Azure Cosmos DB: Develop with the Table API in .NET

**Step 1:** Login to your Microsoft Azure portal, click on **Create new resource** and select **Azure CosmosDB** from **Databases** category to create a **CosmosDB Account** and give the necessary details as follows.

**ID:** A Unique Name

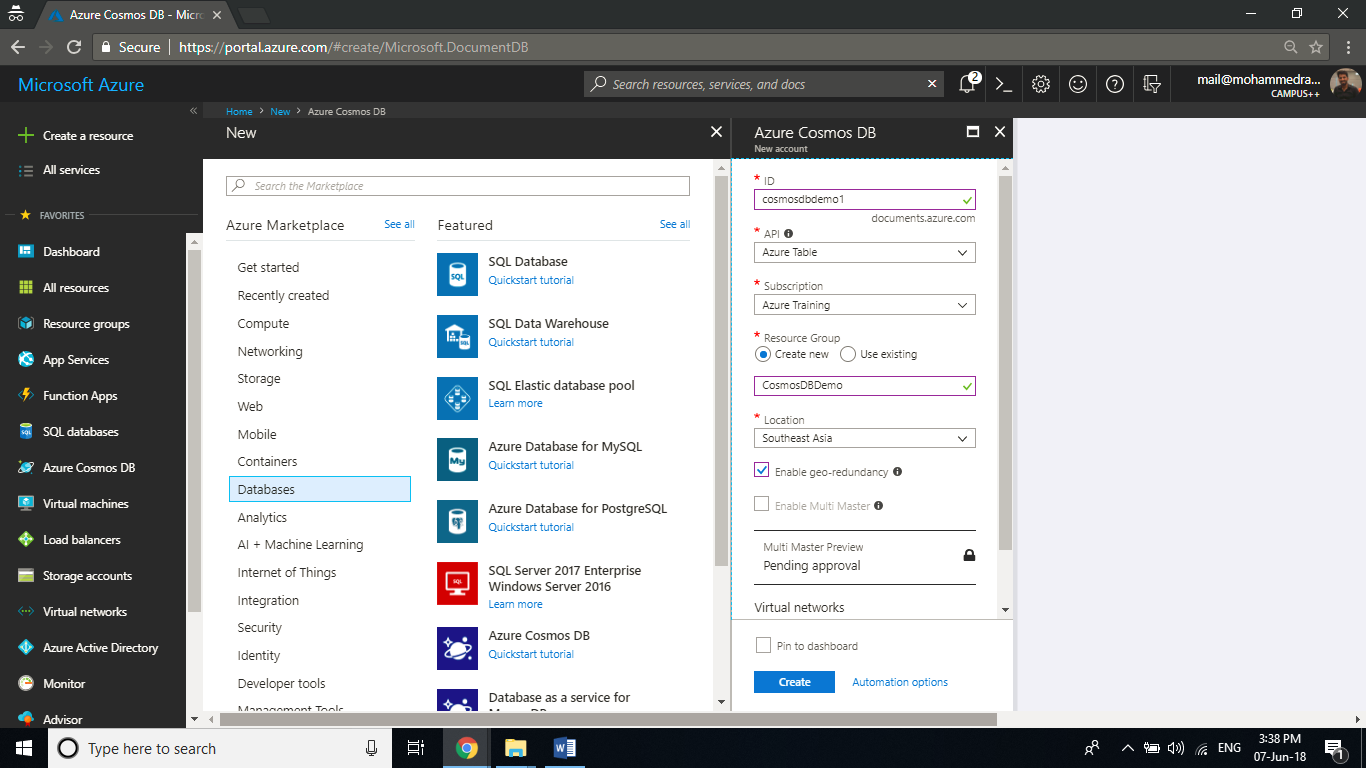
**API:** Azure table

**Subscription:** Select your subscription

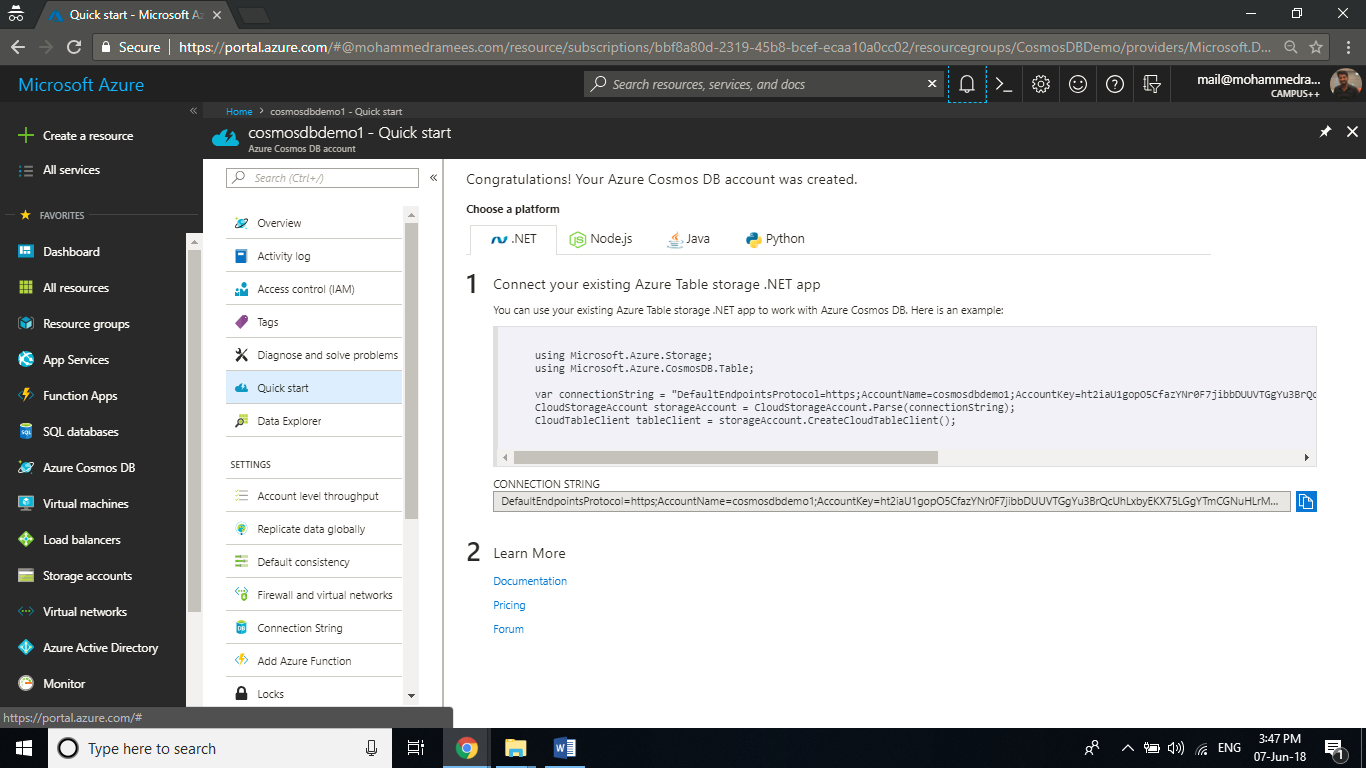
**Resource group**: Create a new as CosmosDBDemo

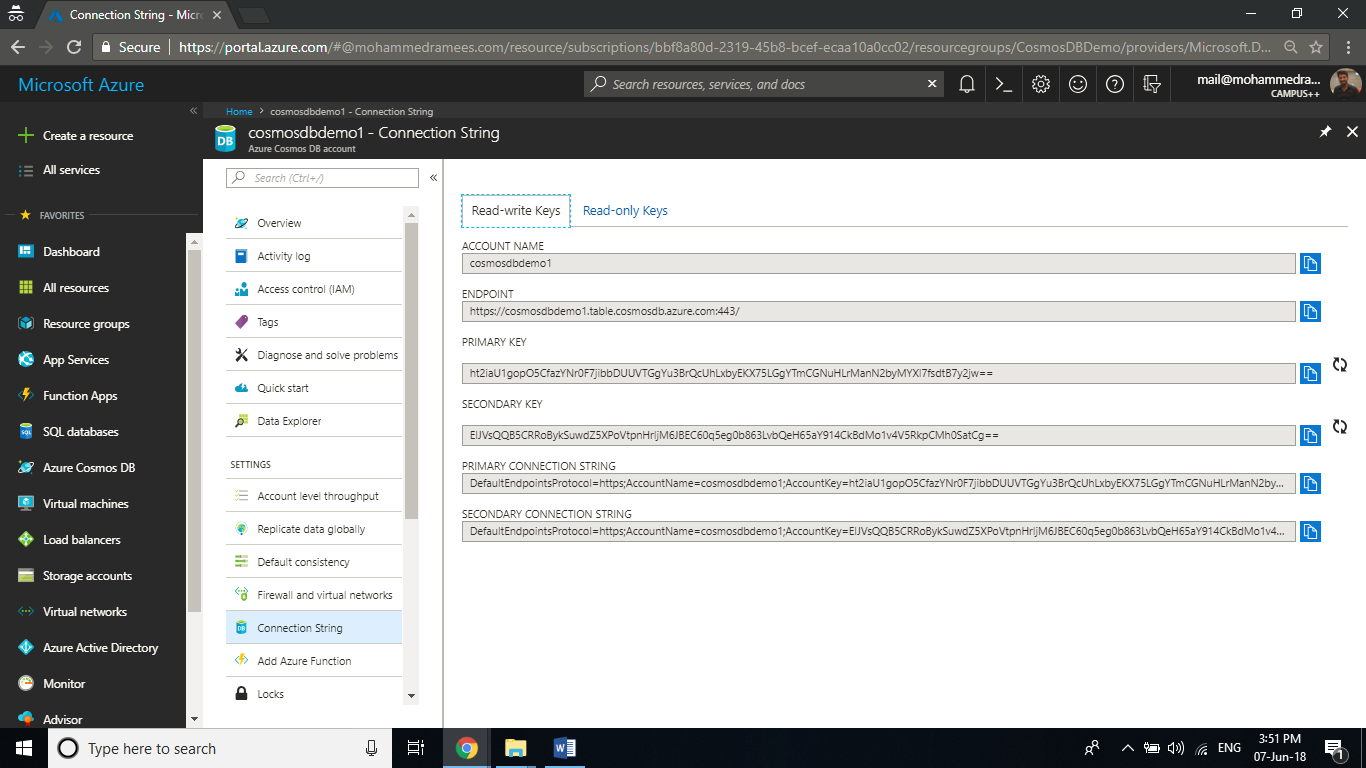
**Location:** Nearest to you

Check Enable geo-redundancy

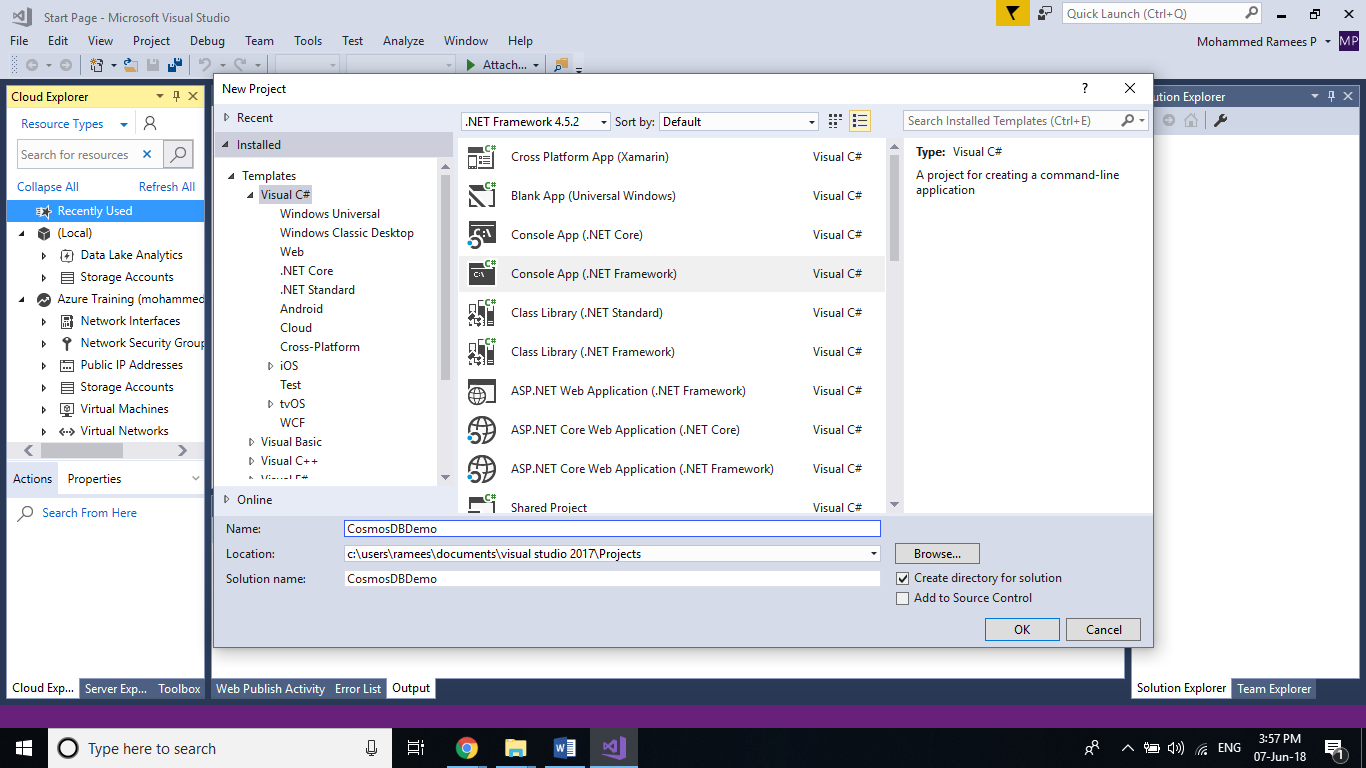


**Step 2:** Once the deployment is succeeded, open the resource and click on **Quick start/Connection String** and note the **Connection String** key for a later purpose.

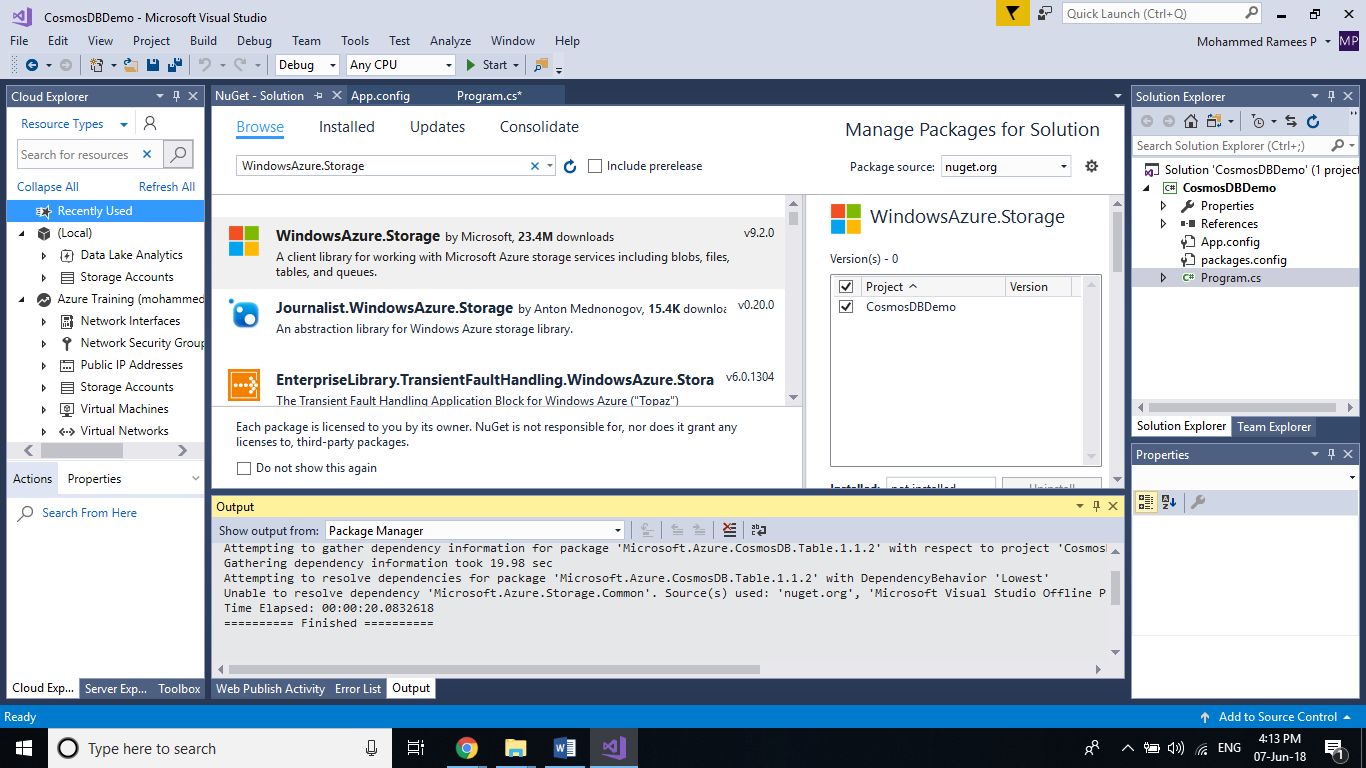


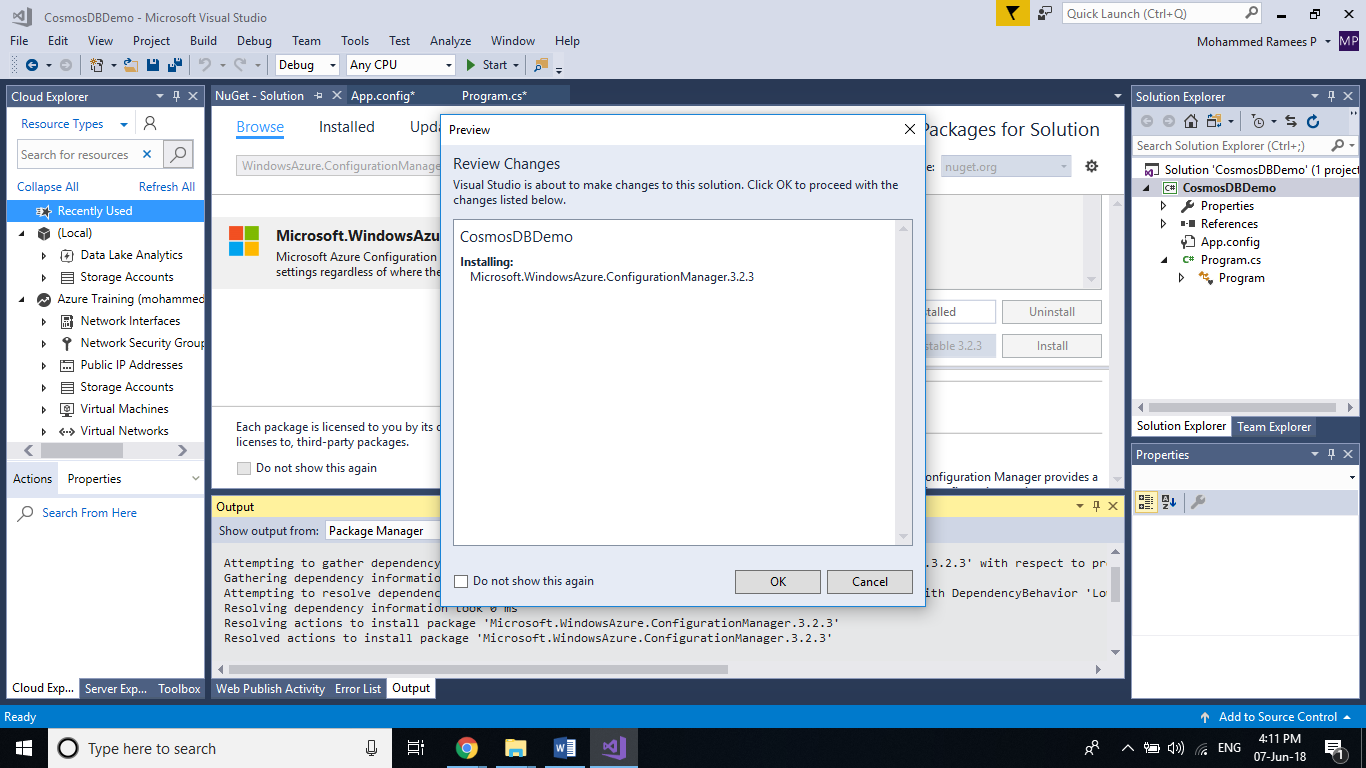


**Step 3:** Create a new C# **Console Application** within Visual Studio and name it as **CosmosDBDemo**.



**Step 4:** Once the project is created, **install two NuGet Packages** named **WindowsAzure.Storage** and **WindowsAzure.ConfigurationManager** to your solution. You can do this by selecting Tools > NuGet Package Manager > Manage NuGet Packages for Solution. In the Browse tab search for WindowsAzure.Storage and WindowsAzure.ConfigurationManager to get the packages complete the installation by reviewing the changes and accepting the license term for both.



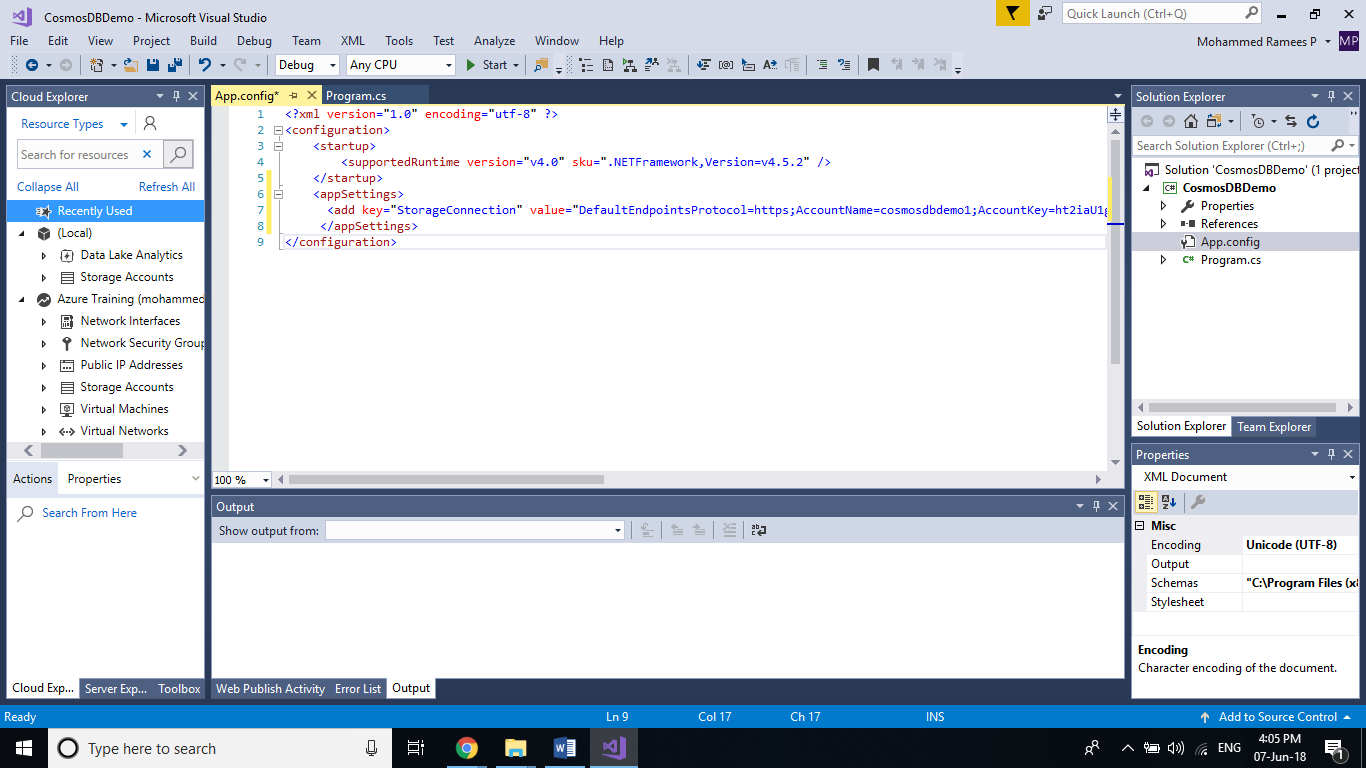


**Step 5:** Now we have to add the **Connection String** to the storage account. For that open the App.config file and add following code snippet within the configuration. Replace the [Connection string] with the connection string value you copied in Step 2.

<appSettings>

<add key="StorageConnection" value="[Connection String]" />

</appSettings>



**Step 6:** Now go to the **Program.cs** Page and add the following code to the Main method. This code will create a connection with the storage account and create a table named customers in CosmosDB.

//Reference to the ConnectionString in the App.Config file

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(

CloudConfigurationManager.GetSetting("StorageConnection"));

CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

CloudTable table = tableClient.GetTableReference("customers");

table.CreateIfNotExists();

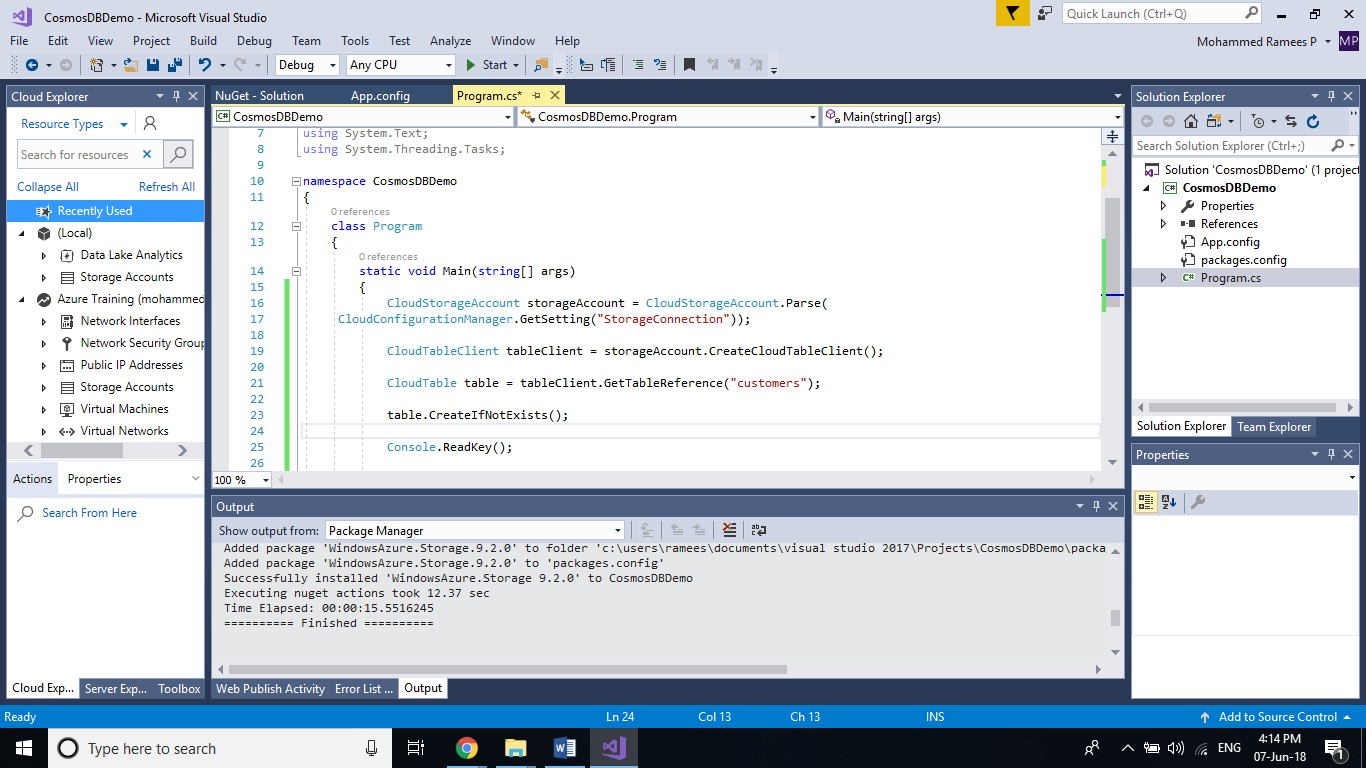
Console.ReadKey();

Add the following namespaces also

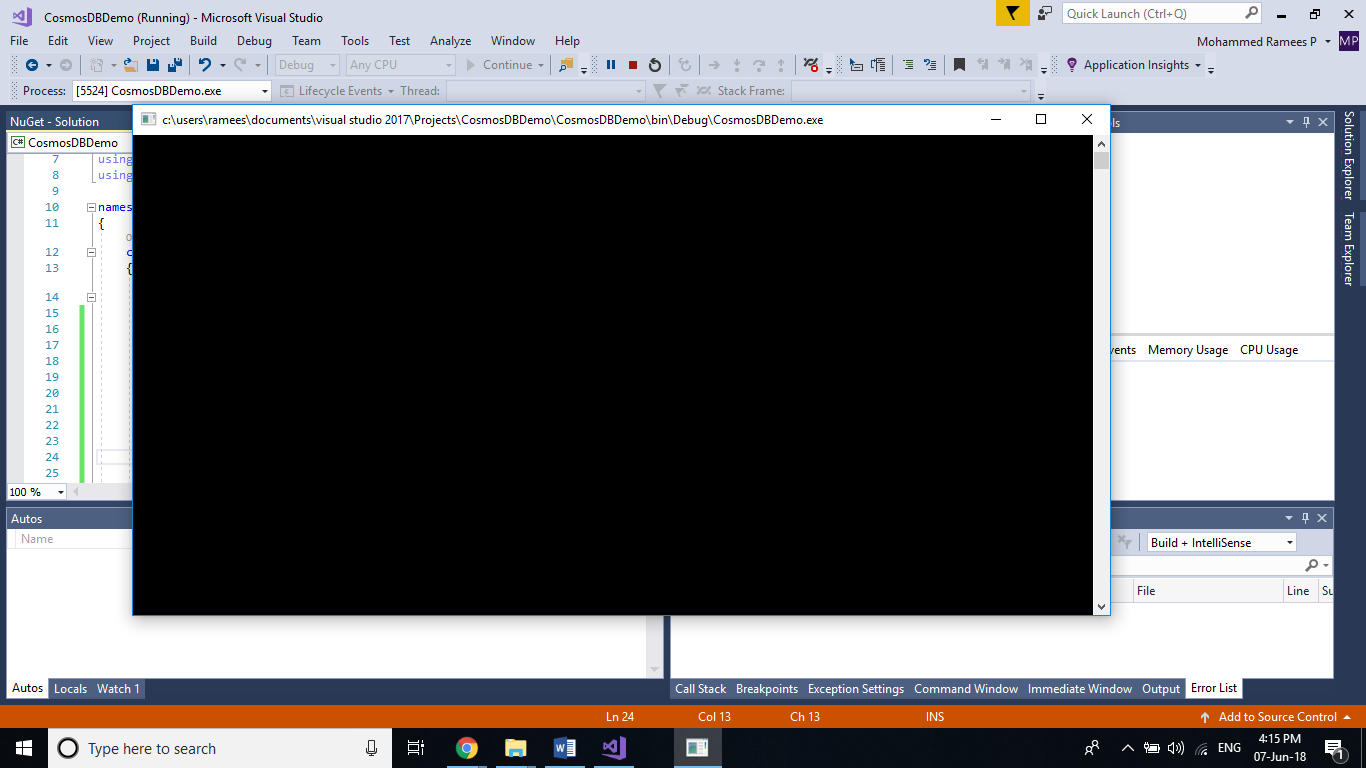
using Microsoft.Azure;

using Microsoft.WindowsAzure.Storage;

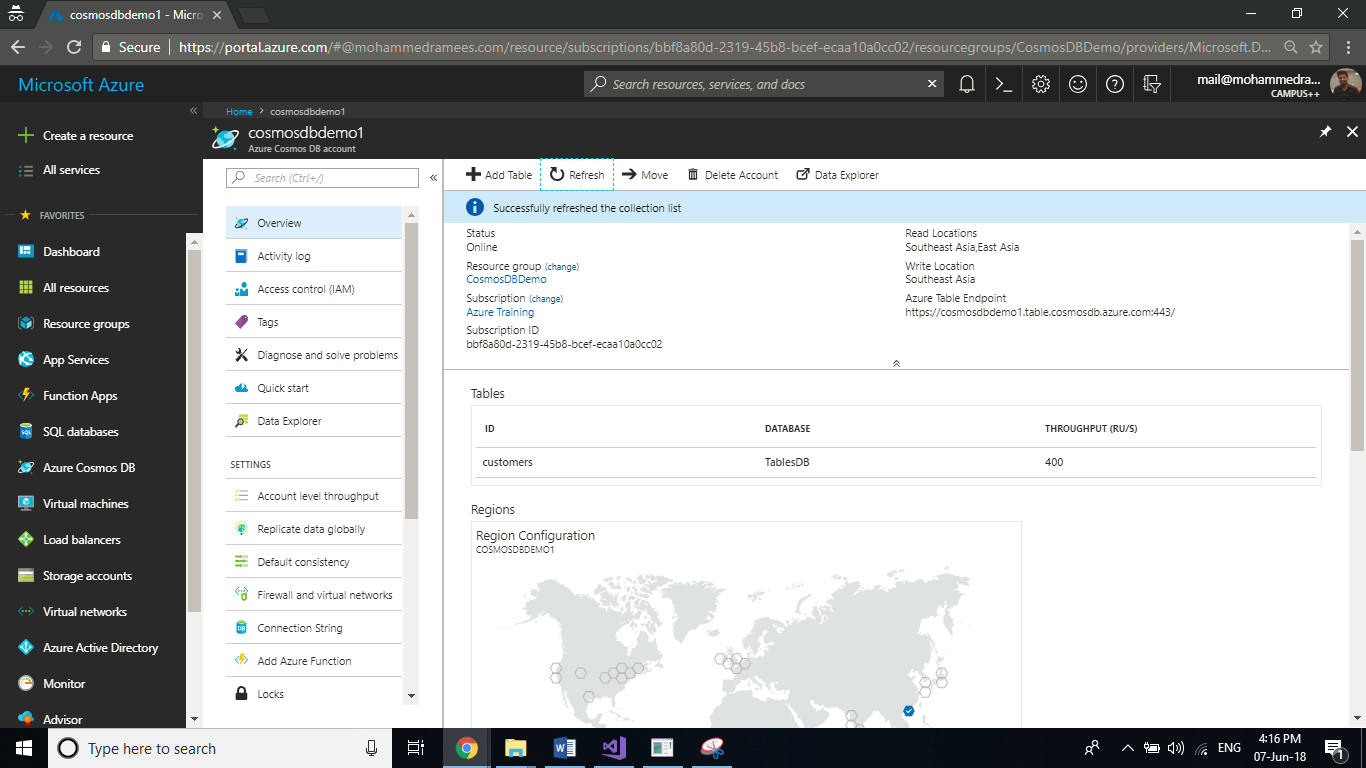
using Microsoft.WindowsAzure.Storage.Table;

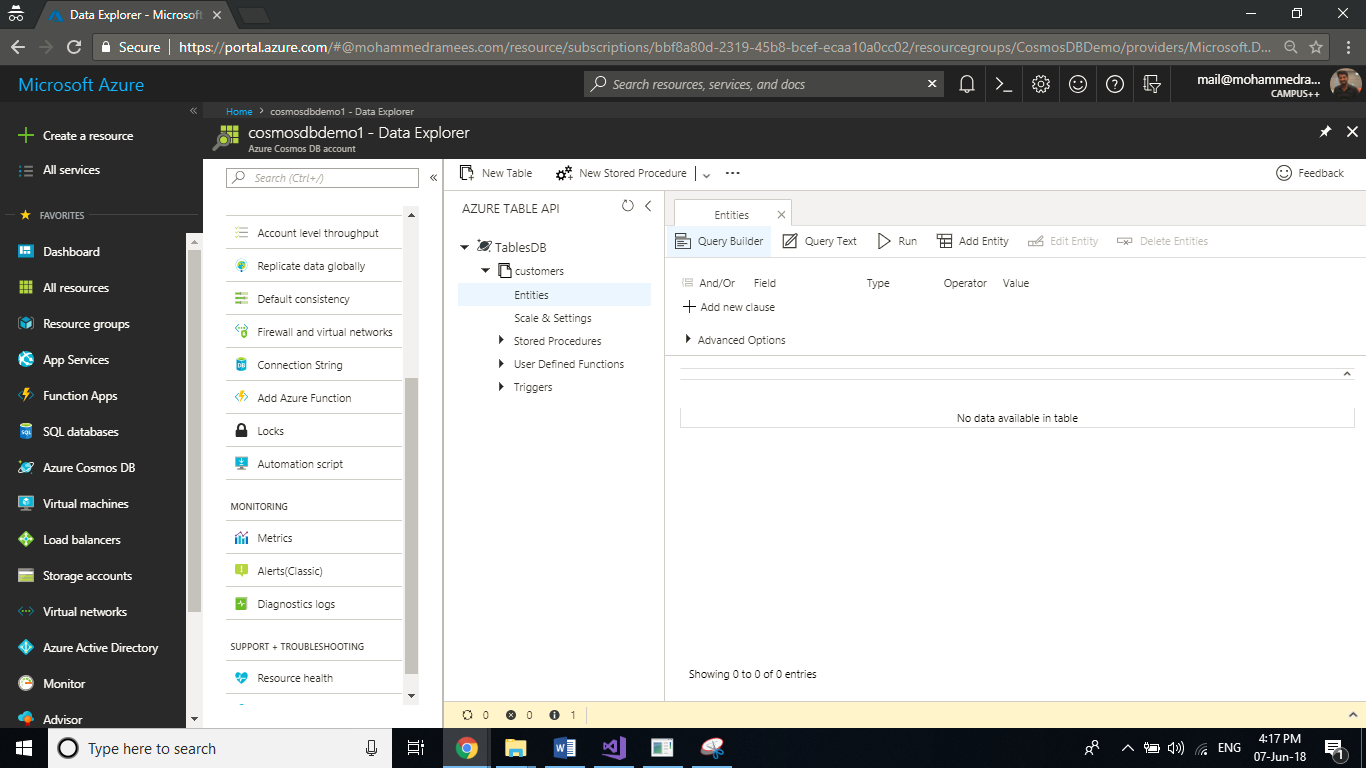


**Step 7:** Click on **Start** to run the program and you will see a console window waiting for an input as we given Console.Readkey() at the end of the code.



**Step 8:** Go back to your CosmosDB account and you can see our newly created table listed under tables section.



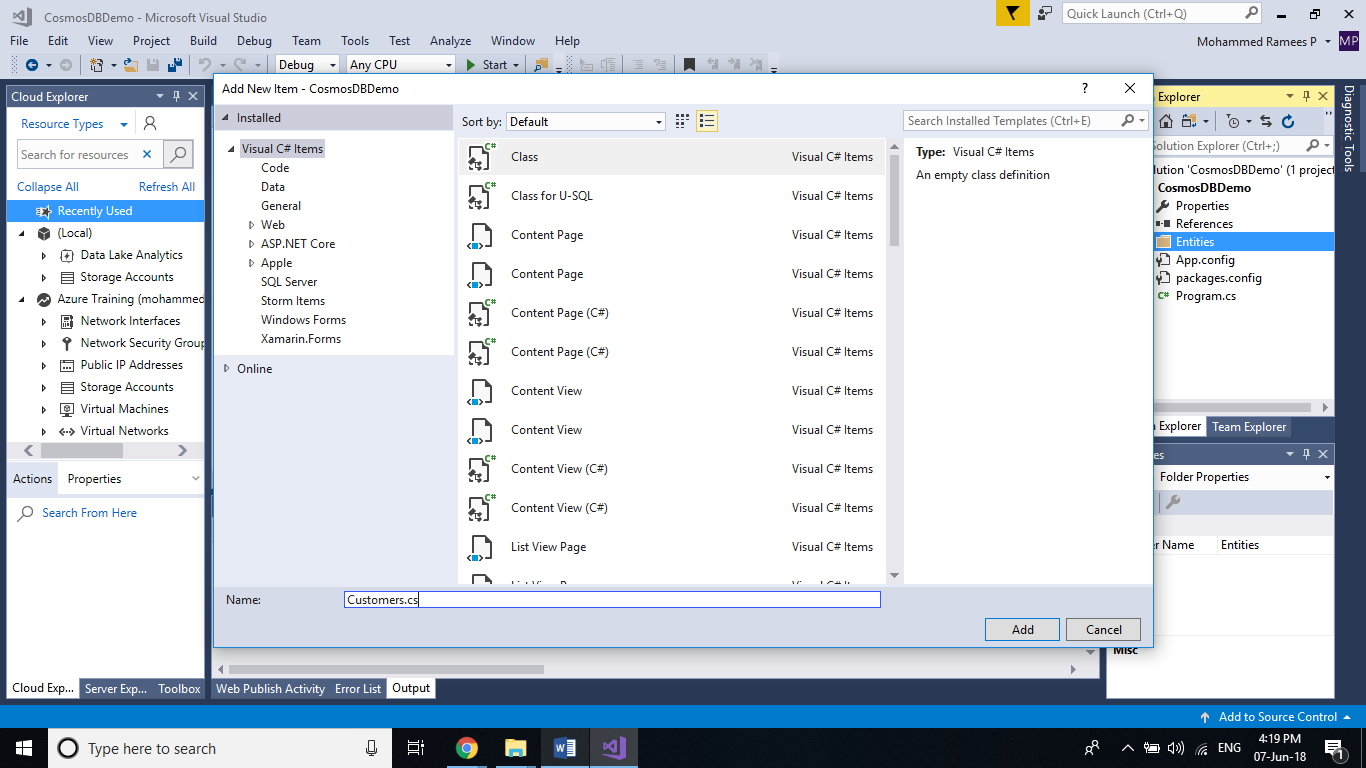


Add an entity to a table

**Step 1:** Open your project in Visual Studio and create **a new class** named **Customers** inside a newly created folder called **Entities**.

Right-click your project in solution explorer 🡪 Add 🡪 New Folder

Right Click on newly created folder 🡪 Add 🡪 New Item 🡪 Select class 🡪 Name the class as Customers.cs 🡪 Click on Add



**Step 2:** In the newly created class **Customers.cs** replace the existing code with the following code.

The custom class you creating here should be derived from the TableEntity class which lies under Microsoft.WindowsAzure.Storagetable namespace. Here we are just building the properties for this class and creating a constructor for it. We are setting the Partition-key as the type of the customer so that it stores all these records on the same storage node on the backend and the Row-key which need to be unique within a partition as the email.

using Microsoft.WindowsAzure.Storage.Table;

namespace TableDemo.Entities

{

class Customer : TableEntity

{

public string CustomerName { get; set; }

public string CustomerEmail { get; set; }

public string CustomerType { get; set; }

public Customer () { }

public Customer(string name, string email, string type)

{

this.CustomerName = name;

this.CustomerEmail = email;

this.CustomerType = type;

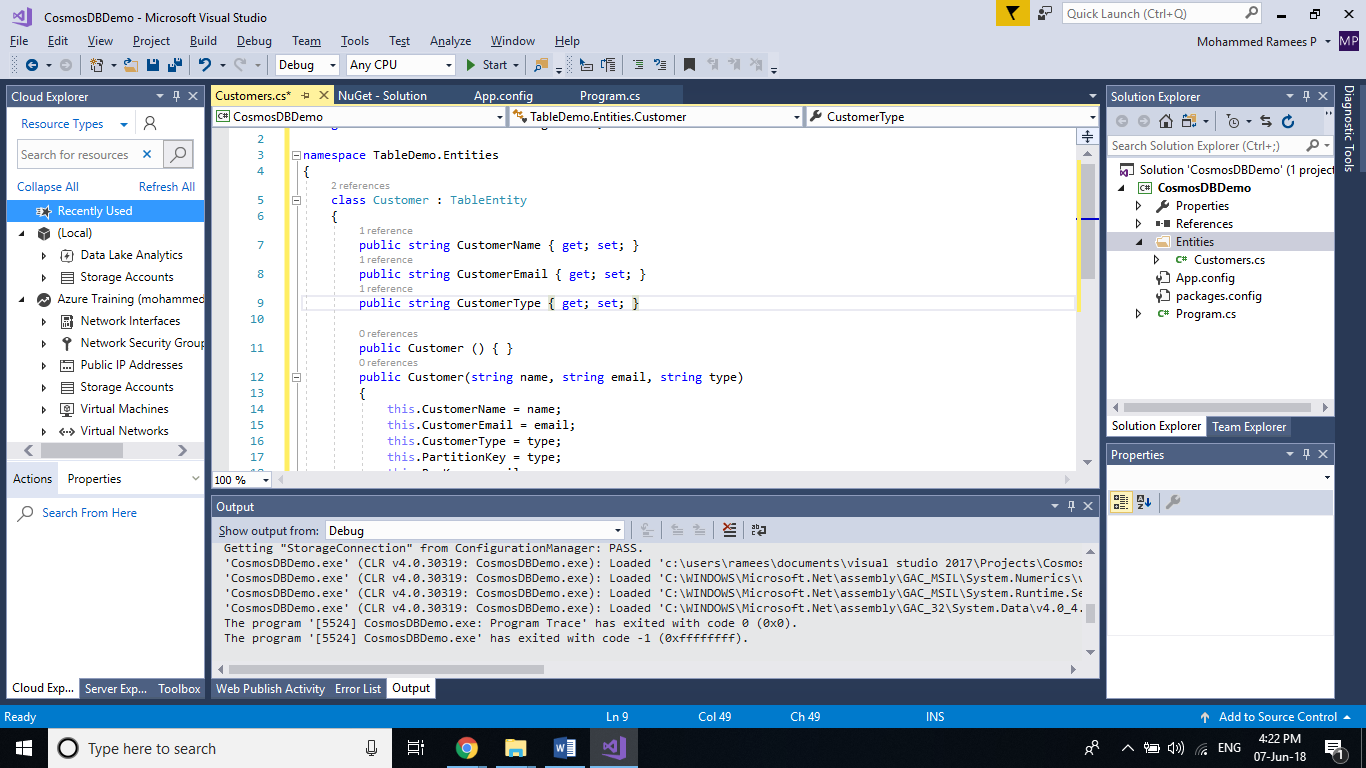
this.PartitionKey = type;

this.RowKey = email;

}

}

}



**Step 3:** Back in the **Program.cs**, Add the following code to your **Main** method after the code for getting table reference. (You may delete the code to create the table if not exists as we are sure that it is already created).

CreateCustomer(table, new Customer("Customer1", "cusmtomer1@mail.com", "registered"));

Console.ReadKey();

Add this method to the same class to create(insert) the customer within the table which is called in the above block of code from the main class.

static void CreateCustomer(CloudTable table, Customer user)

{

TableOperation insert = TableOperation.Insert(user);

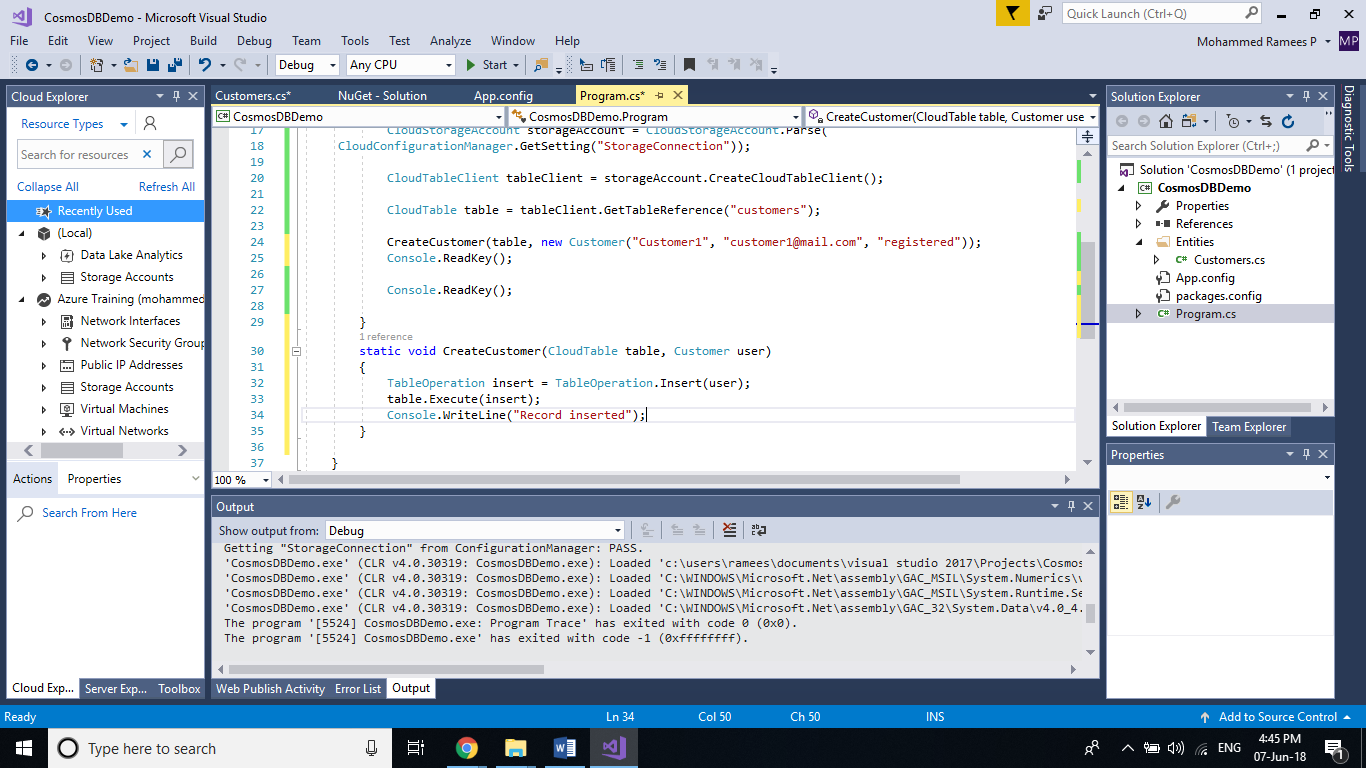
table.Execute(insert);

Console.WriteLine("Record inserted");

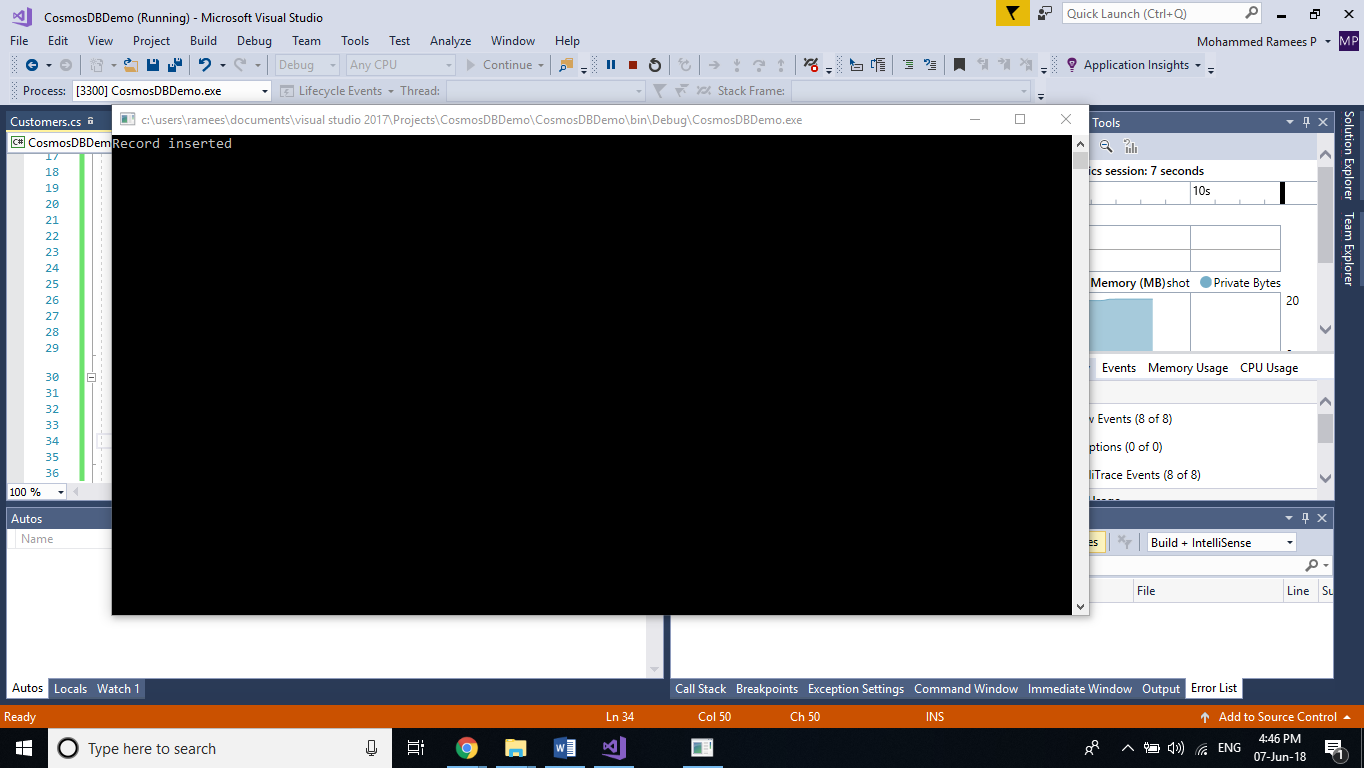
}

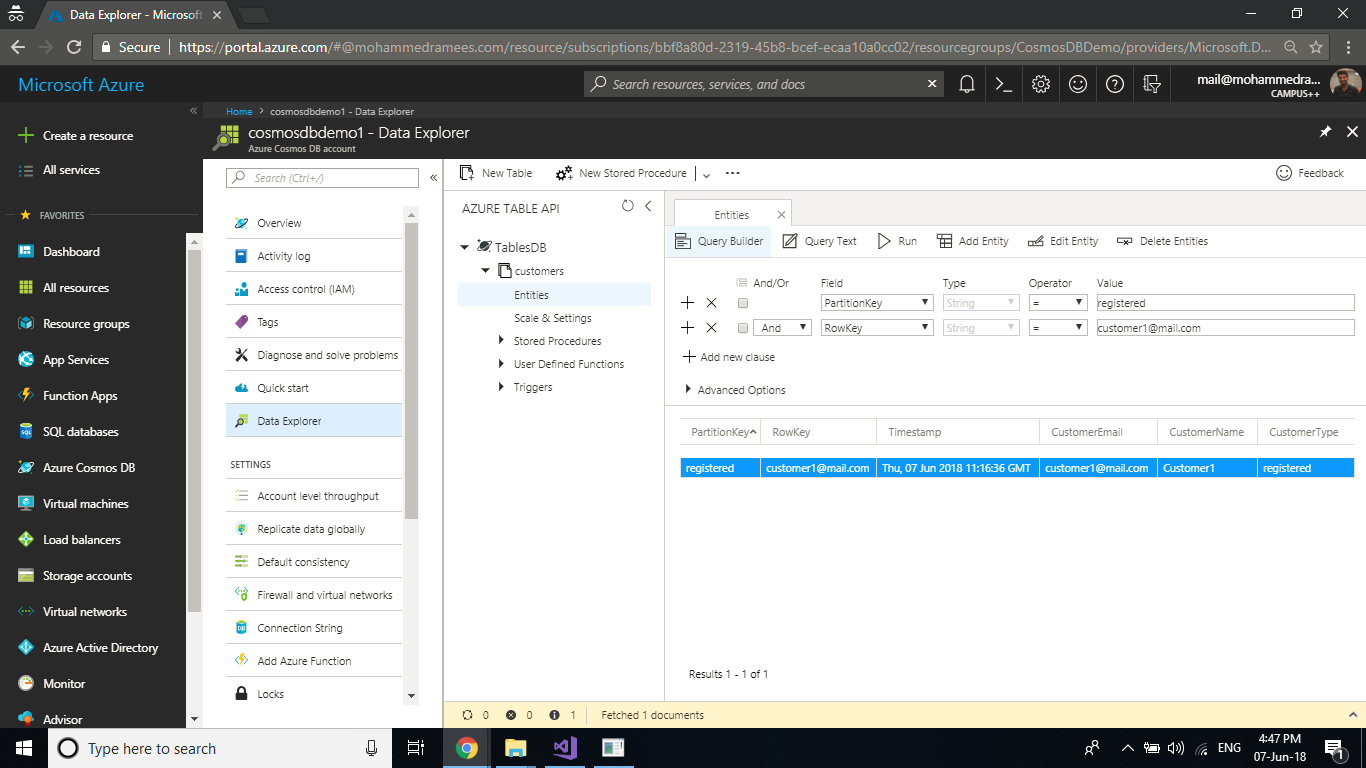
You may also need to add the following namespace to access the customer class from the entity.

using TableDemo.Entities;



**Step 4:** Click on start to run the program and we can assume that the record inserted successfully as there was no error we got the message specified in the same block of code.





Insert a batch of entities

TableBatchOperation batchOperation = new TableBatchOperation();

Customer customer2 = new Customer("Customer2", "customer2@mail.com", "registered");

Customer customer3 = new Customer("Customer3", "customer3@mail.com", "registered");

Customer customer4 = new Customer("Customer4", "customer4@mail.com", "registered");

Customer customer5 = new Customer("Customer5", "customer5@mail.com", "registered");

batchOperation.Insert(customer2);

batchOperation.Insert(customer3);

batchOperation.Insert(customer4);

batchOperation.Insert(customer5);

table.ExecuteBatch(batchOperation);

Retrieve a single entity

TableOperation retrieveOperation = TableOperation.Retrieve<Customer>("registered", "customer1@mail.com");

TableResult retrievedResult = table.Execute(retrieveOperation);

Console.WriteLine(((Customer)retrievedResult.Result).CustomerName);

Console.ReadLine();

Retrieve all entities

TableQuery<Customer> query = new TableQuery<Customer>()

.Where(TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "registered"));

foreach (Customer customer in table.ExecuteQuery(query))

{

Console.WriteLine(customer.CustomerName + " " + customer.CustomerEmail);

}

Console.ReadLine();

Replace the code after getting reference to the table with the following for the remaining operations

Replace an entity

Customer customerEntity = RetrieveCustomer(table, "registered", "customer1@mail.com");

if (customerEntity != null)

{

Console.WriteLine("Enter the new name for the customer");

string newName = Console.ReadLine();

customerEntity.CustomerName = newName;

}

UpdateCustomer(table, customerEntity);

RetrieveAllCustomers(table);

Console.ReadKey();

}

public static Customer RetrieveCustomer(CloudTable table, string partitionKey, string rowKey)

{

TableOperation tableOperation = TableOperation.Retrieve<Customer>(partitionKey, rowKey);

TableResult tableResult = table.Execute(tableOperation);

return tableResult.Result as Customer;

}

static void UpdateCustomer(CloudTable table, Customer customer)

{

TableOperation update = TableOperation.Replace(customer);

table.Execute(update);

}

static void RetrieveAllCustomers(CloudTable table)

{

TableQuery<Customer> query = new TableQuery<Customer>()

.Where(TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "registered"));

foreach (Customer customer in table.ExecuteQuery(query))

{

Console.WriteLine(customer.CustomerName + " " + customer.CustomerEmail);

}

}

Delete an entity

Customer customerEntity = RetrieveCustomer(table, "registered", "customer1@mail.com");

DeleteCustomer(table, customerEntity);

RetrieveAllCustomers(table);

Console.ReadKey();

}

public static Customer RetrieveCustomer(CloudTable table, string partitionKey, string rowKey)

{

TableOperation tableOperation = TableOperation.Retrieve<Customer>(partitionKey, rowKey);

TableResult tableResult = table.Execute(tableOperation);

return tableResult.Result as Customer;

}

static void DeleteCustomer(CloudTable table, Customer customer)

{

TableOperation delete = TableOperation.Delete(customer);

table.Execute(delete);

Console.WriteLine("Record deleted");

}

static void RetrieveAllCustomers(CloudTable table)

{

TableQuery<Customer> query = new TableQuery<Customer>()

.Where(TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "registered"));

foreach (Customer customer in table.ExecuteQuery(query))

{

Console.WriteLine(customer.CustomerName + " " + customer.CustomerEmail);

}

}

Delete the table

var status = table.DeleteIfExists();

if (status == true) Console.WriteLine("Table deleted");

else Console.WriteLine("Table does not exists");

Console.ReadKey();