**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 3**

по дисциплине «Введение в облачные вычисления»

на тему: «Структурированное хранилище данных *Windows Azure Table*»

Выполнил: студент гр. ИТП-31

Коркуц С. И.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2020

**Цель работы:** изучить процедуры настройки хранилища данных *Windows Azure Table* и принципов работы с ним в *Visual Studio* при помощи *Compute Emulator*.

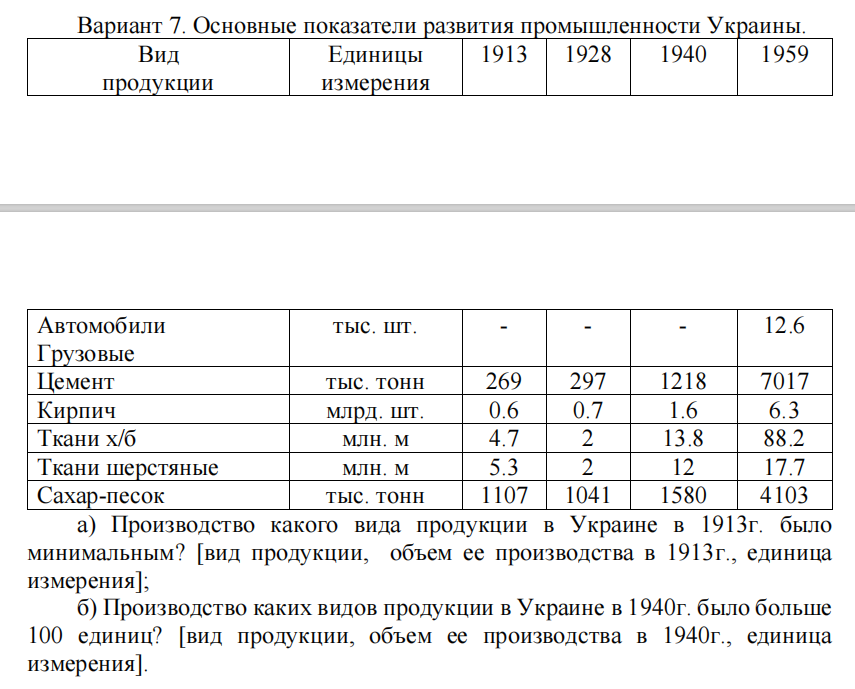
**Задание**

1. С использованием *Storage Emulator* выполнить настройку хранилища данных *Windows Azure Table*.

2. При помощи *Visual Studio* обеспечить и протестировать удаленное подключение к хранилищу.

3. Согласно своему номеру варианта (рис. 1) выбрать из приложения *А* таблицу с исходными данными. Секционировать исходные данные таким образом, чтобы образовалось несколько секций данных.

4. При помощи *Visual Studio* создать хранилище с простой структурой данных согласно индивидуальному заданию (рис. 1) и добавить в него заданные исходные данные.



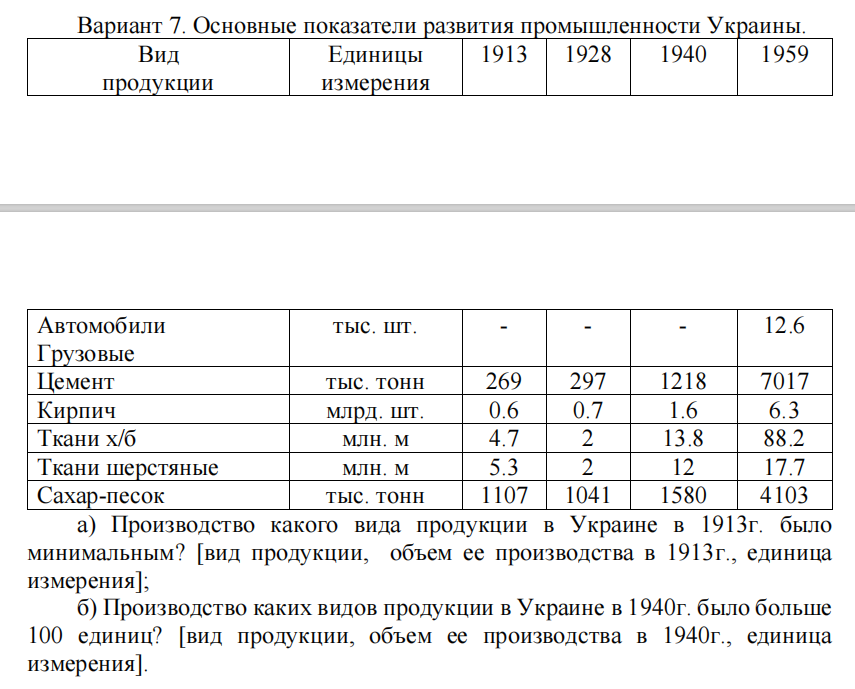


Рисунок 1 – Вариант задания

**Ход работы**

1. При помощи *Visual Studio* обеспечено и протестировано подключение к хранилищу.



Рисунок 2 – Результат подключения к хранилищу

2. Согласно своему номеру варианта (рис. 1) из приложения *А* выбрана таблица с исходными данными. Секционированы исходные данные таким образом, чтобы образовалось несколько секций данных. Хранилище было заполнено в коде программы.

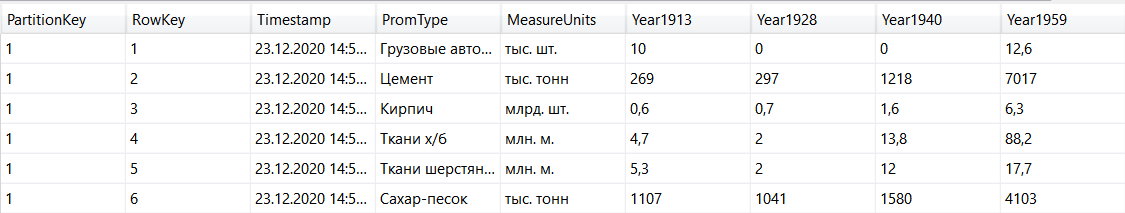


Рисунок 3 – Результат секционирования данных в хранилище

1. Были написаны запросы к хранилищу по варианту задания. Результат выполнения программы на рисунке 4.

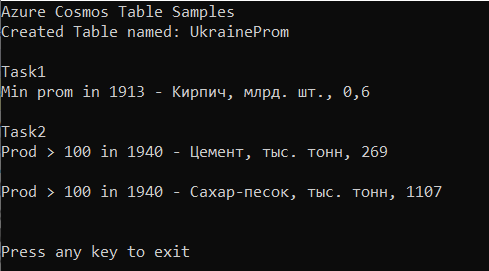


Рисунок 4 – Результат выполнения программы

**Вывод:** в результате лабораторной работы были изучены процедуры настройки хранилища данных *Windows Azure Table* и принципов работы с ним в *Visual Studio* при помощи *Azure Storage Emulator*.

**ПРИЛОЖЕНИЕ А**

**Листинг созданных классов**

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.Azure.Cosmos.Table;

namespace CosmosTableSamples.Model

{

class UkraineProm : TableEntity

{

public UkraineProm()

{

}

public UkraineProm(string partition, string index)

{

PartitionKey = partition;

RowKey = index;

}

public string PromType { get; set; }

public string MeasureUnits { get; set; }

public double Year1913 { get; set; }

public double Year1928 { get; set; }

public double Year1940 { get; set; }

public double Year1959 { get; set; }

}

}

namespace CosmosTableSamples

{

using Microsoft.Extensions.Configuration;

public class AppSettings

{

public string StorageConnectionString { get; set; }

public static AppSettings LoadAppSettings()

{

IConfigurationRoot configRoot = new ConfigurationBuilder()

.AddJsonFile("Settings.json")

.Build();

AppSettings appSettings = configRoot.Get<AppSettings>();

return appSettings;

}

}

}

using System;

namespace CosmosTableSamples

{

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.Azure.Cosmos.Table;

using Model;

class BasicSamples

{

public async Task RunSamples()

{

string tableName = "UkraineProm";

// Create or reference an existing table

CloudTable table = await Common.CreateTableAsync(tableName);

try

{

// Demonstrate basic CRUD functionality

await BasicDataOperationsAsync(table);

}

finally

{

// Delete the table

// await table.DeleteIfExistsAsync();

}

}

private static async Task BasicDataOperationsAsync(CloudTable table)

{

List<UkraineProm> proms = new List<UkraineProm>();

proms.Add(new UkraineProm("1", "1")

{

PromType = "Грузовые автомобили",

MeasureUnits = "тыс. шт.",

Year1913 = 10,

Year1928 = 0,

Year1940 = 0,

Year1959 = 12.6

});

proms.Add(new UkraineProm("1", "2")

{

PromType = "Цемент",

MeasureUnits = "тыс. тонн",

Year1913 = 269,

Year1928 = 297,

Year1940 = 1218,

Year1959 = 7017

});

proms.Add(new UkraineProm("1", "3")

{

PromType = "Кирпич",

MeasureUnits = "млрд. шт.",

Year1913 = 0.6,

Year1928 = 0.7,

Year1940 = 1.6,

Year1959 = 6.3

});

proms.Add(new UkraineProm("1", "4")

{

PromType = "Ткани х/б",

MeasureUnits = "млн. м.",

Year1913 = 4.7,

Year1928 = 2,

Year1940 = 13.8,

Year1959 = 88.2

});

proms.Add(new UkraineProm("1", "5")

{

PromType = "Ткани шерстянные",

MeasureUnits = "млн. м.",

Year1913 = 5.3,

Year1928 = 2,

Year1940 = 12,

Year1959 = 17.7

});

proms.Add(new UkraineProm("1", "6")

{

PromType = "Сахар-песок",

MeasureUnits = "тыс. тонн",

Year1913 = 1107,

Year1928 = 1041,

Year1940 = 1580,

Year1959 = 4103

});

foreach (UkraineProm prom in proms)

{

await SamplesUtils.InsertOrMergeEntityAsync(table, prom);

}

//Task 1

Console.WriteLine("Task1");

double minPromCount1913 = double.MaxValue;

UkraineProm minProm = null;

for(int i = 1; i < 7; i++)

{

var ukProm = await SamplesUtils.RetrieveEntityUsingPointQueryAsync(table, "1", i.ToString());

if(ukProm.Year1913 < minPromCount1913)

{

minPromCount1913 = ukProm.Year1913;

minProm = ukProm;

}

}

Console.WriteLine($"Min prom in 1913 - {minProm.PromType}, {minProm.MeasureUnits}, {minProm.Year1913}\n");

//Task 2

Console.WriteLine("Task2");

for(int i = 1; i < 7; i++)

{

var ukProm = await SamplesUtils.RetrieveEntityUsingPointQueryAsync(table, "1", i.ToString());

if(ukProm.Year1940 > 100)

{

Console.WriteLine($"Prod > 100 in 1940 - {ukProm.PromType}, {ukProm.MeasureUnits}, {ukProm.Year1913}\n");

}

}

}

}

}

// <createStorageAccount>

using System;

namespace CosmosTableSamples

{

using System.Threading.Tasks;

using Microsoft.Azure.Cosmos.Table;

using Microsoft.Azure.Documents;

public class Common

{

public static CloudStorageAccount CreateStorageAccountFromConnectionString(string storageConnectionString)

{

CloudStorageAccount storageAccount;

try

{

storageAccount = CloudStorageAccount.Parse(storageConnectionString);

}

catch (FormatException)

{

Console.WriteLine("Invalid storage account information provided. Please confirm the AccountName and AccountKey are valid in the app.config file - then restart the application.");

throw;

}

catch (ArgumentException)

{

Console.WriteLine("Invalid storage account information provided. Please confirm the AccountName and AccountKey are valid in the app.config file - then restart the sample.");

Console.ReadLine();

throw;

}

return storageAccount;

}

// </createStorageAccount>

// <CreateTable>

public static async Task<CloudTable> CreateTableAsync(string tableName)

{

string storageConnectionString = AppSettings.LoadAppSettings().StorageConnectionString;

// Retrieve storage account information from connection string.

CloudStorageAccount storageAccount = CreateStorageAccountFromConnectionString(storageConnectionString);

// Create a table client for interacting with the table service

CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new TableClientConfiguration());

// Create a table client for interacting with the table service

CloudTable table = tableClient.GetTableReference(tableName);

if (await table.CreateIfNotExistsAsync())

{

Console.WriteLine("Created Table named: {0}", tableName);

}

else

{

Console.WriteLine("Table {0} already exists", tableName);

}

Console.WriteLine();

return table;

}

// </CreateTable>

public static async Task<CloudTable> CreateTableAsync(CloudTableClient tableClient, string tableName)

{

Console.WriteLine("Create a Table for the demo");

// Create a table client for interacting with the table service

CloudTable table = tableClient.GetTableReference(tableName);

try

{

if (await table.CreateIfNotExistsAsync())

{

Console.WriteLine("Created Table named: {0}", tableName);

}

else

{

Console.WriteLine("Table {0} already exists", tableName);

}

}

catch (StorageException)

{

Console.WriteLine(

"If you are running with the default configuration please make sure you have started the storage emulator. Press the Windows key and type Azure Storage to select and run it from the list of applications - then restart the sample.");

Console.ReadLine();

throw;

}

catch (Exception e)

{

Console.WriteLine(e);

}

Console.WriteLine();

return table;

}

}

}

using System;

namespace CosmosTableSamples

{

class Program

{

public static void Main(string[] args)

{

Console.WriteLine("Azure Cosmos Table Samples");

BasicSamples basicSamples = new BasicSamples();

basicSamples.RunSamples().Wait();

Console.WriteLine();

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

Console.Read();

}

}

}

using System;

namespace CosmosTableSamples

{

using System.Threading.Tasks;

using Microsoft.Azure.Cosmos.Table;

using Model;

class SamplesUtils

{

// <QueryData>

public static async Task<UkraineProm> RetrieveEntityUsingPointQueryAsync(CloudTable table, string partitionKey, string rowKey)

{

try

{

TableOperation retrieveOperation = TableOperation.Retrieve<UkraineProm>(partitionKey, rowKey);

TableResult result = await table.ExecuteAsync(retrieveOperation);

UkraineProm ukProm = result.Result as UkraineProm;

return ukProm;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

// </QueryData>

// <InsertItem>

public static async Task<UkraineProm> InsertOrMergeEntityAsync(CloudTable table, UkraineProm entity)

{

if (entity == null)

{

throw new ArgumentNullException("entity");

}

try

{

// Create the InsertOrReplace table operation

TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(entity);

// Execute the operation.

TableResult result = await table.ExecuteAsync(insertOrMergeOperation);

UkraineProm insertedProm = result.Result as UkraineProm;

return insertedProm;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

/// <summary>

/// Check if given connection string is for Azure Table storage or Azure CosmosDB Table.

/// </summary>

/// <returns>true if azure cosmosdb table</returns>

public static bool IsAzureCosmosdbTable()

{

string storageConnectionString = AppSettings.LoadAppSettings().StorageConnectionString;

return !String.IsNullOrEmpty(storageConnectionString) && (storageConnectionString.Contains("table.cosmosdb") || storageConnectionString.Contains("table.cosmos"));

}

}

}