Microsoft

Azure Cosmos DB 自習書

Azure Cosmos DB Table API 編

この自習書では、Azure Table Storage と互換性のある Azure Cosmos DB の Table API をサポートするテーブル データベースの使用方法を学習します。

発行日：2018 年 12 月 11 日

Edifist
エディフィストラーニング株式会社

Microsoft Partner
Microsoft
Gold Learning
Gold Data Platform
Silver Data Analytics

更新履歴

版数	発行日	更新履歴
第 1 版	2018 年 12 月 11 日	初版発行

本書の取り扱い注意事項

本書は、2018 年 12 月時点の Azure Cosmos DB をベースに作成しています。読者が本書を読まれる時点では、Azure ポータルの UI や提供される機能などが変更されている可能性があることをあらかじめご了承ください。

なお、Azure サービスの更新情報に関しては、「Azure の更新情報」を参照してください。

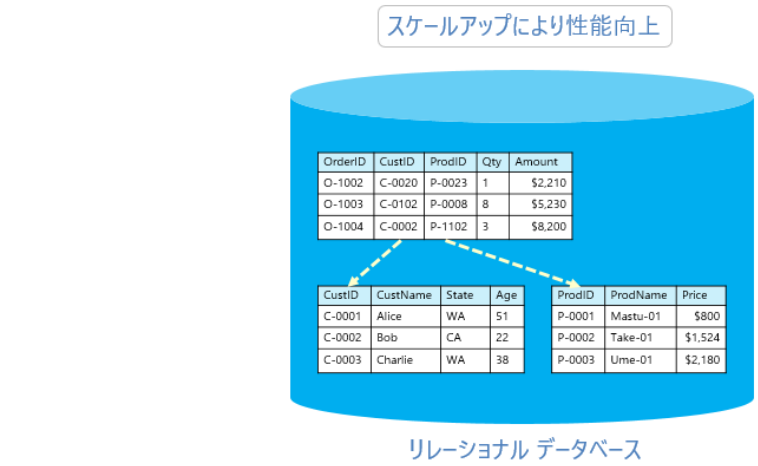
目次

1. はじめに

Azure Cosmos DB は、Microsoft Azure PaaS (Platform as a Services) 環境で提供されている NoSQL で利用可能なデータベース サービスです。Azure Cosmos DB の特徴は、キー/バリュー ストア、ドキュメント データベース、グラフ データベースといったマルチ データモデルに対応し、選択したリージョンに即座にデプロイできるグローバル分散型のサービスで、予測可能な応答性能を提供できるように設計されていることです。

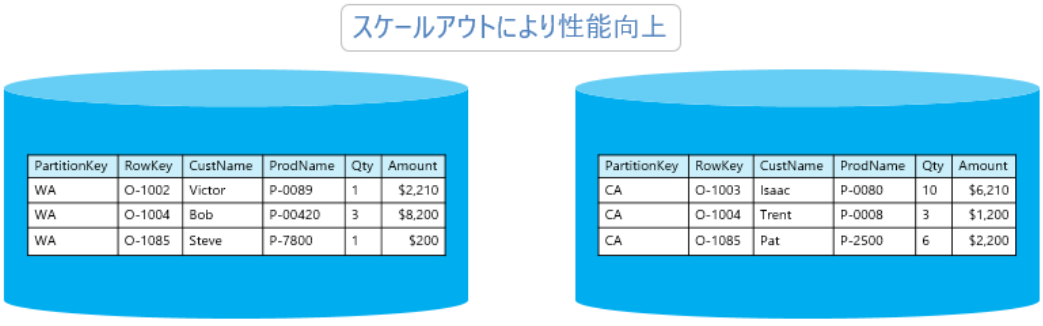
キー/バリュー ストアは、「キー (Key)」に対し「値 (Value)」が格納される NoSQL データベースです。例えば注文番号が「O-1002」で顧客名が「Justin」といったデータを格納する場合、注文番号は「キー」で、顧客名が「値」になります。また、Azure Cosmos DB のテーブルでは、値のフィールドを「プロパティ」と呼び「キー」に当たるフィールドが「システム プロパティ」となります。「プロパティ」は複数定義することができ、「キー」である注文番号に対して、「値」として「製品名」、「数量」、「請求先住所」など自由にプロパティを追加することができ、テーブルを定義した後でも、必要に応じて「値」を追加することもできます。キーと値の組を書き込むとき、キー値により、格納先のサーバーが決まります。

リレーショナルデータベースは、通常正規化されたテーブル構造を持つため、読み取りと書き込みを行うデータベースをスケールアウト構成で実装することは、簡単ではありません。



リレーショナル データベース

これに対して、キー/バリュー ストアでは、キー値により分再配置されるため、容易にスケールアウト構成を利用することができます。このため、一台のサーバーで格納できない量のデータを保持でき、一台のサーバーでは処理しきれない負荷に対応させることができます。キー/バリュー ストアは、大量のデータを格納し、安定したスループットでデータにアクセスするようなシナリオで役に立つでしょう。



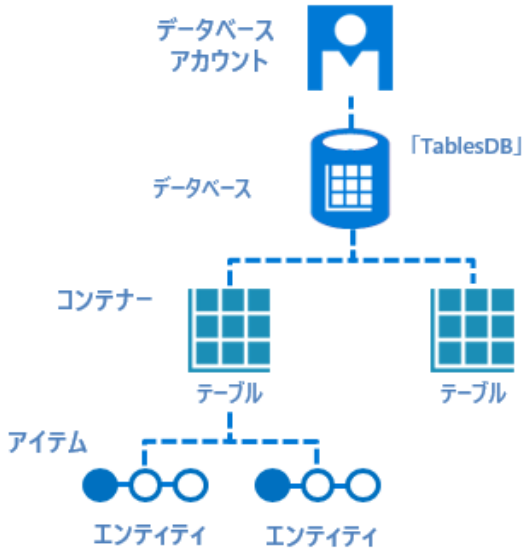
分散キー/バリュー ストア

この自習書では、Azure Cosmos DB に、キー/バリュー ストアを格納するテーブルを作成し Table API を使用して、アプリケーションからアクセスする手順を学習します。最初に、Azure ポータルから、Table API 用のデータベースを作成し、テーブルにシンプルなキー/バリュー形式の店舗情報を格納して、検索できることを確認します。その後、C# で記述したアプリケーションコードからデータベースにアクセスして、Azure ポータルから行った操作をコードで実行する手順をステップ バイ ステップ形式で学習します。なお、本書に記載されたサンプル コードは、Windows 10 で実行することを前提とします。

1.1 Azure Cosmos DB の分散キー・バリュー ストア

Azure Cosmos DB のテーブルは、Azure Storage サービスのキー/バリュー ストアとして提供されている Azure Table Storage 互換の PaaS データ サービスです。Azure Table Storage を使用しているアプリケーションの接続情報を Azure Cosmos DB に変更することで低テイテンシ、高スループットなデータストアを利用することができ、Azure Table Storage では、サポートされない地理レプリケーションやマルチ マスター構成などを利用できます。

Azure Cosmos DB のリソース モデルは、下図のようにデータベース アカウントに従属する階層構造になっています。



データベース アカウントのデプロイ時に使用する API を指定する必要がありますが、キー/バリュー ストアをサポートするテーブル データベースを作成するには、「Azure Table API」を選択してください。なお、一度選択した API は、後から変更することができないことに注意してください。この手順は「2.2.1 Azure ポータルからの Azure Cosmos DB データベース アカウントの作成」で実施します。

また、ドキュメント データベースやグラフ データベースでは、Azure Cosmos DB アカウントに任意の数のデータベースを作成できますが、テーブルを格納するデータベースは名前が「TablesDB」で固定され、データベース アカウントに作成できるのは、1 個だけであることに注意してください。データベースに対して、「コンテナー」として使用されるテーブルは複数個作成できます。テーブルに格納されるアイテムは、「エンティティ」と呼ばれています。

テーブルは、名前から推測できるように、エンティティを表形式で格納しますが、このエンティティの構成方法について理解しておくべき、いくつかの作法があります。

まず、各エンティティを一意に識別するためのキー フィールドは、PartitionKey と RowKey の 2 つのシステム プロパティと、Azure Cosmos DB がエンティティの最終更新日時をトラッキングするために内部的に値が自動生成される Timestamp 列を持ちます。

PartitionKey	RowKey	Timestamp	ShopName	ShopOwner	PhoneNumber
Texas	a0@adventure-works.com	2018-11-22T00:50:32Z	Two-Seater Bikes	Leonetti A.	645-555-0193
Texas	abigail0@adventure-works.com	2018-11-22T00:50:34Z	Genial Bike Associates	Gonzalez Abigail	121-555-0139
California	abraham0@adventure-works.com	2018-11-22T00:50:38Z	Wheel Gallery	Swearengin Abraham	926-555-0136
Texas	aidan0@adventure-works.com	2018-11-22T00:50:42Z	Paint Supply	Delaney Aidan	358-555-0188
Illinois	alan1@adventure-works.com	2018-11-22T00:49:45Z	Shipping Specialists	Brewer Alan	494-555-0134
Oregon	alan4@adventure-works.com	2018-11-22T00:49:05Z	Cycle Clearance	Steiner Alan	792-555-0194

1 個のエンティティは 2 MB までで、最大 255 個までのプロパティを含むことができます。これには、PartitionKey、RowKey、Timestamp の 3 個のシステム プロパティも含まれます。PartitionKey と RowKey は、ともに 文字列型で、値の挿入と置き換えは利用者側で行えますが、Timestamp 値は、サービスが管理するため、利用者は変更することができません。テーブルに格納される各エンティティは、リレーショナル データベース テーブル内の主キーと同様に PartitionKey と RowKey の組み合わせは一意である必要があります。キーとバリューの組からなるエンティティを書き込む際、どのサーバーに格納させるかは PartitionKey ごとに決まります。

なお、Azure Table Storage では、セカンダリ インデックスが作成されないため、PartitionKey と RowKey の 2 つのプロパティだけにインデックスが付けられていましたが、Azure Cosmos DB は、すべてのプロパティに自動的にインデックスが作成されるように変更されています。

また、Azure Cosmos DB では、論理パーティション (最大 10 GB) のサーバー インフラストラクチャ (物理パーティション) への配置は透過的、かつ自動的に管理され、コンテナーのスケーラビリティとパフォーマンスの要求は効率的に満たされます。

スループットとストレージ要件が上がると Azure Cosmos DB は、自動的に論理パーティションをより多くのサーバーに分散配置して、負荷を低減します。クエリでは、PartitionKey と RowKey の両方を指定します。このため、どのプロパティを PartitionKey および RowKey に設定するのかは、非常に重要な設計となります。格納される値とクエリのアクセス パターンを考慮して、エンティティが論理パーティション間で均等に分散されるように PartitionKey を選択してください。

Azure Cosmos DB のテーブルにエンティティを追加する場合、以下のデータ型を使用することができます。

データ型	説明
String	• 最大 64 KB までの文字列
Boolean	• ブール値
Binary	• 最大 64 KB までのバイト配列
DateTime	• 協定世界時 (UTC) で表される 64 ビット値 (1601年1月1日午前12時00分 ～ 9999年12月31日)
Double	• 64 ビットの浮動小数点値
Guid	• 128 ビットのグローバル一意識別子
Int32	• 32 ビットの整数値
Int64	• 64 ビットの整数値

Azure Cosmos DB のテーブル データ モデルの詳細は、英語のドキュメントになりますが、Understanding the Table Service Data Model を参照してください。

エンティティをリレーショナル データベースの行として考えると、Azure Cosmos DB のテーブル構造は、リレーショナル データベースのテーブルと似ているようにも見えますが、リレーショナル データベースと異なるのは、スキーマがないため、エンティティごとにプロパティのデータ型は同じである必要がない点です。

また、Azure Cosmos DB のテーブルは、数十億ものデータ エンティティを含むクラウド アプリケーションのためのデータと大量のデータ トランザクションをサポートできるように設計されています。

Azure Cosmos DB のテーブルは、構造化された非リレーショナル データを格納するのに最適です。Azure Cosmos DB のテーブルは、通常次のような用途で使用されます。

- ログ、アドレス帳、デバイス情報、サービスに必要なメタデータなどのシンプルでフラットなデータをグローバル分散させる
- スケールアウト構成の Web アプリケーションにサービスを提供する数テラ バイトの構造化データの格納
- 高速アクセスのために非正規化されたデータセットの格納
- クラスター化インデックスを使用した高速なデータ アクセスを実行するクエリのためのデータの格納
- WCF Data Service .NET ライブラリで OData プロトコル、LINQ クエリを使用してアプリのためのデータの格納

2. Azure ポータルからのデータベースの作成と操作

この章では、Azure ポータルを使用して、実際に Azure Cosmos DB にテーブルを作成するためのデータベースをデプロイし、シンプルなキー/バリュー型のエンティティを格納して、Azure ポータルの [Data Explorer] ブレードでエンティティを検索する手順を試してみましょう。

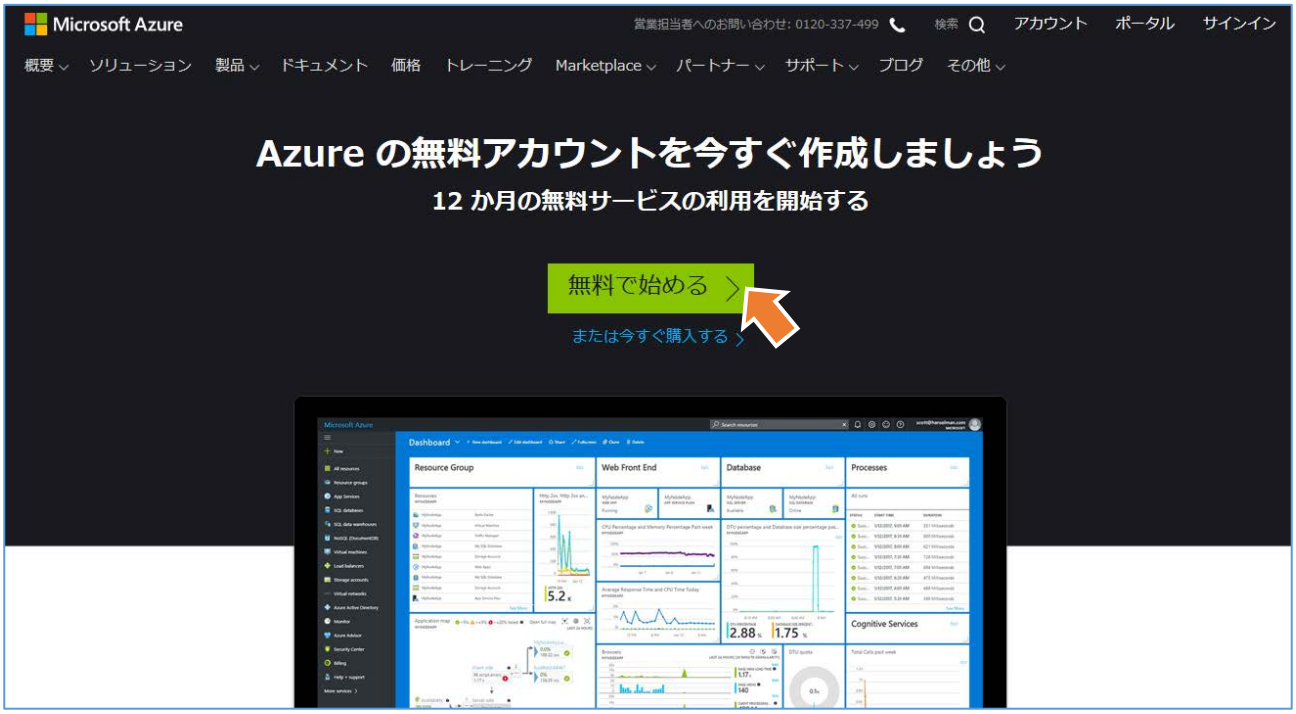
2.1 自習環境の準備

2.1.1Microsoft Azure 無料評価版のサインアップ手順

この手順はオプションです。Microsoft Azure サブスクリプションをお持ちでない場合、次の手順を実行し、Microsoft Azure の 1 か月間の無料評価版のサブスクリプションを取得してください。なお、この手順を完了するには、下記が必要となりますので予めご用意ください。

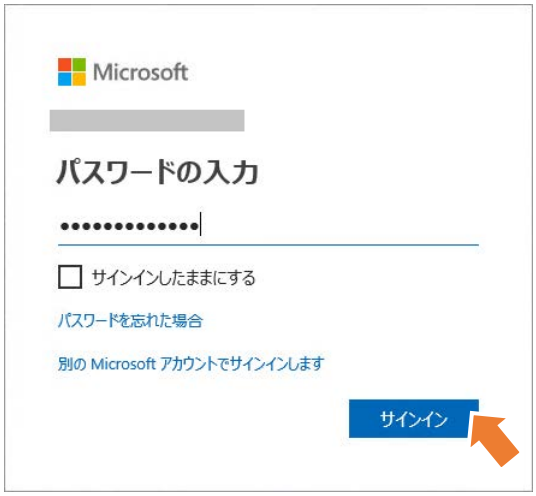
- 確認コードを音声または、ショートメッセージ (SMS) で受け取るための電話番号
- 身元確認のためのクレジットカード

1. Web ブラウザーを起動し、https://azure.microsoft.com/ja-jp/free にアクセスします。
2. [Azure の無料アカウントを今すぐ作成しましょう] が表示されるので、[無料で始める] をクリックします。



3. [サインイン] が表示されます。Microsoft アカウントのメール アドレスを入力し、[次へ] ボタンをクリックします。

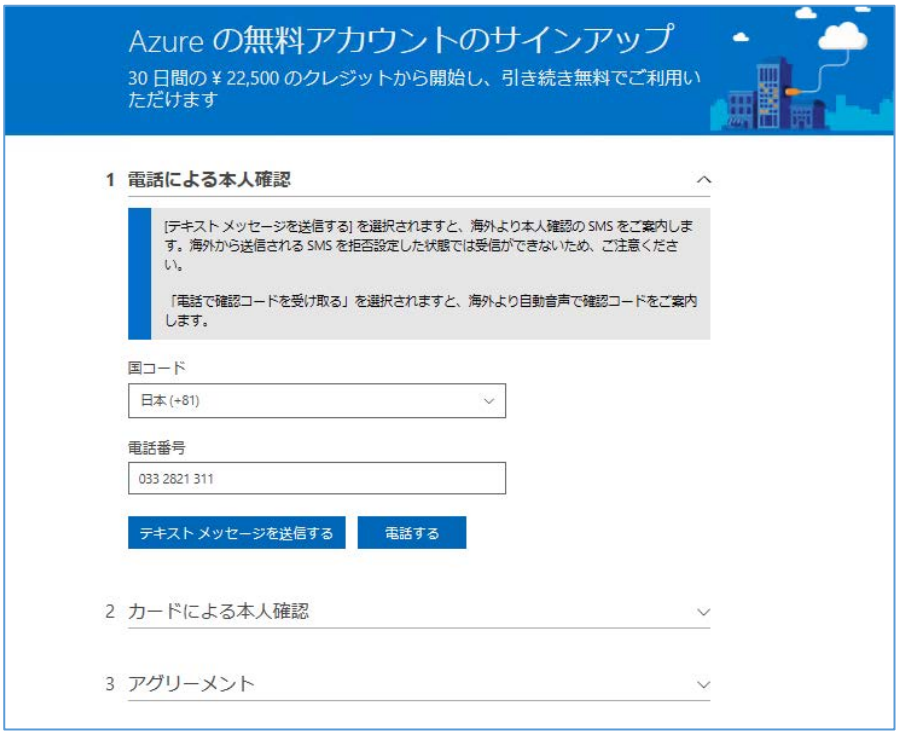
4. パスワードを入力して、[サインイン] をクリックします。



5. 「サインインの状態を維持しますか?」 が表示されたら、[はい] をクリックします。

6. 使用している Microsoft アカウントに Azure サブスクリプションが紐付いていない場合、[Azure の無料アカウントのサインアップ] が表示されるので、サインアップに必要な以下の情報を設定します。

- 電話による本人確認
- カードによる本人確認
- アグリーメント



本人確認が完了したら、[アグリーメント] で [サインアップ] をクリックします。

ワン ポイント
Microsoft Azure の 1 か月間の無料評価版では、2018 年 12 月時点で 22,500 円相当の無料評価利用分が利用可能です。なお、無料評価版の利用は 1 回までとなっており、過去すでに利用された方は無料評価版にサインアップいただけませんので、ご注意ください。なお、無料評価版の使用制限に達した場合の動作については、以下の情報も参考にしてください。 ● [お知らせ] 無料評価版の使用制限に達した場合の動作について http://blogs.msdn.com/b/dsazurejp/archive/2012/12/28/windows-azure-90-days-free-offering.aspx 引き続き、ご使用される場合は、有償のサブスクリプションに切り替えていただきますようお願いいたします。また、既に MSDN Subscription をお持ちの方やスタートアップ企業の方、大学などの教育機関でご利用の方は、以下より詳細をご確認ください。 ● MSDN サブスクライバー向けの Azure の特典 https://azure.microsoft.com/ja-jp/pricing/member-offers/msdn-benefits-details/ ● スタートアップのための Azure https://azure.microsoft.com/ja-jp/overview/startups/ ● 教育機関でのご利用 https://www.microsoftazurepass.com/azureu

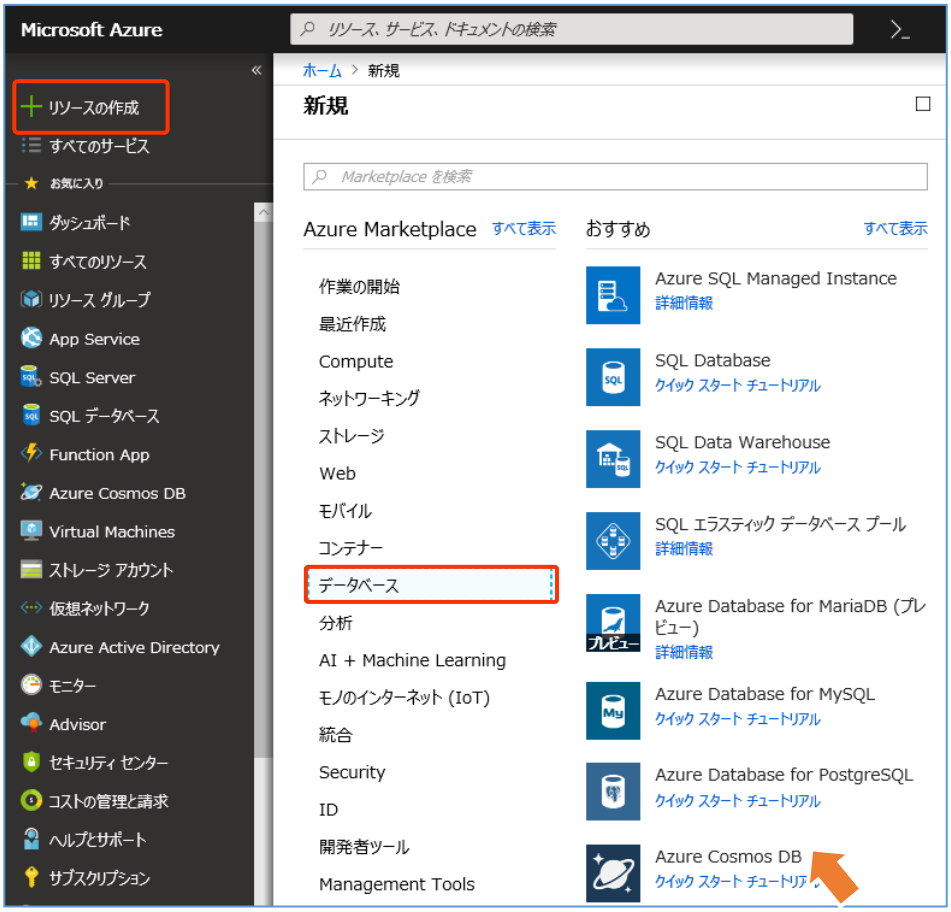
2.2 Azure ポータルからのデータベースの作成と検索

ここでは、キー/バリュー ストアの操作に慣れていただくため、Azure ポータルから以下の操作を行います。

- Table API を使用する Azure Cosmos DB データベース アカウントの作成
- データベースとテーブルの作成
- エンティティの追加と検索

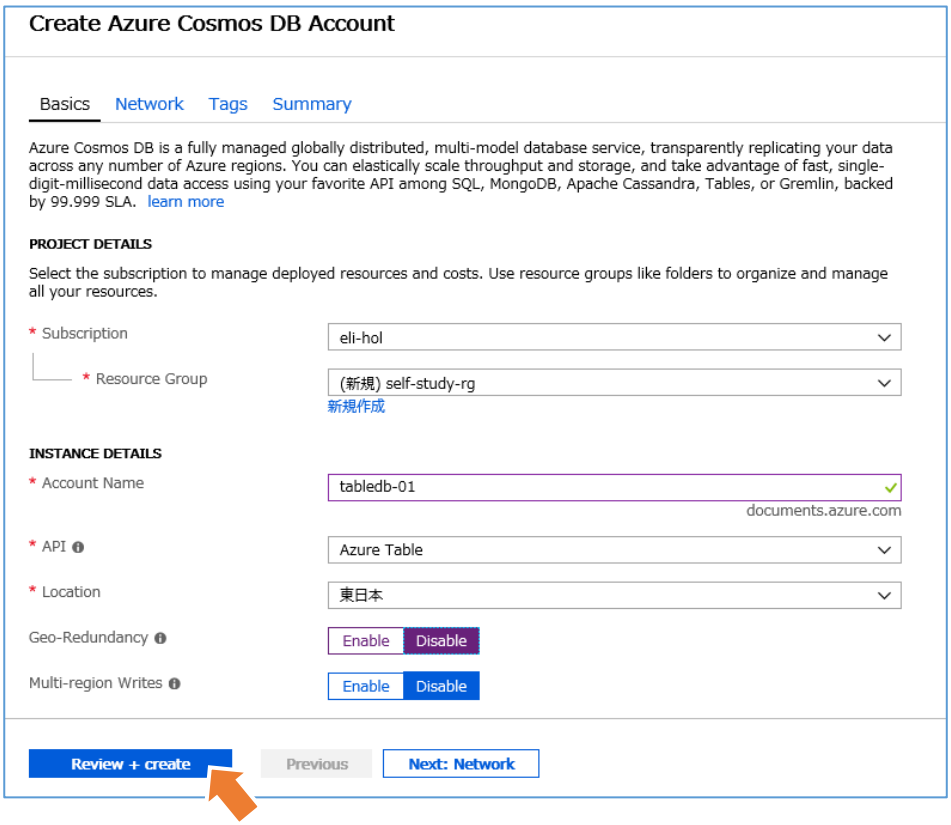
2.2.1 Azure ポータルからの Azure Cosmos DB データベース アカウントの作成

1. Azure 管理ポータルのハブ メニューから [リソースの作成] ⇒ [データベース] ⇒ [Azure Cosmos DB] をクリックします。



2. [Create Azure Cosmos DB Account] ブレードの [Basics] ページが開くので、下表のパラメーター値を設定して [Review + create] ボタンをクリックします。

設定項目	設定値
Subscription	サブスクリプションを選択します。
Resource Group	[新規作成] を選択して、自習書の作業手順で使用するためのリソースグループを作成します。
Account Name	Azure データ センター内で一意となる任意の値 (小文字・数値・ハイフン '-' のみを使用) を入力します。
API	[Azure Table] を選択します。
Location	任意のリージョンを選択します。
Geo-Redundancy	[Disable] を選択します。
Multi-region Writes	[Disable] を選択します。



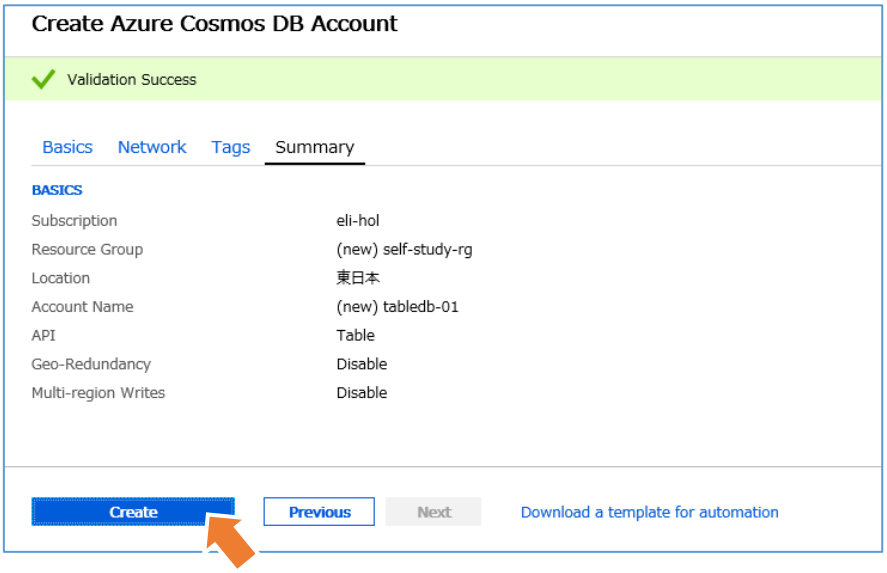
ワン ポイント

本自習書では、Azure Cosmos DB の ID として、「tabledb-01」を使用していますが、ID は Azure データ センター内で一意となる任意の値である必要があるため、既にこの名前が使用されている場合、異なる名前を使用していただく必要があることに注意してください。

実運用環境でデータ アクセスを効率よく行うには、[Location] で Cosmos DB のデータを利用者するアプリケーションと同じ場所を選択してください。また、「Geo-Redundancy」を有効化すると、プライマリ リージョンが存在する地域のペアとして使用されるデータセンターにデータベースが複製されます。さらに、デプロイ完了後、Azure ポータルの [Replicate data globally] ブレードから、任意のリージョンをデータベースの複製先として追加できます。また、[Multi-region Writes] を有効化すると、複製先のリージョンでの書き込みを可能にするマルチマスターを構成できます。

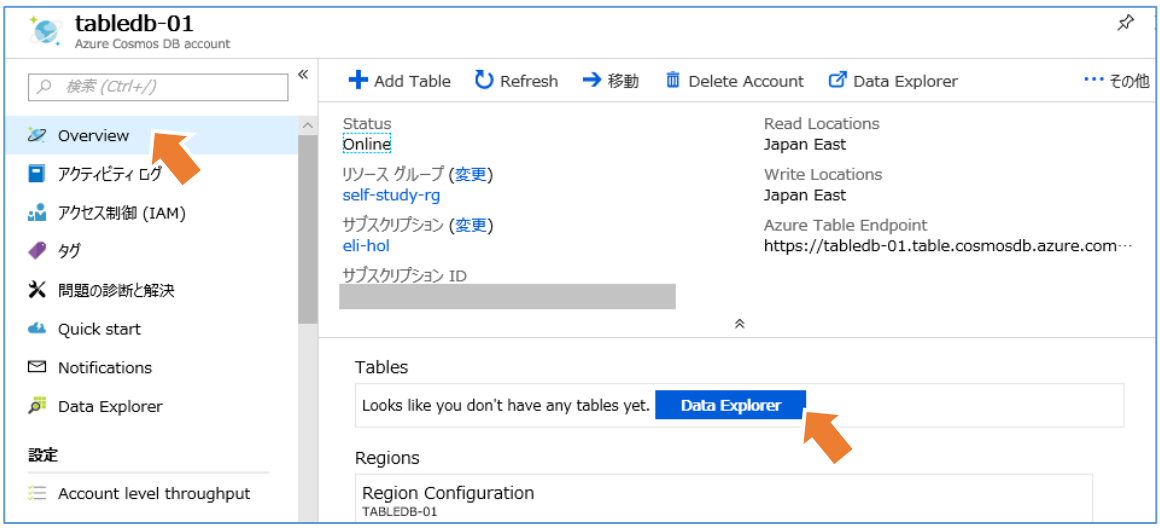
なお、仮想ネットワークに配置したコンピューティング リソースからのアクセスさせる場合は、[Network] タブを使用して許可する Azure 仮想ネットワークを指定してください。

3. [Create Azure Cosmos DB Account] ブレードの [Summary] タブが表示されるので、設定内容を確認して [Create] ボタンをクリックします。



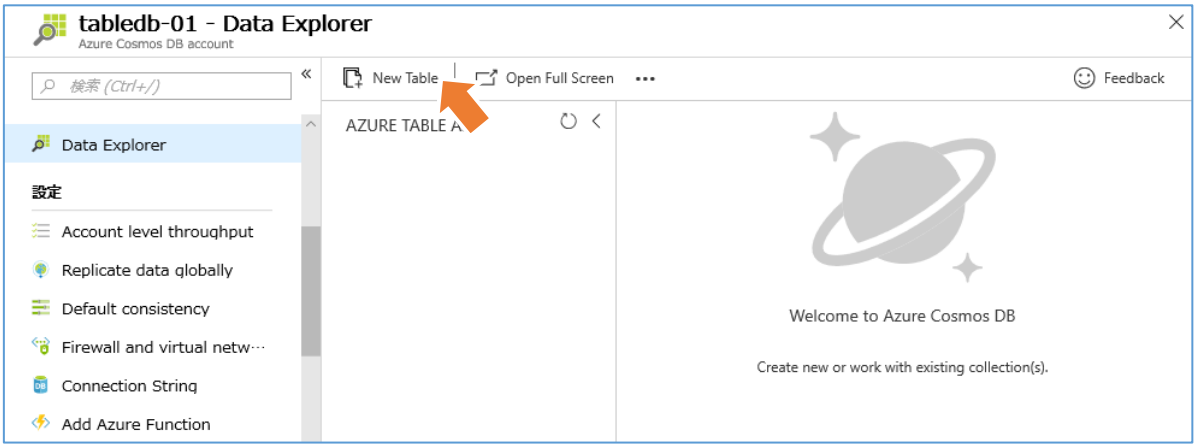
4. 「デプロイが完了しました」が表示されたら、[リソースに移動] をクリックして、作成したリソースを開きます。

5. 作成した Azure Cosmos DB データベース アカウントの [Overview] ブレードで、[Data Explorer] をクリックします。



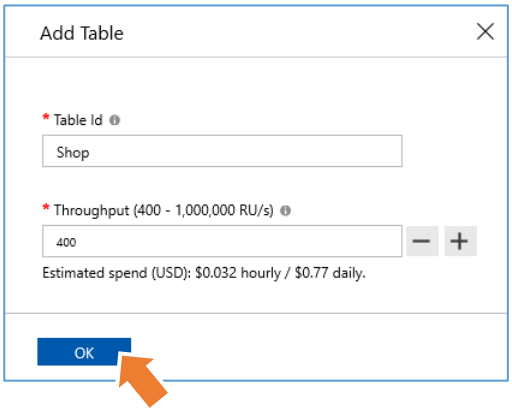
2.2.2 Azure ポータルからのデータベースとテーブルの作成

1. [Data Explorer] ブレードが表示されたら、新しいテーブルを作成するために、[New Table] をクリックします。



6. [Add Table] が表示されるので、下表のパラメーター値を設定して [OK] ボタンをクリックします。

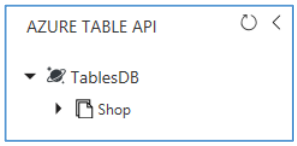
設定項目	設定値
Table Id	「Shop」と入力する
Throughput	400



ワン ポイント

スループット (Throughput) は、1 秒あたりに処理する要求ユニットの数になります。Azure Cosmos DB では、アプリケーションが利用する要求ユニットの量を秒単位で予約することができます。Azure Cosmos DB での各操作は、CPU、メモリ、IOPS が消費されます。各操作により、要求されるリソースの使用量を要求ユニットとして表現しています。要求ユニット使用量に影響を及ぼす要因とアプリケーションのスループット要件を理解しておくと、アプリケーション実行のコスト効率を向上できます。

なお、データベースの設定は明示的に行っていませんが、最初のテーブルを作成すると、「TablesDB」という名前のデータベースが自動的に作成されることも理解しておきましょう。

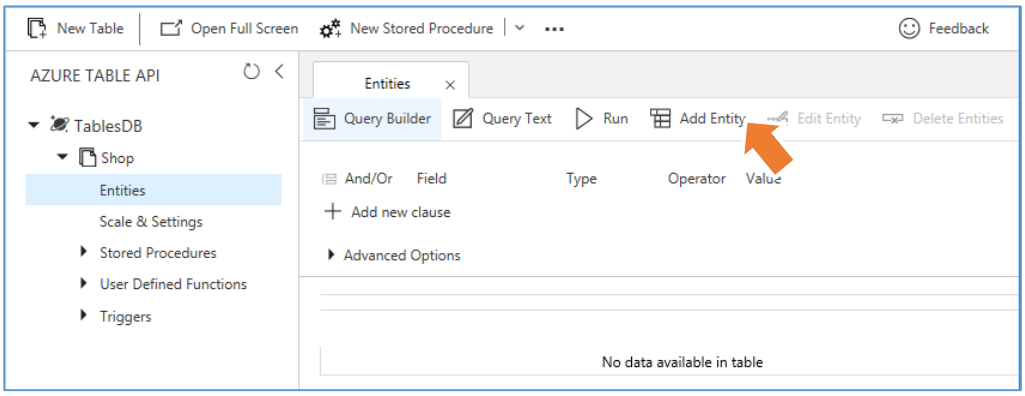


データベース アカウントには任意の数のテーブルを作成することができます。

2.2.3 データ エクスプローラーによるエンティティの追加

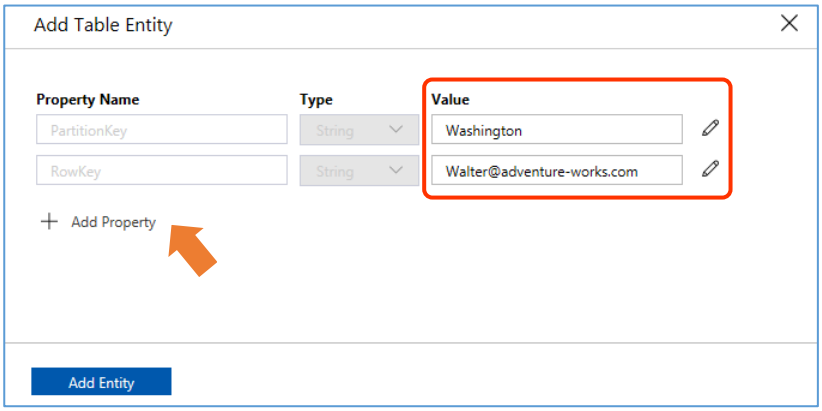
エンティティは、ユーザーが自由に定義できる キー/バリュー型のデータです。テーブルには任意の数のエンティティを保存できます。ここでは、Azure ポータルの [Data Explorer] ブレードを使用して、作成したテーブルにエンティティを追加する方法を確認します。

1. [Data Explorer] ブレードに表示されるツリーで Shop テーブル ノードを展開表示して、[Entites] ノードをクリックした後、[Add Entity] をクリックします。



2. [Add Table Entity] が表示されるので、下表に従い入力して、[Add Property] ボタンをクリックします。

Property Name	Type	Value
PartitionKey	String	Washington
RowKey	String	Walter@adventure-works.com

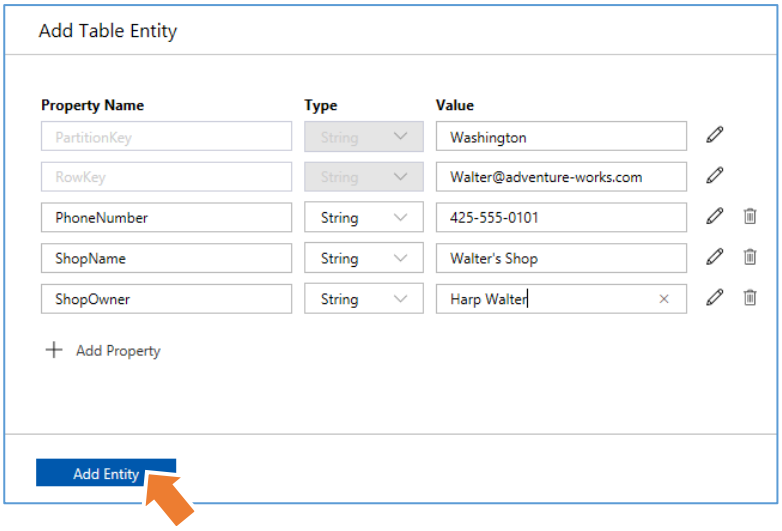


ワン ポイント
各エンティティを一意に識別するために PartitionKey と RowKey の 2 つのシステム プロパティが使用されます。自習書では PartitionKey には州の情報を使用し、PartitionKey には E メールアドレスを使用します。

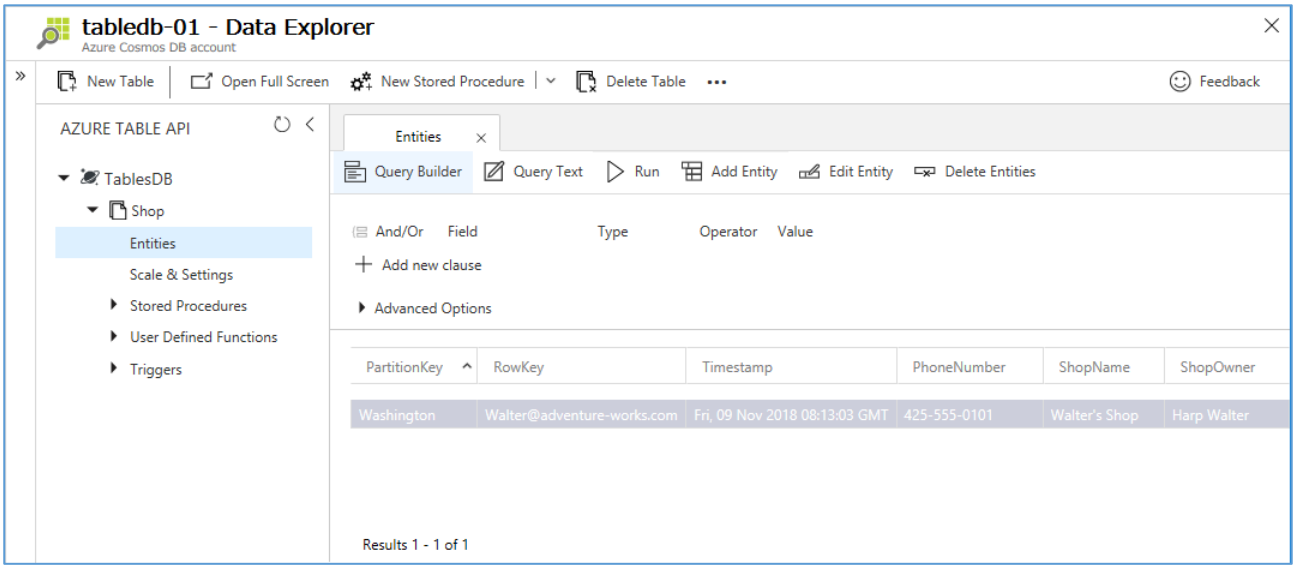
2. Azure ポータルからのデータベースの作成と操作

3. [Add Table Entity] で、下表に従い 3 つのプロパティ名と、その値を追加して、[Add Entity] ボタンをクリックします。

Property Name	Type	Value
PhoneNumber	String	425-555-0101
ShopName	String	Walter's Shop
ShopOwner	String	Harp Walter



4. Shop テーブルに 1 件の店舗情報が格納されたことを確認します。



ワン ポイント
テーブルにエンティティが格納されると、「Timestamp」という名前のシステム プロパティの値が自動生成されます。Timestamp の値は、エンティティが最後に変更された時刻です。

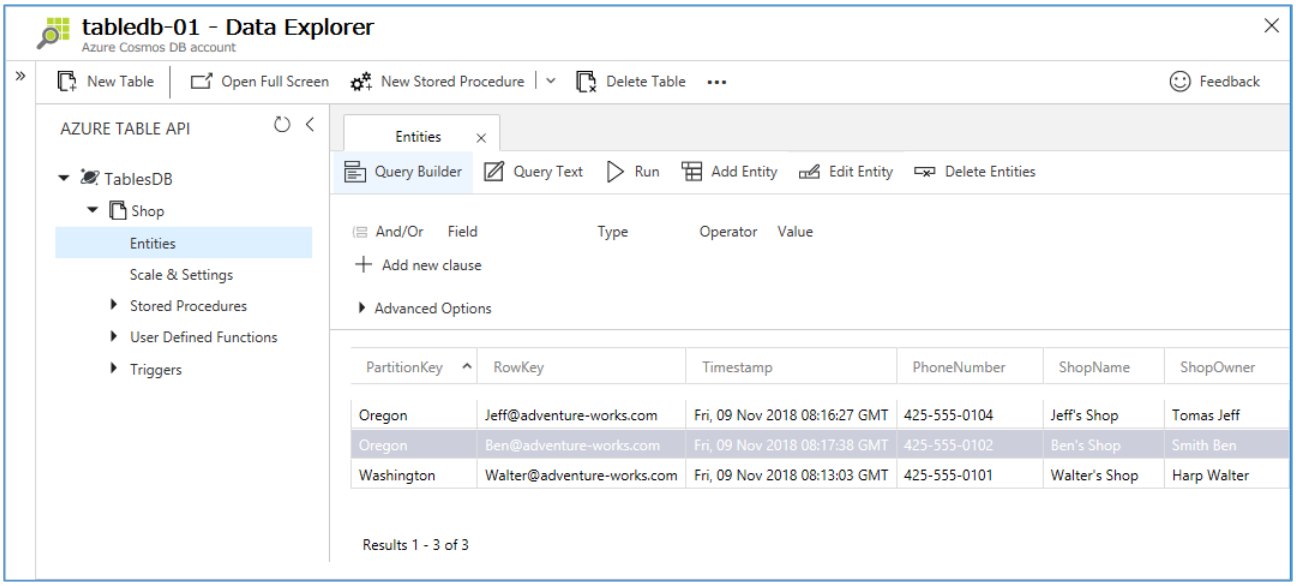
5. 手順 1 から 4 を繰り返し、2 件目のデータとして下表のエンティティを追加します。

Property Name	Type	Value
PartitionKey	String	Oregon
RowKey	String	Jeff@adventure-works.com
PhoneNumber	String	425-555-0104
ShopName	String	Jeff's Shop
ShopOwner	String	Tomas Jeff

6. 手順 1 から 4 を繰り返し、3 件目のデータとして下表のエンティティを追加します。

Property Name	Type	Value
PartitionKey	String	Oregon
RowKey	String	Ben@adventure-works.com
PhoneNumber	String	425-555-0102
ShopName	String	Ben's Shop
ShopOwner	String	Smith Ben

7. エンティティの格納後、[Data Explorer] ブレードは、次の表示になります。

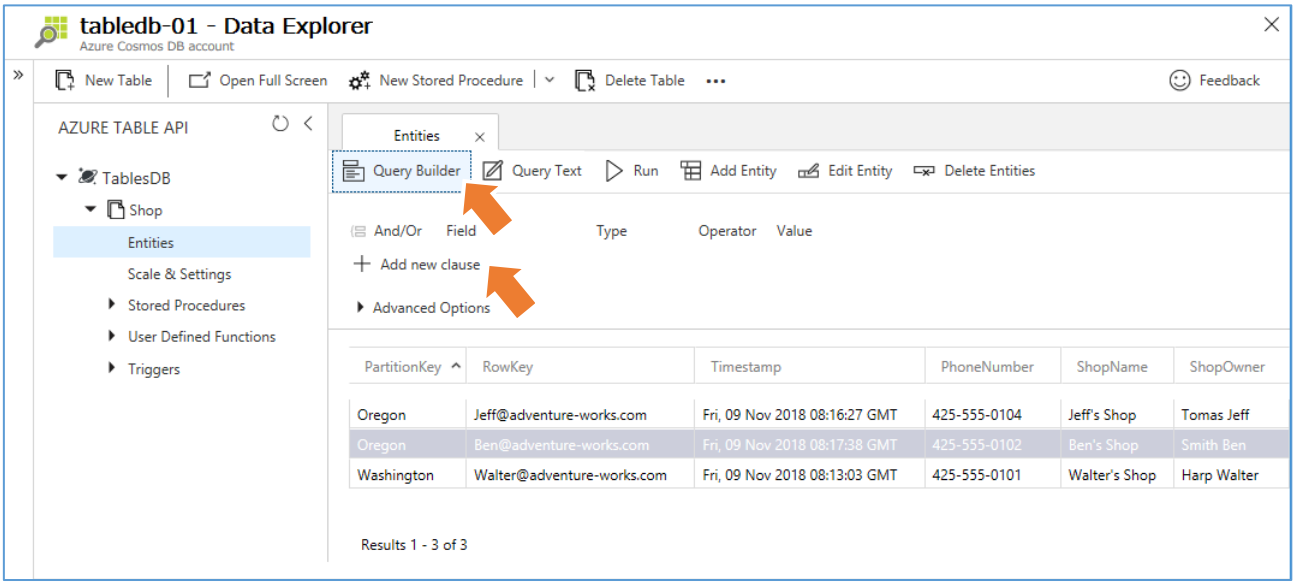


ワン ポイント
PartitionKey が同じエンティティは、同じ論理パーティションに格納されます。

2.2.4 データ エクスプローラー クエリ ビルダーを使用したエンティティの検索、置き換え、および削除

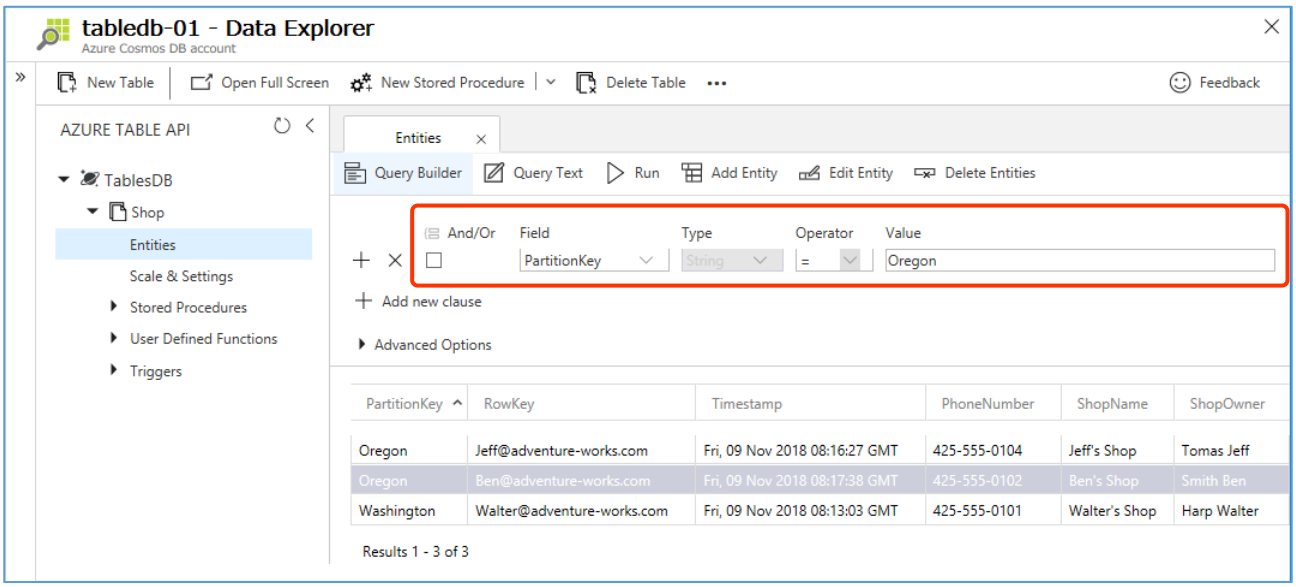
ここでは、Azure ポータルの [Data Explorer] ブレードのクエリ ビルダーを使用して、テーブルに追加したエンティティを検索する方法を確認します。クエリ ビルダーでは、[Add new clause] を使用して、SELECT 構文の WHERE 句に相当する記述が可能です。

1. [Data Explorer] ブレードで [Query Builder] をクリックした後、[Add new clause] をクリックします。



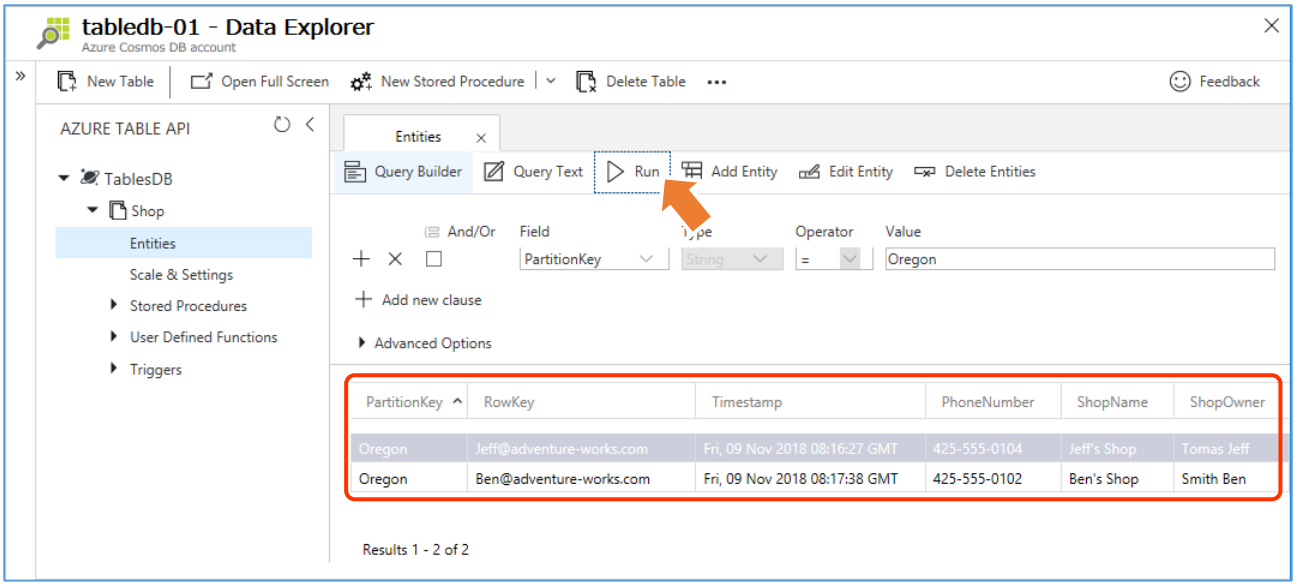
2. 同じパーティションに含まれるすべてのエンティティを取得するために下表に従いフィルター条件を記述します。

And/Or	Field	Type	Operator	Value
	PartitionKey	String	=	Oregon



2. Azure ポータルからのデータベースの作成と操作

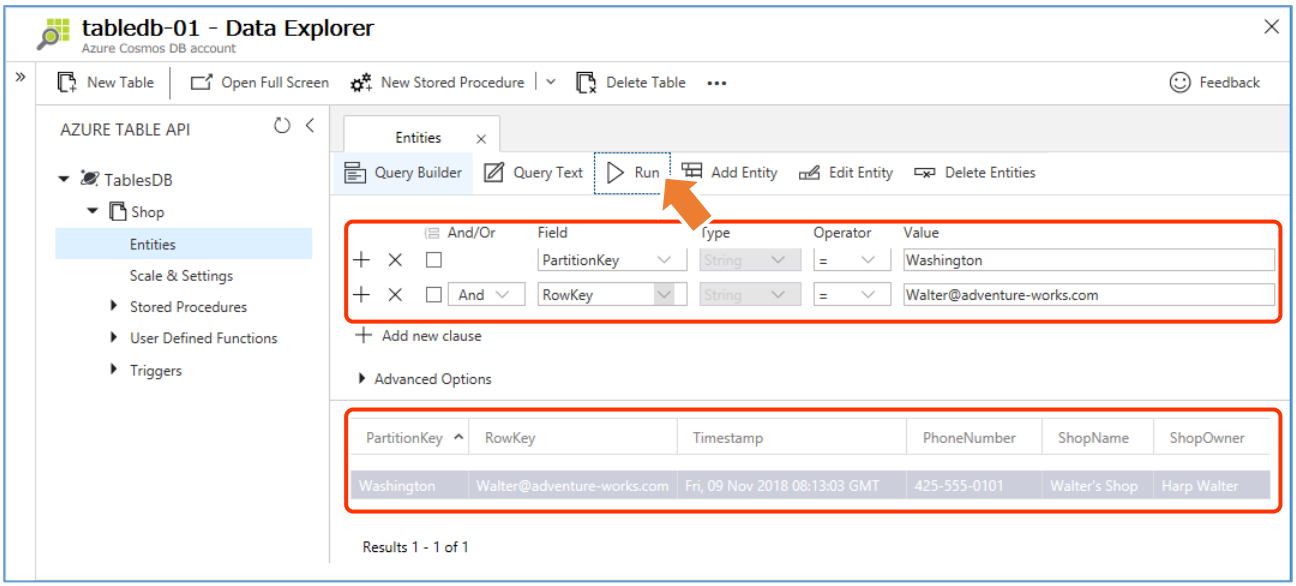
3. [Data Explorer] ブレードで [Run] をクリックして、クエリを実行します。



ワン ポイント
PartitionKey が「Oregon」のエンティティがすべて返されます。なお、検索条件として使用する値は、大文字と小文字が区別されることに注意してください。

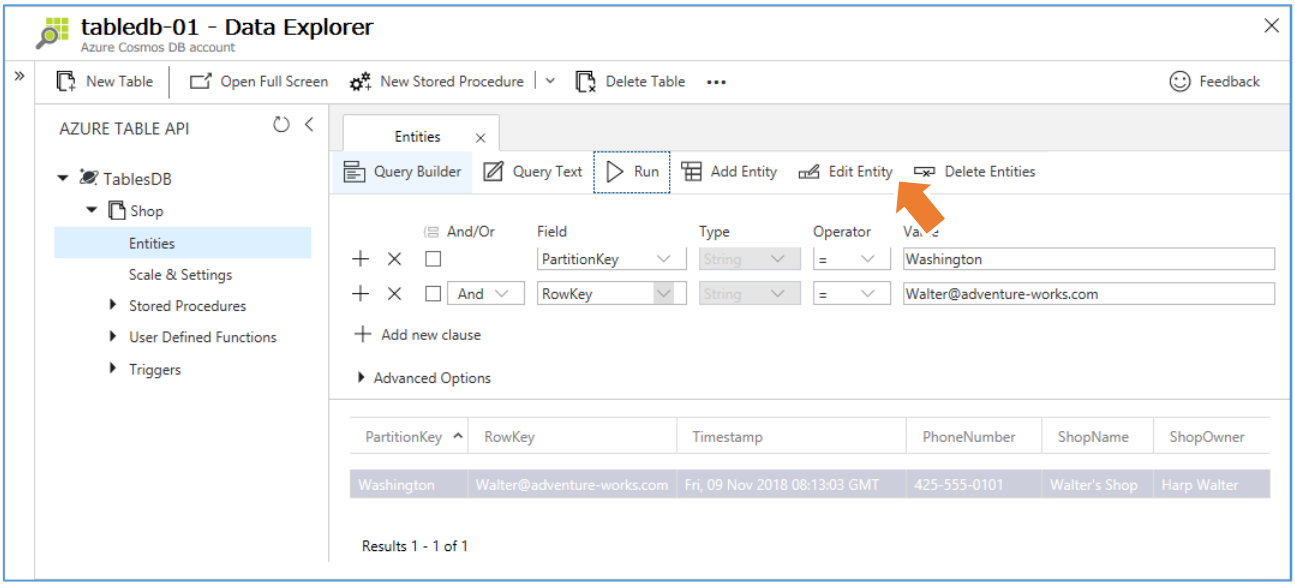
4. PartitionKey と RowKey を指定し、単一のエンティティを取得するために下表に従いフィルター条件を記述して、[Run] をクリックし、クエリを実行します。

And/Or	Field	Type	Operator	Value
	PartitionKey	String	=	Washington
And	RowKey	String	=	Walter@adventure-works.com

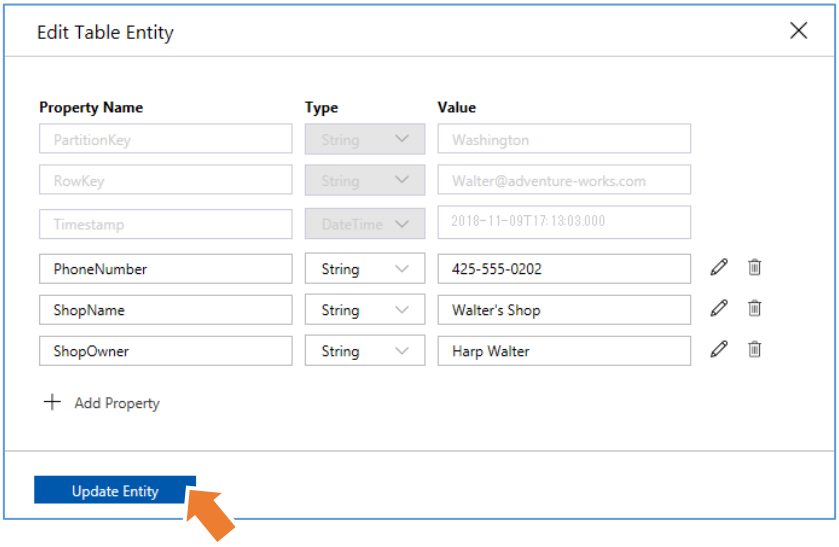


ワン ポイント
PartitionKey と RowKey を指定した検索では、1 件のデータが返されます。

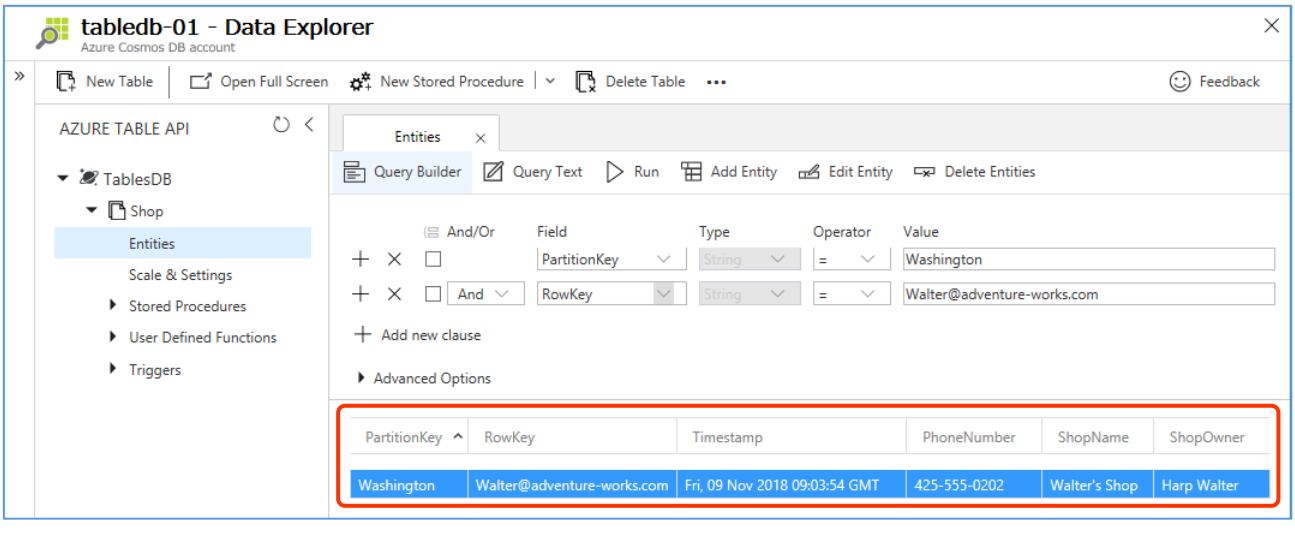
5. [Data Explorer] ブレードで [Edit Entity] をクリックします。



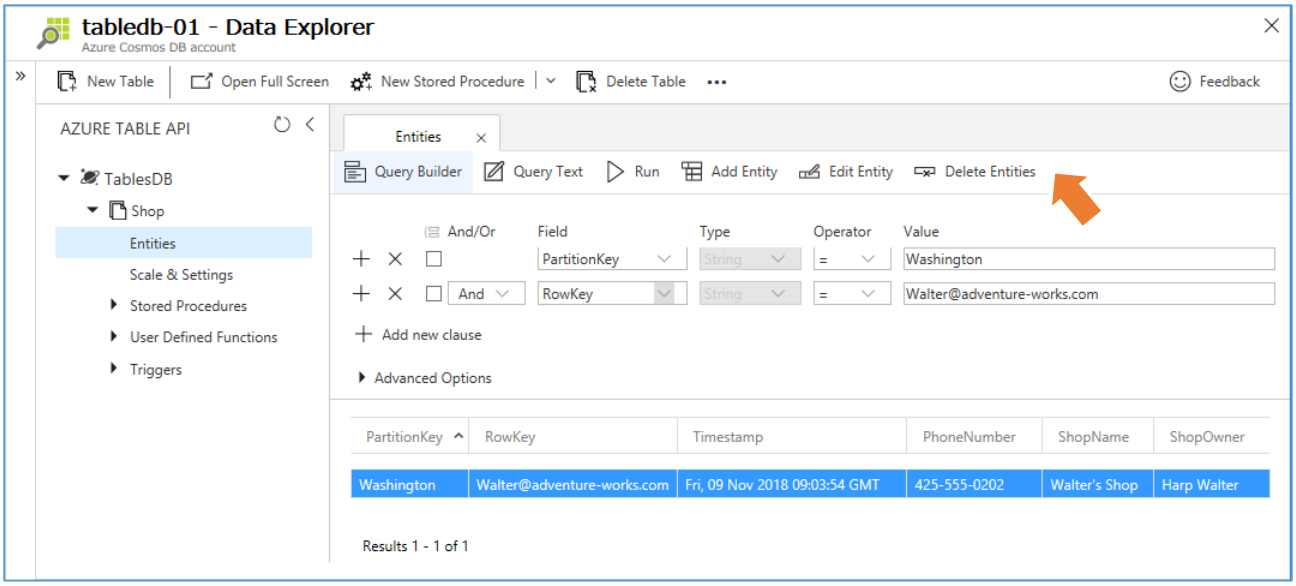
6. [Edit Table Entity] が表示されるので、[PhoneNumber] プロパティを「425-555-0202」に変更して [Update Entity] ボタンをクリックします。



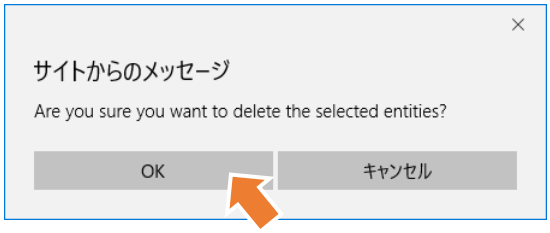
7. 電話番号が修正されたことを確認します。



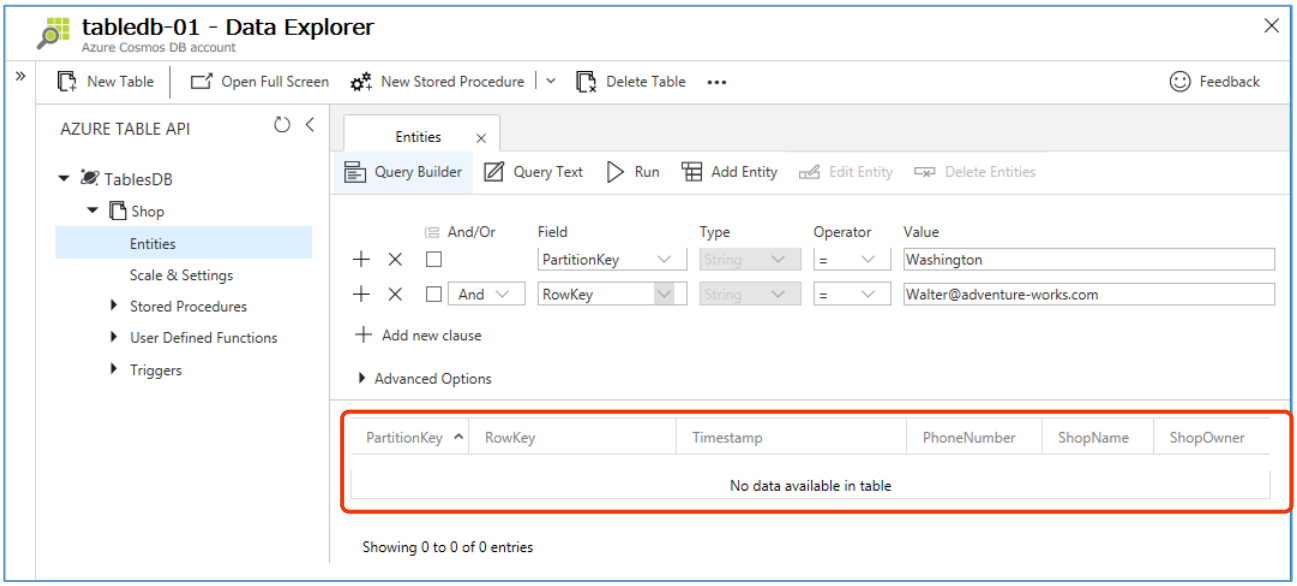
8. [Data Explorer] ブレードで [Delete Entities] をクリックします。



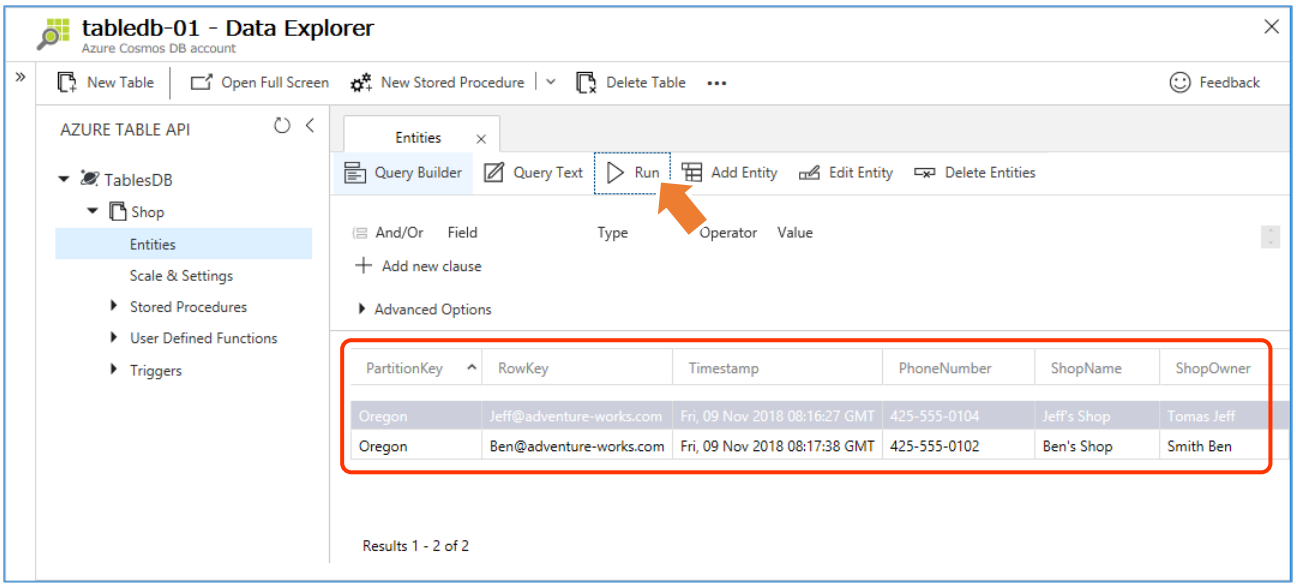
9. 次のメッセージ ボックスが表示されたら、[OK] ボタンをクリックします。



10. 対象のエンティティが削除されたことを確認します。



11. [Data Explorer] ブレードで [X] ボタンをクリックして、クエリ ビルダーに設定した検索条件を削除した後、再度、[Run] をクリックして、テーブルに格納したすべてのエンティティを表示します。

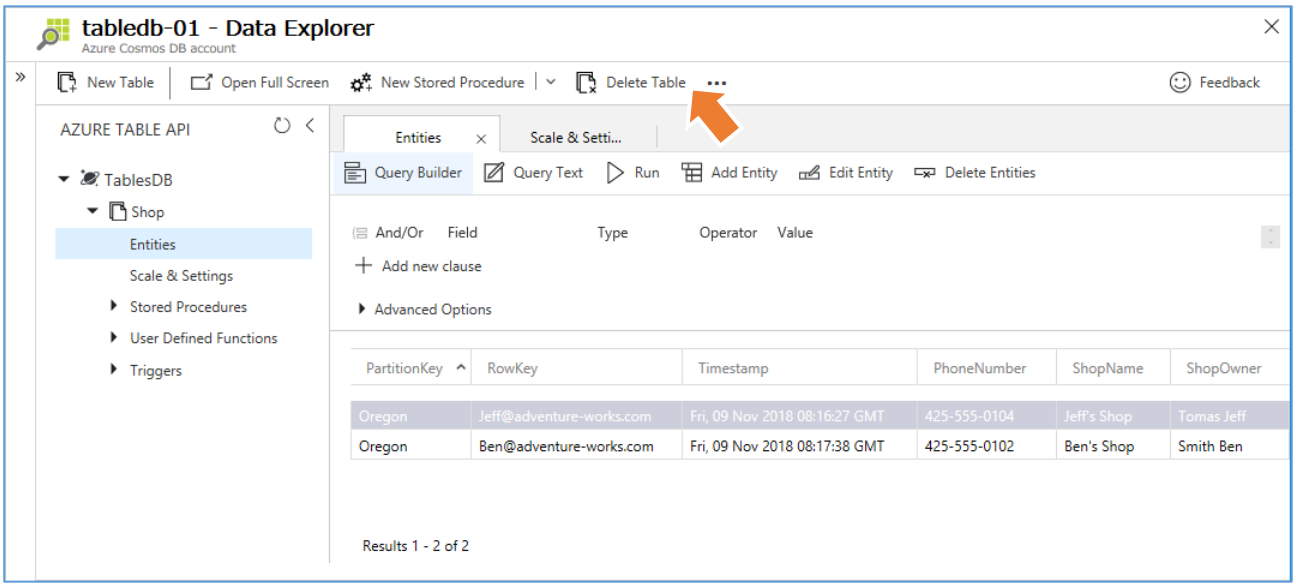


ワン ポイント
現在は、2 件のデータのみが表示されます。

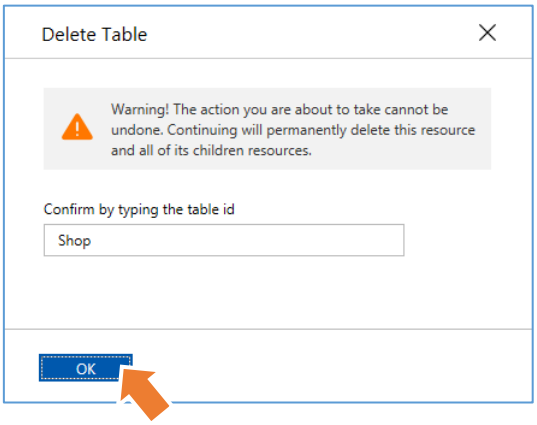
2.2.5 テーブルの削除

作成したテーブルを削除します。3 章では、2 章で作成したテーブルとエンティティをアプリケーション コードから作成します。同じ名前の Shop テーブルを作成しますので、この章で作成したテーブルは削除しておきます。

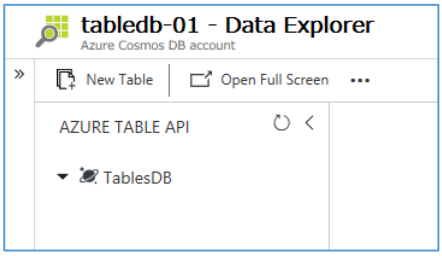
1. 作成した shop テーブルを削除するために、[Data Explorer] ブレードで [Delete Table] をクリックします。



2. [Delete Table] が表示されるので、[Confirm by typing the table id] に「**Shop**」と入力して [OK] をクリックします。



3. データベースからテーブルが削除されたことを確認します。



ワン ポイント
コレクションにあたるテーブルは、課金の対象になります。データベースからテーブルを削除することで課金を停止することができます。

3. Table API SDK を使用する .NET コンソール アプリケーションの作成

2018 年 11 月時点では、Azure Cosmos DB の Table API を使用するアプリの開発に使用できる SDK は下表の種類が提供されています。

SDK	ダウンロード	説明
Azure Cosmos DB Table .NET API	NuGet Gallery	この SDK を使用して Azure Table Storage と Azure Cosmos DB Table API の両方に接続することができます。 Microsoft .NET Framework 4.5.1 での使用がサポートされています。
Azure Cosmos DB Table .NET Standard API	NuGet Gallery	Azure Cosmos DB 上の Table データ モデルにアクセスするためのクロス プラットフォーム対応 .NET ライブラリで、Microsoft .NET Standard 2.0 での使用がサポートされています。 この最初のリリース (0.9.1-preview) では、テーブルおよびエンティティの CRUD とクエリの機能全体、および Cosmos DB Table SDK For .NET Framework に類似する API がサポートされています。
Azure Cosmos DB Table API for Node.js	Node Package Manager (NPM)	この SDK を使用して Azure Table Storage と Azure Cosmos DB Table API の両方に接続することができます。
Azure Cosmos DB Table API for Java	Apache Maven	この SDK を使用して Azure Table Storage と Azure Cosmos DB Table API の両方に接続することができます。
Azure Cosmos DB Table API SDK for Python	Python Package Index (PyPI)	この SDK を使用して Azure Table Storage と Azure Cosmos DB Table API の両方に接続することができます。 Python 2.7、3.3、3.4、3.5、3.6 のバージョンをサポートしています。

本自習書では、Azure Cosmos DB Table .NET API を使用する C# のコードを記述して、Azure Cosmos DB アカウントに接続する方法を学習します。なお、現在サポートされている Microsoft .NET Framework フレームワークのバージョンは、.NET Framework 4.5.1 および .NET Standard 2.0 で、.NET Core は、現時点では、未サポートであることに注意してください。

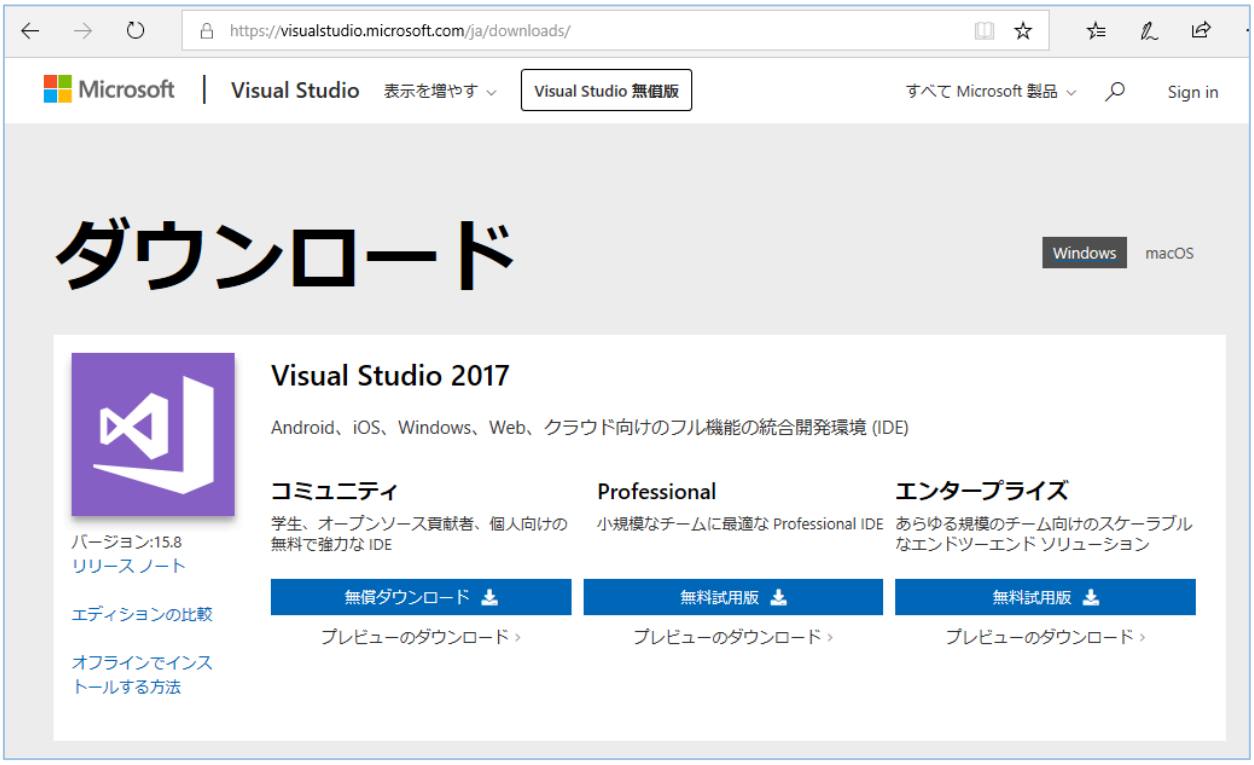
サポートされる SDK の最新情報は、「Azure Cosmos DB Table API と Azure Table Storage を使用した開発」を参照してください。

自習書では、以降の手順で C# を使用して、Azure Cosmos DB に接続する簡単なコンソール アプリを作成し、エンティティの作成や検索を行うコードを確認します。テーブルに複数のエンティティを追加し、アプリケーション コードでエンティティの検索やプロパティ値を変更する方法なども確認します。

3.1 Visual Studo 2017 のインストール手順

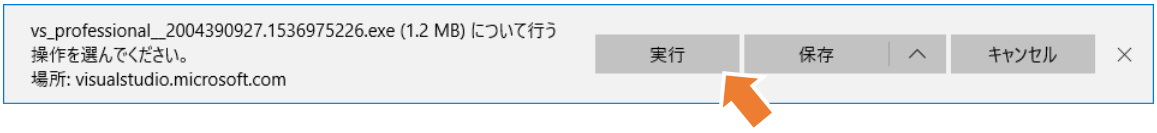
自習書の実行環境に Visual Studio 2017 をインストールしていない場合、以降の手順実行で、Visual Studio 2017 をインストールできます。Visual Studio 2017 は、https://visualstudio.microsoft.com/ja/downloads から入手できます。

1. Web ブラウザーで https://visualstudio.microsoft.com/ja/downloads にアクセスして、表示されるページで、使用する Visual Studo のダウンロード ボタンをクリックします。

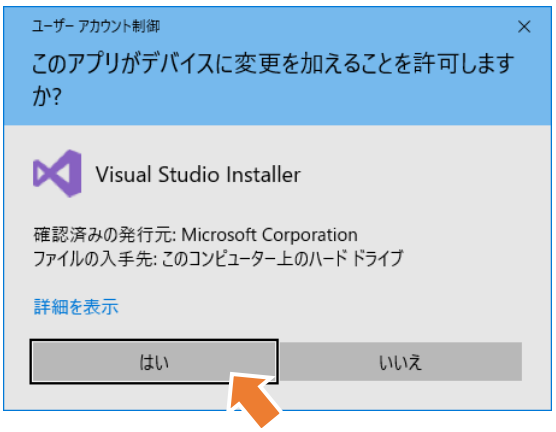


ワン ポイント
Mac OS 版の Visual Studio を使用する場合、[ダウンロード] の右側の [macOS] をクリックしてください。

2. Web ブラウザーの下部にメッセージが表示されたら、[実行] をクリックします。

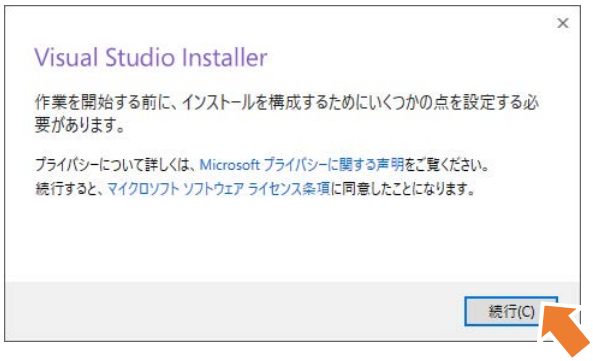


3. [ユーザー アカウントの制御] が表示されたら、[はい] をクリックします。



3. Table API SDK を使用する .NET コンソール アプリケーションの作成

4. 次のメッセージ ボックスで、リンクをクリックして「Microsoft のプライバシーに関する声明」と「マイクロソフト ソフトウェア ライセンス条項」を確認し、よろしければ [続行] をクリックして インストールを開始します。

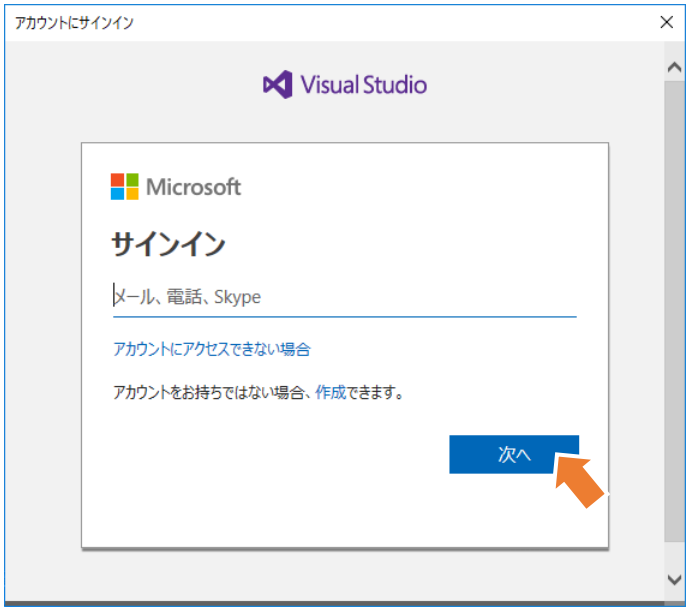


5. ワークロードを選択するページが開くので、「.NET デスクトップ開発」と「Azure の開発」を選択して [インストール] ボタンをクリックします。

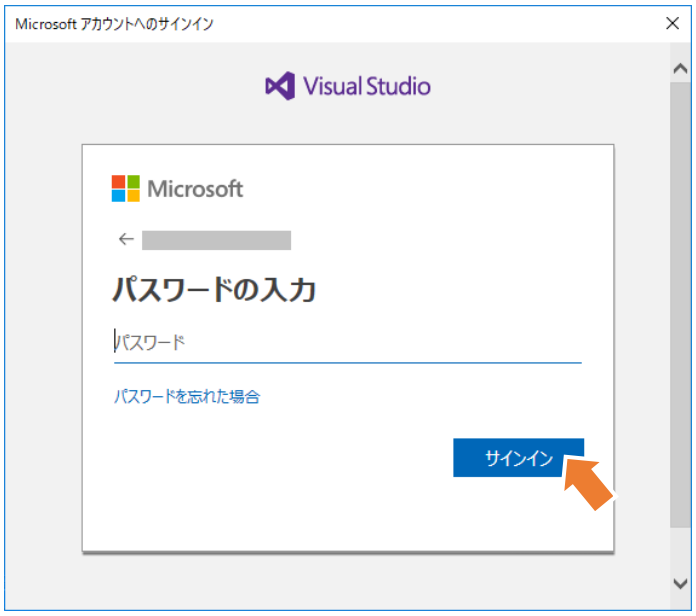
> **ワン ポイント**
>
> C# で Azure Cosmos DB に接続する .NET コンソール アプリを作成するために必要最小限のワークロードを選択しています。

6. インストール完了後に、[ようこそ] が表示されるので、[サインイン] ボタンをクリックします。

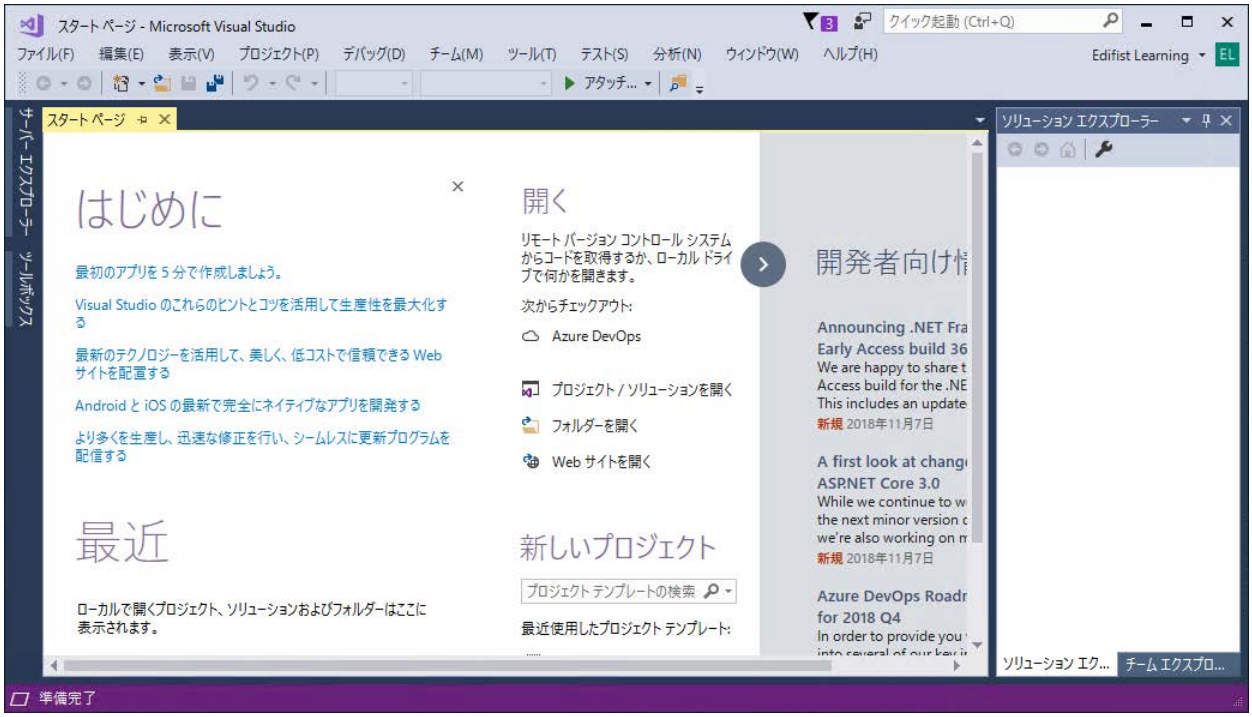
7. [サインイン] ページでは、前の章の手順を実行したときに使用した、Azure サブスクリプションが関連付けられたアカウントを入力して、[次へ] ボタンをクリックします。



8. Azure アカウントのパスワードを入力して [サインイン] ボタンをクリックします。



9. Visual Studo が起動することを確認します。



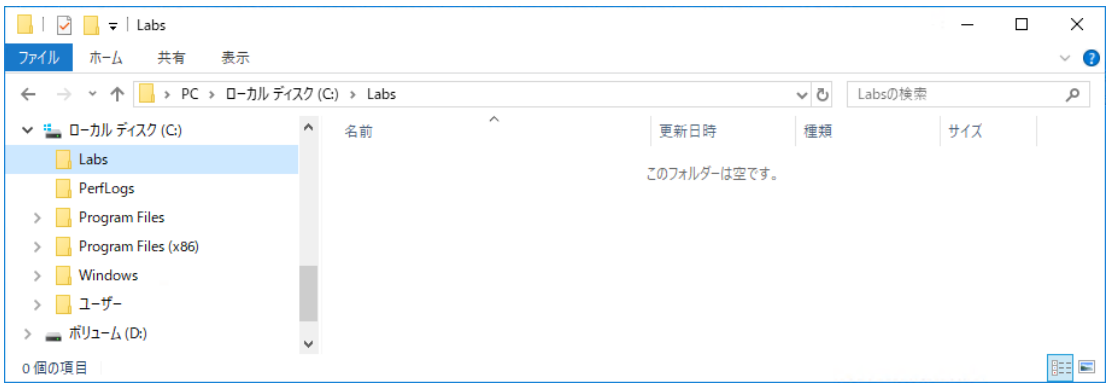
3.2 コンソール アプリのプロジェクト作成と NuGet パッケージのダウンロード

ここでは最初に、コンソール アプリのプロジェクトを作成します。Azure Cosmos DB アカウントに接続し、テーブルに対する操作を行うのに必要なクラスが含まれる NuGet パッケージ（下表）をダウンロードし、作成したコンソール アプリのプロジェクトから参照できるように構成します。

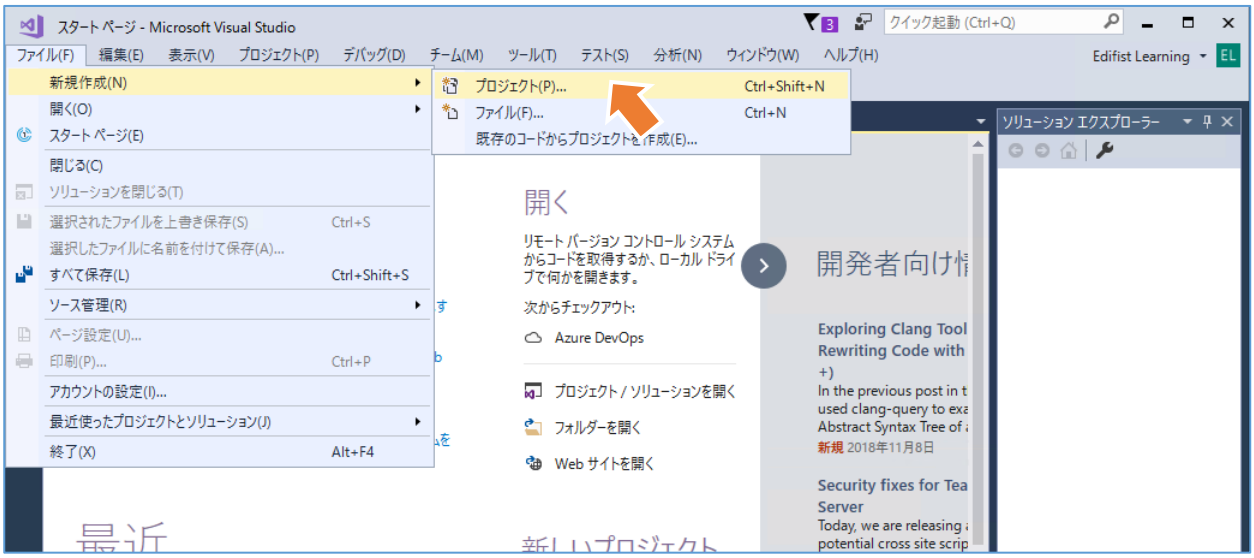
パッケージ名	説明
Microsoft.Azure.Storage.Common (9.0.0.1 以下)	Azure Cosmos DB アカウントに接続するための CloudStorageAccount クラスのインスタンスを作成できます。
Microsoft.Azure.CosmosDB.Table	CloudTableClient クラスのインスタンスを作成できます。
Microsoft.WindowsAzure.ConfigurationManager	構成設定を読み込むための統一された API を提供します。

なお、Microsoft.Azure.Storage.Common は、9.0.0.1 以下のバージョン（9.0.0.1 は、現在 プレビューです）を使用する必要があることに注意してください。

1. Windows エクスプローラーで、自習用のディスク ドライブに、「Labs」という名前のフォルダーを作成します。

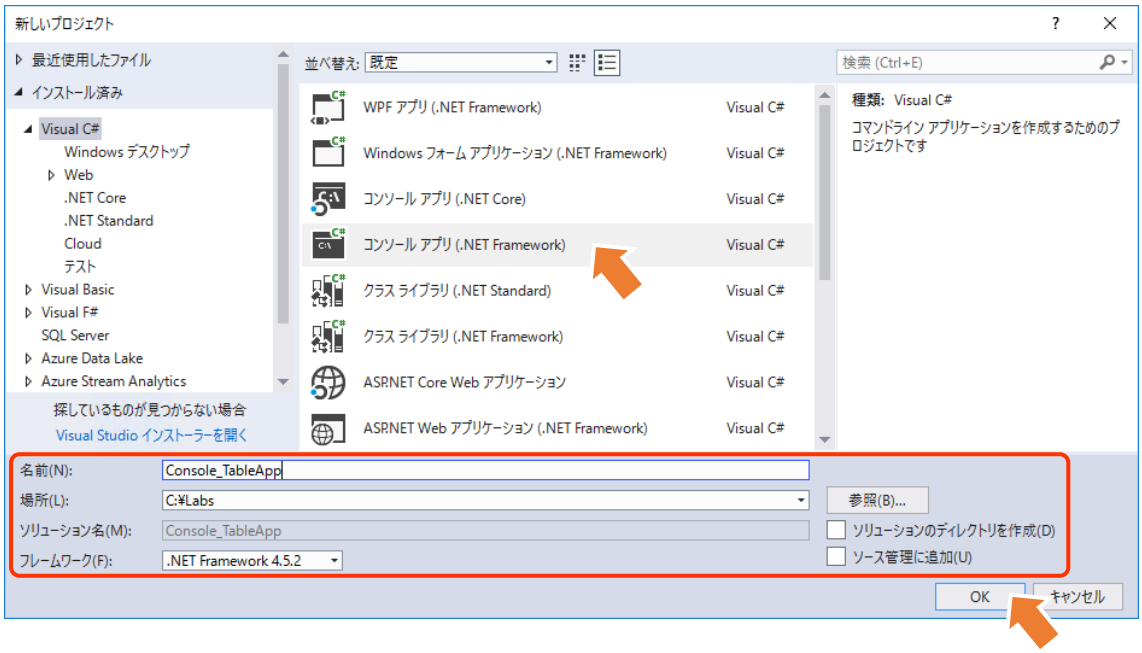


2. Visual Studio の [ファイル] メニューから [新規作成] ⇒ [プロジェクト] を選択します。

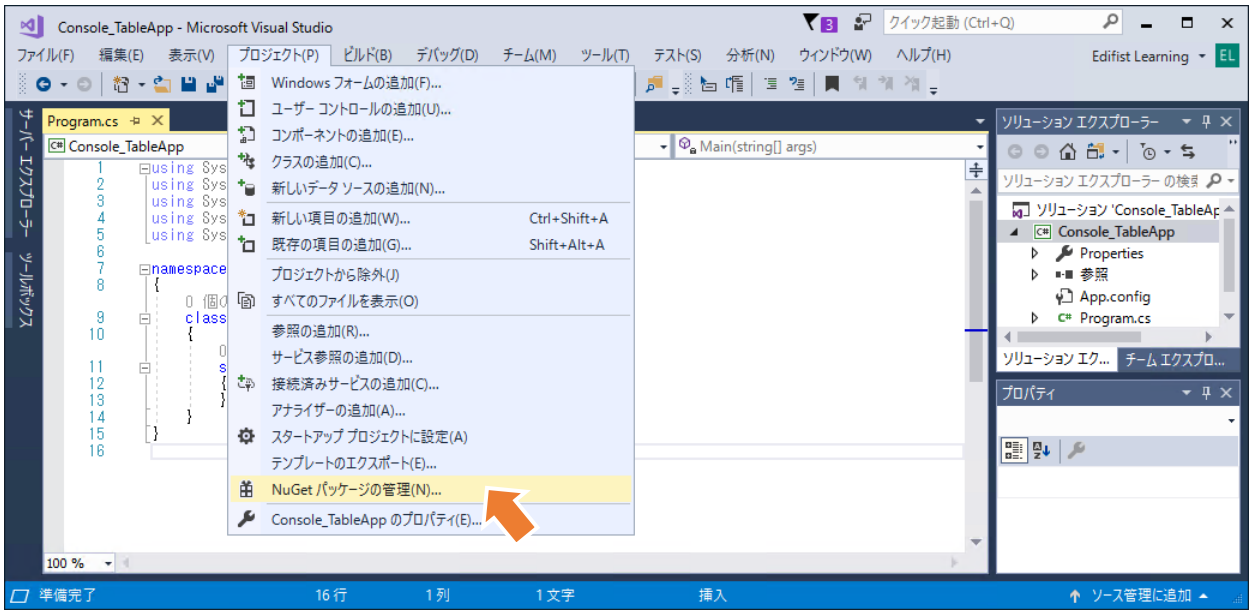


3. [新しいプロジェクト] が開くので、[Visual C#] ノードの下の [コンソール アプリ (.NET Framework)] テンプレートを選択して、下表に従いプロジェクトを構成した後、[OK] ボタンをクリックします。

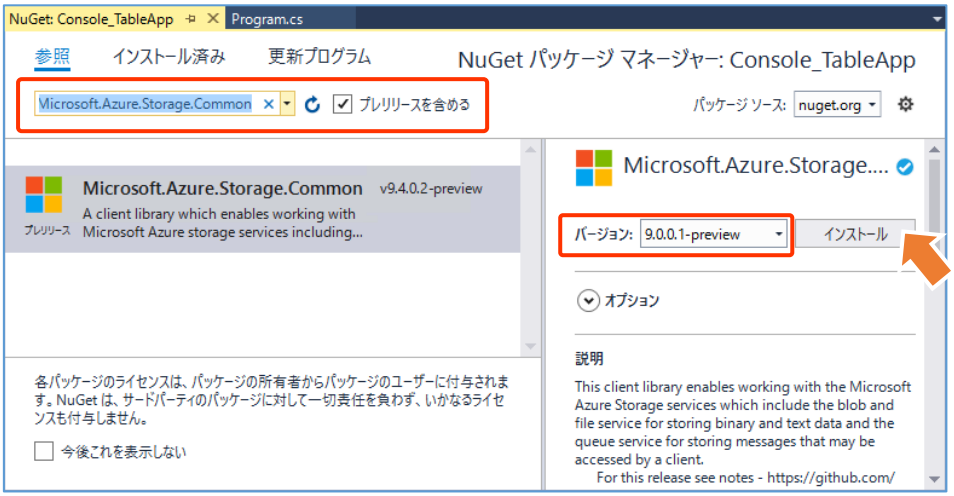
項目	設定値
名前	Console_TableApp
場所	C:¥Labs
フレームワーク	.NET Framework 4.5.2
ソリューションのディレクトリを作成	選択を外す
ソース管理に追加	選択を外す



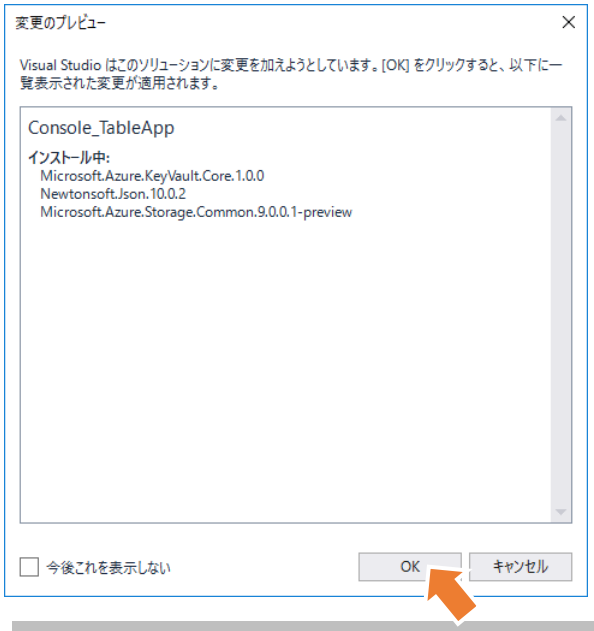
4. 必要な NuGet パッケージを取得するために、ソリューション エクスプローラーでプロジェクトを右クリックし、[NuGet パッケージの管理] をクリックします。



5. NuGet パッケージ マネージャーの [参照] タブを選択し、[プレリリースを含める] チェックボックスを選択した後、[検索] ボックスに 「**Microsoft.Azure.Storage.Common**」 と入力し、表示される Microsoft.Azure.Storage.Common NuGet パッケージを選択し、[バージョン] で「9.0.0.1-preview」を選択して、[インストール] ボタンをクリックします。

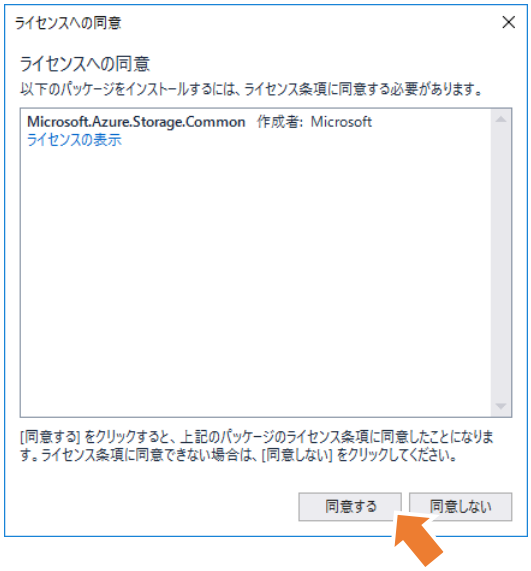


6. [変更のプレビュー] が表示されたら、[OK] ボタンをクリックします。



ワンポイント
選択した NuGet に含まれるパッケージが、ソリューションにインストールされます。

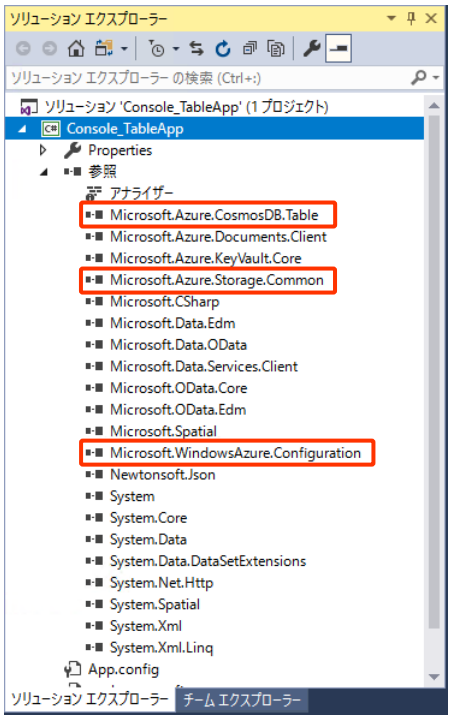
7. この後、[ライセンスへの同意] が表示されたら、[同意する] ボタンをクリックします。



8. 手順 5 から 7 を繰り返して、下表の NuGet パッケージも追加します。

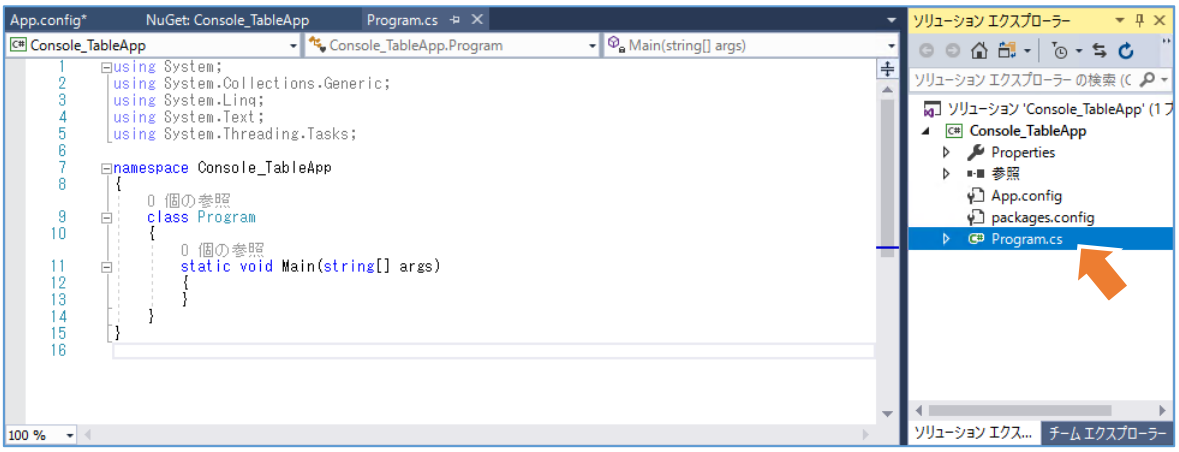
パッケージ名	バージョン
Microsoft.Azure.CosmosDB.Table	2.0.0
WindowsAzure.ConfigurationManager	3.2.3

9. ソリューション エクスプローラーのツリーで [参照] ノードを展開表示して、「Microsoft.Azure.Storage.Common」、「Microsoft.Azure.CosmosDB.Table」、「WindowsAzure.ConfigurationManager」 が追加されていることを確認します。



3. Table API SDK を使用する .NET コンソール アプリケーションの作成

10. ソリューション エクスプローラーのツリーで Program.cs ファイルをクリックして、エディターで開きます。



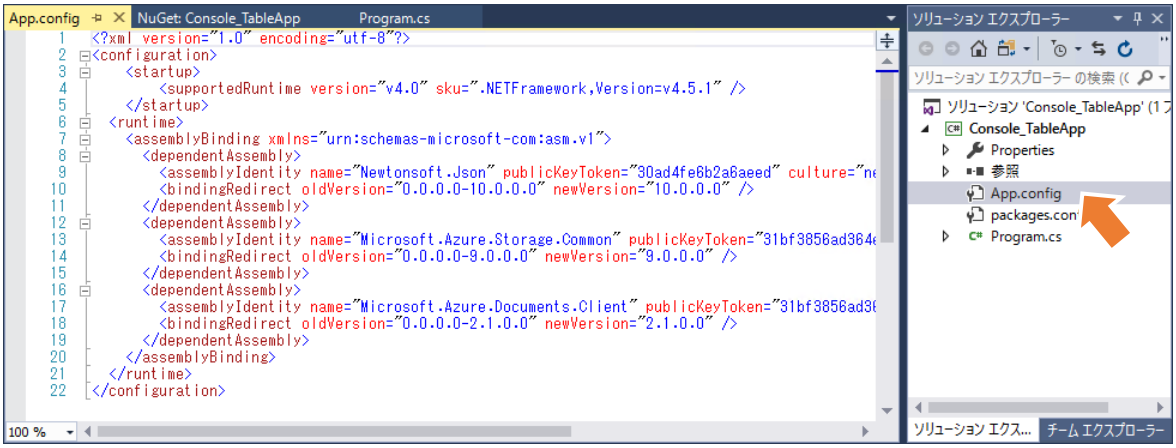
11. NuGget パッケージのクラス参照のために、Program.cs ファイルの先頭の using ディレクティブのブロックに次のコードを追加しておきます。

```
using Microsoft.Azure;
using Microsoft.Azure.Storage;
using Microsoft.Azure.CosmosDB.Table;
```

3.3 Azure Cosmos DB アカウントへの接続とテーブルの作成

Azure Cosmos DB アカウントへの接続のために、App.Config に接続文字列を記述して、CloudStorageAccount クラスをインスタンス化します。CloudStorageAccount インスタンスの CreateCloudTableClient() メソッドで、Table Service クライアント (CloudTableClient インスタンス) を作成することができ、このインスタンスのメソッドで、接続先の Azure Cosmos DB アカウントに対して、テーブル作成などの操作を行うことができます。

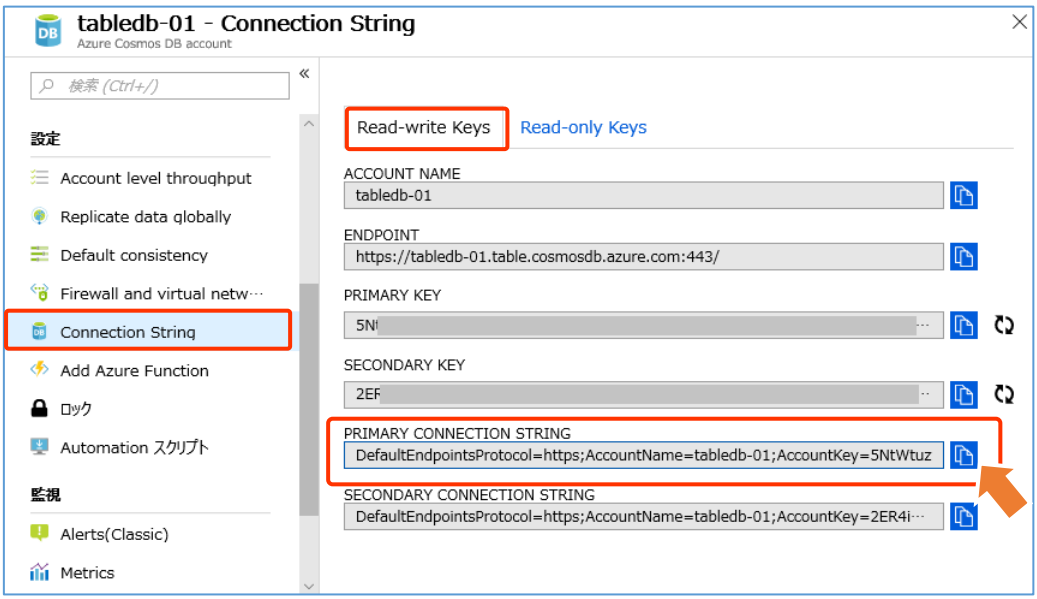
1. ソリューション エクスプローラーのツリーで App.config ファイルをクリックして、エディターで開きます。



2. <configuration><startup> 要素の下に次の <appSettings> 要素を追加します。

```
<appSettings>
    <add key="StorageConnectionString" value="<接続文字列>" />
</appSettings>
```

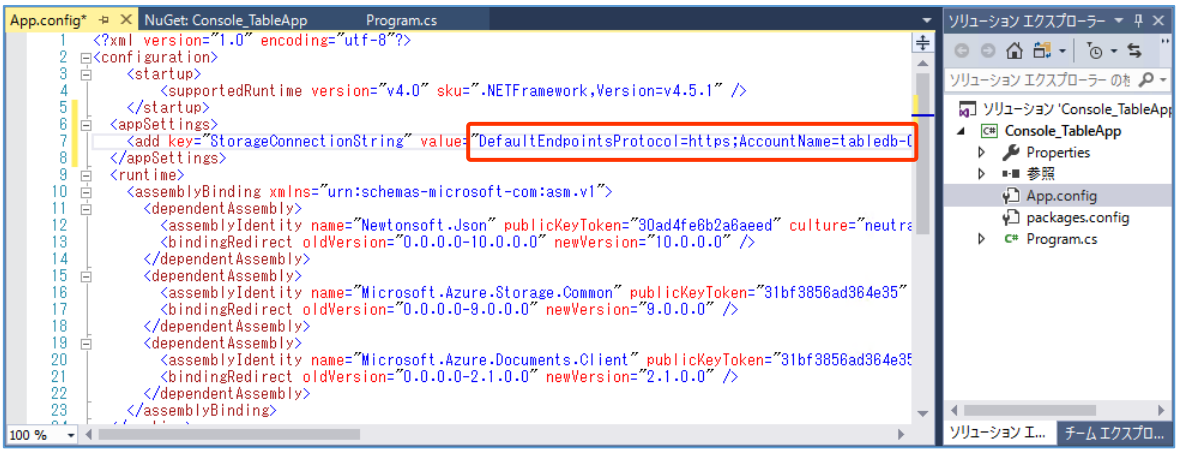
3. Azure ポータルに戻り、アプリケーション コードから接続する Azure Cosmos DB アカウントの [Conection String] ブレードを開いて [Read-write Keys] タブの、[PRIMARY CONNECTION STRING] を取得します。



3. Table API SDK を使用する .NET コンソール アプリケーションの作成

ワンポイント
[Connection String] ブレードには、プライマリとセカンダリの 2 種類のキーに対応した接続文字列が提供されています。これにより、管理者の変更などで、キーの再作成が必要になった場合、一時的にセカンダリに切り替え、プライマリ キーの再作成後、再び接続情報をプライマリに変更することで、アプリケーションからのデータ アクセスを中断しないで、キーを更新できるようにしてきます。 また、接続情報は、[Read-write Keys] タブと [Read-only Keys] タブで提供されています。[Read-only Keys] タブの接続文字列では、アカウントの読み取り操作のみが許可されます。データの追加、削除、置き換え操作が必要ないアプリケーションでは、読み取り専用の接続文字列を使用します。

4. Azure ポータルから コピーした接続文字列を App.config ファイルの appSettings 要素 StorageConnectionString の value 値として **<接続文字列>** に貼り付けます。



5. App.Config に記述した接続文字列を使用して、CloudStorageAccount クラスをインスタンス化するため、Program.cs ファイルの Main() メソッドに次のコードを追加します。

```
// CloudStorageAccount クラスをインスタンス化する
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
        CloudConfigurationManager.GetSetting("StorageConnectionString"));
```

6. 接続ポリシーを設定するため、次のコードを Main() メソッドに追加します。

```
// 接続ポリシーの設定
TableConnectionPolicy tableConnectionPolicy = new TableConnectionPolicy();
tableConnectionPolicy.EnableEndpointDiscovery = true;
tableConnectionPolicy.UseDirectMode = true;
tableConnectionPolicy.UseTcpProtocol = true;
```

ワン ポイント
tableConnectionPolicy は、Azure CosmosDB アカウントに接続する Table Service クライアントの接続ポリシーを表します。tableConnectionPolicy.EnableEndpointDiscovery プロパティを True に構成すると、レプリケートされたデータベース アカウントのエンドポイント検出を有効化することができます。また、[Multi-region Writes](複数リージョンの書き込み) を有効化している場合、EnableEndpointDiscovery プロパティを有効化して、TableConnectionPolicy.PreferredLocations プロパティを構成することで、ローカル リージョンに優先的にアクセスするようにアプリケーションを構成できます。

7. CloudStorageAccount インスタンスの CreateCloudTableClient() メソッドを使用して、Table Service クライアント (CloudTableClient インスタンス) を作成するため、次のコードを Main() メソッドに追加します。

```
// Table Service クライアントの作成
CloudTableClient tableClient = storageAccount.CreateCloudTableClient(tableConnectionPolicy);
```

8. Azure Cosmos DB アカウントに「Shop」という名前のテーブルが存在していない場合に、スループットを 400 に設定した Shop テーブルを作成するため、次のコードを Main() メソッドに追加します。

```
// Shop テーブルへの参照を取得
CloudTable table = tableClient.GetTableReference("Shop");

// テーブルが存在しない場合、400 RU/秒のスループットを指定してテーブルを作成
table.CreateIfNotExists(throughput: 400);
```

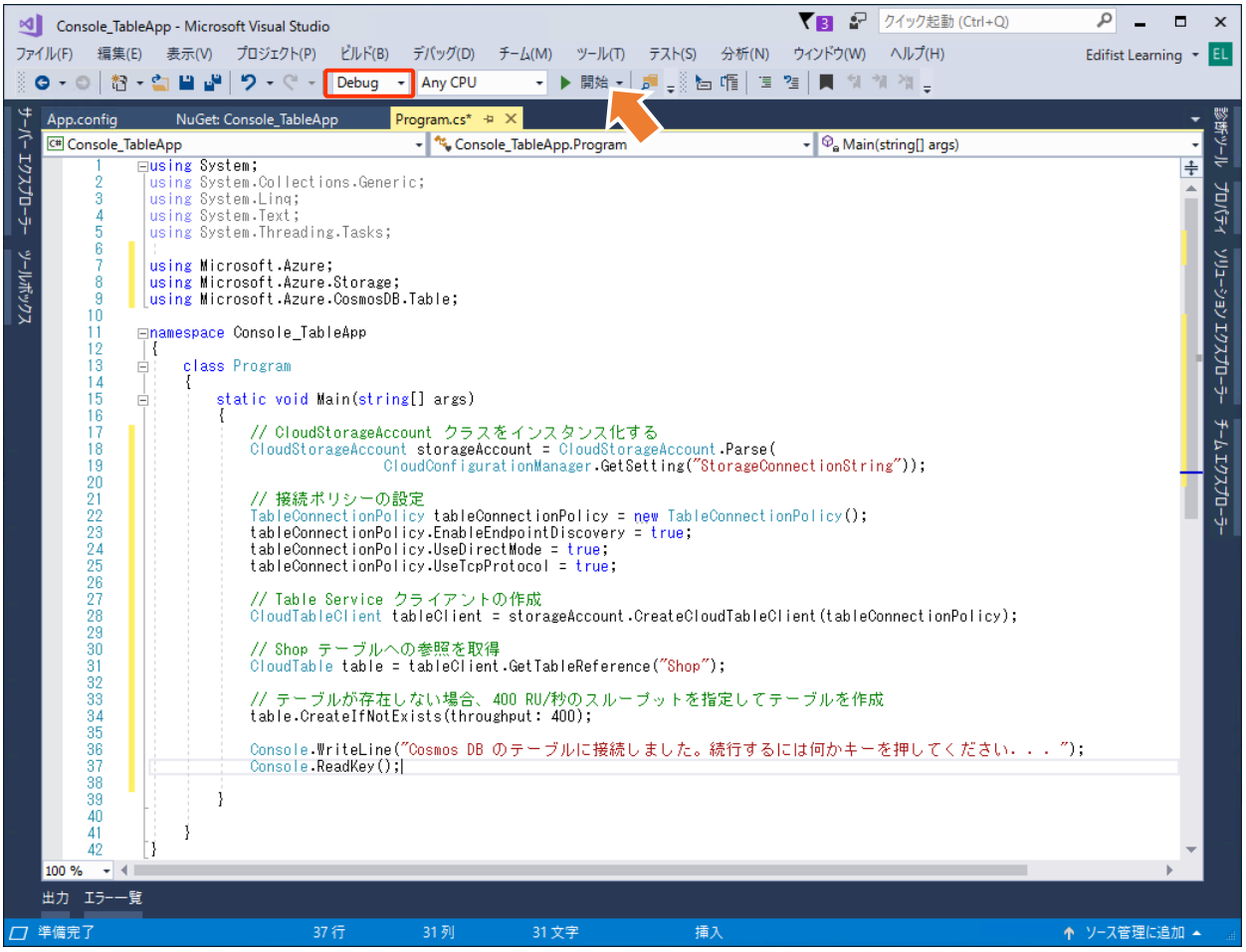
ワン ポイント

CreateIfNotExists メソッドは、対象テーブルが存在しない場合のみ、テーブルを作成できます。なお、作成したテーブルを削除するには、同様に GetTableReference メソッドで対象テーブルの参照を取得して、DeleteIfExists メソッドを使用することができます。

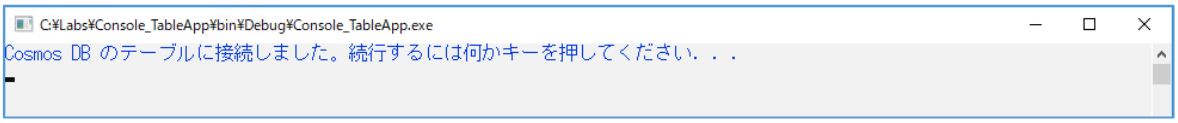
9. コンソールにメッセージ出力するために、次のコードを Main() メソッドに追加します。

```
Console.WriteLine("Cosmos DB のテーブルに接続しました。続行するには何かキーを押してください. . . ");
Console.ReadKey();
```

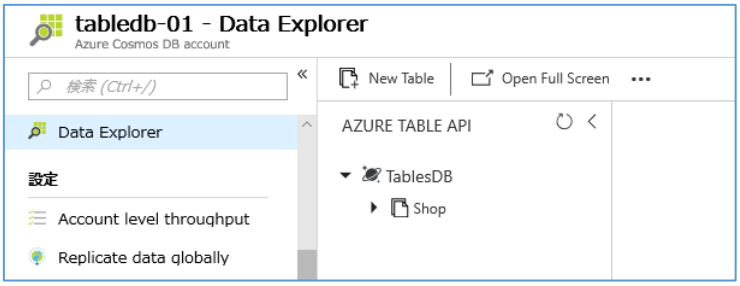
10. ここまで記述したコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックしてデバッグ実行します。



11. コマンド プロンプトが起動して、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



12. Azure ポータルで [Data Explorer] ブレードを参照し、TablesDB データベースに Shop テーブルが作成されていることを確認します。



3.4 アプリケーション コードによるエンティティの追加

テーブルが作成できましたので、以降のステップではテーブルに対する基本的な CRUD 操作を確認します。最初に、アプリケーション コードを記述して、作成したテーブルに 2 種類の方法 (1 件ずつ追加する方法と複数件をまとめて追加する方法) でエンティティを追加します。

1. Shop エンティティを定義するために、program クラスに次のコードを追加します。

```
// Shop エンティティを定義
public class ShopEntity : TableEntity
{
    public ShopEntity(string State, string EmailAddress)
    {
        this.PartitionKey = State;
        this.RowKey = EmailAddress;
    }
    public ShopEntity() { }

    public string ShopName { get; set; }
    public string ShopOwner { get; set; }
    public string PhoneNumber { get; set; }
}
```

ワン ポイント

Azure Cosmos DB のテーブルに格納するエンティティは TableEntity クラスから拡張して定義します。各エンティティを一意に識別するために PartitionKey と RowKey の 2 つのシステム プロパティを使用します。PartitionKey に州の情報を使用し、PartitionKey には E メールアドレスを使用しています。

2. ShopEntity クラスを使用し、1 件のエンティティをテーブルに追加する AddOneEntity メソッドを作成するために、program クラスに次のコードを追加します。

```
public void AddOneEntity(CloudTable table)
{
    // Shop エンティティの作成
    ShopEntity shop1 = new ShopEntity("Washington", "Walter@adventure-works.com");
    shop1.ShopName = "Walter's Shop";
    shop1.ShopOwner = "Harp Walter";
    shop1.PhoneNumber = "425-555-0101";

    // Shop エンティティを挿入するための TableOperation オブジェクトの作成
    TableOperation insertOperation = TableOperation.Insert(shop1);

    // 挿入操作の実行
    table.Execute(insertOperation);
}
```

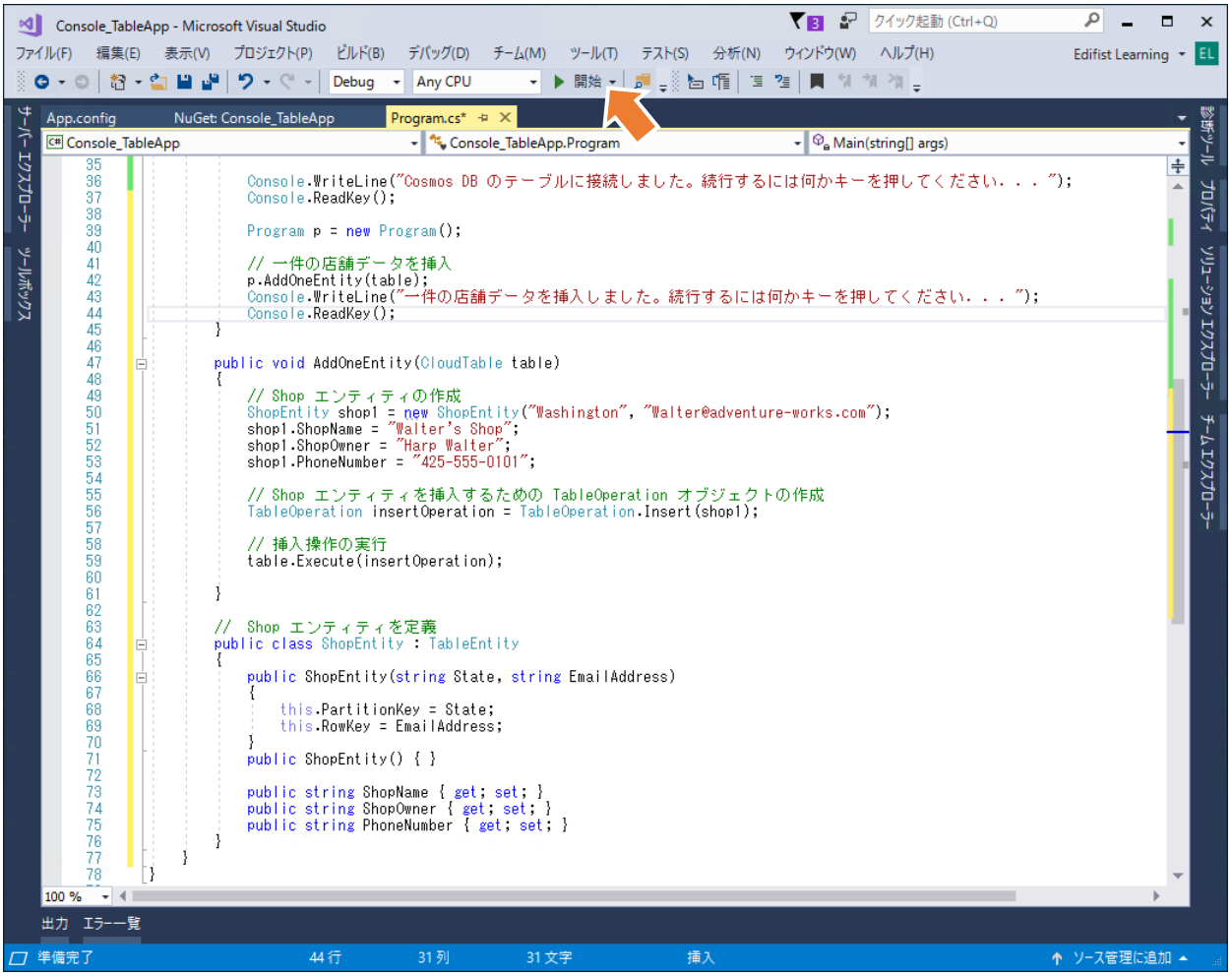
ワン ポイント
TableOperation クラスは、Eauals、Retrive、Insert、Delete、Merge、Replace、InsertOrMerge、InsertOrReplace などテーブルに対する単一操作を行うメソッドを提供しています。

3. program クラスに作成した AddOneEntity メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

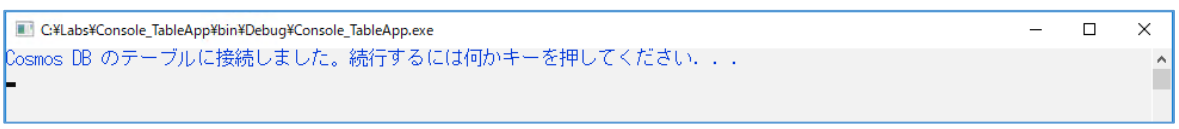
```
Program p = new Program();


// 一件の店舗データを挿入
p.AddOneEntity(table);
Console.WriteLine("一件の店舗データを挿入しました。続行するには何かキーを押してください. . . ");
Console.ReadKey();
```

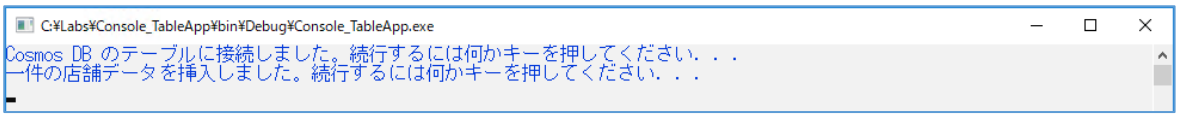
4. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックします。



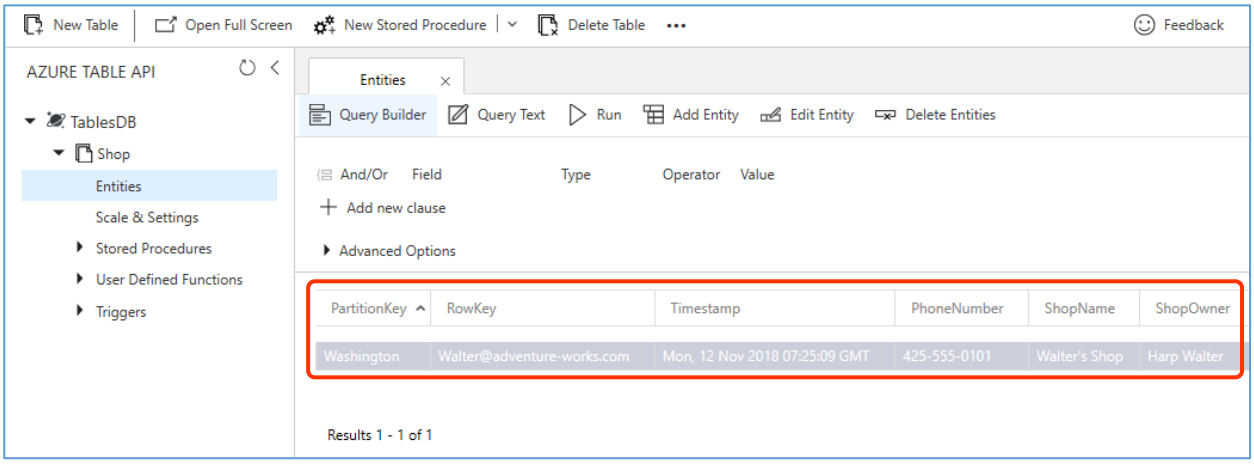
5. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



6. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。

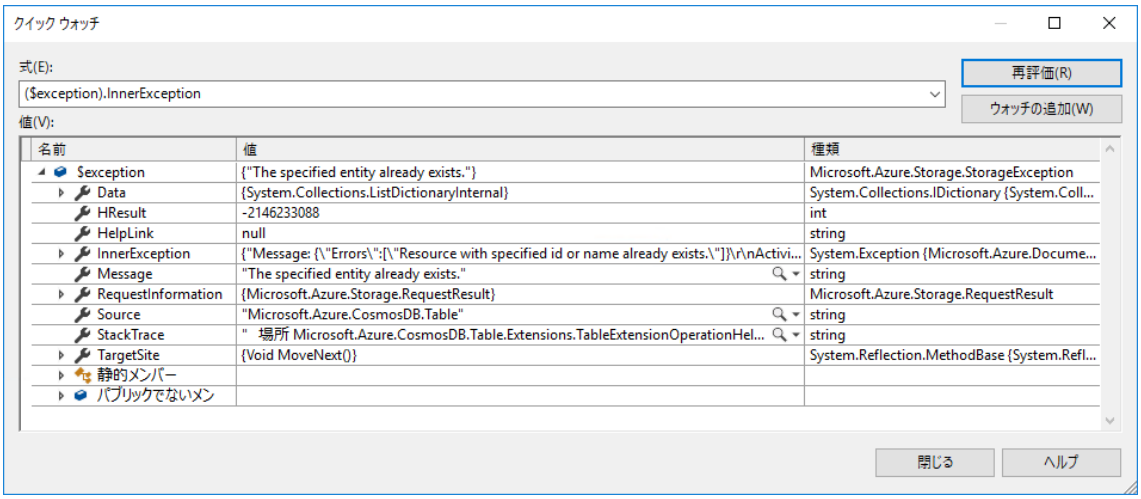


7. Azure ポータルで [Data Explorer] ブレードを参照し、Shop テーブルにエンティティが追加されていることを確認します。



ワンポイント

ここで挿入したエンティティは、2 章の Azure ポータルの [Data Explorer] ブレードを使用して、追加したエンティティと同じ値です。もし、2 章の演習の最後に行うテーブル削除を行っていない場合は、同じキー値を持つエンティティが既に存在しているため、以下の「The specified entity already exists.」エラーが返されます。



これは、作成したコンソール アプリを複数回実行した場合でも同様で、キーが重複したことが原因です。この場合、既存のエンティティを削除して、このプログラムを実行すれば正常終了します。

8. 次に複数件のエンティティをまとめてテーブルに追加する AddMultiEntity メソッドを作成するために、program クラスに次のコードを追加します。

```
public void AddMultiEntity(CloudTable table)
{
    // TableBatchOperation オブジェクトの作成
    TableBatchOperation batchOperation = new TableBatchOperation();

    // 1 件目の Shop エンティティの作成
    ShopEntity shop1 = new ShopEntity("Oregon", "Jeff@adventure-works.com");
    shop1.ShopName = "Jeff's Shop";
    shop1.ShopOwner = "Tomas Jeff";
    shop1.PhoneNumber = "425-555-0104";

    // 2 件目の Shop エンティティの作成
    ShopEntity shop2 = new ShopEntity("Oregon", "Ben@adventure-works.com");
    shop2.ShopName = "Ben's Shop";
    shop2.ShopOwner = "Smith Ben";
    shop2.PhoneNumber = "425-555-0102";

    // 2 件の Shop エンティティを TableBatchOperation オブジェクトに渡す
    batchOperation.Insert(shop1);
    batchOperation.Insert(shop2);

    // 一括挿入操作の実行
    table.ExecuteBatch(batchOperation);
}
```

ワン ポイント

TableBatchOperation クラスは、Entity Group Transaction を呼び出すことによって、テーブルに対するバッチ操作を単一のアトミック操作として実行するメソッドのコレクションを提供しています。例えば複数の挿入操作で、1 件のエラーが発生した場合、バッチ内のすべての操作はキャンセルされます。
リレーショナル データベースのトランザクション処理と異なる点は、単一のテーブルに対してのみ動作し、バッチ処理に含めることができるテーブル操作の最大数は 100 まで、サイズが 4 MB までに制限され、同じパーティションに格納されたエンティティを使用する必要があることです。異なるパーティションに格納されるエンティティのバッチ操作ではエラーになるため、PartitionKey 値が同じエンティティを対象にしてください。

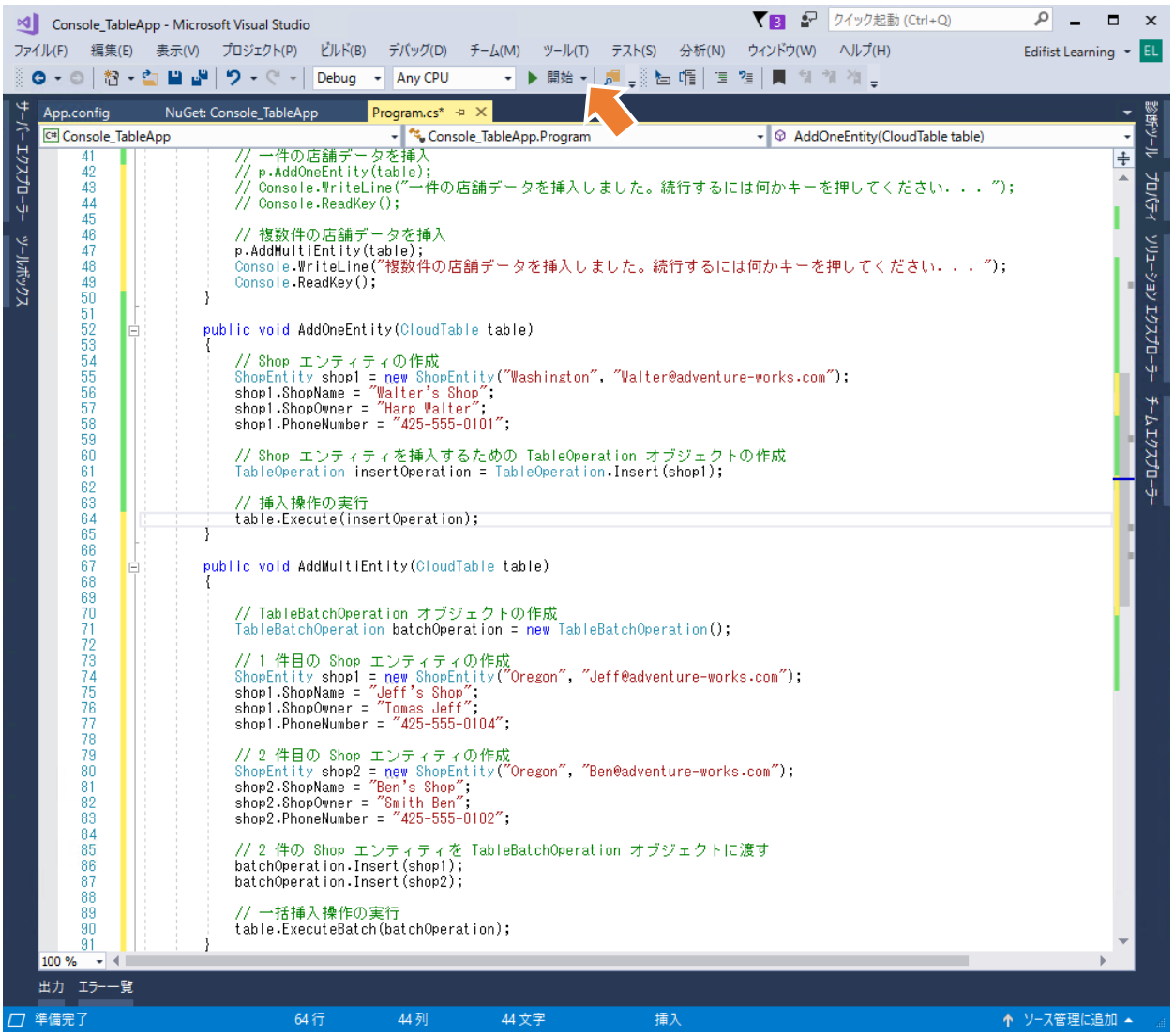
9. 重複したキー値を持つエンティティの挿入を避けるため、Main() メソッド の AddEntity メソッドを呼び出す行を注釈化します。

```
    // 一件の店舗データを挿入
    // p.AddOneEntity(table);
    // Console.WriteLine("一件の店舗データを挿入しました。続行するには何かキーを押してください．．．");
    // Console.ReadKey();
```

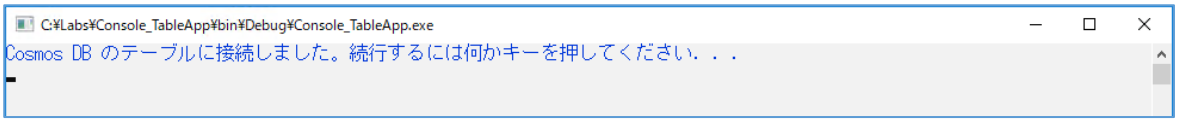
10. program クラスに作成した AddMultiEntity メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

```
// 複数件の店舗データを挿入
p.AddMultiEntity(table);
Console.WriteLine("複数件の店舗データを挿入しました。続行するには何かキーを押してください. . . ");
Console.ReadKey();
```

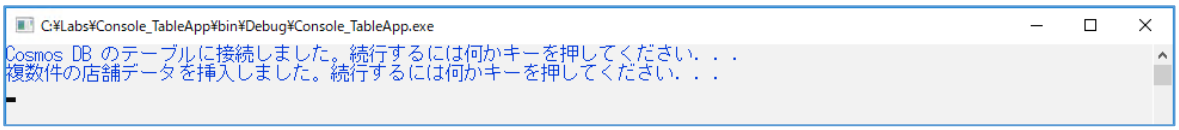
11. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックして、デバッグ実行します。



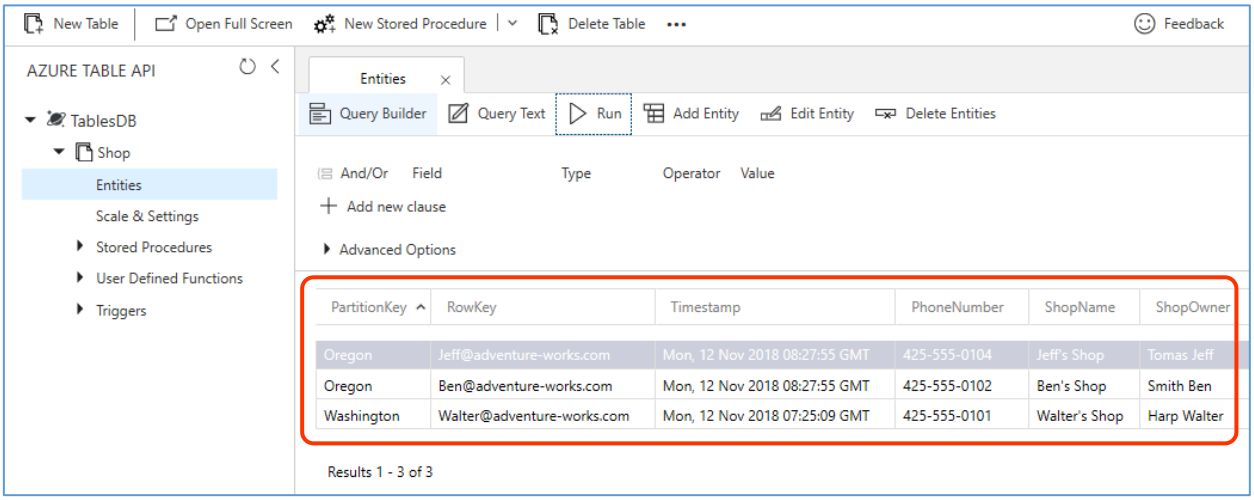
12. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



13. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



14. Azure ポータルで [Data Explorer] ブレードを参照し、Shop テーブルにエンティティが追加されていることを確認します。



3.5 アプリケーション コードによるエンティティの検索

テーブルに 3 件のエンティティが格納されましたので、以降のステップでは、テーブルに格納したエンティティに対する検索、置き換え、および削除操作を試してみます。
キー値を指定し、テーブルから 1 件のエンティティを取得する場合は、TableOperation クラスの Retrive メソッドを使用できます。また、全件のデータを取得したり、特定のプロパティに対して検索条件を指定し、エンティティの集合を取得するクエリの実行には TableQuery クラスを使用できます。

1. キー値を指定して、取得した Shop エンティティから電話番号を表示する GetPhoneNumberFromOneEntity メソッドを作成するために、program クラスに次のコードを追加します。

```
public void GetPhoneNumberFromOneEntity(CloudTable table)
{
    // キー値を指定して Shop エンティティを取得するための TableOperation オブジェクトの作成
    TableOperation retrieveOperation = TableOperation
        .Retrieve<ShopEntity>("Washington", "Walter@adventure-works.com");

    // エンティティを取得する操作の実行
    TableResult retrievedResult = table.Execute(retrieveOperation);

    // 取得したエンティティの電話番号を表示
    if (retrievedResult.Result != null)
    {
        Console.WriteLine(((ShopEntity)retrievedResult.Result).PhoneNumber);
    }
    else
    {
        Console.WriteLine("The phone number could not be retrieved.");
    }
}
```

2. 重複したキー値を持つエンティティの挿入を避けるため、Main() メソッドの AddMultiEntity メソッドを呼び出す行を注釈化します。

```
    // 複数件の店舗データを挿入
    // p.AddMultiEntity(table);
    // Console.WriteLine("複数件の店舗データを挿入しました。続行するには何かキーを押してください. . . ");
    // Console.ReadKey();
```

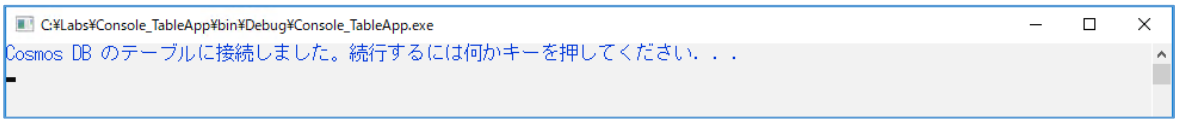
3. Table API SDK を使用する .NET コンソール アプリケーションの作成

3. program クラスに作成した GetPhoneNumberFromOneEntity メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

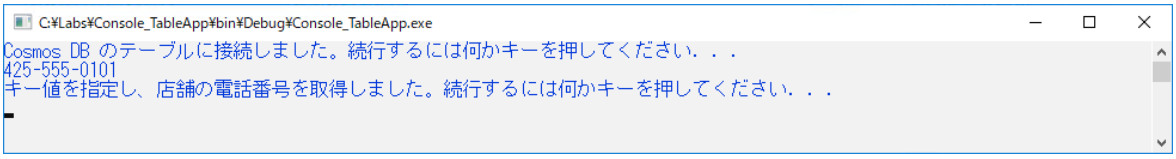
```
        // キー値を指定して店舗の電話番号を取得
        p.GetPhoneNumberFromOneEntity(table);
        Console.WriteLine("キー値を指定し、店舗の電話番号を取得しました。続行するには何かキーを押してください. . . ");
        Console.ReadKey();
```

4. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックして、デバッグ実行します。

5. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



6. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



7. TableQuery クラスを使用して、テーブルに格納した、すべてのエンティティを取得する GetAllEntity メソッドを作成するために、program クラスに次のコードを追加します。

```
public void GetAllEntity(CloudTable table)
{
    // すべてのエンティティを取得する TableQuery オブジェクトを作成
    TableQuery<ShopEntity> query = new TableQuery<ShopEntity>();

    // TableQuery オブジェクトを実行して取得した店舗データの表示
    foreach (ShopEntity entity in table.ExecuteQuery(query))
    {
        Console.WriteLine("{0}¥t{1}¥t{2}¥t{3}¥t{4}", entity.PartitionKey, entity.RowKey,
            entity.ShopName, entity.ShopOwner, entity.PhoneNumber);
    }
}
```

8. Main() メソッドの GetPhoneNumberFromOneEntity メソッドを呼び出す行を注釈化します。

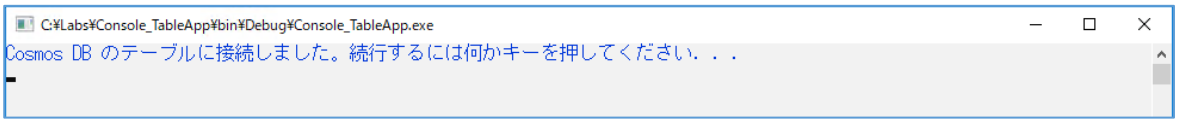
```
        // キー値を指定して店舗の電話番号を取得
        // p.GetPhoneNumberFromOneEntity(table);
        // Console.WriteLine("キー値を指定し、店舗の電話番号を取得しました。続行するには何かキーを押してください. . . ");
        // Console.ReadKey();
```

9. program クラスに作成した GetAllEntity メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

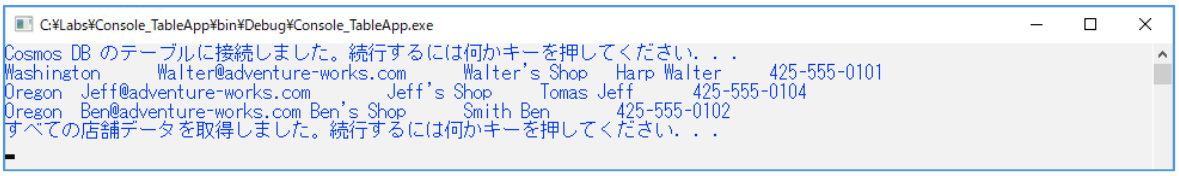
```
        // すべての店舗データを取得
        p.GetAllEntity(table);
        Console.WriteLine("すべての店舗データを取得しました。続行するには何かキーを押してください. . . ");
        Console.ReadKey();
```

10. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックして、デバッグ実行します。

11. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



12. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



13. TableQuery クラスを使用して、テーブルに格納したオレゴン州の店舗のエンティティを取得する GetAllEntityFromPartition メソッドを作成するために、program クラスに次のコードを追加します。

```
public void GetAllEntityFromPartition(CloudTable table)
{
    // オレゴン州の店舗のエンティティを取得する TableQuery オブジェクトを作成
    TableQuery<ShopEntity> query = new TableQuery<ShopEntity>()
        .Where(TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "Oregon"));
    foreach (ShopEntity entity in table.ExecuteQuery(query))

    // 取得した店舗データの表示
    {
        Console.WriteLine("{0}¥t{1}¥t{2}¥t{3}¥t{4}", entity.PartitionKey, entity.RowKey,
                                entity.ShopName, entity.ShopOwner, entity.PhoneNumber);
    }
}
```

14. Main() メソッドの GetAllEntity メソッドを呼び出す行を注釈化します。

```
        // すべての店舗データを取得
        // p.GetAllEntity(table);
        // Console.WriteLine("すべての店舗データを取得しました。続行するには何かキーを押してください. . . ");
        // Console.ReadKey();
```

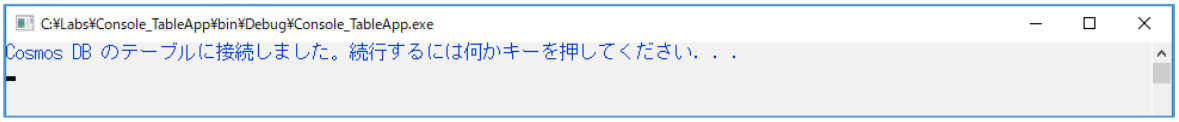
3. Table API SDK を使用する .NET コンソール アプリケーションの作成

15. program クラスに作成した GetAllEntityFromPartition メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

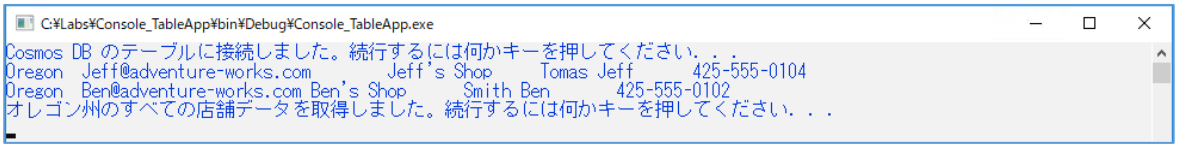
```
// 同一パーティション内のすべての店舗データを取得
p.GetAllEntityFromPartition(table);
Console.WriteLine("オレゴン州のすべての店舗データを取得しました。続行するには何かキーを押してください. . . ");
Console.ReadKey();
```

16. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックして、デバッグ実行します。

17. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



18. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



3.6 アプリケーション コードによるエンティティの置き換え

エンティティを置き換えるには、以下の手順を実行します。

- 対象のエンティティをテーブルから取得する
- エンティティ オブジェクトを変更する
- 変更をテーブルに戻して保存する

置換処理は、TableOperation クラスの Replace や Merge または InsertOrReplace や InsertOrMerge メソッドを使用できます。

1. キー値を指定して取得した Shop エンティティの電話番号を置き換える UpdateOneEntity メソッドを作成するために、program クラスに次のコードを追加します。

```
public void UpdateOneEntity(CloudTable table)
{
    // キー値を指定して対象のエンティティを取得する TableOperation オブジェクトを作成
    TableOperation retrieveOperation = TableOperation
                                        .Retrieve<ShopEntity>("Washington", "Walter@adventure-works.com");

    // エンティティを取得する操作の実行
    TableResult retrievedResult = table.Execute(retrieveOperation);

    // 取得した結果を ShopEntity にマップする
    ShopEntity updateEntity = (ShopEntity)retrievedResult.Result;

    if (updateEntity != null)
    {
        // 新しい電話番号を設定
        updateEntity.PhoneNumber = "425-555-0202";

        // 対象の ShopEntity を置き換える TableOperation オブジェクトを作成する
        TableOperation updateOperation = TableOperation.Replace(updateEntity);

        // 置換操作を実行して、テーブルに反映する
        table.Execute(updateOperation);

        Console.WriteLine("Entity updated.");
    }
    else
    {
         Console.WriteLine("Entity could not be retrieved.");
    }
}
```

3. Table API SDK を使用する .NET コンソール アプリケーションの作成

2. Main() メソッドの GetAllEntityFromPartition メソッドを呼び出す行を注釈化します。

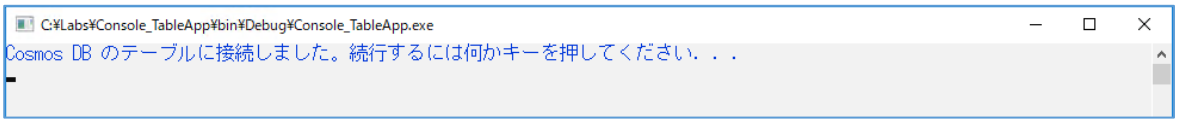
```
// 同一パーティション内のすべての店舗データを取得
// p.GetAllEntityFromPartition(table);
// Console.WriteLine("オレゴン州のすべての店舗データを取得しました。続行するには何かキーを押してください. . . ");
// Console.ReadKey();
```

3. program クラスの UpdateOneEntity メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

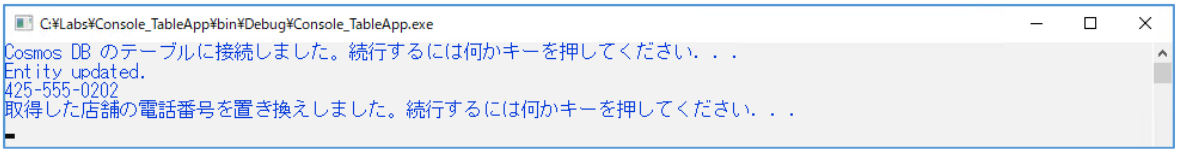
```
// キー値を指定して取得した店舗の電話番号を置き換え
p.UpdateOneEntity(table);
p.GetPhoneNumberFromOneEntity(table);
Console.WriteLine("取得した店舗の電話番号を置き換えました。続行するには何かキーを押してください. . . ");
Console.ReadKey();
```

4. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックして、デバッグ実行します。

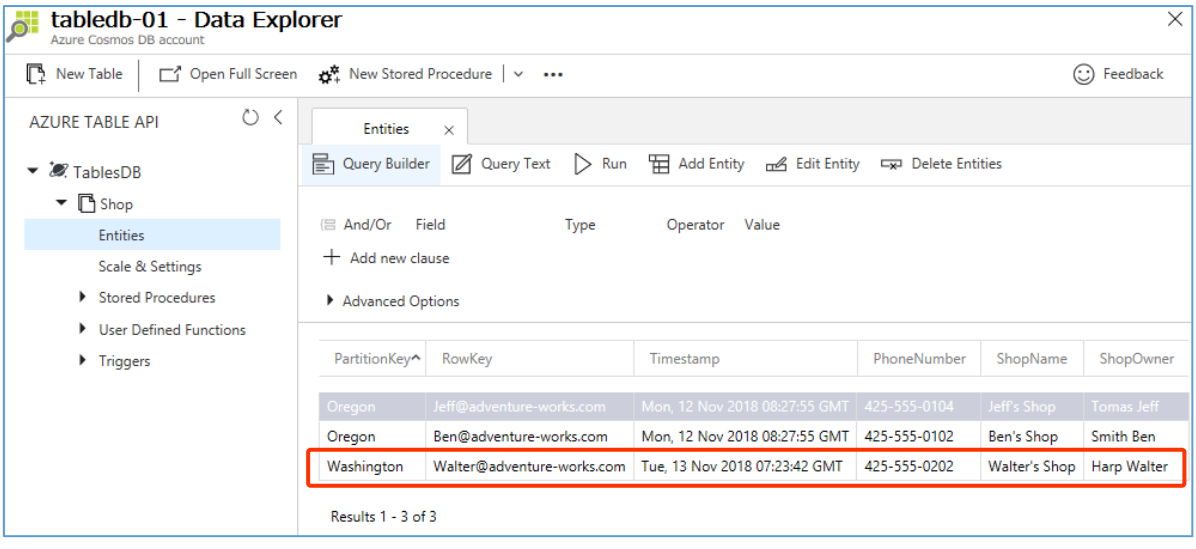
5. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



6. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



7. Azure ポータルで [Data Explorer] ブレードから対象のエンティティを参照して、電話番号が置き換えられたことを確認します。



3.7 アプリケーション コードによるエンティティの削除

ここでは、アプリケーション コードにより、エンティティを 1 件ずつ削除する方法と複数件をまとめて追加する方法を確認します。エンティティを 1 件ずつ削除するには、以下の手順を実行します。

- 対象のエンティティをテーブルから取得する
- 対象のエンティティを削除する

1 件ずつ削除するには、TableOperation クラスの Delete メソッドを使用します。

1. キー値を指定して取得した Shop エンティティを削除する DeleteOneEntity メソッドを作成するために、program クラスに次のコードを追加します。

```
public void DeleteOneEntity(CloudTable table)
{
    // キー値を指定して対象のエンティティを取得する TableOperation オブジェクトを作成
    TableOperation retrieveOperation = TableOperation
                                        .Retrieve<ShopEntity>("Washington", "Walter@adventure-works.com");

    // エンティティを取得する操作の実行
    TableResult retrievedResult = table.Execute(retrieveOperation);

    // 取得した結果を ShopEntity にマップする
    ShopEntity deleteEntity = (ShopEntity)retrievedResult.Result;

    // 対象の ShopEntity を削除する TableOperation オブジェクトを作成する
    if (deleteEntity != null)
    {
        TableOperation deleteOperation = TableOperation.Delete(deleteEntity);

        // 削除操作を実行して、テーブルに反映する
        table.Execute(deleteOperation);

        Console.WriteLine("Entity deleted.");
    }
    else
    {
    Console.WriteLine("Could not retrieve the entity.");
    }
}
```

2. Main() メソッドの UpdateOneEntity メソッドを呼び出す行を注釈化します。

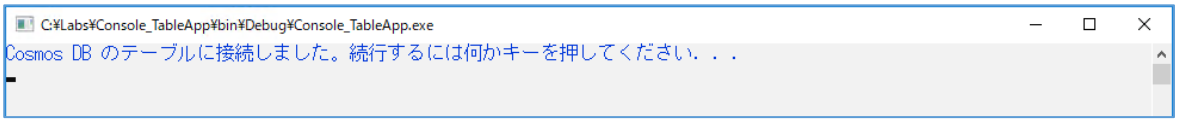
```
// キー値を指定して取得した店舗の電話番号を置き換え
// p.UpdateOneEntity(table);
// p.GetPhoneNumberFromOneEntity(table);
// Console.WriteLine("取得した店舗の電話番号を置き換えしました。続行するには何かキーを押してください. . . ");
// Console.ReadKey();
```

3. program クラスに作成した DeleteOneEntity メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

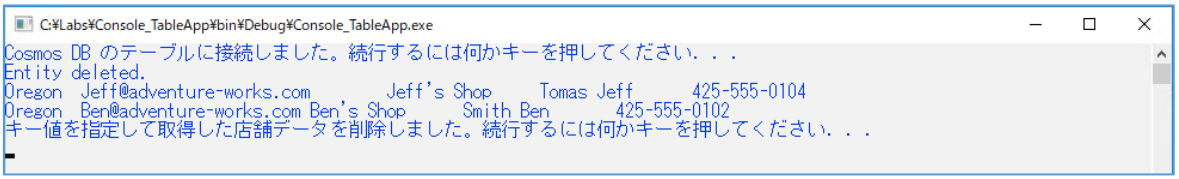
```
// キー値を指定して取得した店舗データを削除
p.DeleteOneEntity(table);
p.GetAllEntity(table);
Console.WriteLine("キー値を指定して取得した店舗データを削除しました。続行するには何かキーを押してください. . . ");
Console.ReadKey();
```

4. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックして、デバッグ実行します。

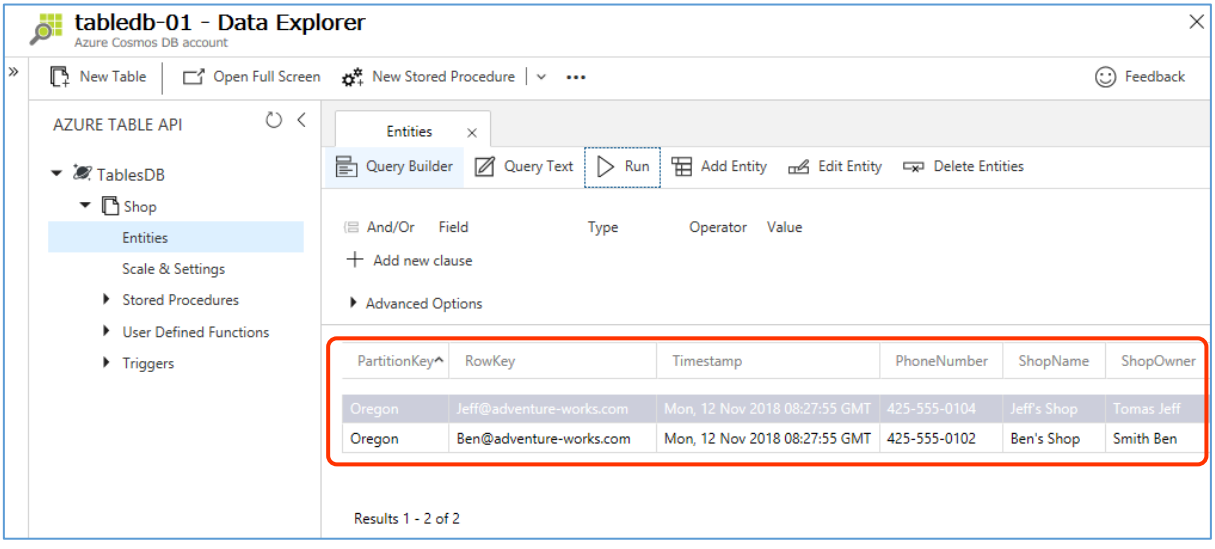
5. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



6. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



7. Azure ポータルの [Data Explorer] ブレードから対象のエンティティを参照して、削除されたことを確認します。



8. 次に複数件のエンティティをまとめてテーブルに追加する DeleteMultiEntity メソッドを作成するために、program クラスに次のコードを追加します。

```
public void DeleteMultiEntity(CloudTable table)
{
    // TableBatchOperation オブジェクトの作成
    TableBatchOperation batchOperation = new TableBatchOperation();

    // 1 件目の削除対象の Shop エンティティを作成
    ShopEntity shop1 = new ShopEntity("Oregon", "Jeff@adventure-works.com");
    shop1.ETag = "*";

    // 2 件目の削除対象の Shop エンティティの作成
    ShopEntity shop2 = new ShopEntity("Oregon", "Ben@adventure-works.com");
    shop2.ETag = "*";

    // 2 件の Shop エンティティを TableBatchOperation オブジェクトに渡す
    batchOperation.Delete(shop1);
    batchOperation.Delete(shop2);

    // 一括削除操作の実行
    table.ExecuteBatch(batchOperation);
}
```

ワン ポイント

エンティティの ETag プロパティ値は、エンティティを生成すると内部的に設定され、楽観的同時実行制御を行うために使用されます。例えば、エンティティを取得して置き換える場合、ETag プロパティが現在格納されているものと一致する必要があります。TableBatchOperation を使用して、複数のユーザーが同じエンティティを編集していて、お互いの変更を上書きしたくないような場合、この機能が必要となります。自習書では、Etag に “*” を設定していますが、これにより同時実行制御は無効化されていることに注意してください。

9. 重複したキー値を持つエンティティの挿入を避けるため、Main() メソッド の DeleteEntity メソッドを呼び出す行を注釈化します。

```
    // 一件の店舗データを削除
    // p.DeleteOneEntity(table);
    // Console.WriteLine("一件の店舗データを削除しました。続行するには何かキーを押してください．．．");
    // Console.ReadKey();
```

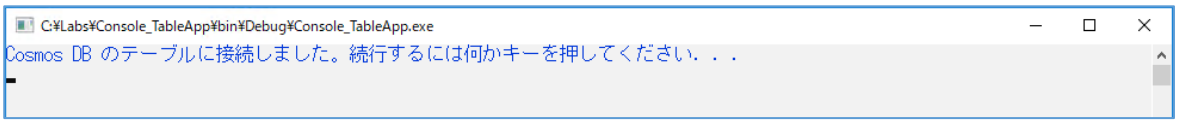
10. program クラスに作成した AddMultiEntity メソッドを Main() メソッドから呼び出すために、次のコードを追加します。

```
    // 複数件の店舗データを削除
    p.DeleteMultiEntity(table);
    Console.WriteLine("複数件の店舗データを削除しました。続行するには何かキーを押してください．．．");
    Console.ReadKey();
```

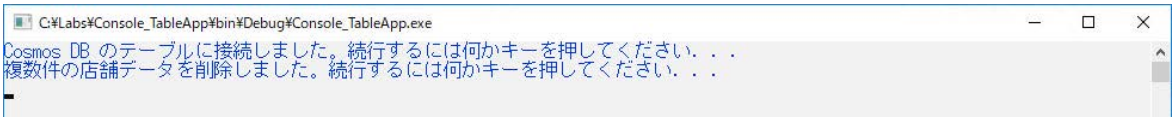
3. Table API SDK を使用する .NET コンソール アプリケーションの作成

11. ここで記述したテーブルを作成するコードを検証するために、[ソリューション構成] が「Debug」に設定されていることを確認して、[開始] ボタンをクリックして、デバッグ実行します。

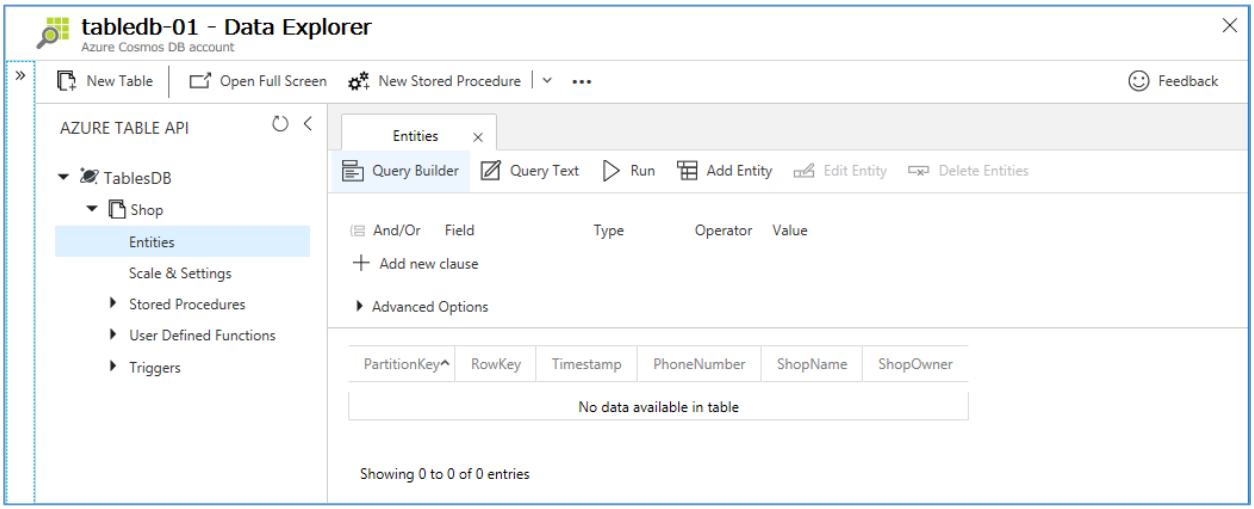
12. コマンド プロンプトが起動し、次のメッセージが表示されたら、キーボードの任意のキーをクリックして、処理を進めます。



13. 次のメッセージが表示されたら、キーボードの任意のキーをクリックして、コンソール アプリケーションのデバッグ実行を止めます。



14. Azure ポータルの [Data Explorer] ブレードから対象のエンティティを参照して、削除されたことを確認します。



ワン ポイント
すべてのエンティティが削除された状態になります。

4. データの移行

この章では、Azure Table Storage に格納されたエンティティを Azure Cosmos DB のテーブルに移行する利点を説明し、移行手順を紹介します。

4.1 Azure Table Storage から Azure Cosmos DB のテーブルに移行する利点

Azure Table Storage と Azure Cosmos DB Table API は同じキー/バリュー形式のデータ モデルを共有し、同じ SDK を使用して、CRUD 操作を行うことができます。Azure Cosmos DB に移行することで、Azure Cosmos DB 入門編で説明した以下の機能を使用できます。

- 世界規模での専用スループット
- ターンキー グローバル分散
- 自動セカンダリ インデックス作成
- 定義済みの 5 つの整合性レベル
- 99 パーセンタイルで 10 ミリ秒未満の待ち時間
- 高可用性の保証

Azure Table Storage と Azure Cosmos DB Table API の機能的な相違点については、下表を参照してください。

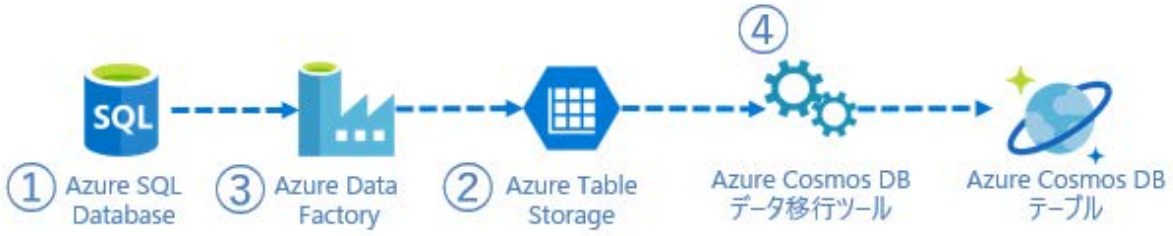
項目	Azure Table Storage	Azure Cosmos DB Table API
スループット	● 可変スループット モデル ● 20,000 操作/秒までの制限	● アカウントにはスループットの上限がない ● テーブルあたり 10,000,000 操作/秒以上にも対応可能 ● テーブルごとの予約済みスループットは SLA によって保証される
グローバル分散	● オプションで 1 つの読み取り可能なセカンダリ リージョンを構成可能 ● 利用者からの geo フェールオーバー操作は開始できない	● 30 以上のリージョンを対象にターンキー グローバル分散が可能 ● いつでも、どのリージョンにでも、自動フェールオーバー、および、手動フェールオーバーを実行可能
インデックス	● PartitionKey と RowKey のプライマリ インデックスのみが作成される	● すべてのプロパティに対して、自動でインデックスが作成される
エンティティ	● 最大サイズ 1 MB	● 最大サイズ 2 MB
クエリ	● プライマリ インデックス キーを使用しない場合、テーブル スキャン操作になる	● すべてのプロパティでインデックスを使用したクエリ実行が可能
データ整合性	● プライマリ リージョン内では厳密な整合性 ● セカンダリ リージョンでは最終的な整合性	● アプリケーション要件に基づき可用性、待ち時間、スループット、および整合性のトレードオフが構成される定義済みの 5 つの整合性レベル (厳密、有界整合性制約、セッション、一貫性のあるプレフィックス、最終的)から選択可能
レイテンシ SLA	● 待ち時間に SLA の設定はない	● 読み取りと書き込みの待ち時間は数ミリ秒 ● 99 パーセンタイルの 10 ミリ秒未満の読み取り待ち時間と 15 ミリ秒未満の書き込み待ち時間を SLA (99.99 %) で保証
可用性 SLA	● 99.99 % の可用性	● 単一リージョン (99.99 %)

		● 複数リージョン シングル マスター構成 (書き込み 99.99 %、読み取り 99.999 %) ● 複数リージョン マルチマスター構成 (99.999 %)
課金	● 選択したストレージ構成とサイズによる課金	● テーブルに設定したスループットによる課金

なお、Azure Cosmos DB の包括的な SLA の詳細は、「Azure Cosmos DB の SLA」を参照してください。上記表に示された Azure Cosmos DB Table API における利点を活用されたいと考えた場合、Azure Table Storage から Azure Cosmos DB のテーブルにエンティティを移行することが可能です。以降では、ツールを使用したデータ移行手順を説明します。

4.2 データの移行手順

Azure Table Storage に格納されたエンティティを Azure Cosmos DB のテーブルに移行する作業では、Azure Cosmos DB データ移行ツール、または AzCopy ユーティリティを使用できます。自習書では、データ移行ツールを使用して Azure Table Storage に格納されたエンティティを Azure Cosmos の DB テーブルに移行する方法を説明しています。



なお、前準備として、ソースとして使用する Azure Table Storage にエンティティを格納するため、以下 ①~③ の手順で Azure SQL Database に Adventureworkslt サンプルデータベースから Azure Data Factory のデータ コピー ウィザードを使用して、Saleslt.Customer テーブルのデータを Azure Table Storage にコピーしています。

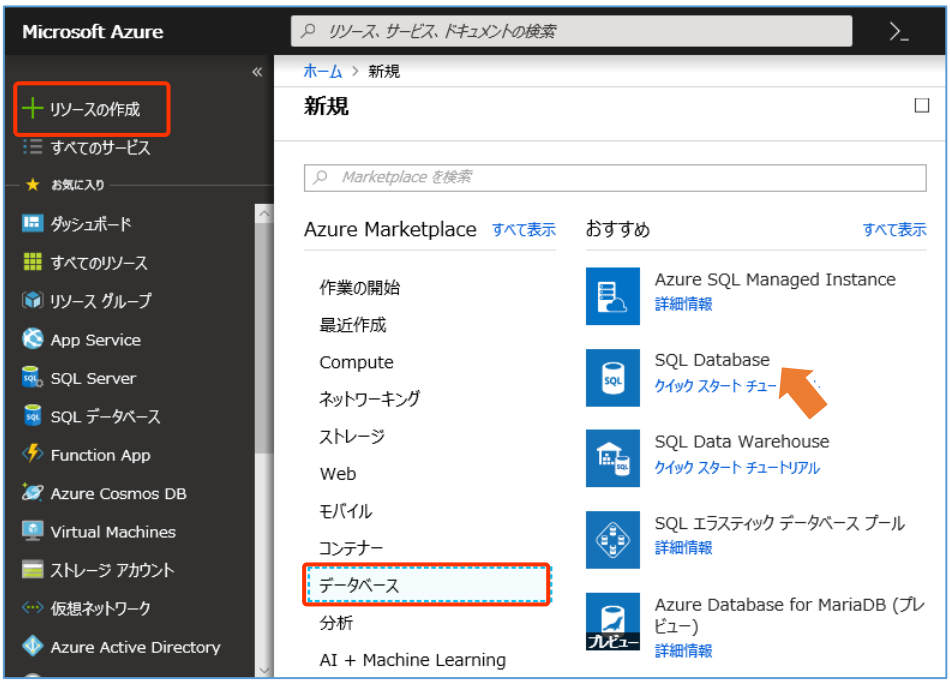
① Azure SQL Database の Adventureworkslt サンプル データベースをデプロイする
② Azure ストレージ アカウントをデプロイして Table Storage にテーブルを作成する
③ Azure Data Factory コピー データ ウィザードで Azure SQL Database のテーブルのデータを Azure Table Storage のテーブルにコピーする
④ データ移行ツールで Azure Table Storage のテーブルのエンティティを Azure Cosmos の DB テーブルに移行する

現時点では、Azure Data Factory のデータ コピー ウィザードは、Azure SQL Database のテーブルを直接、Azure Cosmos DB のテーブルにコピーすることができません。このため、この章で紹介する手順は、Azure SQL Database のテーブルから、Azure Cosmos の DB テーブルにデータ移行するシナリオでも役に立つでしょう。

4.2.1 Azure SQL Database の AdventureworksIt サンプル データベースをデプロイする

最初に Azure ポータルから Azure SQL Database の AdventureworksIt サンプルデータベースをデプロイします。AdventureworksIt データベースに含まれる SalesIt.Customer テーブルには Adventure Works 社の商品を販売する店舗情報が格納されています。Azure Table Storage にコピーするデータ フィールドを選択するビューを追加しておきます。

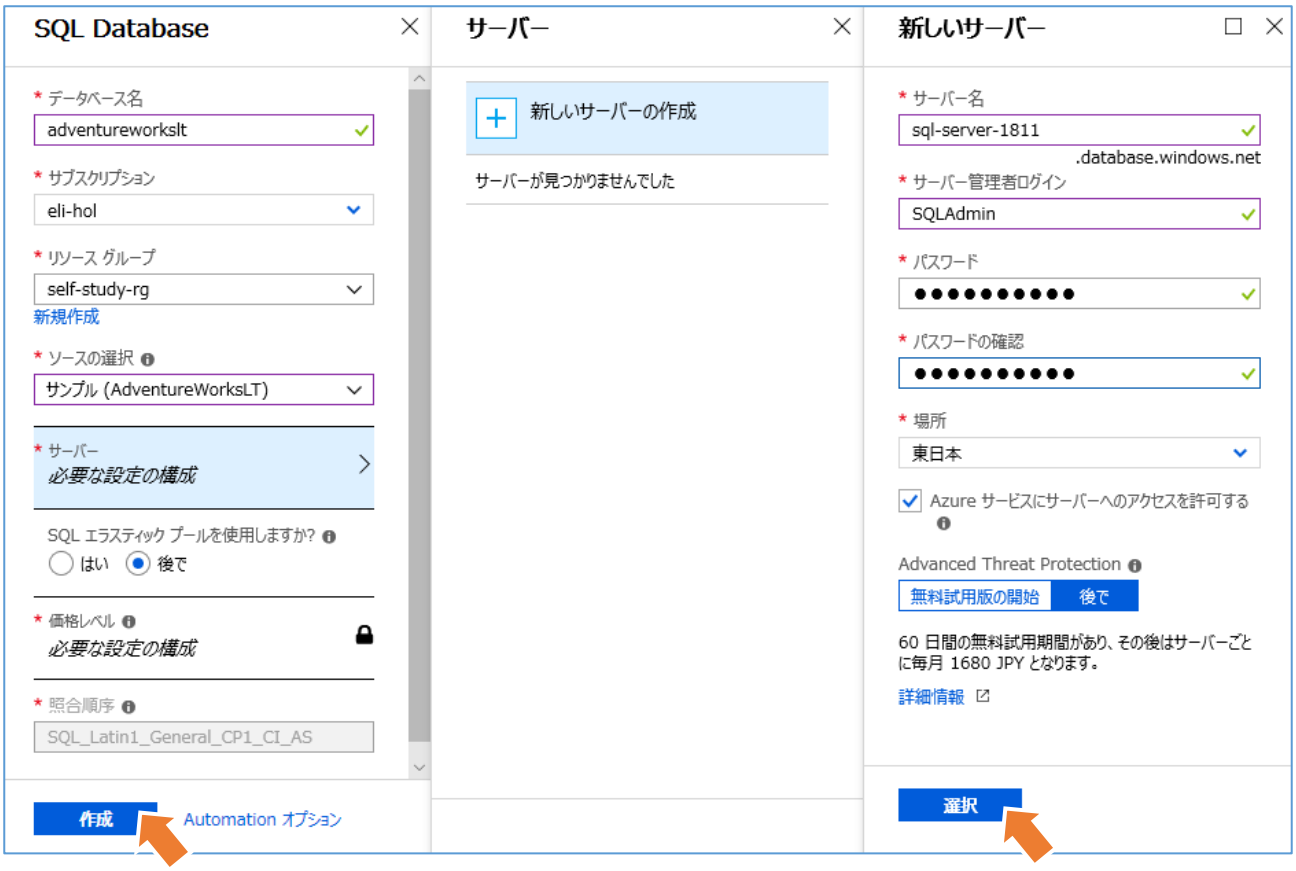
1. Azure 管理ポータルのハブ メニューから [リソースの作成] ⇒ [データベース] ⇒ [SQL Database] をクリックします。



2. [SQL Database] ブレードが表示されるので下表に従い入力し、[作成] をクリックします。

パラメーター	設定する値
データベース名	adventureworksIt
サブスクリプション	お持ちのサブスクリプション名
リソース グループ	Azure Cosmos DB を作成したリソースグループを選択します。
ソースの選択	サンプル (AdventureWorksLT)
サーバー	演習用に使用できる論理サーバーがない場合、下表に従い設定し、新規作成します。 パラメーター: 設定する値 サーバー名: Azure データ センター内で一意となる任意の値 (小文字・数値・ハイフン '-' のみを使用) サーバー管理者ログイン: 任意のログイン名 パスワード: 任意のパスワード 場所: 選択可能な任意の日本国内のリージョン Azure サービスにサーバーへのアクセスを許可する: 選択します Advenced Treat Protection: 後で
SQL エラステック プールを使用しますか?	後で
価格レベル	Basic, 2GB

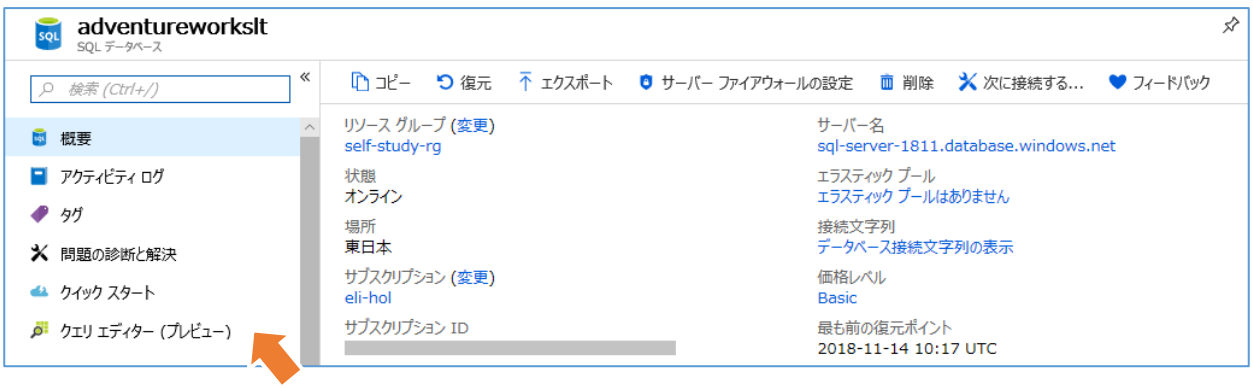
4. データの移行



ワンポイント
新規に論理サーバーを作成した場合、デプロイ完了まで、3~4 分程度かかります。

3. デプロイが完了すると Azure ポータルの [通知] で「展開が成功しました」と表示されるので、[リソースに移動] をクリックします。

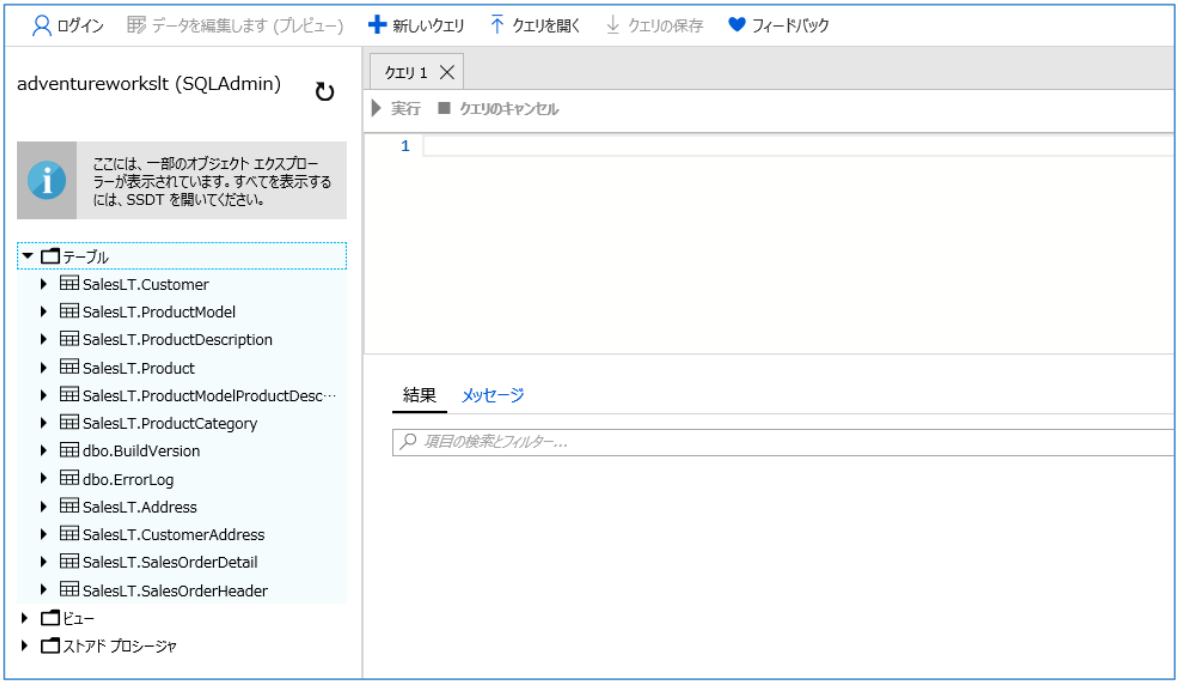
4. adventureworkslt のブレードが表示されたら、[クエリ エディター (プレビュー)] をクリックして、クエリ エディターを開きます。



4. データの移行

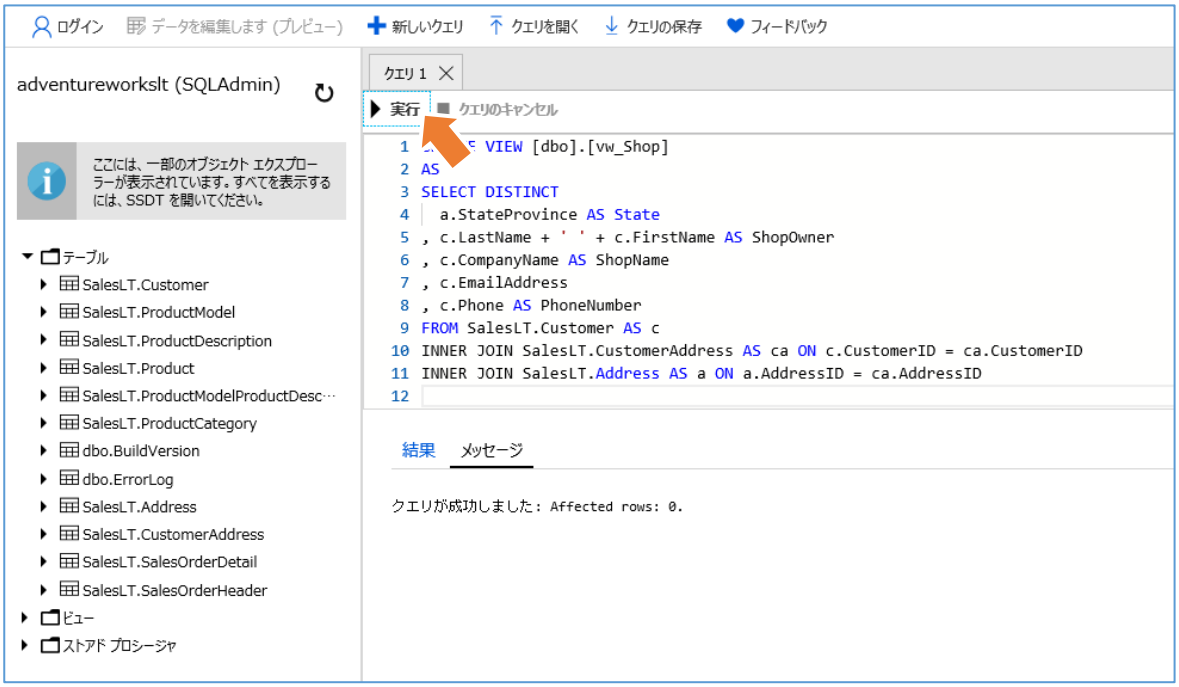
5. [クエリ エディター (プレビュー)] ブレードの [承認の種類] で [SQL Server 認証] を選択して、論理サーバーに設定したサーバー管理者ログインとパスワードを入力して、[OK] ボタンをクリックします。

6. クエリ エディターが開いたら、[テーブル] ノードを展開して、データベースに作成されているテーブルを確認します。



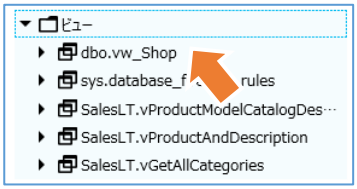
7. dbo.vw_Shop という名前で、移行対象の列のみを選択するビューを作成するために、以下の CREATE VIEW ステートメントをコピーして、エディターに貼り付けて、[実行] をクリックします。

```
CREATE VIEW [dbo].[vw_Shop]
AS
SELECT DISTINCT
    a.StateProvince AS State
, c.LastName + ' ' + c.FirstName AS ShopOwner
, c.CompanyName AS ShopName
, c.EmailAddress
, c.Phone AS PhoneNumber
FROM SalesLT.Customer AS c
INNER JOIN SalesLT.CustomerAddress AS ca ON c.CustomerID = ca.CustomerID
INNER JOIN SalesLT.Address AS a ON a.AddressID = ca.AddressID
```



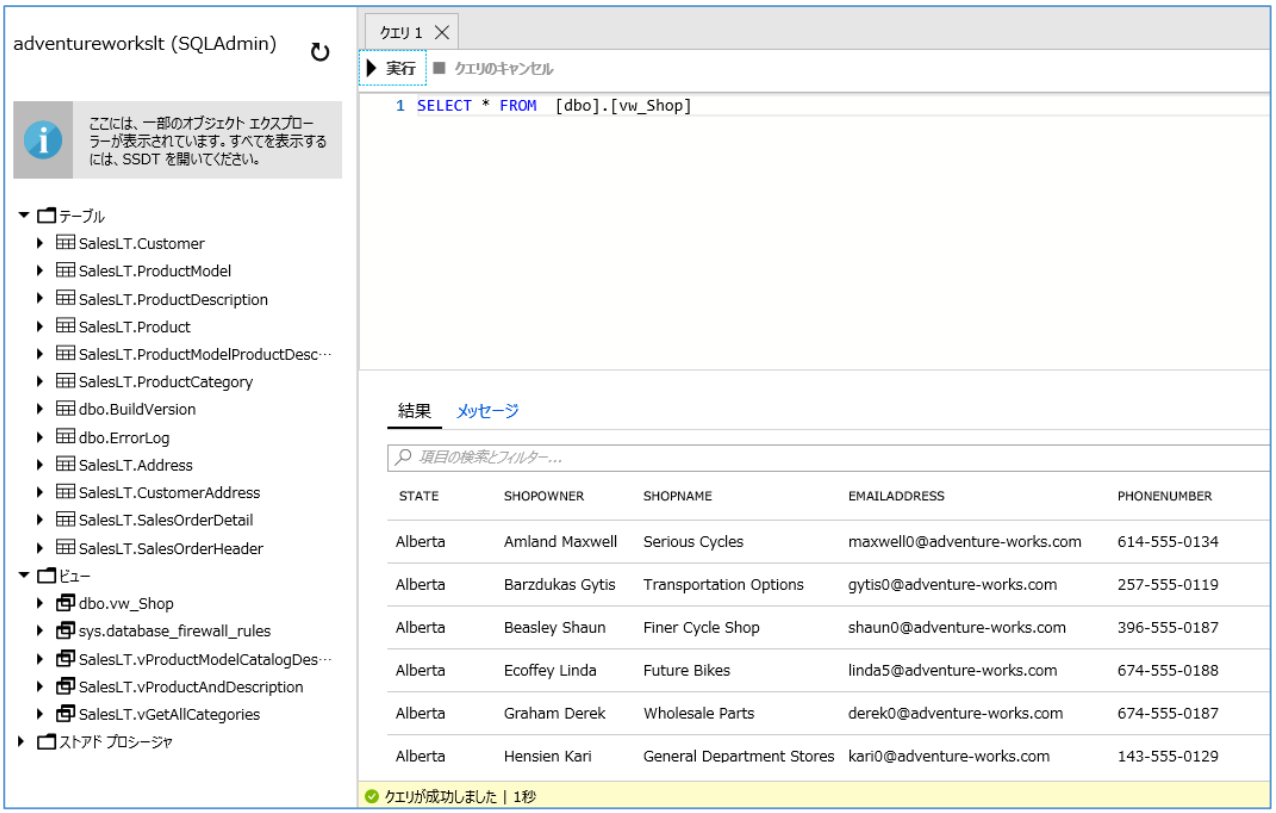
ワンポイント
クエリが正常に実行されると「クエリが成功しました: Affected rows: 0.」というメッセージが表示されます。

8. [ビュー] ノードを展開して、データベースに dbo.vw_Shop ビューが作成されたことを確認します。



9. 作成したビューを確認するため、以下の SELECT ステートメントをエディターに貼り付けた後、[実行] をクリックします。

```
SELECT * FROM [dbo].[vw_Shop]
```



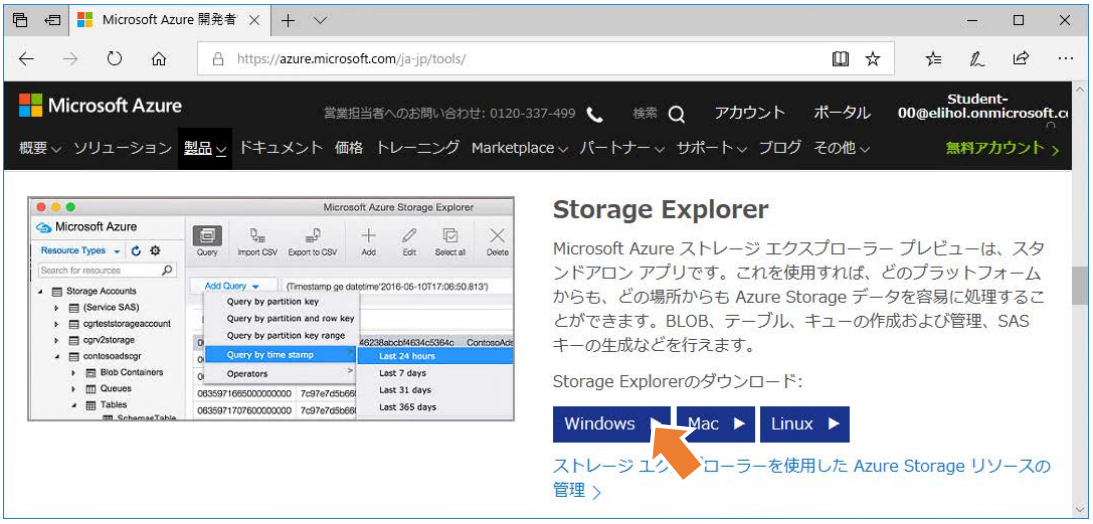
ワンポイント

407 件の店舗情報が表示されます。作成したビューで取得されフィールドは、2 章、3 章で Azure Cosmos DB に作成したテーブル構造に合わせています。以降の手順では、Azure ストレージ アカウントをデプロイして Table Storage にテーブルを作成します。

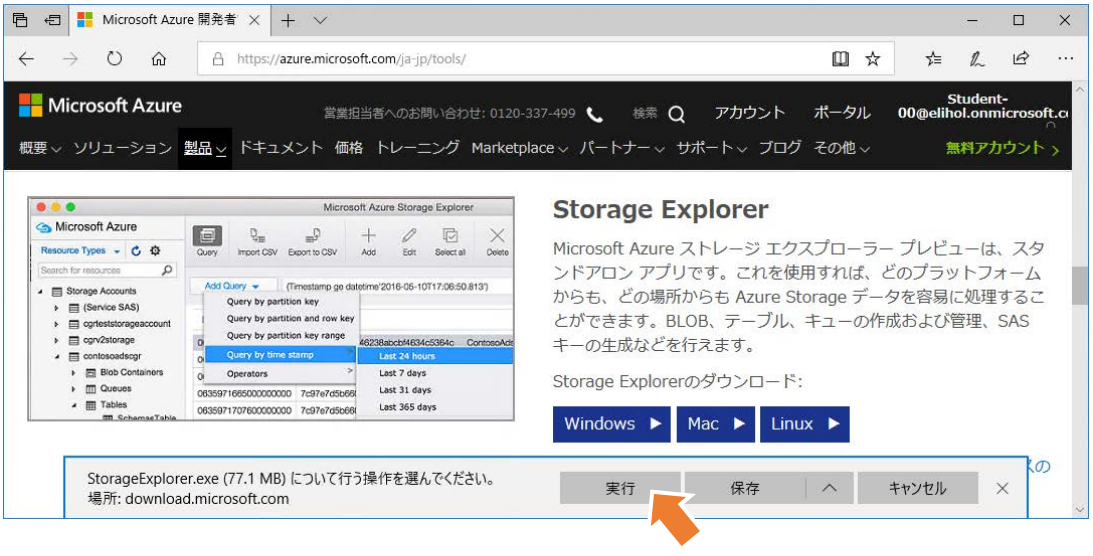
4.2.2 Azure ストレージ アカウントをデプロイして Table Storage にテーブルを作成する

ここでは、Azure ストレージ アカウントをデプロイして Table Storage にテーブルを作成します。作成したテーブルに対するプロパティの構成やエンティティの追加作業は、現時点では Azure ポータルからは行えませんが、Microsoft Azure Storage Explorer から実行可能なため、ローカルに Storage Explorer をインストールします。

1. Microsoft Edge で「**https://azure.microsoft.com/ja-jp/tools/**」にアクセスし、表示されたページをスクロールし、「Storage Explorer」が表示されたら、使用している OS 環境に対応するボタンをクリックします。

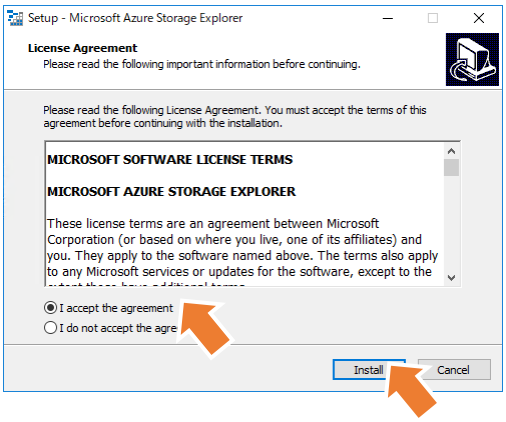


2. [実行] ボタンをクリックして、Storage Explorer のインストールを開始します。

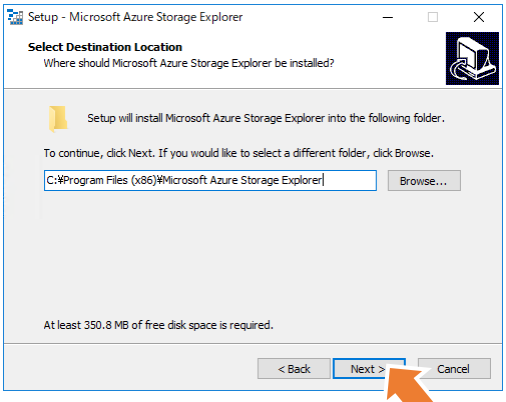


3. [ユーザー アカウント制御] が表示されたら [はい] ボタンをクリックします。

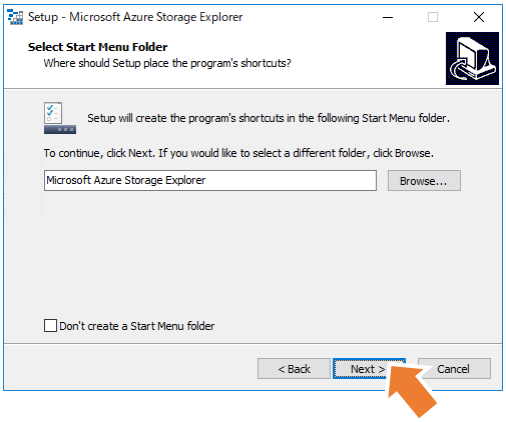
4. [License Agreement] で [I accept the agreement] を選択して、[Install] ボタンをクリックします。



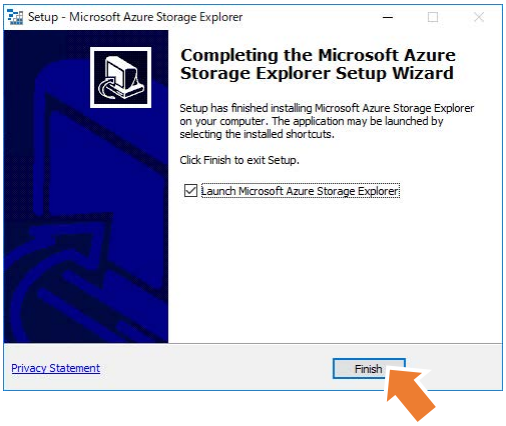
5. [Select Destination Location] が表示されるので、既定の設定で、[Next] ボタンをクリックします。



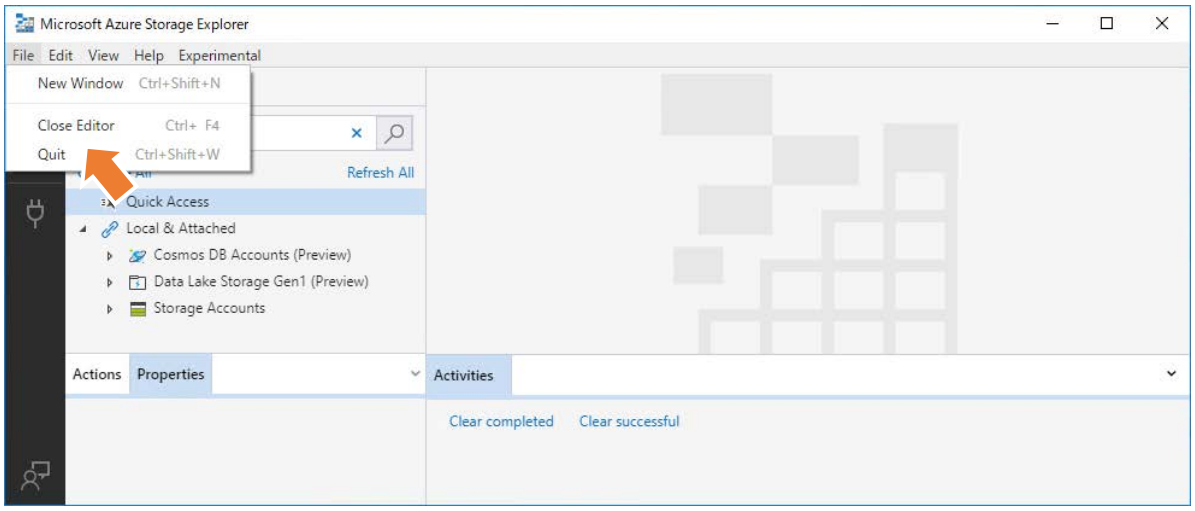
6. [Select Start Menu Folder] が表示されるので、既定の設定で、[Next] ボタンをクリックします。



7. インストールが完了したら、[Finish] ボタンをクリックします。

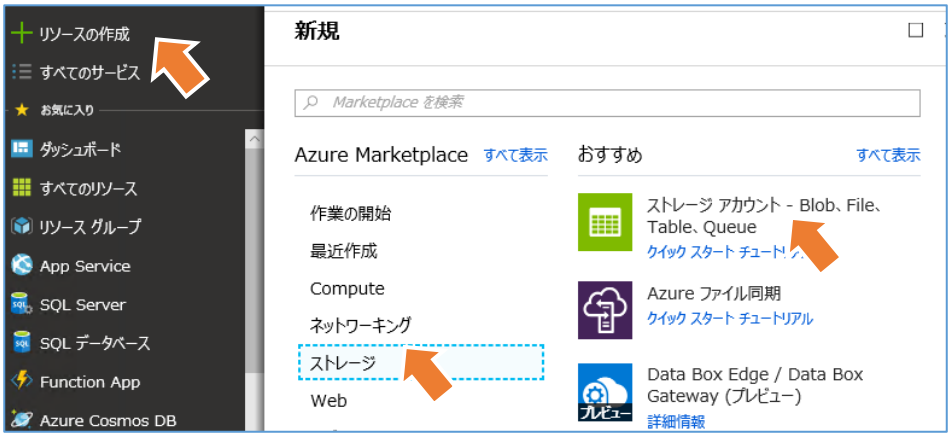


8. Microsoft Azure Storage Explorer が起動することを確認して、[File] メニューから [Quit] を選択して終了します。



ワンポイント
Microsoft Azure Storage Explorer を使用すると、ストレージ アカウントだけでなく、Azure Cosmos DB アカウントに対する様々な操作も可能になります。

9. Azure ポータルのハブ メニューで [リソースの作成] をクリックして [ストレージ]⇒[ストレージ アカウント – Blob、File、Table、Queue] を選択します。



10. [ストレージ アカウントの作成] の [基本] ブレードで下表に従い入力し、[確認および作成] をクリックします。

カテゴリ	パラメーター	設定する値
プロジェクトの詳細	サブスクリプション	お持ちのサブスクリプション名を選択します。
	リソース グループ	Azure Cosmos DB を作成したリソースグループを選択します。
インスタンスの詳細	ストレージアカウント名	ストレージ アカウントの名前を入力します。この値は、グローバルで一意となる必要があります。名前には、「-」を含めることができないことに注意してください。入力したストレージ アカウント名にドメイン名「.core.windows.net」がつけられます。
	場所	選択可能な任意の日本国内のリージョンを選択します。
	パフォーマンス	「Standard」を選択します。
	アカウントの種類	「StorageV2 (汎用 v2)」を選択します。
	レプリケーション	「ローカル冗長ストレージ (LRS)」を選択します。
	アクセス層	「ホット」を選択します。

ワンポイント

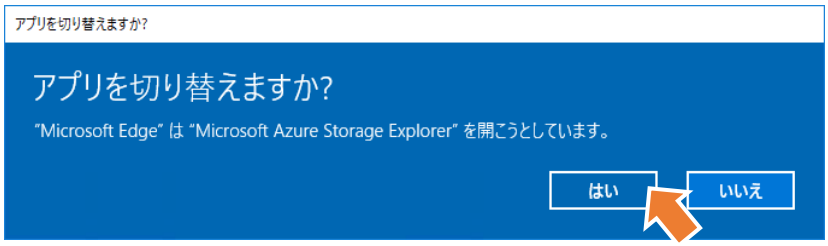
ストレージは、「レプリケーション」の選択により、異なる冗長構成が設定されます。リモートのデータセンターにレプリケーションさせて、セカンダリのデータを参照する場合は、「読み取りアクセス地理冗長ストレージ (RA-GRS)」を選択します。この場合、「パフォーマンス レベル」では、HDD ベースの Standard のみが選択できます。

11. [ストレージア カウントの作成] で設定内容を確認し、[作成] をクリックします。

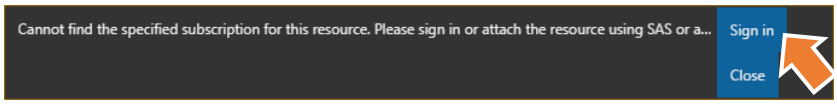
12. Azure ポータルの [通知] で「展開が成功しました」と表示されるので、[リソースに移動] をクリックします。

13. Azure ストレージ アカウントの [概要] ブレードがで、[Explorer で開く] をクリックします。メッセージが表示されたら、[はい] をクリックします。

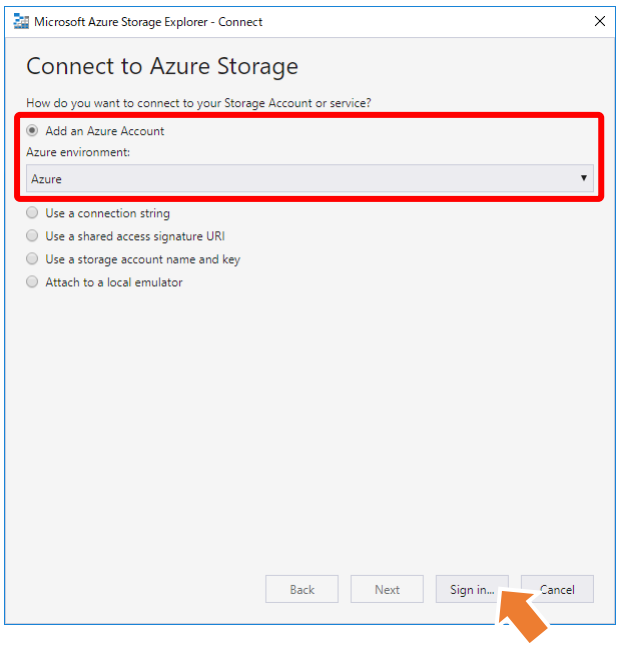
14. 次のメッセージが表示されたら、[はい] をクリックして、Storage Explorer を起動します。



15. Storage Explorer で [Sgin in] をクリックします。

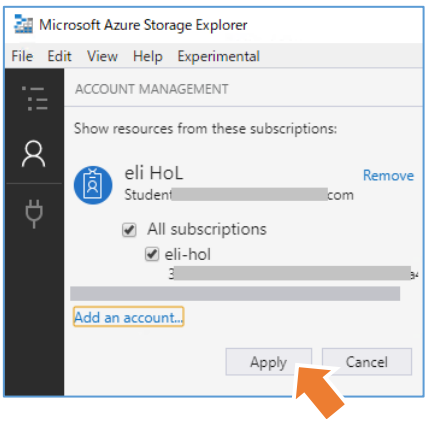


16. [Add an Azure Account] オプションの [Azure environment] リストで「Azure」が選択されていることを確認して、[Sign in] ボタンをクリックします。

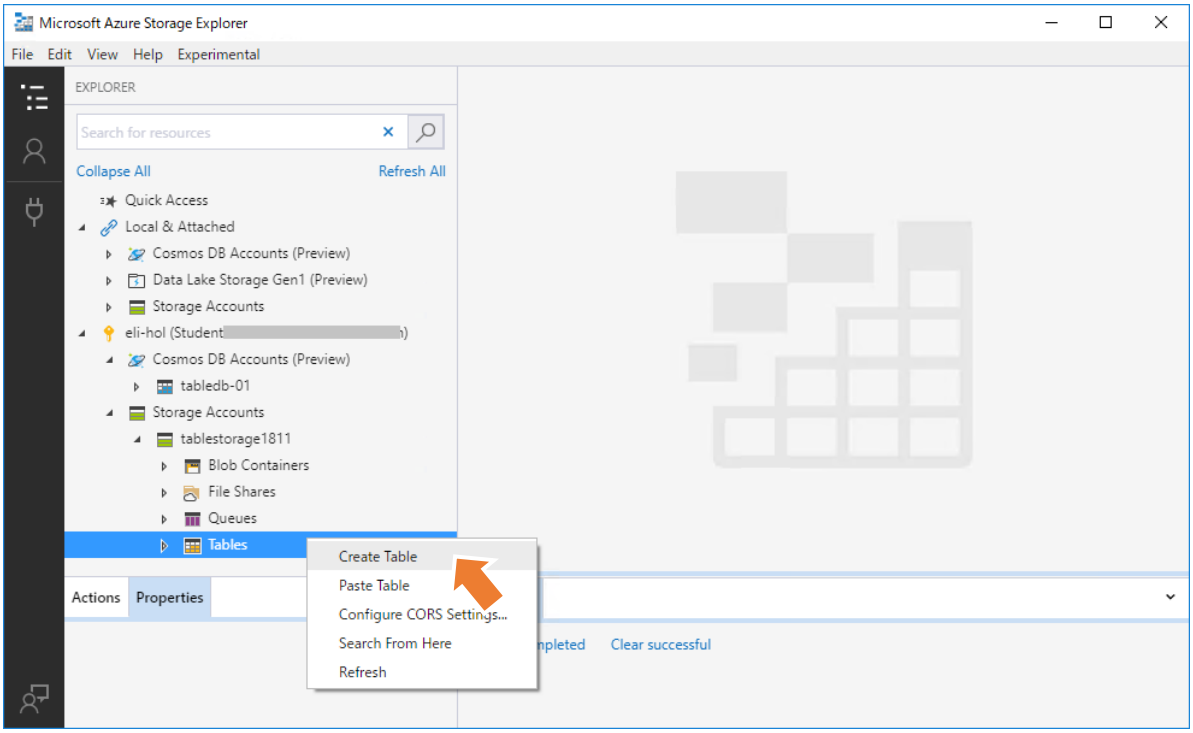


17. ここで [Sign in] が表示されたら、自習書の学習用に使用しているアカウントを入力して、[Next] ボタンをクリックし、[Enter password] でパスワードを入力し、[Sign in] ボタンをクリックします。

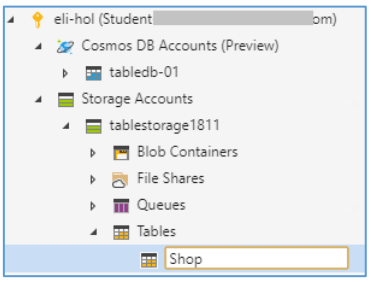
18. [ACCOUNT MANAGEMENT] のツリーで、サブスクリプションに接続したことを確認して [Apply] ボタンをクリックします。



19. Storage Explorer のツリーで、演習用に作成したストレージ アカウントの [Tables] ノードを右クックして、表示されるメニューで [Create Table] をクリックします。

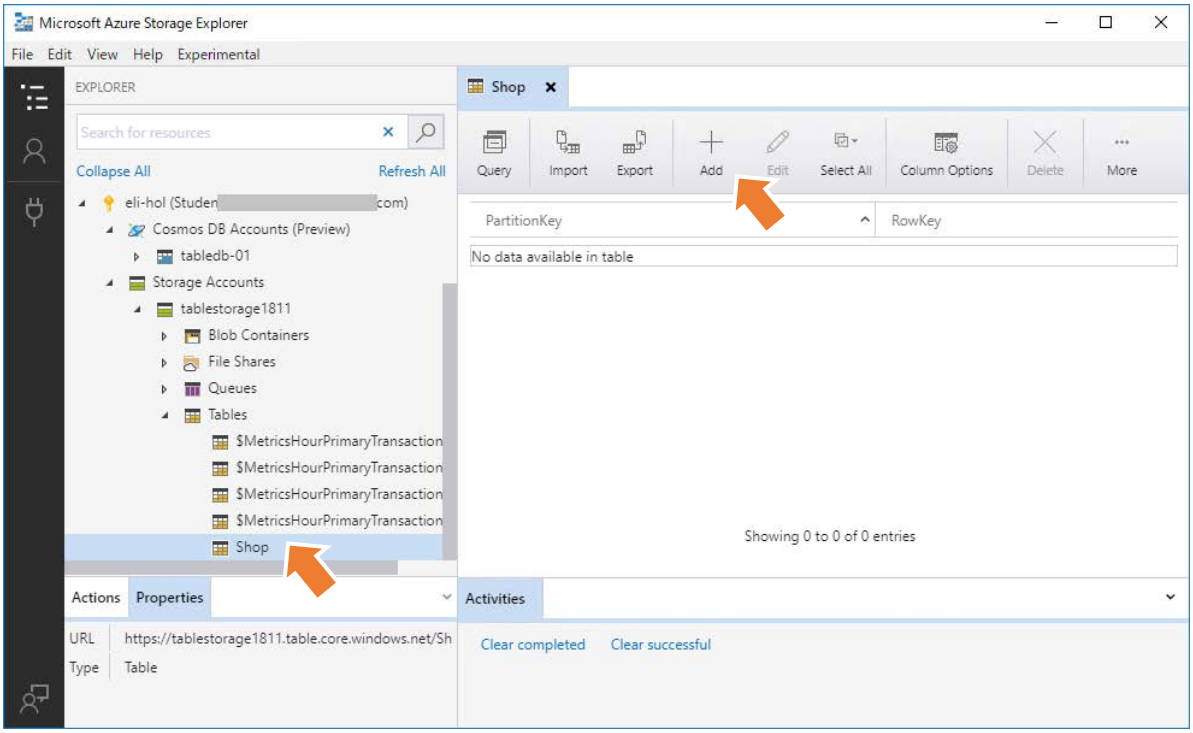


20.「Shop」という名前でテーブルを作成します。



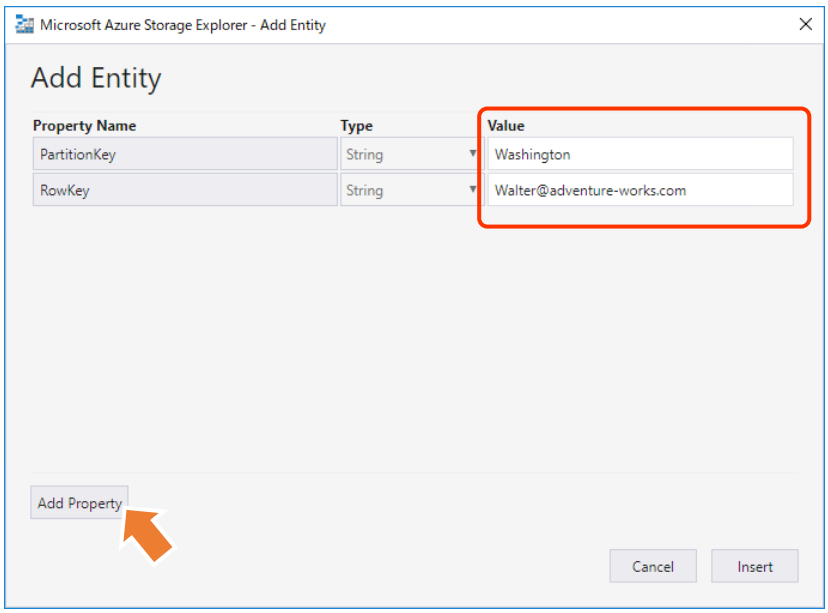
4. データの移行

21. テーブルにエンティティを追加するため、Storage Explorer のツリーで、「Shop」テーブルを選択して、右側に表示される領域で [Add] をクリックします。



22. [Add Entity] が表示されるので、下表に従い入力して、[Add Property] ボタンをクリックします。

Property Name	Type	Value
PartitionKey	String	Washington
RowKey	String	Walter@adventure-works.com

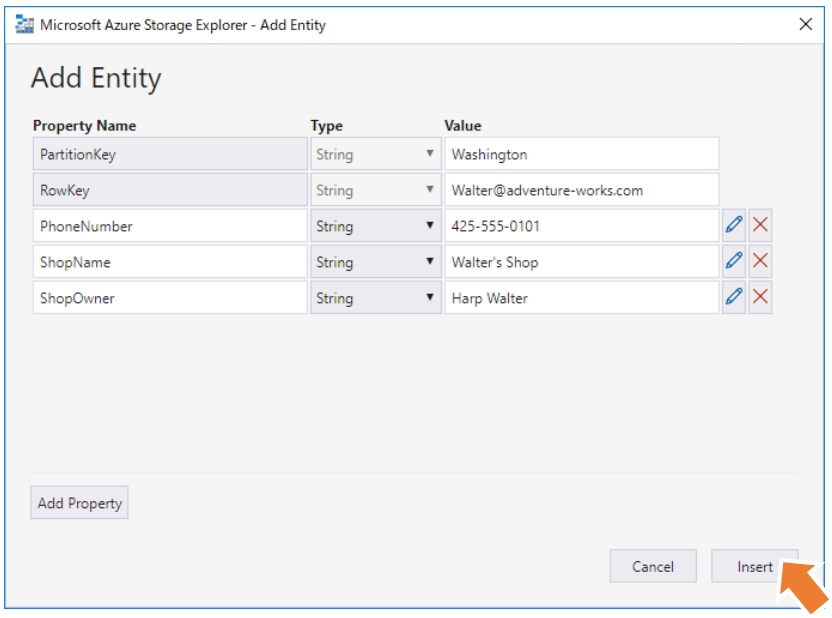


ワン ポイント
Azure Storage Table でも Azure Cosmos の DB テーブルと同様に各エンティティを一意に識別するために PartitionKey と RowKey の 2 つのシステム プロパティが使用されます。PartitionKey には州の情報を使用し、PartitionKey には E メールアドレスを使用します。

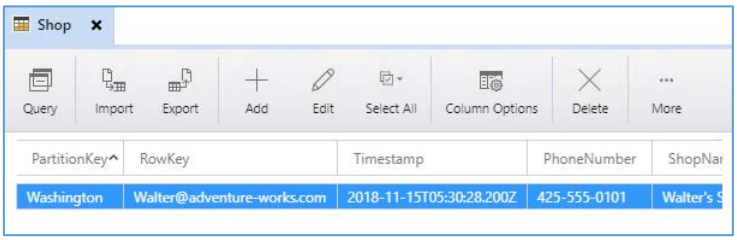
4. データの移行

23. [Add Entity] の [Add Property] ボタンをクリックしながら、下表に従い 3 つのプロパティ名と、その値を追加して、[Insert] ボタンをクリックします。

Property Name	Type	Value
PhoneNumber	String	425-555-0101
ShopName	String	Walter's Shop
ShopOwner	String	Harp Walter



24. Shop テーブルに 1 件の店舗情報が格納されたことを確認します。

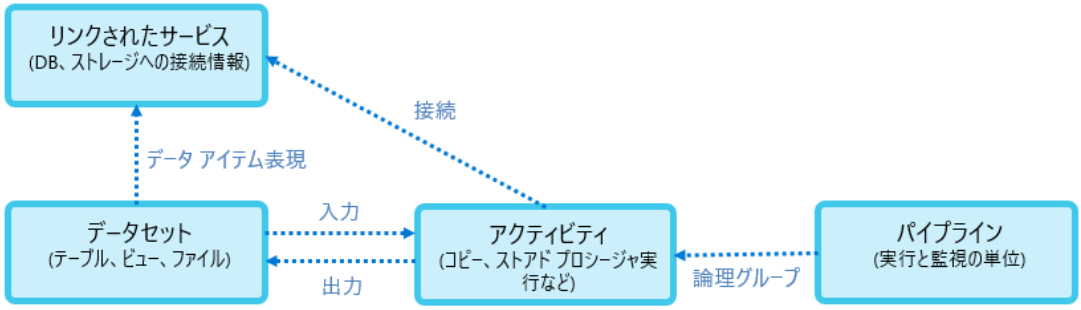


ワンポイント
ここまでの手順で、407 件の店舗情報を参照できる Azure SQL Database のビューと 1 件の店舗情報を格納した Azure Table Storage のテーブルが用意できました。以降の手順では、Azure SQL Database のビューから Azure Table Storage のテーブルにデータを転送する Azure Data Factory パイプラインを作成して実行します。

4.2.3 Azure Data Factory コピー データ ウィザードで Azure SQL Database のテーブルから Azure Table Storage のテーブルにデータをコピーする

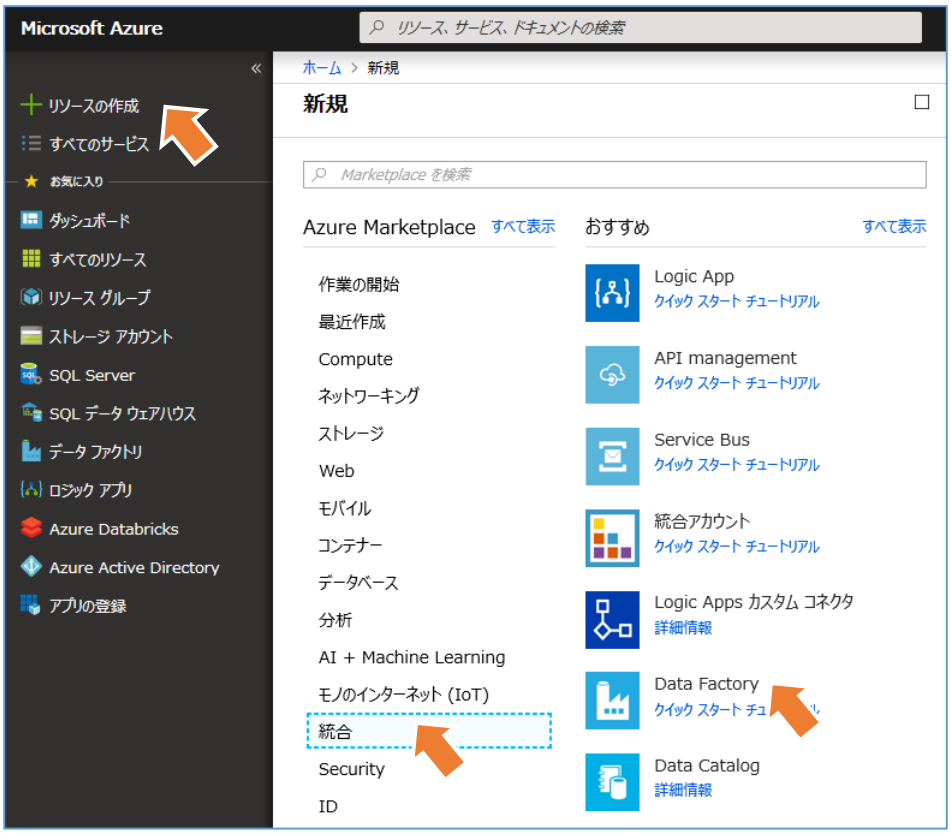
Azure Data Factory は、クラウドおよび、オンプレミスの様々なデータソースに接続できるハイブリッド データ統合 (ETL) サービスです。データがクラウドやオンプレミスのどこにあっても、エンタープライズ レベルのセキュリティを確保しながらデータを操作できます。ここでは、最初に Azure Data Factory v2 をデプロイします。コピー データ ウィザードを使用して、Azure SQL Database のビューから Azure Table Storage のテーブルにデータを転送する Azure Data Factory パイプラインを作成し、実行します。

コピー データ ウィザードを使用して、データをコピーするアクティビティを持つパイプラインを作成します。パイプラインは、アクティビティの論理的なグループです。Azure Data Factory で作成するコンポーネントには、下図の関係性があります。



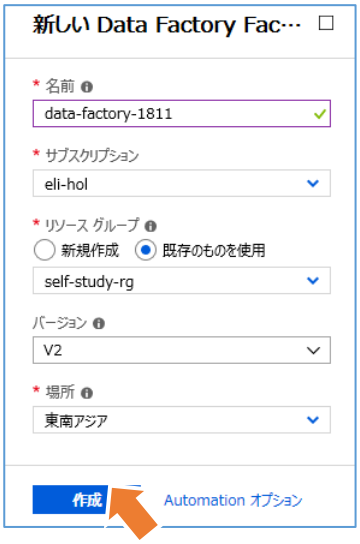
最初に「リンクされたサービス」を定義します。「リンクされたサービス」は、外部ソースに対する接続情報です。その後に「リンクされたサービス」に格納されたデータアイテムの表現として「データセット」を定義します。「データセット」は、データ ストア内のデータ構造が定義されたアイテム (テーブル、ビュー、ファイル名など) を設定します。各データセットを入出力として使用するコピー データ アクティビティ作成します。

1. Azure ポータルのハブ メニューで [リソースの作成] をクリックして [統合] ⇒ [Data Factory] を選択します。

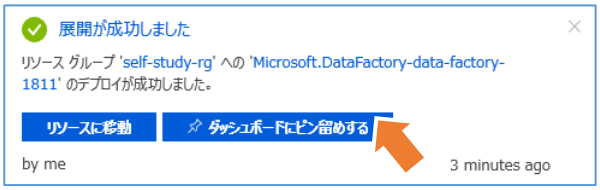


2. [新しい Data Factory] ブレードで下表に従い入力し、[作成] をクリックします。

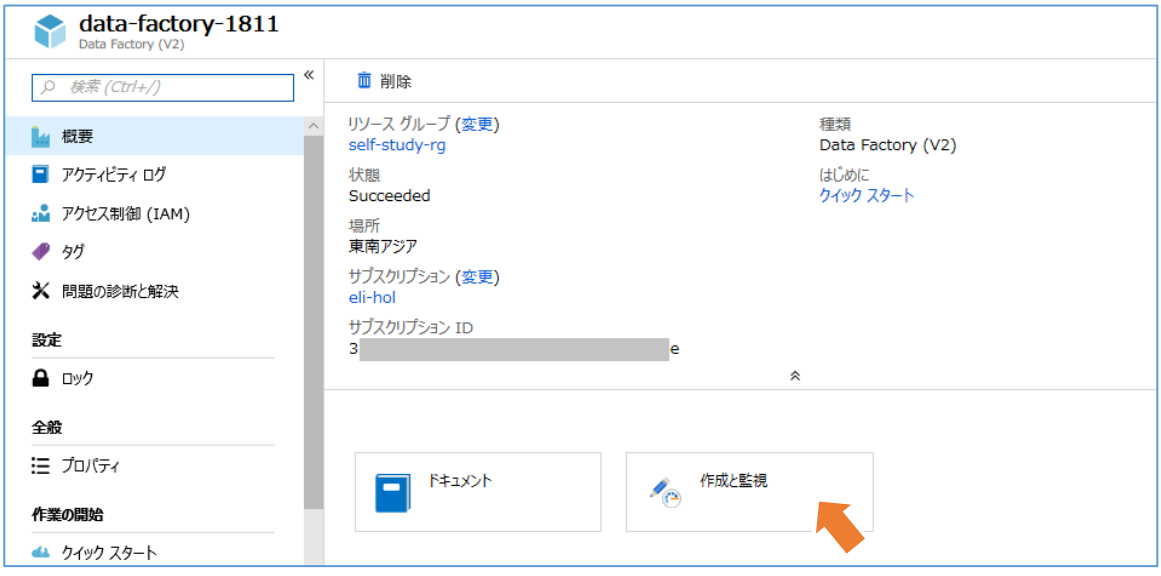
パラメーター	設定する値
名前	Data Factory の名前を入力します。この値は、グローバルで一意となる必要があります。
サブスクリプション	お持ちのサブスクリプション名を選択します。
リソース グループ	Azure Cosmos DB を作成したリソースグループを選択します。
バージョン	「V2」を選択します。
場所	Data Factory を作成する場所 (データセンターの場所) を選択します。日本リージョンは、まだ選択できないため、一番近い「東南アジア」を選択します。なお、Data Factory のデータ移動を実行するサービスは、データを保持しているリージョンで実行されます。



3. デプロイ完了後、[通知] で「展開が成功しました」と表示されるので [リソースに移動] をクリックします。

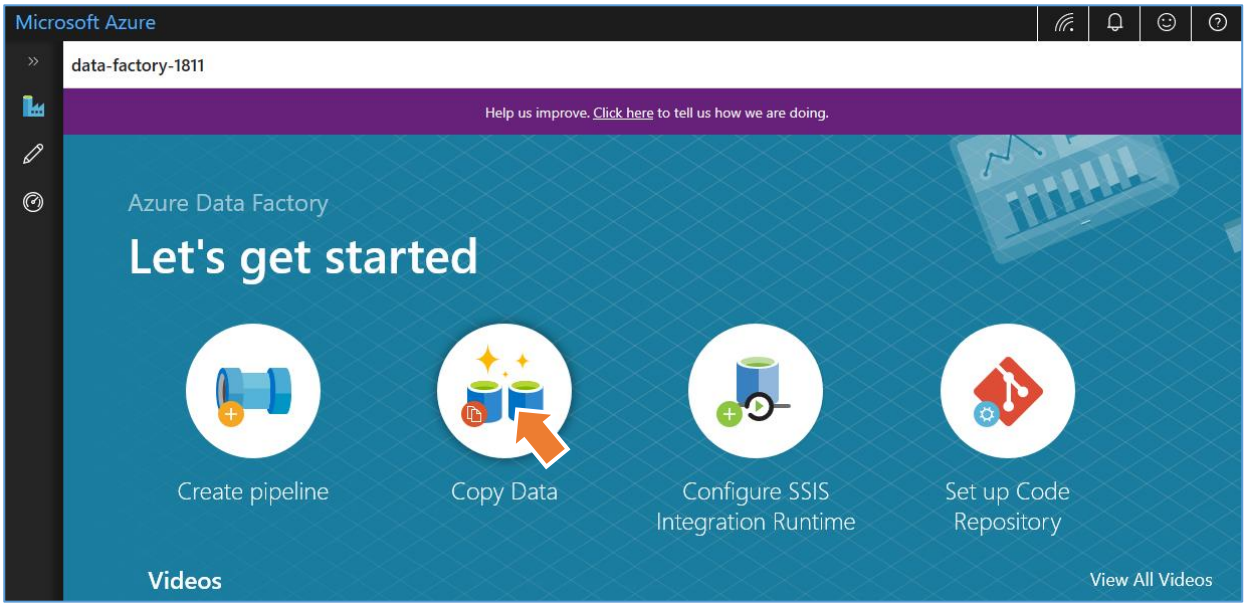


4. Azure Data Factory の [概要] ブレードが表示されるので、[作成と監視] をクリックします。



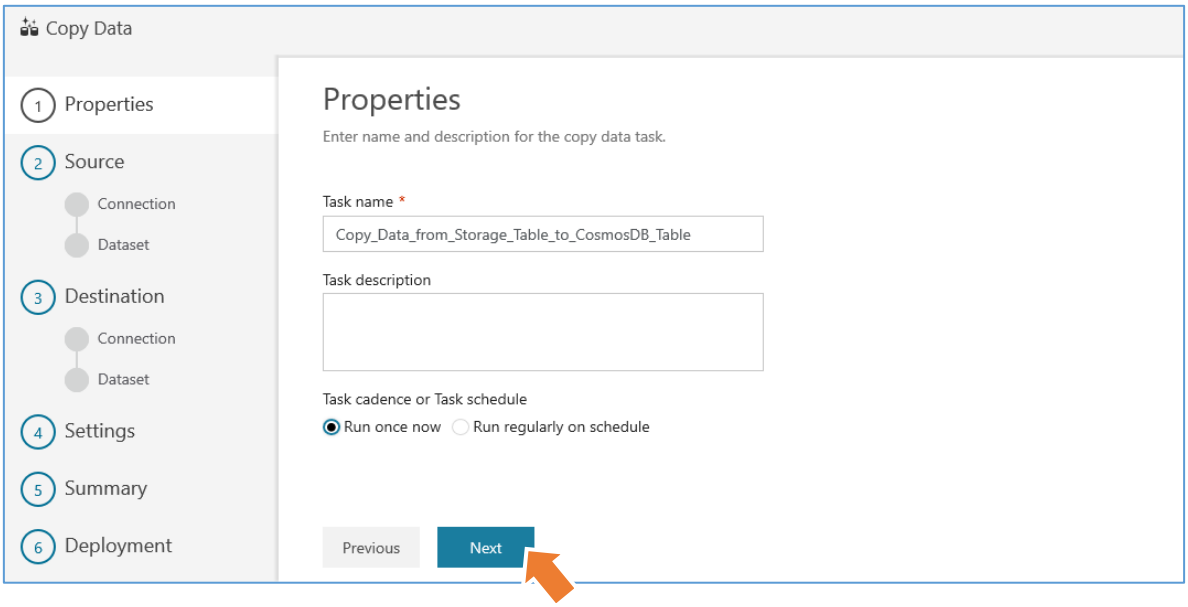
4. データの移行

5. この操作により、Azure Data Factory ユーザー インターフェイス (UI) アプリケーションが起動します。コピー ウィザードを起動するために [Let's get started] ページの [Copy Data] をクリックします。



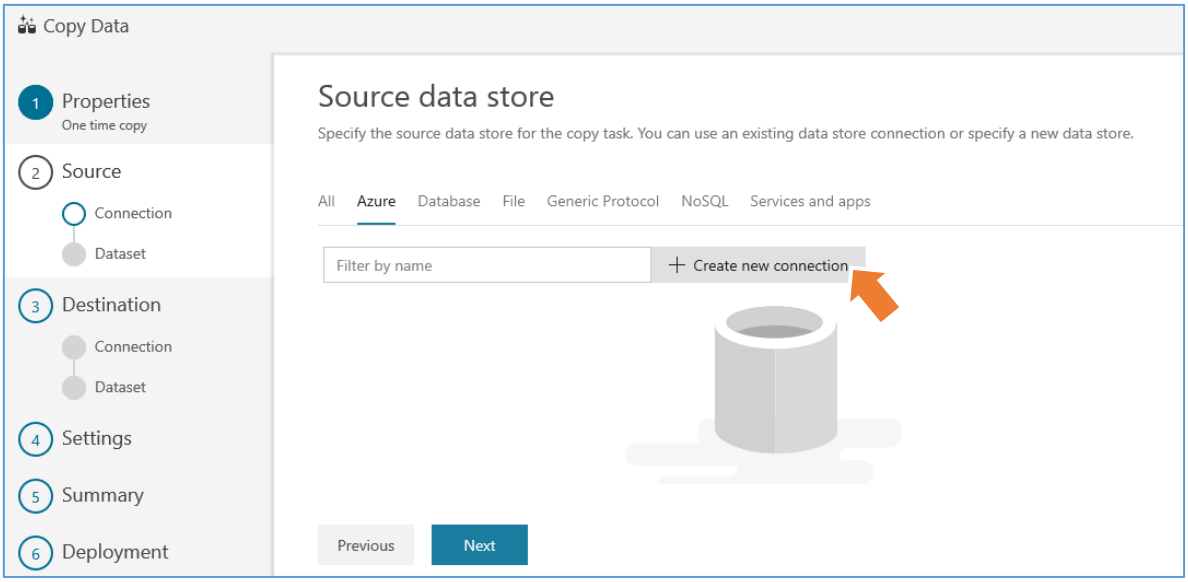
ワン ポイント
[Let's get started] ページから、コピー データ ウィザードを起動して、データソース間でデータを移行するためのパイプラインを作成することができます。パイプラインの作成は、次の手順で行います。 ● ソースのリンクされたサービスの定義 ● ソースのデータセットの定義 ● 転送先のリンクされたサービスの定義 ● 転送先のデータセットの定義 ● パイプラインの定義

6. コピー データ ウィザードが起動するので [Properties] ページの [Task name] に「Copy_Data_from_Storage_Table_to_CosmosDB_Table」と入力して [Task cadence or Task schedule] で [Run once now] が選択されていることを確認して [Next] ボタンをクリックします。

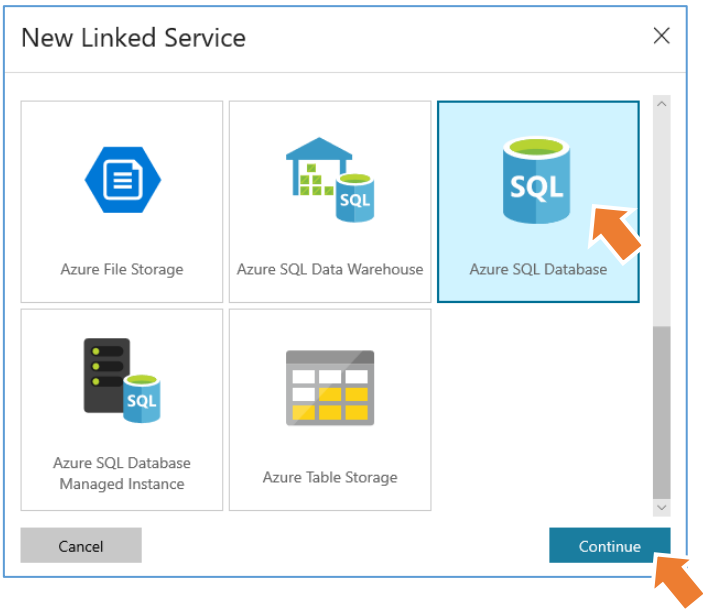


ワン ポイント
1 度だけの実行の場合、[Run once now] オプションを選択します。[Run regulary on schedule] を選択するとパイプラインが保存され、定期的に実行するためのスケジュールを構成できます。

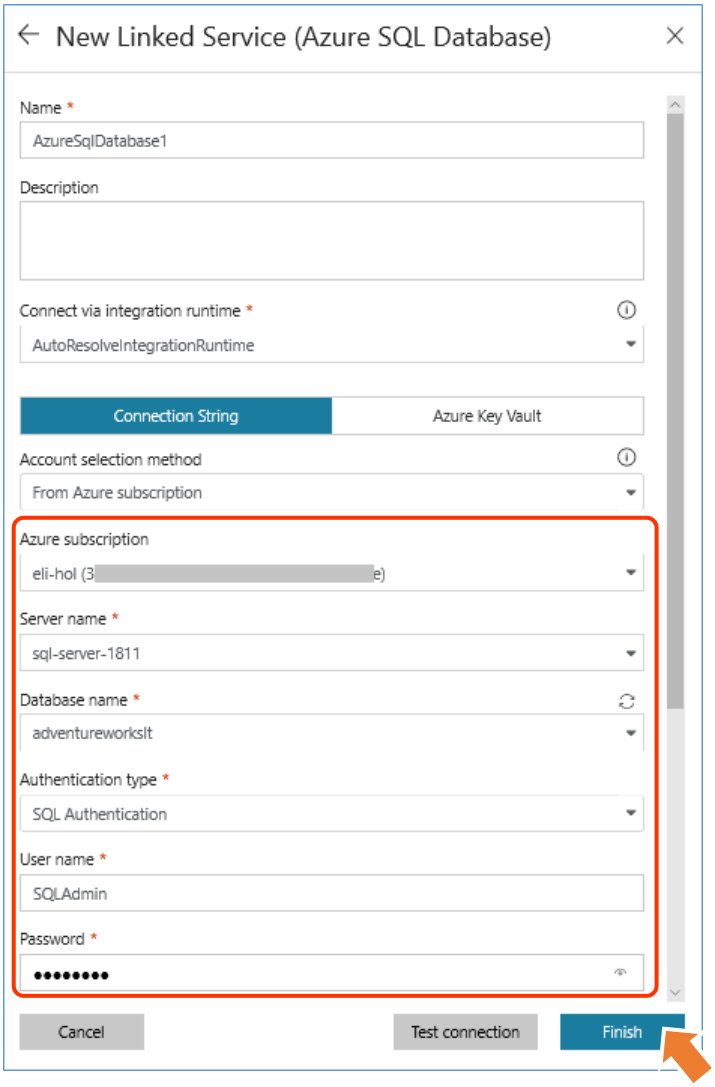
7. [Source data store] ページで、[Create new connection] をクリックします。



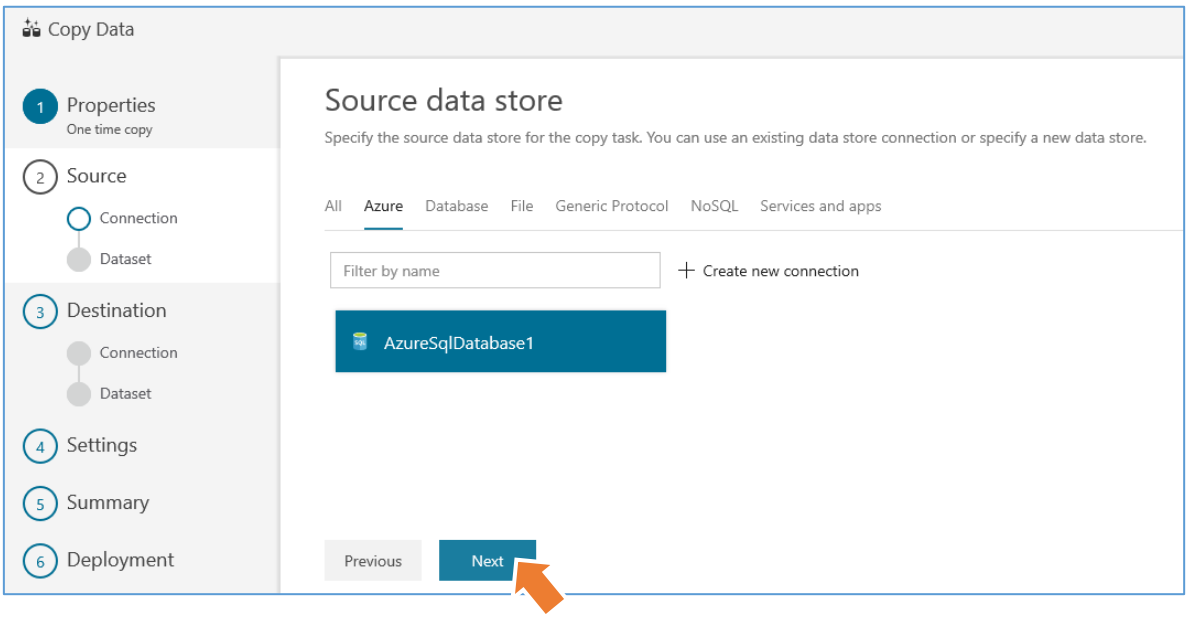
8. [New Linked Service] が表示されるので、[Azure SQL Database] を選択して、[Continue] を繰り返してクリックします。



9. [New Linked Service (Azure SQL Database)] が表示されるので、adventureworkslt データベースを作成した論理サーバーへの接続情報を設定して、[Finish] をクリックします。

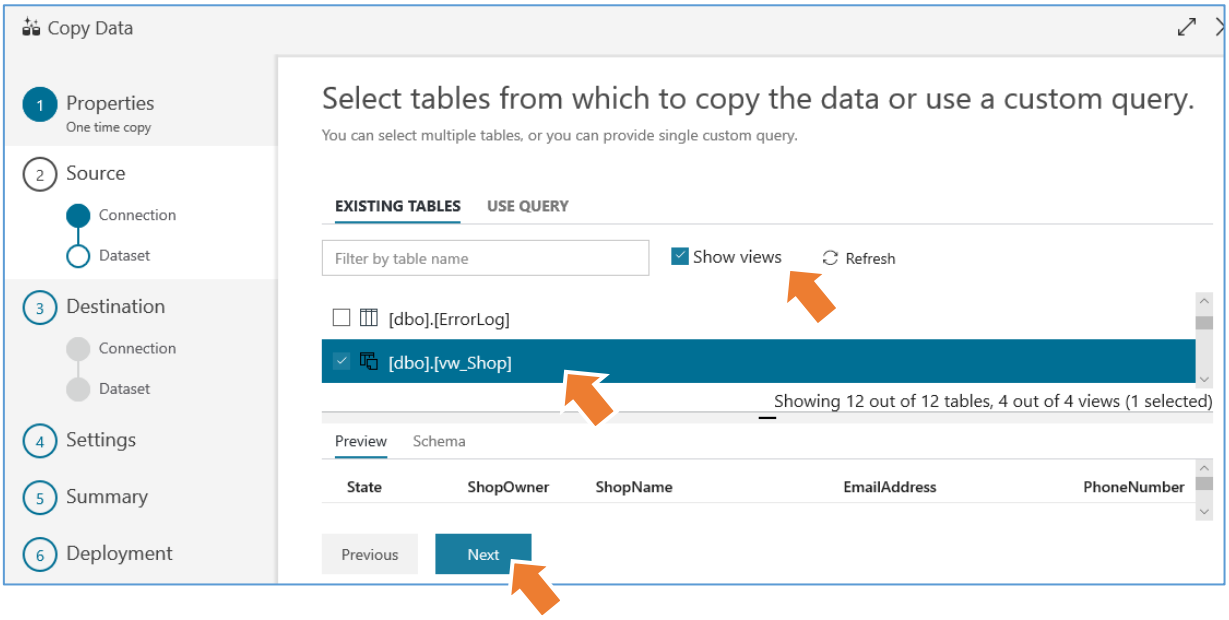


10. [Source data store] ページに戻るので [Next] をクリックします。

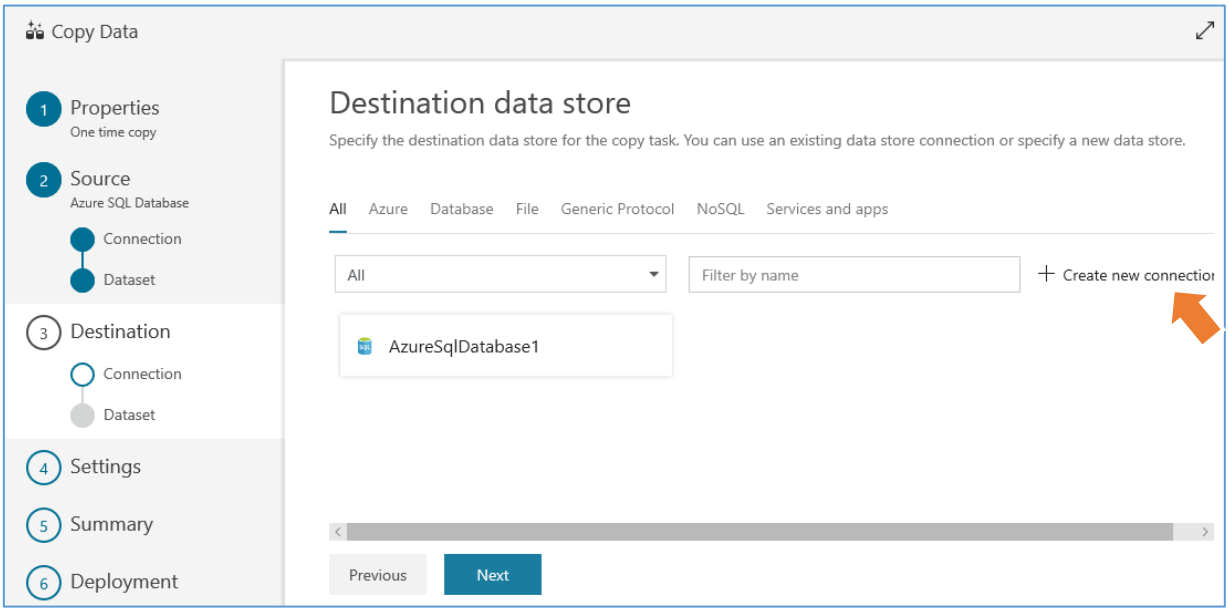


4. データの移行

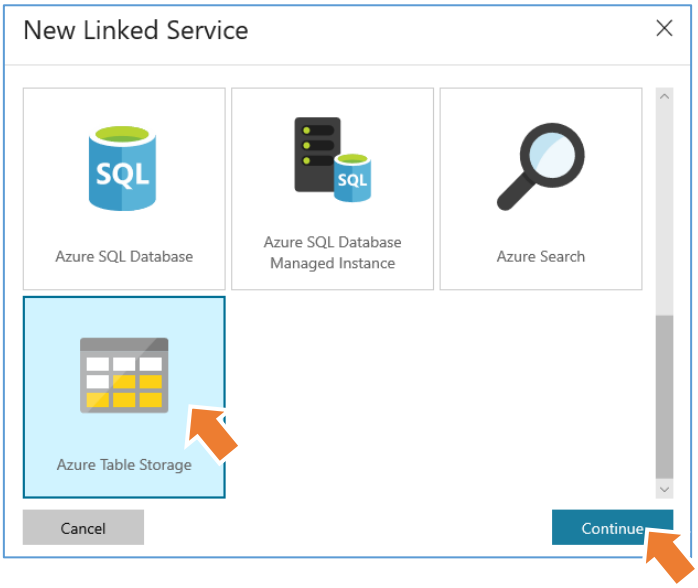
11. [Select tables from which to copy the data or use a custom query] の [EXISTING TABLES] タブで、[Show views] を選択して、テーブルとビューのリストから [dbo].[vw_Show] ビューを選択して、[Next] をクリックします。



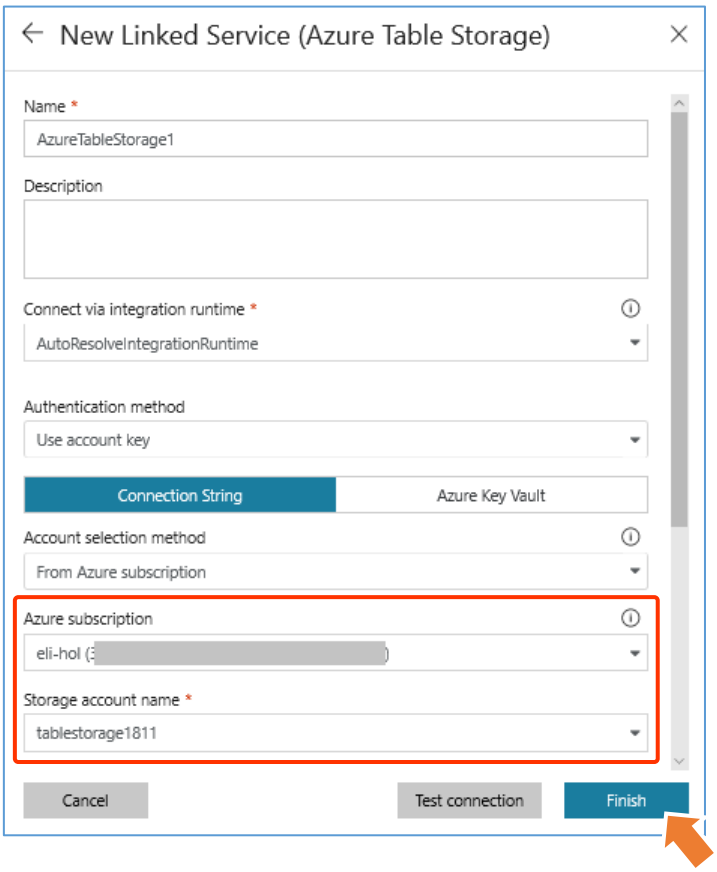
12. [Destination data store] ページで、[Create new connection] をクリックします。



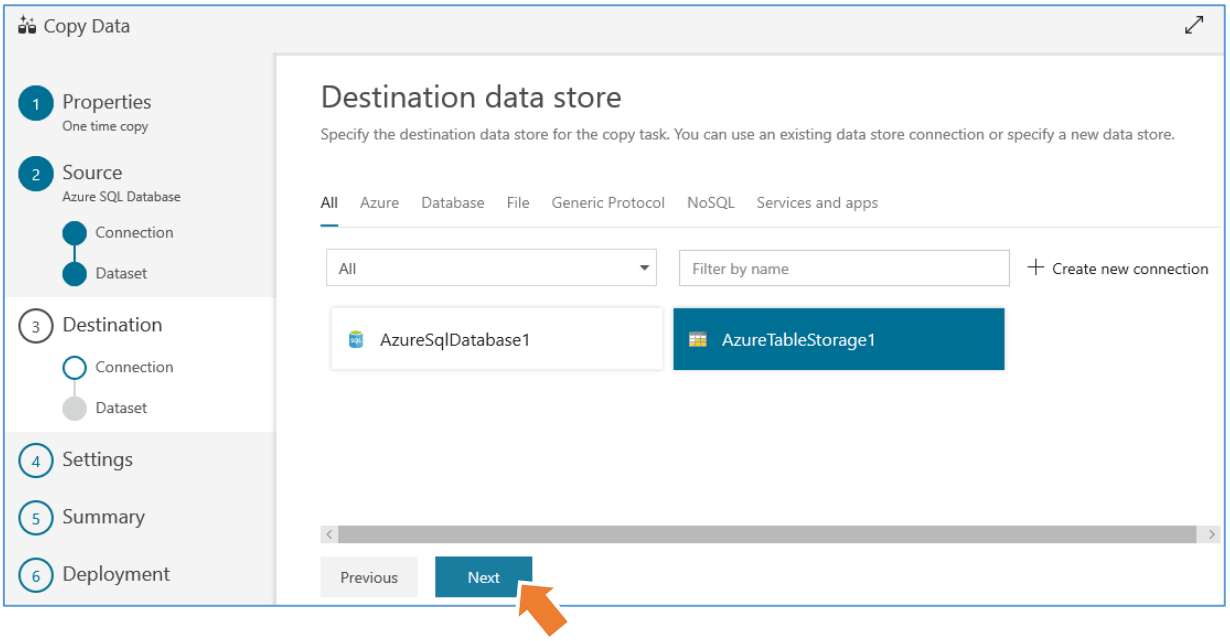
13. [New Linked Service] で、[Azure Table Storage] を選択して、[Continue] を繰り返してクリックします。



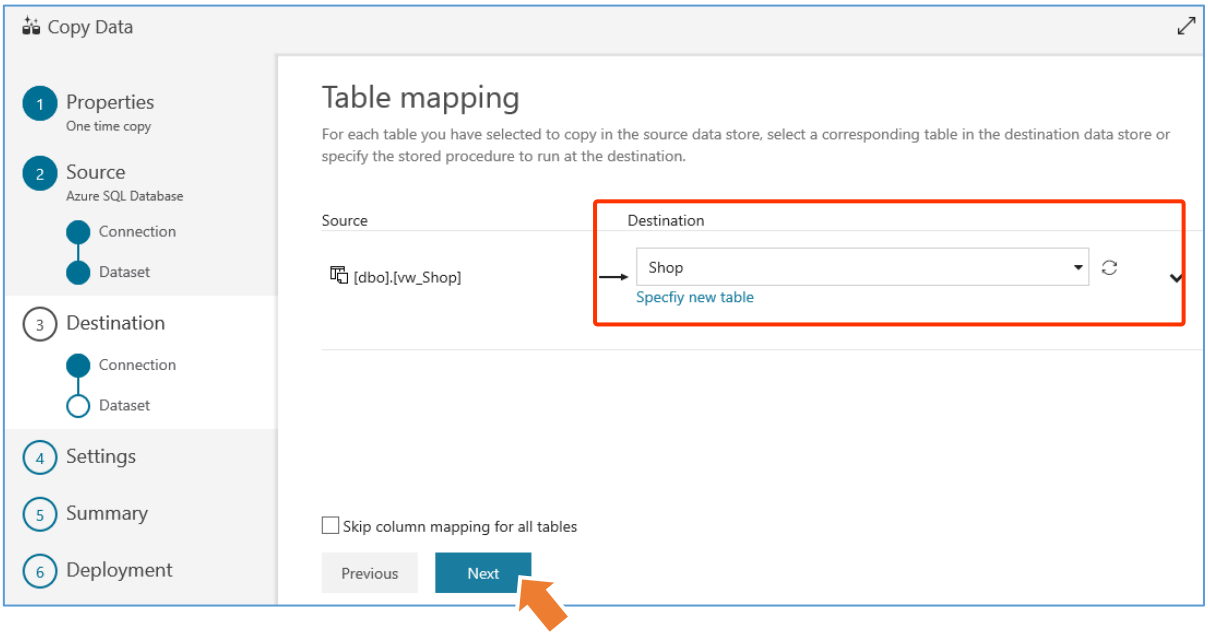
14. [New Linked Service (Azure Table Storage)] が表示されるので、Storage Explorer で Shop テーブルを作成したストレージ アカウントへの接続情報を設定して、[Finish] をクリックします。



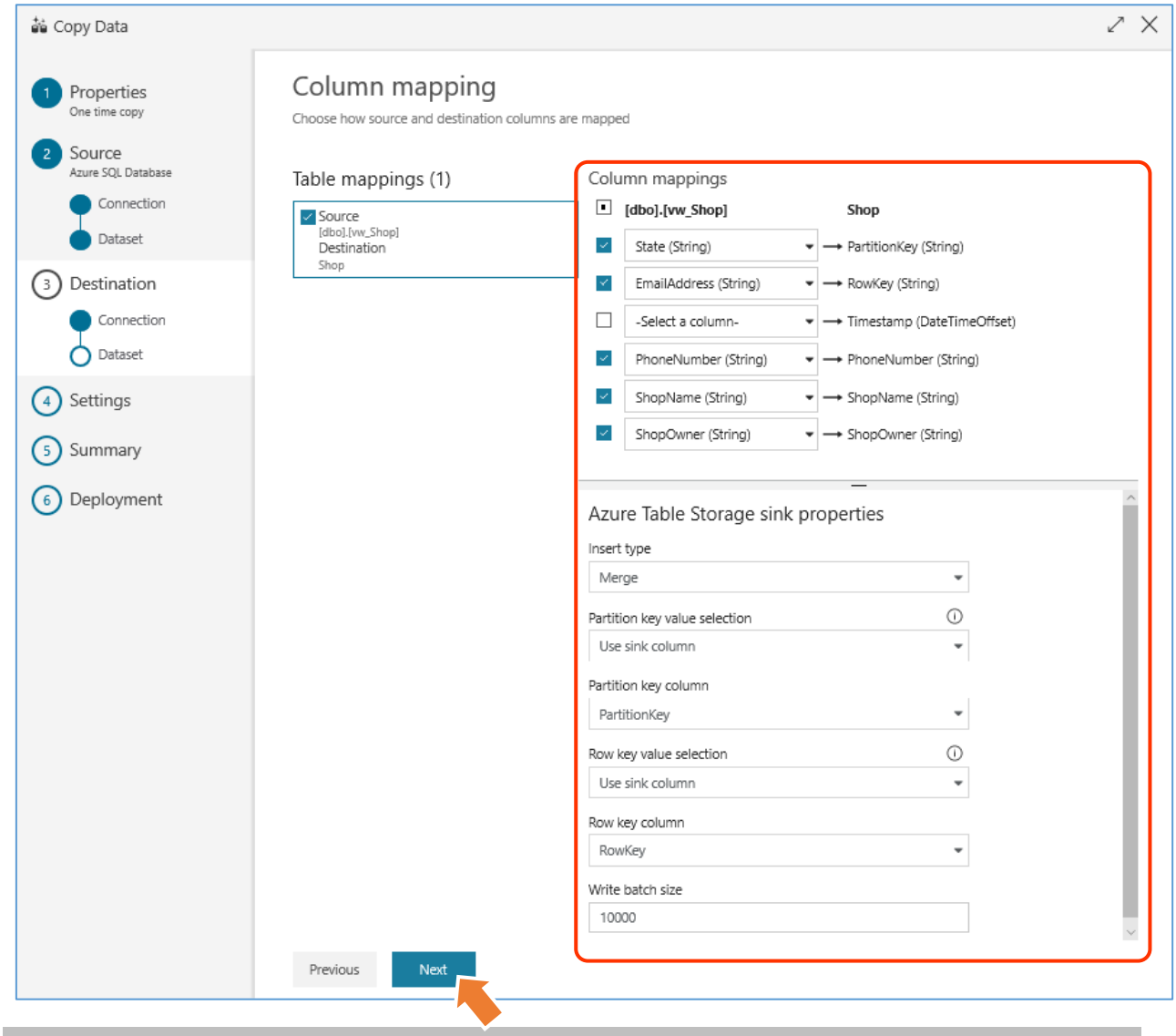
15. [Destination data store] ページで、[Next] をクリックします。



16. [Table mapping] が表示されるので、[Destination] で [Use existing table] をクリックした後、[Shop] を選択して、[Next] をクリックします。



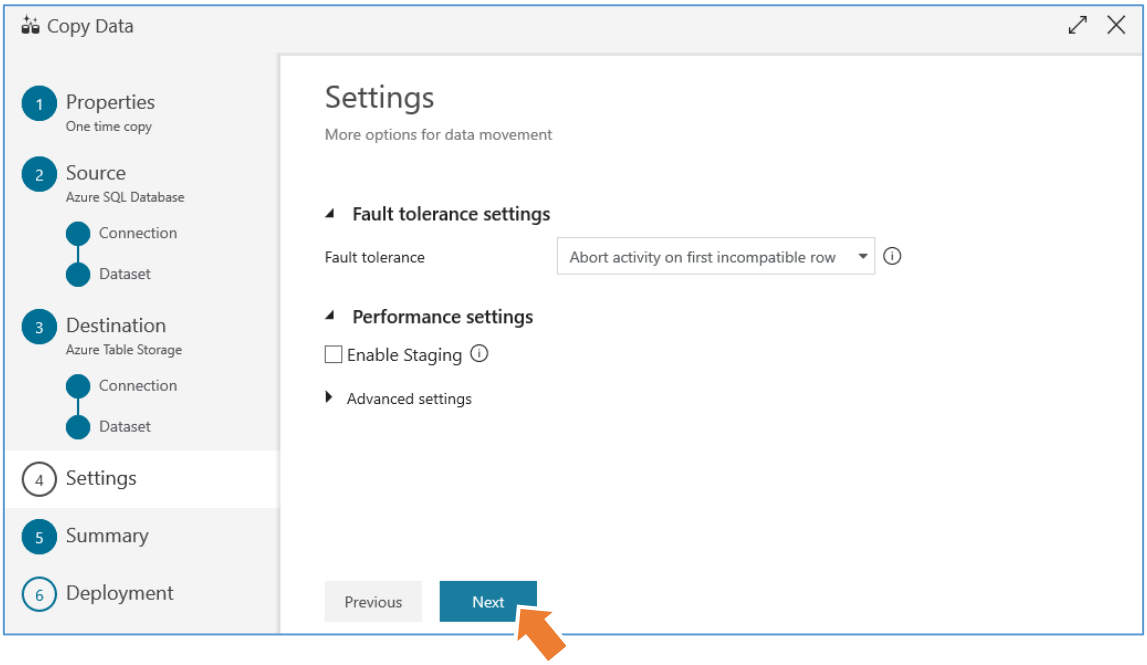
17. [Column mapping] で [dbo].[vw_Show] ビューから、[Shop] テーブルへの列マッピングを以下のように設定して、[Next] をクリックします。



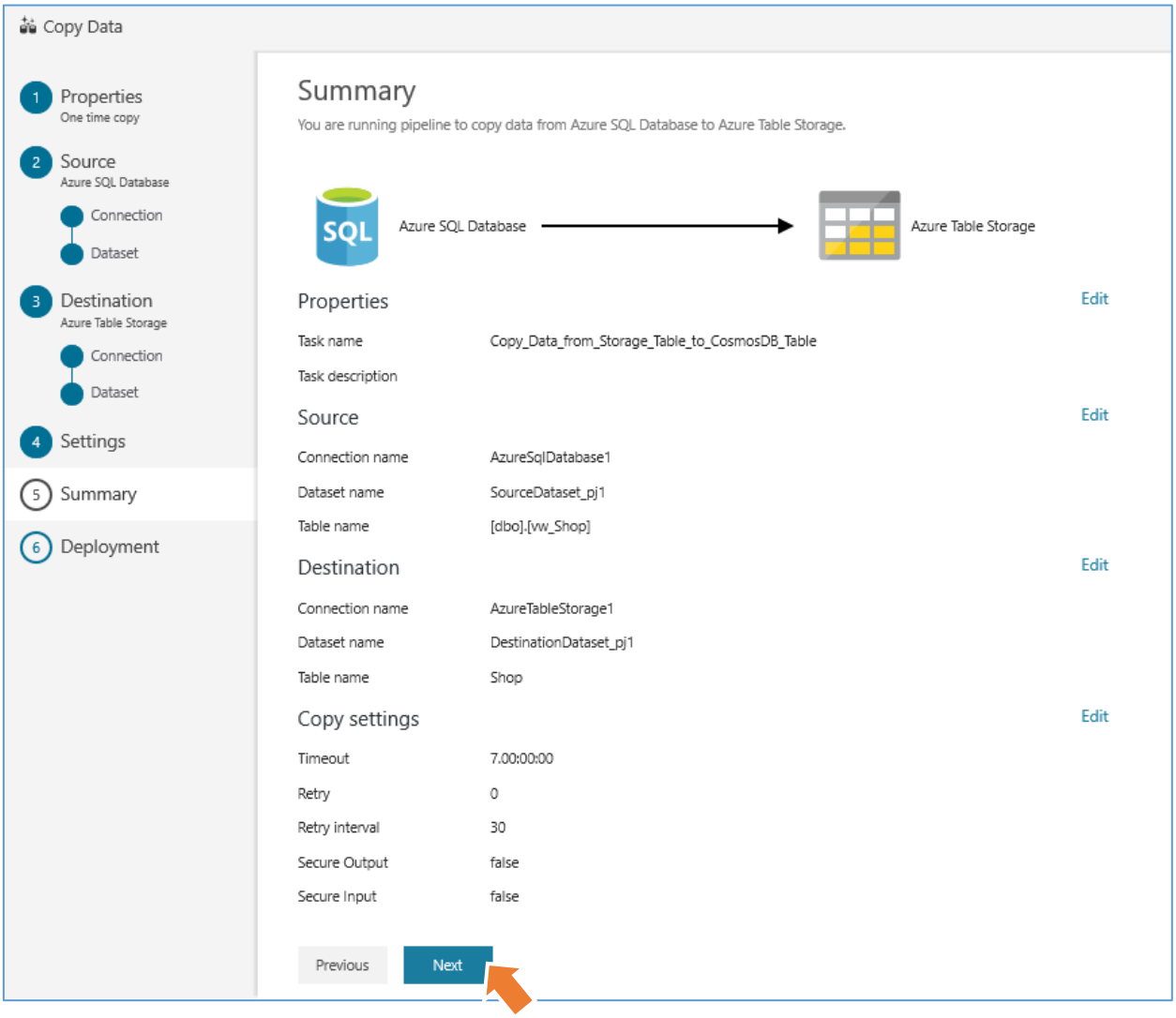
ワン ポイント

Shop テーブルの Timestamp 列の値は、自動生成されるため、列マッピングは不要です。[Azure Table Storage sink properties] では、Partition Key として、[PartitionKey] プロパティを使用し、Row Key として [RowKey] プロパティを使用することを設定します。

18. [Settings] では、既定の設定のままで、[Next] をクリックします。

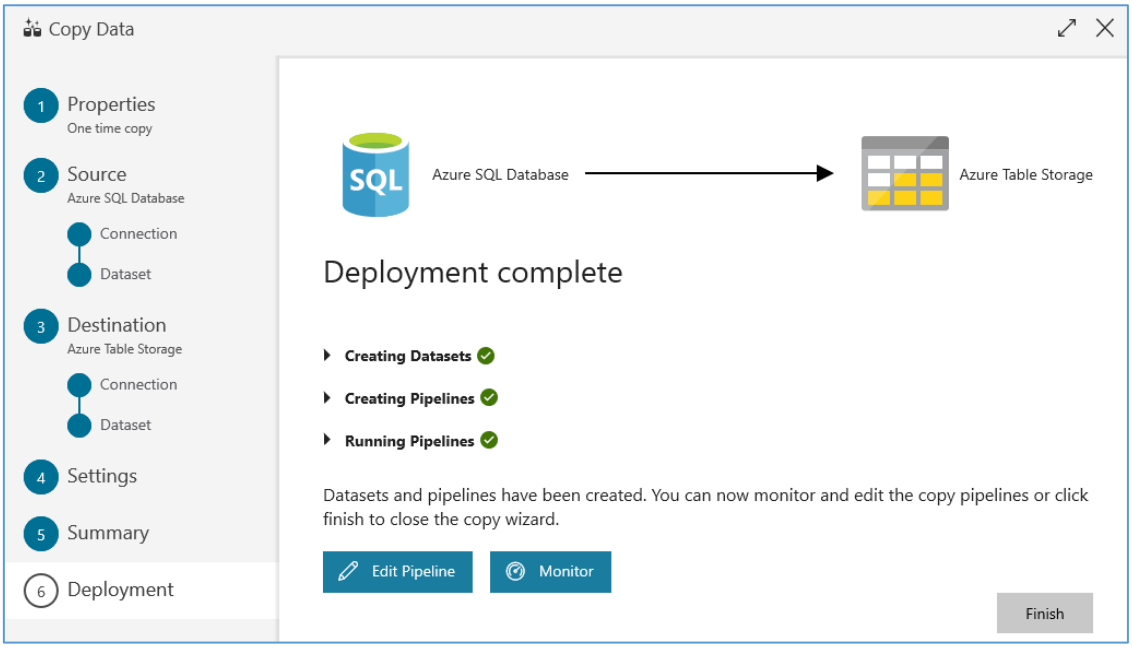


19. [Summary] で設定内容を確認して、[Next] をクリックしてデータのコピーを開始します。

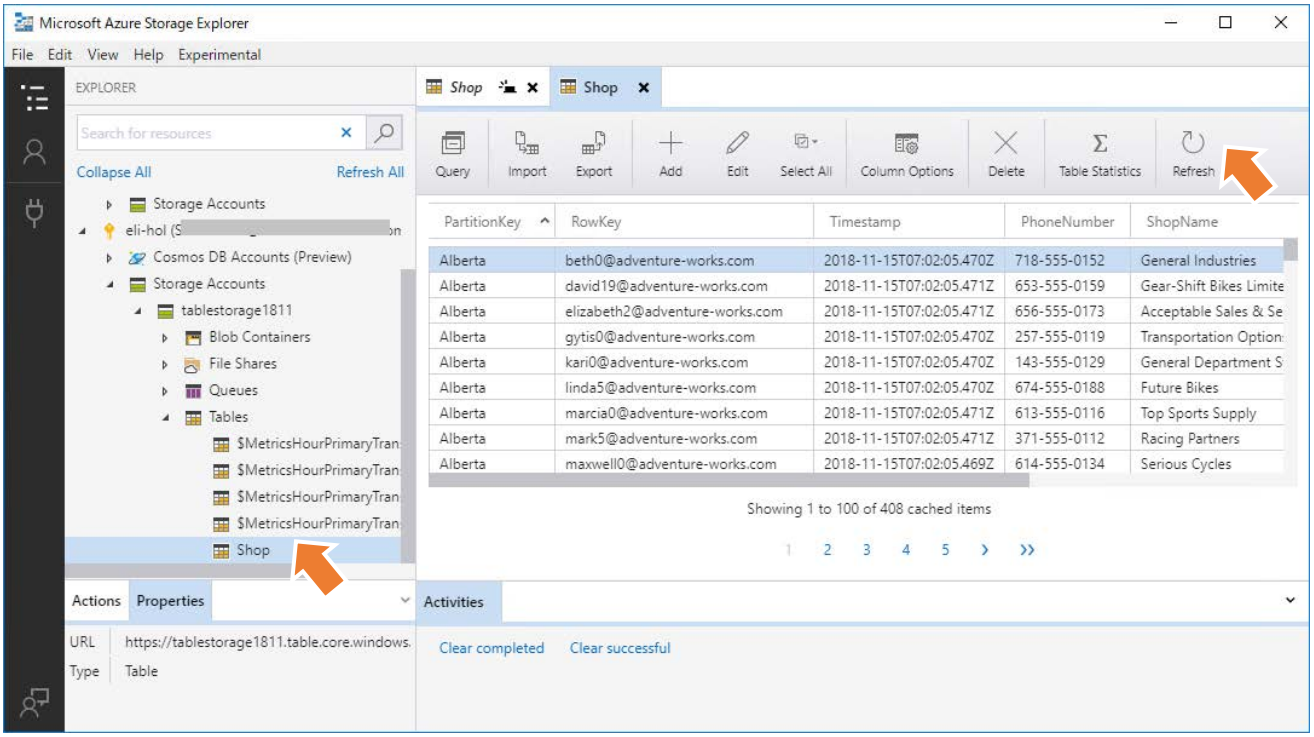


4. データの移行

20. データのコピーが正常に完了することを確認します。

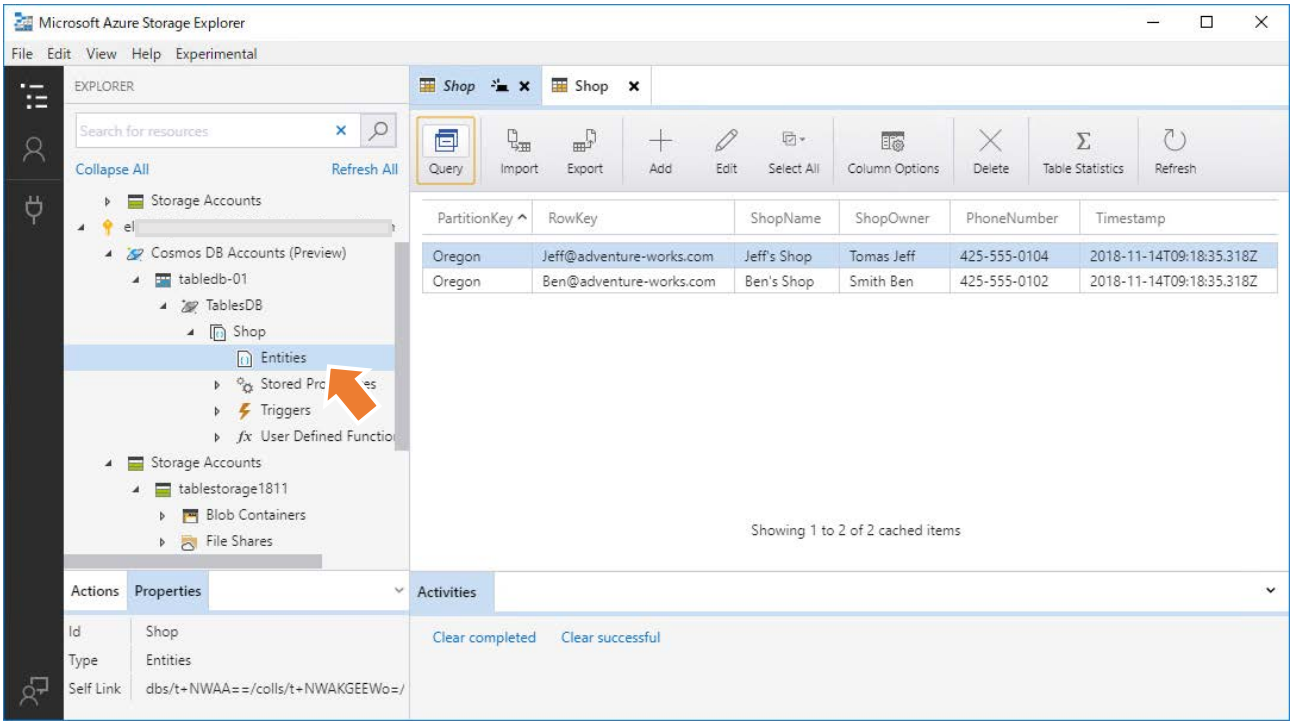


21. データ コピー完了後、Storage Explorer のツリーで、Azure Storage Table に作成した Shop テーブルを選択し、表示を更新するため、[Refresh] をクリックして、エンティティが追加されたことを確認します。



ワンポイント
ここまでの手順で、Azure SQL Database から 407 件の店舗情報が、Azure Table Storage のテーブルに転送されました。

22. Storage Explorer のツリーで、Azure Cosmos DB の Shop テーブルの [Entities] ノードをクリックして、現在、格納されているエンティティを確認します。



ワンポイント

Azure Cosmos DB の Shop テーブルの [Entities] ノードには、3 章で追加した 2 件のエンティティが格納されています。
以降の手順で Azure Table Storage のテーブルに格納したエンティティを Azure Cosmos DB の Shop テーブルに転送します。

4.2.4 データ移行ツールで Azure Table Storage のテーブルのエンティティを Azure Cosmos DB のテーブルに移行する

データ移行ツールは、「データ移行ツールを使用して Azure Cosmos DB にデータを移行する」ページの「インストール」の「プリコンパイル済みのバイナリをダウンロード」からダウンロードすることができ、Microsoft .NET Framework 4.51 以降がインストールされた環境で実行できます。dt-1.8.1.zip という名前のプリコンパイル済みのバイナリ ファイルには、次の 2 種類のツールが含まれています。

- Dtui.exe: グラフィカル インターフェイス バージョンのツール
- Dt.exe: コマンド ライン バージョンのツール

現在、UI ベースのデータ移行ツール (Dtui.exe) は、Azure Cosmos DB の Table API アカウントに対する接続をサポートしていません。したがって、Azure Table Storage から Azure Cosmos の DB テーブルへのエンティティの移行は、コマンド ライン バージョンのツール (Dt.exe) を使用してデータを移行する手順を説明します。

Azure Storage テーブルに格納したエンティティを Azure Cosmos DB の Shop テーブルに転送するために Dt.exe に指定するパラメーターは、以下のとおりです。

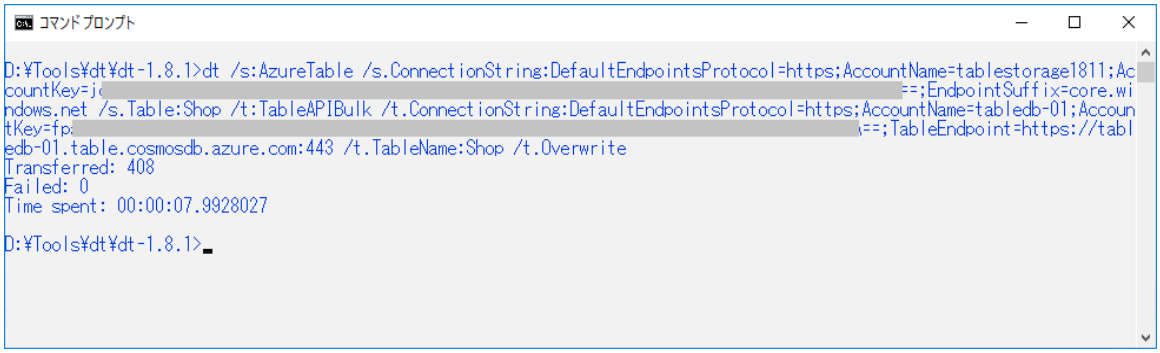
```
dt /s:AzureTable /s.ConnectionString:DefaultEndpointsProtocol=https;AccountName=<Azure Table Storage Account Name>;AccountKey=<Storage Account Key>;EndpointSuffix=core.windows.net /s.Table:<Storage Account Table Name> /t:TableAPIBulk /t.ConnectionString:DefaultEndpointsProtocol=https;AccountName=<Azure Cosmos DB Account Name>;AccountKey=<Azure Cosmos DB Account Key>;TableEndpoint=https://<Azure Cosmos DB Account Name>.table.cosmosdb.azure.com:443 /t.TableName:<Azure Cosmos DB Table Name> /t.Overwrite
```

プレースホルダ	説明	取得先	例
<Azure Table Storage Account Name>	ソースデータを格納したテーブルのストレージ アカウント名	アウトレージ アカウントの [アクセス キー] ブレード	selfstudytablestorage01
<Storage Account Key>	ソースデータを格納したテーブルのストレージ アカウント キー	アウトレージ アカウントの [アクセス キー] ブレード	自動生成される 512 ビットのキーを使用します
<Storage Account Table Name>	ソースデータを格納したテーブル名	アウトレージ アカウントの [テーブル] ブレード	shop
<Azure Cosmos DB Account Name>	移行先の Azure Cosmos DB アカウント名	Azure Cosmos DB アカウントの [Connection String] ブレード	table-db-01
<Azure Cosmos DB Account Key>	移行先の Azure Cosmos DB アカウント キー	Azure Cosmos DB アカウントの [Connection String] ブレード	自動生成される 512 ビットのキーを使用します
<Azure Cosmos DB Table Name>	移行先の Azure Cosmos DB のテーブル名	Azure Cosmos DB アカウントの Data Explorer	Shop

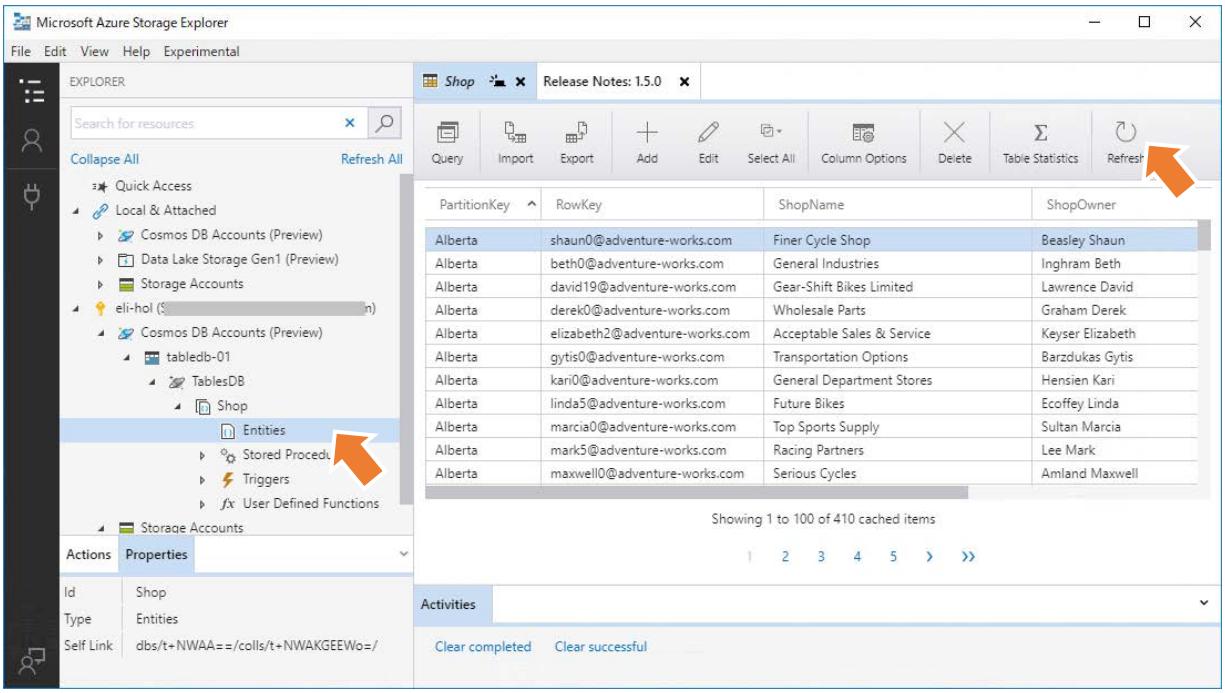
4. データの移行

1. コマンド プロンプトを起動して dt.exe を展開したフォルダー パスに移動します。

2. Azure Table Storage のテーブルから Azure Cosmos の DB テーブルにエンティティをコピーするために、上記のコマンドの引数の説明を確認しながら、ご自身で作成した、リソースの情報に置き換えて、入力して実行します。



3. データ コピー完了後に、Storage Explorer のツリーで、Azure Cosmos DB の Shop テーブルの下の [Entities] ノードを選択して、表示を更新するため、[Refresh] をクリックして、エンティティが追加されたことを確認します。

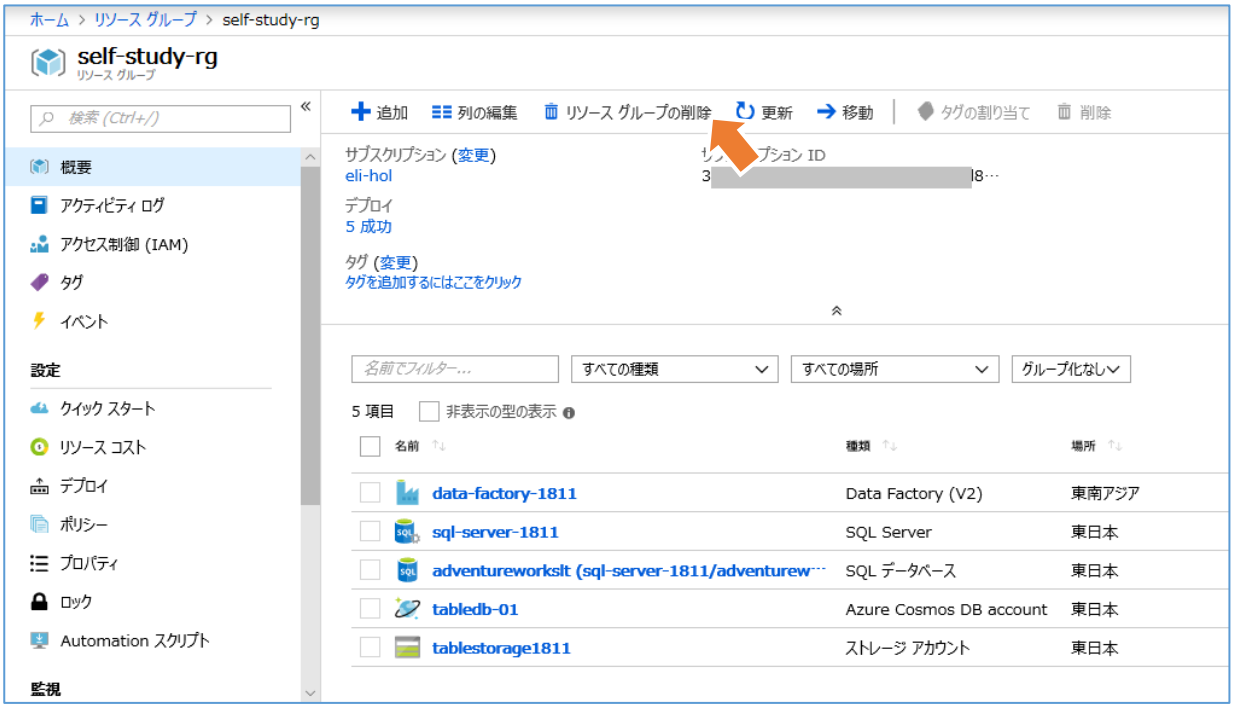


4.3 リソースの削除手順

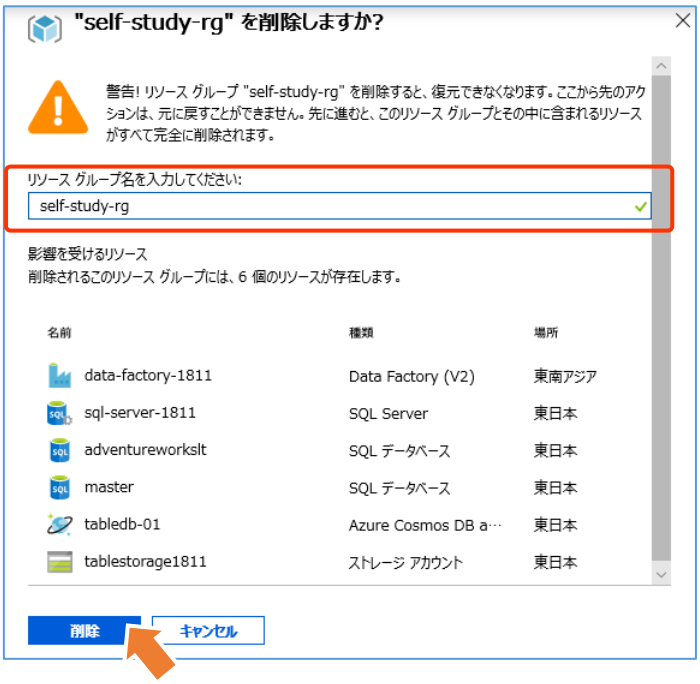
自習書の手順に従って作成したリソースを残しておくと課金が継続することに注意してください。最後の手順では、作成したリソース グループを削除します。個々のリソースを個別に削除することもできますが、リソース グループを削除することで、そのリソース グループ内のすべてのリソースをまとめて削除できます。なお、検証を続けたい場合は、検証の完了後に行ってください。

1. Azure ポータルのハブ メニューから [リソース グループ] をクリックして、表示されるリソース グループのリストから自習書用に作成したリソース グループをクリックします。

2. 自習書用に作成したリソース グループのブレードで [リソース グループの削除] をクリックします。



3. 次のメッセージが表示されます。ここで [リソース グループ名を入力してください] に削除するリソース グループ名を入力して、[削除] ボタンをクリックします。



ワン ポイント
自習書用に作成したリソース グループを削除すると、その中にあるすべてのリソースが削除され、元に戻すことはできないことに注意してください。

参考文献

本書の執筆で参考にした文献、サイトは以下のとおりです。自習書の復習と今後の学習の際に参考にしてください。

Azure Cosmos DB のドキュメント

Azure Cosmos DB の機能全般を説明しているドキュメントです。このサイトで提供される様々な API とプログラミング モデルに対応するチュートリアルとサンプルは、Azure Cosmos DB を使用したソリューションの構築に役立つでしょう。

Azure データ アーキテクチャ ガイド

このガイドでは、Microsoft Azure でデータ中心のソリューションを設計するための体系的なアプローチについて説明しています。