**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 2**

по дисциплине:

«Разработка приложений баз данных для информационных систем»

на тему:

«Использование *ENTITY FRAMEWORK* и *LINQ* для работы с базами данных.»

Вариант 2

Выполнила: студентка гр. ИТП-31

Гайкевич Т. Г.

Принял: доцент

Асенчик О. Д.

Гомель 2020

**Цель работы:**

Ознакомиться с возможностями *ENTITY FRAMEWORK* и получить навыки написания *LINQ* запросов к объектам, связанным с таблицами базы данных СУБД *MS SQL* сервер.

**Задание:**

3.1. Создать с использованием *.NET Core Entity Framework Core* консольное приложение, содержащее набор классов, моделирующих предметную область соответствующей своему варианту и ранее созданную и заполненной тестовыми данными задания базой *MS SQL Server*. Для этого необходимо создать:

* Классы, моделирующие не менее чем три таблицы базы данных согласно вашему варианту.
* Класс контекста данных.

3.2. Выполнить, используя объекты *Entity Framework Core* и *LINQ*:

1. Выборку всех данных из таблицы, стоящей в схеме базы данных нас стороне отношения «один» – 1 шт.
2. Выборку данных из таблицы, стоящей в схеме базы данных на стороне отношения «один», отфильтрованные по определенному условию, налагающему ограничения на одно или несколько полей – 1 шт.
3. Выборку данных, сгруппированных по любому из полей данных с выводом какого-либо итогового результата (*min*, *max*, *avg*, *сount* или др.) по выбранному полю из таблицы, стоящей в схеме базы данных нас стороне отношения «многие» – 1 шт.
4. Выборку данных из двух полей двух таблиц, связанных между собой отношением «один-ко-многим» – 1 шт.
5. Выборку данных из двух таблиц, связанных между собой отношением «один-ко-многим» и отфильтрованным по некоторому условию, налагающему ограничения на значения одного или нескольких полей – 1 шт.
6. Вставку данных в таблицы, стоящей на стороне отношения «Один» – 1 шт.
7. Вставку данных в таблицы, стоящей на стороне отношения «Многие» – 1 шт.:
8. Удаление данных из таблицы, стоящей на стороне отношения «Один» – 1 шт.
9. Удаление данных из таблицы, стоящей на стороне отношения «Многие» – 1 шт.
10. Обновление удовлетворяющих определенному условию записей в любой из таблиц базы данных – 1 шт.

**Вариант задания:**

Предметная область задания изображена на рисунке 1.

**Изображение выглядит как снимок экрана

Автоматически созданное описание**

Рисунок 1 – Предметная область

**Ход работы:**

**Задание 1**

Первым делом была открыта консоль управления пакетами. Там была прописана команда для автоматического создания моделей в *C#* приложении из готовой базы данных. Непосредственно сами команды:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet ef dbcontext Scaffold "Server=<servername>,1433;Initial Catalog=<dbName>;Persist Security Info=False;User ID=<userID>;Password=<password>;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;" Microsoft.EntityFrameworkCore.SqlServer

Далее все классы были переименованы в единственное число и распределены по папкам. Структура созданного консольного приложения изображена на рисунке 2.

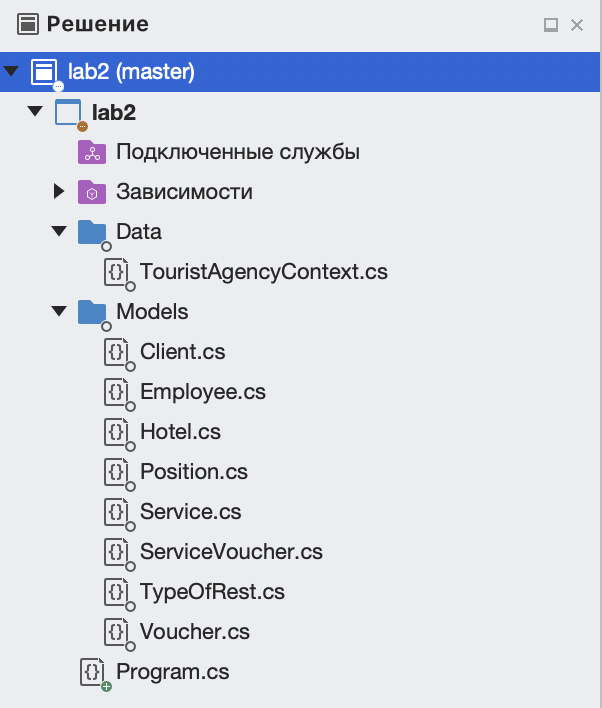


Рисунок 2 – Структура созданного приложения

**Задание 2**

Были созданы 10 *LINQ* запросов к базе данных. Код всех созданных запросов находится в классе *Program.cs* (Приложение А)консольного приложения. Каждому запросу соответствует отдельный метод. Результат выполнения запросов на рисунках 3 – 6.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Результат выполнения первого запроса

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Результат выполнения второго запроса



Рисунок 5 – Результат выполнения запросов 3 – 5

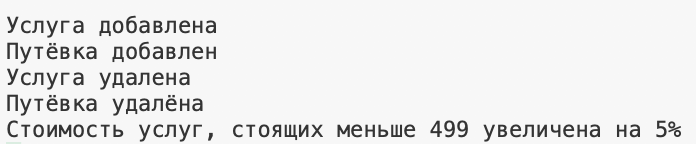


Рисунок 6 – Результат выполнения запросов 6 – 10

**Вывод:**

При помощи *Reverse Engineering* по существующей базе данныхбыли созданы классы, описывающие предметную область приложения. Они были переименованы и распределены по папкам. Далее было написано 10 *LINQ* запросов для всех *CRUD* операций к базе данных.

**ПРИЛОЖЕНИЕ А**

Листинг созданных классов

*Program.cs*

using System;

using lab2.Model;

using System.Collections;

using System.Linq;

using System.ComponentModel;

namespace lab2

{

class Program

{

static TouristAgencyContext db;

static void Main(string[] args)

{

db = new TouristAgencyContext();

Problem1(db);

Problem2(db);

Problem3(db);

Problem4(db);

Problem5(db);

Problem6(db);

Problem7(db);

Problem8(db);

Problem9(db);

Problem10(db);

}

//Для вывода коллекции

static void Print(string sqltext, IEnumerable items)

{

Console.WriteLine(sqltext);

Console.WriteLine("Записи: ");

foreach (var item in items)

{

Console.WriteLine(item.ToString());

}

Console.WriteLine();

}

static void Problem1(TouristAgencyContext db)

{

var clients = db.Clients

.Select(client => client);

Print("Вывод всех данныз из таблицы Clients", clients.Take(5).ToList());

}

static void Problem2(TouristAgencyContext db)

{

var hotels = db.Hotels

.Where(h => h.NumberOfStars < 4 && h.NumberOfStars > 0)

.Select(hotel => hotel);

Print("Результат выборки из таблицы Hotels.\n" +

"Ограничение на поле NumberOfStars (< 4).", hotels.Take(5).ToList());

}

static void Problem3(TouristAgencyContext db)

{

var sevises = db.ServicesVouchers

.GroupBy(c => c.Service.Cost).Average(c => c.Key);

Console.WriteLine($"Средняя стоимость всех услуг {sevises}");

}

static void Problem4(TouristAgencyContext db)

{

var sevicesVouchersPlusServices = db.ServicesVouchers

.Join(db.Services, serviceVoucher => serviceVoucher.Id, service => service.Id,

(serviceVoucher, service) => new

{

serviceVoucher.ReservationMark,

serviceVoucher.PaymentMark,

service.Name,

service.Description,

service.Cost

})

;

Print("Результат выборки из таблиц ServicesVouchers и Services.", sevicesVouchersPlusServices.Take(10).ToList());

}

static void Problem5(TouristAgencyContext db)

{

var vouchersPlusClients = db.Vouchers

.Join(db.Clients, voucher => voucher.Id, client => client.Id,

(voucher, client) => new

{

voucher.Name,

voucher.StartDate,

voucher.EndDate,

client.Surname,

client.Birthday,

Gender = client.Gender ? "Мужчина" : "Женщина",

client.Address,

client.Phone,

client.PassportData

})

.Where(client => client.Gender == "Женщина")

;

Print("Результат выборки из таблиц Vouchers и Clients с фильтром по полю Gender. " +

"Вывести клиентов одного пола", vouchersPlusClients.Take(10).ToList());

}

static void Problem6(TouristAgencyContext db)

{

Service service = new Service

{

Name = "TestService",

Description = "TestServiceDecription",

Cost = 777

};

db.Services.Add(service);

db.SaveChanges();

Console.WriteLine("Услуга добавлена");

}

static void Problem7(TouristAgencyContext db)

{

Random rand = new Random();

Hotel hotel = db.Hotels.ToList().ElementAt(rand.Next(0, db.Hotels.Count() - 1));

Employee employee = db.Employees.ToList().ElementAt(rand.Next(0, db.Employees.Count() - 1));

Client client = db.Clients.ToList().ElementAt(rand.Next(0, db.Clients.Count() - 1));

TypeOfRest typeOfRest = db.TypesOfRest.ToList().ElementAt(rand.Next(0, db.TypesOfRest.Count() - 1));

Voucher voucher = new Voucher

{

Name = "TestVoucher",

StartDate = DateTime.Now,

EndDate = DateTime.Now,

Hotel = hotel,

HotelId = hotel.Id,

TypeOfRest = typeOfRest,

TypeOfRestId = typeOfRest.Id,

Employee = employee,

EmployeeId = employee.Id,

Client = client,

ClientId = client.Id

};

db.Vouchers.Add(voucher);

db.SaveChanges();

Console.WriteLine("Путёвка добавлен");

}

static void Problem8(TouristAgencyContext db)

{

db.Services.Remove(db.Services.ToList()[db.Services.Count() - 1]);

db.SaveChanges();

Console.WriteLine("Услуга удалена");

}

static void Problem9(TouristAgencyContext db)

{

db.Vouchers.Remove(db.Vouchers.ToList()[db.Vouchers.Count() - 1]);

db.SaveChanges();

Console.WriteLine("Путёвка удалёна");

}

static void Problem10(TouristAgencyContext db)

{

var costServices = db.Services.Where(service => service.Cost < 499);

foreach (var service in costServices)

{

service.Cost \*= (decimal)1.05;

}

db.SaveChanges();

Console.WriteLine("Стоимость услуг, стоящих меньше 499 увеличена на 5%");

}

}

}

*TouristAgencyContext.cs*

using Microsoft.EntityFrameworkCore;

namespace lab2.Model

{

public partial class TouristAgencyContext : DbContext

{

public TouristAgencyContext()

{

}

public virtual DbSet<Client> Clients { get; set; }

public virtual DbSet<Employee> Employees { get; set; }

public virtual DbSet<Hotel> Hotels { get; set; }

public virtual DbSet<Position> Positions { get; set; }

public virtual DbSet<Service> Services { get; set; }

public virtual DbSet<ServiceVoucher> ServicesVouchers { get; set; }

public virtual DbSet<TypeOfRest> TypesOfRest { get; set; }

public virtual DbSet<Voucher> Vouchers { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer("Server=tcp:touristagency.database.windows.net,1433;Initial Catalog=TouristAgency;Persist Security Info=False;User ID=tgaykevich;Password=Tg9798316;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30");

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Client>(entity =>

{

entity.Property(e => e.Address)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.Birthday).HasColumnType("date");

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.PassportData)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.Patronymic)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.Surname)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

});

modelBuilder.Entity<Employee>(entity =>

{

entity.Property(e => e.Birthday).HasColumnType("date");

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.Patronymic)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.Surname)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

});

modelBuilder.Entity<Hotel>(entity =>

{

entity.Property(e => e.Address)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.ContactPerson)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.Country)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.HotelPhoto)

.HasMaxLength(100)

.IsUnicode(false);

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.RoomFoto)

.HasMaxLength(100)

.IsUnicode(false);

entity.Property(e => e.Town)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

});

modelBuilder.Entity<Position>(entity =>

{

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

});

modelBuilder.Entity<Service>(entity =>

{

entity.Property(e => e.Cost).HasColumnType("money");

entity.Property(e => e.Description)

.IsRequired()

.HasMaxLength(150)

.IsUnicode(false);

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

});

modelBuilder.Entity<ServiceVoucher>(entity =>

{

entity.Property(e => e.PaymentMark)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.ReservationMark)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.HasOne(d => d.Service)

.WithMany(p => p.ServiceVoucher)

.HasForeignKey(d => d.ServiceId)

.HasConstraintName("FK\_\_ServicesV\_\_Servi\_\_40F9A68C");

entity.HasOne(d => d.Voucher)

.WithMany(p => p.ServiceVoucher)

.HasForeignKey(d => d.VoucherId)

.HasConstraintName("FK\_\_ServicesV\_\_Vouch\_\_40058253");

});

modelBuilder.Entity<TypeOfRest>(entity =>

{

entity.Property(e => e.Description)

.IsRequired()

.HasMaxLength(150)

.IsUnicode(false);

entity.Property(e => e.Limitation)

.IsRequired()

.HasMaxLength(100)

.IsUnicode(false);

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

});

modelBuilder.Entity<Voucher>(entity =>

{

entity.Property(e => e.EndDate).HasColumnType("date");

entity.Property(e => e.Name)

.IsRequired()

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.StartDate).HasColumnType("date");

entity.HasOne(d => d.Client)

.WithMany(p => p.Voucher)

.HasForeignKey(d => d.ClientId)

.HasConstraintName("FK\_\_Vouchers\_\_Client\_\_44CA3770");

entity.HasOne(d => d.Employee)

.WithMany(p => p.Voucher)

.HasForeignKey(d => d.EmployeeId)

.HasConstraintName("FK\_\_Vouchers\_\_Employ\_\_43D61337");

entity.HasOne(d => d.Hotel)

.WithMany(p => p.Voucher)

.HasForeignKey(d => d.HotelId)

.HasConstraintName("FK\_\_Vouchers\_\_HotelI\_\_41EDCAC5");

entity.HasOne(d => d.TypeOfRest)

.WithMany(p => p.Voucher)

.HasForeignKey(d => d.TypeOfRestId)

.HasConstraintName("FK\_\_Vouchers\_\_TypeOf\_\_42E1EEFE");

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

}

*Client.cs*

using System;

using System.Collections.Generic;

namespace lab2.Model

{

public partial class Client

{

public Client()

{

Voucher = new HashSet<Voucher>();

}

public int Id { get; set; }

public string Surname { get; set; }

public string Name { get; set; }

public string Patronymic { get; set; }

public DateTime Birthday { get; set; }

public bool Gender { get; set; }

public string Address { get; set; }

public int Phone { get; set; }

public string PassportData { get; set; }

public virtual ICollection<Voucher> Voucher { get; set; }

public override string ToString()

{

return $"Клиент по фамилии {Surname};\nпо имени: {Name}\n";

}

}

}

*Employee.cs*

using System;

using System.Collections.Generic;

namespace lab2.Model

{

public partial class Hotel

{

public Hotel()

{

Voucher = new HashSet<Voucher>();

}

public int Id { get; set; }

public string Name { get; set; }

public string Country { get; set; }

public string Town { get; set; }

public string Address { get; set; }

public int Phone { get; set; }

public int NumberOfStars { get; set; }

public string ContactPerson { get; set; }

public string HotelPhoto { get; set; }

public string RoomFoto { get; set; }

public virtual ICollection<Voucher> Voucher { get; set; }

public override string ToString()

{

return $"Название отеля:{Name};\nКоличество звёзд: {NumberOfStars}\n";

}

}

}

*Position.cs*

using System;

using System.Collections.Generic;

namespace lab2.Model

{

public partial class Position

{

public int Id { get; set; }

public string Name { get; set; }

}

}

*Servece.cs*

using System;

using System.Collections.Generic;

namespace lab2.Model

{

public partial class Service

{

public Service()

{

ServiceVoucher = new HashSet<ServiceVoucher>();

}

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public decimal Cost { get; set; }

public virtual ICollection<ServiceVoucher> ServiceVoucher { get; set; }

}

}

*ServiceVoucher.cs*

namespace lab2.Model

{

public partial class ServiceVoucher

{

public int Id { get; set; }

public string ReservationMark { get; set; }

public string PaymentMark { get; set; }

public int VoucherId { get; set; }

public int ServiceId { get; set; }

public virtual Service Service { get; set; }

public virtual Voucher Voucher { get; set; }

}

}

*TypeOfRest.cs*

using System;

using System.Collections.Generic;

namespace lab2.Model

{

public partial class TypeOfRest

{

public TypeOfRest()

{

Voucher = new HashSet<Voucher>();

}

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public string Limitation { get; set; }

public virtual ICollection<Voucher> Voucher { get; set; }

}

}

*Voucher.cs*

using System;

using System.Collections.Generic;

namespace lab2.Model

{

public partial class Voucher

{

public Voucher()

{

ServiceVoucher = new HashSet<ServiceVoucher>();

}

public int Id { get; set; }

public string Name { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public int HotelId { get; set; }

public int TypeOfRestId { get; set; }

public int EmployeeId { get; set; }

public int ClientId { get; set; }

public virtual Client Client { get; set; }

public virtual Employee Employee { get; set; }

public virtual Hotel Hotel { get; set; }

public virtual TypeOfRest TypeOfRest { get; set; }

public virtual ICollection<ServiceVoucher> ServiceVoucher { get; set; }

}

}