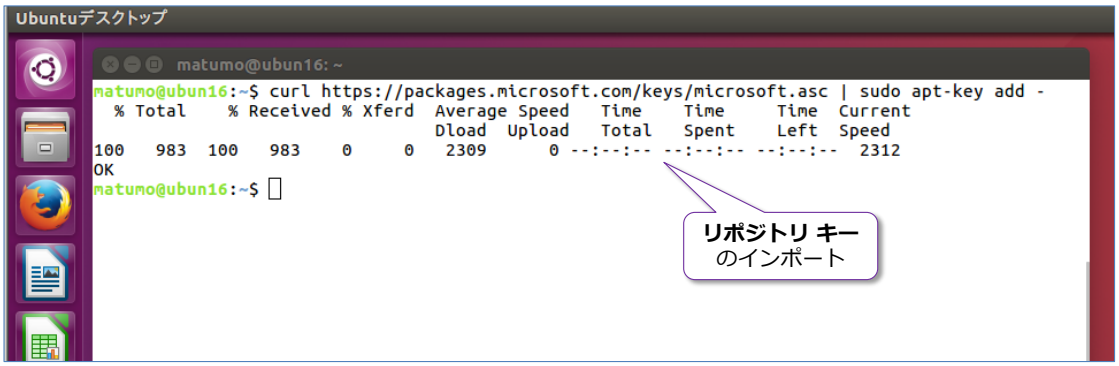
Let's Try

それでは、Linux 環境への SQL Server 2017 のインストールを試してみましょう。ここでは、**Ubuntu 16.04 LTS** 上に SQL Server 2017 をインストールする手順を例に説明しますが、**RHEL**（Red Hat Enterprise Linux）や **SUSE Enterprise Linux** を利用する場合でもほとんど同じようにインストールすることができます（パッケージ マネージャとして apt-get を利用するか、yum を利用するか、zypper を利用するかの違い程度です）。

Ubuntu への SQL Server 2017 のインストールは、**5 個のコマンド**を実行するだけで簡単に完了するので、ぜひ試してみてください。

1. Ubuntu に SQL Server 2017 をインストールするには、まず、次のようにリポジトリ キーをインポートします。

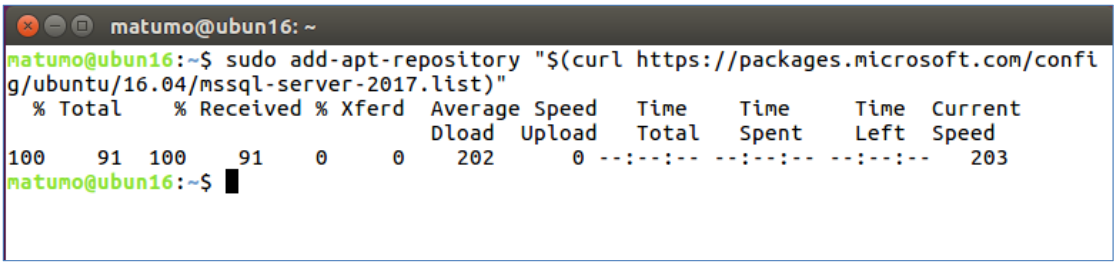
```
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```



2. 次に、リポジトリを登録します（以下のコマンドには改行が入っていますが、実際にコマンド

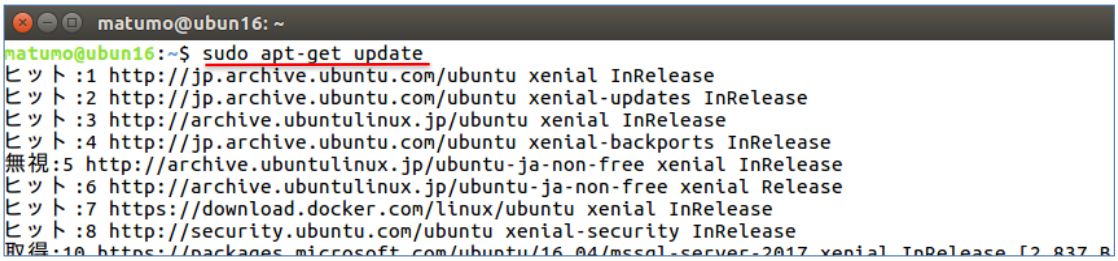
を入力するときは、改行せずに 1 行で記述するようにしてください)。

```
sudo add-apt-repository "$(curl https://packages.microsoft.com/config/ubuntu/16.04/mssql-server-2017.list)"
```



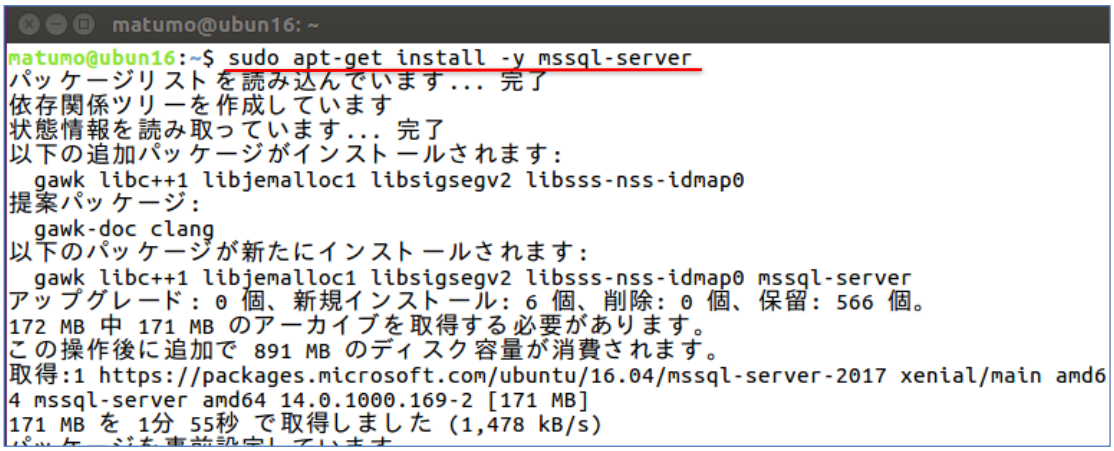
3. 次に、パッケージのアップデートを実行します。

```
sudo apt-get update
```

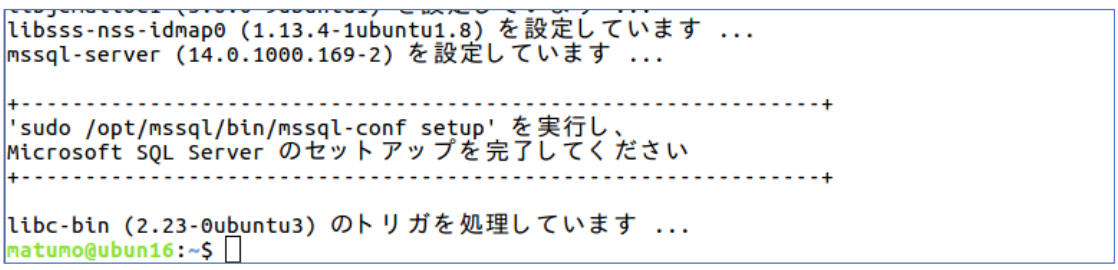


4. パッケージのアップデートが完了したら、次は SQL Server 2017 のインストールです。これを行うには、次のように「**mssql-server**」を指定します。

```
sudo apt-get install -y mssql-server
```

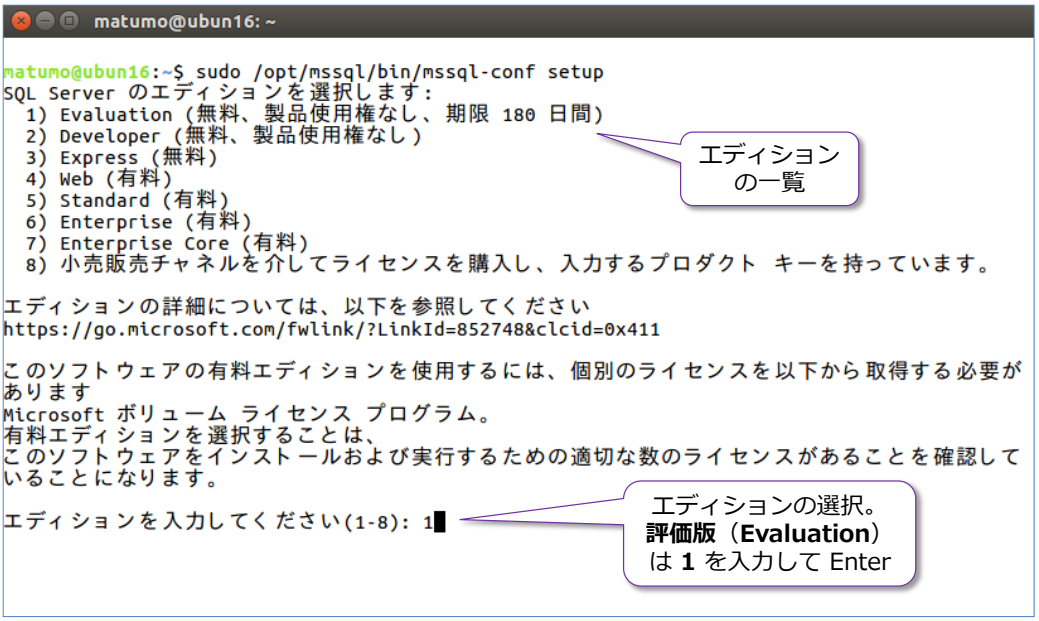


:



5. 最後に、**SQL Server のセットアップ**（エディションの選択や言語の選択、管理者アカウントへのパスワード設定など）を行います。これを行うには、次のように「**/opt/mssql/bin**」ディレクトリの「**mssql-conf setup**」を実行します。

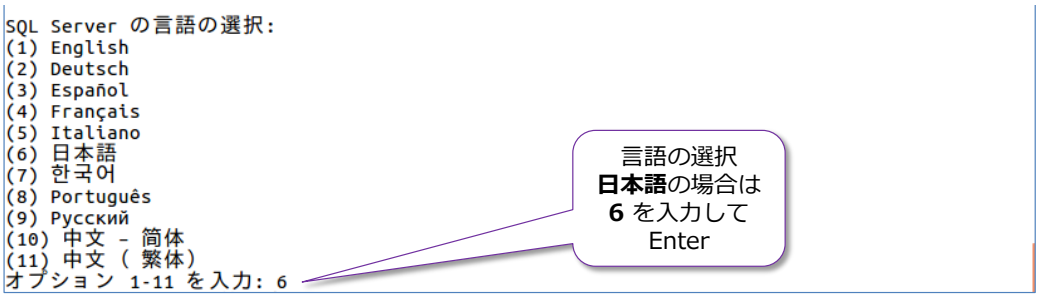
```
sudo /opt/mssql/bin/mssql-conf setup
```



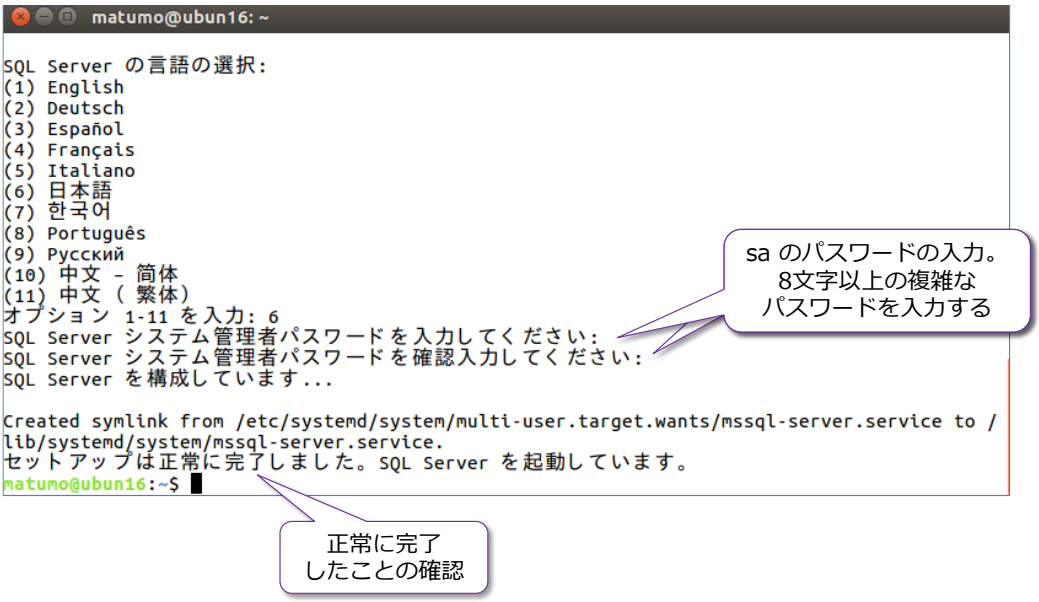
最初にエディションを選択しますが、**評価版**（**Evaluation**）を利用する場合は「**1**」を入力して、**Enter** キーを押下します。

次に、**ライセンス条項**（使用許諾契約書）に関する情報が表示されるので、内容を確認した上で、同意する場合は「**yes**」と入力します。

次に、**言語の選択**が表示されるので、**日本語**を利用する場合は「**6**」を入力します。



最後に **SQL Server の管理者アカウント**である「**sa**」に対する任意のパスワードを設定しますが、パスワードは 8 文字以上の複雑なもの（大文字、小文字、数字、記号の 4 種類のうち、いずれかの 3 種類の文字を含めるもの）に設定する必要があります。

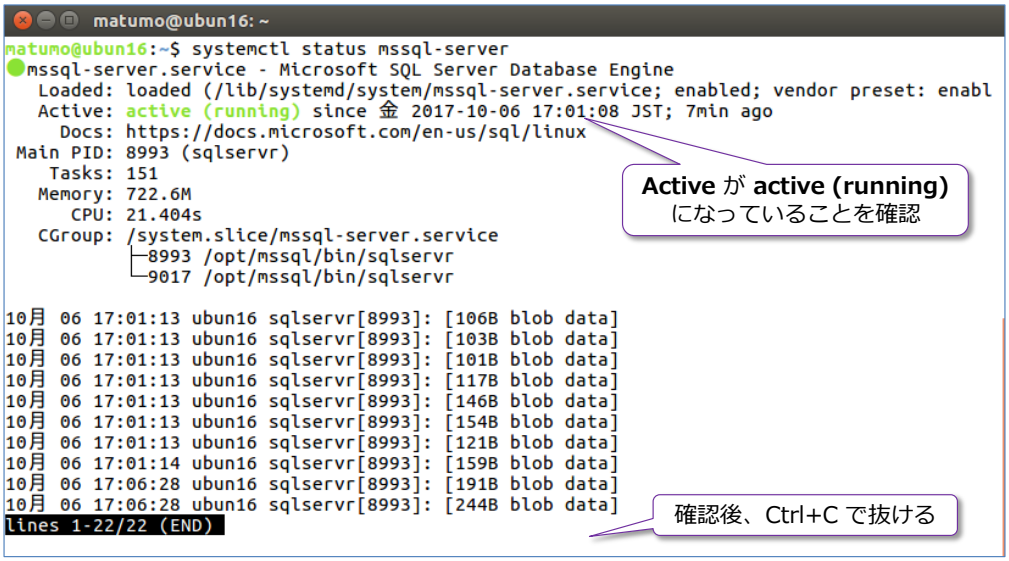


以上で、最後に「**セットアップは正常に完了しました**」と表示されれば、SQL Server のインストールが完了です。

インストールの確認

1. インストールが完了したことを確認するには、次のように **systemctl** で「**mssql-server**」の **status** を確認します。

```
systemctl status mssql-server
```



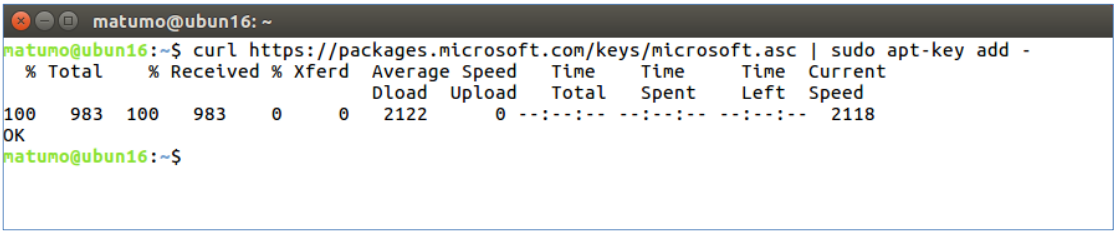
Active が **active (running)** になっていれば、SQL Server は正常に動作しています。

sqlcmd ツールのインストール

次に、SQL Server に接続するために、**sqlcmd** ツールをインストールします。

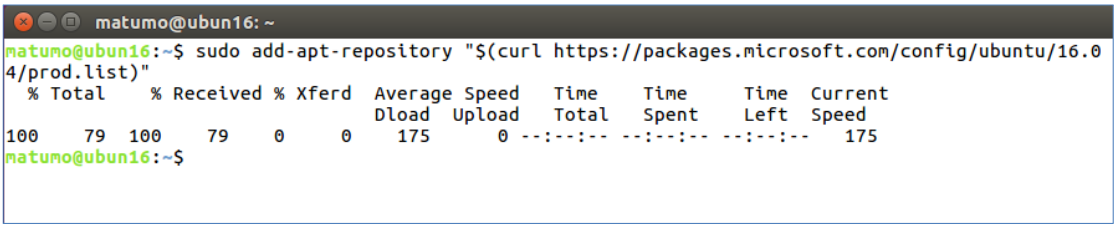
1. **sqlcmd** ツールをインストールするには、まず、リポジトリ キーをインポートします。

```
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
```



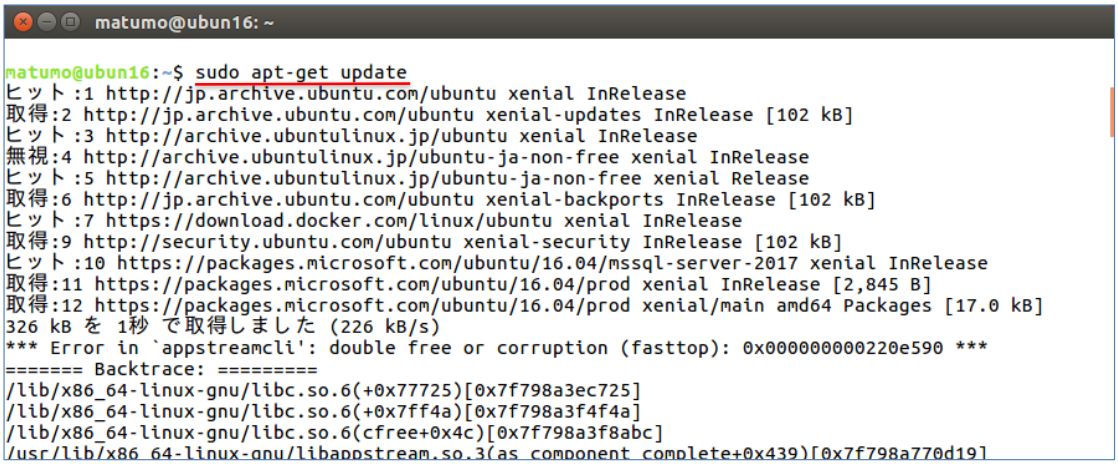
2. 次に、リポジトリを登録します。

```
sudo add-apt-repository "$(curl https://packages.microsoft.com/config/ubuntu/
16.04/prod.list)"
```



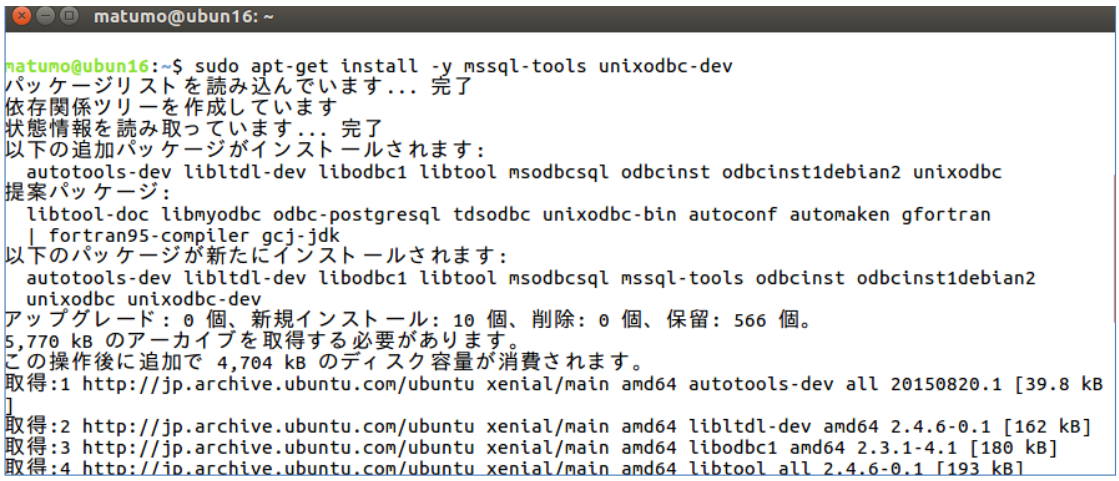
3. 次に、パッケージのアップデートを実行します。

```
sudo apt-get update
```

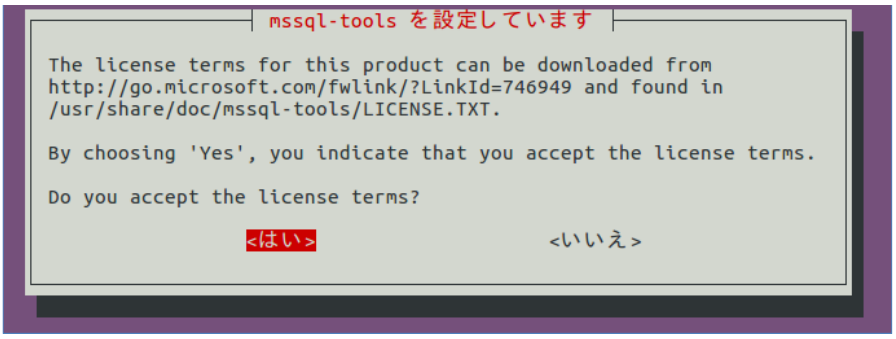


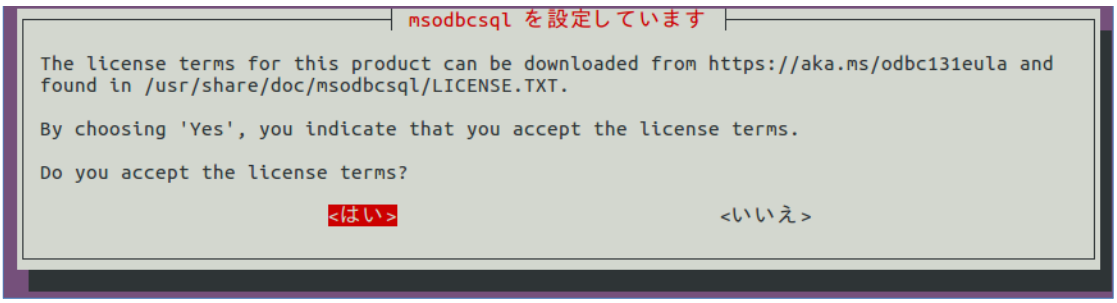
4. パッケージのアップデートが完了したら、次は **sqlcmd** ツールのインストールを行います。これは、次のように「**mssql-tools**」と「**unixodbc-dev**」を指定します。

```
sudo apt-get install -y mssql-tools unixodbc-dev
```



インストールの途中では、次のように **license terms**（**使用許諾契約書**）の確認が求められるので、内容を確認した上で、同意する場合は「**はい**」を選択します。





以上で **sqlcmd** ツールのインストールが完了です。

5. **sqlcmd** ツールは、「**/opt/mssql-tools/bin**」ディレクトリに格納されているので、次のように実行することができます。

```
/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P 'saに設定したパスワード'
```

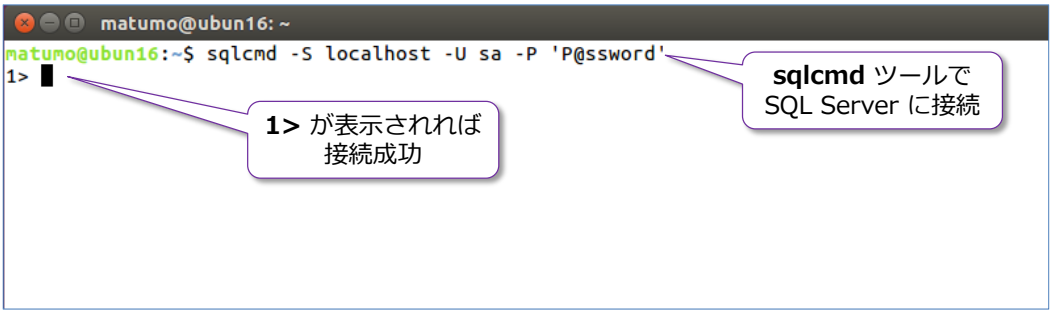
6. 上の手順のように毎回パスを記述するのが面倒な場合は、次のように **PATH** を設定しておくと便利です。

```
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
source ~/.bashrc
```

```
matumo@ubun16: ~
matumo@ubun16:~$ echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
matumo@ubun16:~$ echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
matumo@ubun16:~$ source ~/.bashrc
matumo@ubun16:~$
```

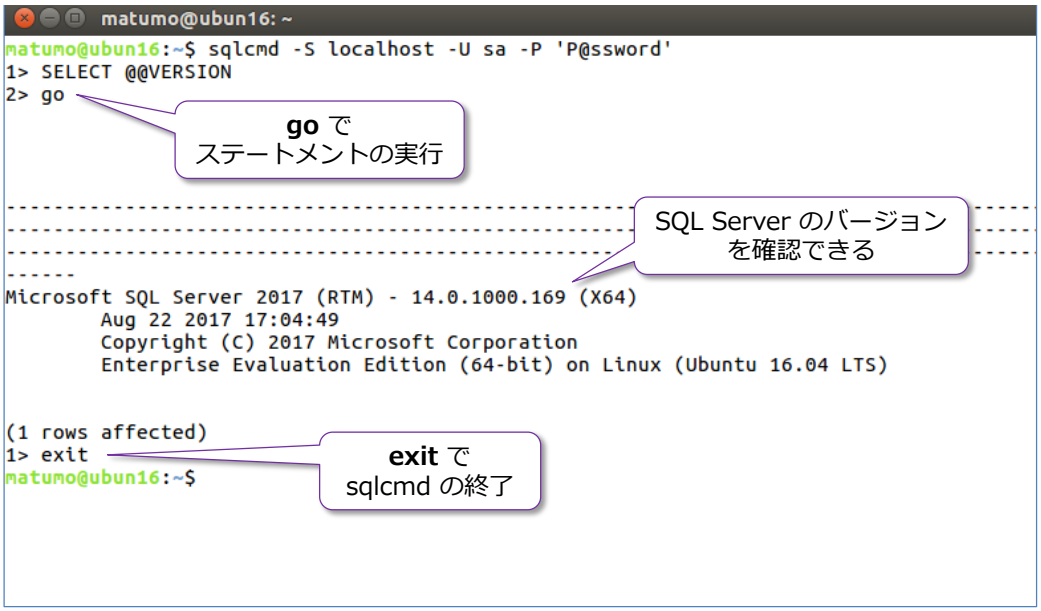
7. **PATH** を設定した後は、次のように **sqlcmd** と記述するだけで利用できるようになります。

```
sqlcmd -S localhost -U sa -P 'saに設定したパスワード'
```



sqlcmd で Transact-SQL ステートメントを実行するには「**go**」を付けます。

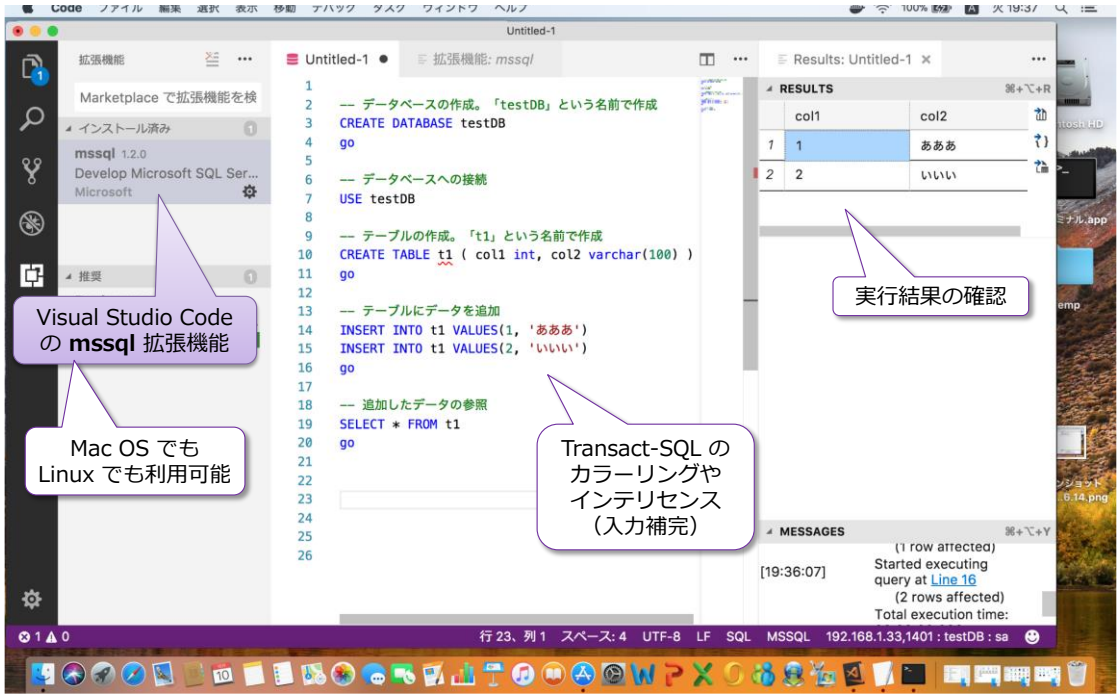
ここでは、SQL Server のバージョンを取得する SELECT ステートメントを「**SELECT @@VERSION**」と入力して、改行（Enter キー）、次に「**go**」を付けて、ステートメントを実行してみましょう。



sqlcmd を終了するには「**exit**」と入力します。

このように **sqlcmd** ツールを利用すれば、SQL Server 2017 に対して Transact-SQL ステートメントを実行できるようになります。また、合わせて、同じパス（**/opt/mssql-tools/bin**）には **bcp** ツールもインストールされるので、テキスト ファイル（CSV ファイルなど）を一括インポート（ファイルのデータをデータベース内のテーブルに格納）したい場合に利用できます。

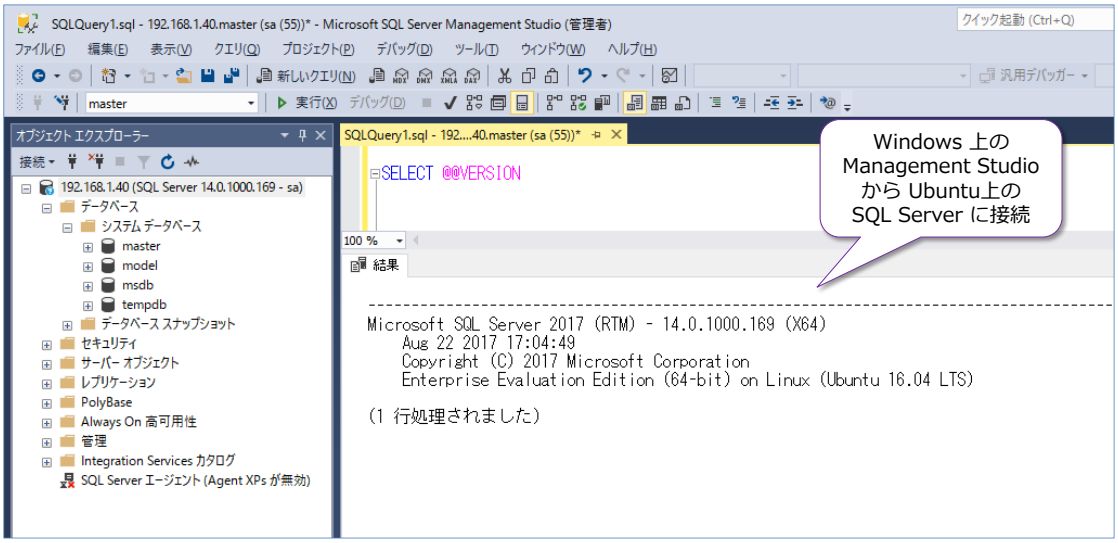
Docker のところで紹介しましたが、**sqlcmd** ツールは、コマンドライン ベースで使いづらい部分があるので、お勧めのツールなのが **Visual Studio Code** の **mssql 拡張機能**です



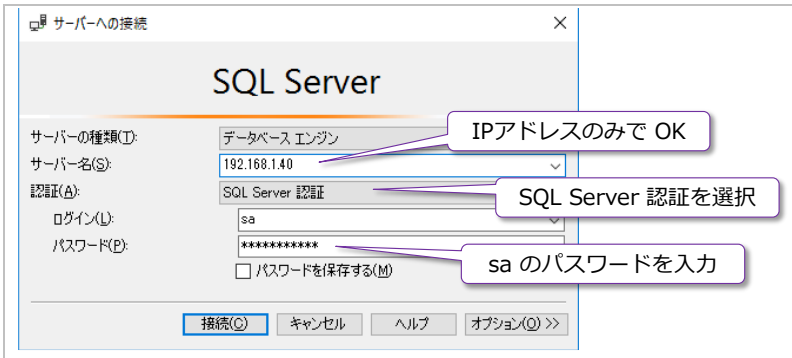
このツールの利用方法や、Visual Studio Code を利用したアプリケーション開発については、本自習書シリーズの No.2「**SQL Server 2017 on Linux**」編で詳しく説明しているので、こちらもぜひご覧いただければと思います。

Note： Windows の Management Studio から Linux 上の SQL Server に接続

Linux 上の SQL Server 2017 には、Windows 上の **Management Studio** ツール（SQL Server の管理ツール）から接続することも可能です。



Management Studio を利用して Linux 上の SQL Server に接続する場合は、次のように行えます。



本文中で試したインストール手順では、SQL Server 2017 は、ポート番号「**1433**」(SQL Server の既定のポート番号) を利用するようにインストールされるので、「**サーバー名**」には IP アドレスを入力するだけで大丈夫です (ポート番号は、**1433** の場合は省略できます)。なお、Linux 上でファイアウォールを有効化している場合には、ファイアウォールで **1433** ポートを解放しておく必要があります。また、hosts や DNS を利用している場合には、IP アドレスではなく、ホスト名で接続することもできます。

あとは、Windows 上の SQL Server を操作するのと同じように Linux 上の SQL Server を GUI で操作することができます。

RHEL や SUSE への SQL Server 2017 のインストール

再掲になりますが、SQL Server 2017 がサポートしている Linux プラットフォームは、次のとおりです。

- **Red Hat Enterprise Linux 7.3** または **7.4** Workstation, Server, and Desktop (ファイル システム: XFS または EXT4)
- **SUSE Enterprise Linux Server v12 SP2** (ファイル システム: EXT4)
- **Ubuntu 16.04 LTS** (ファイル システム: EXT4)

Red Hat Enterprise Linux や SUSE Enterprise Linux Server にインストールする手順については、以下の URL が参考になります。

Red Hat Enterprise Linux への SQL Server 2017 のインストール
https://docs.microsoft.com/ja-jp/sql/linux/quickstart-install-connect-red-hat

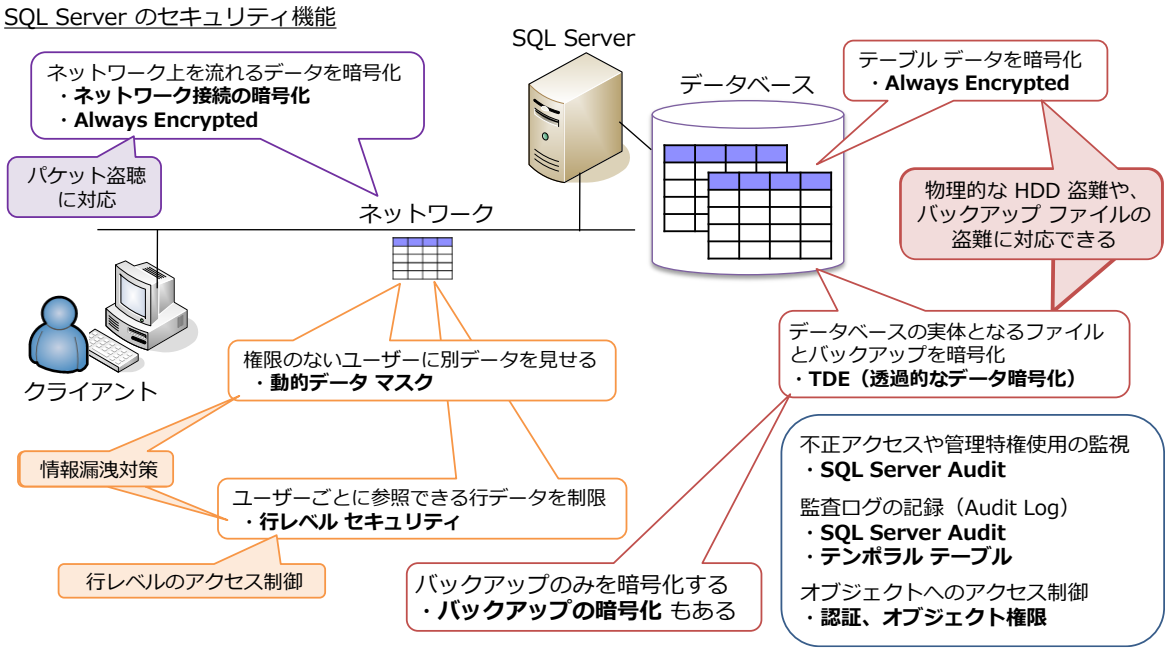
SUSE Enterprise Linux Server への SQL Server 2017 のインストール
https://docs.microsoft.com/ja-jp/sql/linux/quickstart-install-connect-suse

2.3 SQL Server 2017 on Linux のセキュリティ

SQL Server 2017 on Linux では、SQL Server で利用できる**標準のセキュリティ機能**をそのまま利用することができます。その主なものは、次のとおりです。

- **動的データ マスク**（Dynamic Data Masking）
- **行レベル セキュリティ**（Row Level Security）
- **バックアップの暗号化**（Backup Encryption）
- **ネットワーク接続の暗号化**（Encrypting Connection）
- **TDE（透過的なデータ暗号化）**、Enterprise エディションでのみ利用可能
- **SQL Server Audit（監査）**
- **Always Encrypted による列データの暗号化**
- **テンポラル テーブル**
- **認証、オブジェクト権限**

これらの機能は、次のような目的で利用できます。



情報漏洩対策として「**動的データ マスク**」や「**行レベル セキュリティ**」、ネットワークのパケット盗聴対策として「**ネットワーク接続の暗号化**」や「**Always Encrypted**」、HDD などのハードウェアの物理的な盗難への対策として「**TDE**」や「**バックアップの暗号化**」、不正アクセス対策や監査ログの記録（Audit Trail）として「**SQL Server Audit**」や「**テンポラリ テーブル**」、各種のオブジェクトへのアクセス制御として「**認証、オブジェクト権限**」といった形で、昨今求められているセキュリティ要件は、これらを利用することで簡単に満たすことができます。

なお、これらの機能のうち、TDE（透過的なデータ暗号化）の利用には Enterprise エディションが必要になりますが、その他の機能は Standard エディションでも利用することができます。どの

機能がどのエディションで利用できるかどうかについては、以下の URL が参考になります。

エディションと SQL Server 2017 on Linux のサポートされる機能
https://docs.microsoft.com/ja-jp/sql/linux/sql-server-linux-editions-and-components-2017

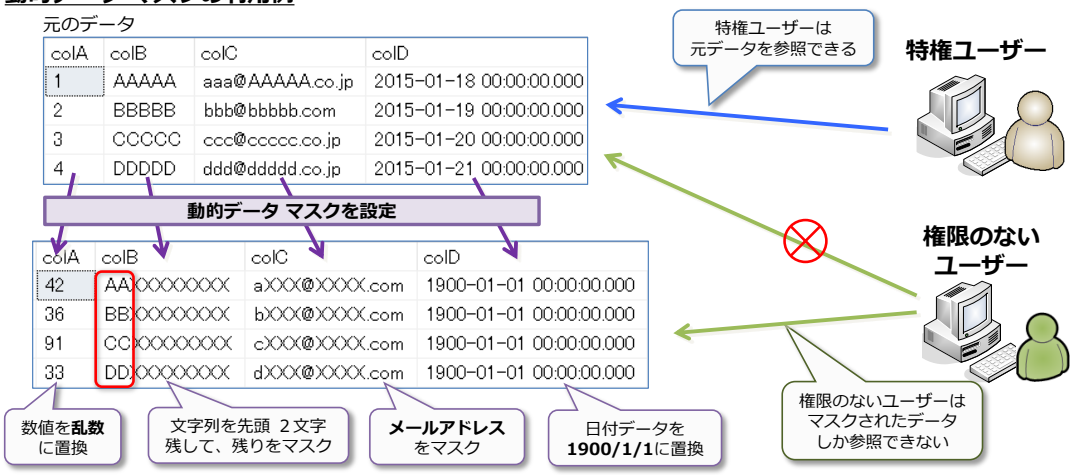
セキュリティ機能の概要

それぞれのセキュリティ機能の概要は、次のとおりです。

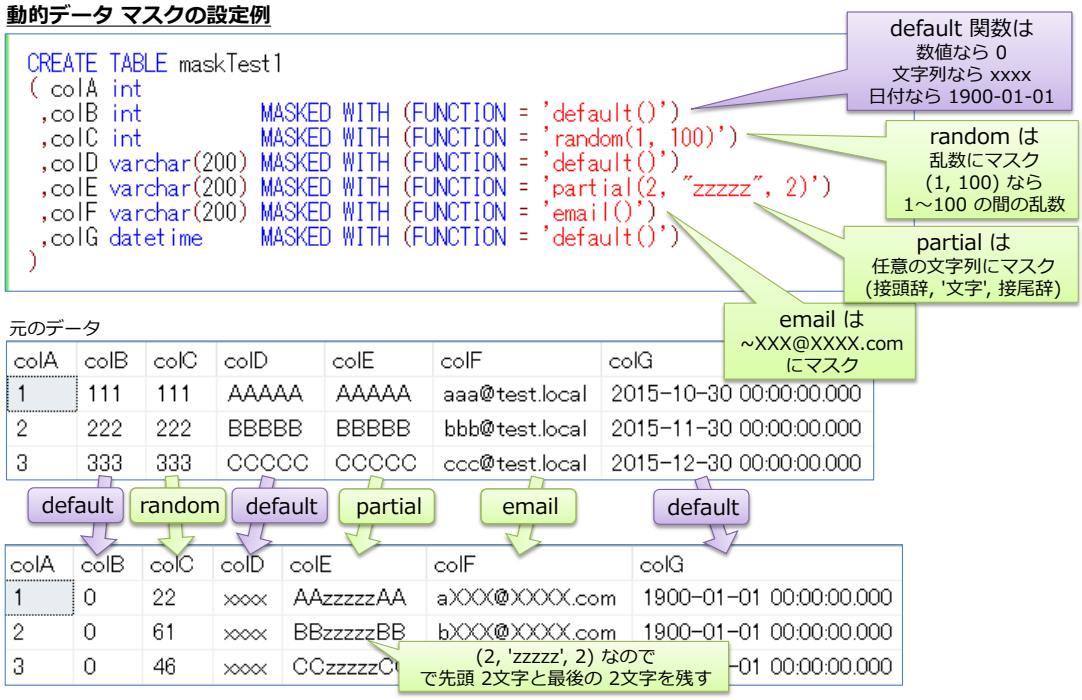
動的データ マスク（Dynamic Data Masking）

動的データ マスクは、顧客情報（クレジット カード番号やマイナンバーなど）や**機密情報をマスク**（別の値に置換）して、**情報漏洩**を防止できる機能です。

動的データ マスクの利用例



動的データ マスクの設定例

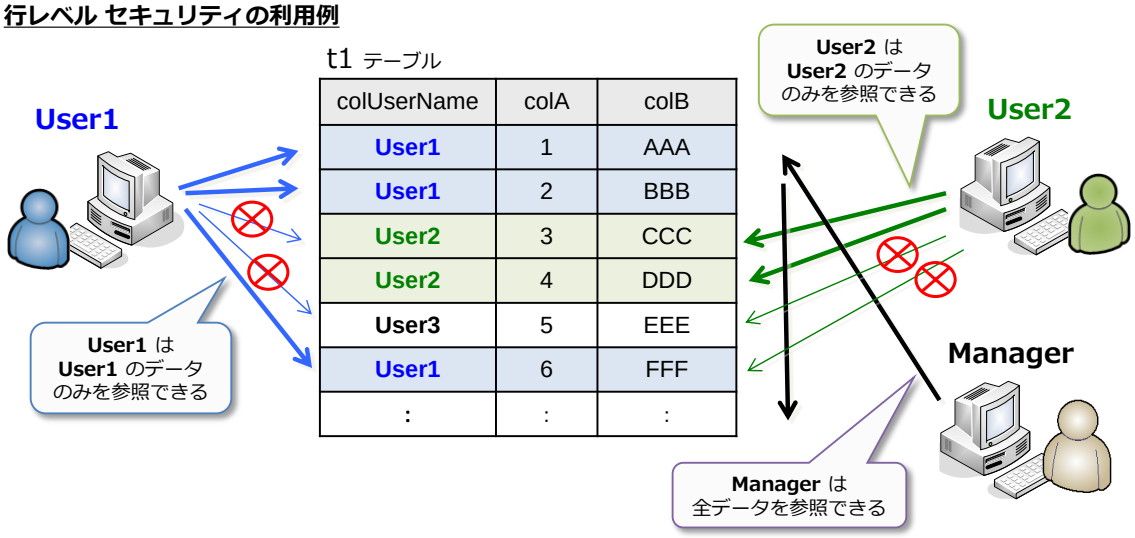


動的データ マスクでは、default 関数や random 関数、email 関数などを利用して、列データをマスクすることができます。

行レベル セキュリティ（Row Level Security）

行レベル セキュリティは、**行レベルのアクセス制御**を実現できる機能で、ユーザーごとに、参照できる行データを制限することができます。

行レベル セキュリティの利用例



バックアップの暗号化（Backup Encryption）

バックアップの暗号化は、バックアップ データを暗号化できるので、バックアップ ファイルの盗難への対策になります。

```
BACKUP DATABASE Northwind
TO DISK = N'/tmp/North_enc.bak'
WITH
  ENCRYPTION
   ( ALGORITHM = AES_256,
     SERVER CERTIFICATE = MyTestDBBackupEncryptCert    )
go
```

バックアップ ファイルを暗号化できる

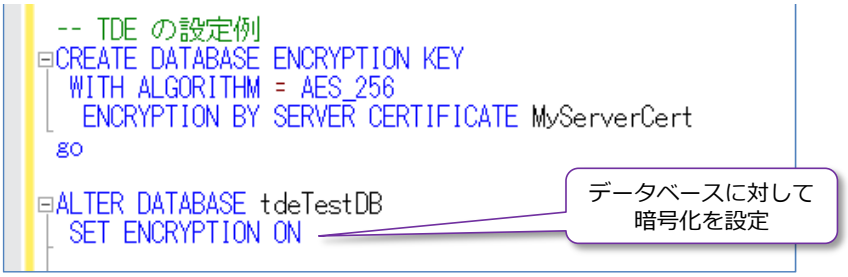
ネットワーク接続の暗号化（Network Encryption）

ネットワーク接続の暗号化は、ネットワーク上を流れるパケットを暗号化できるので、パケット盗聴への対策になります。SQL Server 2017 でネットワーク接続を有効化しておくと、クライアント（アプリケーション）からは、接続文字列に以下を追加することで暗号化接続を指定できるようになります。

```
JDBC
"encrypt=true; trustServerCertificate=false;"
ODBC
"Encrypt=Yes; TrustServerCertificate=no;"
ADO.NET
"Encrypt=True; TrustServerCertificate=False;"
```

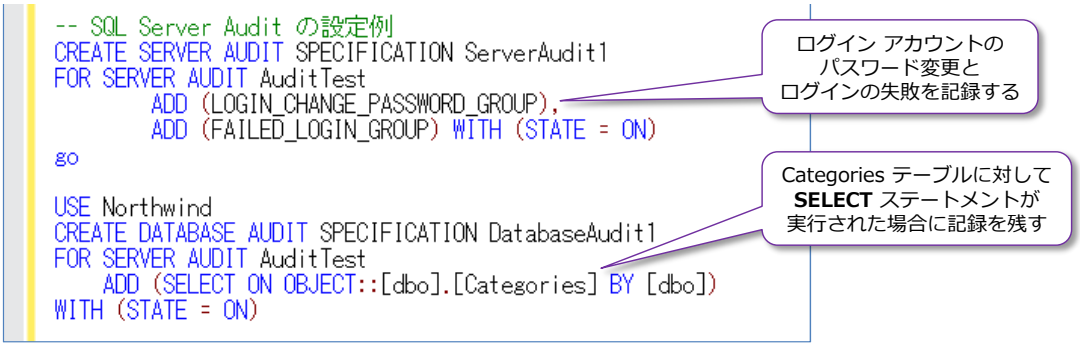
TDE（透過的なデータ暗号化）

TDE は、データベースを丸ごと暗号化することができるので、HDD などのハードウェアの物理的な盗難への対策になります。TDE を設定したデータベースは、バックアップ ファイルも暗号化されるので、前掲のバックアップの暗号化を利用しなくても、バックアップ データを暗号化して保護することができます。



SQL Server Audit（監査）

SQL Server Audit では、ユーザーの操作をすべて記録することができるので、**Audit Trail**（監査証跡）や、各種の**法令遵守**に利用できます（J-SOX や内部統制、PCI DSS など、コンプライアンスを実現するために欠かせない機能になります）。



Always Encrypted による列データの暗号化

Always Encrypted は、ネットワーク上を流れるデータも、データベース内に格納されるデータも、**すべて暗号化**して格納できる機能です（列データを暗号化して、アプリケーションも透過的に利用することができます）。これを利用している場合は、TDE を利用していなくても、データベース（の列データが暗号化されているので）物理的な盗難防止になります。また、前掲のネットワーク接続の暗号化を利用していなくても、ネットワーク上のデータを暗号化することができます。

このように、SQL Server には、セキュリティを強化できる機能が非常に豊富に用意されているので、セキュリティ要件を簡単に満たすことができます。こうした Linux 上で利用できるセキュリティ機能については、本自習書シリーズの No.2「**SQL Server 2017 on Linux**」編で詳しく説明しているので、こちらもぜひご覧いただければと思います。

2.4 SQL Server 2017 on Linux で利用できる機能／利用できない機能

ここでは、SQL Server 2017 on Linux で利用できる機能と利用できない機能について説明します。

SQL Server 2017 on Linux で利用できる機能

SQL Server on Linux で利用できる機能は、次のとおりです。

SQL Server データベース エンジンに関して

SQL Server on Linux では、データベースに関する**基本操作**（DB 作成、テーブル作成、データの追加／更新／削除、ビューやストアド プロシージャ、トリガー、インデックスの作成）ができることはもちろん、性能向上を実現できる**列ストア インデックス**や、**インメモリ OLTP**、**データ パーティション**、**データ圧縮**も利用できます。

また、第 4 章で説明する**クエリ ストア**や**自動チューニング**、**Adaptive Query Processing**（適応型 JOIN など）、**グラフ データベース**、**非クラスター化列ストア インデックスのオンライン再構築**、**再開可能なインデックス再構築**、**スマート バックアップ**なども利用することができます。

SQL Server on Linux の **Standard** エディションでは、利用できる **CPU** に関して 4 ソケットまたは 24 コアのいずれかの小さい方に制限されますが、**Enterprise** エディションであれば OS がサポートできる最大ソケット数／コア数まで利用することができます。また、**メモリ**に関しては、**Standard** エディションでは **128GB** に制限（列ストア セグメントのキャッシュとインメモリ OLTP のメモリ最適化テーブルは **32GB** に制限）されますが、**Enterprise** エディションであれば OS がサポートできる最大容量のメモリまで利用することができます。

Enterprise エディションでのみ利用できる主な機能としては、自動チューニングや、適応型 JOIN、非クラスター化列ストア インデックスのオンライン再構築、再開可能なインデックス再構築、並列インデックス処理、パーティションのパラレル処理、リソース ガバナー、NUMA 対応のラージ ページ メモリ、I/O リソース管理、分散パーティション ビュー、パラレル整合性チェック、TDE（透過的なデータ暗号化）などがあります。どの機能がどのエディションで利用できるのかについては、以下の URL が参考になります。

エディションと SQL Server 2017 on Linux のサポートされる機能
https://docs.microsoft.com/ja-jp/sql/linux/sql-server-linux-editions-and-components-2017

SQL Server on Linux では、前掲の**セキュリティ機能**（ユーザー作成やオブジェクト権限の設定だけでなく、監査や行レベル セキュリティ、動的データ マスク、TDE、バックアップ暗号化、ネットワーク接続の暗号化、Always Encrypted、テンポラル テーブル、包含データベースなど）も利用できます。

また、その他のシステムとの連携に役立つ**リンク サーバー**や **bcp**、**BULK INSERT** なども利用することができます（後述の Integration Services のパッケージ実行も可能です）。

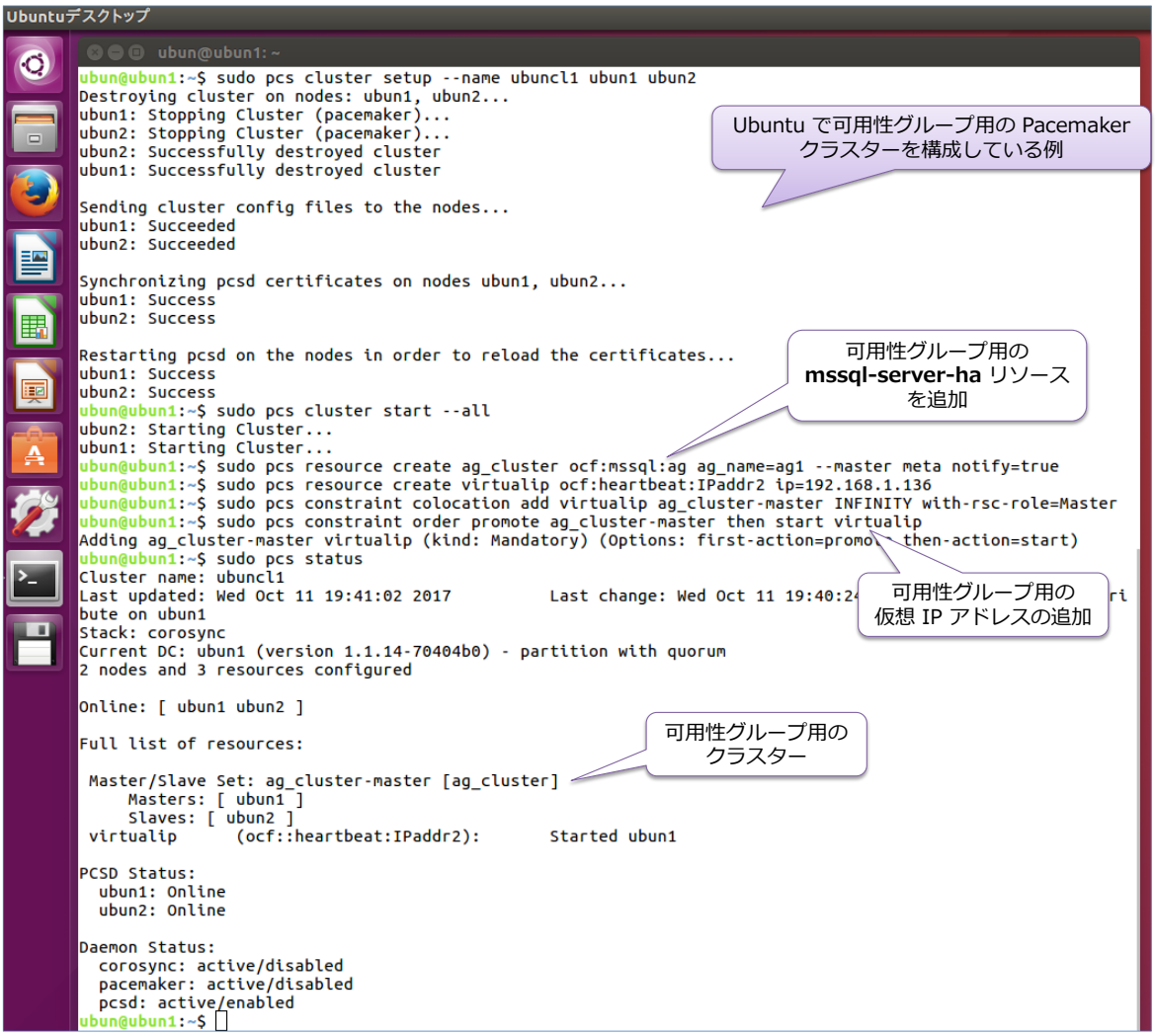
SQL Server でのバックアップのスケジュールなどで定番となっている **SQL Server Agent ジョブ**機能についても、Transact-SQL ステートメントの定期実行がサポートされているので、バックアップやインデックスの再構築といった各種のメンテナンス系の SQL をスケジュール実行することができます。また、**データベース メール**機能もサポートしているので、ジョブの成功や失敗をメールで通達するといったことも行えます。

次章で説明する **Machine Learning Services**（ML Services）は未サポートですが、**PREDICT 関数はサポート**しているので、この関数を利用して ML Services の Revoscalepy や RevoScaleR で機械学習したモデルにアクセスして、予測を実行することができます（後述）。

AlwaysOn 可用性グループ（Availability Group）**による冗長構成に関して**

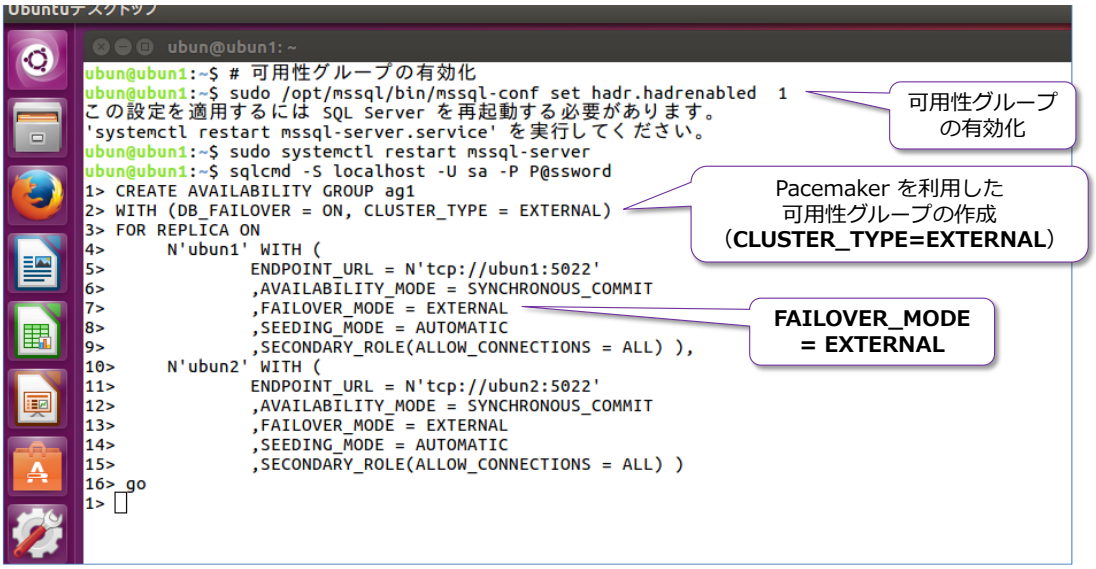
Linux 環境でも **AlwaysOn 可用性グループ**を構成できます。Windows 環境の場合は、WSFC（Windows Server フェールオーバー クラスター）を利用して可用性グループを構成しますが、Linux 環境の場合は、**Pacemaker** を利用します。

Ubuntu で可用性グループ用の Pacemaker クラスターを作成している例



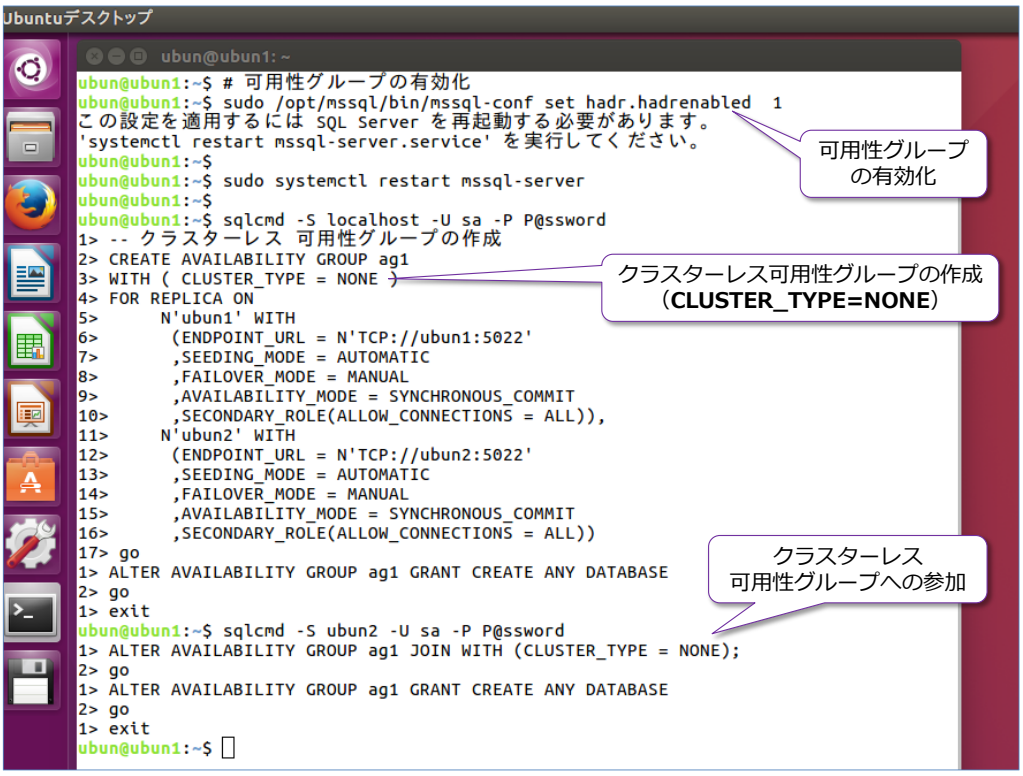
可用性グループを作成するときは、次のように「**CLUSTER_TYPE=EXTERNAL**」と指定すること

で Pacemaker を利用した可用性グループを利用できるようになります。



可用性グループでは、**クラスター レス**（クラスター不要）の構成もサポートして、この場合は Pacemaker は必要ありません。ただし、これは高可用性が目的のものではなく、**読み取り専用スケール**（読み取り性能を向上させる目的）として可用性グループを利用することになります。

Ubuntu でクラスター レス可用性グループを作成している例



可用性グループは、Windows と Linux にまたがって構成することもできます。また、SQL Server 2016 からの新機能である**分散可用性グループ**（Distributed Availability Group）に、Windows と Linux の可用性グループを含めることもできます。

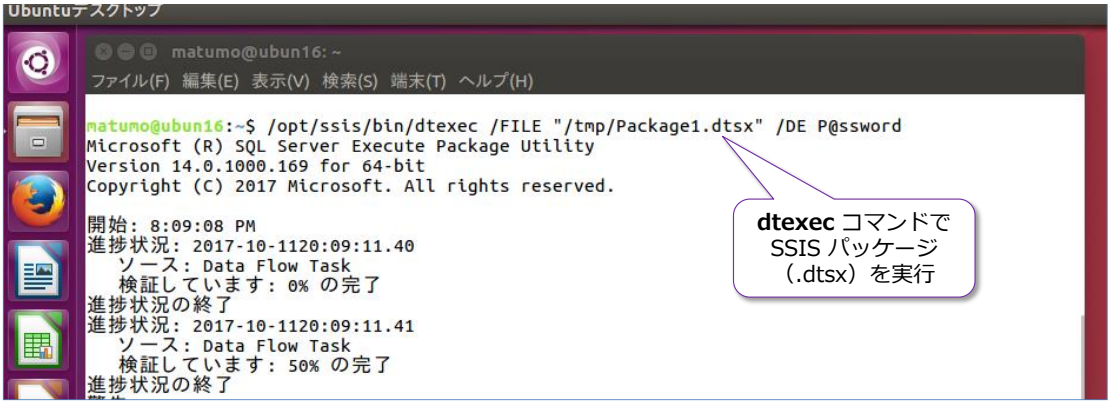
Windows 環境から Linux 環境への移行に関して

Windows 上の SQL Server 環境で取得したバックアップを、Linux 環境にリストアすることも、何の問題もなく行えるので、**移行**（マイグレーション）も簡単です（リストア時の考慮事項は、Windows 環境でのリストアの場合と全く同様です）。

既存の Windows 環境に対して、Linux を含めた可用性グループを構成すれば、まったく同じデータベース（ミラー化した複製データベース）を Linux 上に作成することができるので、段間的な移行用途として可用性グループを利用することもできます。

Integration Services（SSIS）に関して

Integration Services（SSIS）は、SQL Server 2017 on Linux でも利用することができます。ただし、SSIS パッケージの作成は、従来どおり Windows 上の **SSDT**（SQL Server Data Tools）を利用して行って、作成したパッケージ ファイル（**.dtsx**）を Linux 上にコピーして、**dtexec** コマンドで実行するという形になります。



ただし、Integration Services に関して、以下の機能については利用することができません。

- SSIS カタログ データベース
- SQL Server Agent ジョブでのパッケージ実行ジョブの登録
- Windows 認証、サードパーティ コンポーネント、CDC（変更データキャプチャ）
- SSIS スケールアウト、Azure Feature Pack for SSIS、Hadoop／HDFS サポート、Microsoft Connector for SAP BW

SQL Server 2017 on Linux では利用できない機能

SQL Server 2017 on Linux では利用できない機能は、次のとおりです。

	利用できない機能
データベース エンジン	トランザクション／マージ レプリケーション ストレッチ データベース、Polybase、サードパーティ接続の分散クエリ、拡張システム ストアド プロシージャ（xp_cmdshell など）、Filetable、バッファ プール拡張、 CLR アセンブリでの EXTERNAL_ACCESS／UNSAFE 権限
SQL Server Agent	ジョブのサブシステム： CmdExec、PowerShell、Queue Reader、SSIS、SSAS、SSRS 警告（Alert）、ログ リーダー エージェント、CDC、 Managed Backup
可用性	データベース ミラーリング（DBM）
セキュリティ	拡張キー管理 リンク サーバー／可用性グループでの Active Directory 認証
その他のサービス	SQL Server Browser、 Machine Learning Services (ただし、PREDICT 関数は利用可能。3章で説明)、 StreamInsight、 Analysis Services（SSAS）、 Reporting Services（SSRS）、 Data Quality Services（DQS）、 Master Data Services（MDS）

こうした on Linux での未サポート機能については、以下のリリースノートが参考になります。

Release notes for SQL Server 2017 on Linux

Unsupported features and services

https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-release-notes#Unsupported

STEP 3. Machine Learning Services（機械学習サービス）

この STEP では、SQL Server 2017 で提供された **Machine Learning Services**（ML Services、機械学習サービス）の概要を説明します。SQL Server 上に統合された Python によって、どういったことができるのかなどを説明します。

この STEP では、次のことを学習します。

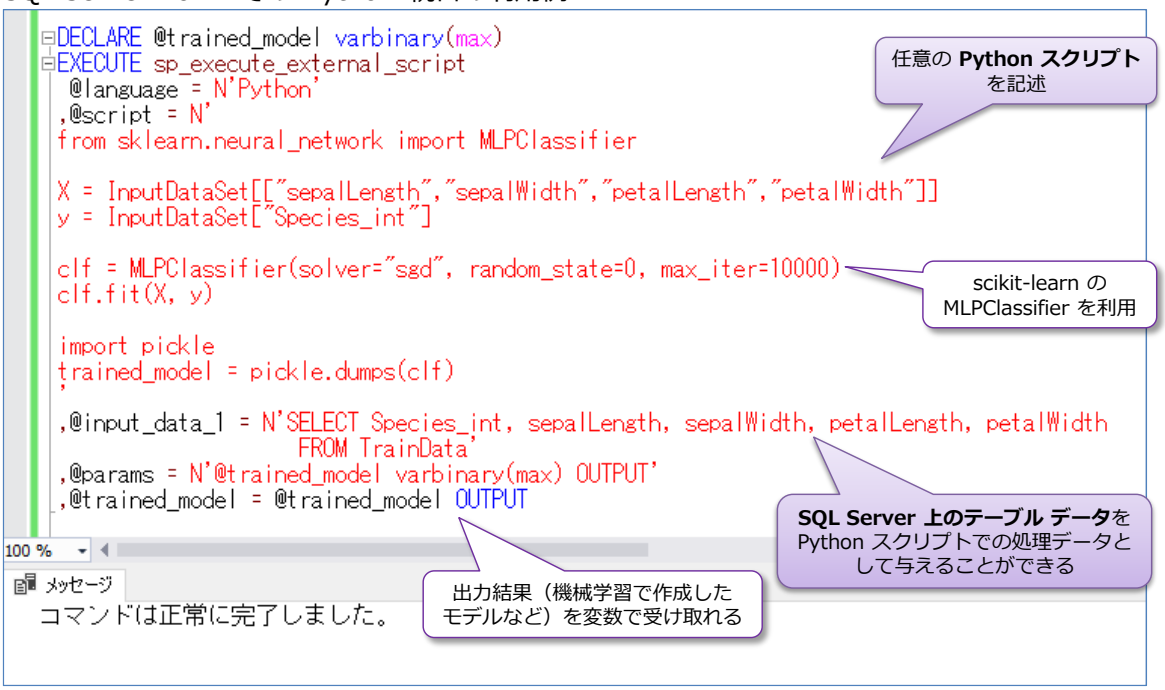
- ✓ Machine Learning Services の概要／インストール方法
- ✓ Python を利用した Machine Learning（機械学習）の例
- ✓ R を利用した Machine Learning（機械学習）の例

3.1 Machine Learning Services の概要／インストール方法

SQL Server 2017 からの新機能である「**Machine Learning Services**」(ML Services) は、SQL Server 2016 で提供されていた「**SQL Server R Services**」(R 統合) 機能を強化して、名称変更したものです。

SQL Server 2016 の「**SQL Server R Services**」では、R 言語を SQL Server に統合（ビルトイン）することによって、**機械学習**（Machine Learning）によるモデルの作成や予測（Predict）を行うことができましたが、SQL Server 2017 からは、**Python**（現在、機械学習／データ サイエンティストに最も人気があり、最も利用者数が多い言語）も統合されるようになりました。

SQL Server 2017 での Python 統合の利用例



これによって、昨今のトレンドである**ニューラル ネットワーク**や**ディープ ラーニング**（Deep Learning：深層学習）を利用した画像認識や音声認識、自然言語処理、各種の予測（Predict）およびモデル作成といった、いわゆる **AI**（人工知能）を実装することができます（SQL Server は、初めて AI 機能を搭載したデータベース製品でもあります）。

SQL Server 上に統合した **Python** では、Python の定番のライブラリである「**NumPy**」や「**pandas**」、「**scikit-learn**」、「**pickle**」、「**PIL**」などを利用できることはもちろんのこと、**ディープ ラーニング**での**定番フレームワーク**である「**Microsoft Cognitive Toolkit**（**CNTK**）」や「**Chainer**」、「**Google TensorFlow**」、「**Caffe**」、「**Theano**」なども利用できます（通常の Python と同様、**pip** でインストールして利用できます）。

Machine Learning Services（以降、**ML Services** と記述）の最大のメリットは、Python スクリプトで利用する**入力データ**（訓練データやテスト データ）に、**SELECT** ステートメントを記述して SQL Server 上のデータを直接指定できる点です。通常の Python では、データベースのデータを利用するには、データベース サーバーへの接続やクエリを実行するためのスクリプトを記述しなければなりませんが、ML Services では、上の画面のように「**@input_data_1**」引数に

SELECT ステートメントを記述して、それをスクリプト内で利用することができます（スクリプト内では、**pandas** の **DataFrame** として利用できます）。

また、ML Services の組み込みのライブラリである「**Revoscalepy**」や「**RevoScaleR**」で作成したモデル（rx_serialize_model 関数でシリアライズ化したモデル）に対しては、Transact-SQL ステートメントの **PREDICT** 関数を利用してアクセスできるのもメリットです（後述）。

ML Services は、**GPU**（Graphics Processing Unit）にも対応しているので、GPU を利用すれば CPU よりも高速に演算を行うことができます。

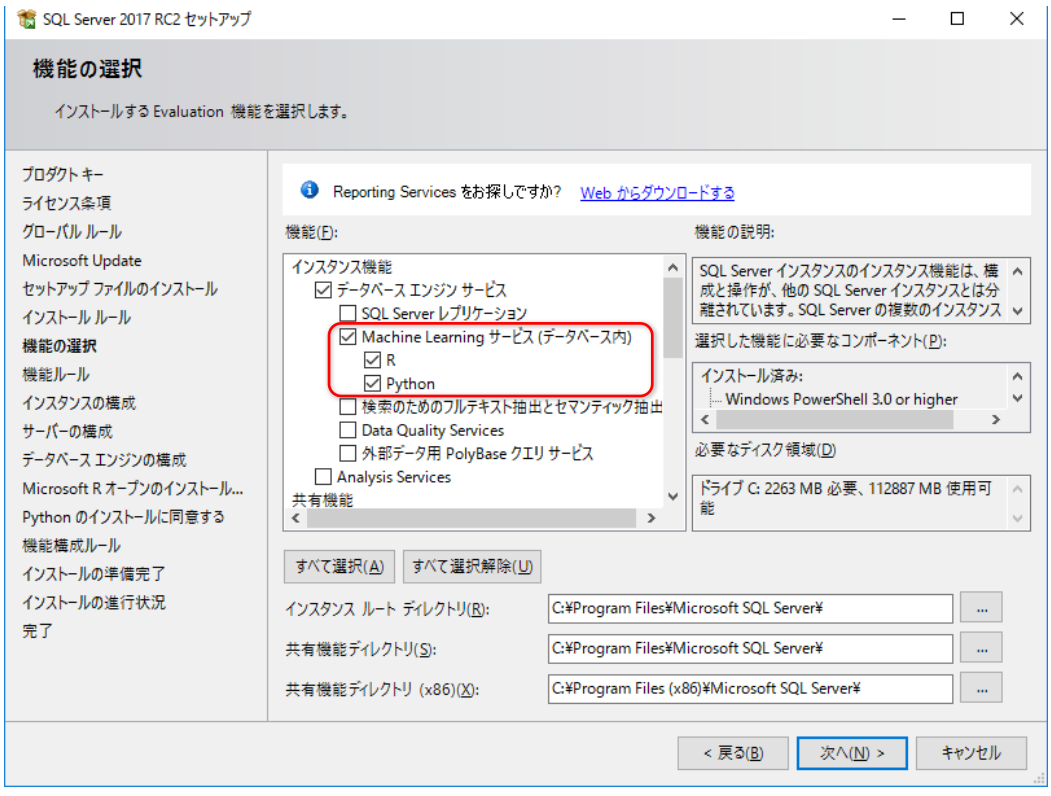
ML Services（Machine Learning Services）のインストール

SQL Server 2017 で、ML Services を利用するために必要となる作業は、次のとおりです。

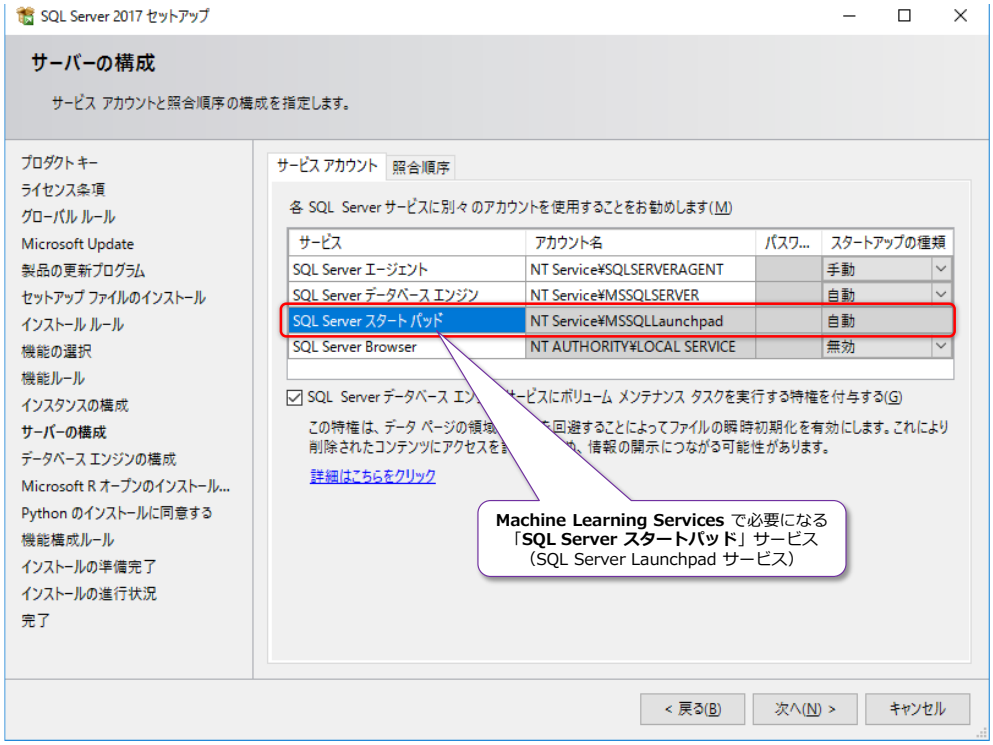
- SQL Server のインストール時に「**Machine Learning サービス（データベース エンジン内）**」を選択して、「**R**」または「**Python**」を選択（両方選択しても OK）
- sp_configure で「**external scripts enabled**」を「**1**」に変更して、SQL Server サービスを再起動する

インストール手順は、次のとおりです。

1. まずは、SQL Server 2017 のインストール時に、次のように［**機能の選択**］ページで、［**データベース エンジン サービス**］の［**Machine Learning サービス（データベース エンジン内）**］をチェックして、「**R**」や「**Python**」（利用したい言語）をチェックします。

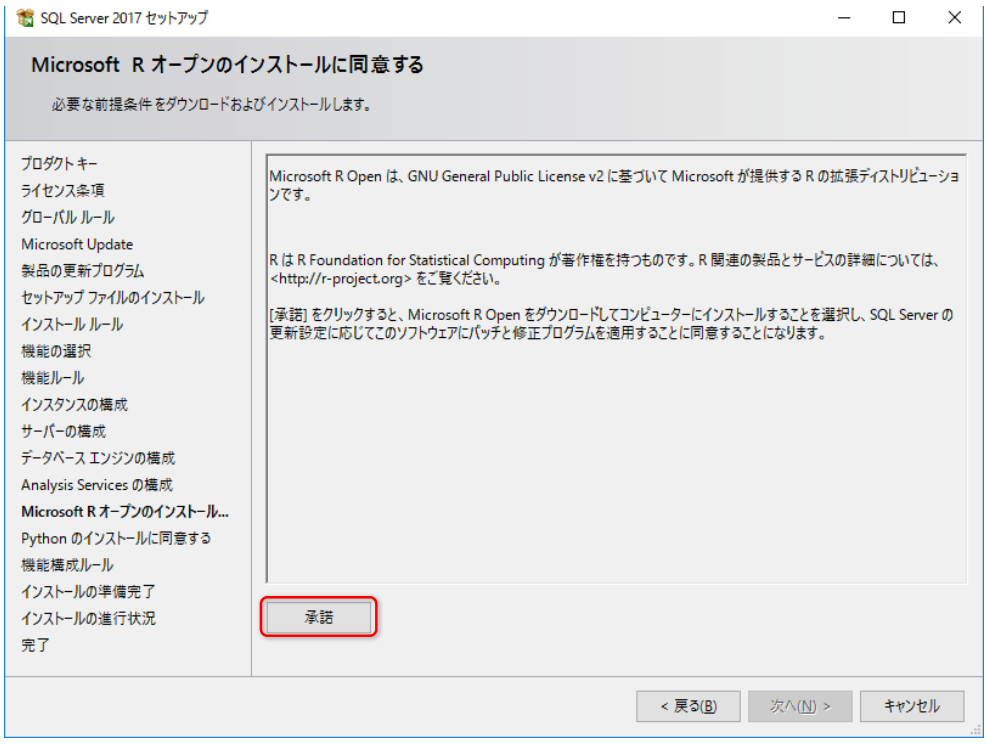


これらを選択した場合は、次のように［**サーバーの構成**］ページで、「**SQL Server スタートパッド**」（**SQL Server Launchpad**）サービスが表示されます。



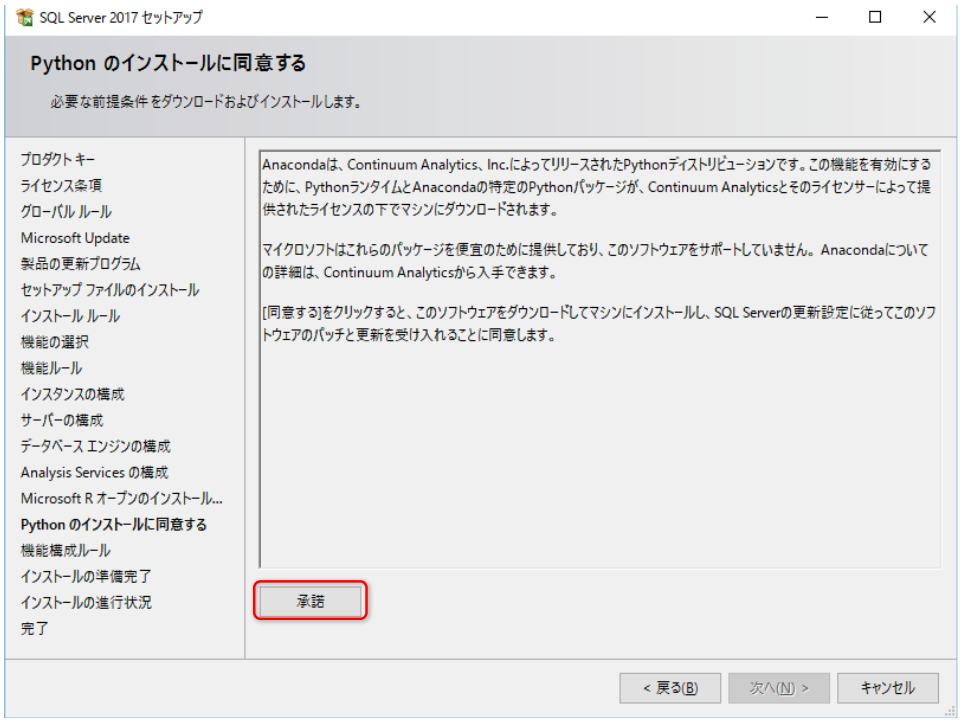
このサービスは、ML Services を利用するために必要になるので、［**スタートアップの種類**］が「**自動**」になっていることを確認します（自動起動するように設定します）。

ML Services の利用言語として「**R**」を選択している場合は、次のように［**Microsoft R オープンのインストールに同意する**］ページも表示されます。



内容を確認した上で、[**承諾**] ボタンをクリックすれば、SQL Server に統合された R 言語を利用できるようになります。

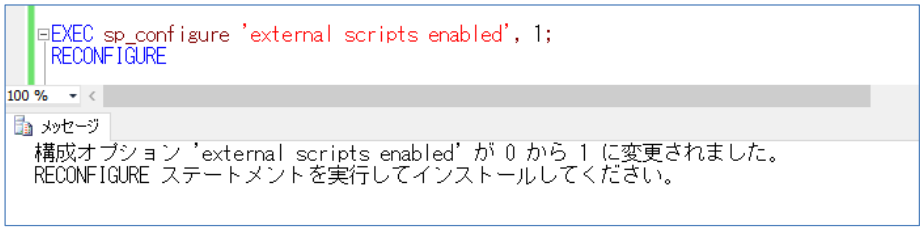
ML Services の利用言語として「**Python**」を選択している場合は、次のように［**Python のインストールに同意する**］ページも表示されます。



内容を確認した上で、［**承諾**］ボタンをクリックすれば、SQL Server に統合された Python を利用できるようになります（Python 統合では **Anaconda** を利用しています）。

2. ML Services のインストールが完了した後は、**sp_configure** を利用して、「**external scripts enabled**」を「**1**」に変更します。これを行うには、Management Studio を起動して、クエリ エディターを開き、次のように実行します。

```
EXEC sp_configure 'external scripts enabled', 1;
RECONFIGURE
```



3. 次に、SQL Server サービスを再起動します。

以上で、ML Services を利用できるようになり、SQL Server に統合（ビルトイン）された Python や R をクエリ エディターから実行できるようになります（スクリプトの実行には、後述の **sp_execute_external_script** システム ストアド プロシージャを利用します）。

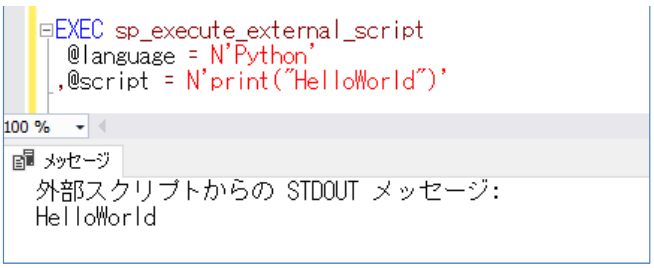
Python スクリプトの実行 ～sp_execute_external_script～

クエリ エディターから Python スクリプトを実行するには、次のように **sp_execute_external_script** システム ストアド プロシージャを利用します。

```
EXEC sp_execute_external_script
       @language = N'Python'
     , @script = N'実行したい Python スクリプト'
     , @input_data_1 = N'Python で処理したい入力データ'
     , @input_data_1_name = N'input_data_1 に対して設定する名前。既定値は InputDataSet'
     , @output_data_1_name = N'Python スクリプトで処理した結果。既定値は OutputDataSet'
     , @params = N'追加の変数定義。結果を変数で受け取る場合などに利用'
     WITH RESULT SETS ((列名 データ型 null/not null, ･･･));
```

@language で「**Python**」を指定して、**@script** に「**Python スクリプト**」を記述することで、任意の Python スクリプトを実行することができます。また、Python スクリプトに与えたい入力データは「**@input_data_1**」および「**@input_data_1_name**」で指定し、Python スクリプトで処理した結果（出力データ）は「**@output_data_1_name**」で指定できます。「**@input_data_1_name**」を省略した場合は **InputDataSet**、「**@output_data_1_name**」を省略した場合は **OutputDataSet** という名前が補われます。

また、最小限のパラメーターは、**@language** と **@script** の 2 つで、以下のように実行することもできます（Python の print で単純に文字列を出力）。



R スクリプトの実行

sp_execute_external_script では、**@language** で「**R**」を指定することで、R スクリプトを実行することができます（その他の引数の利用方法は Python の場合と同様です）。

```
EXEC sp_execute_external_script
       @language = N'R'
     , @script = N'実行したい R スクリプト'
     , @input_data_1 = N'R で処理したい入力データ'
     , ～
```

3.2 Python を利用した Machine Learning（機械学習）の例

前述したように ML Services の Python では、Python の定番のライブラリである「**NumPy**」や「**pandas**」、「**scikit-learn**」、「**pickle**」、「**PIL**」などを利用できることはもちろんのこと、**ディープ ラーニング**での**定番フレームワーク**である「**Microsoft Cognitive Toolkit（CNTK）**」や「**Chainer**」、「**Google TensorFlow**」、「**Caffe**」、「**Theano**」なども利用できます。通常の Python と同様、**pip** でインストールして利用できます。

インストールされている Python のバージョンは **3.5.2** で、**Anaconda** の 4.3.22 がインストールされているので、NumPy や scikit-learn、pickle 、PIL は、import を記述するだけで、pip でインストールすることなく利用できます。ただし、pandas については、内部的に import 済みになっているので、import を記述するとエラーになってしまいます。pandas は、pandas という名前で「**pandas.DataFrame**」のような形でスクリプト内でそのままの名前で利用する必要があります。

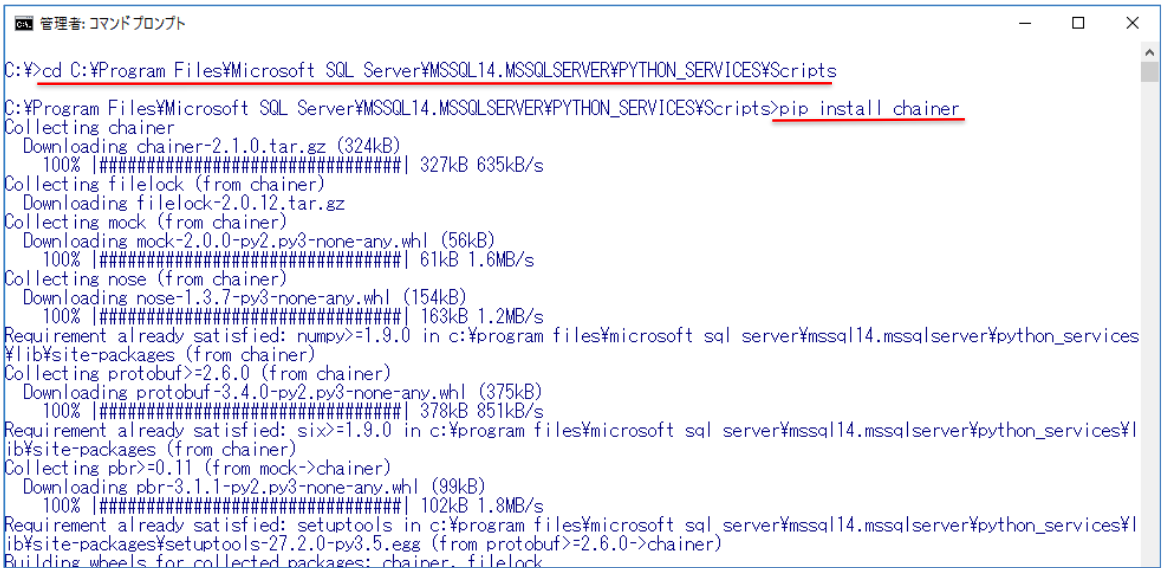
Chainer や **TensorFlow** などは、**pip** でインストールすることによって利用できるようになりますが、**pip** は、以下のフォルダーに格納されています（既定のインスタンスの場合）。

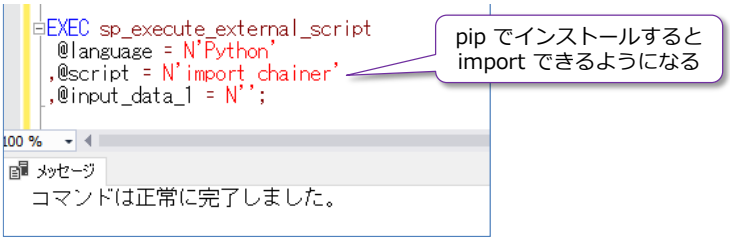
pip.exe の場所

C:¥Program Files¥Microsoft SQL Server¥MSSQL14.MSSQLSERVER¥**PYTHON_SERVICES**¥Scripts

したがって、コマンド プロンプト（"管理者として実行" で起動する必要があります）を開いて、このフォルダーに移動（**cd**）して、pip を実行するようにします。

Chainer をインストールする場合



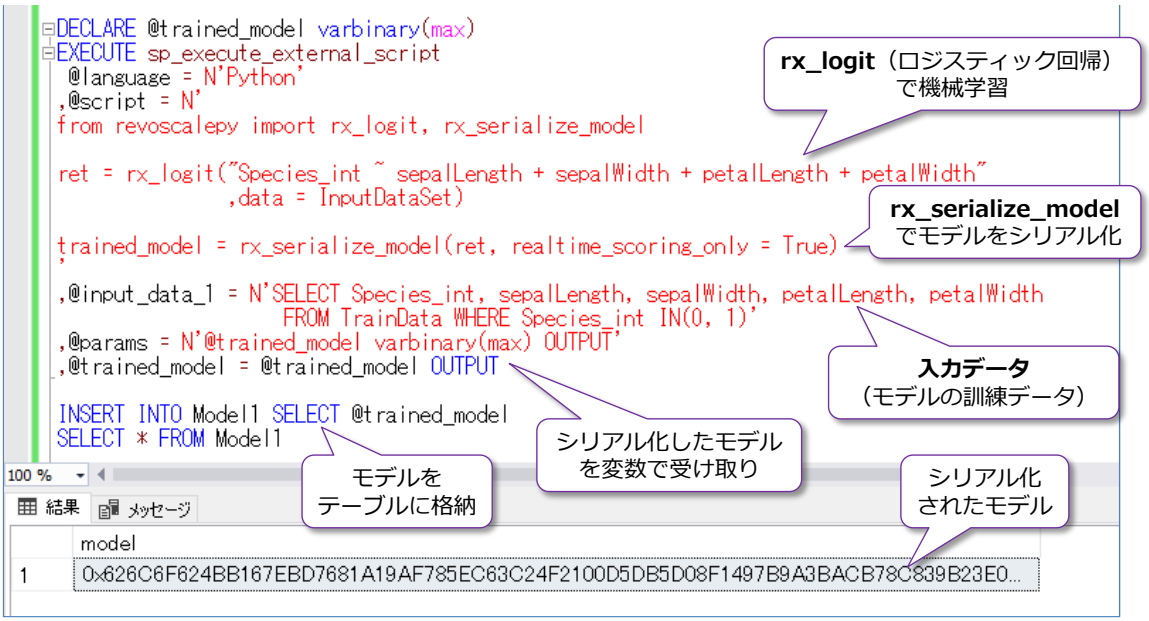


Revoscalepy と Native Scoring（ネイティブ スコアリング）

ML Services の前身は、SQL Server 2016 での「**SQL Server R Services**」になりますが、これは、オープンソースの R の性能的な欠点を補うために、Enterprise 向けに性能強化を計ったプラットフォームとして提供されていた「**Revolution R**」を買収したものでした（現在は、Microsoft R という名前に名称変更しています）。

Revolution R では、**RevoScaleR** と呼ばれる性能強化を計ったパッケージを提供していて、予測を行う **rxPredict** や、ロジスティック回帰の **rxLogit**、決定木の **rxDTree**、ランダム フォレストの **rxDForest**、ナイーブベイズの **rxNaiveBayes** など、**rx** 接頭辞の付いた関数が含まれていました。これらは、SQL Server 2017 の ML Services の R でも引き続き利用することができ、Python の場合は、これらを Python 化した「**Revoscalepy**」として提供されていて、**rx_predict** や **rx_logit**、**rx_dtree**、**rx_dforest** といった形で利用できます（Python では、全て小文字、rx の後に _ が付きます）。

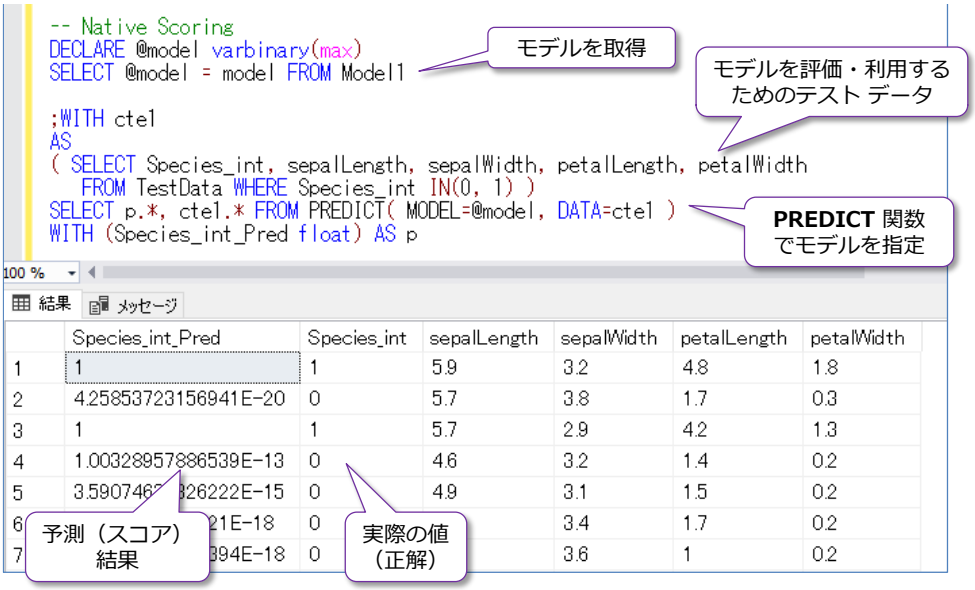
次の画面は、**rx_logit**（ロジスティック回帰）を利用して機械学習をしている場合の例です。



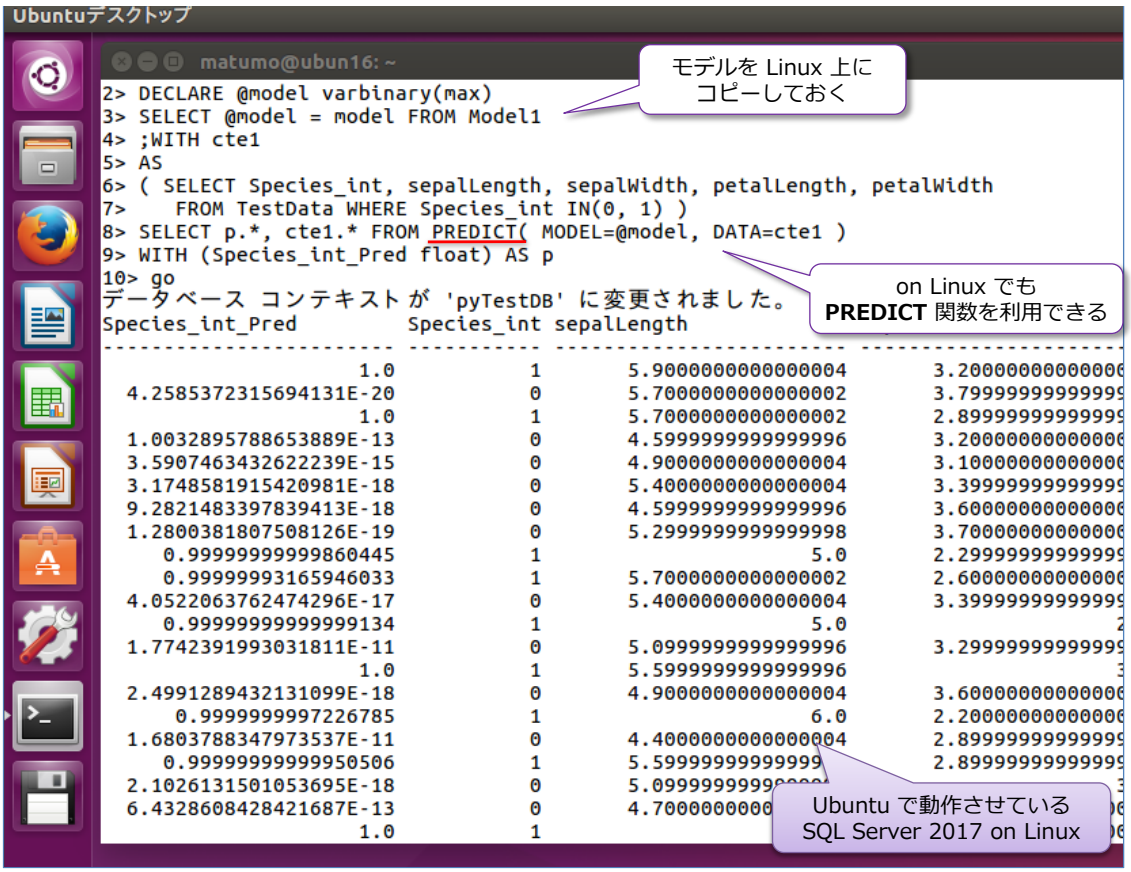
これは、R でお馴染みの iris（あやめ）データを利用して、Species（あやめの種類）を sepal（がく辺）や petal（花弁）の長さや幅で予測するためのモデルを作成しています。

作成したモデルは、**Revoscalepy** の **rx_serialize_model** 関数を利用することで、ネイティブ形式（後述の **PREDICT** 関数で利用できるモデルの形）で保存することができます。SQL Server 2016 の R Services では、保存したモデルは、**rxPredict** 関数を利用して予測を行っていましたが、SQL Server 2017 からは **rxPredict** 関数（Python では rx_predict）を利用しなくても、予測が行える **PREDICT** 関数が提供されました。

PREDICT 関数は、Transact-SQL ステートメントの一部として実行することができるので、この機能はネイティブ スコアリング（Native Scoring）とも呼ばれています。具体的には、次のように利用できます。



この **PREDICT** 関数は、SQL Server 2017 on Linux でもサポートされています。on Linux では、ML Services はサポートされていませんが、PREDICT 関数がサポートされているので、モデルを on Linux 上にコピーしておけば、次のように on Linux 上で予測を行うこともできます。モデルのコピーは、リンク サーバーを利用すれば、簡単に行えます。



CNN（Convolutional Neural Network：畳み込みニューラル ネットワーク）

ML Services の Python を利用すれば、**ディープ ラーニング**の実装も簡単に行えます。ここでは、**Keras**（TensorFlow や Microsoft Cognitive Toolkit、Theano をバックエンドに利用するニューラル ネットワークのライブラリ）を利用して、**CNN**（Convolutional Neural Network：畳み込みニューラル ネットワーク）を実装する例を紹介します。CNN は、画像認識や音声認識で非常によく利用されているニューラル ネットワークで、現在のディープ ラーニングは、ほとんどが CNN がベースになっています。

```
EXECUTE sp_execute_external_script
  @language = N'Python'
,@script = N'
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

train_img_path = InputDataSet["imgPath"]
train_labels = InputDataSet["imgLabel"]
# print(train_data)
# print(train_label)

img_rows, img_cols = 28, 28
num_classes = 10
input_shape = (img_rows, img_cols, 1)

import numpy as np
from PIL import Image
import os

train_imgs = []
# trainLabels = []

for img_path in train_img_path:
    img = Image.open(img_path)
    img = img.convert(''L'')
    img.thumbnail((img_rows, img_cols))
    img = np.array(img, dtype=np.float32)
    img = 1-np.array(img / 255)
    img = img.reshape(img_rows, img_cols, 1)
    train_imgs.append(img)

train_labels = keras.utils.to_categorical(train_labels, num_classes)
```

Keras を利用

CNN（畳み込みニューラル ネットワーク）のモデルを作成

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation=''relu'',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation=''relu''))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation=''relu''))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation=''softmax''))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=[''accuracy''])

epochs = 13
model.fit(np.array(train_imgs), train_labels,
          epochs=epochs)

model.save(''C:¥¥temp¥¥my_model.h5'')
model.save_weights(''C:¥¥temp¥¥my_model_weights.h5'')
#trained_model = model.to_json()
'

,@input_data_1 = N'SELECT imgPath, imgLabel FROM TrainData'
```

入力データ
（モデルの訓練データ）

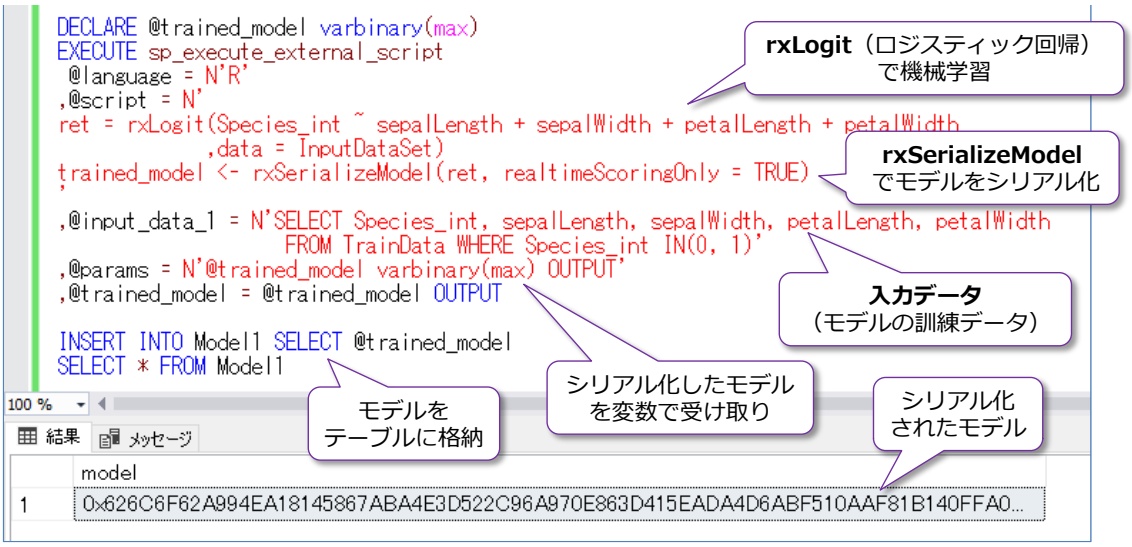
このように、ML Services を利用すれば、通常の Python と同様にプログラミングすることができます。また、入力データに関しては SQL Server から直接取得することができるので、その分のコードを記述する必要がなくなります。

こうした Python を利用した Machine Learning（機械学習）については、本自習書シリーズの No.3「**SQL Server 2017 Machine Learning Services**」編（現在制作中）で詳しく説明しているので、こちらもぜひご覧いただければと思います。

3.3 R を利用した Machine Learning（機械学習）の例

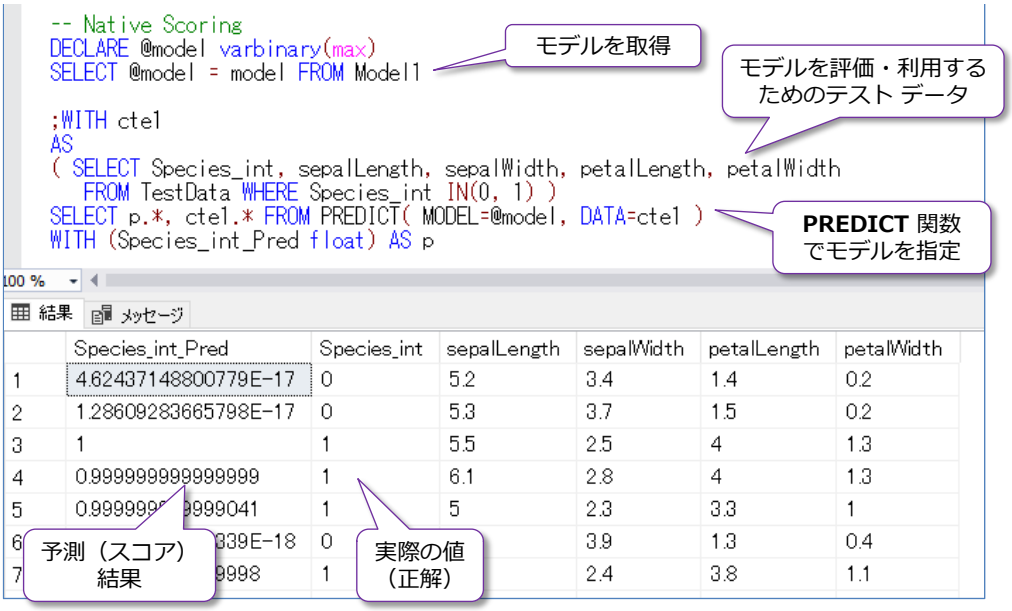
ML Services での R は、SQL Server 2016 の場合と同様に利用することができます。SQL Server 2017 からの新機能としては、Python のところで紹介した **PREDICT** 関数です。

例えば、**RevoScaleR** の **rxLogit**（ロジスティック回帰）を利用して機械学習をする場合は、次のように記述できます。



作成したモデルは、**RevoScaleR** の **rxSerializeModel** 関数を利用することで、ネイティブ形式（**PREDICT** 関数で利用できるモデルの形）で保存できます。

PREDICT 関数は、次のように利用できます。



こうした R を利用した Machine Learning については、本自習書シリーズの No.3「**SQL Server 2017 Machine Learning Services**」編（現在制作中）で詳しく説明しているので、こちらもぜひご覧いただければと思います。

STEP 4. SQL Server 2017 の注目の新機能

この STEP では、SQL Server 2017 で提供された注目の新機能である「**グラフ データベース**」や「**自動チューニング**」、「**Adaptive Query Processing**」、「**クエリ ストアや DTA の強化**」、「**列ストア インデックス／インメモリ OLTP 機能の強化**」、「**再開可能なオンライン インデックス再構築**」などを説明します。

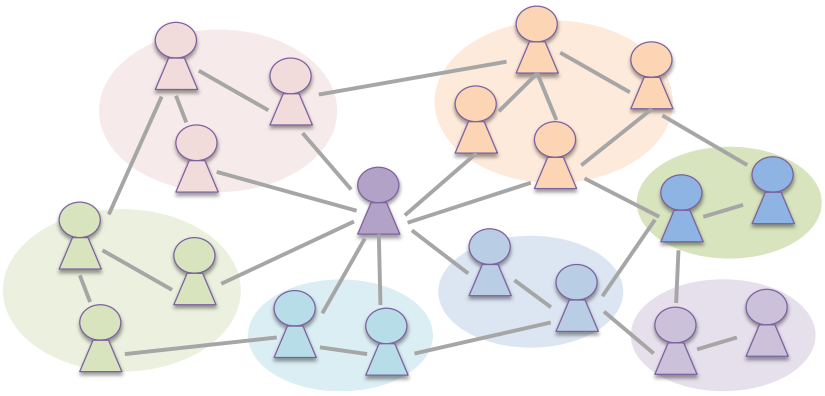
この STEP では、次のことを学習します。

- ✓ グラフ データベース
- ✓ 自動チューニング
- ✓ Adaptive Query Processing
- ✓ クエリ ストアの強化、DTA の強化
- ✓ 列ストア インデックスの強化
- ✓ インメモリ OLTP の強化
- ✓ 再開可能なオンライン インデックス再構築
- ✓ AlwaysOn 可用性グループの強化

4.1 グラフ データベース（Graph Database）

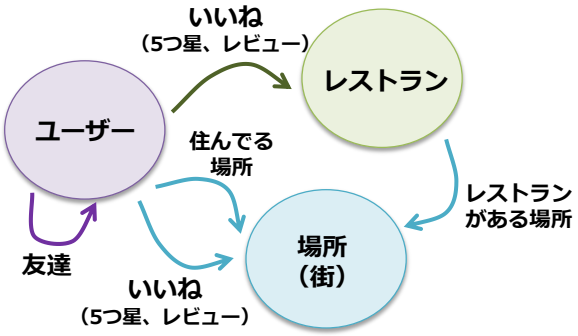
SQL Server 2017 には、**グラフ データベース**機能が提供されています。グラフ データベースは、Facebook や Twitter などの **SNS**（ソーシャル ネットワーキング サービス）における**データの繋がり**（ソーシャル グラフ：人のつながり）を表現するのに適したデータベースです。

ソーシャル グラフ（SNS における人のつながり）



グラフ データベースは、こういった人のつながりや、**多対多**（Many-To-Many）の関係を表現するのが得意なデータベースです。例えば SNS においては、複数のユーザーが、いろいろな場所に住み、いろいろな対象物に興味があるといった関係があります。

SNS での関係

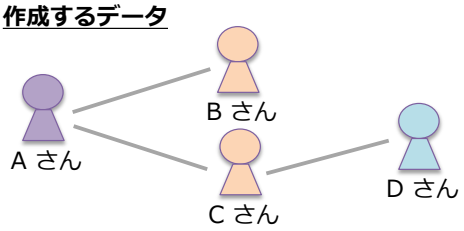


こういったデータを表現しやすいのがグラフ データベースです。

リレーショナル vs. グラフ

ここでは、次のようなユーザー間のつながり（友達関係）に関して、リレーショナル データベースを利用した場合とグラフ データベースを利用した場合でどういった違いがあるのかを説明します。

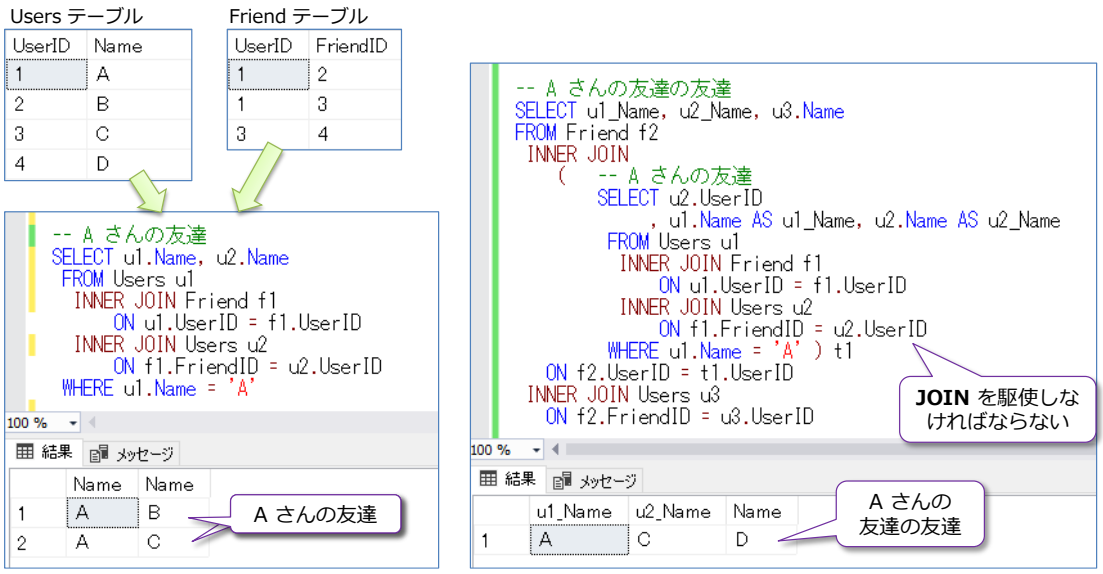
作成するデータ



この図は、A さんの友達は B さんと C さん、C さんの友達は D さんというデータです。

このデータを、グラフ データベースではなく、従来ながらのリレーショナル データベース（RDB）で表現すると、次のようになります。

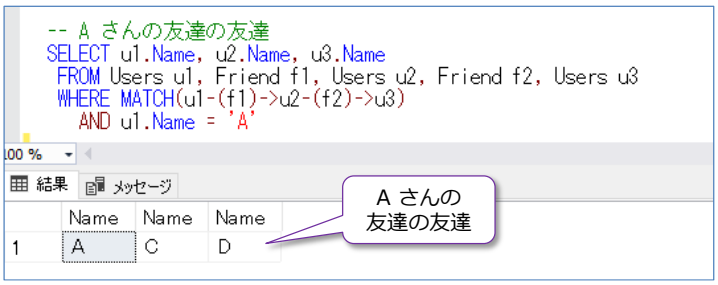
リレーショナル データベースで "つながり" を表現する場合



上記のほかに、CTE（共通テーブル式）を利用する方法もありますが、こうした "つながり" のあるデータを取得するには、リレーショナル データベースの場合は **JOIN** を駆使していかなければなりません。

これに対して、グラフ データベースを利用した場合は、次のように **MATCH** 句を利用して、データを簡単に取得することができます。

グラフ データベースでは MATCH でデータのつながりを簡単に取得できる



Let's Try

それでは、これを試してみましょう。上の例で説明した **Users** と **Friend** をグラフ データベースとして作成してみましょう。

1. まずは、グラフ データベースを試すためのデータベースを作成するために、クエリ エディターで、次のように **CREATE DATABASE** ステートメントを実行して、「**graphTestDB**」という名前のデータベースを作成します。

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

2. 次に、ユーザーを格納するための「**Users**」テーブル、友達関係（繋がり）を格納するための「**Friend**」テーブルを作成しますが、グラフ データベースでは、前者は **NODE**（ノード）、後者は **EDGE**（エッジ）テーブルとして作成します。データの繋がりに相当するテーブルは Edge として作成します。

```
USE GraphTestDB

-- Node テーブルの作成
CREATE TABLE Users
( UserID int PRIMARY KEY
 ,Name varchar(50) ) AS NODE

-- Edge テーブルの作成
CREATE TABLE Friend AS EDGE
```

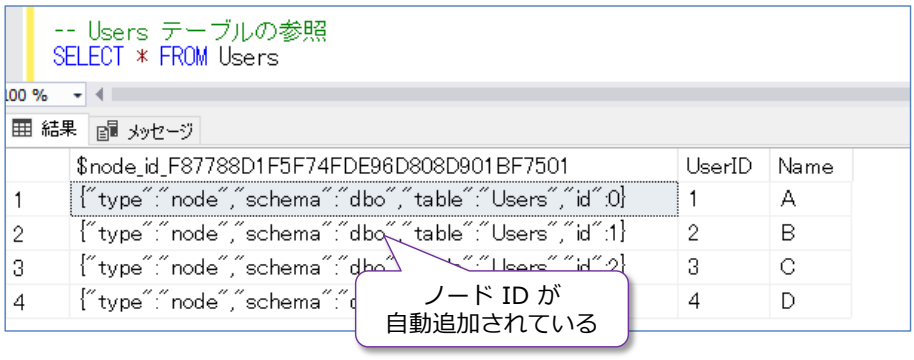
CREATE TABLE の最後に、「**AS NODE**」と付けることで Node テーブル、「**AS EDGE**」と付けることで Edge テーブルを作成することができます。

3. 次に、Node である **Users** テーブルにデータを INSERT します（A さん～D さんを追加）。これは通常のテーブルと同様に行えます。

```
-- Node テーブルにデータを追加（通常のテーブルと同様）
INSERT INTO Users VALUES (1, 'A')
INSERT INTO Users VALUES (2, 'B')
INSERT INTO Users VALUES (3, 'C')
INSERT INTO Users VALUES (4, 'D')
```

4. 次に **Users** テーブルを参照してみます。

```
-- Users テーブルの参照
SELECT * FROM Users
```



「**$node_id_～**」という名前の**ノード ID**（ノードを内部的に管理している ID）列が自動的に追加されていることを確認できます。このノード ID は、「**$node_id**」という名前で参照することができます（SELECT ステートメントの選択リストで指定できます）。

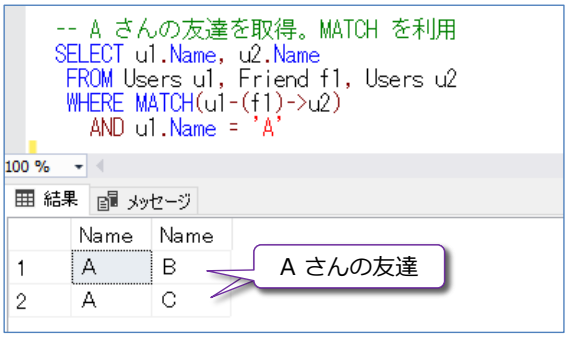
5. 次に、**Edge**（データの繋がり）である「**Friend**」テーブルにデータを **INSERT** します（A さんの友達は B さんと C さん、C さんの友達は D さんとして追加します）。

```
-- Friend テーブルへの INSERT
-- A さんの友達は B さんと C さん
INSERT INTO Friend
 VALUES ( ( SELECT $node_id FROM Users WHERE Name = 'A' ),
          ( SELECT $node_id FROM Users WHERE Name = 'B' ) )

INSERT INTO Friend
 VALUES ( ( SELECT $node_id FROM Users WHERE Name = 'A' ),
          ( SELECT $node_id FROM Users WHERE Name = 'C' ) )

-- C さんの友達は D さん
INSERT INTO Friend
 VALUES ( ( SELECT $node_id FROM Users WHERE Name = 'C' ),
          ( SELECT $node_id FROM Users WHERE Name = 'D' ) )
```

Edge テーブルへのデータの INSERT は、Node テーブルから「**$node_id**」（内部的にノードに対して設定される ID）を取得して、それを指定するようにします。1 つめの INSERT ステートメントでは、WHERE 句に「**WHERE Name='A'**」と指定することで、A さんのノード ID を取得して、「**INSERT INTO Friend VALUES(A さんのノード ID, B さんのノード ID)**」という形で、A さんの友達は、B さんであることを追加しています。

6. 次に、A さんの友達を検索してみます。これには **MATCH** 句を次のように利用します。

```
-- A さんの友達を取得。MATCH を利用
SELECT u1.Name, u2.Name
 FROM Users u1, Friend f1, Users u2
 WHERE MATCH(u1-(f1)->u2)
   AND u1.Name = 'A'
```

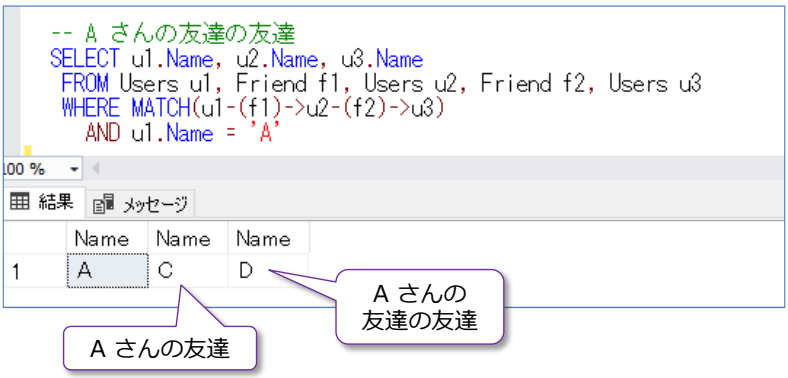


MATCH では「**(u1-(f1)->u2)**」と記述していますが、**u1** と **u2** は **Users**（Node テーブル）に対する別名、**f1** は **Friend**（Edge テーブル）に対する別名です。u1 では、WHERE 句で A さんに絞っていて、「**A さん - (Edge) -> Users**」という形になり、A さんと繋がりのあるデータ（Friend）を Users テーブルから検索（Match するものを探す）という形です。

このように、グラフ データベースを利用すれば、JOIN を記述することなく、繋がりのあるデータを **MATCH** で簡単に取得することができます。

7. 次に、A さんの友達の友達を検索してみましょう。

```
-- A さんの友達の友達
SELECT u1.Name, u2.Name, u3.Name
 FROM Users u1, Friend f1, Users u2, Friend f2, Users u3
 WHERE MATCH(u1-(f1)->u2-(f2)->u3)
   AND u1.Name = 'A'
```



前のクエリとの違いは、「**Friend f2, Users u3**」と「**u2-(f2)->u3**」が追加されている点です。u2 は A さんの友達（B さんと C さん）を取得でき、「**B／C さん - (Edge) -> Users**」という形で、B／C さんと繋がりのあるデータ（Friend）を Users テーブルから検索（Match するものを探す）という形です。

このように、グラフ データベースを利用すれば、データの繋がりを簡単に取得できるようになるので大変便利です。グラフ データベースは、SQL Server のデータベース エンジンに統合された機能なので、通常のテーブルと同様、**列ストア インデックス**を作成することもできます。

その他、グラフ データベースについては、オンライン ブックの以下のトピックが参考になると思います。

Graph processing with SQL Server and Azure SQL Database
https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview

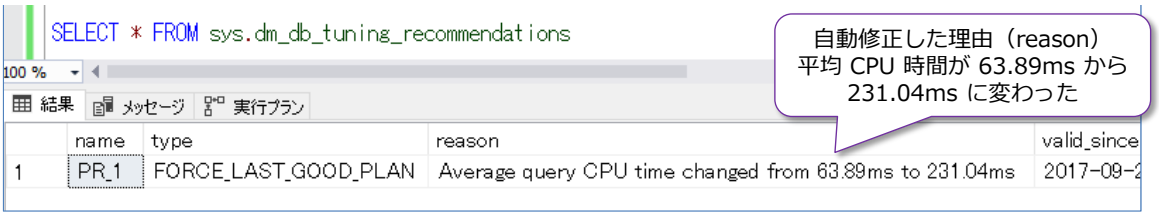
SQL Graph Architecture
https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-architecture

Ignite 2017 Session : Graph extensions in Microsoft SQL Server 2017 and Azure SQL Database
https://myignite.microsoft.com/videos/55108

4.2 自動チューニング（Automatic Tuning）

SQL Server 2017 では、ついに**自動チューニング**機能が提供されました。この機能は、SQL Server 2016 で提供された「**クエリ ストア**」（Query Store）機能を進化させたものです。クエリ ストアでは**クエリの実行履歴**と**実行プラン**を保存することができましたが（SQL Server 2017 からはクエリの**待機**に関する情報も保存可能）、この保存したクエリ履歴を利用して、遅くなったクエリを自動判別して、遅くなる前の正常に実行できていた状態に自動的に戻すことができるのが、SQL Server 2017 における自動チューニング機能です。

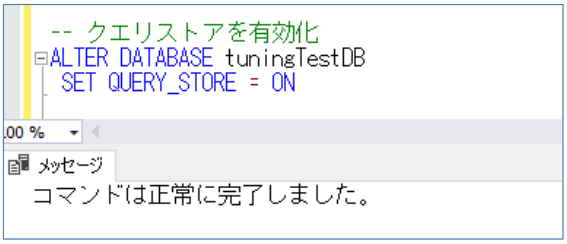
技術的には、クエリの実行プラン（SQL ステートメントをどのように内部実行するのかを、クエリ オプティマイザーが決定した実行計画）を利用して、1 つのクエリに対して複数の実行プランが存在している場合に、その実行プランの性能差を比較して、最適な実行プランを自動的に選択する、という機能です（このため **Automatic Plan Correction**：自動プラン修正とも呼ばれています）。自動的に変更があったクエリは、次のように **dm_db_tuning_recommendations** 動的管理ビュー（DMV）に記録されていて、変更理由も確認することができます。



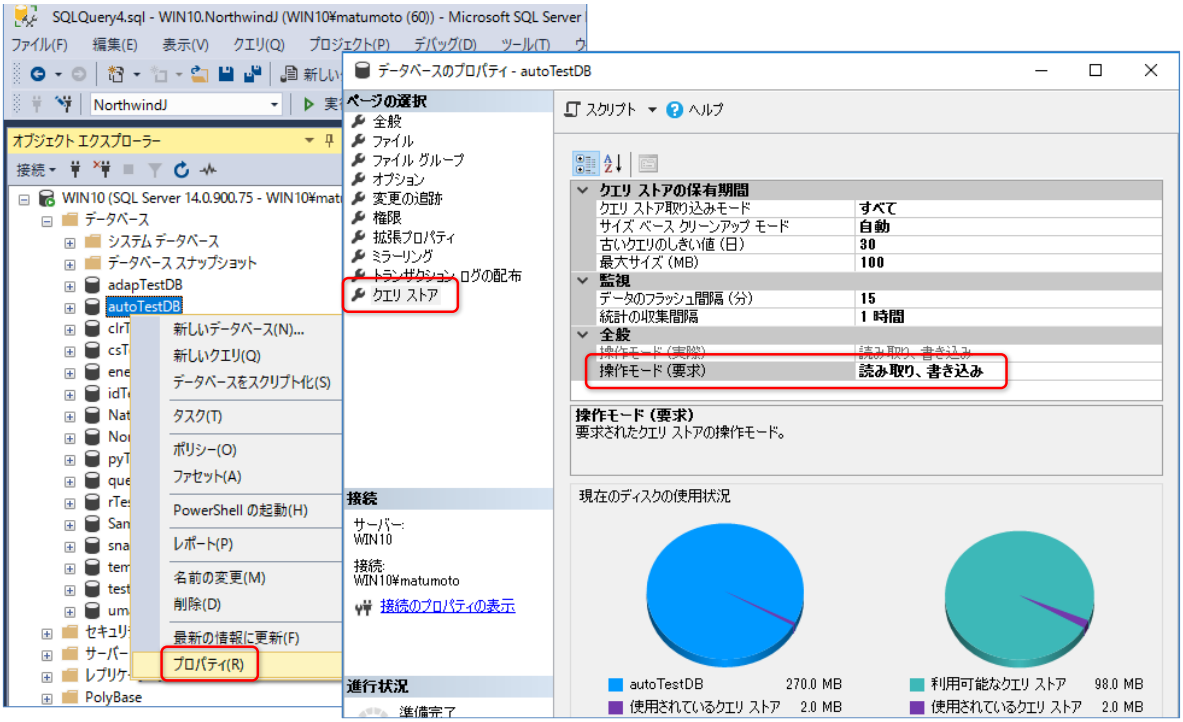
自動チューニングの利用方法

前述したように、自動チューニングは、**クエリ ストア**の機能を利用しているので、まずはクエリ ストアを有効化する必要があります。クエリ ストアを有効化するには、次のように **ALTER DATABASE** ステートメントを実行します（SQL Server 2017 on Linux でも利用することができます）。

```
ALTER DATABASE データベース名
  SET QUERY_STORE = ON
```



クエリ ストアは、Management Studio を利用している場合は、次のようにデータベースのプロパティを開いて、[**クエリ ストア**] ページで [**操作モード（要求）**] を「**読み取り、書き込み**」に変更することでも有効化することができます。



クエリ ストアを有効化すると、**クエリの実行履歴**と**実行プラン**を保存できるようになります（既定では 100MB 分の量を蓄積することができて、上の画面で［**最大サイズ**］を設定すれば、蓄積量を変更することもできます）。また、SQL Server 2017 からは、**クエリの待機（Wait）**に関する情報（クエリがどういったリソースによって内部的な待ちが発生したのか）も保存できるようになっています（詳しくは後述します）。

自動チューニング機能を有効化するには、次のように **ALTER DATABASE** ステートメントを実行します。

```
ALTER DATABASE データベース名
 SET AUTOMATIC_TUNING ( FORCE_LAST_GOOD_PLAN = ON )
```

SET オプションで「**AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON)**」を指定することで、自動チューニングを有効化することができます。この「**FORCE_LAST_GOOD_PLAN**」は、Last（最後の）Good（良かった）Plan（実行プラン）に Force（強制変更）するという意味のオプションです。

Let's Try

それでは、自動チューニング機能を試してみましょう。

1. まずは、自動チューニングを試すためのデータベースを作成します。クエリ エディターで、次のように **CREATE DATABASE** ステートメントを実行して、「**tuningTestDB**」という名前のデータベースを作成します。

```
CREATE DATABASE tuningTestDB
```

2. 次に、データベースに対して**クエリ ストア**機能を有効化します。

```
-- クエリストアを有効化
ALTER DATABASE tuningTestDB
 SET QUERY_STORE = ON
```

クエリ ストアを有効化すれば、自動チューニング機能を有効化できるようになりますが、ここではあえて、有効化をしない場合の動作を確認するために、自動チューニング機能は無効（既定値）のまま、先に進めます。

3. 次に、データベース内にテーブルを 2 つ作成します（**t1** と **t2** という名前で作成します）。

```
-- データベース内にテーブルを 2つ作成
USE tuningTestDB
CREATE TABLE t1
( a int IDENTITY PRIMARY KEY
 ,b int )

CREATE TABLE t2
( a int IDENTITY PRIMARY KEY
 ,b int )
```

どちらも「**a**」列と「**b**」列の 2 列のみで、int データ型、「**a**」列は IDENTITY で 1 からの連番を割り振って、PRIMARY KEY 制約に設定します。これによって、「**a**」列には自動的にクラスター化インデックスも作成されます。

4. 次に、t1 テーブルと t2 テーブルにデータを **100 万件**ずつ **INSERT** します。次のように **WHILE** ループを利用してデータを追加します（**b** 列には「**100**」という値を固定値として指定しています）。

```
-- データを 100万件ずつ INSERT する
SET NOCOUNT ON
BEGIN TRAN
    DECLARE @i int = 1
    WHILE @i <= 1000000
      BEGIN
        INSERT INTO t1 VALUES(100)
        INSERT INTO t2 VALUES(100)
        SET @i += 1
      END
COMMIT
SET NOCOUNT OFF
```

5. 100 万件のデータの追加が完了したら、次に、「**b**」列に「**999**」という値を指定して、**1 件**ずつ **t1** と **t2** テーブルにデータを INSERT します。

```
-- 999 を 1件だけ追加
INSERT INTO t1 VALUES(999)
INSERT INTO t2 VALUES(999)
```

999 を 1 件だけ INSERT して、意図的に「**b**」列に偏りのあるデータがあるようにしています。

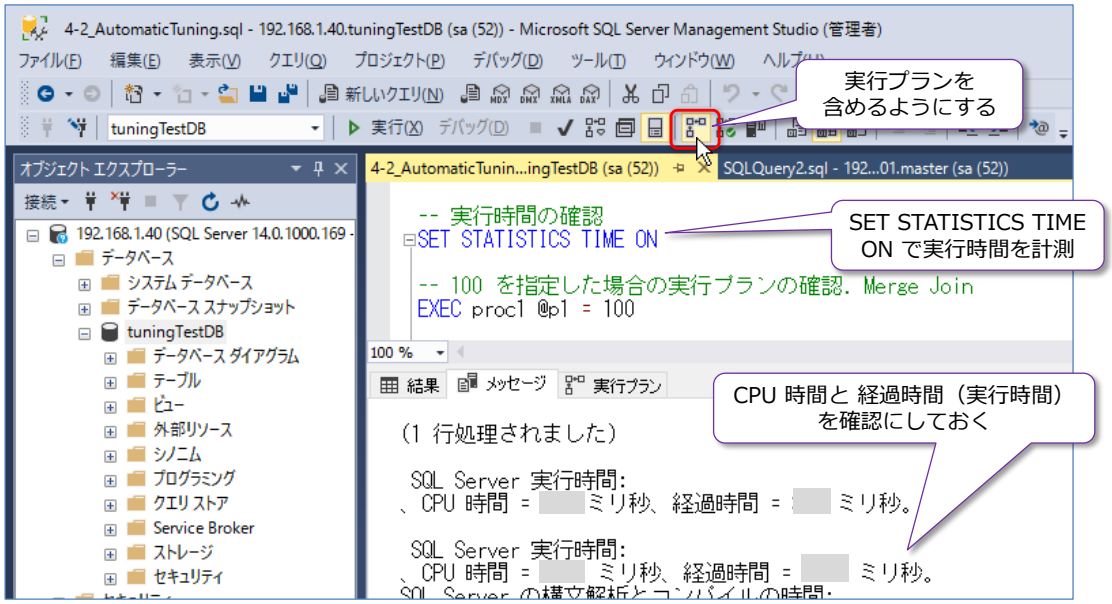
6. 次に、**t1** と **t2** テーブルを **JOIN** するストアド プロシージャを作成します。

```
-- ストアドプロシージャの作成。proc1 という名前
CREATE PROC proc1
@p1 int
AS
SELECT COUNT(*)
 FROM t1 INNER JOIN t2
  ON t1.a = t2.a
  WHERE t1.b = @p1
go
```

ストアド プロシージャの名前は「**proc1**」として、入力パラメーターとして「**@p1**」、**t1** と **t2** テーブルを **INNER JOIN**（内部結合）して、WHERE 句では「**t1.b = @p1**」のようにパラメーターで与えられた値で、「**b**」列の絞り込みを行うようにします。

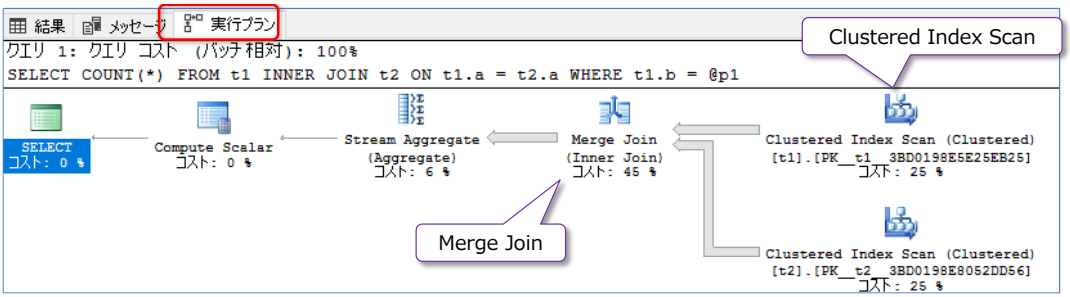
7. 次に、作成した「**proc1**」ストアド プロシージャを実行してみて、実行時間と実行プランを確認してみます。入力パラメーターには「**100**」を与えて実行します。

```
-- 実行時間の確認
SET STATISTICS TIME ON

-- 100 を指定した場合の実行プランの確認
EXEC proc1 @p1 = 100
```



「**SET STATISTICS TIME ON**」を付けて実行時間を計測するようにして、ツールバーの［**実際の実行プランを含める**］ボタンをクリックして、実行プランを確認できるようにしてから、ストアド プロシージャを実行します。

入力パラメーターで与えた「**100**」に相当するデータは、「**t1.b = 100**」で 100 万件ものデータになるので、実行プランには **Clustered Index Scan**（全データのスキャン）が選択されて、**t1** と **t2** テーブルの JOIN には **Merge Join** が採用されています。



JOIN では、結合するテーブルの両方のデータ量が多い場合には、Merge または Hash Join が採用されることが多くなります。

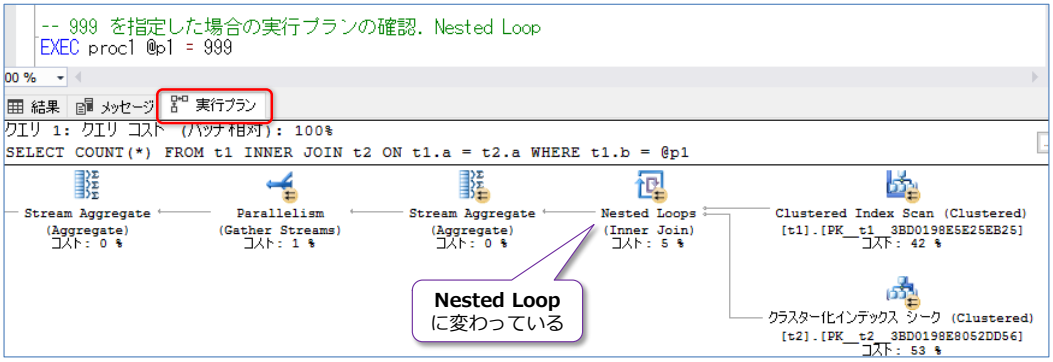
8. 次に、実行プランをクリアするために、次のように **ALTER DATABASE SCOPED CONFIGURATION** ステートメントを実行して**プロシージャ キャッシュをクリア**します。

```
-- プロシージャ キャッシュのクリア
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
```

なお、SQL Server 2014 以前のバージョンでは、「**DBCC FREEPROCCACHE**」コマンドを実行することで、プロシージャ キャッシュをクリアすることができましたが、上記のように **ALTER DATABASE SCOPED CONFIGURATION**（データベース スコープ構成の変更）ステートメントを利用することで、接続中のデータベースに関するキャッシュ情報のみをクリアすることができます。

9. プロシージャ キャッシュのクリアが完了したら、今度は、入力パラメーターに「**999**」を与えてストアド プロシージャを実行します。

```
-- 999 を指定した場合の実行時間と実行プランの確認
EXEC proc1 @p1 = 999
```



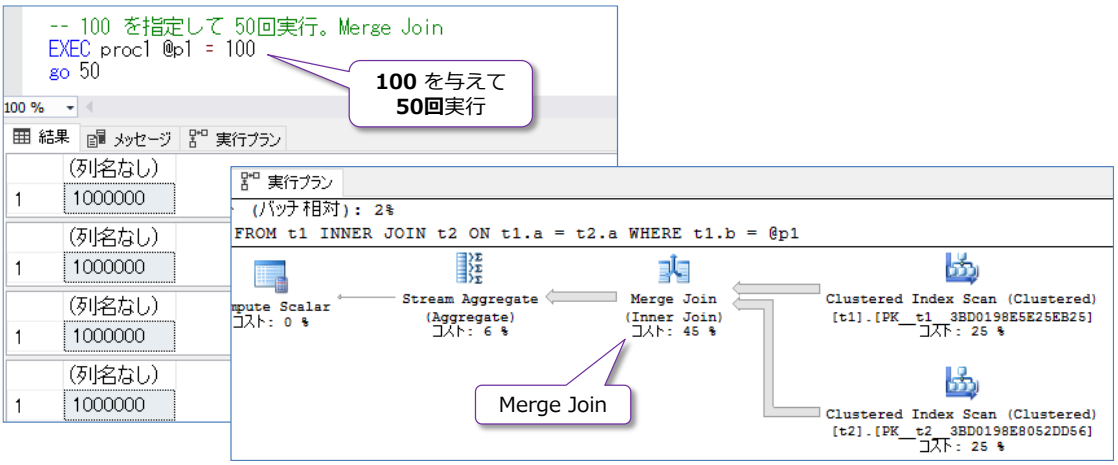
入力パラメーターで与えた「**999**」に相当するデータは、「**t1.b = 999**」で、たったの 1 件のみなので、**t1** と **t2** テーブルの JOIN が **Nested Loop** に変わっていることを確認できます。JOIN では、結合するテーブルのうち、どちらかのデータ量が小さい場合には、Nested Loop が採用されることが多くなります。

10. 続いて、もう一度プロシージャ キャッシュをクリアします。

```
-- プロシージャ キャッシュのクリア
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
```

11. プロシージャ キャッシュのクリアが完了したら、今度は、入力パラメーターに「**100**」を与えてストアド プロシージャを **50 回**実行します（次のように **go 50** と指定することで、50 回実行することができます）。

```
-- 100 を指定して 50回実行。Merge Join
EXEC proc1 @p1 = 100
go 50
```



実行プランは、**Merge Join** になることを確認しておきます。また、各実行ごとの CPU 時間と実行時間も確認しておきます。もし、Merge Join にならない場合は、プロシージャ キャッシュを再度クリアしてから、もう一度 50 回ストアド プロシージャを実行してみてください。

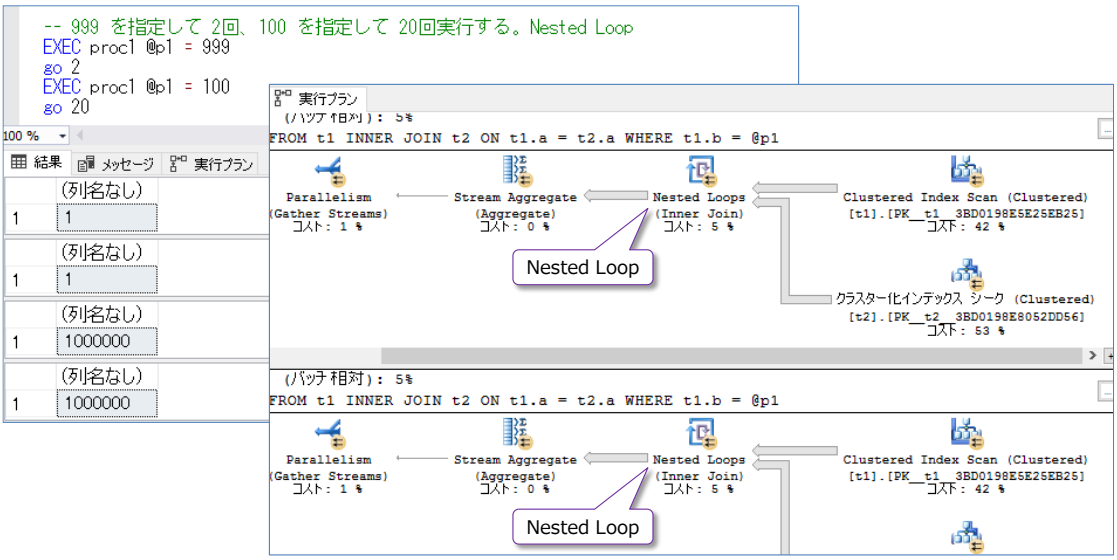
12. 次に、プロシージャ キャッシュをクリアします。

```
-- プロシージャ キャッシュのクリア
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
```

13. 続いて、今度は、ストアド プロシージャの入力パラメーターに「**999**」を与えて **2 回**、「**100**」を与えて **20 回**実行します。

```
-- 999 を指定して 2回、100 を指定して 20回実行する。Nested Loop
EXEC proc1 @p1 = 999
go 2
```

```
EXEC proc1 @p1 = 100
go 20
```



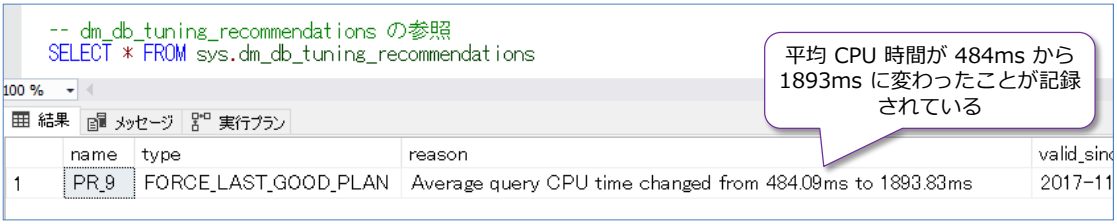
実行プランが **Nested Loop** に変わることを確認しておきます。また、各実行ごとの CPU 時間と実行時間も確認しておきます。100 を与えた場合の CPU 時間と実行時間は、Merge Join を利用したときよりも遅くなっていることを確認できると思います。もし、Nested Loop に変わっていない場合は、プロシージャ キャッシュを再度クリアしてから、もう一度同じようにストアド プロシージャを実行してみてください。

dm_db_tuning_recommendations ビューの参照

クエリ ストアを有効にしていると、1 つのクエリに対して実行プランが複数あって、それらの実行プランの実行時間／CPU 時間に大きな差がある場合に、それを記録しておく機能があります（これは SQL Server 2017 からの新機能です）。記録されたものは、**dm_db_tuning_recommendations** 動的管理ビューで参照することができます。

1. **dm_db_tuning_recommendations** 動的管理ビューは、次のように参照できます。

```
-- dm_db_tuning_recommendations の参照
SELECT * FROM sys.dm_db_tuning_recommendations
```



この例では、平均 CPU 時間（Average CPU time）が **484ms** から **1893ms**（1.9 秒）に変わったクエリがあることが記録されていることを確認できます。このクエリは、上の手順で実行してきた「**proc1**」によるものですが、どういったクエリかどうかなどを取得する方法

（query_id からクエリ情報を取得可能）については、後述します。

もし、このビューの結果が空になっている場合は、実行時間に大きな差がなかったとみなされているので、もう一度、プロシージャ キャッシュをクリアしてから、再度同じ手順を実行してみてください。それでも空の場合は、データ件数をさらに 100 万件追加して試してみてください。

この後は、**自動チューニング**機能を有効化しますが、自動チューニングでは、このビューにリストされるクエリに対して、実行プランを自動修正することを実施してくれます。この例では、Merge Join を利用した実行プランのほうが 484ms で実行できたのに対して、Nested Loop だと 1,893ms に遅くなってしまっているので、Merge Join の実行プランを利用するように自動修正してくれるのが、自動チューニング機能になります。

自動チューニング機能の有効化

次に、**自動チューニング**機能を有効化してみましょう。

1. 自動チューニングを有効化するには、次のように **ALTER DATABASE** ステートメントを実行します。

```
-- 自動チューニングの有効化
ALTER DATABASE データベース名
 SET AUTOMATIC_TUNING ( FORCE_LAST_GOOD_PLAN = ON )
```

2. 次に、クエリ ストアの情報をリセットするために、次のように **ALTER DATABASE** ステートメントを実行して、クエリ ストアをクリアします（**QUERY_STORE CLEAR ALL** を実行）。

```
-- クエリ ストアを完全にクリア
ALTER DATABASE tuningTestDB SET QUERY_STORE CLEAR ALL
-- プロシージャ キャッシュのクリア
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
```

また、プロシージャ キャッシュもクリアしておきます。

3. クエリ ストアとプロシージャ キャッシュのクリアが完了したら、前の手順と同じようにストアド プロシージャを実行します。まずは、入力パラメーターに「**100**」を与えてストアド プロシージャを **50 回**実行します。

```
-- 100 を指定して 50回実行。Merge Join
EXEC proc1 @p1 = 100
go 50
```

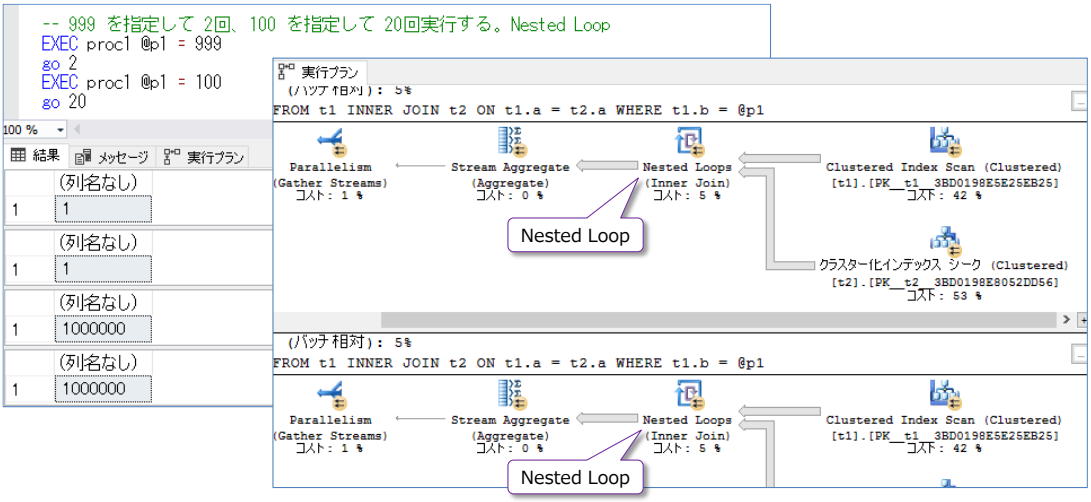
これの実行プランは、**Merge Join** になることを確認しておきます。

4. 次に、プロシージャ キャッシュをクリアします。

```
-- プロシージャ キャッシュのクリア
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE
```

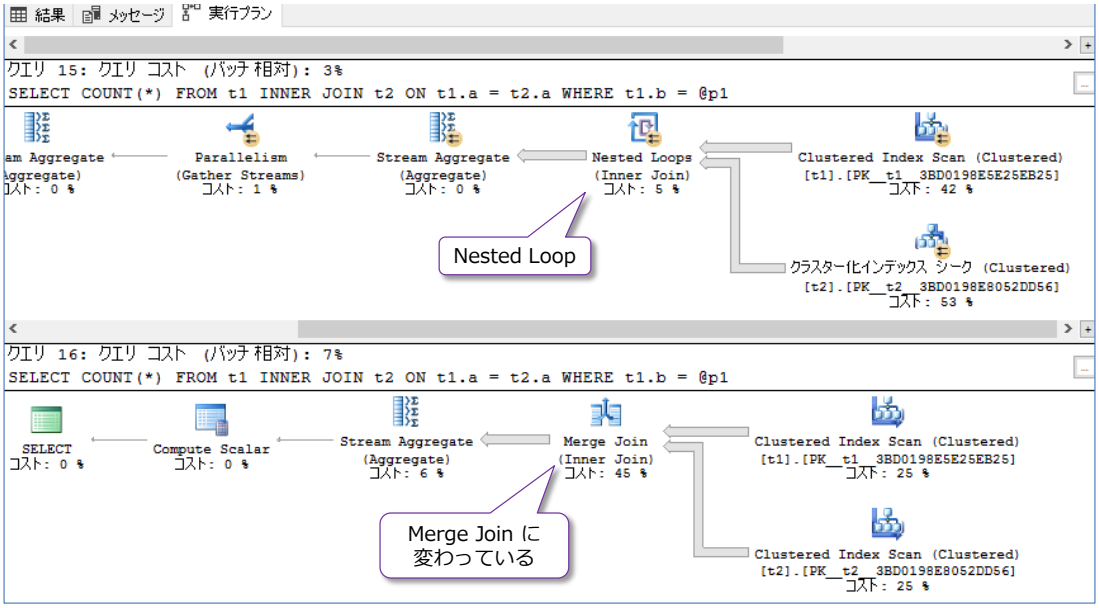
5. 続いて、今度は、ストアド プロシージャの入力パラメーターに「**999**」を与えて **2 回**、「**100**」を与えて **20 回**実行します。

```
-- 999 を指定して 2回、100 を指定して 20回実行する。Nested Loop
EXEC proc1 @p1 = 999
go 2
EXEC proc1 @p1 = 100
go 20
```



この実行プランは **Nested Loop** に変わることを確認して、各実行ごとの CPU 時間と実行時間も確認しておきます。

この結果を、下にスクロールしていくと、次のように **Nested Loop** から **Merge Join** に自動的に変わっている場所を見つけることができます。



自動チューニングを有効化していると、Nested Loop の実行では遅くなると判断して、自動的に Merge Join を利用する実行プランに自動修正してくれます。

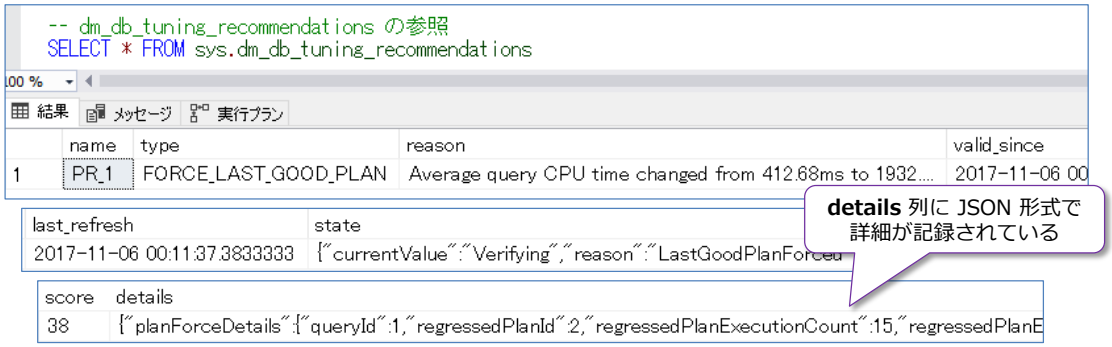
もし、Merge Join に変わっていない場合は、プロシージャ キャッシュを再度クリアしてから、もう一度同じようにストアド プロシージャを実行してみてください。なお、マシン スペックが良い場合や、環境によっては、両者のクエリに大きな差が出ない場合もあります。この場合は、次の手順の **dm_db_tuning_recommendations** 動的管理ビューを参照して、クエリが記録されているかどうかを確認してみてください。大きな差がないと判断された場合には、自動修正する必要はないと判断されてクエリが記録されない形になっています。

dm_db_tuning_recommendations ビューの参照

自動チューニングによって、自動修正されたクエリは、**dm_db_tuning_recommendations** 動的管理ビューで参照することができます。

1. **dm_db_tuning_recommendations** 動的管理ビューを参照してみます。

```
-- dm_db_tuning_recommendations の参照
SELECT * FROM sys.dm_db_tuning_recommendations
```



この結果を一番右までスクロールすると、**details** 列があり、この中に JSON 形式で詳細が記録されていることを確認できます。ここには、「**"queryId":1**」（クエリ ID）や「**"regressedPlanId":2**」（遅くなった実行プランの ID）、「**"recommendedPlanId":1**」（推奨／自動変更した実行プランの ID）、「**"script":"exec sp_query_store_force_plan @query_id = 1, @plan_id = 1**"」（自動修正に利用されたスクリプト＝プラン強制を実施するスクリプト）などが記録されています。

Note： もし結果に何も表示されない場合

前述したように、dm_db_tuning_recommendations 動的管理ビューでは、大きな差があったと判断されたクエリのみ（自動修正する必要があると判断されたクエリのみ）が記録されています。もし、本文中の手順で、クエリが表示されない場合は、データ件数を増やしてみたり、以下の URL のクエリなどを試してみてください。

Demo: Identify and fix plan change regression in SQL Server 2017 RC1
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/07/20/demo-identify-and-fix-plan-change-regression-in-sql-server-2017-rc1/

2. **details** 列は、次のように **JSON_VALUE** 関数を利用して、分解することができます。

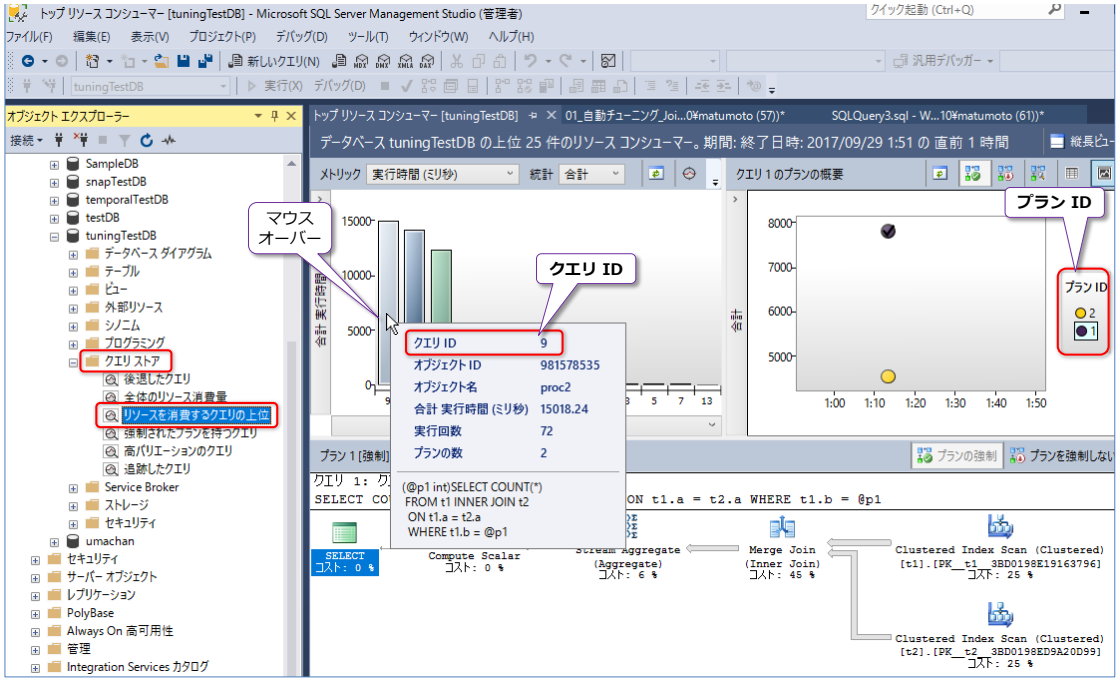
```
-- details 列を分解
SELECT  name, reason,
    JSON_VALUE(details, '$.planForceDetails.queryId') as query_id,
    JSON_VALUE(details, '$.planForceDetails.regressedPlanId') as regressedPlanId,
    JSON_VALUE(details, '$.planForceDetails.recommendedPlanId') as recommendedPlanId,
    JSON_VALUE(details, '$.implementationDetails.script') as script
FROM sys.dm_db_tuning_recommendations
```

```
-- details 列を分解
SELECT  name, reason,
        JSON_VALUE(details, '$.planForceDetails.queryId') as query_id,
        JSON_VALUE(details, '$.planForceDetails.regressedPlanId') as regressedPlanId,
        JSON_VALUE(details, '$.planForceDetails.recommendedPlanId') as recommendedPlanId,
        JSON_VALUE(details, '$.implementationDetails.script') as script
FROM sys.dm_db_tuning_recommendations
```

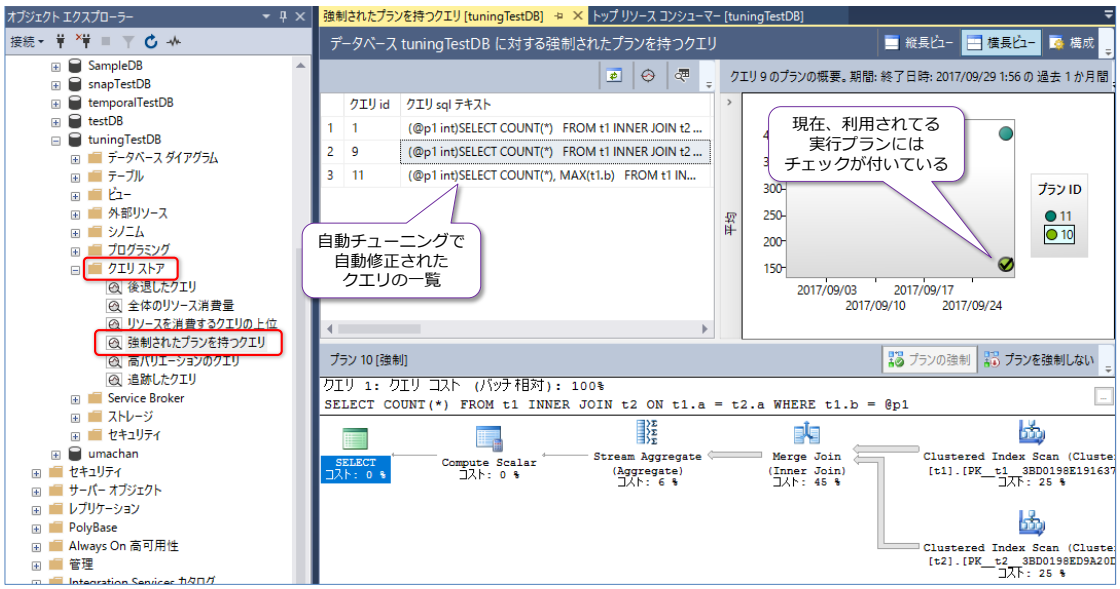
100 % 結果 メッセージ 実行プラン

	name	reason	query_id	regressedPlanId	recommendedPlanId	script
1	PR_1	Average query CPU time changed from 48.72ms to 221.04ms	1	2	1	exec sp_q
2	PR_9	Average query CPU time changed from 66.62ms to 328.61ms	9	11	10	exec sp_q
3	PR_11	Average query CPU time changed from 73.24ms to 300.96ms	11	14	13	exec sp_q

3. **クエリ ID** や**実行プランの ID** は、クエリ ストアを参照することで確認することができます。次のように［**クエリ ストア**］フォルダーの［**リソースを消費するクエリの上位**］をクリックして確認できます。

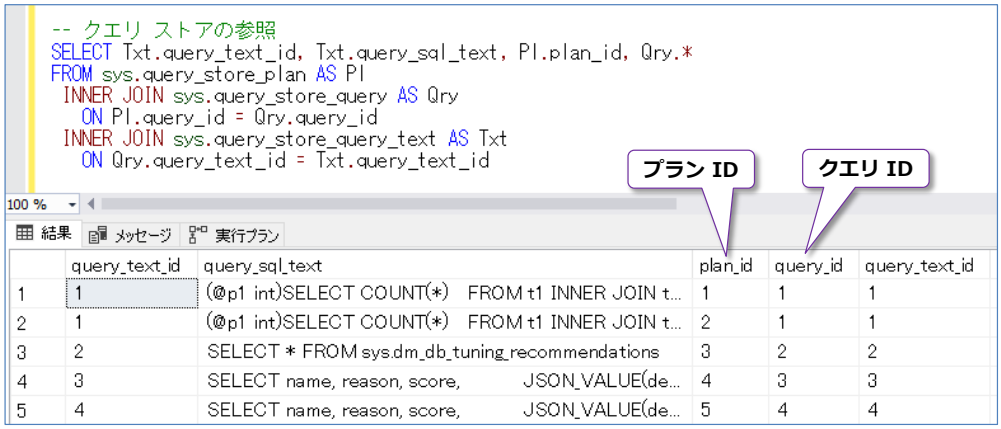


また、自動チューニングによって、自動修正されたクエリは、内部的にはプラン強制が実行されているので、［**クエリ ストア**］フォルダーの［**強制されたプランを持つクエリ**］をクリックすることで確認することもできます。



4. **クエリ ID** や**実行プランの ID** は、クエリ ストアを参照することができる動的管理ビューである「**query_store_query**」と「**query_store_plan**」、「**query_store_query_text**」を利用して、次のように確認することもできます。

```
-- クエリ ストアの参照
SELECT Txt.query_text_id, Txt.query_sql_text, Pl.plan_id, Qry.*
 FROM sys.query_store_plan AS Pl
  INNER JOIN sys.query_store_query AS Qry
    ON Pl.query_id = Qry.query_id
  INNER JOIN sys.query_store_query_text AS Txt
    ON Qry.query_text_id = Txt.query_text_id
```



	query_text_id	query_sql_text	plan_id	query_id	query_text_id
1	1	(@p1 int)SELECT COUNT(*) FROM t1 INNER JOIN t...	1	1	1
2	1	(@p1 int)SELECT COUNT(*) FROM t1 INNER JOIN t...	2	1	1
3	2	SELECT * FROM sys.dm_db_tuning_recommendations	3	2	2
4	3	SELECT name, reason, score, JSON_VALUE(de...	4	3	3
5	4	SELECT name, reason, score, JSON_VALUE(de...	5	4	4

以上のように、SQL Server 2017 では、ついに自動チューニング機能が実装されました。これまでの SQL Server では、実行プランを修正したい場合に、手動でプラン強制を行う必要がありましたが、SQL Server 2017 で自動チューニングを有効化すれば、変化のあった実行プランを自動修正してくれます。

また、この手順で試したように、自動チューニングを有効化しなくても、クエリ ストアを有効化しておくだけで、**dm_db_tuning_recommendations** 動的管理ビューに推奨（自動チューニング

が有効の場合は自動修正されるクエリ）を記録しておいてくれるので、いきなり自動チューニングには頼りたくはない... という方は、クエリ ストアを有効化してみて、どういったクエリがリストされるかを確認してみるのがお勧めです。

クエリ ストアでは、後述のクエリに関する待機（Wait）に関する情報も、SQL Server 2017 から記録されるようになったので、ぜひ活用してみてください。

自動チューニングについては、以下の URL が参考になると思います。

Automatic tuning
https://docs.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning

sys.dm_db_tuning_recommendations (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-tuning-recommendations-transact-sql

Demo: Identify and fix plan change regression in SQL Server 2017 RC1
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/07/20/demo-identify-and-fix-plan-change-regression-in-sql-server-2017-rc1/

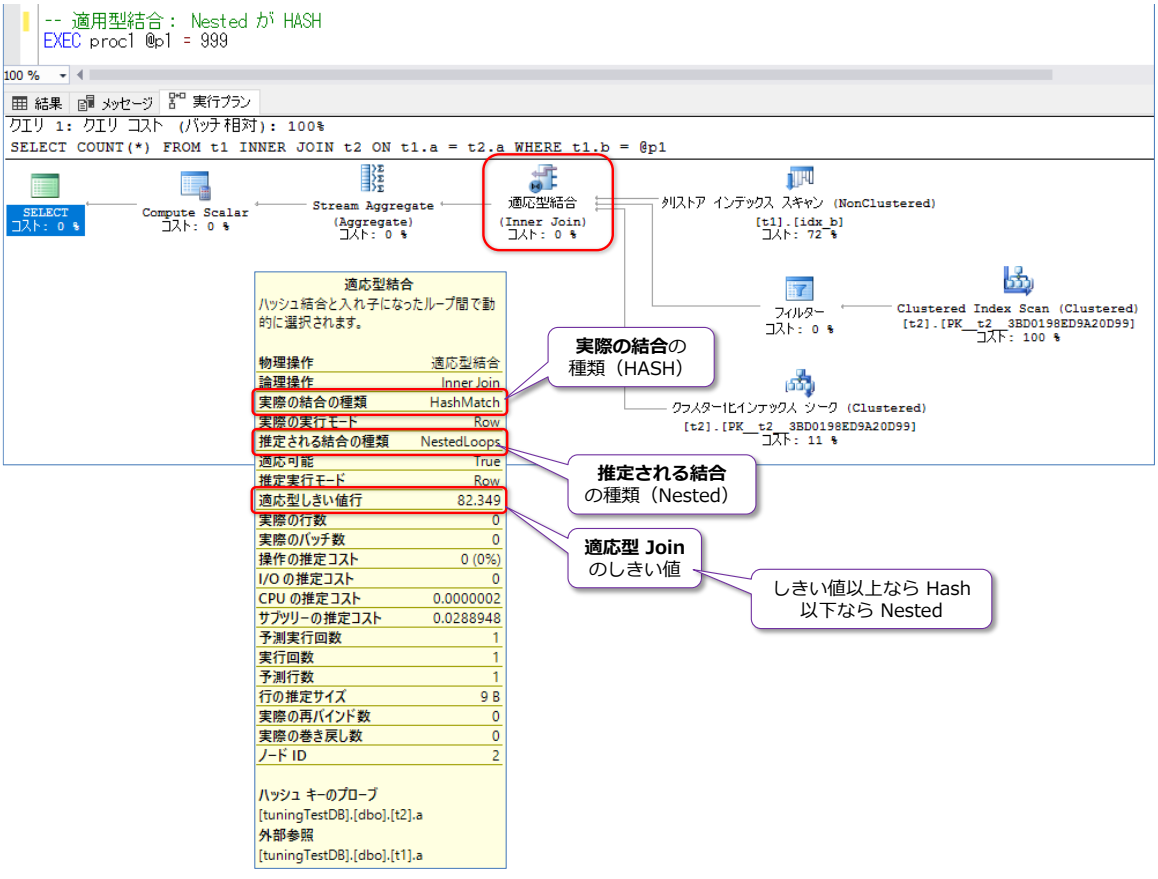
4.3 Adaptive Query Processing（適応型クエリ処理）

SQL Server 2017 では、**Adaptive Query Processing**（適応型クエリ処理）というインテリジェントな、性能向上のためのクエリ処理機能が提供されています。これは、列ストア インデックス（Column-store Index）を作成していて、後述の互換性レベル「**140**」を利用している場合に、利用することができ、次の機能が提供されています。

- Batch Mode Adaptive Join（バッチ モードでの適応型 Join）
- Batch Mode Memory Grant Feedback（バッチモードでのメモリ割当てフィードバック）
- Interleaved Execution for MSTVF （Multi-Statement Table Valued Function）

Batch Mode Adaptive Join（バッチ モードでの適応型 Join）

列ストア インデックスを作成している場合は、**適応型 Join**（**Adaptive Join**）という内部処理で性能向上を実現できる場合があります。



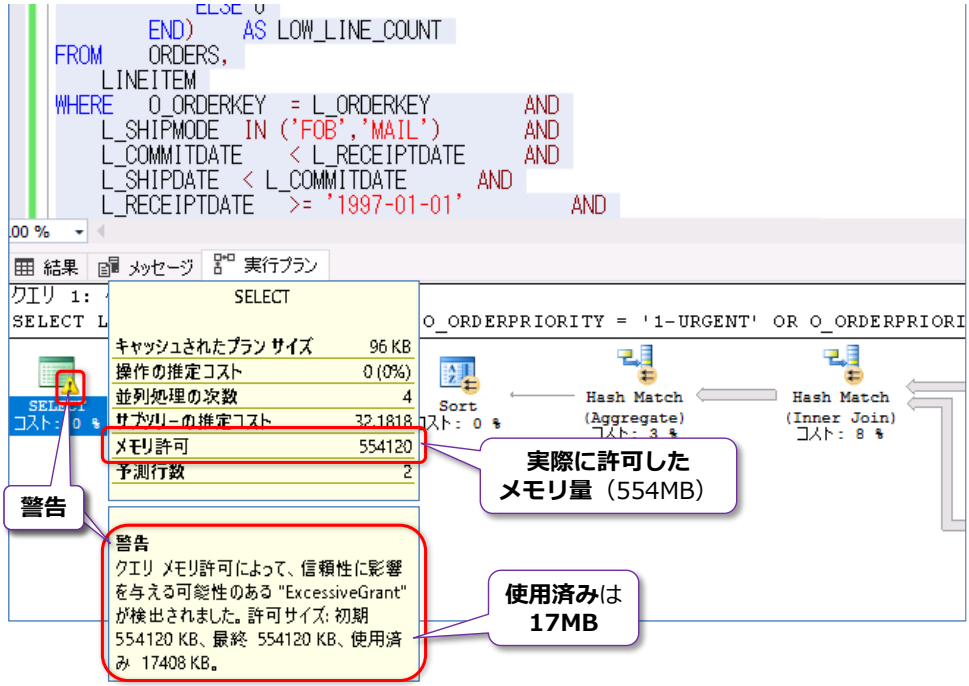
Join では、Nested Loop や Hash Join などが通常選択されますが、Nested Loop は片方のテーブルのデータ量が小さい場合、Hash Join は両方のテーブルのデータ量が多い場合に効率の良い Join 方式です。しかし、クエリに与える入力パラメーターによっては、片方のテーブルのデータが小さくなったり、大きくなったりするのは、実際の現場のデータでは良く起こりえます。

そういった場合に、**適応型 Join** が採用されれば、データ量が少ない場合に Nested Join、データ

量が多い場合に Hash Join を**自動選択**してくれる（実行プランとしては、適応型 Join という形でどちらの Join にも対応／変化できるようになっている）機能です。

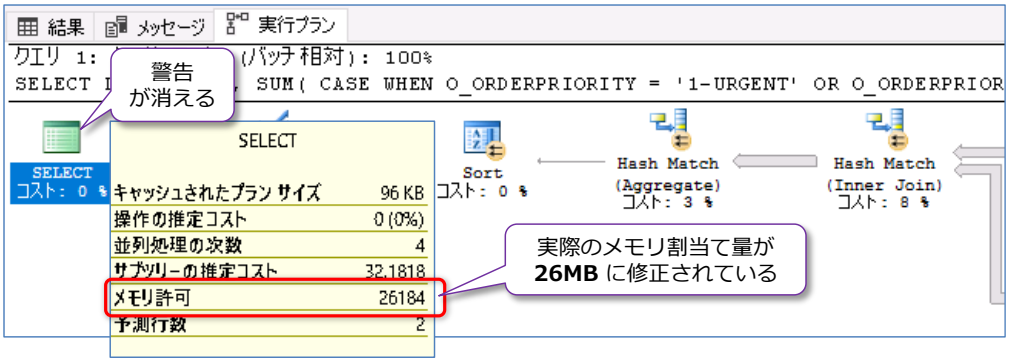
Batch Mode Memory Grant Feedback（メモリ割り当てフィードバック）

これも列ストア インデックスを利用している場合に、インテリジェントに動作する機能で、**クエリのメモリ割り当て**に誤認識があった場合に、2 回目の実行以降に自動修正されるというものです。例えば、メモリ割り当てに誤認識がある場合は、次のように実行プランに**警告**が表示されます。



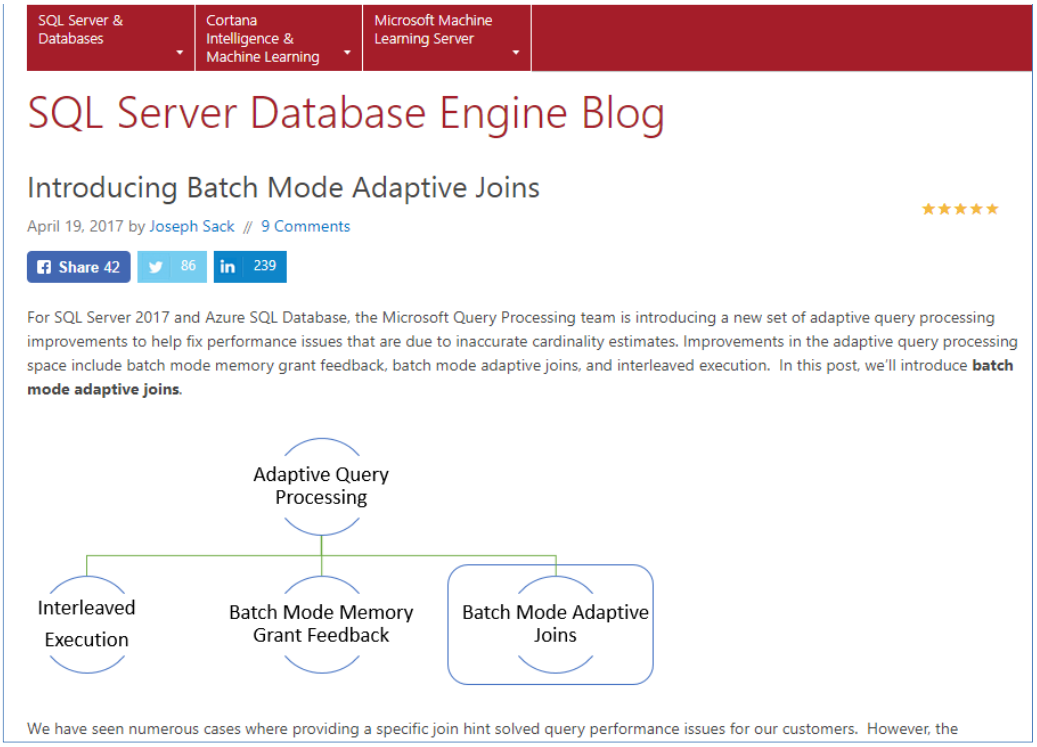
警告に表示されている「**使用済み**」という部分が、実際に使用したメモリ量（画面は **17MB**）になりますが、**実際に許可（Grant）したメモリ量**とは大きな差がある場合（画面は **554MB**）を警告してくれています。このような警告がある場合でも、2 回目以降の同じクエリの実行時には、**Memory Grant Feedback**（メモリ割り当て量の修正）機能が効いて、メモリ割り当て量を自動修正してくれます（互換性レベルで **140** を利用している必要あり）。

2回目の実行時にはメモリ割当て量が自動修正される（互換性レベル 140 を利用している場合）



Adaptive Query Processing については、SQL Server チームの以下のブログ記事が参考になると思います。

Introducing Batch Mode Adaptive Joins
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/04/19/introducing-batch-mode-adaptive-joins/



動画： SQL Server 2017 Adaptive QP
https://channel9.msdn.com/Shows/Data-Exposed/SQL-Server-2017-Adaptive-QP?term=sql%20server%202017

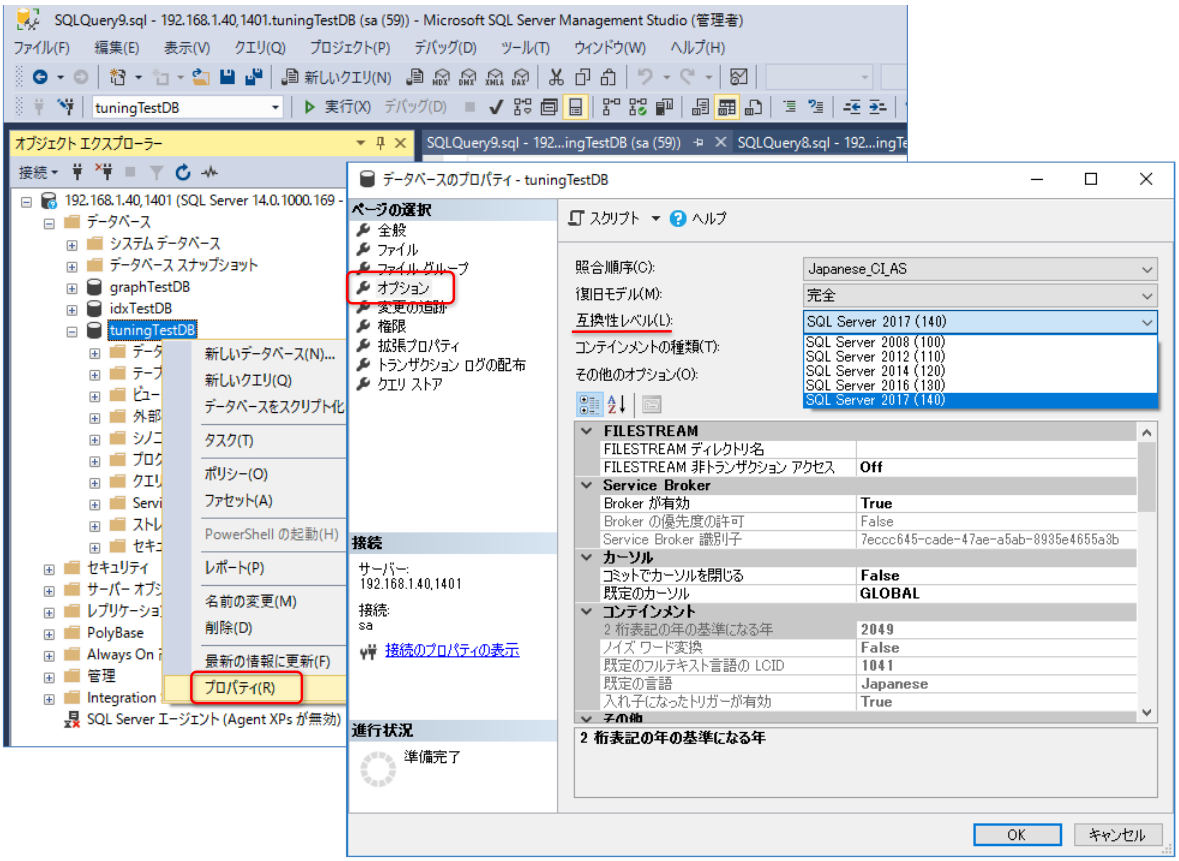
Introducing Interleaved Execution for Multi-Statement Table Valued Functions
https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/04/19/introducing-interleaved-execution-for-multi-statement-table-valued-functions/

Performance implications of using multi-Statement TVFs with optional parameters
https://blogs.msdn.microsoft.com/sqlcat/2017/10/16/performance-implications-of-using-multi-statement-tvfs-with-optional-parameters/

4.4 データベースの互換性レベル 140

SQL Server 2017 では、データベースを新しく作成した場合の**互換性レベル**の既定値は「**140**」（SQL Server 2017 レベル）になります。SQL Server 2017 は、内部的なバージョン番号が「**14.0**」なので、**140** という数字になります。

データベースの互換性レベルを確認／変更するには、Management Studio でデータベースを右クリックして、［**プロパティ**］をクリックし、［**オプション**］タブを開きます。



あるいは、次のように **ALTER DATABASE** ステートメントを実行しても、互換性レベルを変更することができます。

```
ALTER DATABASE データベース名
  SET COMPATIBILITY_LEVEL = 140
```

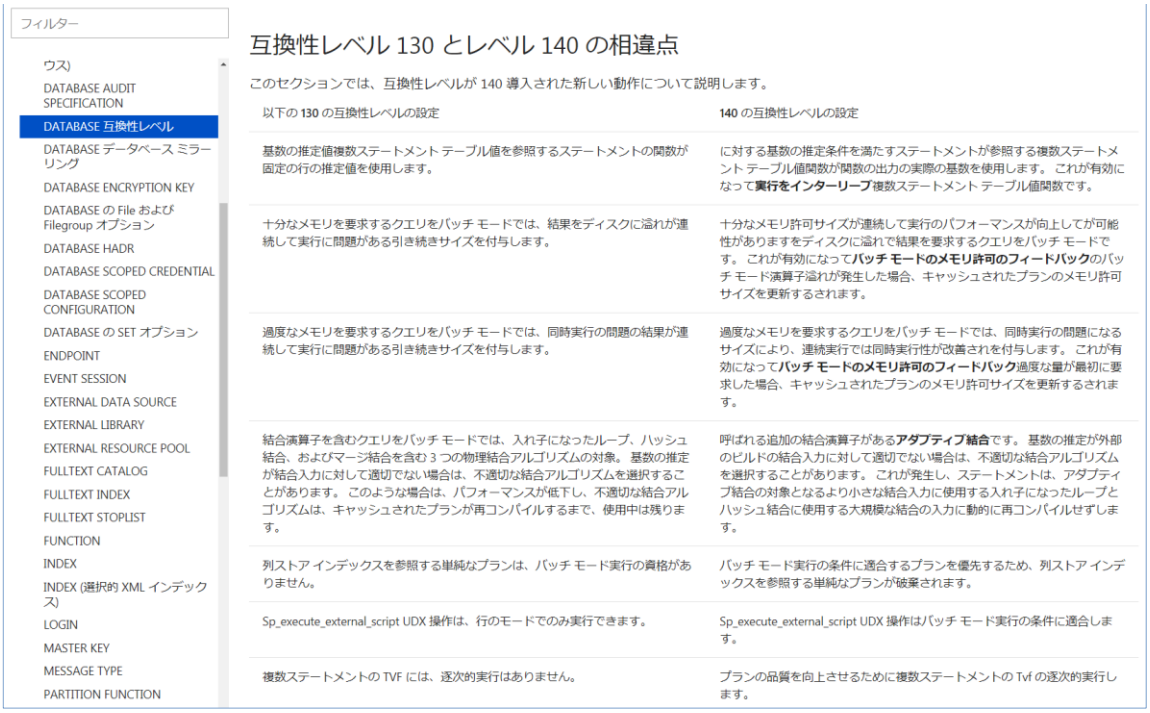
前述の **Adaptive Query Processing**（バッチモードでの適応型 JOIN や Memory Grant Feedback）を利用するには、データベースの互換性レベルが 140 である必要があります。SQL Server 2016 や 2014 など、古いバージョンの SQL Server で取得したバックアップを SQL Server 2017 上にリストアしたり、古いバージョンのデータベース ファイル（.mdf／.ldf）を SQL Server 2017 上にアタッチした場合には、そのバージョンの互換性レベルが保たれるので、140 レベルに上げても問題ないか、検証してみることをお勧めします。

検証にあたっては、クエリ ストア機能を利用することもお勧めです。クエリ ストアを有効化した

後に、互換性レベルを変更する前後（元々の互換性レベルと 140 に変更した後）で、同じクエリを実行して、それをクエリ ストアに記録させます。これによって、異なる実行プランで実行されたかどうかをチェックすることができ、どのぐらいの性能だったのか（性能が向上したのか、あるいは性能が低下したのか etc）についても簡単にチェックすることができます。

互換性レベルの詳細については、オンライン ブックの以下のトピックがお勧めです。

ALTER DATABASE (Transact-SQL の互換性レベル)
https://docs.microsoft.com/ja-jp/sql/t-sql/statements/alter-database-transact-sql-compatibility-level



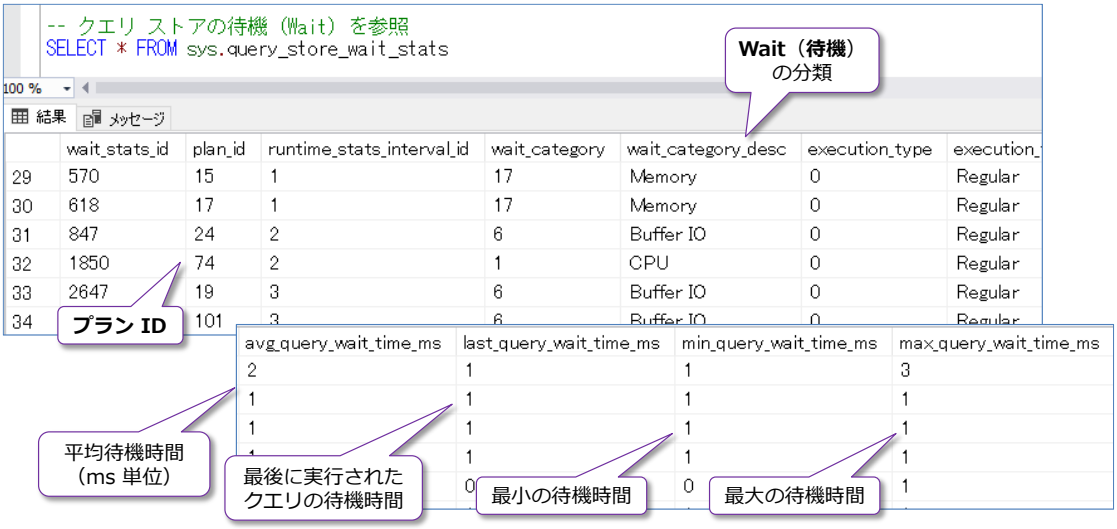
4.5 クエリ ストア機能の強化（クエリの Wait 情報の記録）

SQL Server 2017 では、クエリ ストアで、**クエリの待機**（**Wait**）に関する情報を記録できるようになりました。待機（Wait）は、クエリを実行するときに発生した内部的なリソースに関する待ち（待機）のことで、CPU 待ちや、ディスクへの書き込み待ち、ロック待ち、ラッチ待ち、並列クエリの場合の集約待ちなど、さまざまな種類の待機があります。

したがって、**待機**（Wait）を利用すれば、どのリソースが原因でクエリの実行時間がかかっているのかを特定することができるので、遅くなったクエリや、高負荷になっているクエリの原因（ボトルネック）を調べるのに役立ちます。クエリ ストアには、過去に実行したクエリの実行履歴が格納されているので、後から過去に振り返って、待機を調査できるので大変便利です。

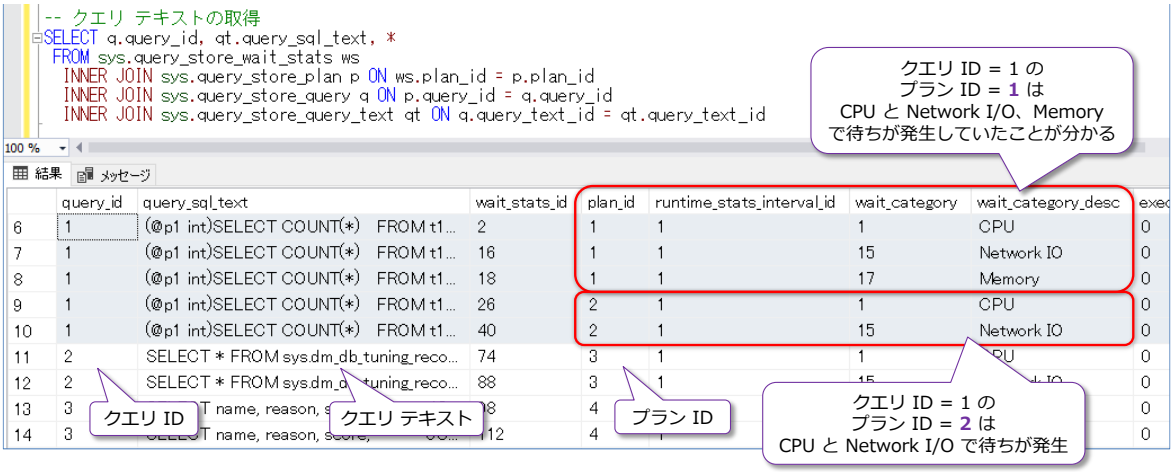
クエリ ストアに格納された待機（Wait）は、sys.**query_store_wait_stats** 動的管理ビューを利用して参照することができます。

```
-- クエリ ストアの待機（Wait）を参照
SELECT * FROM sys.query_store_wait_stats
```



クエリ ストアのクエリ情報（**プラン ID** や**クエリ ID**、**クエリ テキスト**など）は「**query_store_plan**」や「**query_store_query**」、「**query_store_query_text**」動的管理ビューを利用して参照できるので、次のように JOIN することで、クエリ テキストを取得することができます。

```
-- クエリ テキストの取得
SELECT q.query_id, qt.query_sql_text, *
 FROM sys.query_store_wait_stats ws
  INNER JOIN sys.query_store_plan p
    ON ws.plan_id = p.plan_id
  INNER JOIN sys.query_store_query q
    ON p.query_id = q.query_id
  INNER JOIN sys.query_store_query_text qt
    ON q.query_text_id = qt.query_text_id
```

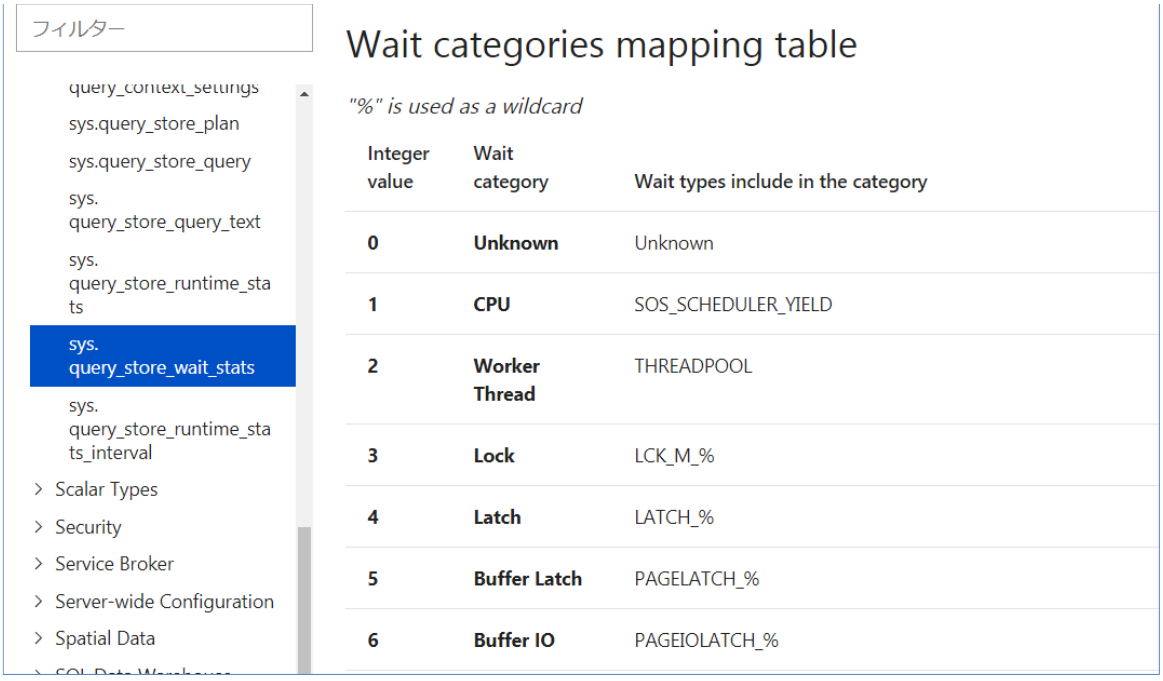


このように、SQL Server 2017 からは、クエリ ストアに待機（Wait）情報が格納されるようになったので、クエリのボトルネックを調べる場合に大変便利になりました。

クエリ ストアの動的管理ビューに関しては、オンライン ブックの以下のトピックが参考になります。

クエリのストアを使用した、パフォーマンスの監視
https://docs.microsoft.com/ja-jp/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store

sys.query_store_wait_stats (Transact-SQL)
https://docs.microsoft.com/ja-jp/sql/relational-databases/system-catalog-views/sys-query-store-wait-stats-transact-sql

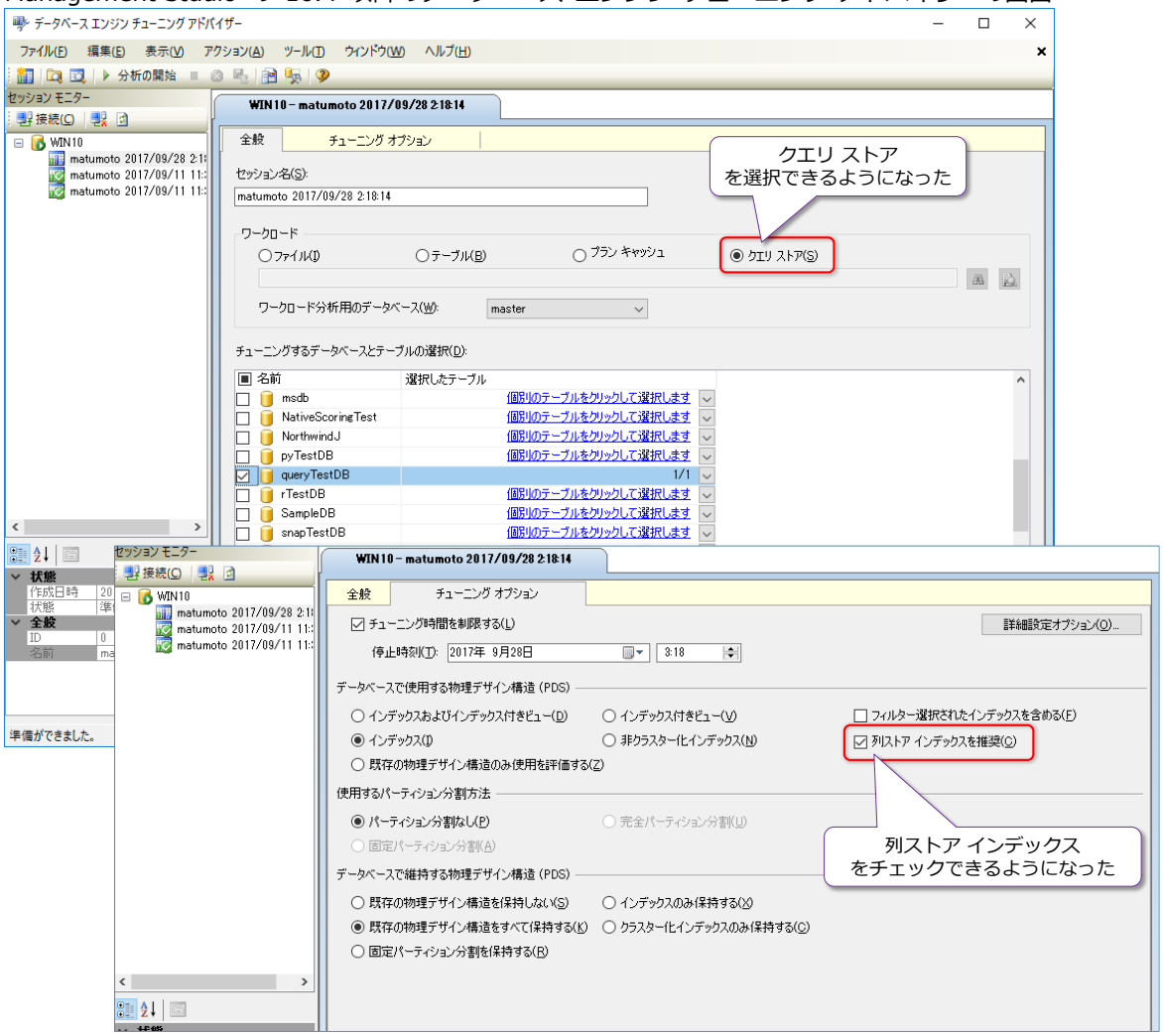


4.6 データベース エンジン チューニング アドバイザーの強化

データベース エンジン チューニング アドバイザー（**DTA**）は、データベースに対するクエリを**自動分析**して、作成した方が良いインデックスや、削除した方が良いインデックス、お勧めのデータパーティション構造などを提案してくれる機能です。

DTA は、Management Studio の 16.4 以降で強化されていて、**クエリ ストア**をデータ ソース（チューニング対象）にすることができたり、**列ストア インデックス**を採用すべきかどうかもチェックしてくれるようになりました。

Management Studio の 16.4 以降のデータベース エンジン チューニング アドバイザーの画面



Note： データベース エンジン チューニング アドバイザーの歴史は古い

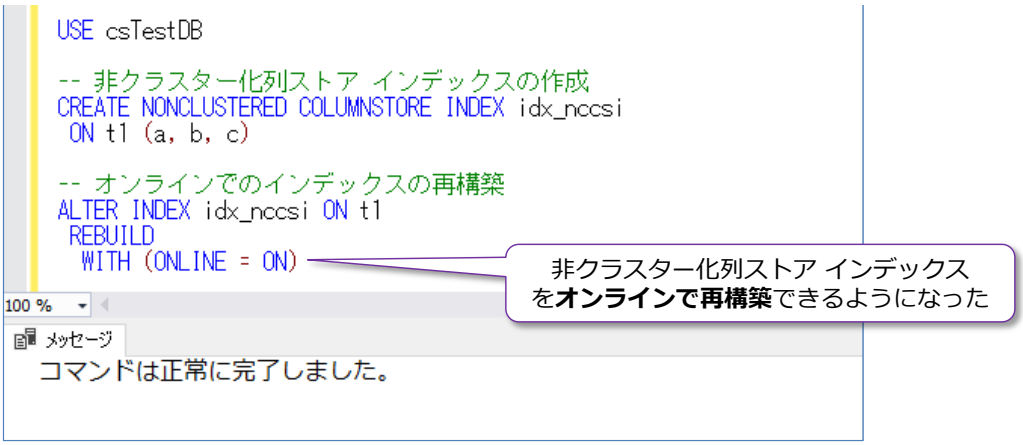
DTA ツールの歴史は非常に古く、SQL Server 7.0 のとき（18 年前！）から提供されています。SQL Server 7.0 のときはインデックス チューニング ウィザードと呼ばれていて、インデックスに関する提案のみが可能でした。このツールは、その後 DTA と呼ばれるようになって、インデックス付きビューや、データパーティション構造へのお勧めに対応したり、データソースとしてプラン キャッシュを利用できりょうになったりと、バージョンが上がるたびに進化しています。

4.7 列ストア インデックスの強化

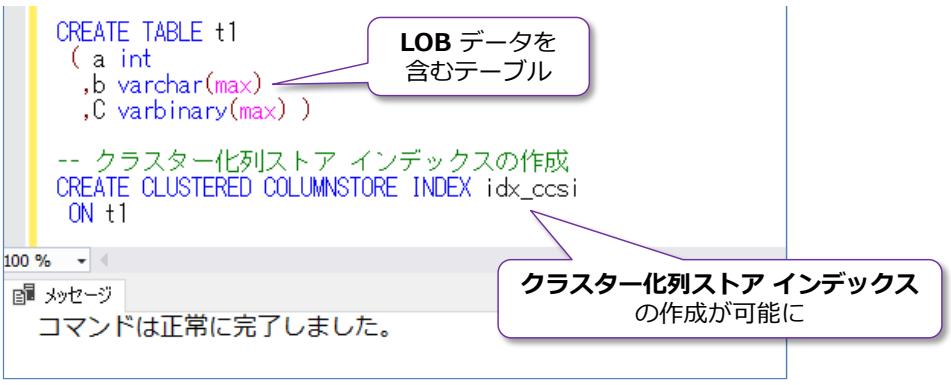
SQL Server 2017 では、列ストア インデックス（カラムストア インデックス）も強化されています。強化ポイントは、次のとおりです。

- バッチ モードにおける**性能向上**（前掲の Adaptive Query Processing）
- 非クラスター化列ストア インデックスでの "**オンライン**" 再構築のサポート
- クラスター化列ストア インデックスで **LOB** データのサポート
  (varchar(max) や nvarchar(max)、varbinary(max) 列のサポート)

列ストア インデックスは、性能を維持するために、定期的なインデックスの再構築が欠かせませんが、SQL Server 2016 までは "**オフライン**" での再構築しかできませんでした。SQL Server 2017 からは、非クラスター化列ストア インデックス（Non Clustered Columnstore Index）で、オンライン再構築がサポートされるようになりました。



また、SQL Server 2017 では、クラスター化列ストア インデックスで **LOB**（ラージ オブジェクト）データがサポートされるようになったので、varchar(max) や nvarchar(max)、varbinary(max) 列があったとしても、クラスター化列ストア インデックスを作成できるようになりました。



4.8 インメモリ OLTP 機能の強化

SQL Server 2017 では、インメモリ OLTP 機能も大きく強化されています。その主なものは、次のとおりです。

- メモリ最適化テーブルでの 8 個のインデックス上限の廃止
- メモリ最適化テーブルでの**計算列**のサポート（計算列に対するインデックスの作成も可能）
- **sp_rename** のサポート（メモリ最適化テーブルやネイティブ コンパイル ストアド プロシージャに対して、名称変更が可能に）
- ネイティブ コンパイル モジュールで以下のサポートを追加
  CASE 式、**JSON** 関数、**CROSS APPLY** 演算子、**TOP (N) WITH TIES** のサポート
- メモリ最適化テーブルにおけるトランザクション ログの Redo 処理が**並列対応**（これによって、復旧時間が短縮するので、AlwaysOn 可用性グループ構成でのスループットの向上に繋がる）
- メモリ最適化テーブルでの **bw-tree** 非クラスター化インデックスのビルド性能の向上
- **sp_spaceused** のサポート
- メモリ最適化ファイル グループを Azure Storage 上に保存可能に

インメモリ OLTP は、完全にインメモリで動作させることで、**OLTP の性能**を向上させることができる機能として SQL Server 2014 から提供されましたが、SQL Server 2016 では OLTP だけでなく、**分析ワークロード**にも対応できるようにクラスター化列ストア インデックスをサポートしました（Operational Analytics の実現）。また、SQL Server 2016 では、各種の制限の緩和（ALTER のサポートや、インデックスでの BIN 以外の照合順序のサポートなど）によって、非常に使いやすくなりました（既存のディスク ベースのテーブルを、インメモリ OLTP に移行しやすくなりました）。

そして、SQL Server 2017 からは、さらに制限が緩和されて、インメモリ OLTP におけるテーブルである「**メモリ最適化テーブル**」に作成できるインデックスの上限が 8 個だったものを撤廃、メモリ最適化テーブルに対して計算列を作成することもできるようになりました。

また、**sp_rename** で名称変更もできるようになったり、ネイティブ コンパイル モジュールで **CASE** 式サポートされたり、**JSON** 対応、**CROSS APPLY** へのサポートも追加されました。

```
-- ネイティブ コンパイル SP での CASE 式のサポート
CREATE PROCEDURE native_sp1
 WITH SCHEMABINDING, NATIVE_COMPILATION
AS BEGIN
    ATOMIC WITH
     (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'Japanese')
    SELECT
         CASE
            WHEN col1 = 1 THEN 'a'
            WHEN col1 = 2 THEN 'b'
            ELSE 'c' END AS col1
     FROM dbo.t1
   END
go
```

ネイティブ コンパイル ストアド プロシージャ で **CASE** 式をサポート

```
100 %
メッセージ
コマンドは正常に完了しました。
```

```
-- メモリ最適化テーブルに JSON データを格納
CREATE TABLE jsonInMem
(  colA int PRIMARY KEY NONCLUSTERED
  ,colB nvarchar(1000)
        CONSTRAINT chkJSON1 CHECK ( ISJSON(colB) > 0 )
  ,colC AS CAST(JSON_VALUE(colB, '$.key2') AS int) PERSISTED  )
WITH ( MEMORY_OPTIMIZED = ON )

-- JSON データの INSERT
INSERT INTO jsonInMem VALUES(1, N'{"key1": "test1", "key2": 999}')
INSERT INTO jsonInMem VALUES(2, N'{"key1": "test2", "key2": 777}')

-- ネイティブ コンパイル SP で JSON 関数を利用
CREATE PROCEDURE native_sp2 (@vals nvarchar(100))
WITH SCHEMABINDING, NATIVE_COMPILATION
AS BEGIN
    ATOMIC WITH
     (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'Japanese')

    SELECT colA, colB, colC, JSON_VALUE(colB,'$.key1') AS key1
    FROM dbo.jsonInMem
        INNER JOIN OPENJSON(@vals)
            ON colA = value
END
```

メモリ最適化テーブル での CHECK 制約に **ISJSON** を利用可能に

計算列で **JSON** 関数を利用可能に

ネイティブ コンパイル ストアド プロシージャ で **JSON** 関数をサポート

```
100 %
メッセージ
コマンドは正常に完了しました。
```

このように インメモリ OLTP は、SQL Server 2017 でさらに使いやすくなって、ディスク ベースのテーブルと同じように使えるようになってきているので、既存のテーブルからの移行が非常にしやすくなりました。

4.9 再開可能なオンライン インデックス再構築

SQL Server 2017 では、インデックスの**オンライン再構築**の際に、途中で停止をしたり、一時停止をして、操作を停止時から再開するといったことができるようになりました。また、途中でログの切り捨てもできるので、ディスク容量が足りなくなった場合などでも、再構築を継続できるようになりました。

これを行うには、**ALTER INDEX** ステートメントでのインデックスの再構築時に、次のように **WITH** オプションで **RESUMABLE=ON** を指定します。

```
-- 再開可能にするには RESUMABLE=ON を指定する
ALTER INDEX インデックス名 ON テーブル名
 REBUILD
  WITH ( ONLINE = ON, RESUMABLE = ON )
```

このように開始したインデックスの再構築は、次のように再構築を一時中断したり、進行状況（何%ぐらい再構築が完了したのか）などを確認したりできます。

```
-- 再構築を一時中断 (PAUSE)
ALTER INDEX インデックス名 ON テーブル名
 PAUSE

-- 再構築の進行状況を確認 (index_resumable_operations ビューを利用)
SELECT percent_complete,page_count,
        OBJECT_SCHEMA_NAME(object_id) AS schema_name,
        OBJECT_NAME(object_id) AS object_name, name
FROM sys.index_resumable_operations

-- 再構築を再開 (RESUME)
ALTER INDEX インデックス名 ON テーブル名
 RESUME
```

Let's Try

それでは、これを試してみましょう。

1. まずは、インデックスの再構築を試すためのデータベースを作成します。クエリ エディターで、次のように **CREATE DATABASE** ステートメントを実行して、「**idxTestDB**」という名前のデータベースを作成します。

```
CREATE DATABASE idxTestDB
```

2. 次に、「**t1**」という名前のテーブルを作成します。

```
USE idxTestDB

```

```
CREATE TABLE t1
 ( a int IDENTITY(1, 1)
                CONSTRAINT pk_t1 PRIMARY KEY
  ,b int
  ,c char(200) DEFAULT 'dummy1'
  ,d char(200) DEFAULT 'dummy2' )
```

a 列を **PRIMARY KEY** にして、既定の**クラスター化インデックス**を作成するようにしています（インデックス名は、制約の名前と同様、**pk_t1** になります）。

3. 次に、データを **100 万件** INSERT します。

```
-- データを 100万件追加
SET NOCOUNT ON
BEGIN TRAN
    DECLARE @i int = 1
    WHILE @i <= 1000000
      BEGIN
        INSERT INTO t1(b) VALUES(100)
        SET @i += 1
      END
COMMIT
SET NOCOUNT OFF
```

4. データの INSERT が完了したら、まずは**オフライン**でのインデックスの再構築を実行してみます。

```
-- インデックスの再構築（オフライン）
ALTER INDEX pk_t1 ON t1
 REBUILD
```

このように、**REBUILD** のみを指定したインデックスの再構築は、"**オフライン**" での再構築と呼ばれて、再構築処理中は、このインデックスに対するアクセスがブロックされます。

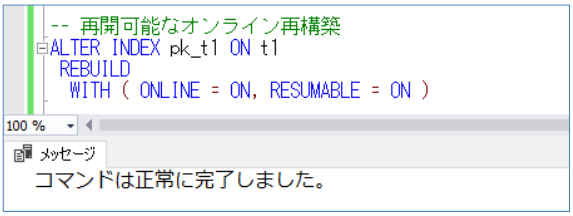
5. 次に、**オンライン**でのインデックスの再構築を実行してみます。

```
-- オンラインのインデックスの再構築
ALTER INDEX pk_t1 ON t1
 REBUILD
  WITH ( ONLINE = ON )
```

オンラインでのインデックス再構築の場合は、**WITH** オプションで「**ONLINE=ON**」を指定します。このように、オンラインでインデックスを再構築した場合は、再構築中でも該当インデックスへのアクセスが可能になります。

6. 続いて、SQL Server 2017 からの新機能である "**再開可能な**" オンラインでのインデックスの再構築を実行してみましょう。

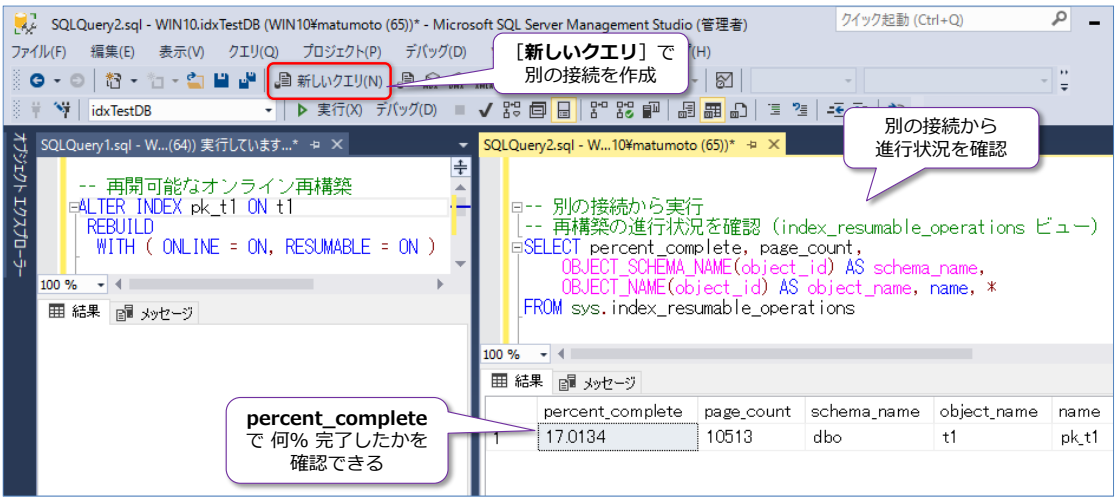
```
-- 再開可能なオンライン再構築
ALTER INDEX pk_t1 ON t1
 REBUILD
  WITH ( ONLINE = ON, RESUMABLE = ON )
```



このように **RESUMABLE=ON** を追加することで、再開可能なオンラインでのインデックスの再構築が行えます。

7. 再構築が完了したら、次に、同じステートメントを実行して、もう一度再開可能なオンラインでのインデックスの再構築を実行し、その実行中に、今度は別の接続から **index_resumable_operations** 動的管理ビューを参照して、再構築の進行状況を確認してみましょう。別の接続は、ツールバーの［**新しいクエリ**］をクリックして作成します。

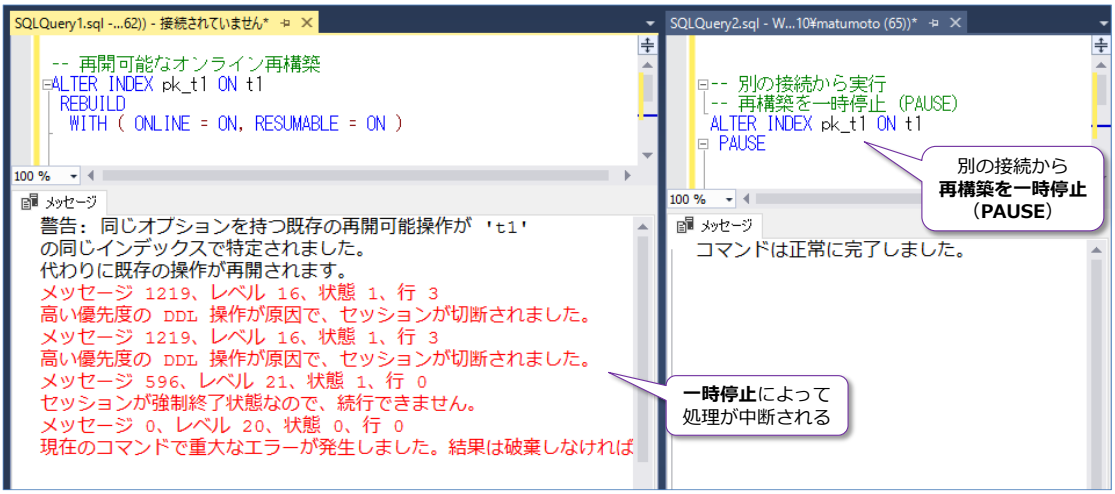
```
-- 別の接続から再構築の進行状況を確認
SELECT percent_complete, page_count,
        OBJECT_SCHEMA_NAME(object_id) AS schema_name,
        OBJECT_NAME(object_id) AS object_name, name, *
FROM sys.index_resumable_operations
```



このように、**RESUMABLE=ON** を指定した再開可能なオンライン インデックス再構築は、進行状況を確認することができます。

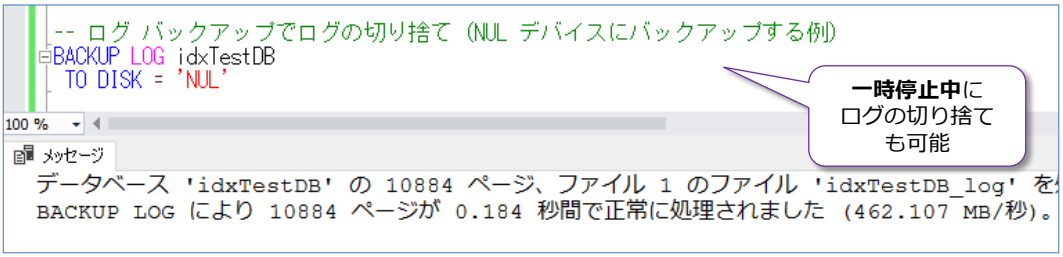
8. 次に、もう一度、再開可能なオンライン インデックス再構築を実行して、その実行中に別の接続から、**再構築を一時停止**（**PAUSE**）してみます。

```
--別の接続から再構築を一時停止（PAUSE）
ALTER INDEX pk_t1 ON t1
 PAUSE
```



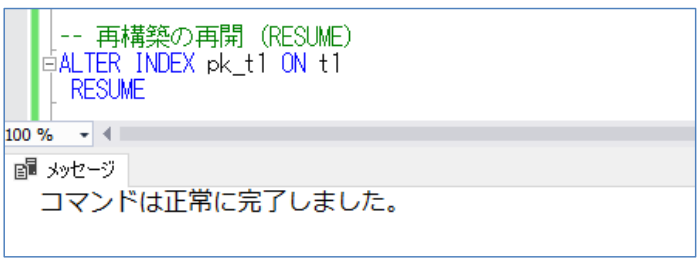
9. 再構築を一時中断した状態では、前述の **index_resumable_operations** 動的管理ビューで進行状況（何% 完了したのか）を確認したり、ログのバックアップを実行する（ログの切り捨てによってログ領域の解放を行う）といったことができます。

```
-- ログ バックアップでログの切り捨て (NUL デバイスにバックアップする例)
BACKUP LOG idxTestDB
 TO DISK = 'NUL'
```



10. 一時停止したインデックスの再構築を再開するには、次のように **RESUME** を実行します。

```
-- 再構築の再開 (RESUME)
ALTER INDEX pk_t1 ON t1
 RESUME
```



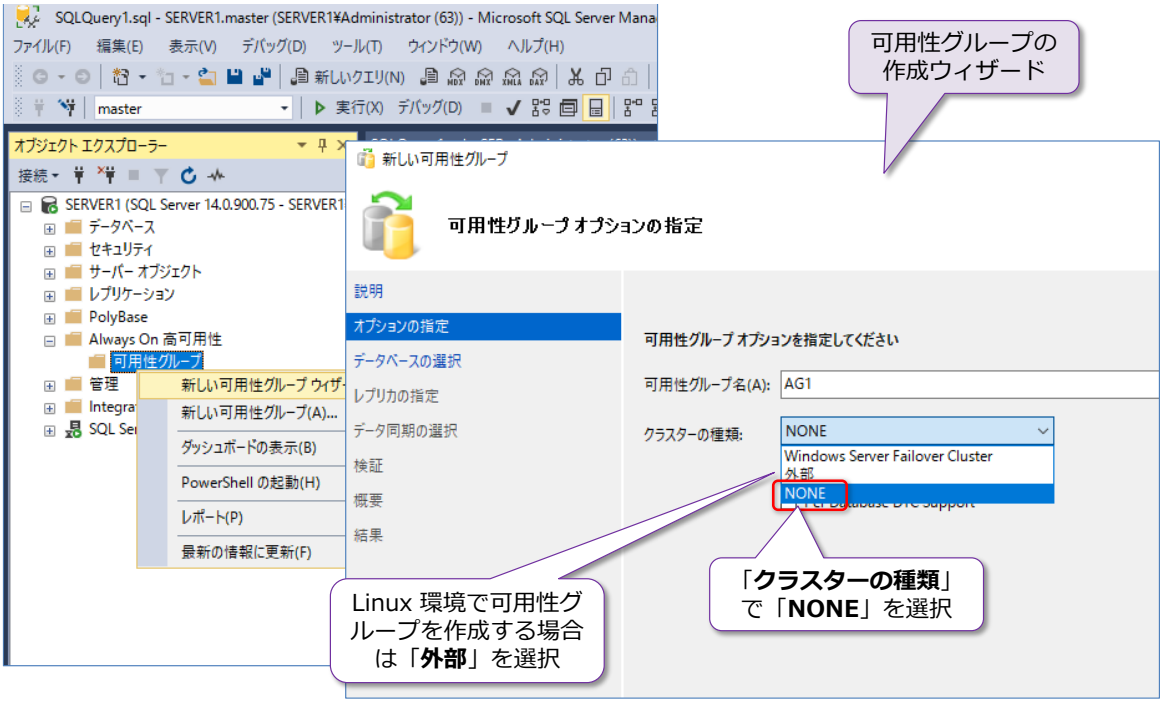
このように、SQL Server 2017 からは、オンラインでのインデックス再構築の際に、一時停止ができるようになったので、ディスク容量が足りなくなった場合などに対応できるようになりました。

4.10 AlwaysOn 可用性グループの強化

SQL Server 2017 では、**AlwaysOn 可用性グループ**（Availability Group）も強化されています。その主なものは,次のとおりです。

- DTC（分散トランザクション）のサポート
- Linux 環境での可用性グループのサポート
  （Windows と Linux をまたがった可用性グループも構成可能）
- クラスター レスの可用性グループのサポート
  （分析／レポーティング用途などの読み取り専用ワークロードをスケール アウト）

クラスター レス（クラスター不要）の可用性グループを構成すれば、分析（Analytics）用途や、レポーティングのための読み取り専用のワークロードをスケール アウトするために利用することができます。可用性グループを作成するときに、次のようにクラスターの種類（**CLUSTER_TYPE**）で「**NONE**」を選択します。



STEP 5. その他の新機能

この STEP では、「**Transact-SQL の強化**」や「**セットアップ時の変更点**」、「**スマート バックアップ**」、「**新しい DMV**」、「**テンポラル テーブルの強化**」、「**BI 機能の強化**」など、SQL Server 2017 で提供されたその他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。

- ✓ Transact-SQL の強化
- ✓ セットアップ時の変更点（tempdb ファイルの設定）
- ✓ スマート バックアップ
- ✓ 新しい DMV（動的管理ビュー）
- ✓ テンポラル テーブルの強化
- ✓ SQL CLR のセキュリティ強化
- ✓ BI 機能の強化

5.1 Transact-SQL の強化

SQL Server 2017 では、Transact-SQL ステートメントも強化されています。その主なものは,次のとおりです。

- 新しい関数の追加
  TRIM、**CONCAT_WS**、**TRANSLATE**、**STRING_AGG（WITHIN GROUP）**
- **SELECT INTO** での**ファイル グループの指定**
- 一括操作での **FORMAT = CSV** による CSV ファイルの取り込みのサポート
  （BULK INSERT や、OPENROWSET(BULK...) で利用可能）
- **IDENTITY_CACHE** オプションのサポート（IDENTITY 値のキャッシュをオフにすることがデータベース単位で可能に）
- 日本語向けの**新しい照合順序**の追加
  Japanese_Bushu_Kakusu_140 と Japanese_XJIS_140 照合順序の追加、
  それぞれ _VSS（Variation-selector-sensitive）に対応

新しい関数

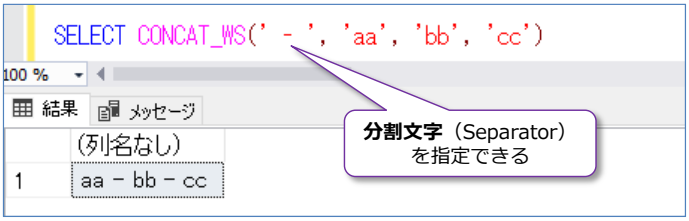
SQL Server 2017 では、Transact-SQL の新しい関数として、**TRIM** や **CONCAT_WS**、**TRANSLATE**、**STRING_AGG**（WITHIN GROUP）などがサポートされました。

TRIM は、今までなかったのが不思議なぐらいの関数ですが、左右の余分なスペース（空白）を除去できるものです。

```
-- 従来の書き方は RTIM と LTRIM を入れ子で利用
SELECT RTRIM(LTRIM('     test    '))

-- SQL Server 2017 の新機能「TRIM 関数」を利用して左右のスペースを除去
SELECT TRIM('     test    ')
```

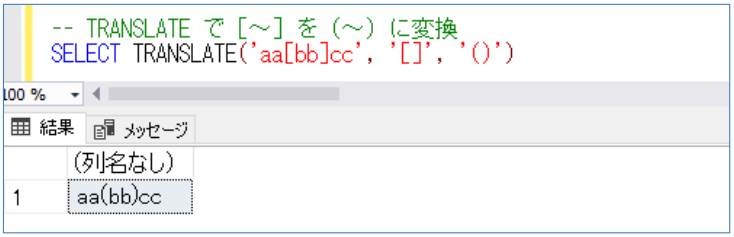
CONCAT_WS は、文字列を連結する **CONCAT** の **WS**（With Separator）版で、文字列を連結するときに、任意の Separator（分割文字）を追加できるものです。

```
-- CONCAT_WS の利用例。第２引数以降の文字列を、第１引数で指定した文字で連結
SELECT CONCAT_WS(' - ', 'aa', 'bb', 'cc')
```



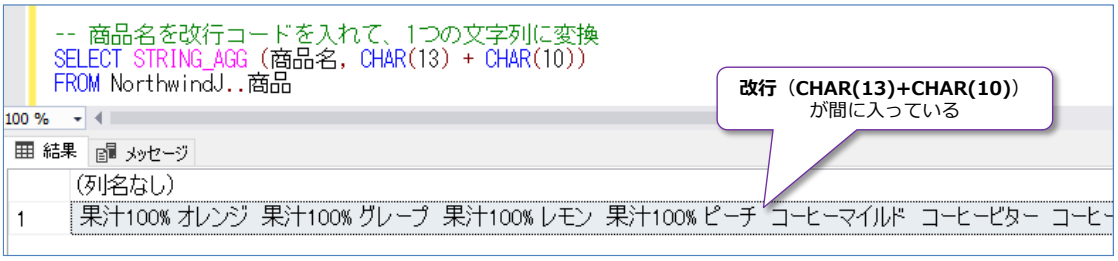
TRANSLATE は、特定の文字パターンを、別の文字パターンに簡単に置き換えられる、従来の REPLACE 関数の高機能版のようなものです。例えば、

```
-- TRANSLATE で [～] を (～) に変換
SELECT TRANSLATE('aa[bb]cc', '[]', '()')

-- 従来の REPLACE の場合
SELECT REPLACE( REPLACE('aa[bb]cc', '[', '('), ']', ')')
```



STRING_AGG は、文字列の結果を 1 つの文字列としてまとめ上げることができる関数です。

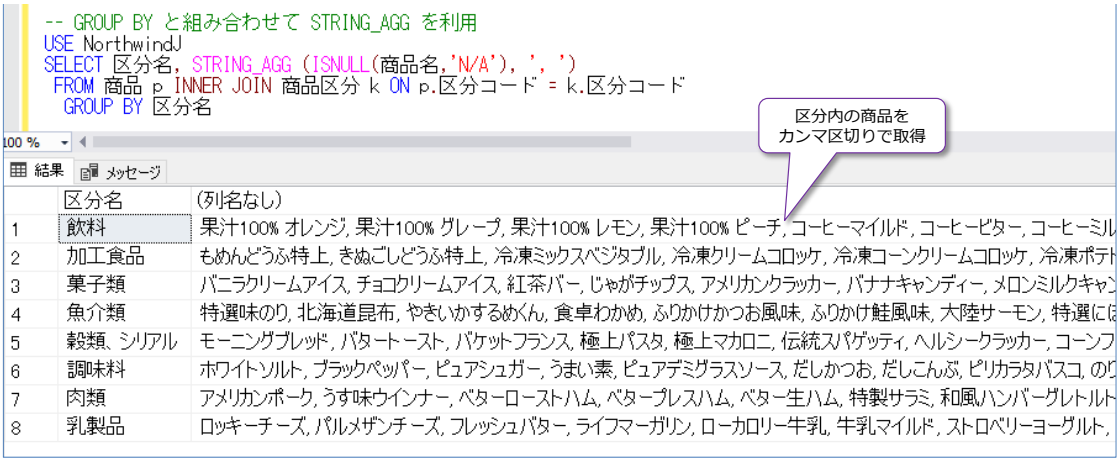
```
-- 商品名を改行コードを入れて、1つの文字列に変換
SELECT STRING_AGG (商品名, CHAR(13) + CHAR(10))
FROM NorthwindJ..商品
```



なお、上の例では **NorthwindJ** データベースの**商品**テーブルを利用していますが、このデータベースの作成方法については、巻末の付録に記載しています。

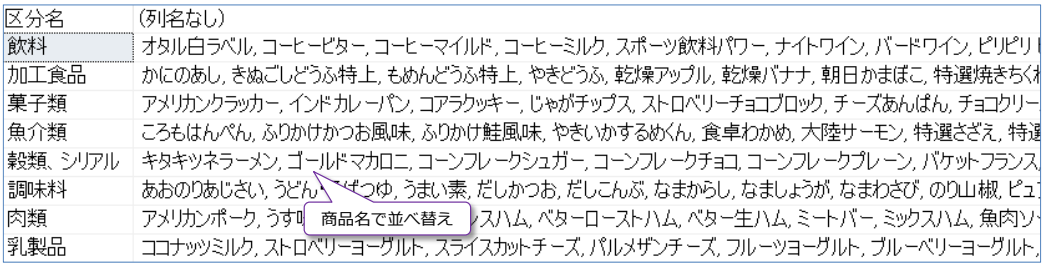
STRING_AGG は、GROUP BY 演算とともに利用すると、集計データに対して、説明的な要素を付加できるので、大変便利です。例えば、次のように商品区分（飲料や加工食品など）で GROUP BY（グループ化）をして、商品名を **STRING_AGG** で連結（カンマ区切りを指定）すれば、区分内の商品一覧を 1 行で表現することができます。

```
-- GROUP BY と組み合わせて STRING_AGG を利用
USE NorthwindJ
SELECT 区分名, STRING_AGG (ISNULL(商品名,'N/A'), ', ')
 FROM 商品 p INNER JOIN 商品区分 k
   ON p.区分コード = k.区分コード
  GROUP BY 区分名
```



また、**STRING_AGG** 関数では、**WITHIN GROUP (ORDER BY ～)** を利用することで、文字列を集約するときに並べ替えを指定することもできます。

```
-- WITHIN GROUP で ORDER BY を付けて並べ替えが可能。商品名で並べ替え
USE NorthwindJ
SELECT 区分名, STRING_AGG (ISNULL(商品名,'N/A'), ', ')
                WITHIN GROUP (ORDER BY 商品名 ASC)
FROM 商品 p INNER JOIN 商品区分 k ON p.区分コード = k.区分コード
GROUP BY 区分名
```



SELECT INTO でのファイル グループの指定

SQL Server 2017 では、**SELECT INTO** で新しいテーブルを作成するときに、**ファイル グループ**を指定できるようになりました。DWH（データ ウェアハウス）環境では、SELECT INTO を多用しますが、これまでのバージョンでは、既定のファイル グループにデータが格納されてしまっていたので、別のファイル グループに格納したい場合には、「**ALTER DATABASE .. MODIFY FILEGROUP .. DEFAULT**」ステートメントを利用して、SELECT INTO を実行する前後で、既定のファイル グループを変更しなければなりませんでした。

しかし、SQL Server 2017 からは、次のように SELECT INTO で **ON** 句が追加されて、ファイル グループを指定できるようになりました。

```
-- 新しいテーブルを ON 句で指定したファイル グループに格納
SELECT * INTO 新テーブル名 ON ファイル グループ名
 FROM 元テーブル名
```

一括操作での FORMAT = 'CSV'

SQL Server 2017 では、BULK INSERT や OPENROWSET(BULK...) で「**FORMAT = 'CSV'**」がサポートされるようになったので、CSV ファイルを簡単に取り込めるようになりました。

```
-- 従来の書き方（CSV ファイルインポートは FIELDTERMINATOR を利用）
BULK INSERT temp1
 FROM 'C:¥temp¥prod.csv'
 WITH ( FIELDTERMINATOR = ',' )

-- SQL Server 2017 での CSV ファイルインポートは FORMAT = 'CSV' で OK
BULK INSERT temp1
 FROM 'C:¥temp¥prod.csv'
 WITH ( FORMAT = 'CSV' )
```

IDENTITY_CACHE オプションのサポート

IDENTITY 値は、既定ではキャッシュが有効になっているので、SQL Server が不意のシャットダウンなどをした場合には、SQL Server の起動後に、IDENTITY 値にギャップが生まれる場合があります。例えば、IDENITY 値が 101 まで割り振られているときに、キャッシュが 50 あったとして、不意のシャットダウンがあると、次に割り振られる値は 152 になって、102～151 の間の値が抜け番になってしまいます。

これを回避するには、これまでのバージョンでは、トレースフラグ「**272**」を利用する必要がありました。ただし、トレースフラグの場合は、サーバー全体で有効になるので、データベース単位で個別設定することはできませんでした。そこで、SQL Server 2017 からは、データベース単位でキャッシュをオフにできるオプションが追加されました。これは、次のように **ALTER DATABASE SCOPED CONFIGRATION** ステートメント（データベース スコープ構成を変更するためのステートメント）を利用して、「**IDENTITY_CACHE = OFF**」を設定します。

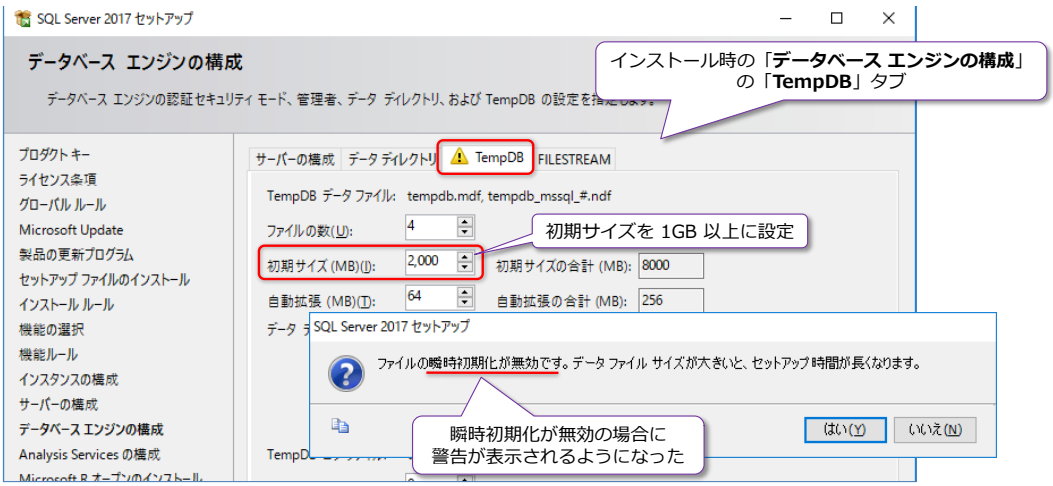
```
USE データベース名
go
ALTER DATABASE SCOPED CONFIGURATION
 SET IDENTITY_CACHE = OFF
```

5.2 セットアップ時の変更点（tempdb ファイルの設定）

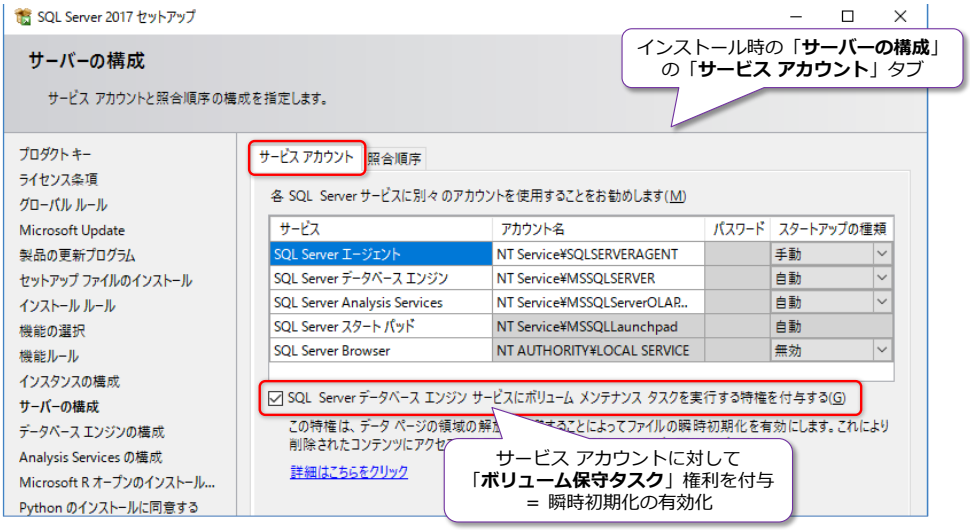
SQL Server 2017 では、インストール時の **tempdb** データベースのファイルに関する設定が、次のように変更されました。

- セットアップ時に、ファイルあたり **256 GB** (262,144 MB) までの初期ファイル サイズを指定
- セットアップ時に、ファイル サイズが **1 GB** より大きい値に設定されている場合、**瞬時初期化**が有効になっていないと、警告が表示される。トランザクション ログ ファイルの場合は、セットアップに時間がかかる主旨の警告が表示される

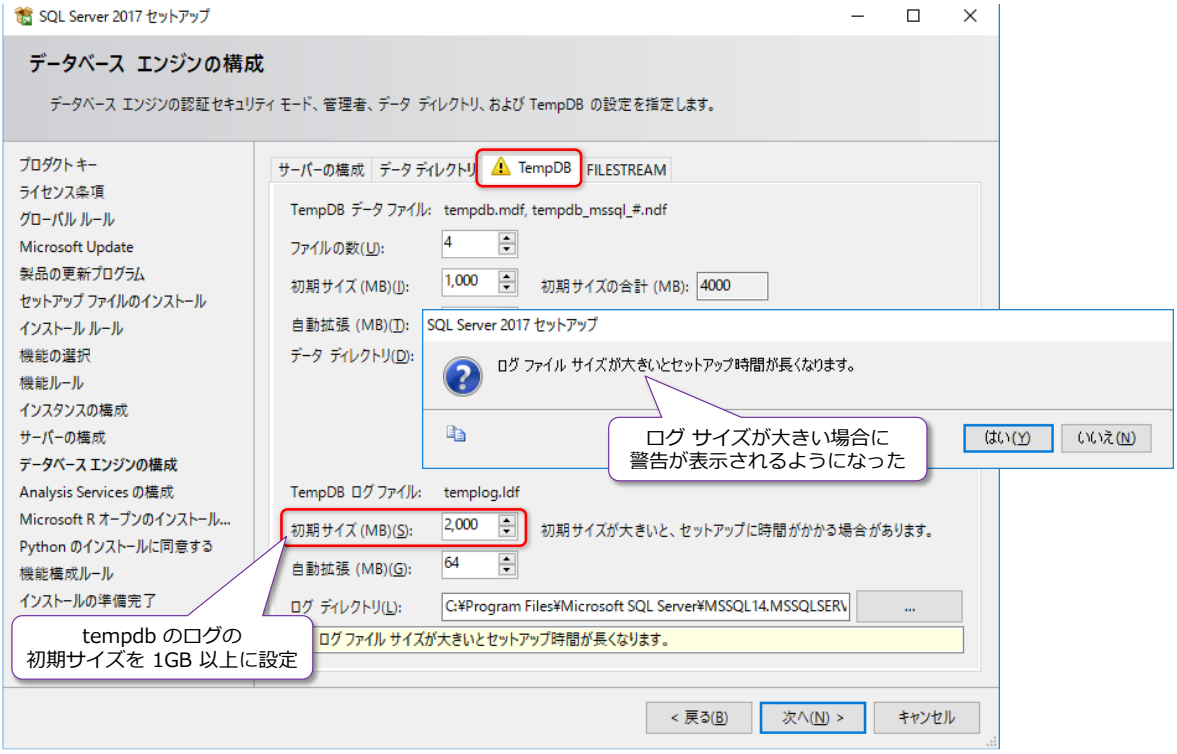
瞬時初期化が有効になっていない場合は、次のように警告が表示されます。



瞬時初期化（IFI： Instant File Initialization）は、**.mdf**（データベース）ファイルに対して利用できる大変便利な機能（ファイル サイズの拡張を瞬間的に完了できる機能）で、インストール時のサービス アカウントの設定ページで、次のように有効化することができます（サービス アカウントに対して、「**ボリュームの保守タスク**」ユーザー権利を付与することで、瞬時初期化を有効化できます）。



tempdb のトランザクション ログ ファイルを 1GB 以上に設定する場合は、次のようにセットアップに時間がかかる主旨の警告が表示されるようになりました。



5.3 スマート バックアップ

SQL Server 2017 では、バックアップ機能も強化されて、効率的にバックアップを取得（適切な頻度でログ バックアップを実行したり、差分バックアップと完全バックアップを適切に使い分けたり）するなど、スマートなバックアップを支援する機能が追加されています。

- 差分バックアップを効率良く実施するために、dm_db_file_space_usage ビューに **modified_extent_page_count** 列が追加（変更があったページ数を確認可能）
- ログ バックアップを効率良く実施するために、**dm_db_log_stats** ビューの提供（VLF: 仮想ログ ファイルの統計情報を確認可能）

差分バックアップのスマート バックアップ

差分バックアップは、データベースに対する更新が多い場合には、完全バックアップを実行するのと変わらなくなってしまう場合があります。データベース内のほとんどのページが更新されているのであれば、差分ではなくて、完全バックアップを実行しておいた方が、万が一のときの復旧時（リストア時）に、復旧時間（＝ダウンタイム）を小さくすることができます。

このような問題（差分と完全バックアップの適切な使い分け）を解決するために提供されたのが、sys.dm_db_file_space_usage 動的管理ビュー（DMV）の **modified_extent_page_count** 列です。この列には、データベースに対する変更のあったページ数が記録されているので、どのぐらいのデータ量が差分バックアップの対象になるのかを、バックアップを実行する前に確認することができます。

したがって、バックアップを実行する前に、**modified_extent_page_count** を参照して、**70～80%** を下回る場合には**差分バックアップ**を実行して、それ以上の場合には**完全バックアップ**を実行する、といった使い方ができるようになります。

トランザクション ログのスマート バックアップ

トランザクション ログ バックアップは、ログ バックアップを頻繁に取得しすぎると、バックアップ操作との競合や、内部的な **VLF**（仮想ログ ファイル）の断片化が発生したりして、性能への影響が出てしまいます。また、取得するログ バックアップ数が多い場合には、復旧時のダウンタイムにも影響します。これを回避するには、適切なタイミングでトランザクション ログのバックアップを取得するようにしますが、適切なタイミングでとは、ログに蓄積されたデータ量に応じて、バックアップを実行するという意味です。

従来は、トランザクション ログの VLF 情報は、**DBCC LOGINFO** コマンドで取得することができましたが、DBCC コマンドの結果は、再利用しづらく面倒でした。これを解決するために提供されたのが sys.**dm_db_log_stats** 動的管理ビュー（DMV）です。これを利用すれば、VLF の数やサイズを簡単に確認して、結果を再利用することができます（VLF の数に応じて、ログ バックアップを実施するのかどうかを判断するなどで利用できます）。

Let's Try

それでは、これを試してみましょう。ここでは、再開可能なインデックスの再構築のときに利用した「**idxTestDB**」データベースを利用して、差分バックアップのスマート バックアップを試してみます。

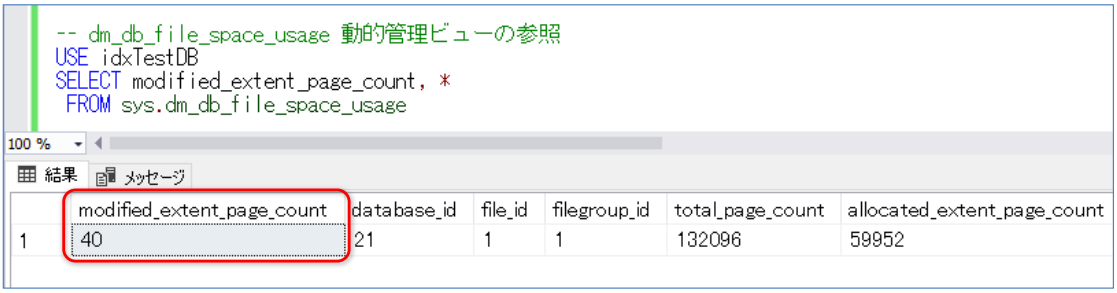
1. まずは、一度完全バックアップを実行します。

```
-- 完全バックアップを取得（NUL デバイスを利用）
BACKUP DATABASE idxTestDB
 TO DISK = 'NUL'
```

バックアップ先は NUL デバイスを指定して、実際のバックアップ ファイルは作成しない形で完全バックアップをシミュレートします。

2. 次に、**dm_db_file_space_usage** 動的管理ビューの **modified_extent_page_count** 列を参照します。

```
-- dm_db_file_space_usage 動的管理ビューの参照
USE idxTestDB
SELECT modified_extent_page_count, *
 FROM sys.dm_db_file_space_usage
```

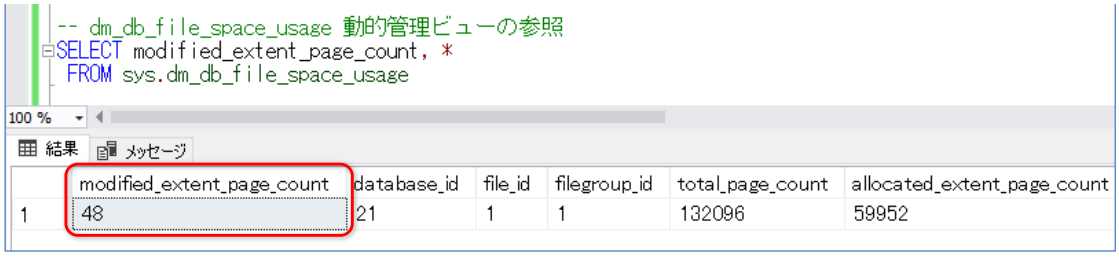


3. 次に、データを 1 件だけ UPDATE します。

```
-- 1件 UPDATE
UPDATE t1
 SET b = 111
  WHERE a = 1
```

4. UPDATE が完了したら、再度 **dm_db_file_space_usage** 動的管理ビューを参照します。

```
SELECT modified_extent_page_count, *
 FROM sys.dm_db_file_space_usage
```



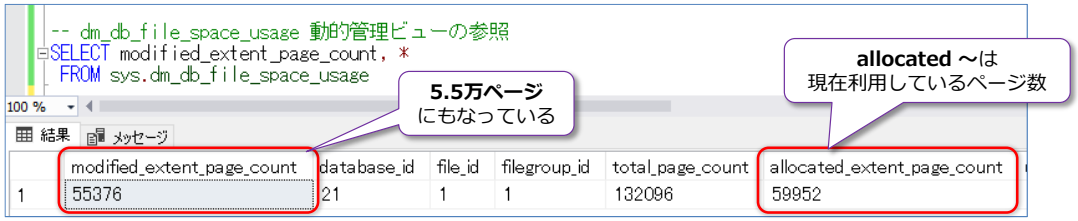
modified_extent_page_count 列の値が変化していることを確認できます（手順 2 では **40** でしたが、1 件の UPDATE 後は **48** に変わっています）。

5. 続いて、全データ（100 万件分）を UPDATE してみます（WHERE 句を付けずに UPDATE ステートメントを実行します）。

```
-- 100万件（全データ）の UPDATE
UPDATE t1
 SET b = 999
```

6. UPDATE が完了したら、再度 **dm_db_file_space_usage** 動的管理ビューを参照します。

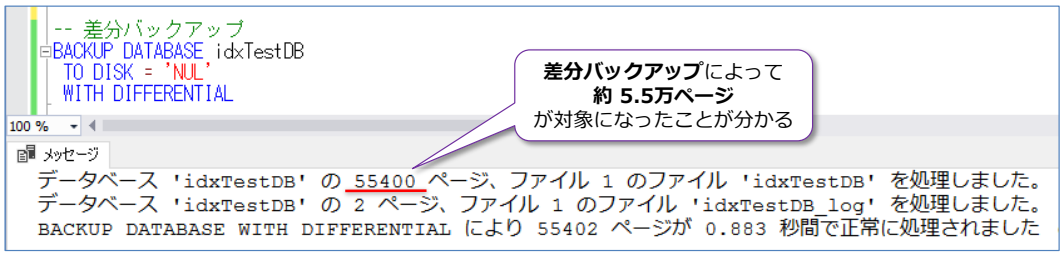
```
SELECT modified_extent_page_count, *
 FROM sys.dm_db_file_space_usage
```



今後は、**modified_extent_page_count** 列の値が **5.5 万**ページ以上になっていることを確認できます。現在利用中のページ数は、**allocated_extent_page_count** 列で確認できるので（**約 6 万**ページを使用中）、**90% 以上**のページに変更が加わっていることが分かります。

7. 次に、差分バックアップを実行します。

```
-- 差分バックアップを実行
BACKUP DATABASE idxTestDB
 TO DISK = 'NUL'
 WITH DIFFERENTIAL
```

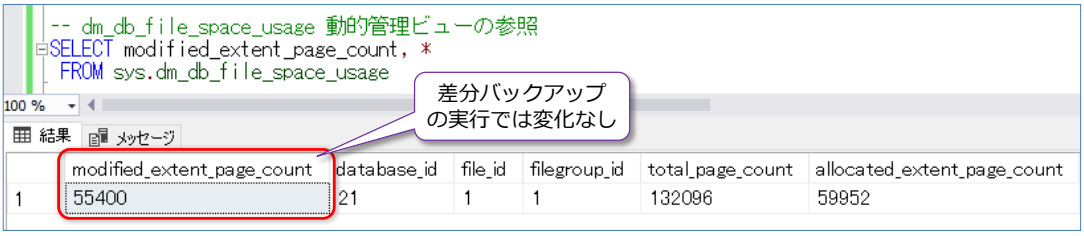


事前に **modified_extent_page_count** 列で確認した値とほぼ同じページ数が差分バック

アップの対象となっていたことを確認できます。

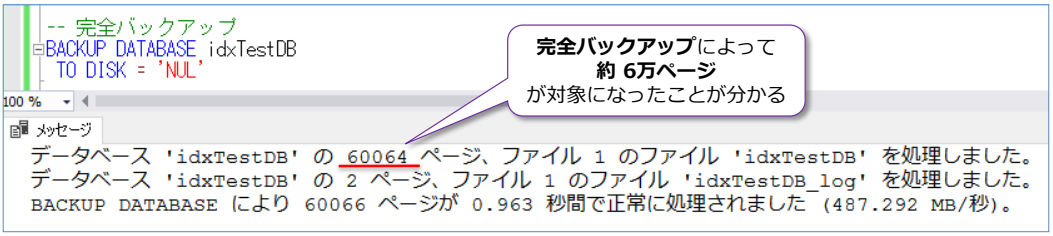
8. 次に、もう一度 **dm_db_file_space_usage** 動的管理ビューを参照します。

```
SELECT modified_extent_page_count, *
 FROM sys.dm_db_file_space_usage
```



9. 次に、完全バックアップを実行します。

```
-- 完全バックアップ
BACKUP DATABASE idxTestDB
 TO DISK = 'NUL'
```



完全バックアップによって、**約 6 万**ページが対象になったことを確認できます。これは、事前に **allocated_extent_page_count** 列で確認した値とほぼ同じページ数です。

10. 次に、もう一度 **dm_db_file_space_usage** 動的管理ビューを参照します。

```
SELECT modified_extent_page_count, *
 FROM sys.dm_db_file_space_usage
```

modified_extent_page_count 列の値がリセットされていることを確認できます。

完全バックアップは、差分バックアップの基準になり、差分バックアップには、完全バックアップ以降の差分情報のみ（**modified_extent_page_count** 列の値分のデータ）が含まれるというわけです。

しかし、**modified_extent_page_count** 列の値（差分ページ数）が、**allocated_exte**

nt_page_count 列の値（実際の利用ページ数）に近づいていった場合には、差分バックアップで取得するデータ量が非常に多くなるので、完全バックアップを実行するのと変わらなくなってしまいます。

したがって、差分ページ数が **70～80%** を下回る場合には**差分バックアップ**、それ以上の場合には**完全バックアップ**を実行するといった具合にバックアップを行うことで、スマートに差分バックアップを実行できるようになります。

11. 差分ページ数に応じて、バックアップの種類を変更するには、次のように Transact-SQL ステートメントを記述します。

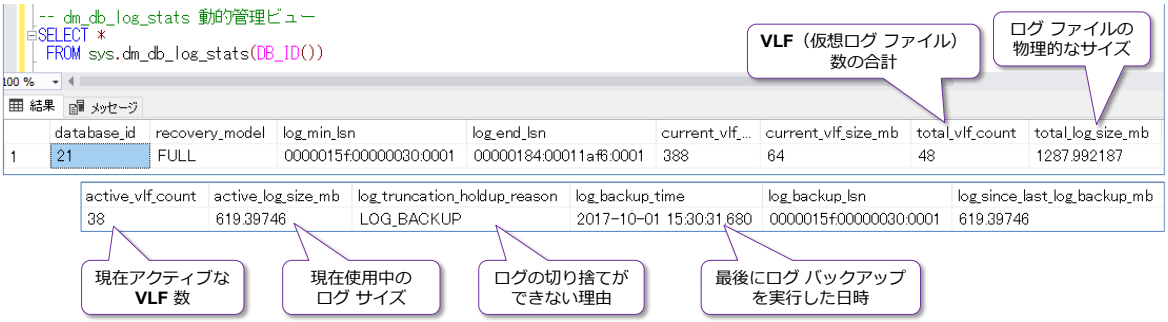
```
-- 差分ページ数の割合を取得
DECLARE @p float
SELECT @p = modified_extent_page_count * 1.0 / allocated_extent_page_count
 FROM sys.dm_db_file_space_usage

-- 割合が 80% 以上なら完全バックアップ、それ以下なら差分を実施
IF @p >= 0.8
  BEGIN
        -- 完全バックアップ
        BACKUP DATABASE idxTestDB
         TO DISK = 'NUL'
  END
ELSE
  BEGIN
        -- 差分バックアップ
        BACKUP DATABASE idxTestDB
         TO DISK = 'NUL'
          WITH DIFFERENTIAL
  END
```

modified_extent_page_count を、**allocated_extent_page_count** で除算して、その値が **0.8**（80%）以上かどうかで IF 分岐することで、バックアップの種類を変更することができます。

トランザクション ログのスマート バックアップ

トランザクション ログのバックアップをスマートに行うには、sys.**dm_db_log_stats** 動的管理ビューを利用すると便利です（SQL Server 2017 からの新機能）。

```
-- dm_db_log_stats 動的管理ビュー
SELECT *
 FROM sys.dm_db_log_stats(DB_ID())
```



トランザクション ログは、内部的には **VLF**（Virtual Log File：仮想ログ ファイル）という単位で分割されていますが、**dm_db_log_stats** 動的管理ビューを利用すれば、現在の VLF 数や、アクティブな VLF 数、ログ ファイルの使用中のサイズ、最後にログ バックアップを実行した日時などを簡単に確認することができます。また、**log_truncation_holdup_reason** 列には、ログが切り捨てできない理由も記録されているので、可用性グループでのログ転送が原因で切り捨てができない場合や、レプリケーションが原因で切り捨てができない、CHECKPOINT が原因で切り捨てができないといった、ログの切り捨てに関する理由を確認することもできます。

なお、以前のバージョンでは、トランザクション ログ／VLF に関する情報は、**dm_db_log_info** 動的管理ビューや **DBCC LOGINFO** コマンドで参照することができましたが、これらは VLF に関する明細データ（1 件 1 件の詳細データ）であったため、利用するのは面倒でした。しかし、SQL Server 2017 からは、これらの明細データを要約／統計（stats: statistics）した **dm_db_log_stats** 動的管理ビューが提供されたので、簡単に概要を確認できるようになりました。これを利用すれば、ログの使用量／アクティブな VLF 数が増えた場合にのみ、ログ バックアップを実行すれば良いようになるので、無駄にログ バックアップを取得することなく、スマートにログ バックアップを実行できるようになります。

スマート ログ バックアップについては、SQL Server Tiger Team の以下のブログ記事が参考になります。

Smart Transaction log backup, monitoring and diagnostics with SQL Server 2017
https://blogs.msdn.microsoft.com/sql_server_team/smart-transaction-log-backup-monitoring-and-diagnostics-with-sql-server-2017/

5.4 新しい DMV（動的管理ビュー）

SQL Server 2017 では、**DMV**（動的管理ビュー：Dynamic Management View）も強化されています。その主なものは,次のとおりです。

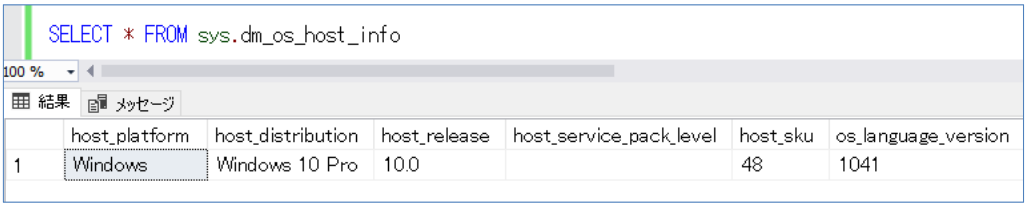
- sys.dm_db_file_space_usage に **modified_extent_page_count** 列の追加（前掲）
- sys.**dm_db_log_stats** の追加（前掲）
- sys.**dm_os_host_info** の追加
  Windows／Linux の両方について、OS（オペレーティング システム）情報を参照
- sys.**dm_os_sys_info** に 3 つの新しい列が追加
  socket_count、cores_per_socket、numa_node_count
- sys.**dm_db_stats_histogram** の追加（SQL Server 2016 SP1 CU2～）
  統計情報の確認が可能。従来の DBCC SHOWSTATISTICS に相当
- sys.**dm_tran_version_store_space_usage** の追加
  データベースごとのバージョン ストア使用量を確認可能
- sys.**dm_exec_query_statistics_xml** の追加（SQL Server 2016 SP1 CU2～）
  実行中のクエリの XML 形式の実行プランを確認可能

sys.dm_os_host_info

dm_os_host_info 動的管理ビューは、OS（オペレーティング システム）に関する情報を参照できるビューです。

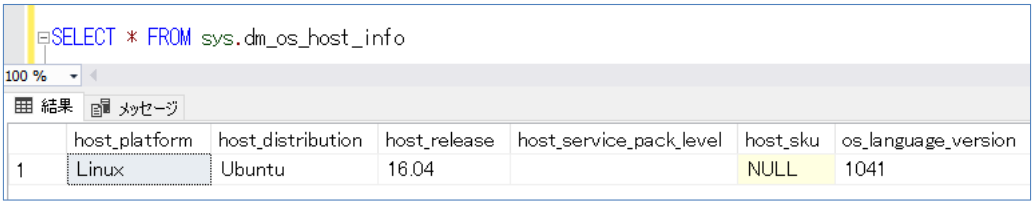
```
-- OS に関する情報を取得
SELECT * FROM sys.dm_os_host_info
```

Windows の場合



	host_platform	host_distribution	host_release	host_service_pack_level	host_sku	os_language_version
1	Windows	Windows 10 Pro	10.0		48	1041

Ubuntu の場合



	host_platform	host_distribution	host_release	host_service_pack_level	host_sku	os_language_version
1	Linux	Ubuntu	16.04		NULL	1041

sys.dm_os_sys_info に 3 つの新しい列が追加

dm_os_sys_info 動的管理ビューでは、CPU のソケット数（**socket_count**）やソケットあたりのコア数（**cores_per_socket**）、NUMA ノード数（**numa_node_count**）を取得できるようになりました。

```
-- CPU のソケット／NUMA に関する情報を取得
SELECT socket_count, cores_per_socket, numa_node_count, *
 FROM sys.dm_os_sys_info
```

```
-- CPU のソケット／NUMA に関する情報を取得
SELECT socket_count, cores_per_socket, numa_node_count, *
FROM sys.dm_os_sys_info
```

100 %

結果 メッセージ

	socket_count	cores_per_socket	numa_node_count	cpu_ticks	ms_ticks	cpu_count	hyperthread_ratio
1	2	1	1	2677546108806964	723461173	2	1

sys.dm_db_stats_histogram の追加

この動的管理ビューは、SQL Server 2016 SP1 の CU2 から利用することができますが、以前のバージョンでの **DBCC SHOWSTATISTICS** コマンドに相当する結果を取得できるものです。

```
-- インデックスの統計情報を取得
USE idxTestDB
SELECT * FROM sys.dm_db_stats_histogram(OBJECT_ID('t1'), 1)
```

```
USE idxTestDB
SELECT * FROM sys.dm_db_stats_histogram(OBJECT_ID('t1'), 1)
```

t1 テーブルの
IndexID = **1** の統計を取得

100 %

結果 メッセージ

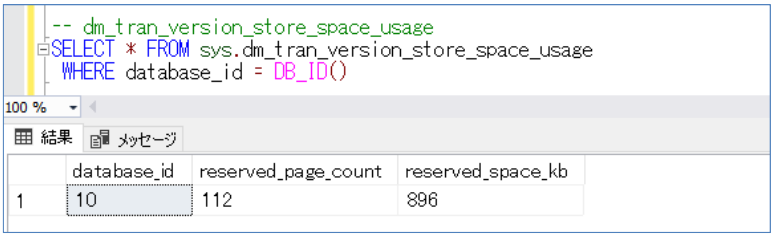
	object_id	stats_id	step_number	range_high_key	range_rows	equal_rows	distinct_range_rows	average_range_rows
1	1029578706	1	1	590	0	1	0	1
2	1029578706	1	2	4312	5078.25	1	3721	1.364754
3	1029578706	1	3	8383	3489.99	1	3490	1
4	1029578706	1	4	11722	5057.352	1	3338	1.515084
5	1029578706	1	5	16833	5266.333	1	5110	1.030594
6	1029578706	1	6	20451	5308.129	1	3617	1.46755
7	1029578706	1	7	20544	104.4907	1	92	1.135769

dm_db_stats_histogram 動的管理ビューでは、第 1 引数にテーブルの ID、第 2 引数にインデックスの ID を指定することで、該当インデックスの統計（Statistics）を参照することができます。従来の DBCC SHOWSTATISTICS コマンドだと、結果を利用するためにはテーブルを作成したりしなくてはならず面倒だったのが、このビューのおかけで結果を再利用しやすくなりました。

sys.dm_tran_version_store_space_usage の追加

dm_tran_version_store_space_usage 動的管理ビューは、**スナップショット分離レベル**や**READ_COMMITTED_SNAPSHOT** を利用している場合に、tempdb 上のバージョン ストア領域をどのぐらい消費しているのかを確認するのに役立つものです。

```
-- dm_tran_version_store_space_usage
SELECT * FROM sys.dm_tran_version_store_space_usage
 WHERE database_id = DB_ID()
```

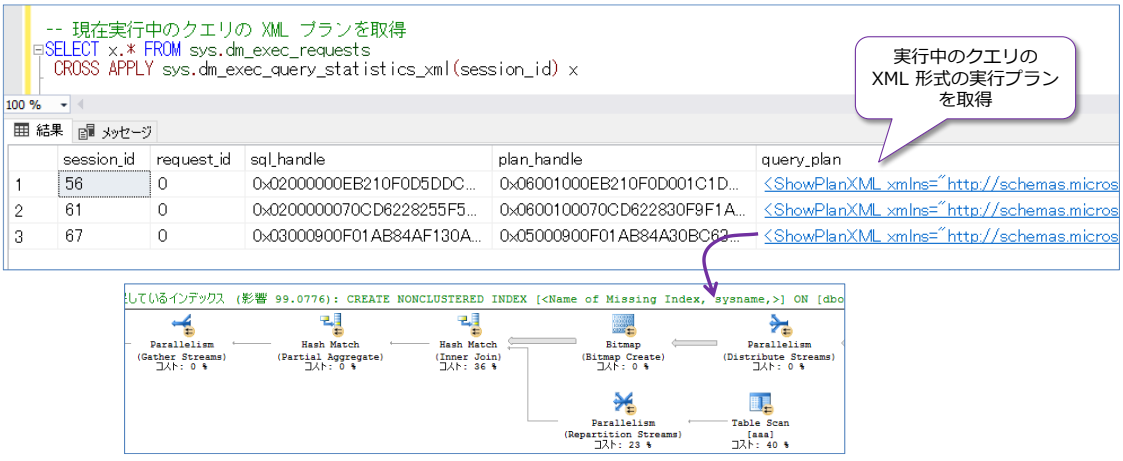


以前のバージョンでは、**dm_tran_version_store** 動的管理ビューを利用してバージョン ストアの情報を取得できましたが、この結果は、明細（Row ストア 1 件 1 件の詳細データ）であったため、利用するのは面倒でした。SQL Server 2017 からは、**dm_tran_version_store_space_usage** 動的管理ビューを利用すれば、合計値を簡単に取得できるようになったので便利です。

sys.dm_exec_query_statistics_xml の追加

この動的管理ビューは、SQL Server 2016 SP1 の CU2 から利用することができますが、実行中のクエリの XML 形式の実行プランを確認することができます。

```
-- 現在実行中のクエリの XML プランを取得
SELECT * FROM sys.dm_exec_requests
 CROSS APPLY sys.dm_exec_query_statistics_xml(session_id)
```



なお、このビューで結果を参照するには、トレースフラグ **7412** を有効化しておく必要があります。また、これを設定すると性能面でオーバーヘッドが出るので、トラブル時や性能テスト時にのみ利用することをお勧めします。

5.5 テンポラル テーブルの強化

テンポラル テーブル（Temporal Tables）は、テーブルに対する過去の更新履歴を、自動的に履歴テーブルに保存できる機能で、SQL Server 2016 から提供されました。SQL Server 2017 では、**テンポラル テーブル**が次のように強化されています。

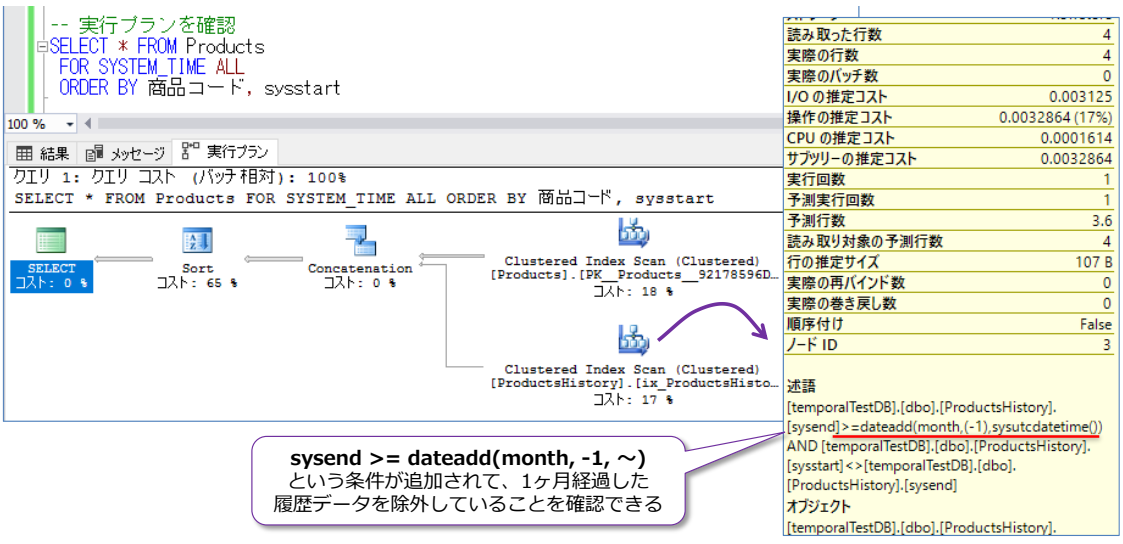
- テンポラル テーブルで**保持ポリシー**をサポート（HISTORY_RETENTION_PERIOD）
- テンポラル テーブルで **CASCADE DELETE** と **CASCADE UPDATE** をサポート

テンポラル テーブルの保持ポリシー

テンポラル テーブルでは、履歴テーブル（History Table）のデータを保持する期間を設定できるようになりました。例えば、1ヶ月間に設定したい場合は、次のように「**HISTORY_RETENTION_PERIOD = 1 MONTHS**」と設定します。

```
-- テンポラル テーブルに保持ポリシーを設定
CREATE TABLE Products
(  商品コード int NOT NULL PRIMARY KEY CLUSTERED
  ,商品名 nvarchar(80)
  ,sysstart datetime2 (0) GENERATED ALWAYS AS ROW START
  ,sysend datetime2 (0) GENERATED ALWAYS AS ROW END
  ,PERIOD FOR SYSTEM_TIME (sysstart, sysend)
) WITH
 ( SYSTEM_VERSIONING = ON
     (  HISTORY_TABLE = dbo.ProductsHistory,
        HISTORY_RETENTION_PERIOD = 1 MONTHS ) )
```

MONTHS の部分は、異なる時間単位（**DAYS**、**WEEKS**、**YEARS**）を指定することもできます。このように、保持ポリシーを設定すると、履歴データへのアクセス時に、次のように内部的にフィルターが設定されて、古いデータを参照できないようになります（また、古いデータは自動削除されます）。



5.6 SQL CLR のセキュリティ強化

SQL Server 2017 では、**SQL CLR**（CLR 統合機能）のセキュリティも強化されています。その主なものは,次のとおりです。

- sp_configure オプションの **clr strict security** が既定でオン
- 信頼できる CLR のホワイト リスト機能の追加
  (sp_add_trusted_assembly、sp_drop_trusted_assembly、sys.trusted_assemblies)

SQL Server 2017 では、信頼できる CLR を**ホワイト リスト**に追加したり、削除できるようになりました。**sp_add_trusted_assembly** でホワイト リストに追加、**sp_drop_trusted_assembly** でホワイト リストから削除、**sys.trusted_assemblies** ビューでホワイト リストの確認をすることができます。

ホワイト リストへの登録は、次のように行えます。

```
DECLARE @asmBin varbinary(max) = 0x4D5A900003000000040000000FFFF00 ～
DECLARE @hash varbinary(64) = HASHBYTES('SHA2_512', @asmBin)

-- ホワイトリスト
EXEC sp_add_trusted_assembly @hash
,N'プロジェクト名, version=0.0.0.0, culture=neutral, publickeytoken=null
  ,processorarchitecture=msil'
```

SQL CLR のバイナリに対して **HASHBYTES** 関数で **SHA2_512** を利用してハッシュ値を生成して、それを **sp_add_trusted_assembly** の第 1 引数に与えます。

SQL CLR のセキュリティ強化については、以下のオンライン ブックのトピックが参考になります。

CLR の厳密なセキュリティ
https://docs.microsoft.com/ja-jp/sql/database-engine/configure-windows/clr-strict-security

sys.sp_add_trusted_assembly (Transact-SQL)
https://docs.microsoft.com/ja-jp/sql/relational-databases/system-stored-procedures/sys-sp-add-trusted-assembly-transact-sql

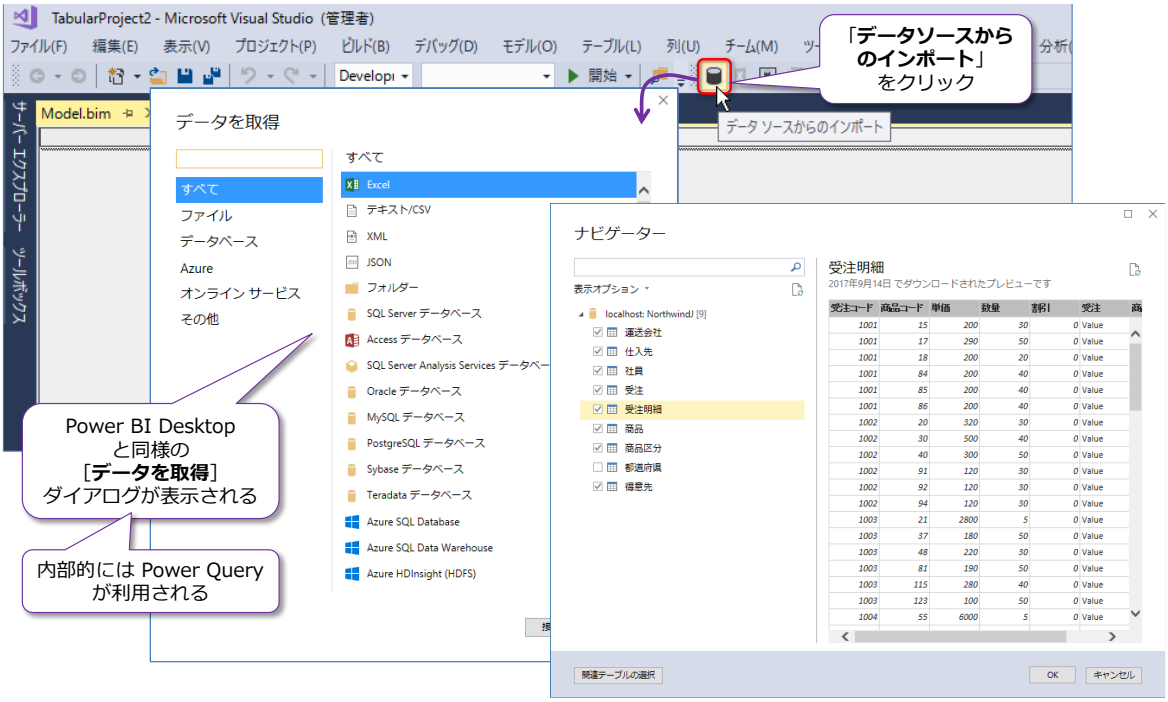
5.7 BI 機能の強化

SQL Server 2017 では、BI 機能も強化されています。

- **Analysis Services**（SSAS）の強化
  Power Query Formula Language への対応、データ ソースの取得が Power Query に変更（Power BI Desktop におけるデータ ソースの取得と同様のインターフェースに変更）、ドリルスルー データの指定、Ragged 階層対応、Object レベル セキュリティ、DAX 強化、DMV 強化、DISCOVER_CALC_DEPENDENCY
- **Reporting Services**（SSRA）の強化
  レポート コメント、DAX クエリ デザイナー for SSDT の提供、REST API 対応など。軽量のインストーラーに変更（SQL Server 2017 のインストールとは切り離されて、インストーラーを別途ダウンロードして、インストールする形に変更）
- **Integration Services**（SSIS）の強化
  スケール アウト、Linux 対応など
- マスター データ サービス（MDS）の強化

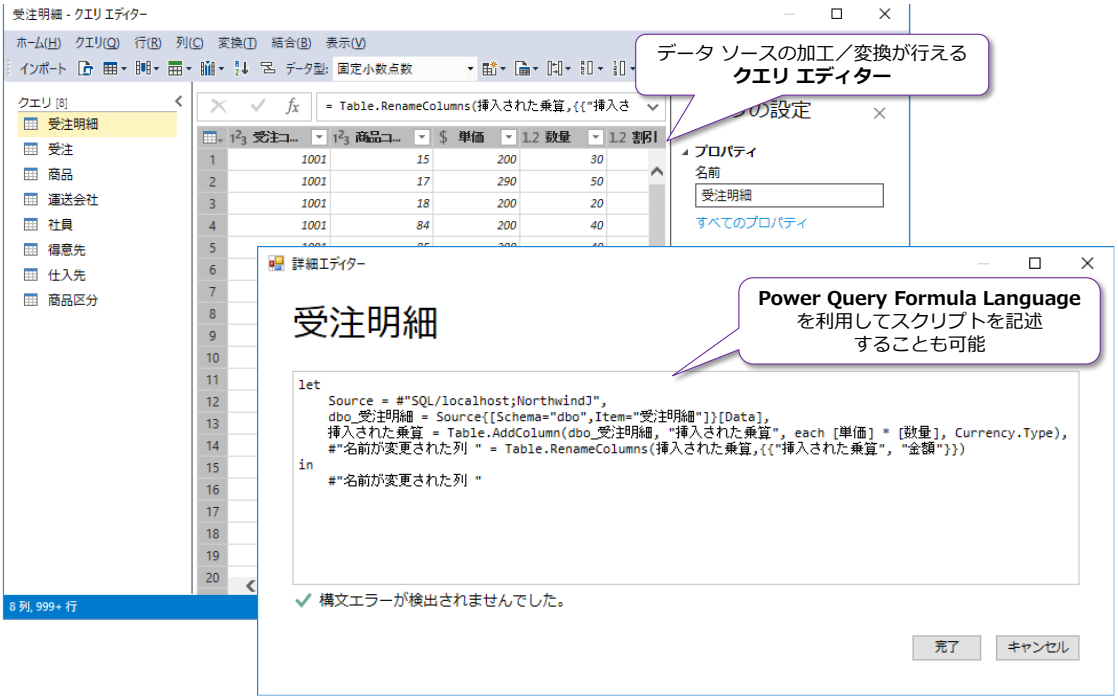
Analysis Services の強化（Power Query Formula Language への対応 etc）

SQL Server 2017 の Analysis Services は、データ ソースの取得が **Power Query** に変更されました（SQL Server 2017 からの新しい互換性レベル **1400** を利用する場合）。これは、**Power BI Desktop** ツールで利用されていたデータ ソースの取得と同様のインターフェースになり、**Power Query Formula Language**（通称 M 言語）にも対応したことになります。



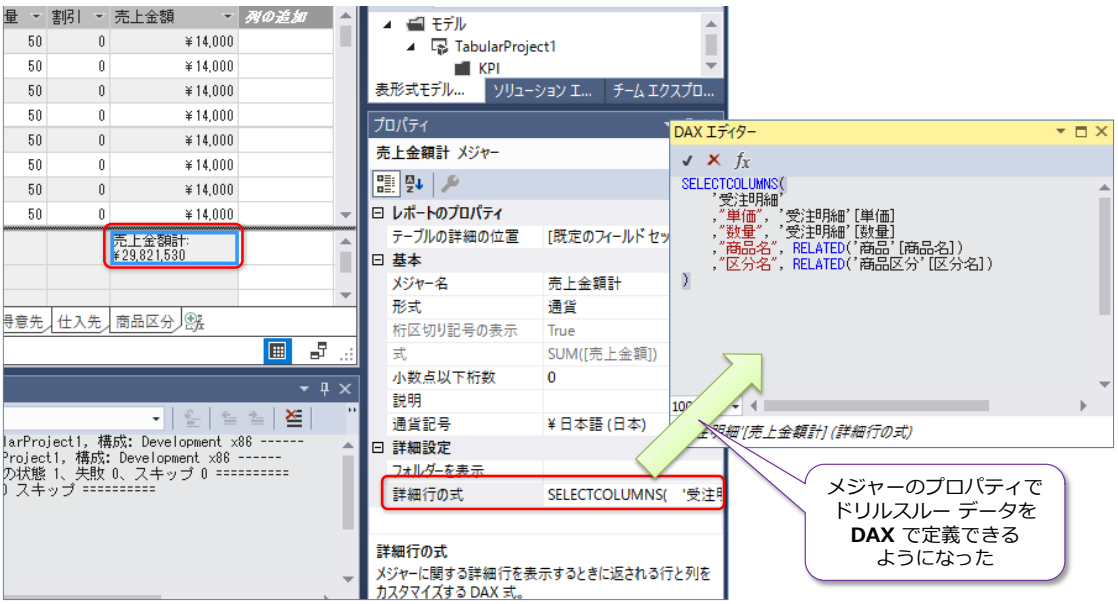
これによって、Power BI Desktop でサポートされていた様々なデータ ソースを Analysis

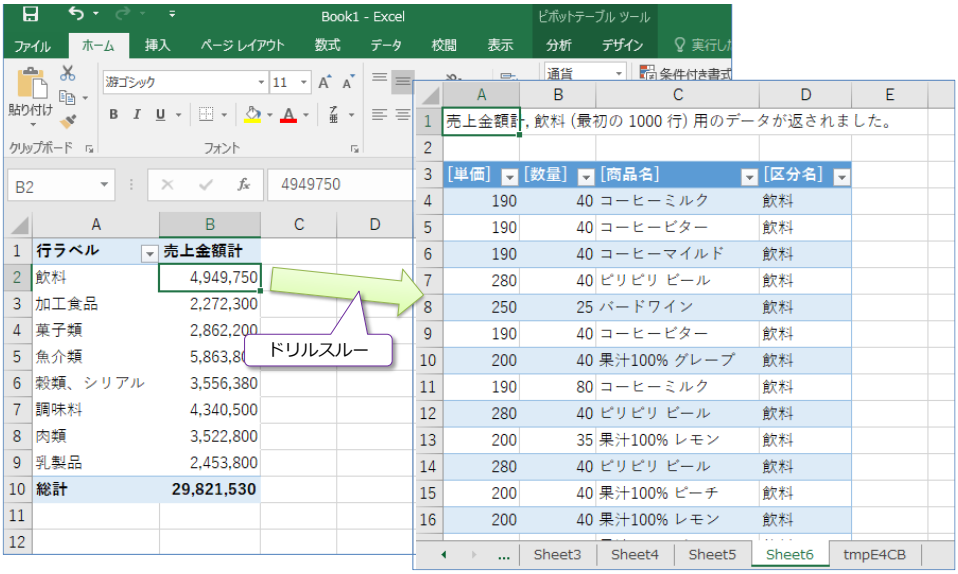
Services でも利用できるようになり、かつ Power Query Formula Language を利用して、データ ソースの加工／変換（いわゆる ETL 処理）も行えるようになりました。



なお、Power Query ではなく、従来通りのデータ ソースの取得方法も利用可能で、この場合は、Analysis Services のプロジェクトを作成するときに、互換性レベルで「**1200**」（SQL Server 2016 レベル）を選択するようにします。

ドリルスルー データについては、次のようにメジャーのプロパティで［**詳細行の式**］が追加されて、**DAX 式**でドリルスルー データを指定できるようになりました。





その他の Analysis Services の新機能（Ragged 階層、オブジェクト レベル セキュリティ、DAX 強化、DISCOVER_CALC_DEPENDENCY など）については、Analysis Services Team の以下のブログ記事が参考になります。

What's new for SQL Server vNext on Windows CTP 1.1 for Analysis Services
https://blogs.msdn.microsoft.com/analysisservices/2016/12/16/whats-new-for-sql-server-vnext-on-windows-ctp-1-1-for-analysis-services/

What's new in SQL Server 2017 CTP 2.0 for Analysis Services
https://blogs.msdn.microsoft.com/analysisservices/2017/04/19/whats-new-in-sql-server-2017-ctp-2-0-for-analysis-services/

What's new in SQL Server 2017 CTP 2.1 for Analysis Services
https://blogs.msdn.microsoft.com/analysisservices/2017/05/18/whats-new-in-sql-server-2017-ctp-2-1-for-analysis-services/

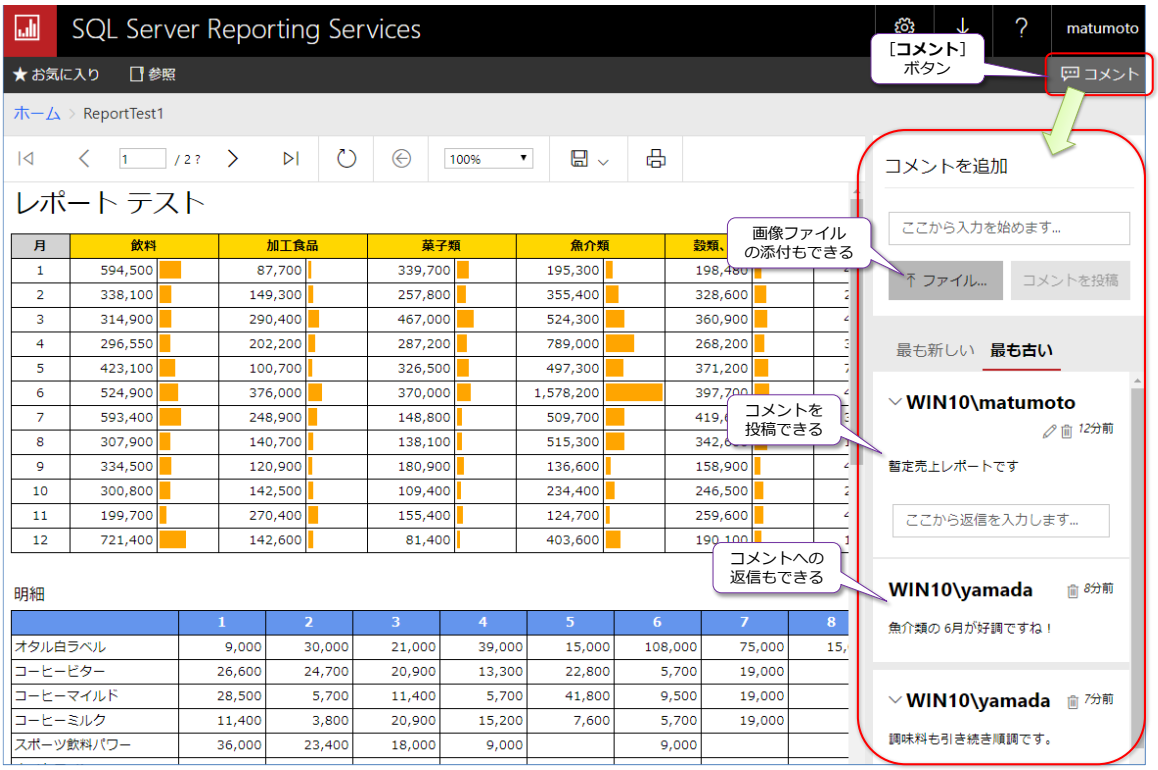
What's new in SQL Server 2017 RC1 for Analysis Services
https://blogs.msdn.microsoft.com/analysisservices/2017/07/17/whats-new-in-sql-server-2017-rc1-for-analysis-services/

Introducing a Modern Get Data Experience for SQL Server vNext on Windows CTP 1.1 for Analysis Services
https://blogs.msdn.microsoft.com/analysisservices/2016/12/16/introducing-a-modern-get-data-experience-for-sql-server-vnext-on-windows-ctp-1-1-for-analysis-services/

Reporting Services の強化、変更点

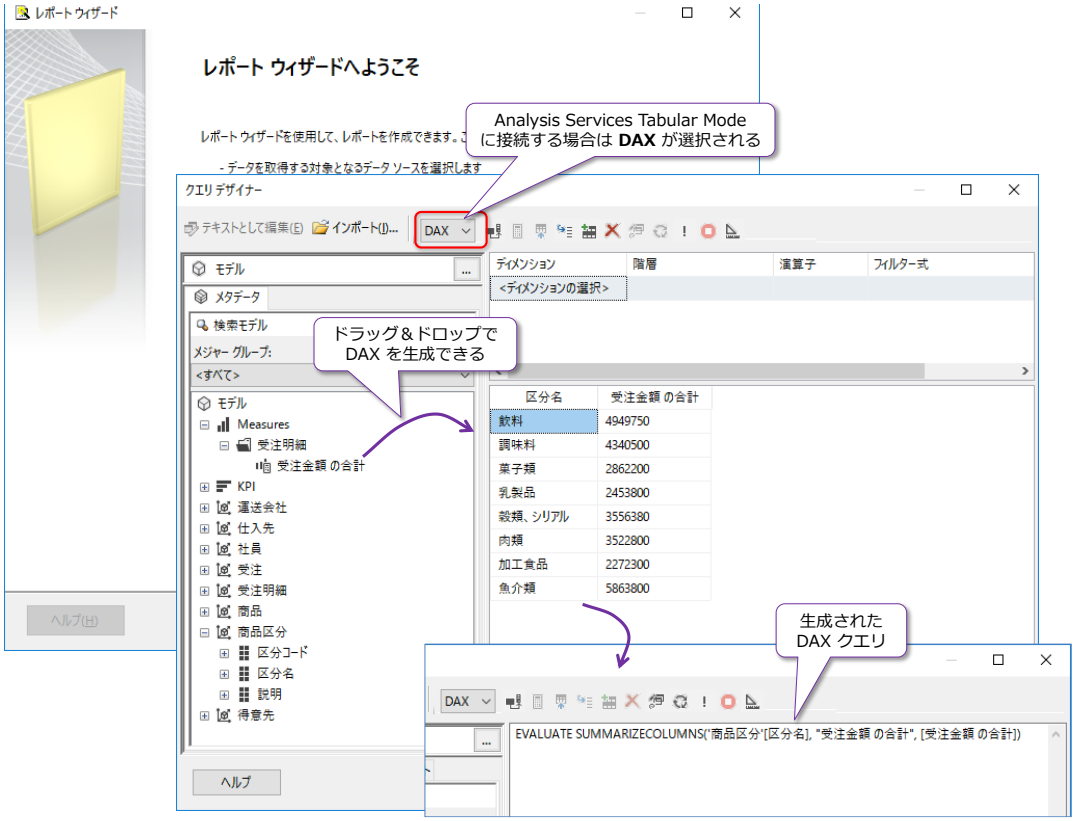
Reporting Services では、**レポート コメント**機能の追加や、**DAX クエリ デザイナー**の提供（レポート ビルダーで提供されていた DAX クエリ デザイナーを SSDT でも利用可能に）、**REST API** のサポートなどが強化されています。

レポート コメントは、次のようにレポートに対して、任意のユーザーがコメントを追加できる機能です。



レポート ポータルでレポートを表示すると、[**コメント**] ボタンをクリックできるようになっていて、ここでコメントを追加することができます。

DAX クエリ デザイナー for SSDT は、次のように利用できます（Analysis Services Tabular Mode を選択時は、DAX が既定値になります）。



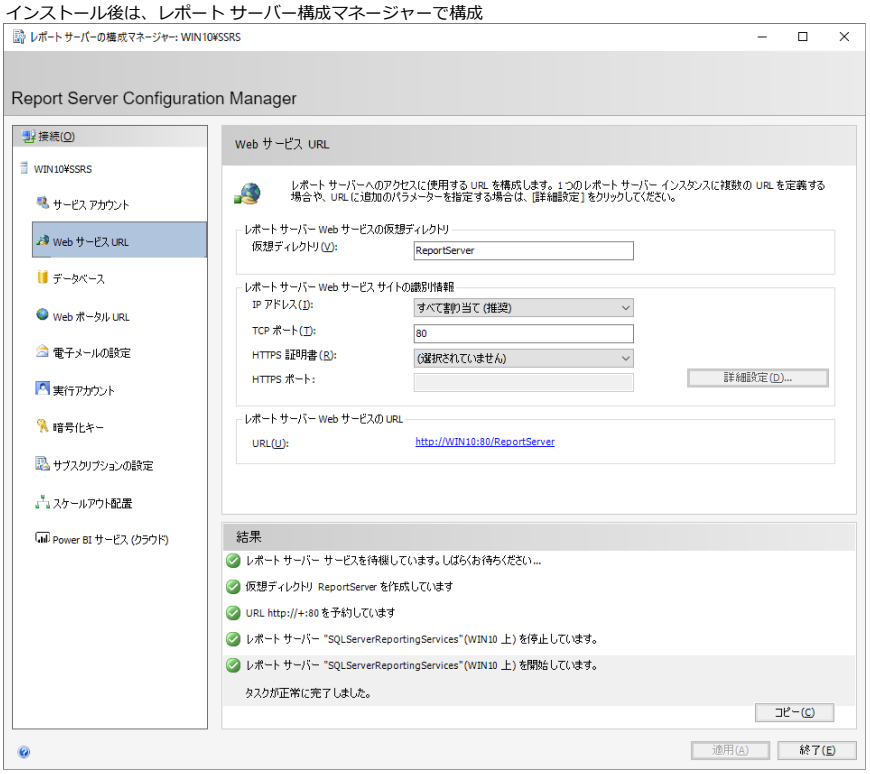
Reporting Services のダウンロード／インストール

Reporting Services は、SQL Server 2017 のセットアップ（インストーラー）からは切り離されて、別途ダウンロードして、インストールする形に変更されました。Reporting Services のダウンロードは、次の URL から行うことができます。

Microsoft SQL Server 2017 Reporting Services
https://www.microsoft.com/ja-JP/download/details.aspx?id=55252

Reporting Services のインストール

インストール後は、レポート サーバー構成マネージャーで構成



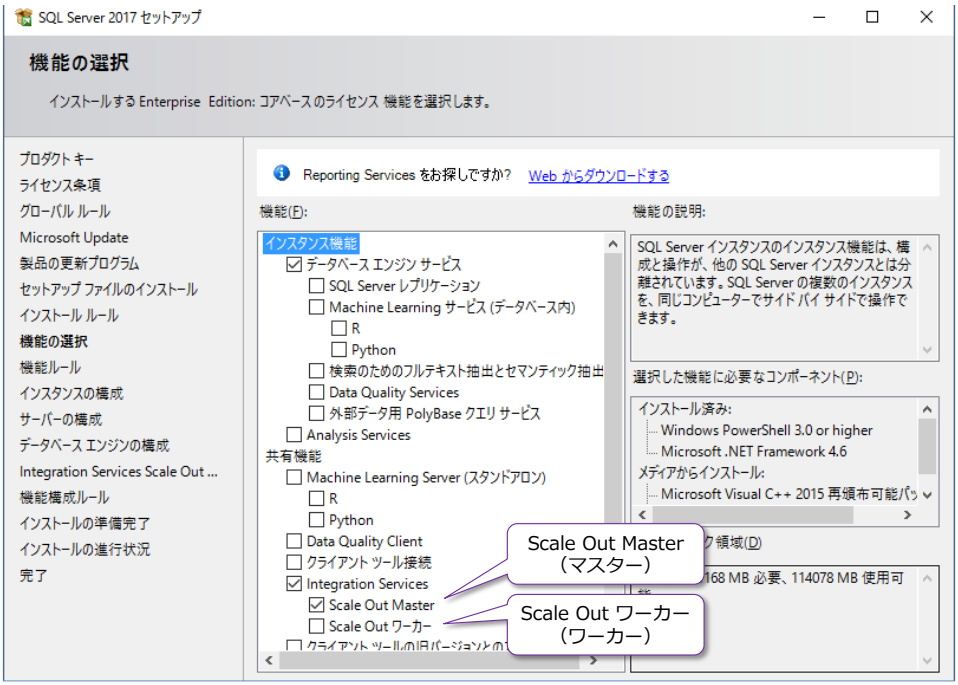
その他の Reporting Services の新機能については、Reporting Services Team の以下のブログ記事が参考になります。

SQL Server 2017 Reporting Services now generally available
https://blogs.msdn.microsoft.com/sqlrsteamblog/2017/10/02/sql-server-2017-reporting-services-now-generally-available/

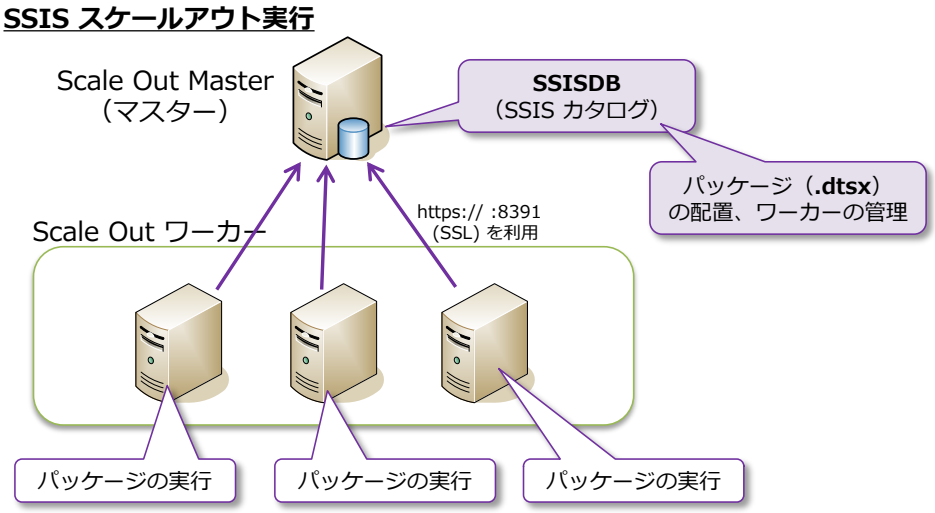
Integration Services の新機能（スケールアウト、Linux 対応）

Integration Services では、パッケージの**スケールアウト実行**や、**Linux** 対応などが強化されています。

スケールアウトを構成するには、SQL Server 2017 のインストール時の［**機能の選択**］ページで、次のように「**Scale Out Master**」または「**Scale Out ワーカー**」を選択します。



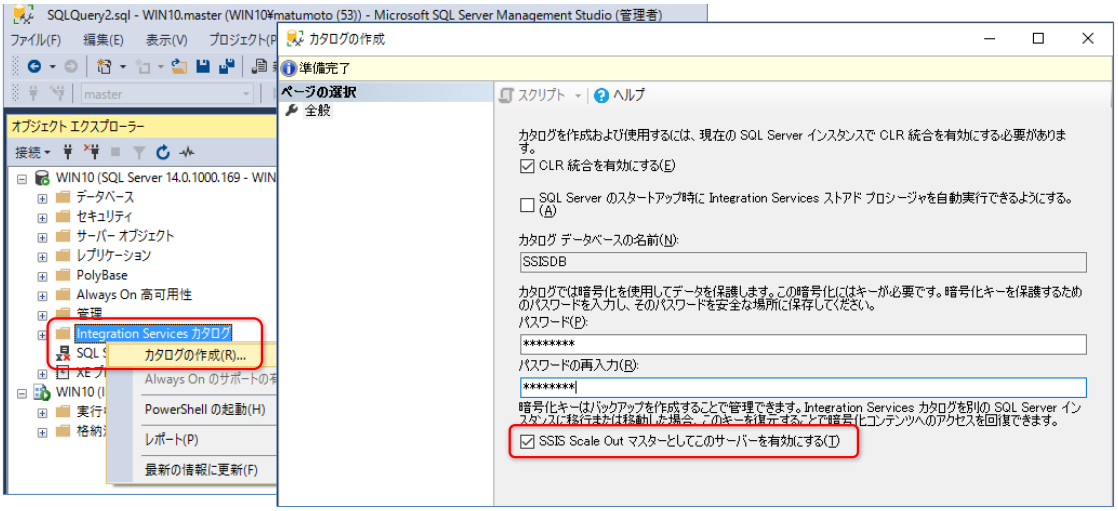
Integration Services のスケールアウトは、次のように**マスター**と**ワーカー**で構成されます。



マスターは、**SSISDB**（SSIS カタログ データベース）を保持して、ここへパッケージを配置（Deploy）します。また、パッケージの実行指示（どのワーカーで実行するのか）や、実行の進行状況の確認といったワーカーの管理も行います。

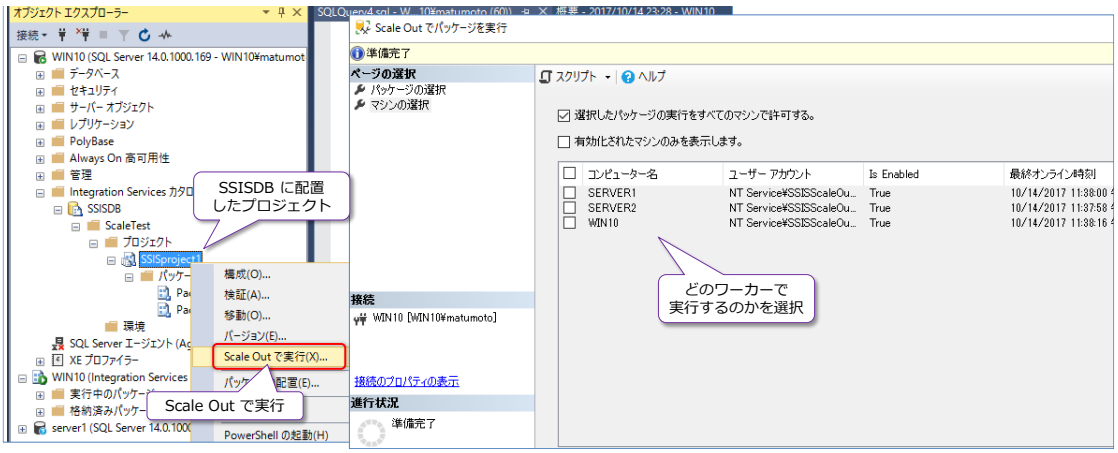
なお、インストール時にマスターとワーカーを両方選択した場合は、マスター兼ワーカーにすることもできます。

マスターでは、SSISDB を作成する際に、次のように「**SSIS Scale Out マスターとしてこのサーバーを有効にする**」をチェックするようにします。



マスターとして有効化した後は、次のようにワーカーを登録して、有効化できるようになります。

あとは、SSISDB カタログに配置（Deploy）したパッケージ（.dtsx）を実行するときに、[**Scale Out で実行**] をクリックすれば、スケールアウト実行することができます。



おわりに

最後まで試された皆さん、いかがでしたでしょうか？ SQL Server 2017 には、たくさんの新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。SQL Server がついにマルチ プラットフォーム化して、Mac OS や Linux で動かせるようになったのは本当に衝撃です。Visual Studio Code などのアプリケーション開発ツールを利用すれば、Windows を利用することなく、Mac OS や Linux だけでアプリケーション開発（C# や ASP.NET、Java、Python、node.js など）もできてしまいます。

SQL Server 2017 on Linux は、ただ単に Linux 上で SQL Server を動かせるようにしただけはなく、ほとんどの SQL Server の機能（データベース エンジンに関する機能）を、Windows 上の SQL Server とまったく同じように利用できるというところは本当にびっくりしました（クエリ ストアも、自動チューニングも、データ パーティションも、インメモリ OLTP も、各種のセキュリティ機能も全く同じスクリプトで利用することができます）。もし、Windows 上の Management Studio から Linux 上の SQL Server をリモート操作できるなら、Linux 上で動作させているのを忘れてしまうぐらい、まったく同じように動作させることができます。

また、Python をデータベース エンジンに統合（ビルトイン）したことによって、ディープ ラーニング（GPU 利用も OK）も可能になりました。最近の SQL Server の進化は、昨今のマイクロソフト社の方向性と同様、オープン化が目覚ましく、今までの Windows や .NET にとらわれないアプリケーション開発が行えるのは、本当にいろいろな可能性を感じさせてくれます。

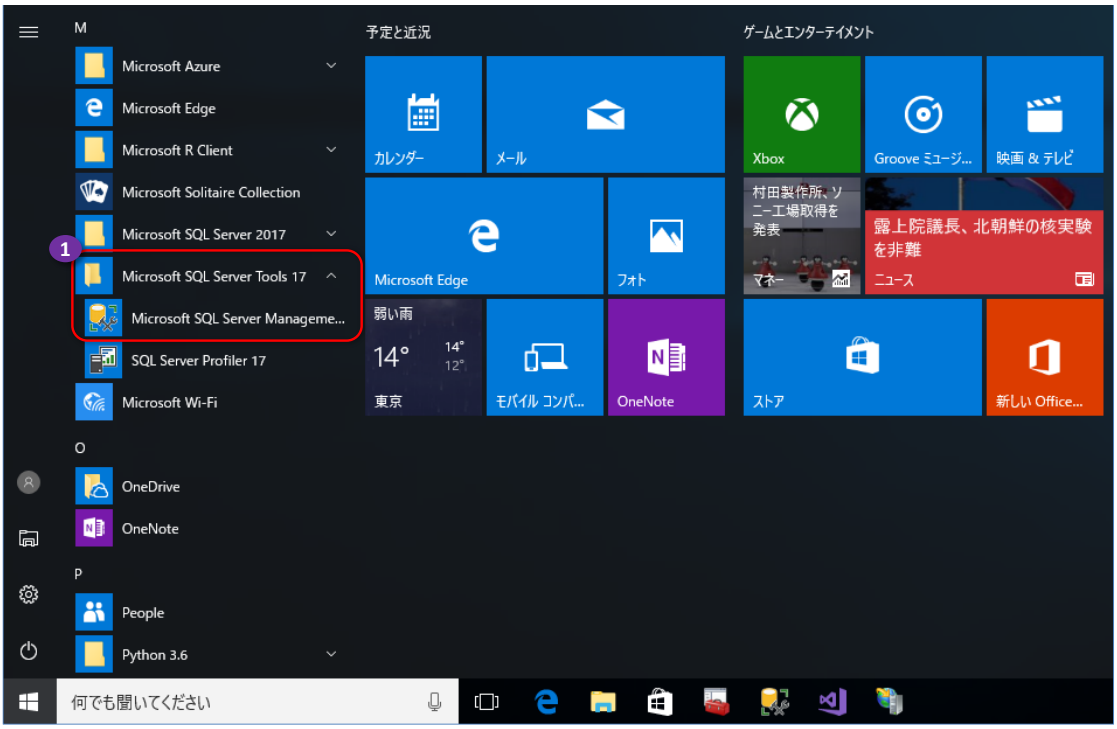
もちろん、SQL Server のデータベース エンジンとしての、既存機能の強化も怠っておらず、ついに自動チューニング機能の搭載や、クエリ ストアの強化（クエリの Wait 情報を過去に遡って見られるのは本当に便利です）、列ストア インデックス／インメモリ OLTP のさらなる進化、クラスター レス可用性グループ、スマート バックアップなど、役立つ機能が盛りだくさんです。

また、BI 機能も進化しており、Analysis Services に Power BI（データ取得部分での Power Query）の統合、Reporting Services のレポート コメント、Integration Services のスケールアウト実行など、現場で役立つ機能が提供されています。

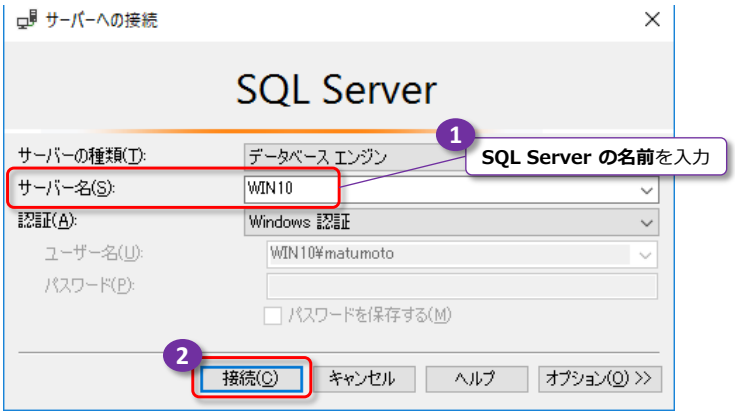
5.8 付録：サンプル データベース（NorthwindJ）の作成

この自習書で利用しているサンプル データベース「**NorthwindJ**」を作成する手順は、次のとおりです。

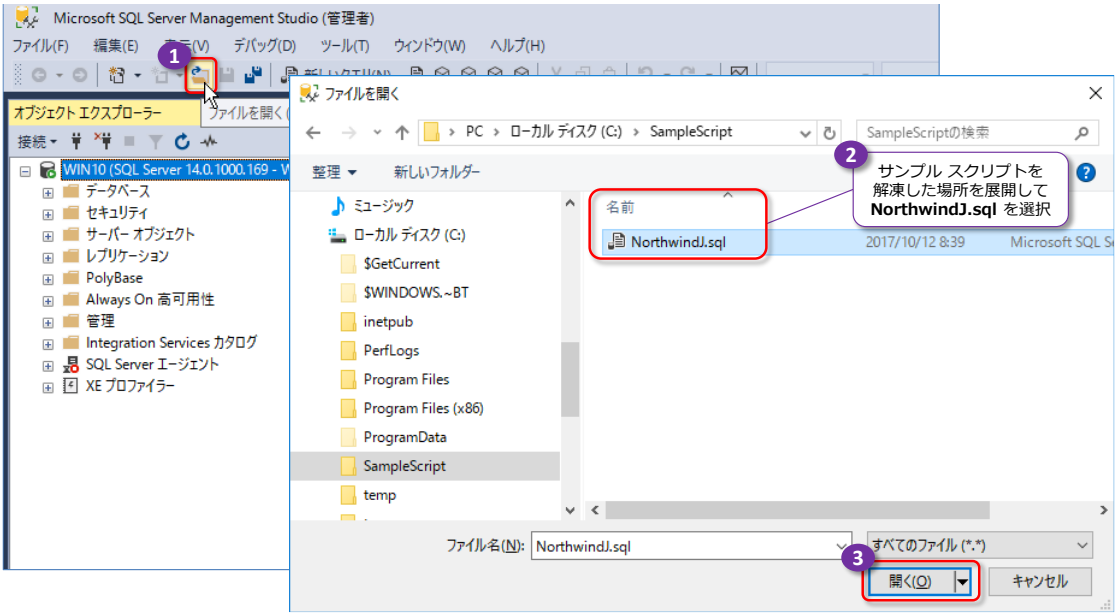
1. まずは、Management Studio を起動するために、[**スタート**] メニューの [**Microsoft SQL Server Tools 17**] から [**Microsoft SQL Server Management Studio 17**] をクリックします。



2. 起動後、[**サーバーへの接続**] ダイアログが表示されたら、[**サーバー名**] に SQL Server の名前を入力して、[**接続**] ボタンをクリックします。

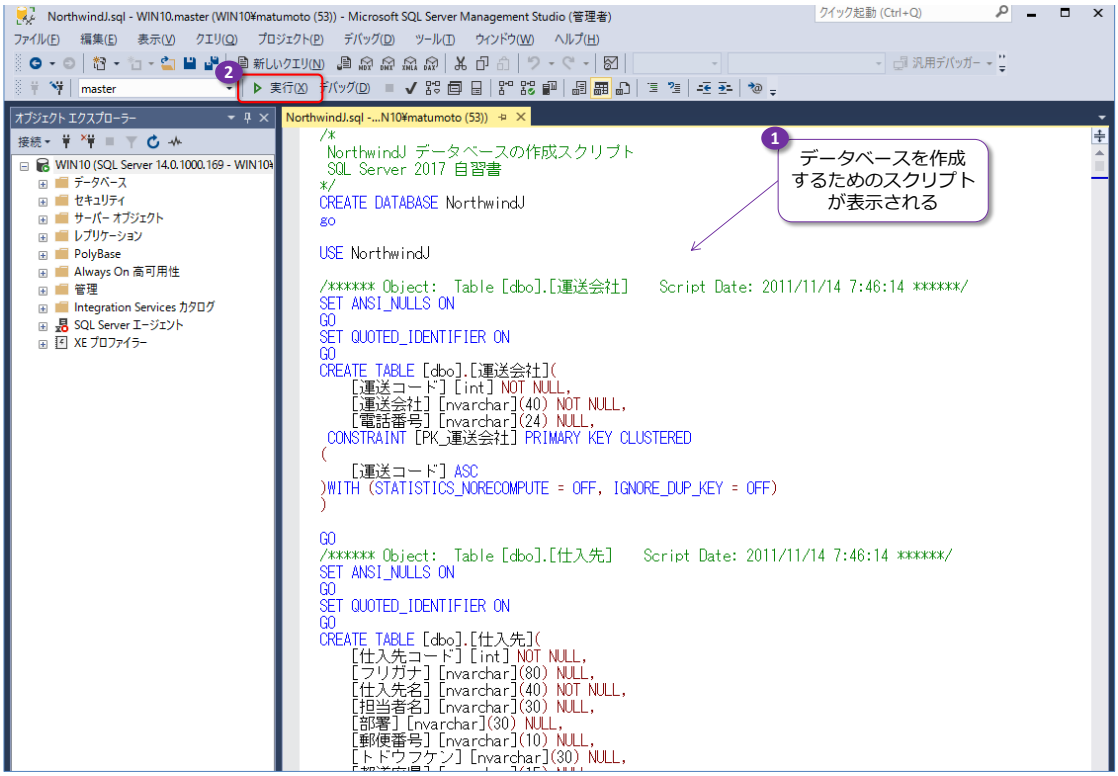


3. 接続完了後、次のようにツールバーの [**ファイルを開く**] ボタンをクリックします。

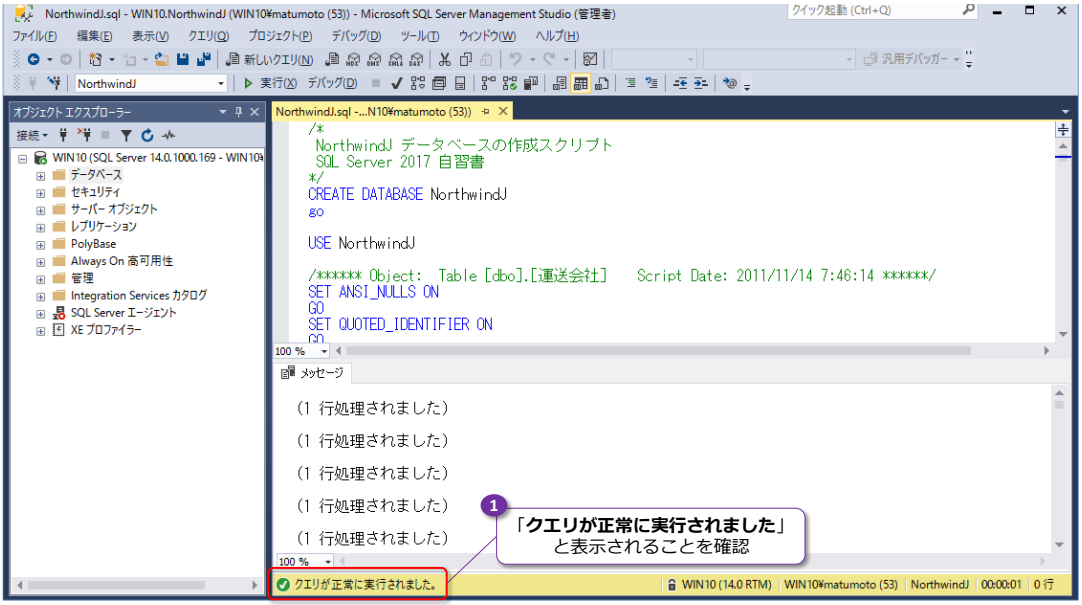


これにより、[**ファイルを開く**] ダイアログが表示されるので、サンプル スクリプトを解凍したフォルダーを展開して、「**NorthwindJ.sql**」ファイルを選択し、[**開く**] ボタンをクリックします。

4. 次のようにデータベースを作成するためのスクリプトが表示されるので、ツールバーの [**実行**] ボタンをクリックして、スクリプトを実行します。



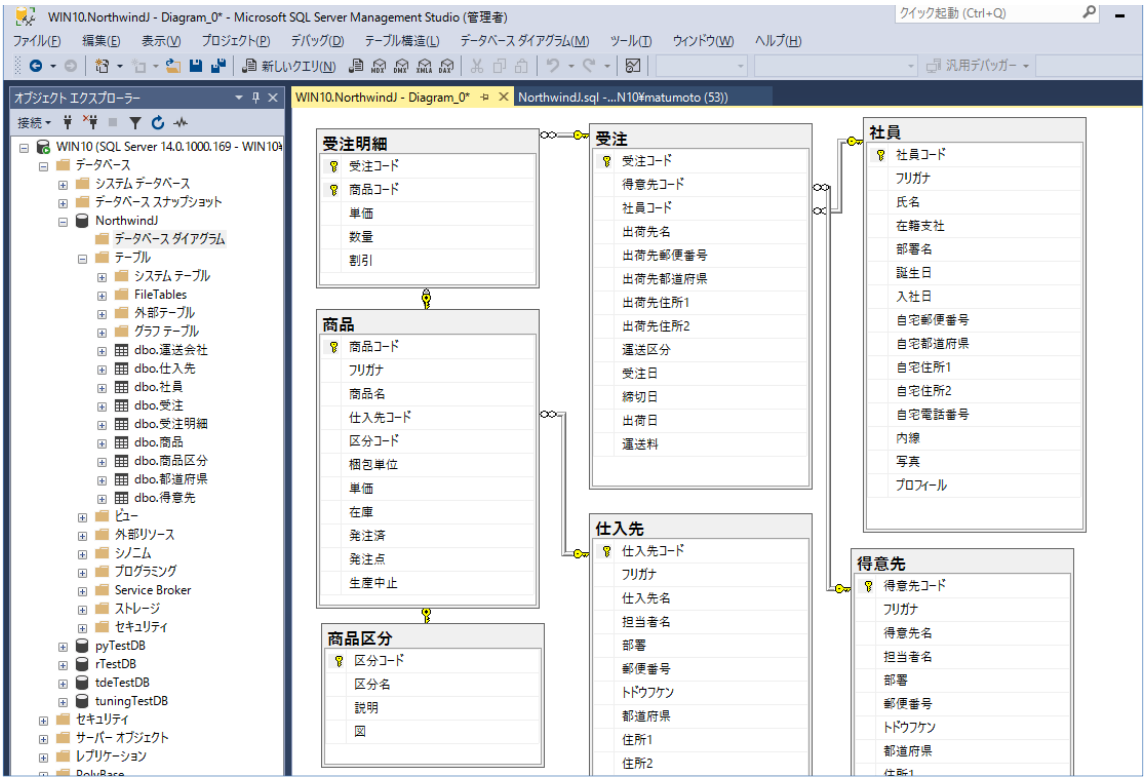
5. 数秒後に実行が完了して、次のように画面下に「**クエリが正常に実行されました**」と表示されることを確認します。



以上でデータベースの作成が完了です。

NorthwindJ データベースの構成

NorthwindJ データベースは、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、この自習書の手順を試すために、一部のデータを加工したものです。具体的なスキーマ構成は次のとおりです。



このデータベースは、商品の販売管理を題材として、「**商品**」や「**商品区分**」、「**受注**」、「**受注明細**」テーブルなどが格納されています。

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/）

SQLQuality（エスキューエル・クオリティ）は、**日本で唯一の SQL Server 専門の独立系コンサルティング会社**です。過去のバージョンから最新バージョンまでの SQL Server を知りつくし、多数の実績と豊富な経験を持つ、OS や .NET にも詳しい **SQL Server の専門家（キャリア 20 年以上）がすべての案件に対応します**。人気メニューの「**パフォーマンス チューニング サービス**」は、100%の成果を上げ、過去すべてのお客様環境で驚異的な性能向上を実現。チューニング スキルは**世界トップレベル**を自負、検索エンジンでは（英語情報を含めて）ヒットしないノウハウを多数保持。ここ数年は **BI／DWH システム構築支援**のご依頼が多く、支援だけでなく実際の構築も行う。

主なコンサルティング実績／構築実績

- 大手製造業の「**CAD 端末の利用状況の見える化**」システム構築
  Oracle や CSV（Notes）、TSV ファイル、Excel からデータを抽出し、SQL Server 2012 上に DWH を構築
  見える化レポートには Reporting Services を利用
- 大手映像制作会社の **BI システム構築**（会計／業務システムにおける予実管理／原価管理など）
  従来 Excel で管理していたシートを Reporting Services のレポートへ完全移行。
  Oracle や勘定奉行からデータを抽出して、SQL Server 上に DWH を構築
- 大手流通系の **DWH／BI システム構築支援**（POS データ／在庫データ分析／ABC 分析／ポイントカード分析）
- 大手アミューズメント企業の **BI システム構築支援**（人事システムにおける人材パフォーマンス管理）
  Reporting Services による勤怠状況の見える化レポートの作成、PostgreSQL／人事システムからのデータ抽出
- 外資系医療メーカーの **BI システム**構築支援（Analysis Services と Excel による販売分析システム）
  OLAP キューブによる売上および顧客データの多次元分析／自由分析（ユーザーによる自由操作が可能）
- 大手流通系の **DWH システム**のパフォーマンス チューニング
  データ量 100 億件の DWH、**総ステップ数 2 万**越えのストアド プロシージャのパフォーマンス チューニング
- ミッション クリティカルな**金融システム**でのトラブル シューティング／定期メンテナンス支援
- SQL Server の下位バージョンからの**移行／アップグレード**支援（32 ビットから x64 への対応も含む）
- 複数台の SQL Server の **Hyper-V 仮想環境**への移行支援（サーバー統合支援）
- ハードウェア リプレース時の**ハードウェア選定**（最適なサーバー、ストレージの選定）、**高可用性環境**の構築
- **2 時間**かかっていた日中バッチ実行時間を、わずか **5 分**へ短縮（**95.8%** の性能向上）
- **Java 環境**（Tomcat、Seasar2、S2Dao）の SQL Server パフォーマンス チューニング etc

コンサルティング時の作業例（パフォーマンス チューニングの場合）

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

松本美穂（まつもと・みほ）

有限会社エスキューエル・クオリティ 代表取締役 Microsoft MVP for SQL Server（2004 年 4 月～）
経産省認定データベース スペシャリスト／MCDBA／MCSD for .NET／MCITP Database Administrator
SQL Server の日本における最初のバージョンである「SQL Server 4.21a」から SQL Server に携わり、現在、SQL Server を中心とするコンサルティングを行っている。得意分野はパフォーマンス チューニングと Reporting Services。著書の『SQL Server 2000 でいってみよう』と『ASP.NET でいってみよう』（いずれも翔泳社刊）は、トップ セラー（前者は 28,500 部、後者は 16,500 部発行）。近刊に『SQL Server 2016 の教科書』（ソシム刊）がある。

松本崇博（まつもと・たかひろ）

有限会社エスキューエル・クオリティ 取締役 Microsoft MVP for SQL Server（2004 年 4 月～）
経産省認定データベース スペシャリスト／MCDBA／MCSD for .NET／MCITP Database Administrator
SQL Server の BI システムとパフォーマンス チューニングを得意とするコンサルタント。アプリケーション開発（ASP／ASP.NET、C#、VB 6.0、Java、Access VBA など）やシステム管理者（IT Pro）経験もあり、SQL Server だけでなく、アプリケーションや OS、Web サーバーを絡めた、総合的なコンサルティングが行えるのが強み。Analysis Services と Excel による BI システムも得意とする。マイクロソフト認定トレーナー時代の 1998 年度には、Microsoft CPLS トレーナー アワード（Trainer of the Year）を受賞。