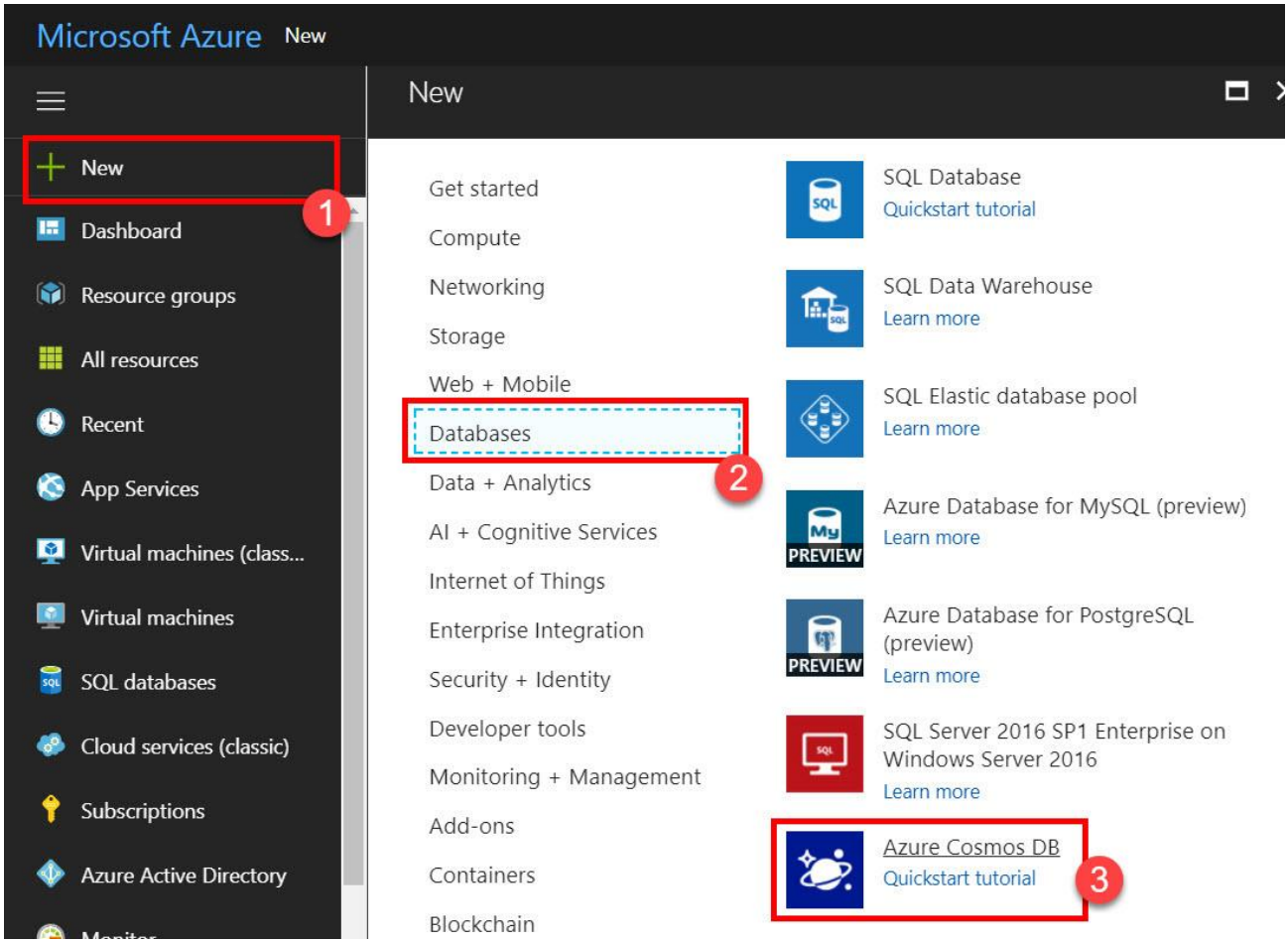
Azure Cosmos DB – Build a .Net Application using Table API

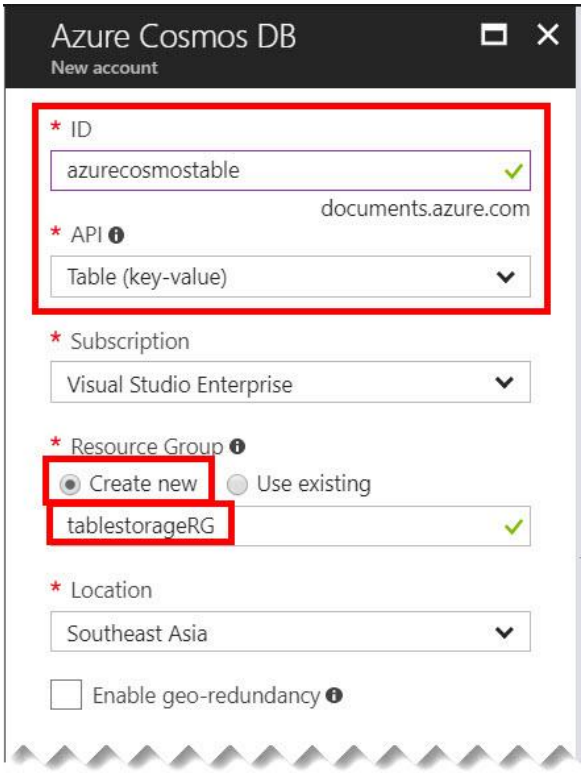
Step 1: Open Microsoft Azure Portal

https://portal.azure.com

Step 2: Click on **+New -> Databases -> Azure Cosmos DB**



Step 3: Enter Azure Cosmos DB Account Details



ID: Enter Unique Cosmos DB ID

Ex. azurecosmostable

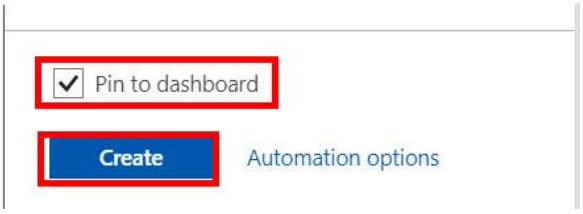
API: Table (key-value)

Subscription: Choose any working Azure Subscription

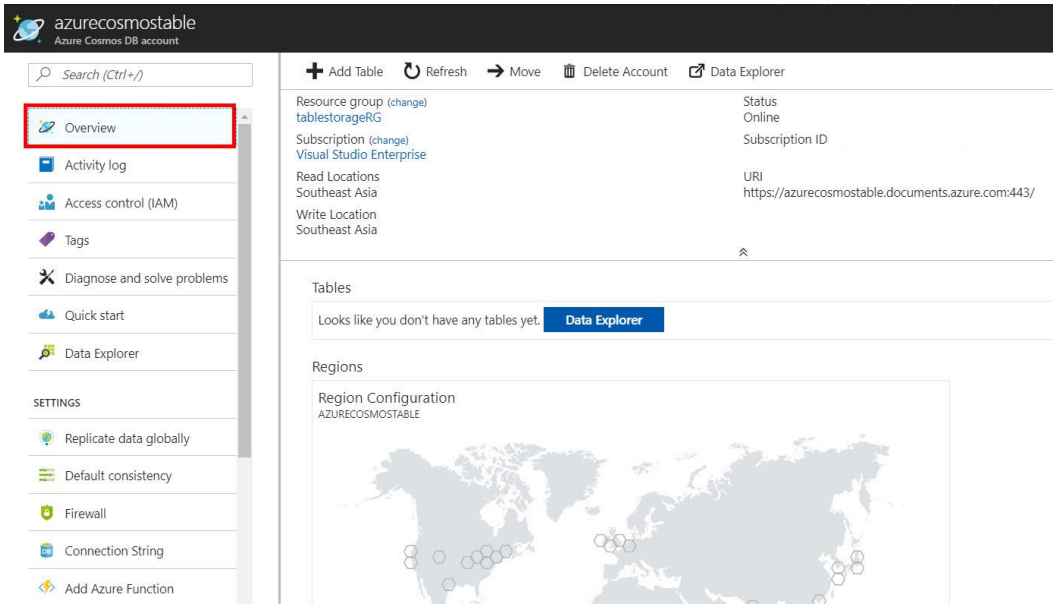
Resource Group: Create new or Use Existing

Ex. tablestorageRG

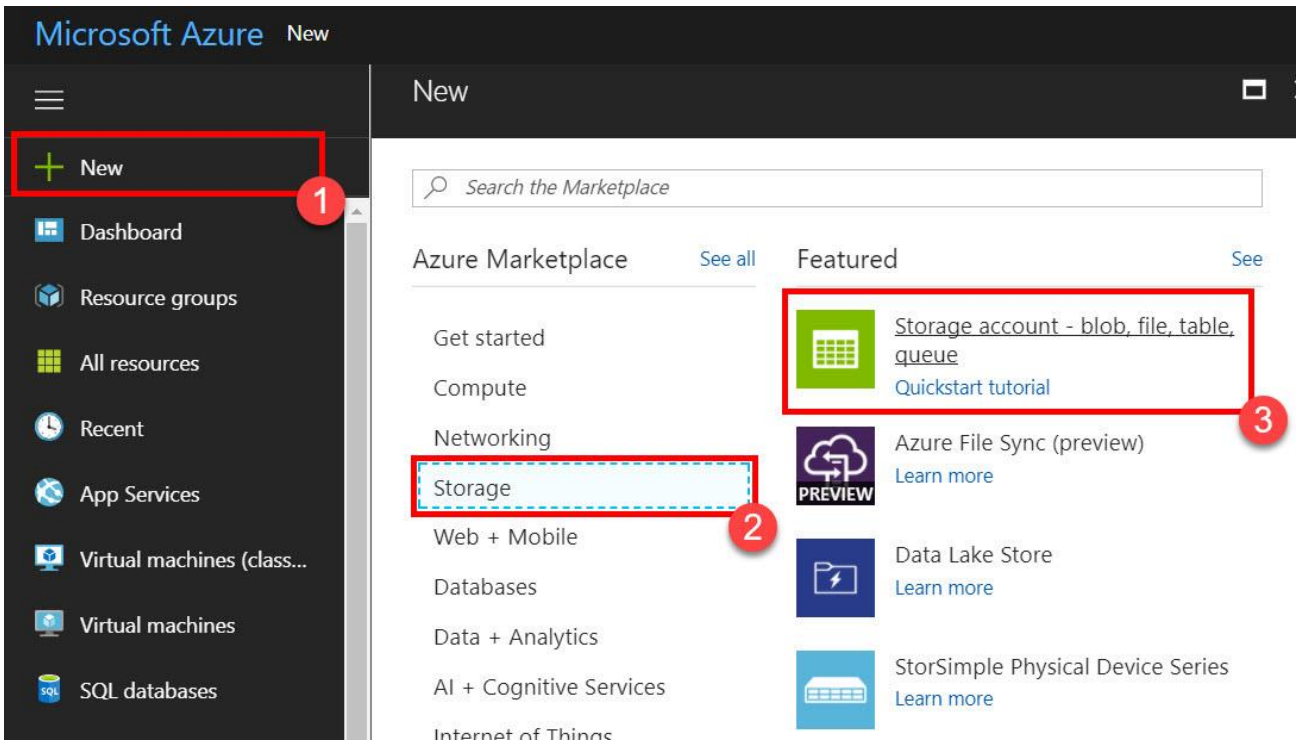
Location: Choose nearest region



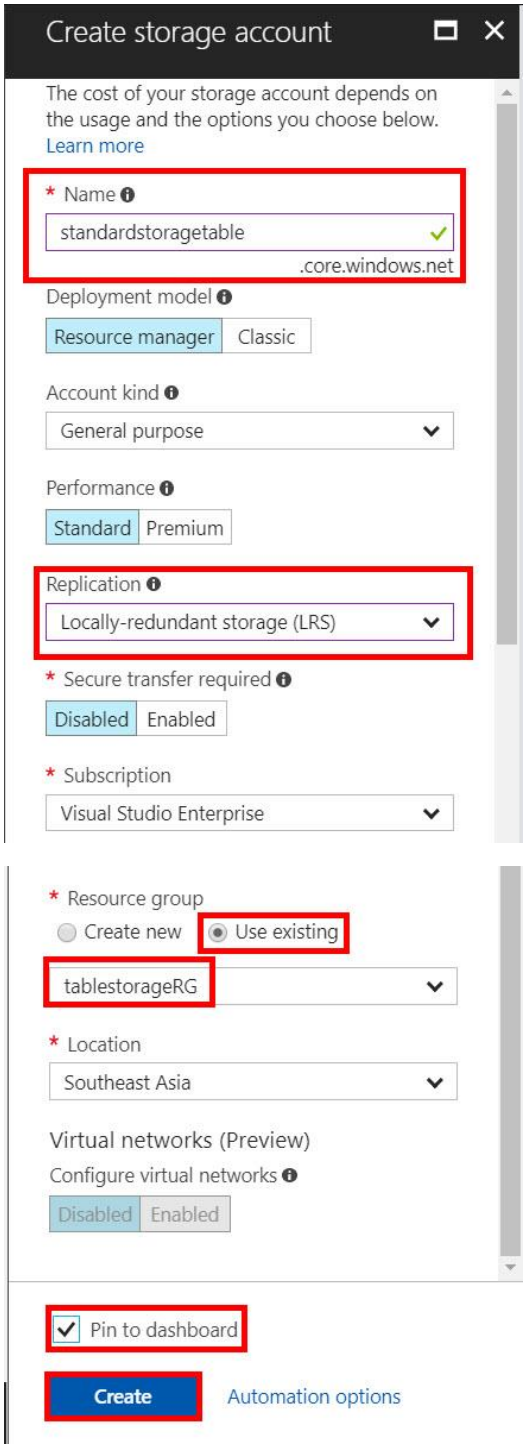
Wait for few minutes to create Cosmos DB



Step 4: Again, Click on **+New -> Storage -> Storage account – blob, file, table, queue**



Step 5: Enter Storage Details



Name: Enter Unique Storage Name

Ex. standardstoragetable

Deployment Model: Resource Manager

Account Kind: General purpose

Performance: Standard

Replication: Locally-redundant storage (LRS)

Secure transfer required: Disabled

Subscription: Choose any working Azure Subscription

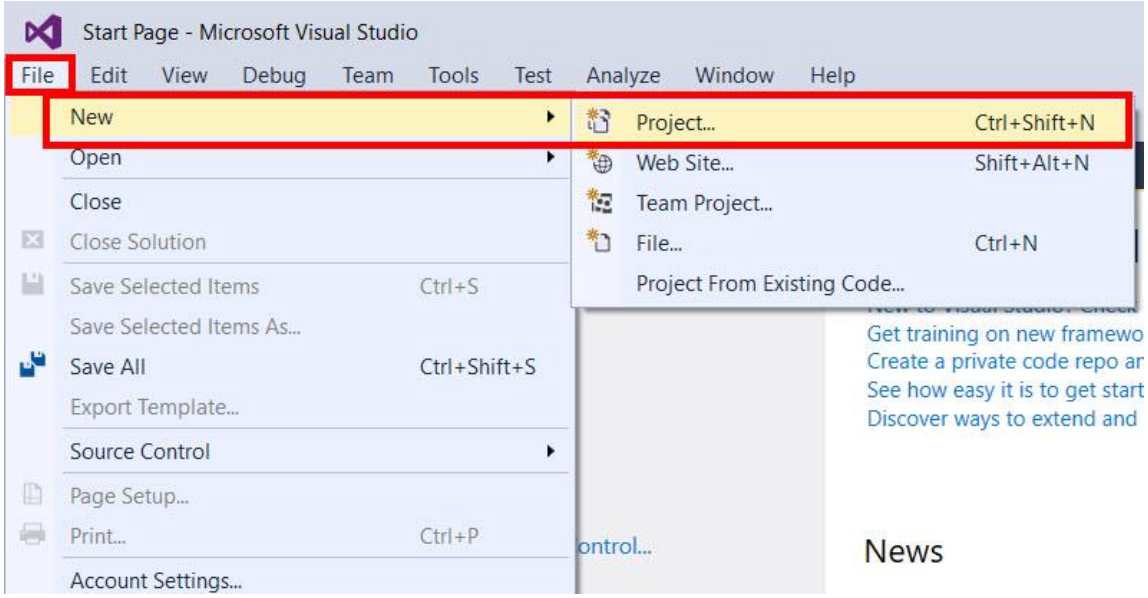
Resource Group: Choose existing Group

Ex. tablestorageRG

Location: Choose nearest region

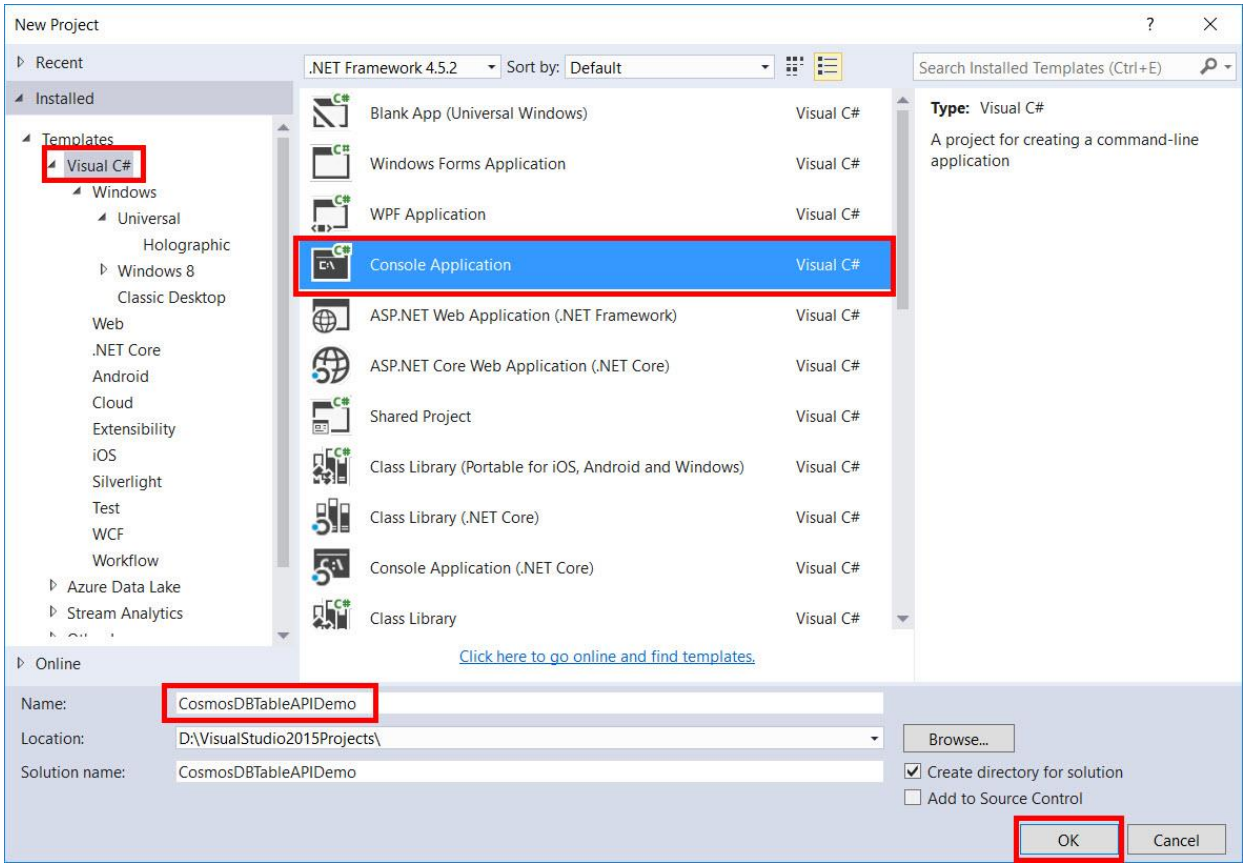
Step 6: Start Visual Studio 2015/2017

Click on **File Menu -> New -> Project**

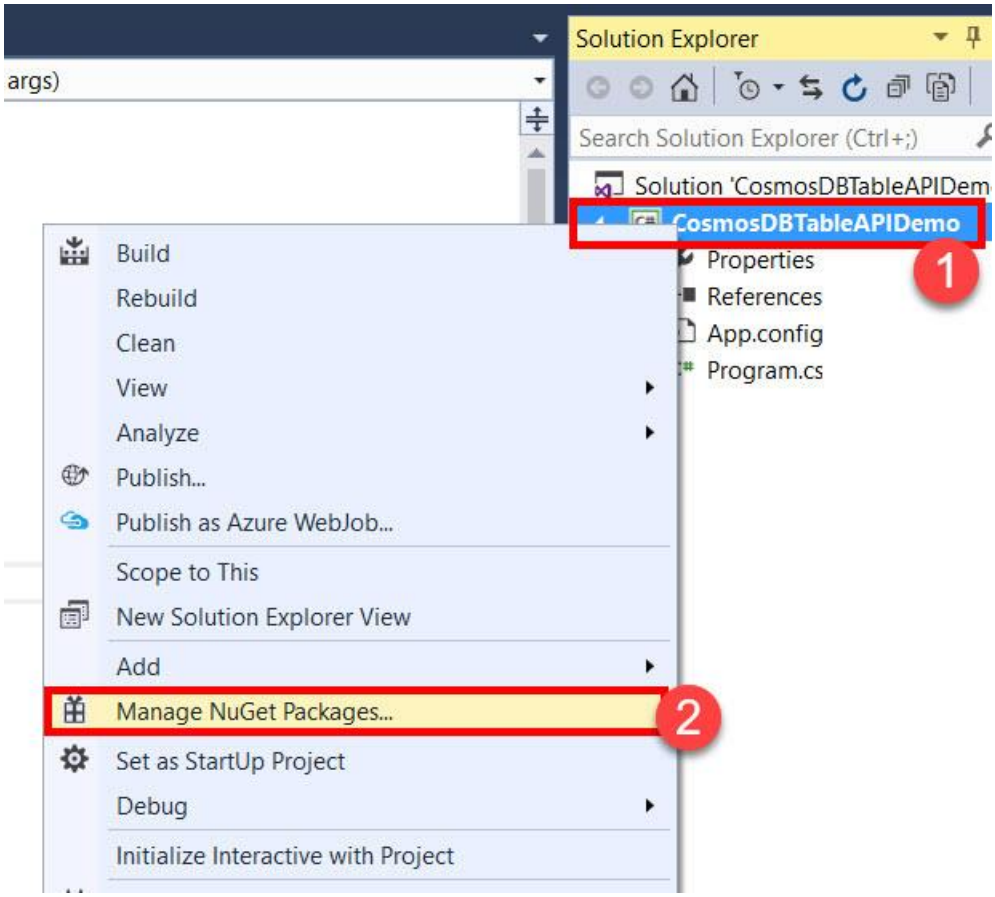


Step 7: Choose **Visual C# -> Console Application**

Enter Project Name



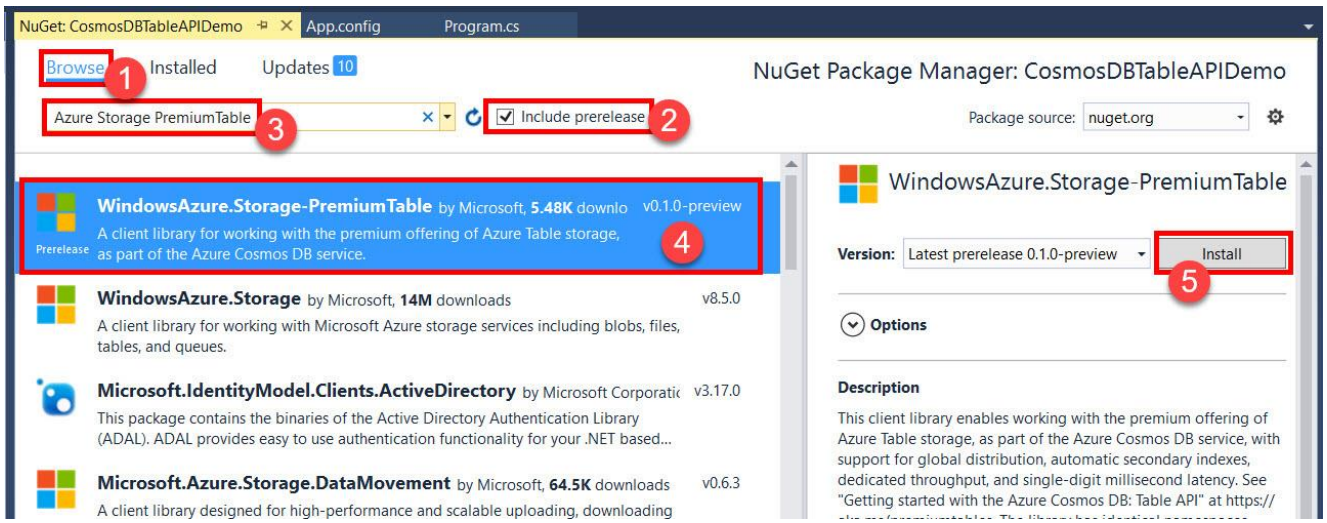
Step 8: Right click on **Project Name** & Click on **Manage NuGet Packages…**



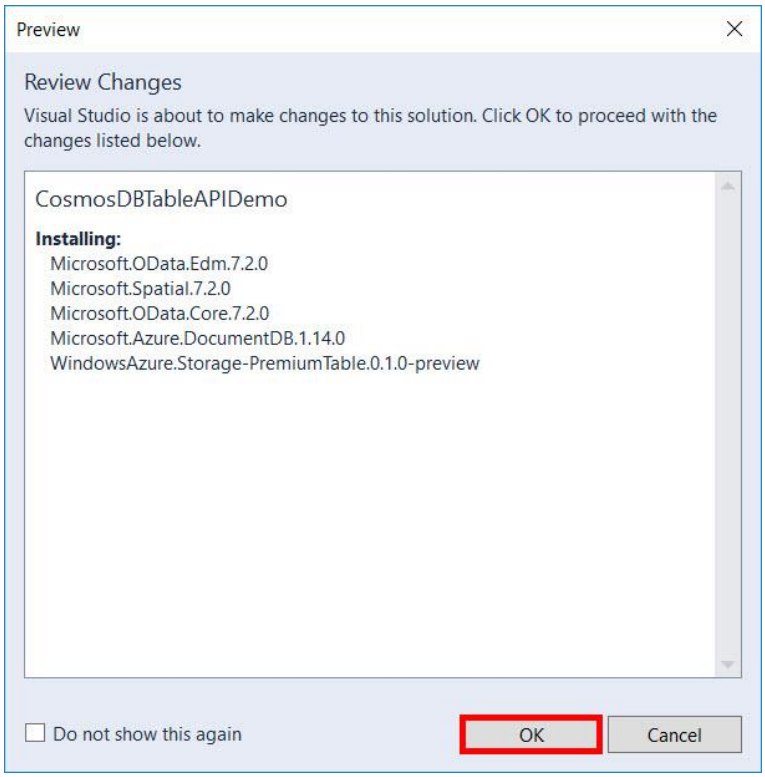
Step 9: Click on **Browse** option

Check **Include prerelease** option

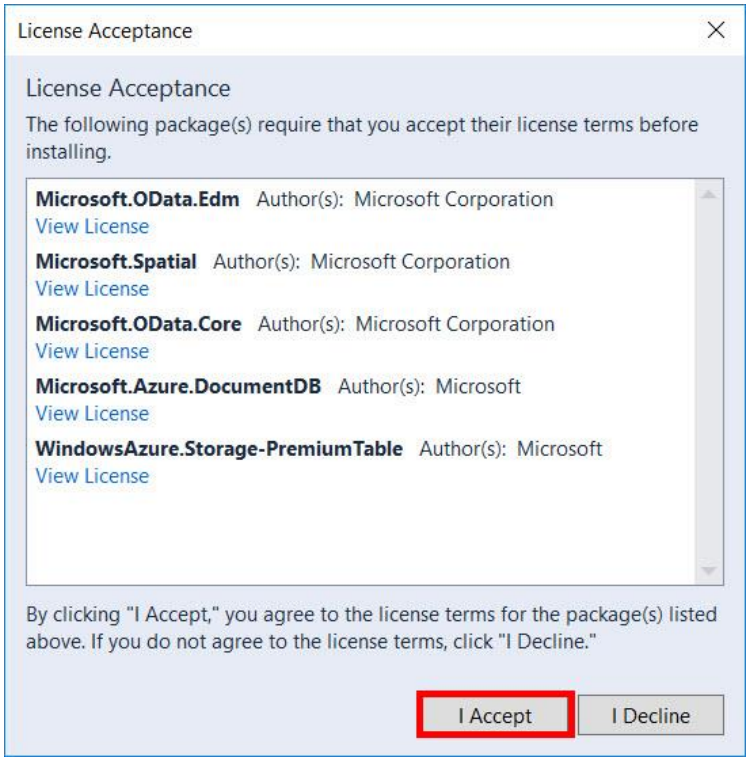
Search for **Azure Storage PremiumTable** & click on **Install** Button.



Click on **OK** button for review changes.



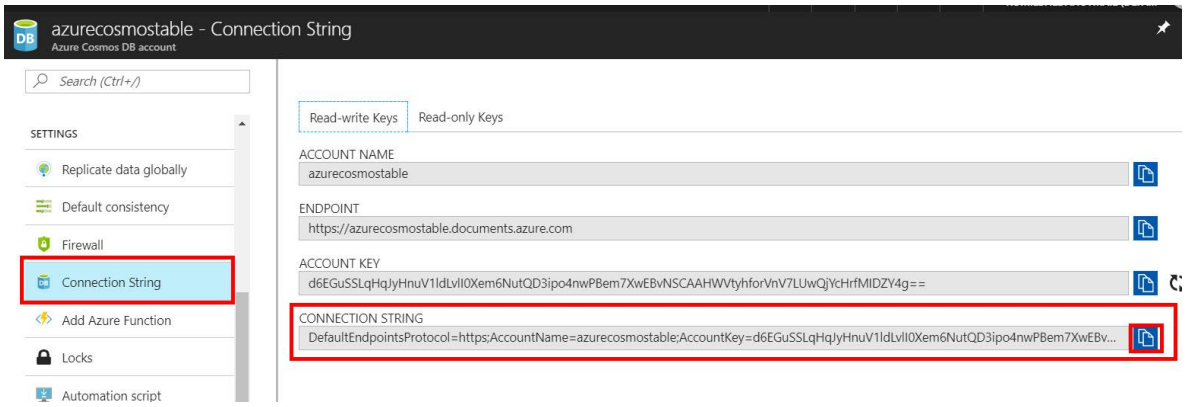
Click on **I Accept** button for License Acceptance



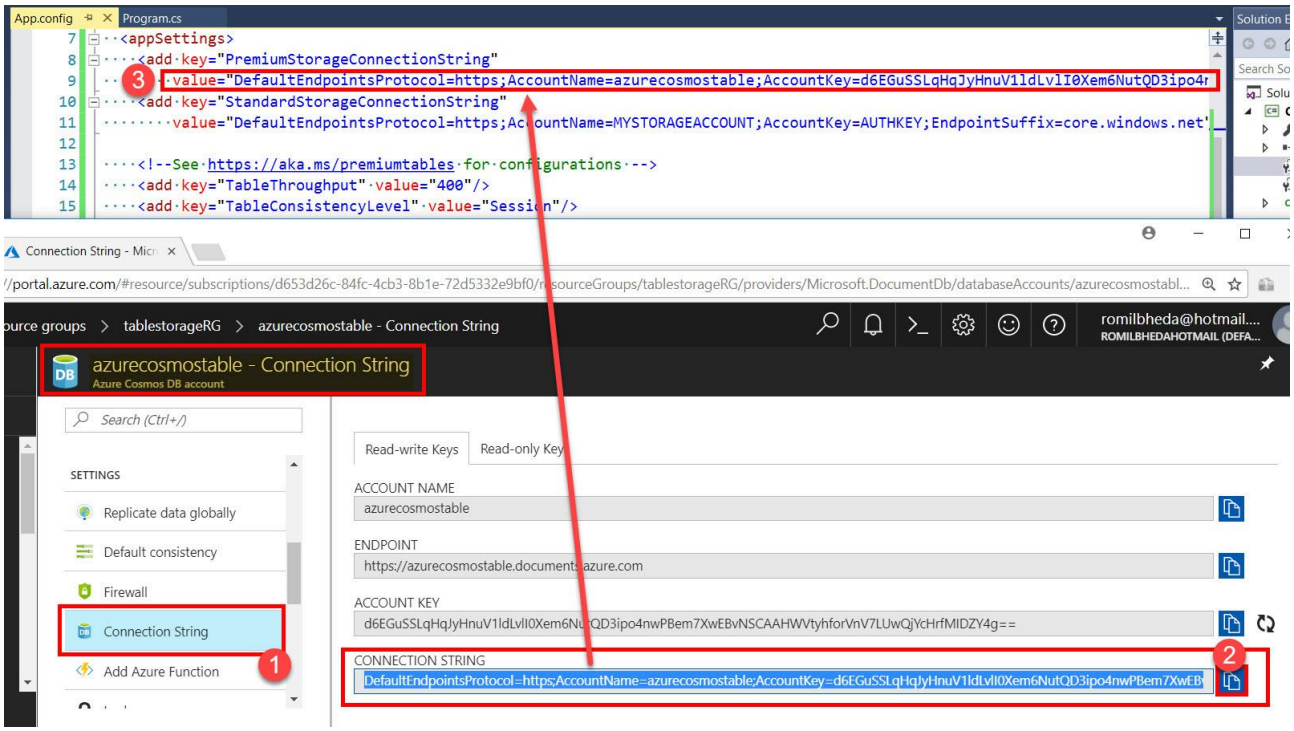
Step 10: Open **App.config** file & add highlighted code below Code:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>

    <appSettings>

      <add key="PremiumStorageConnectionString"

value="DefaultEndpointsProtocol=https;AccountName=MYSTORAGEACCOUNT;AccountKey=AUTHKEY;
TableEndpoint=https://COSMODB.documents.azure.com" />
      <add key="StandardStorageConnectionString"

value="DefaultEndpointsProtocol=https;AccountName=MYSTORAGEACCOUNT;AccountKey=AUTHKEY;
EndpointSuffix=core.windows.net" />

      <!--See https://aka.ms/premiumtables for configurations -->
      <add key="TableThroughput" value="400"/>
      <add key="TableConsistencyLevel" value="Session"/>
      <add key="TablePreferredLocations" value="West US,East US"/>
    </appSettings>
</configuration>
```

Step 11: Navigate to Azure Portal & Choose **Connection String** option & Copy Connection String.

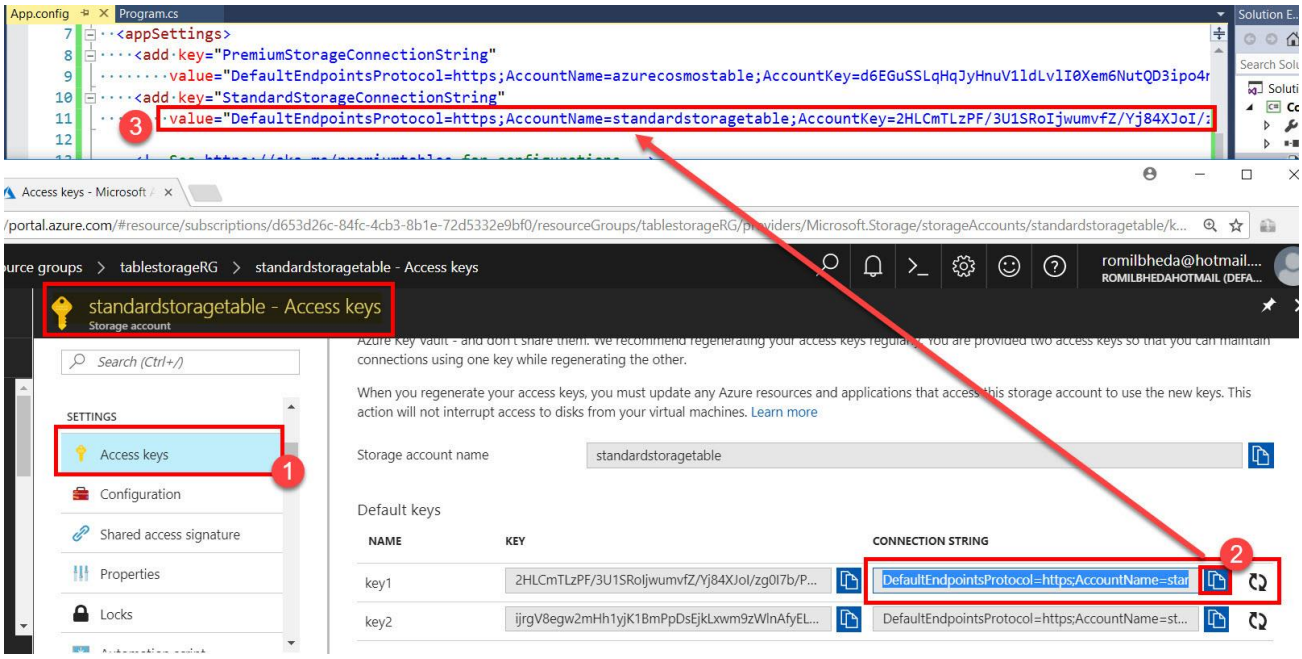


Step 12: Replace **PremiumStorageConnectionString** value with **Azure Cosmos DB Table** Connection String

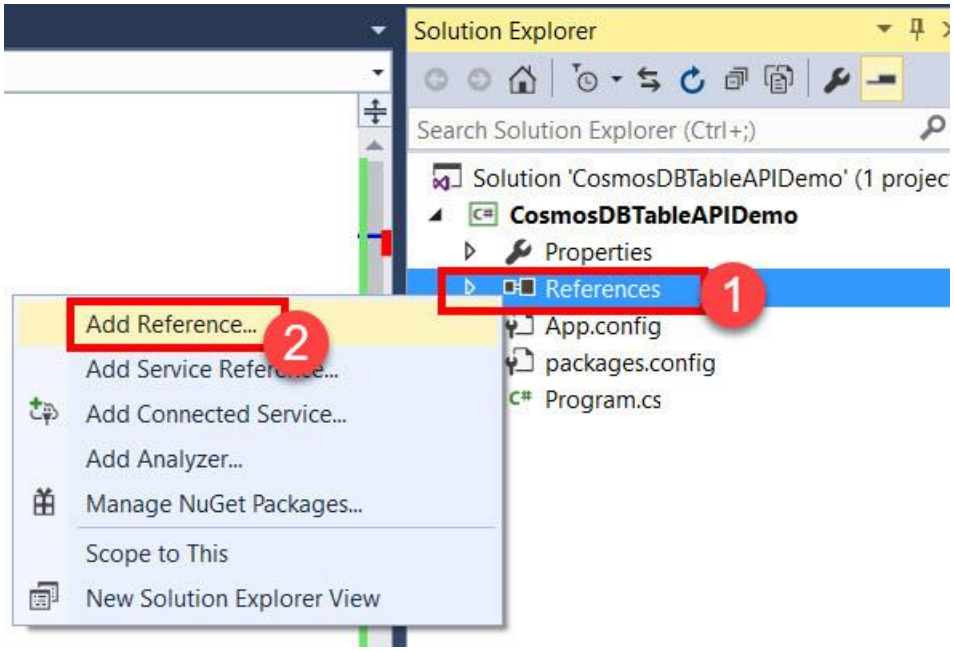


Step 13: Now navigate to **Azure Storage Account**.

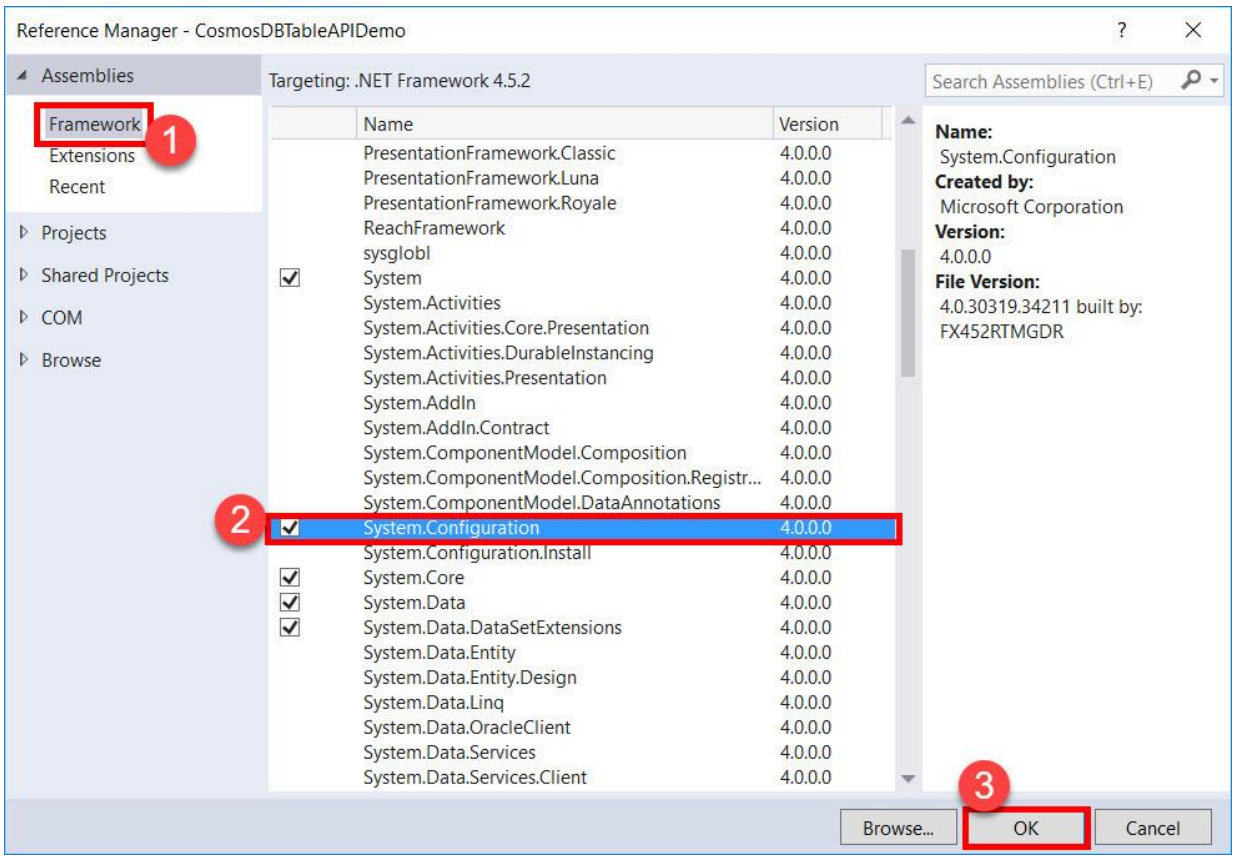
Copy Connection String of Azure Storage Account & replace **StandardStorageConnectionString** value with that.



Step 14: Right Click on **Reference -> Add References…**



Step 15: Select **System.Configuration** from the list.



Step 16: Open **Program.cs** file & add below references:

```
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Table;
using System.Diagnostics;
using System.Configuration;
```

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   using Microsoft.WindowsAzure.Storage;
8   using Microsoft.WindowsAzure.Storage.Table;
9   using System.Diagnostics;
10  using System.Configuration;
11
```

Step 17: Also declare class as **public** & add below code:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   using Microsoft.WindowsAzure.Storage;
8   using Microsoft.WindowsAzure.Storage.Table;
9   using System.Diagnostics;
10  using System.Configuration;
11
12  namespace CosmosDBTableAPIDemo
13  {
14      public class Program
15      {
16          public static void Main(string[] args)...
38
39          public void Run(CloudTableClient tableClient, int numIterations)...
176
177         public string GetRandomString(int length)...
183
184         public class CustomerEntity ...
200     }
201 }
202
```

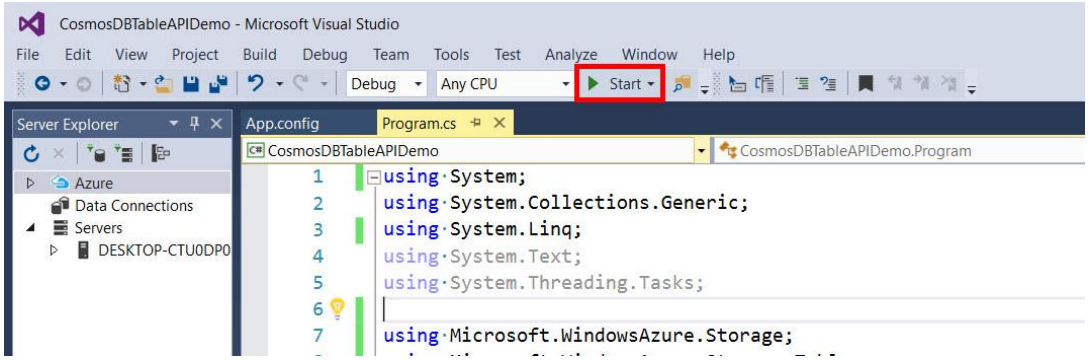
```
namespace CosmosDBTableAPIDemo
{
    public class Program
    {
        public static void Main(string[] args)
        {
            string connectionString =
ConfigurationManager.AppSettings["PremiumStorageConnectionString"];
            if (args.Length >= 1 && args[0] == "Standard")
            {
                connectionString =
ConfigurationManager.AppSettings["StandardStorageConnectionString"];
            }

            int numIterations = 100;
            if (args.Length >= 2)
            {
                numIterations = int.Parse(args[1]);
            }

            CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(connectionString);

            CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

            Program p = new Program();

            p.Run(tableClient, numIterations);
        }

        public void Run(CloudTableClient tableClient, int numIterations)
        {
            Console.WriteLine("Creating Table if it doesn't exist...");

            CloudTable table = tableClient.GetTableReference("people");
            table.CreateIfNotExists();

            List<CustomerEntity> items = new List<CustomerEntity>();
            List<double> latencies = new List<double>();
            Stopwatch watch = new Stopwatch();

            Console.WriteLine("Running inserts: ");
            for (int i = 0; i < numIterations; i++)
            {
                watch.Start();

                CustomerEntity item = new CustomerEntity()
                {
                    PartitionKey = Guid.NewGuid().ToString(),
                    RowKey = Guid.NewGuid().ToString(),
                    Email = $"{GetRandomString(6)}@contoso.com",
                    PhoneNumber = "425-555-0102",
                    Bio = GetRandomString(1000)
                };

                TableOperation insertOperation = TableOperation.Insert(item);
                table.Execute(insertOperation);
                double latencyInMs = watch.Elapsed.TotalMilliseconds;

                Console.Write($"\r\tInsert #{i + 1} completed in {latencyInMs} ms.");
                items.Add(item);
                latencies.Add(latencyInMs);
```

```
                watch.Reset();
            }

            latencies.Sort();
            Console.WriteLine($"\n\tp0:{latencies[0]}, p50:
{latencies[(int)(numIterations * 0.50)]}, p90: {latencies[(int)(numIterations *
0.90)]}. p99: {latencies[(int)(numIterations * 0.99)]}");
            Console.WriteLine("\n");

            Console.WriteLine("Running retrieves: ");
            latencies.Clear();

            for (int i = 0; i < numIterations; i++)
            {
                watch.Start();

                TableOperation retrieveOperation =
TableOperation.Retrieve<CustomerEntity>(items[i].PartitionKey, items[i].RowKey);
                table.Execute(retrieveOperation);
                double latencyInMs = watch.Elapsed.TotalMilliseconds;

                Console.Write($"\r\tRetrieve #{i + 1} completed in {latencyInMs} ms");
                latencies.Add(latencyInMs);

                watch.Reset();
            }

            latencies.Sort();
            Console.WriteLine($"\n\tp0:{latencies[0]}, p50:
{latencies[(int)(numIterations * 0.50)]}, p90: {latencies[(int)(numIterations *
0.90)]}. p99: {latencies[(int)(numIterations * 0.99)]}");
            Console.WriteLine("\n");

            Console.WriteLine("Running query against secondary index: ");
            latencies.Clear();

            for (int i = 0; i < numIterations; i++)
            {
                watch.Start();

                TableQuery<CustomerEntity> rangeQuery = new
TableQuery<CustomerEntity>().Where(
                    TableQuery.GenerateFilterCondition("Email",
QueryComparisons.Equal, items[i].Email));

                int count = 0;
                foreach (CustomerEntity entity in table.ExecuteQuery(rangeQuery))
                {
                    // Process query results
                    count++;
                }

                double latencyInMs = watch.Elapsed.TotalMilliseconds;
                Console.Write($"\r\tQuery #{i + 1} completed in {latencyInMs} ms");
                latencies.Add(latencyInMs);

                watch.Reset();
            }

            latencies.Sort();
```

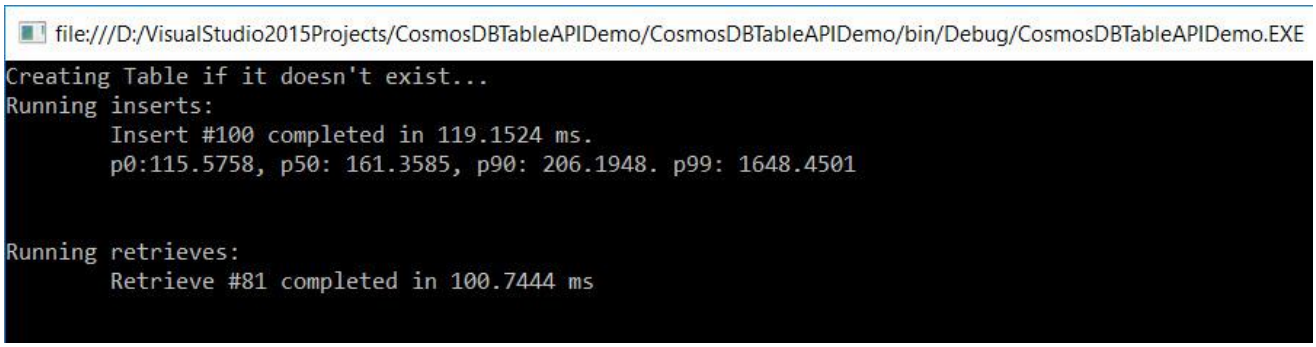
```
            Console.WriteLine($"\n\tp0:{latencies[0]}, p50:
{latencies[(int)(numIterations * 0.50)]}, p90: {latencies[(int)(numIterations *
0.90)]}. p99: {latencies[(int)(numIterations * 0.99)]}");
            Console.WriteLine("\n");

            Console.WriteLine("Running replace: ");
            latencies.Clear();

            for (int i = 0; i < numIterations; i++)
            {
                watch.Start();

                // Same latency as inserts, p99 < 15ms, and p50 < 6ms
                items[i].PhoneNumber = "425-555-5555";
                TableOperation replaceOperation = TableOperation.Replace(items[i]);
                table.Execute(replaceOperation);

                double latencyInMs = watch.Elapsed.TotalMilliseconds;
                Console.Write($"\r\tReplace #{i + 1} completed in {latencyInMs} ms");
                latencies.Add(latencyInMs);

                watch.Reset();
            }

            latencies.Sort();
            Console.WriteLine($"\n\tp0:{latencies[0]}, p50:
{latencies[(int)(numIterations * 0.50)]}, p90: {latencies[(int)(numIterations *
0.90)]}. p99: {latencies[(int)(numIterations * 0.99)]}");
            Console.WriteLine("\n");

            Console.WriteLine("Running deletes: ");
            latencies.Clear();

            for (int i = 0; i < numIterations; i++)
            {
                watch.Start();

                // Same latency as inserts, p99 < 15ms, and p50 < 6ms
                TableOperation deleteOperation = TableOperation.Delete(items[i]);
                table.Execute(deleteOperation);

                double latencyInMs = watch.Elapsed.TotalMilliseconds;
                Console.Write($"\r\tDelete #{i + 1} completed in {latencyInMs} ms");
                latencies.Add(latencyInMs);

                watch.Reset();
            }

            latencies.Sort();
            Console.WriteLine($"\n\tp0:{latencies[0]}, p50:
{latencies[(int)(numIterations * 0.50)]}, p90: {latencies[(int)(numIterations *
0.90)]}. p99: {latencies[(int)(numIterations * 0.99)]}");
            Console.WriteLine("\n");

            Console.WriteLine("Press any key to exit...");
            Console.ReadLine();
        }

        public string GetRandomString(int length)
        {
            Random random = new Random(System.Environment.TickCount);
            string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
```

```
            return new string(Enumerable.Repeat(chars, length).Select(s =>
s[random.Next(s.Length)]).ToArray());
        }

        public class CustomerEntity : TableEntity
        {
            public CustomerEntity(string lastName, string firstName)
            {
                this.PartitionKey = lastName;
                this.RowKey = firstName;
            }

            public CustomerEntity() { }

            public string Email { get; set; }

            public string PhoneNumber { get; set; }

            public string Bio { get; set; }
        }
    }
}
```

Step 18: Now run the Project.

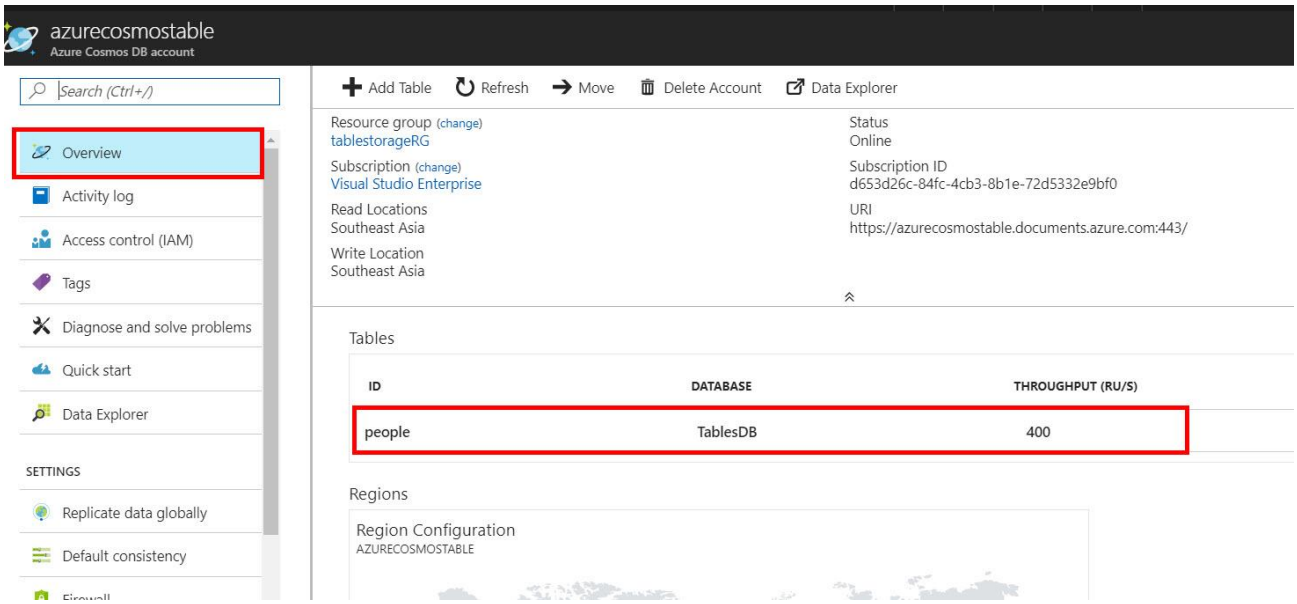


100 fake record will insert to the table. Other operations will perform like retrieve, second indexes & deletion.



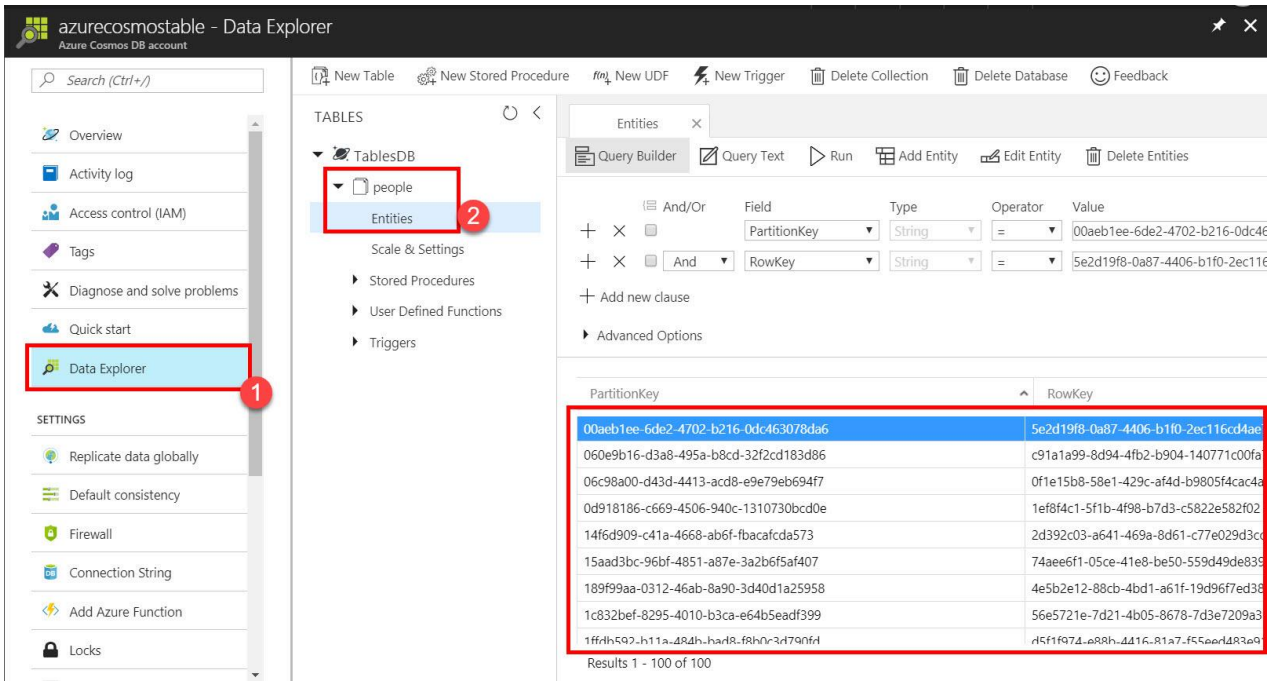
Note: When Insert option completed please navigate to Azure Portal & Open Azure CosmosDB Table to check the record.

Azure CosmosDB -> Overview -> Click on people table name



Data Explorer -> Open people table & click on **Entities** option.

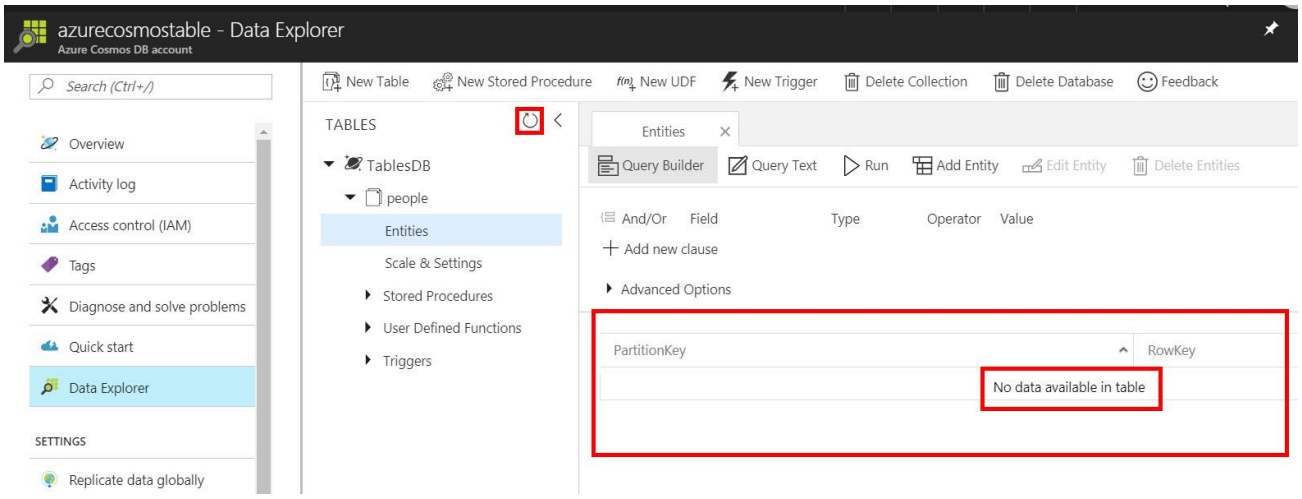
100 fake records listed as below:



Step 19: After few seconds you can see running deletes operation also performed successfully.

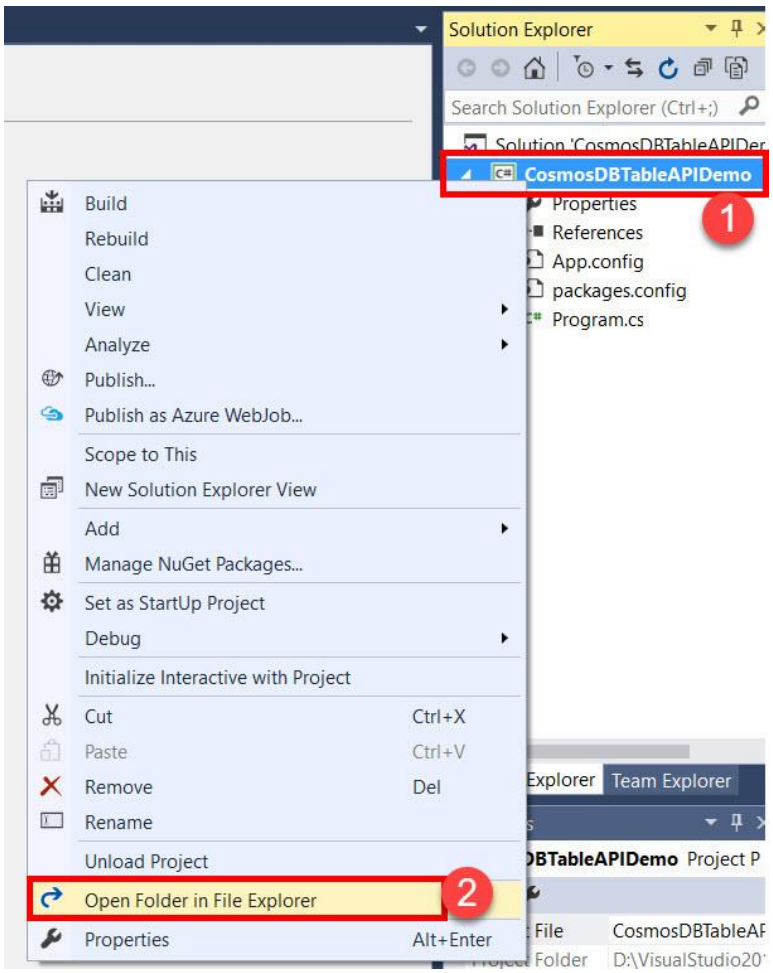
```
file:///D:/VisualStudio2015Projects/CosmosDBTableAPIDemo/CosmosDBTableAPIDemo/bin/Debug/CosmosDBTableAPIDemo.EXE

Creating Table if it doesn't exist...
Running inserts:
        Insert #100 completed in 119.1524 ms.
        p0:115.5758, p50: 161.3585, p90: 206.1948. p99: 1648.4501


Running retrieves:
        Retrieve #100 completed in 101.6587 ms
        p0:99.1999, p50: 102.8125, p90: 107.9822. p99: 591.2665


Running query against secondary index:
        Query #100 completed in 397.5491 ms
        p0:338.9635, p50: 405.1568, p90: 567.8768. p99: 1098.5045


Running replace:
        Replace #100 completed in 122.5509 ms
        p0:113.9689, p50: 136.5769, p90: 205.8461. p99: 251.7804


Running deletes:
        Delete #100 completed in 105.9409 ms
        p0:103.0015, p50: 106.0301, p90: 114.7203. p99: 226.5397


Press any key to exit...
```

It will delete all the records.



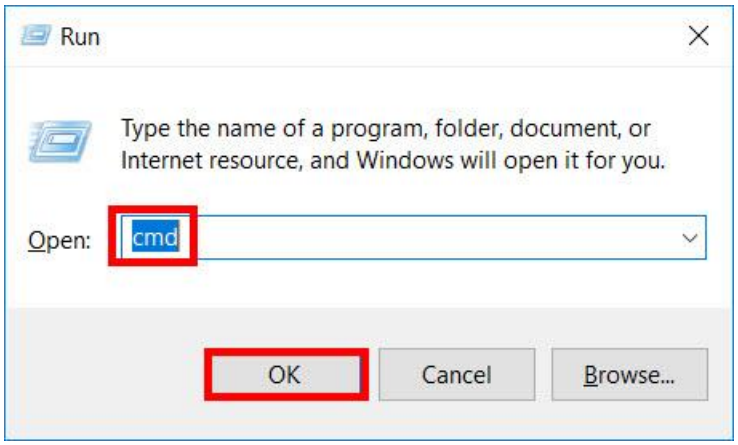
Step 20: Right click on **Project Name -> Click on Open Folder in File Explorer**.

File Explorer will open with Project path & please note down that path



Step 21: Open **Run** or Press **Windows button + R**.

Type **cmd** & hit enter key or click on **OK** button.



Step 22:

Navigate to **Project Path** & add **bin\Debug** folder.

Ex. **D:\Projects\CosmosDBTableAPIDemo\CosmosTableAPIDemo\bin\Debug>**

Now type **CosmosDBTableAPIDemo.exe Premium 100**

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\RomilBheda>d:

D:\>cd VisualStudio2015Projects

D:\VisualStudio2015Projects>cd CosmosDBTableAPIDemo

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo>cd CosmosDBTableAPIDemo

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo\CosmosDBTableAPIDemo>cd bin

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo\CosmosDBTableAPIDemo\bin>cd Debug

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo\CosmosDBTableAPIDemo\bin\Debug>CosmosDBTableAPIDemo.exe Premium 100
```

Step 23: Open another Command Prompt windows & again navigate to same path.

Now type **CosmosDBTableAPIDemo.exe Standard 100**

```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\RomilBheda>d:

D:\>cd VisualStudio2015

D:\VisualStudio2015>cd..

D:\>cd VisualStudio2015Projects

D:\VisualStudio2015Projects>cd CosmosDBTableAPIDemo

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo>cd CosmosDBTableAPIDemo

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo\CosmosDBTableAPIDemo>cd bin

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo\CosmosDBTableAPIDemo\bin>cd Debug

D:\VisualStudio2015Projects\CosmosDBTableAPIDemo\CosmosDBTableAPIDemo\bin\Debug>CosmosDBTableAPIDemo.exe Standard 100
```

Step 24: So parallel both queries are executing so you can see Azure CosmosDB Table API with Premium works faster compare to Standard or Azure Storage Table.