# Module 6: Storing Tabular Data in Azure

# Lab: Storing Event Registration Data in Azure Storage Tables

### Scenario

Even though event registrations could be stored in SQL, you have a unique need. Each event requires a different registration form that can be changed at any time. Essentially, registrations could be of any schema. A relational database such as SQL requires a well-defined schema. Because of your business requirement, you require a database that can store items with flexible structures (or schemas). To facilitate this you have elected to use Azure Table Storage for your event registrations.

### Objectives

After you complete this lab, you will be able to:

* Use the Azure Store SDK to create a Table storage client.
* Use the Azure Store SDK to create a table.
* Use the Azure Store SDK to add entities to a table.
* Use the Azure Store SDK to query entities.
* Use Cloud Explorer in Visual Studio 2017 to view Table storage entities.

### Lab Setup

**Estimated Time: 45 minutes**

Before starting this lab, you must complete the lab in Module 2. For the lab in this module, you will use the available host machine. Also, you must complete the following steps:

1. On the host computer, click **Start**, type **Remote**, and then click **Remote Desktop Connection**.
2. In Remote Desktop Connection, provide the name of your virtual machine in the **Computer** box by using the following format:
   * **[Your VM IP Address]:[*Your VM RDP Port*]**

* **Note:** The name and port for your virtual machine might be saved in the Computer drop-down list. If this is the case, use this value instead of typing it in manually. If you are unsure about your virtual machine’s RDP port, use either of the Azure portals to find your virtual machine’s endpoints. The endpoint with the name **Remote Desktop** is the correct port for RDP. This port is randomized to protect your virtual machine from unauthorized access.

1. In Remote Desktop Connection, click **Connect**. Wait until the RDP client accesses the virtual machine.
2. If necessary, sign in by using the following credentials:
   * User name: **Student**
   * Password: **AzurePa$$w0rd**
3. Verify that you received the credentials to sign in to the Azure portal from your training provider. You will use these credentials and the Azure account throughout the labs in this course.

## Exercise 1: Populating the Sign-In Form with Registrant Names

### Scenario

In this exercise, you will:

* Query table entities by using a **CloudTableClient** instance.
* Use LINQ-to-Objects on a set of entities.

The main tasks for this exercise are as follows:

1. Create an instance of the CloudTable class.
2. Retrieve strongly-typed registration records by partition key.
3. Use LINQ-to-Objects to project a list of registrant names.

#### Task 1: Create an instance of the CloudTable class

1. Open the **Contoso.Events** solution from the following location:
2. File location: **Allfiles (F):\Mod06\Labfiles\Starter\Contoso.Events**
3. In the **Contoso.Events.Worker** project, open the **TableStorageHelper.cs** file.
4. From the **GetRegistrantNames** method, remove the existing code.
5. Using the **\_tableClient** variable, create a new **CloudTable** instance by using the **GetTableReference** method with the following parameters:
   * tableName: **EventRegistrations**

* CloudTable table = \_tableClient.GetTableReference("EventRegistrations");

#### Task 2: Retrieve strongly-typed registration records by partition key

1. By using the following code, create a **string** filter that finds entities with a partition key equivalent to the **eventKey** provided by the **GetRegistrantsNames** method’s parameters.

* string partitionKey = eventKey;   
  string filter = TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, partitionKey);

1. Create a **TableQuery** with the filter and run the query by using the **ExecuteQuery** method of the *table* variable as shown in the following code:

* TableQuery<Registration> query = new TableQuery<Registration>().Where(filter);   
  IEnumerable<Registration> registrations = table.ExecuteQuery<Registration>(query);

#### Task 3: Use LINQ-to-Objects to project a list of registrant names

1. Create a LINQ query that takes the registrations, orders them by **LastName**, then orders them by **FirstName**, and then projects them into a string of the format **“{LastName}, {FirstName}”** by using the following code:

* IEnumerable<string> names = registrations  
   .OrderBy(r => r.LastName)  
   .ThenBy(r => r.FirstName)  
   .Select(r => String.Format("{1}, {0}", r.FirstName, r.LastName));
* **Note:** You can alternatively use the query syntax for LINQ-to-Objects, as shown below:
* IEnumerable names = from r in registrations  
   orderby r.LastName, r.FirstName  
   select r.FirstName + ", " + r.LastName

1. Return the LINQ query as the result of the **GetRegistrantsNames** method.

**Results:** After completing this exercise, you will have queried entities by row key or partition key from Table storage.

## Exercise 2: Updating the Events Website to use Storage Tables

### Scenario

In this exercise, you will:

* Update the existing Contoso.Events application to use Table storage.

The main tasks for this exercise are as follows:

1. Update the register controller action to store the registration record.
2. Update the register ViewModel to retrieve the dynamic stub registration from the table.

#### Task 1: Update the register controller action to store the registration record

1. In the **Contoso.Events.Web** project, open the **RegisterController.cs** file.
2. Remove the existing code from the **StoreRegistration** method.
3. Create a **CloudStorageAccount** variable by using the **Microsoft.WindowsAzure.Storage.ConnectionString** stored in the Cloud configuration files as shown in the following code:

* string connectionString = ConfigurationManager.AppSettings["Microsoft.WindowsAzure.Storage.ConnectionString"];  
  var storageAccount = Microsoft.WindowsAzure.Storage.CloudStorageAccount.Parse(connectionString);

1. Create a *CloudTable* variable by using the storage account and a Storage table named **EventRegistrations** as shown in the following code:

* var tableClient = storageAccount.CreateCloudTableClient();   
  var table = tableClient.GetTableReference("EventRegistrations");

1. Create and run a **TableOperation** to insert the dynamic registration object into the Storage table by using the following code:

* var operation = TableOperation.Insert(registration);   
  table.Execute(operation);

1. Parse the **RowKey** of the registration as a **System.Guid** and return the value as the result of the **StoreRegistration** method by using the following code:

* Guid rowKey = Guid.Parse(registration.RowKey);   
  return rowKey;

#### Task 2: Update the register ViewModel to retrieve the dynamic stub registration from the table

1. In the **Contoso.Events.ViewModels** project, open the **RegisterViewModel.cs** file.
2. Locate the **RegisterViewModel** constructor that has a single **string** parameter.
3. Create a *CloudStorageAccount* variable by using the **Microsoft.WindowsAzure.Storage.ConnectionString** stored in the Cloud configuration files as shown in the following code:

* string connectionString = ConfigurationManager.AppSettings["Microsoft.WindowsAzure.Storage.ConnectionString"];  
  var storageAccount = Microsoft.WindowsAzure.Storage.CloudStorageAccount.Parse(connectionString);

1. Create a *CloudTable* variable by using the storage account, and a Storage table named **EventRegistrations** as shown in the following code:

* var tableClient = storageAccount.CreateCloudTableClient();   
  var table = tableClient.GetTableReference("EventRegistrations");

1. Create a **string** filter that finds entities with a **PartitionKey** equivalent to the **eventKey** provided by the **GetRegistrantsNames** method’s parameters as shown in the following code:

* string partitionKey = String.Format("Stub\_{0}", eventKey);   
  string filter = TableQuery.GenerateFilterCondition("PartitionKey",QueryComparisons.Equal, partitionKey);

1. Create a **TableQuery** with the filter and run the query by using the **ExecuteQuery** method of the *table* variable, as shown in the following code:

* TableQuery query = new TableQuery().Where(filter);   
  IEnumerable<DynamicTableEntity> tableEntities = table.ExecuteQuery(query);

1. Because the query returns only a single entity, use the **SingleOrDefault** extension method to get the *DynamicTableEntity* variable, and then pass that variable into the **DynamicEntity.GenerateDynamicItem** static method. Set the **RegisterViewModel**’s **RegistrationStub** property to the result of the method call by using the following code:

* DynamicTableEntity tableEntity = tableEntities.SingleOrDefault();   
  this.RegistrationStub = DynamicEntity.GenerateDynamicItem(tableEntity);

**Results:** After completing this exercise, you will have used the Azure Storage SDK to retrieve and create entities.

## Exercise 3: Verifying that the Events Website is Using Azure Storage Tables for Registrations

### Scenario

In this exercise, you will:

* Use an Azure Storage account to test storage entities in development.
* Use Visual Studio to debug an application that uses development storage.

The main tasks for this exercise are as follows:

1. Create a storage account instance.
2. Run the data generation console project to populate the Azure storage table with data.
3. Use Cloud Explorer in Visual Studio 2017 to view table storage registrations.
4. Debug the cloud web project to register for the event.
5. Use Cloud Explorer in Visual Studio 2017 to view the new table storage registration.

#### Task 1: Create a Storage Account Instance

1. Sign in to the Azure Portal (<https://portal.azure.com>).
2. Create a Storage Account using the following details:
   * Name: **provide a globally unique value**
   * Replication: **Locally Redundant Storage (LRS)**
   * Resource Group: **20532**
   * Location: **Select the region that is closest to your location**
3. Record the *account name* for your newly created Storage Account instance.
4. Select an access key and record the *connection string* for that key.

#### Task 2: Run the data generation console project to populate the Azure storage table with data

1. Locate the **Contoso.Events.Data.Generation** project.
2. Update the **app.config** file in the project to use your *Storage Account*'s connection string in the **StorageConnectionString** *AppSetting*.
3. Debug the **Contoso.Events.Data.Generation** project to generate the SQL and storage tables data.

#### Task 3: Use Cloud Explorer in Visual Studio 2017 to view table storage registrations

1. In the **Cloud Explorer** pane, open the **Storage Accounts** node.
2. Locate your newly created **Storage Account**.
3. In the node associated with your Storage Account, open the **EventRegistrations** table.
4. Explore the table entities that are auto generated by the console project.

#### Task 4: Debug the web and worker projects to register for the event

1. Update your solution configuration to start both the **Contoso.Events.Web** and **Contoso.Events.Worker** projects when the solution is debugged.
2. Update the **web.config** file in the **Contoso.Events.Web** project to use your *Storage Account*'s connection string in the **Microsoft.WindowsAzure.Storage.ConnectionString** *AppSetting*.
3. Update the **app.config** file in the **Contoso.Events.Worker** project to use your *Storage Account*'s connection string in the following locations:
   * **StorageConnectionString** *AppSetting*
   * **AzureWebJobsStorage** *ConnectionString*
   * **AzureWebJobsDashboard** *ConnectionString*
4. Debug the **Contoso.Events.Web** and **Contoso.Events.Worker** projects.
5. Register for an event by using the website.

#### Task 5: Use Cloud Explorer in Visual Studio 2017 to view the new table storage registration

1. Switch to the **Contoso.Events – Microsoft Visual Studio** window
2. Open the **Cloud Explorer** pane.
3. Open the **EventRegistrations** table again in the Azure Storage Account.
4. Locate your recently created **Registration** entity.
5. Close the **Internet Explorer** and **Contoso.Events – Microsoft Visual Studio** applications.

**Results:** After completing this exercise, you will have used Visual Studio and the Azure to create a comprehensive development environment for Azure Storage.

©2016 Microsoft Corporation. All rights reserved. The text in this document is available under the [Creative Commons Attribution 3.0 License](https://creativecommons.org/licenses/by/3.0/legalcode), additional terms may apply. All other content contained in this document (including, without limitation, trademarks, logos, images, etc.) are ***not*** included within the Creative Commons license grant. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it. Some examples are for illustration only and are fictitious. No real association is intended or inferred. Microsoft makes no warranties, express or implied, with respect to the information provided here.