**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 4**

по дисциплине «Введение в облачные вычисления»

на тему: «Хранилище *Windows Azure Table* с реляционной структурой»

Выполнил: студент гр. ИТП-31

Коркуц С. И.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2020

**Цель работы:** освоить способы разработки реляционных структур на базе *Windows Azure Table*.

**Задание**

Согласно варианту разработать структуру из 3 связанных таблиц согласно тематике, соответствующей варианту задания.

Вариант задания:



**Ход работы**

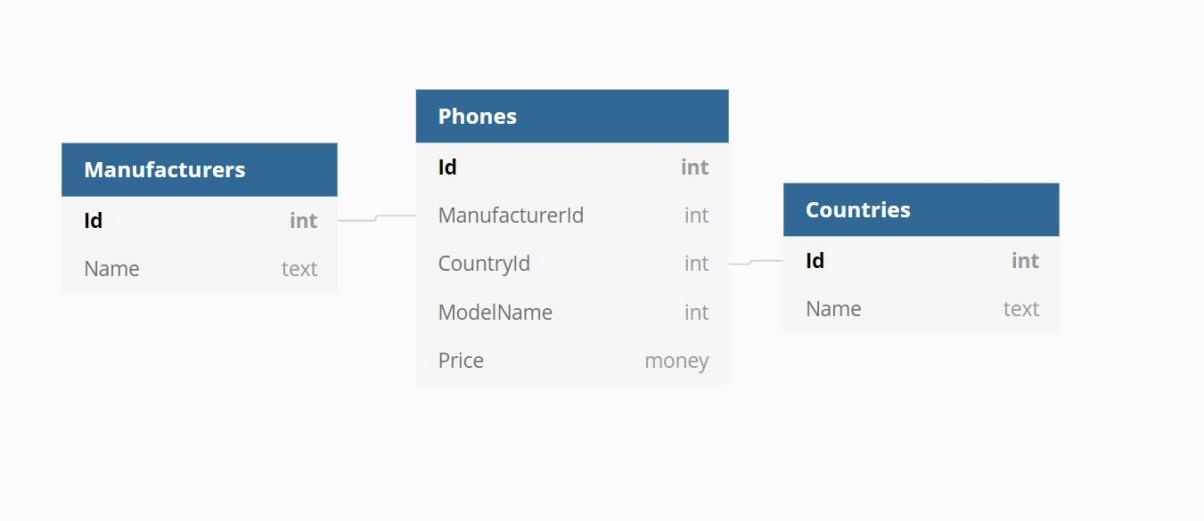
****

Рисунок 1 – Схема базы данных

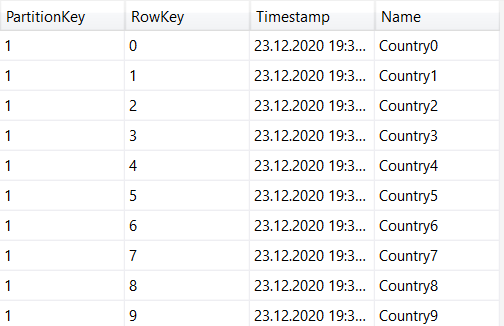


Рисунок 2 – Таблица «*Countries*» в хранилище *Windows Azure Table*

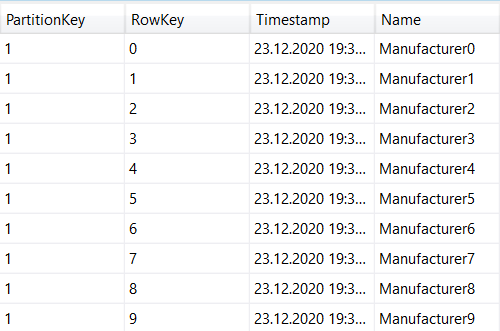


Рисунок 3 – Таблица «*Manufactures*» в хранилище *Windows Azure Table*



Рисунок 3 – Таблица «*Phones*» в хранилище *Windows Azure Table*

**Вывод:** в результате лабораторной работы был освоен способ разработки реляционной структуры на базе *Windows Azure Table*.

**ПРИЛОЖЕНИЕ А**

**Листинг созданных классов**

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.Azure.Cosmos.Table;

namespace CosmosTableSamples.Models

{

public class Country : TableEntity

{

public string Name { get; set; }

public Country()

{

}

public Country(int id)

{

PartitionKey = "1";

RowKey = id.ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.Azure.Cosmos.Table;

namespace CosmosTableSamples.Models

{

public class Manufacturer : TableEntity

{

public string Name { get; set; }

public Manufacturer()

{

}

public Manufacturer(int id)

{

PartitionKey = "1";

RowKey = id.ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.Azure.Cosmos.Table;

namespace CosmosTableSamples.Models

{

public class Phone : TableEntity

{

public int ManufacturerId { get; set; }

public int CountryId { get; set; }

public string ModelName { get; set; }

public int Price { get; set; }

public Phone()

{

}

public Phone(int id)

{

PartitionKey = "1";

RowKey = id.ToString();

}

}

}

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;

}

}

}

// <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);

await table.CreateIfNotExistsAsync();

return table;

}

}

}

using CosmosTableSamples.Models;

using Microsoft.Azure.Cosmos.Table;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

namespace CosmosTableSamples

{

public class EntityOperations

{

public static async Task<Manufacturer> GetManufacturerAsync(CloudTable table, int id)

{

string partitionKey = "1";

try

{

TableOperation retrieveOperation = TableOperation.Retrieve<Manufacturer>(partitionKey, id.ToString());

TableResult result = await table.ExecuteAsync(retrieveOperation);

Manufacturer manufacturer = result.Result as Manufacturer;

return manufacturer;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public List<Manufacturer> GetManufacturers(CloudTable table)

{

TableQuery<Manufacturer> manufacturersQuery = new TableQuery<Manufacturer>();

List<Manufacturer> manufacturers = new List<Manufacturer>();

foreach(Manufacturer manufacturer in table.ExecuteQuery(manufacturersQuery))

{

manufacturers.Add(manufacturer);

}

return manufacturers;

}

public async Task<Manufacturer> InsertOrMergeManufacturerAsync(CloudTable table, Manufacturer manufacturer)

{

try

{

// Create the InsertOrReplace table operation

TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(manufacturer);

// Execute the operation.

TableResult result = await table.ExecuteAsync(insertOrMergeOperation);

Manufacturer insertedManufacturer = result.Result as Manufacturer;

return insertedManufacturer;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

private async Task<Manufacturer> DeleteManufacturerAsync(CloudTable table, Manufacturer manufacturer)

{

try

{

// Create the InsertOrReplace table operation

TableOperation deleteOperation = TableOperation.Delete(manufacturer);

// Execute the operation.

TableResult result = await table.ExecuteAsync(deleteOperation);

Manufacturer deletedManufacturer = result.Result as Manufacturer;

return deletedManufacturer;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public async Task CascadeDeleteManufacturer(CloudTable phonesTable, CloudTable manufacturersTable, Manufacturer manufacturer)

{

try

{

var phones = GetPhones(phonesTable);

foreach(Phone phone in phones)

{

if(phone.ManufacturerId == int.Parse(manufacturer.RowKey))

{

await DeletePhoneAsync(phonesTable, phone);

}

}

await DeleteManufacturerAsync(manufacturersTable, manufacturer);

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public static async Task<Country> GetCountryAsync(CloudTable table, int id)

{

string partitionKey = "1";

try

{

TableOperation retrieveOperation = TableOperation.Retrieve<Country>(partitionKey, id.ToString());

TableResult result = await table.ExecuteAsync(retrieveOperation);

Country country = result.Result as Country;

return country;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public List<Country> GetCountries(CloudTable table)

{

TableQuery<Country> countriesQuery = new TableQuery<Country>();

List<Country> countries = new List<Country>();

foreach (Country country in table.ExecuteQuery(countriesQuery))

{

countries.Add(country);

}

return countries;

}

public async Task<Country> InsertOrMergeCountryAsync(CloudTable table, Country country)

{

try

{

// Create the InsertOrReplace table operation

TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(country);

// Execute the operation.

TableResult result = await table.ExecuteAsync(insertOrMergeOperation);

Country insertedCountry = result.Result as Country;

return insertedCountry;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

private async Task<Country> DeleteCountryAsync(CloudTable table, Country country)

{

try

{

// Create the InsertOrReplace table operation

TableOperation deleteOperation = TableOperation.Delete(country);

// Execute the operation.

TableResult result = await table.ExecuteAsync(deleteOperation);

Country deletedCountry = result.Result as Country;

return deletedCountry;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public async Task CascadeDeleteCountry(CloudTable phonesTable, CloudTable countriesTable, Country country)

{

try

{

var phones = GetPhones(phonesTable);

foreach (Phone phone in phones)

{

if (phone.CountryId == int.Parse(country.RowKey))

{

await DeletePhoneAsync(phonesTable, phone);

}

}

await DeleteCountryAsync(countriesTable, country);

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public static async Task<Phone> GetPhoneAsync(CloudTable table, int id)

{

string partitionKey = "1";

try

{

TableOperation retrieveOperation = TableOperation.Retrieve<Phone>(partitionKey, id.ToString());

TableResult result = await table.ExecuteAsync(retrieveOperation);

Phone phone = result.Result as Phone;

return phone;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public List<Phone> GetPhones(CloudTable table)

{

TableQuery<Phone> phonesQuery = new TableQuery<Phone>();

List<Phone> phones = new List<Phone>();

foreach (Phone phone in table.ExecuteQuery(phonesQuery))

{

phones.Add(phone);

}

return phones;

}

public async Task<Phone> InsertOrMergePhoneAsync(CloudTable table, Phone phone)

{

try

{

// Create the InsertOrReplace table operation

TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(phone);

// Execute the operation.

TableResult result = await table.ExecuteAsync(insertOrMergeOperation);

Phone insertedPhone = result.Result as Phone;

return insertedPhone;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

public async Task<Phone> DeletePhoneAsync(CloudTable table, Phone phone)

{

try

{

// Create the InsertOrReplace table operation

TableOperation deleteOperation = TableOperation.Delete(phone);

// Execute the operation.

TableResult result = await table.ExecuteAsync(deleteOperation);

Phone deletedPhone = result.Result as Phone;

return deletedPhone;

}

catch (StorageException e)

{

Console.WriteLine(e.Message);

Console.ReadLine();

throw;

}

}

}

}

using CosmosTableSamples.Models;

using Microsoft.Azure.Cosmos.Table;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

namespace CosmosTableSamples

{

public class StorageOperations

{

public async Task<CloudTable> CreateTableAsync(string tableName)

{

return await Common.CreateTableAsync(tableName);

}

}

}

*Settings.json*

{

"StorageConnectionString": "UseDevelopmentStorage=true"

}