**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

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

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ЛАБОРАТОРНАЯ РАБОТА №2**

по дисциплине: «Разработка приложений баз данных для информационных систем»

на тему: «Использование *ENTITY FRAMEWORK* и *LINQ* для работы с базами данных»

Выполнил: студент гр. ИТП-31

Ермолин Д. К.

Принял: ректор

Асенчик О.Д.

Гомель 2023

**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:**

1.1. 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.

1.2. 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 нас 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.

1.3. Разместить выполненный проект на *github*.

**Ход работы**

Для работы с базой данных при помощи *ENTITY FRAMEWORK* были установлены необходимые *NuGET* пакеты:

1. *Microsoft.EntityFrameworkCore.Design ­*­– Предоставляет интерфейсы для связи между базой данных и *C*#.
2. *Microsoft.EntityFrameworkCore.SqlServer* – Предоставляет функционал для связи ORM фреймворка с базой данных *MS SQL Server.*
3. *Microsoft.EntityFrameworkCore.Tools* – Предоставляет список команд для работы с миграциями, переносом созданной базы в виде объектов в *SQL* и переносу *SQL* базы данных в объекты *C*#.

После установки всех необходимых пакетов приложение готово для работы с базой данных через *ENTITY FRAMEWORK*. В основе *ENTITY FRAMEWORK* лежит технология *ORM* (*object-relational mapping* – отображения данных на реальные объекты) которая позволят связать таблицы баз данных с *C*# объектами. Для связи объектов по определенным правилам существует три подхода.

Первый запрос выполняет выборку данных из таблицы на стороне отношения один. Он достает данные из таблицы контракты страховых агентов. Листинг данного метода *AllEmployees* указан в приложении А. Пример работы этого запроса указан на рисунке 1.



Рисунок 1 – Пример запроса на выборку данных из таблицы на стороне отношения один

Далее был реализован запрос на выборку данных из таблицы на стороне отношения один с последующей фильтрацией данных. Для этого был реализован метод *EmployeesBelowAge* который находит сотрудников с возрастом меньше 30. Листинг данного метода *EmployeesBelowAge* указан в приложении А. Пример работы этого запроса указан на рисунке 2.

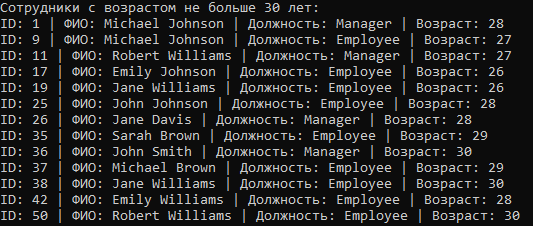


Рисунок 2 – Пример запроса на выборку данных из таблицы на стороне отношения один с фильтрацией данных

Был разработан запрос на выборку данных из таблицы стоящую на стороне отношения многие и агрегирующие полученный результат. В данном случае был разработан метод *GroupedServicesByPrice* который выводит информацию о ценах дополнительных услуг и выводит их количество с одинаковой ценой. Листинг данного метода *GroupedServicesByPrice* указан в приложении А. Пример работы этого запроса указан на рисунке 3.

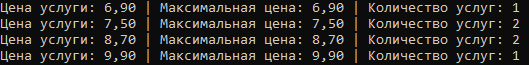


Рисунок 3 – Пример запроса на выборку данных из таблицы на стороне отношения многие с последующей агрегацией данных

Далее был разработан запрос на выборку данных, стоящую на стороне отношения один-многие с выводом данных из обоих таблиц. Для этого был реализован метод *ServiceVouchers* который выбирает данные из таблиц *Voucher* и *AdditionalService.* На экран будет выведено дополнительная услуга из таблицы *AdditionalService* и начало с концом этой услуги из таблицы *Voucher*. Листинг данного метода *ServiceVouchers* указан в приложении А. Пример работы этого запроса указан на рисунке 4.

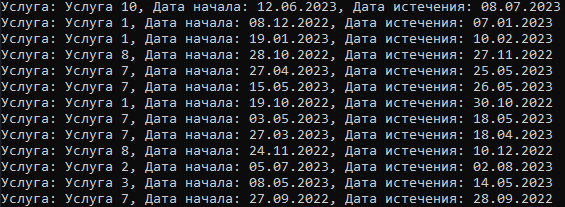


Рисунок 4 – Пример запроса на выборку данных из таблиц на стороне отношения один-многие

В дальнейшем был разработан запрос на выборку данных из таблиц, стоящих на стороне отношения один-многие с последующей фильтрацией данных. Для этого был разработан метод *VouchersWithHighValueDeals*. В данном методе выборка производится из таблиц *Voucher* и *AdditionalService.* И фильтрация производится по полю *Price* в таблице *AdditionalService* и по полю *Id* в таблице *Voucher*. В параметры метода передается цена больше которого должны быть поля с *Price* в таблице *AdditionalService.* Листинг данного метода *VouchersWithHighValueDeals* указан в приложении А. Пример работы этого запроса указан на рисунке 5.

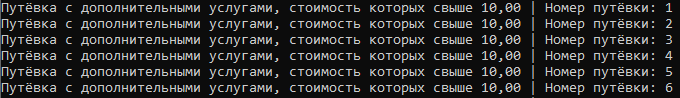


Рисунок 5 – Пример запроса на выборку данных из таблиц на стороне отношения один-многие с последующей фильтрацией данных

Далее был разработан запрос на добавления данных в таблицу, стоящую на стороне отношения один. Для этого был разработан метод *AddNewClient* который добавляет данные в таблицу *Client*. Для добавления данных *SQL* базу в *ENTITY FRAMEWORK* достаточно добавить объект в нужный *DbSet* и у объекта контекста вызвать метод *SaveChanges*. Листинг *InsertOneRelation* указан в приложении А. Пример работы этого запроса указан на рисунке 6.



Рисунок 6 – Пример запроса на вставку данных в таблицу на стороне отношения один

После добавления нового объекта в *DbSet* и вызова метода *SaveChanges.* Добавленный данные сохранятся в *SQL* базе данных. Пример добавленных данных указан на рисунке 7.



Рисунок 7 – Пример добавленных данных

Далее был разработан запрос на добавления данных в таблицу, стоящую на стороне отношения многие. Для этого был разработан метод *AddNewVoucher* который добавляет данные в таблицу *Voucher* в качестве параметров метод принимает объект класса *Voucher*. Листинг данного метода *AddNewVoucher* указан в приложении А. Пример работы этого запроса указан на рисунке 8.



Рисунок 8 – Пример запроса на вставку данных в таблицу на стороне отношения многие

После добавления нового объекта в *DbSet* и вызова метода *SaveChanges.* Добавленный данные сохранятся в *SQL* базе данных. Пример добавленных данных указан на рисунке 9.



Рисунок 9 – Пример добавленных данных

Далее был реализован запрос для удаления данных из таблицы, стоящей на стороне отношения многие. Для этого был реализован метод *DeleteClient* который удаляет данные из таблицы *Client* и в качестве параметра принимает объект класса *Client.* Для удаления данных достаточно вызвать метод *Remove* для объекта.Листинг данного метода *DeleteClient* указан в приложении А. Пример работы этого запроса указан на рисунке 10.



Рисунок 10 – Пример удаления данных на стороне отношения один

Потом был реализован запрос на удаления данных из таблицы, стоящей на стороне отношения многие. Для этого был реализован метод *DeleteVoucher* который удаляет данные в таблице. Если этот класс содержит ссылки на другие объекты базы данных, то они удаляются согласно правилам, заданным при помощи атрибутов*.* Листинг данного метода *DeleteVoucher* указан в приложении А. Пример работы этого запроса указан на рисунке 11.



Рисунок 11 – Пример удаления данных на стороне отношения многие

Далее был разработан запрос на обновления данных в таблице. Для этого был разработан метод *UpdateAdditionalServicePrices* который принимает в качестве параметра *Price*, у которых заработная плата меньше переданной в аргументы метода, то цена увеличивается на 1,2*.* Листинг данного метода *UpdateAdditionalServicePrices* указан в приложении А. Пример работы этого запроса указан на рисунке 12.



Рисунок 12 – Пример обновления данных в базе данных

Чтобы посмотреть код, который был написан для проекта можно по ссылке [*https://github.com/Javaro3/lab2\_DB*](https://github.com/Javaro3/lab2_DB)*.*

**Вывод**: в процессе выполнения лабораторной работы была изучена технология *Entity* *Framework*, которая позволяет взаимодействовать с базой данных, используя объекты на языке программирования *C#.* Кроме того, для работы с объектно-реляционным отображением данных (*ORM*) была применена технология *LINQ*, которая предоставляет удобный способ работы с данными. В ходе выполнения лабораторной работы была выполнена задача по переносу базы данных, созданной на первой лабораторной работе, в среду *C*#.

**ПРИЛОЖНИЕ А**

Листинг класса *AdditionalService*

using System;

using System.Collections.Generic;

namespace lab2.Models;

public partial class AdditionalService

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public string Description { get; set; } = null!;

public decimal Price { get; set; }

public virtual ICollection<Voucher> Vouchers { get; set; } = new List<Voucher>();

}

Листинг класса *Client*

using System;

using System.Collections.Generic;

namespace lab2.Models;

public partial class Client

{

public int Id { get; set; }

public string Fio { get; set; } = null!;

public DateTime DateOfBirth { get; set; }

public string Sex { get; set; } = null!;

public string Address { get; set; } = null!;

public string Series { get; set; } = null!;

public long Number { get; set; }

public long Discount { get; set; }

public virtual ICollection<Voucher> Vouchers { get; set; } = new List<Voucher>();

}

Листинг класса *Employee*

using System;

using System.Collections.Generic;

namespace lab2.Models;

public partial class Employee

{

public int Id { get; set; }

public string Fio { get; set; } = null!;

public string JobTitle { get; set; } = null!;

public int Age { get; set; }

public virtual ICollection<Voucher> Vouchers { get; set; } = new List<Voucher>();

}

Листинг класса *Hotel*

using System;

using System.Collections.Generic;

namespace lab2.Models;

public partial class Hotel

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public string Country { get; set; } = null!;

public string City { get; set; } = null!;

public string Address { get; set; } = null!;

public string Phone { get; set; } = null!;

public int Stars { get; set; }

public string TheContactPerson { get; set; } = null!;

public byte[] Photo { get; set; } = null!;

public virtual ICollection<Voucher> Vouchers { get; set; } = new List<Voucher>();

}

Листинг класса *TypesOfRecreation*

using System;

using System.Collections.Generic;

namespace lab2.Models;

public partial class TypesOfRecreation

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public string Description { get; set; } = null!;

public string Restrictions { get; set; } = null!;

public virtual ICollection<Voucher> Vouchers { get; set; } = new List<Voucher>();

}

Листинг класса *Voucher*

using System;

using System.Collections.Generic;

namespace lab2.Models;

public partial class Voucher

{

public int Id { get; set; }

public DateTime StartDate { get; set; }

public DateTime ExpirationDate { get; set; }

public int HotelId { get; set; }

public int TypeOfRecreationId { get; set; }

public int AdditionalServiceId { get; set; }

public int ClientId { get; set; }

public int EmployessId { get; set; }

public bool Reservation { get; set; }

public bool Payment { get; set; }

public virtual AdditionalService AdditionalService { get; set; } = null!;

public virtual Client Client { get; set; } = null!;

public virtual Employee Employess { get; set; } = null!;

public virtual Hotel Hotel { get; set; } = null!;

public virtual TypesOfRecreation TypeOfRecreation { get; set; } = null!;

}

Листинг класса *HotelView*

using System;

using System.Collections.Generic;

namespace lab2.Views;

public partial class HotelView

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public string Country { get; set; } = null!;

public string City { get; set; } = null!;

public string Address { get; set; } = null!;

public string Phone { get; set; } = null!;

public int Stars { get; set; }

public string TheContactPerson { get; set; } = null!;

public byte[] Photo { get; set; } = null!;

}

Листинг класса *RecreationView*

using System;

using System.Collections.Generic;

namespace lab2.Views;

public partial class RecreationView

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public string Description { get; set; } = null!;

public string Restrictions { get; set; } = null!;

}

Листинг класса *VoucherView*

using System;

using System.Collections.Generic;

namespace lab2.Views;

public partial class VoucherView

{

public int VoucherId { get; set; }

public DateTime StartDate { get; set; }

public DateTime ExpirationDate { get; set; }

public string HotelName { get; set; } = null!;

public string ClientName { get; set; } = null!;

public string EmployeeName { get; set; } = null!;

public string RecreationType { get; set; } = null!;

public string AdditionalService { get; set; } = null!;

public bool Reservation { get; set; }

public bool Payment { get; set; }

}

Листинг класса *TouristAgency1Context*

using System;

using System.Collections.Generic;

using lab2.Models;

using lab2.Views;

using Microsoft.EntityFrameworkCore;

namespace lab2.DBContext;

public partial class TouristAgency1Context : DbContext

{

public TouristAgency1Context()

{

}

public TouristAgency1Context(DbContextOptions<TouristAgency1Context> options)

: base(options)

{

}

public virtual DbSet<AdditionalService> AdditionalServices { get; set; }

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

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

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

public virtual DbSet<HotelView> HotelViews { get; set; }

public virtual DbSet<RecreationView> RecreationViews { get; set; }

public virtual DbSet<TypesOfRecreation> TypesOfRecreations { get; set; }

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

public virtual DbSet<VoucherView> VoucherViews { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

#warning To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723263.

=> optionsBuilder.UseSqlServer("Server=DESKTOP-RC1TE3C;Database=TouristAgency1;Trusted\_Connection=True; TrustServerCertificate=True;");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<AdditionalService>(entity =>

{

entity.HasKey(e => e.Id).HasName("PK\_\_Addition\_\_3214EC07BC28C93C");

entity.Property(e => e.Description).HasMaxLength(200);

entity.Property(e => e.Name).HasMaxLength(50);

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

});

modelBuilder.Entity<Client>(entity =>

{

entity.HasKey(e => e.Id).HasName("PK\_\_Clients\_\_3214EC0709F595A5");

entity.Property(e => e.Address).HasMaxLength(100);

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

entity.Property(e => e.Fio)

.HasMaxLength(150)

.HasColumnName("FIO");

entity.Property(e => e.Series).HasMaxLength(50);

entity.Property(e => e.Sex).HasMaxLength(50);

});

modelBuilder.Entity<Employee>(entity =>

{

entity.HasKey(e => e.Id).HasName("PK\_\_Employee\_\_3214EC077EB2F5F7");

entity.Property(e => e.Fio)

.HasMaxLength(150)

.HasColumnName("FIO");

entity.Property(e => e.JobTitle).HasMaxLength(50);

});

modelBuilder.Entity<Hotel>(entity =>

{

entity.HasKey(e => e.Id).HasName("PK\_\_Hotels\_\_3214EC07F9846FAA");

entity.Property(e => e.Address).HasMaxLength(100);

entity.Property(e => e.City).HasMaxLength(50);

entity.Property(e => e.Country).HasMaxLength(50);

entity.Property(e => e.Name).HasMaxLength(50);

entity.Property(e => e.Phone).HasMaxLength(20);

entity.Property(e => e.Photo).HasColumnType("image");

entity.Property(e => e.TheContactPerson).HasMaxLength(100);

});

modelBuilder.Entity<HotelView>(entity =>

{

entity

.HasNoKey()

.ToView("HotelView");

entity.Property(e => e.Address).HasMaxLength(100);

entity.Property(e => e.City).HasMaxLength(50);

entity.Property(e => e.Country).HasMaxLength(50);

entity.Property(e => e.Id).ValueGeneratedOnAdd();

entity.Property(e => e.Name).HasMaxLength(50);

entity.Property(e => e.Phone).HasMaxLength(20);

entity.Property(e => e.Photo).HasColumnType("image");

entity.Property(e => e.TheContactPerson).HasMaxLength(100);

});

modelBuilder.Entity<RecreationView>(entity =>

{

entity

.HasNoKey()

.ToView("RecreationView");

entity.Property(e => e.Description).HasMaxLength(100);

entity.Property(e => e.Id).ValueGeneratedOnAdd();

entity.Property(e => e.Name).HasMaxLength(50);

entity.Property(e => e.Restrictions).HasMaxLength(50);

});

modelBuilder.Entity<TypesOfRecreation>(entity =>

{

entity.HasKey(e => e.Id).HasName("PK\_\_TypesOfR\_\_3214EC07C89167FD");

entity.ToTable("TypesOfRecreation");

entity.Property(e => e.Description).HasMaxLength(100);

entity.Property(e => e.Name).HasMaxLength(50);

entity.Property(e => e.Restrictions).HasMaxLength(50);

});

modelBuilder.Entity<Voucher>(entity =>

{

entity.HasKey(e => e.Id).HasName("PK\_\_Vouchers\_\_3214EC07A43B24A5");

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

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

entity.HasOne(d => d.AdditionalService).WithMany(p => p.Vouchers)

.HasForeignKey(d => d.AdditionalServiceId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("FK\_Vouchers\_AdditionalServices");

entity.HasOne(d => d.Client).WithMany(p => p.Vouchers)

.HasForeignKey(d => d.ClientId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("FK\_Vouchers\_Clients");

entity.HasOne(d => d.Employess).WithMany(p => p.Vouchers)

.HasForeignKey(d => d.EmployessId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("FK\_Vouchers\_Employees");

entity.HasOne(d => d.Hotel).WithMany(p => p.Vouchers)

.HasForeignKey(d => d.HotelId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("FK\_Vouchers\_Hotels");

entity.HasOne(d => d.TypeOfRecreation).WithMany(p => p.Vouchers)

.HasForeignKey(d => d.TypeOfRecreationId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("FK\_Vouchers\_TypesOfRecreation");

});

modelBuilder.Entity<VoucherView>(entity =>

{

entity

.HasNoKey()

.ToView("VoucherView");

entity.Property(e => e.AdditionalService).HasMaxLength(50);

entity.Property(e => e.ClientName).HasMaxLength(150);

entity.Property(e => e.EmployeeName).HasMaxLength(150);

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

entity.Property(e => e.HotelName).HasMaxLength(50);

entity.Property(e => e.RecreationType).HasMaxLength(50);

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

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

Листинг класса *TouristAgency1Context*

using lab2.DBContext;

using lab2.Models;

using System;

using System.Collections.Generic;

using System.Diagnostics.Contracts;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TouristAgency

{

internal class Program

{

static void Main(string[] args)

{

while (true)

{

Console.WriteLine("Выберите действие:");

Console.WriteLine("1. Выборка всех сотрудников");

Console.WriteLine("2. Выборка сотрудников с возрастом не больше 30");

Console.WriteLine("3. Вывести группированные данные о дополнительных услугах");

Console.WriteLine("4. Вывести дополнительные услуги по датам начала и конца");

Console.WriteLine("5. Вывести данные о путёвках, где дополнительные услуги стоят свыше 10");

Console.WriteLine("6. Добавить нового клиента");

Console.WriteLine("7. Добавить путёвку");

Console.WriteLine("8. Удалить клиента");

Console.WriteLine("9. Удалить путёвку");

Console.WriteLine("10. Обновить цены на услуги");

int choice;

if (int.TryParse(Console.ReadLine(), out choice))

{

switch (choice)

{

case 1:

AllEmployees();

break;

case 2:

EmployeesBelowAge();

break;

case 3:

GroupedServicesByPrice();

break;

case 4:

ServiceVouchers();

break;

case 5:

VouchersWithHighValueDeals();

break;

case 6:

AddNewClient();

break;

case 7:

AddNewVoucher();

break;

case 8:

DeleteClient();

break;

case 9:

DeleteVoucher();

break;

case 10:

UpdateAdditionalServicePrices();

break;

default:

Console.WriteLine("Неверный выбор. Пожалуйста, выберите снова.");

break;

}

}

else

{

Console.WriteLine("Неверный ввод. Пожалуйста, выберите снова.");

}

}

}

static void AllEmployees()

{

using (var context = new TouristAgency1Context()) // Вывод данных о сотрудниках

{

var employees = context.Employees.ToList();

if (employees.Count > 0)

{

Console.WriteLine("Вся информация о сотрудниках:");

foreach (var employee in employees)

{

Console.WriteLine($"ID: {employee.Id} | ФИО: {employee.Fio} | Должность: {employee.JobTitle} | Возраст: {employee.Age}");

}

}

else

{

Console.WriteLine("Сотрудники не найдены.");

}

}

}

static void EmployeesBelowAge()

{

using (var context = new TouristAgency1Context()) // Вывод сотрудников с возрастом не больше 30

{

var selectedEmployees = context.Employees

.Where(employee => employee.Age <= 30)

.ToList();

if (selectedEmployees.Count > 0)

{

Console.WriteLine($"Сотрудники с возрастом не больше 30 лет:");

foreach (var employee in selectedEmployees)

{

Console.WriteLine($"ID: {employee.Id} | ФИО: {employee.Fio} | Должность: {employee.JobTitle} | Возраст: {employee.Age}");

}

}

else

{

Console.WriteLine($"Сотрудники с возрастом не больше 30 лет не найдены.");

}

}

}

static void GroupedServicesByPrice()

{

using (var context = new TouristAgency1Context()) // Группирует по цене дополнительных услуг

{

var groupedServices = context.AdditionalServices

.GroupBy(service => service.Price)

.Select(group => new

{

Price = group.Key,

MaxPrice = group.Max(service => service.Price),

Count = group.Count()

})

.ToList();

foreach (var group in groupedServices)

{

Console.WriteLine($"Цена услуги: {group.Price.ToString("N2")} | Максимальная цена: {group.MaxPrice.ToString("N2")} | Количество услуг: {group.Count}");

}

}

}

static void ServiceVouchers()

{

using (var context = new TouristAgency1Context()) // Вывод дополнительных услуг с их началом и концом

{

var serviceVouchers = context.AdditionalServices

.SelectMany(service => service.Vouchers, (service, voucher) => new

{

ServiceName = service.Name,

StartDate = voucher.StartDate.ToShortDateString(),

ExpirationDate = voucher.ExpirationDate.ToShortDateString()

})

.ToList();

foreach (var entry in serviceVouchers)

{

Console.WriteLine($"Услуга: {entry.ServiceName}, Дата начала: {entry.StartDate}, Дата истечения: {entry.ExpirationDate}");

}

}

}

static void VouchersWithHighValueDeals()

{

using (var context = new TouristAgency1Context()) // Вывод данных о путёвках, где дополнительные услуги стоят свыше 10

{

decimal minimumDealAmount = 10.0m;

var vouchersWithHighValueDeals = context.Vouchers

.Where(voucher =>

voucher.AdditionalService.Price > minimumDealAmount)

.ToList();

foreach (var voucher in vouchersWithHighValueDeals)

{

Console.WriteLine($"Путёвка с дополнительными услугами, стоимость которых свыше {minimumDealAmount:N2} | Номер путёвки: {voucher.Id}");

}

}

}

static void AddNewClient()

{

using (var context = new TouristAgency1Context()) // Добавление нового клиента

{

var newClient = new Client

{

Fio = "Король Василий",

DateOfBirth = new DateTime(1990, 1, 1), // Замените эту дату на дату рождения клиента

Sex = "Мужской", // Замените на пол клиента

Address = "Адрес клиента",

Series = "HB", // Замените на серию паспорта клиента

Number = 1234567890, // Замените на номер паспорта клиента

Discount = 10 // Замените на размер скидки клиента

};

context.Clients.Add(newClient);

context.SaveChanges();

Console.WriteLine("Новый клиент успешно добавлен.");

}

}

static void AddNewVoucher()

{

using (var context = new TouristAgency1Context()) // Добавление новой записи в путёвки

{

var newVoucher = new Voucher

{

StartDate = DateTime.Now,

ExpirationDate = DateTime.Now.AddDays(30),

HotelId = 1, // Замените на ID отеля

TypeOfRecreationId = 1, // Замените на ID типа рекреации

AdditionalServiceId = 1, // Замените на ID дополнительной услуги

ClientId = 1, // Замените на ID клиента

EmployessId = 1, // Замените на ID сотрудника

Reservation = true, // Замените на нужное значение

Payment = false, // Замените на нужное значение

};

context.Vouchers.Add(newVoucher); // Добавление новой путёвки в контекст базы данных

context.SaveChanges(); // Сохранение изменений

Console.WriteLine("Новый ваучер успешно создан.");

}

}

static void DeleteClient()

{

using (var context = new TouristAgency1Context()) // Удаление записи по фамилии

{

var clientToDelete = context.Clients.FirstOrDefault(c => c.Fio == "Король Василий");

if (clientToDelete != null)

{

context.Clients.Remove(clientToDelete); // Удаление клиента из контекста базы данных

context.SaveChanges(); // Сохранение изменений

Console.WriteLine("Клиент успешно удален.");

}

else

{

Console.WriteLine("Клиент не найден.");

}

}

}

static void DeleteVoucher()

{

using (var context = new TouristAgency1Context()) // Удаление записи из таблицы Voucher

{

var voucherToDelete = context.Vouchers.FirstOrDefault(v => v.Id == 51);

if (voucherToDelete != null)

{

context.Vouchers.Remove(voucherToDelete); // Удаление путёвки из контекста базы данных

context.SaveChanges(); // Сохранение изменений

Console.WriteLine("Путёвка успешна удалена.");

}

else

{

Console.WriteLine("Путёвка не найден.");

}

}

}

static void UpdateAdditionalServicePrices()

{

using (var context = new TouristAgency1Context()) // Изменение стоимости услуги

{

var servicesToUpdate = context.AdditionalServices.Where(service => service.Price < 10);

foreach (var service in servicesToUpdate)

{

service.Price \*= 1.2m;

}

context.SaveChanges();

Console.WriteLine("Цены на услуги успешно обновлены.");

}

}

}

}