МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ П. О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

дисциплина «Разработка приложений баз данных для информационных систем»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
«Использование *ENTITY FRAMEWORK* и *LINQ* для работы с базами данных»

ВАРИАНТ №12
Брачное агентство

Выполнил:
студент группы ИТИ-31, Зеленский К.А.
Принял:
доцент Асенчик О.Д.

Гомель 2024

Цель работы: ознакомиться с возможностями *ENTITY FRAMEWORK* и получить навыки написания *LINQ* запросов к объектам, связанным с таблицами базы данных СУБД *MS SQL Server*.

Ход работы и результаты.

На рисунке 1 представлен созданный *GitHub* репозиторий. *https://github.com/kevenmusic/RPBDISLabs/tree/main*.

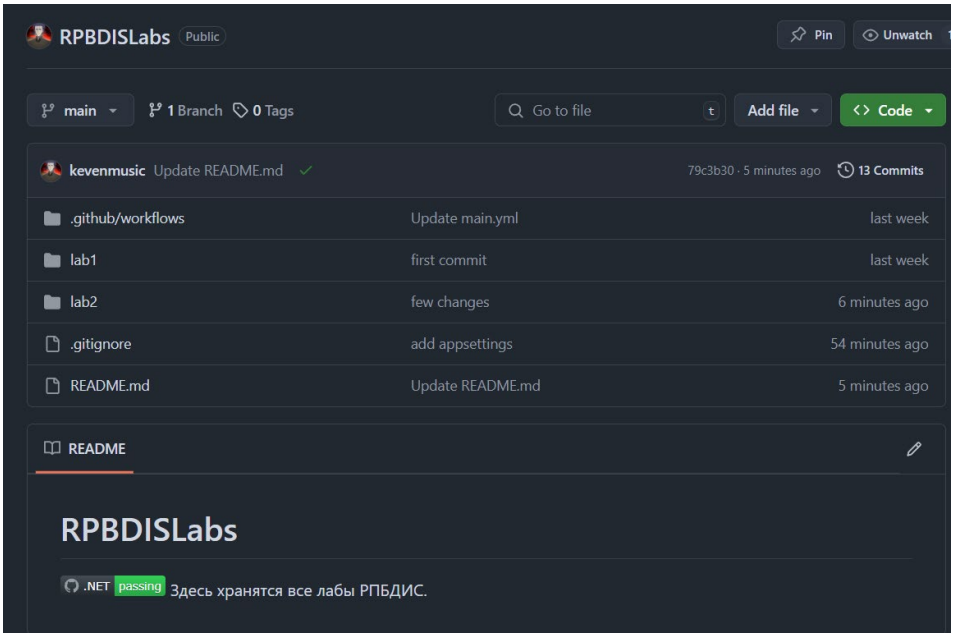

Рисунок 1 – *GitHub* Репозиторий

На рисунке 2 представлена строка подключения конфигурационного *json*-файла.

```
{
  "ConnectionStrings": {
    "SQLConnection": "Server=HOME-PC;Database=MarriageAgency; Trusted_Connection =True; TrustServerCertificate=True"
  }
}
```

Рисунок 2 – Строка подключения

На рисунке 3 представлены сгенерированные модели, контекст данных, а также запросы (Приложение А, В, Д).

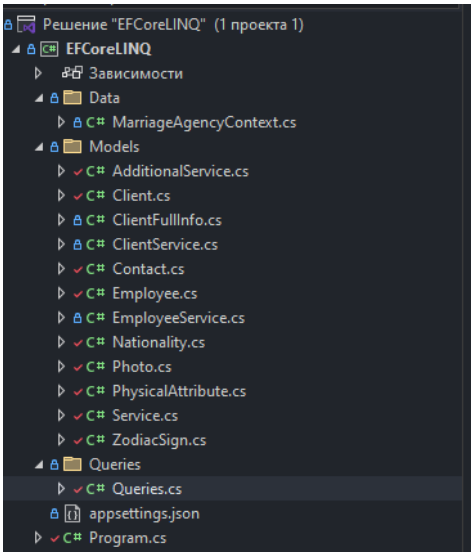

Рисунок 3 – Модели и контекст данных

На рисунке 4 представлена выборка всех данных из таблицы «Клиенты», стоящей в схеме базы данных нас стороне отношения «один» – 1 шт.

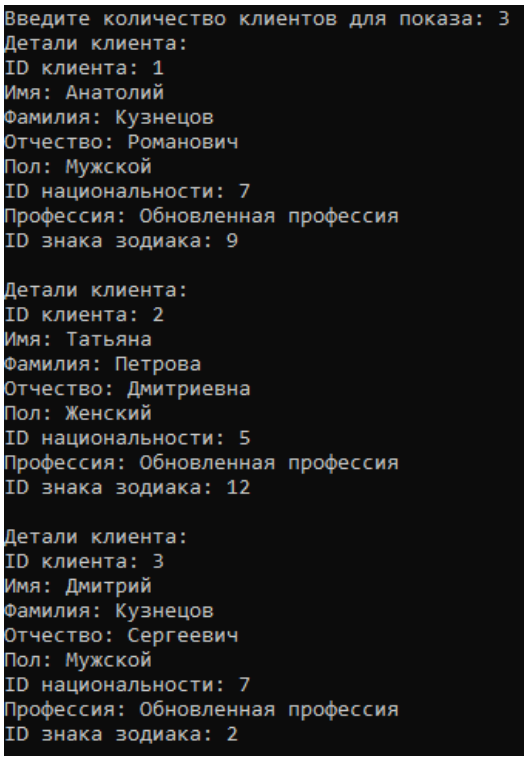

Рисунок 4 – Выборка данных 1

На рисунке 5 представлена выборка данных из таблицы «Клиенты», стоящей в схеме базы данных на стороне отношения «один», отфильтрованных по профессии, налагающему ограничения на одно или несколько полей – 1 шт.

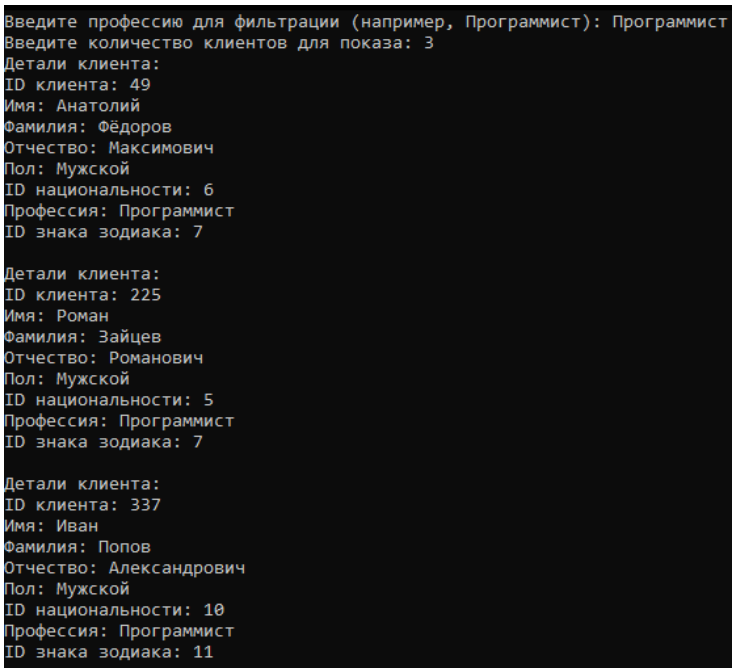

Рисунок 5 – Выборка данных 2

На рисунке 6 представлена выборку данных таблицы «Клиенты», сгруппированных по професси с выводом какого-либо итогового результата (*min, max, avg, count* или др.) по выбранному полю из таблицы, стоящей в схеме базы данных нас стороне отношения «многие» – 1 шт.

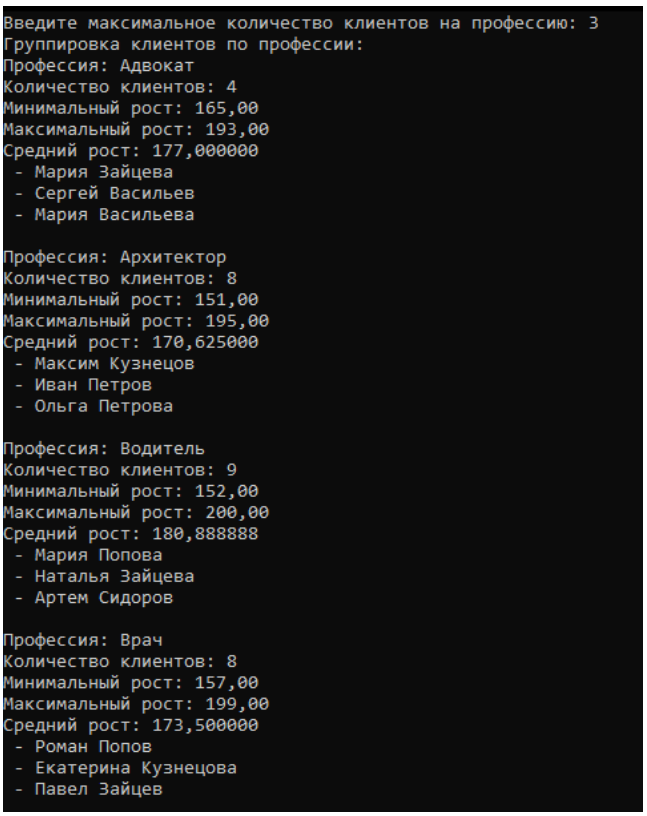

Рисунок 6 – Выборка данных 3

На рисунке 7 представлена выборка данных из двух полей двух таблиц «Услуги» и «Дополнительные услуги», связанных между собой отношением «один-ко-многим» – 1 шт.

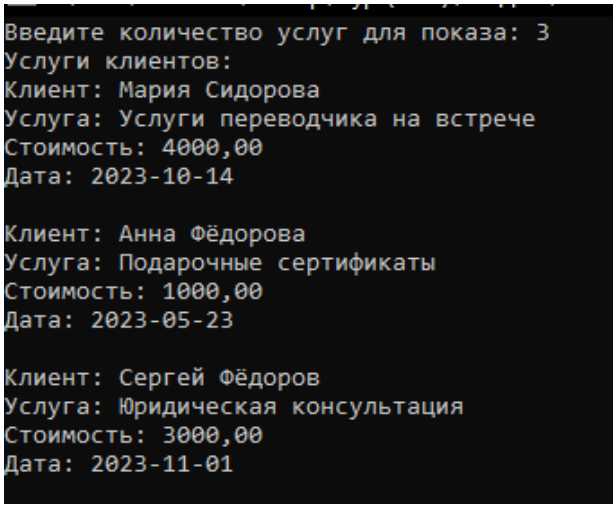

Рисунок 7 – Выборка данных 4

На рисунке 8 представлена выборка данных из двух таблиц «Клиенты» и «Национальности», связанных между собой отношением «один-ко-многим» и отфильтрованным по национальности, налагающему ограничения на значения одного или нескольких полей – 1 шт.

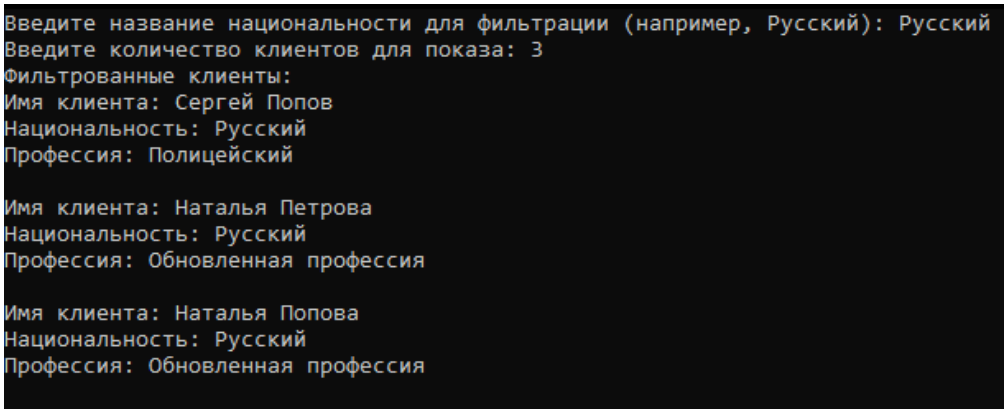

Рисунок 8 – Выборка данных 5

На рисунке 9 представлена вставка данных в таблицу «Клиенты», стоящей на стороне отношения «Один» – 1 шт.

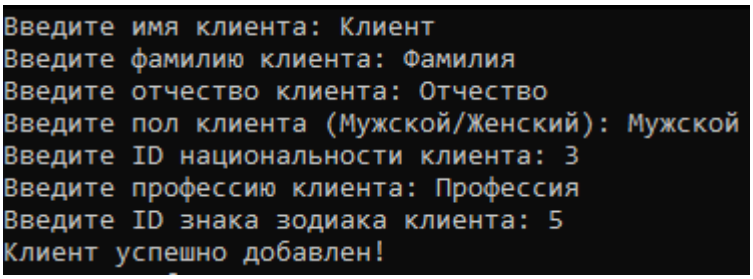

Рисунок 9 – Вставка данных 1

На рисунке 10 представлена вставка данных в таблицу «Услуги», стоящей на стороне отношения «Многие» – 1 шт.

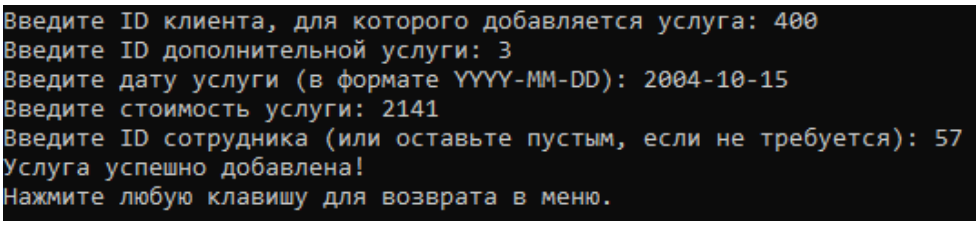

Рисунок 10 – Вставка данных 2

На рисунке 11 представлено удаление данных из таблицы «Клиенты», стоящей на стороне отношения «Один» – 1 шт.

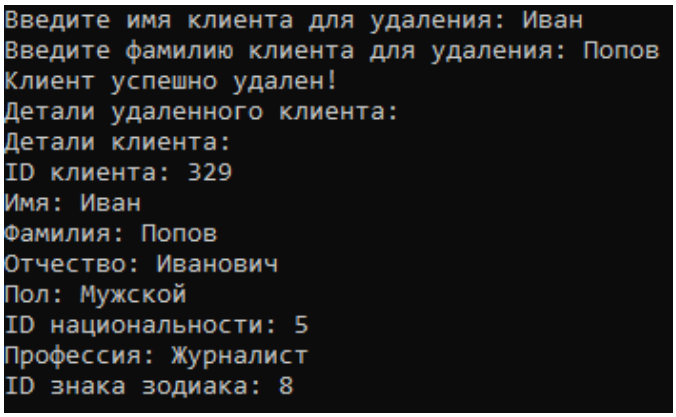

Рисунок 11 – Удаление данных 1

На рисунке 12 представлено удаление данных из таблицы «Услуги», стоящей на стороне отношения «Многие» – 1 шт.

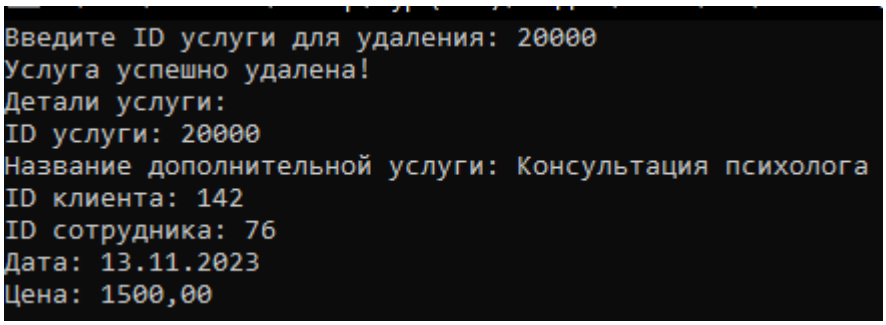

Рисунок 12 – Удаление данных 2

На рисунке 13 представлено обновление данных удовлетворяющих определенному условию записей в любой из таблиц базы данных – 1 шт.

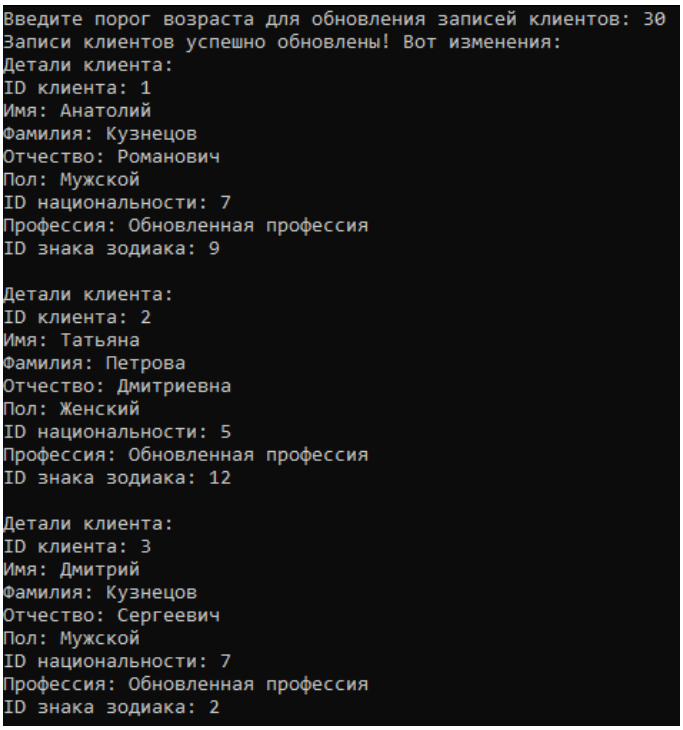

Рисунок 13 – Обновление данных

На рисунке 14 представлен рабочий процесс *GitHub Actions*, который осуществляет компиляцию проекта под две разные платформы при любом изменении в репозитории (Приложение Г).

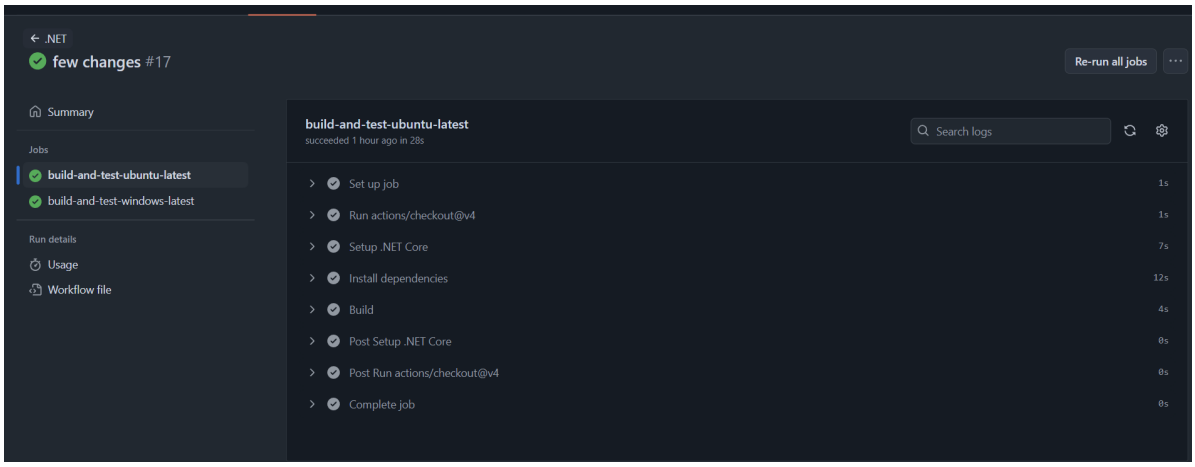

Рисунок 14 – Рабочий процесс *GitHub Actions*

На рисунке 15 представлен отредактированный *README.md* файл опубликованного проекта, вставив в него код для создания эмблемы состояния рабочего процесса (*status badge*), показывающей, чем в данный момент завершился рабочий процесс.

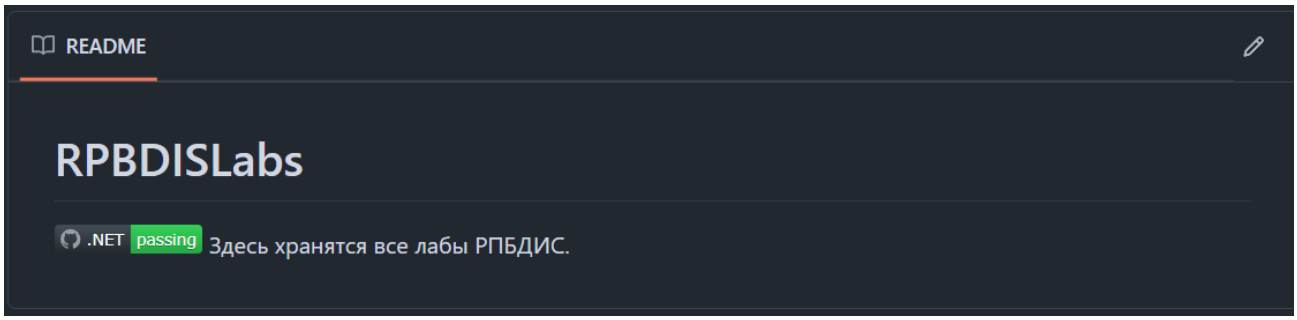

Рисунок 15 – Эмблема состояния

Выводы: был ознакомлен с возможностями *ENTITY FRAMEWORK*, а также были получены навыки написания *LINQ* запросов к объектам, связанным с таблицами базы данных СУБД *MS SQL Server*.

ПРИЛОЖЕНИЕ А
(обязательное)

Текст программы

AdditionalService.cs:
```
namespace EFCoreLINQ.Models;

public partial class AdditionalService
{
    public int AdditionalServiceId { get; set; }

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

    public string? Description { get; set; }

    public decimal Price { get; set; }

    public virtual ICollection<Service> Services { get; set; } = new List<Service>();
}
```

Client.cs:
```
namespace EFCoreLINQ.Models;

public partial class Client
{
    public int ClientId { get; set; }

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

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

    public string? MiddleName { get; set; }

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

    public DateOnly? BirthDate { get; set; }

    public int? ZodiacSignId { get; set; }

    public int? NationalityId { get; set; }

    public string? Profession { get; set; }

    public virtual Contact? Contact { get; set; }

    public virtual Nationality? Nationality { get; set; }

    public virtual Photo? Photo { get; set; }

    public virtual PhysicalAttribute? PhysicalAttribute { get; set; }

    public virtual ICollection<Service> Services { get; set; } = [];

    public virtual ZodiacSign? ZodiacSign { get; set; }
}
```

Contact.cs
```
namespace EFCoreLINQ.Models;

public partial class Contact
{
    public int ClientId { get; set; }
```

```
    public string? Address { get; set; }

    public string? Phone { get; set; }

    public string? PassportData { get; set; }

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

Employee.cs

```
namespace EFCoreLINQ.Models;

public partial class Employee
{
    public int EmployeeId { get; set; }

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

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

    public string? Position { get; set; }

    public DateOnly? BirthDate { get; set; }

    public virtual ICollection<Service> Services { get; set; } = new List<Service>();
}
```

Nationality.cs

```
namespace EFCoreLINQ.Models;

public partial class Nationality
{
    public int NationalityId { get; set; }

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

    public string? Notes { get; set; }

    public virtual ICollection<Client> Clients { get; set; } = new List<Client>();
}
```

PhysicalAttribute.cs

```
namespace EFCoreLINQ.Models;

public partial class PhysicalAttribute
{
    public int ClientId { get; set; }

    public int? Age { get; set; }

    public decimal? Height { get; set; }

    public decimal? Weight { get; set; }

    public int? ChildrenCount { get; set; }

    public string? MaritalStatus { get; set; }

    public string? BadHabits { get; set; }

    public string? Hobbies { get; set; }

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

Service.cs

```
namespace EFCoreLINQ.Models;

public partial class Service
{
    public int ServiceId { get; set; }

    public int AdditionalServiceId { get; set; }

    public int ClientId { get; set; }

    public int EmployeeId { get; set; }

    public DateOnly Date { get; set; }

    public decimal Cost { get; set; }

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

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

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

ZodiacSign.cs

```
namespace EFCoreLINQ.Models;

public partial class ZodiacSign
{
    public int ZodiacSignId { get; set; }

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

    public string? Description { get; set; }

    public virtual ICollection<Client> Clients { get; set; } = new List<Client>();
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Текст программы

Appsettings.json

```
{
  "ConnectionStrings": {
    "SQLConnection": "Server=HOME-PC;Database=MarriageAgency; Trusted_Connection =True; TrustServerCertificate=True"
  }
}
```

ПРИЛОЖЕНИЕ В
(обязательное)

Текст программы

MarriageAgencyContext.cs

```
using System;
using System.Collections.Generic;
using EFCoreLINQ.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;

namespace EFCoreLINQ.Data;

public partial class MarriageAgencyContext : DbContext
{
    public MarriageAgencyContext()
    {
    }

    public MarriageAgencyContext(DbContextOptions<MarriageAgencyContext> options)
        : base(options)
    {
    }

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

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

    public virtual DbSet<ClientFullInfo> ClientFullInfos { get; set; }

    public virtual DbSet<ClientService> ClientServices { get; set; }

    public virtual DbSet<Contact> Contacts { get; set; }

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

    public virtual DbSet<EmployeeService> EmployeeServices { get; set; }

    public virtual DbSet<Nationality> Nationalities { get; set; }

    public virtual DbSet<Photo> Photos { get; set; }

    public virtual DbSet<PhysicalAttribute> PhysicalAttributes { get; set; }

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

    public virtual DbSet<ZodiacSign> ZodiacSigns { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        ConfigurationBuilder builder = new();

        ///Установка пути к текущему каталогу
        builder.SetBasePath(Directory.GetCurrentDirectory());
        // получаем конфигурацию из файла appsettings.json
        builder.AddJsonFile("appsettings.json");
        // создаем конфигурацию
        IConfigurationRoot configuration = builder.AddUserSecrets<Program>().Build();

        /// Получаем строку подключения
        string connectionString = "";

        // Для локального SQL Server
        connectionString = configuration.GetConnectionString("SQLConnection");
```

```
    /// Задание опций подключения
    _ = optionsBuilder
        .UseSqlServer(connectionString)
        .Options;
    optionsBuilder.LogTo(message => System.Diagnostics.Debug.WriteLine(message));
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<AdditionalService>(entity =>
    {
        entity.HasKey(e => e.AdditionalServiceId).HasName("PK__Addition__F92FC43E93B718A2");

        entity.HasIndex(e => e.Name, "UQ__Addition__737584F6E2706B41").IsUnique();

        entity.Property(e => e.AdditionalServiceId).HasColumnName("AdditionalServiceID");
        entity.Property(e => e.Description).HasDefaultValue("Нет описания");
        entity.Property(e => e.Name).HasMaxLength(100);
        entity.Property(e => e.Price).HasColumnType("decimal(10, 2)");
    });

    modelBuilder.Entity<Client>(entity =>
    {
        entity.HasKey(e => e.ClientId).HasName("PK__Clients__E67E1A04EBD6C7DE");

        entity.Property(e => e.ClientId).HasColumnName("ClientID");
        entity.Property(e => e.FirstName).HasMaxLength(50);
        entity.Property(e => e.Gender).HasMaxLength(10);
        entity.Property(e => e.LastName).HasMaxLength(50);
        entity.Property(e => e.MiddleName).HasMaxLength(50);
        entity.Property(e => e.NationalityId).HasColumnName("NationalityID");
        entity.Property(e => e.Profession)
            .HasMaxLength(100)
            .HasDefaultValue("Безработный");
        entity.Property(e => e.ZodiacSignId).HasColumnName("ZodiacSignID");

        entity.HasOne(d => d.Nationality).WithMany(p => p.Clients)
            .HasForeignKey(d => d.NationalityId)
            .OnDelete(DeleteBehavior.SetNull)
            .HasConstraintName("FK__Clients__Nationa__4222D4EF");

        entity.HasOne(d => d.ZodiacSign).WithMany(p => p.Clients)
            .HasForeignKey(d => d.ZodiacSignId)
            .OnDelete(DeleteBehavior.SetNull)
            .HasConstraintName("FK__Clients__ZodiacS__412EB0B6");
    });

    modelBuilder.Entity<ClientFullInfo>(entity =>
    {
        entity
            .HasNoKey()
            .ToView("ClientFullInfo");

        entity.Property(e => e.Address).HasMaxLength(255);
        entity.Property(e => e.ClientId).HasColumnName("ClientID");
        entity.Property(e => e.FirstName).HasMaxLength(50);
        entity.Property(e => e.Gender).HasMaxLength(10);
        entity.Property(e => e.Height).HasColumnType("decimal(5, 2)");
        entity.Property(e => e.LastName).HasMaxLength(50);
        entity.Property(e => e.MaritalStatus).HasMaxLength(50);
        entity.Property(e => e.MiddleName).HasMaxLength(50);
        entity.Property(e => e.Nationality).HasMaxLength(50);
        entity.Property(e => e.PassportData).HasMaxLength(100);
        entity.Property(e => e.Phone).HasMaxLength(20);
        entity.Property(e => e.Profession).HasMaxLength(100);
        entity.Property(e => e.Weight).HasColumnType("decimal(5, 2)");
```

```
    entity.Property(e => e.ZodiacSign).HasMaxLength(50);
});

modelBuilder.Entity<ClientService>(entity =>
{
    entity
        .HasNoKey()
        .ToView("ClientServices");

    entity.Property(e => e.ClientFirstName).HasMaxLength(50);
    entity.Property(e => e.ClientId).HasColumnName("ClientID");
    entity.Property(e => e.ClientLastName).HasMaxLength(50);
    entity.Property(e => e.Cost).HasColumnType("decimal(10, 2)");
    entity.Property(e => e.EmployeeFirstName).HasMaxLength(50);
    entity.Property(e => e.EmployeeLastName).HasMaxLength(50);
    entity.Property(e => e.ServiceId).HasColumnName("ServiceID");
    entity.Property(e => e.ServiceName).HasMaxLength(100);
});

modelBuilder.Entity<Contact>(entity =>
{
    entity.HasKey(e => e.ClientId).HasName("PK__Contacts__E67E1A0495336D48");

    entity.HasIndex(e => e.Phone, "UQ__Contacts__5C7E359E823E174F").IsUnique();

    entity.HasIndex(e => e.PassportData, "UQ__Contacts__E8994A81C0646797").IsUnique();

    entity.Property(e => e.ClientId)
        .ValueGeneratedNever()
        .HasColumnName("ClientID");
    entity.Property(e => e.Address).HasMaxLength(255);
    entity.Property(e => e.PassportData).HasMaxLength(100);
    entity.Property(e => e.Phone).HasMaxLength(20);

    entity.HasOne(d => d.Client).WithOne(p => p.Contact)
        .HasForeignKey<Contact>(d => d.ClientId)
        .HasConstraintName("FK__Contacts__Client__47DBAE45");
});

modelBuilder.Entity<Employee>(entity =>
{
    entity.HasKey(e => e.EmployeeId).HasName("PK__Employee__7AD04FF1A6174395");

    entity.Property(e => e.EmployeeId).HasColumnName("EmployeeID");
    entity.Property(e => e.FirstName).HasMaxLength(50);
    entity.Property(e => e.LastName).HasMaxLength(50);
    entity.Property(e => e.Position)
        .HasMaxLength(100)
        .HasDefaultValue("Неизвестный");
});

modelBuilder.Entity<EmployeeService>(entity =>
{
    entity
        .HasNoKey()
        .ToView("EmployeeServices");

    entity.Property(e => e.ClientFirstName).HasMaxLength(50);
    entity.Property(e => e.ClientId).HasColumnName("ClientID");
    entity.Property(e => e.ClientLastName).HasMaxLength(50);
    entity.Property(e => e.Cost).HasColumnType("decimal(10, 2)");
    entity.Property(e => e.EmployeeFirstName).HasMaxLength(50);
    entity.Property(e => e.EmployeeId).HasColumnName("EmployeeID");
    entity.Property(e => e.EmployeeLastName).HasMaxLength(50);
    entity.Property(e => e.Position).HasMaxLength(100);
    entity.Property(e => e.ServiceId).HasColumnName("ServiceID");
    entity.Property(e => e.ServiceName).HasMaxLength(100);
```

```
});

modelBuilder.Entity<Nationality>(entity =>
{
    entity.HasKey(e => e.NationalityId).HasName("PK__National__F628E7A472623B36");

    entity.HasIndex(e => e.Name, "UQ__National__737584F6FE3F4DE8").IsUnique();

    entity.Property(e => e.NationalityId).HasColumnName("NationalityID");
    entity.Property(e => e.Name).HasMaxLength(50);
    entity.Property(e => e.Notes).HasDefaultValue("Нет доступных заметок");
});

modelBuilder.Entity<Photo>(entity =>
{
    entity.HasKey(e => e.ClientId).HasName("PK__Photos__E67E1A043F13E97A");

    entity.Property(e => e.ClientId)
        .ValueGeneratedNever()
        .HasColumnName("ClientID");
    entity.Property(e => e.Photo1).HasColumnName("Photo");

    entity.HasOne(d => d.Client).WithOne(p => p.Photo)
        .HasForeignKey<Photo>(d => d.ClientId)
        .HasConstraintName("FK__Photos__ClientID__52593CB8");
});

modelBuilder.Entity<PhysicalAttribute>(entity =>
{
    entity.HasKey(e => e.ClientId).HasName("PK__Physical__E67E1A04E7B1AE3E");

    entity.Property(e => e.ClientId)
        .ValueGeneratedNever()
        .HasColumnName("ClientID");
    entity.Property(e => e.Height).HasColumnType("decimal(5, 2)");
    entity.Property(e => e.MaritalStatus)
        .HasMaxLength(50)
        .HasDefaultValue("Одинокий");
    entity.Property(e => e.Weight).HasColumnType("decimal(5, 2)");

    entity.HasOne(d => d.Client).WithOne(p => p.PhysicalAttribute)
        .HasForeignKey<PhysicalAttribute>(d => d.ClientId)
        .HasConstraintName("FK__PhysicalA__Clien__4AB81AF0");
});

modelBuilder.Entity<Service>(entity =>
{
    entity.HasKey(e => e.ServiceId).HasName("PK__Services__C51BB0EA6F6C8CDE");

    entity.Property(e => e.ServiceId).HasColumnName("ServiceID");
    entity.Property(e => e.AdditionalServiceId).HasColumnName("AdditionalServiceID");
    entity.Property(e => e.ClientId).HasColumnName("ClientID");
    entity.Property(e => e.Cost).HasColumnType("decimal(10, 2)");
    entity.Property(e => e.EmployeeId).HasColumnName("EmployeeID");

    entity.HasOne(d => d.AdditionalService).WithMany(p => p.Services)
        .HasForeignKey(d => d.AdditionalServiceId)
        .HasConstraintName("FK__Services__Additi__5DCAEF64");

    entity.HasOne(d => d.Client).WithMany(p => p.Services)
        .HasForeignKey(d => d.ClientId)
        .HasConstraintName("FK__Services__Client__5EBF139D");

    entity.HasOne(d => d.Employee).WithMany(p => p.Services)
        .HasForeignKey(d => d.EmployeeId)
        .HasConstraintName("FK__Services__Employ__5FB337D6");
});
```

```
        modelBuilder.Entity<ZodiacSign>(entity =>
        {
            entity.HasKey(e => e.ZodiacSignId).HasName("PK__ZodiacSi__48424122AA73AB1B");

            entity.HasIndex(e => e.Name, "UQ__ZodiacSi__737584F6CCE6814D").IsUnique();

            entity.Property(e => e.ZodiacSignId).HasColumnName("ZodiacSignID");
            entity.Property(e => e.Description).HasDefaultValue("Нет описания");
            entity.Property(e => e.Name).HasMaxLength(50);
        });

        OnModelCreatingPartial(modelBuilder);
    }

    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

ПРИЛОЖЕНИЕ Г
(обязательное)

Текст программы

Main.yml
```
name: .NET

on:
  push:
  pull_request:
    branches: [ main ]
    paths:
    - '**.cs'
    - '**.csproj'

env:
  DOTNET_VERSION: '8.0' # The .NET SDK version to use

jobs:
  build-and-test:
    name: build-and-test-${{matrix.os}}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]

    steps:
    - uses: actions/checkout@v4

    - name: Setup .NET Core
      uses: actions/setup-dotnet@v4
      with:
        dotnet-version: ${{ env.DOTNET_VERSION }}

    - name: Install dependencies
      run: dotnet restore lab2/EFCoreLINQ/EFCoreLINQ/EFCoreLINQ.csproj

    - name: Build
      run: dotnet build lab2/EFCoreLINQ/EFCoreLINQ/EFCoreLINQ.csproj --configuration Release --no-restore
```

ПРИЛОЖЕНИЕ Д
(обязательное)

Текст программы

Queries.cs

```
using EFCoreLINQ.Data;
using EFCoreLINQ.Models;
using Microsoft.EntityFrameworkCore;

namespace EFCoreLINQ.Queries
{
   /// <summary>
   /// Класс для выполнения запросов к базе данных агентства по браку.
   /// </summary>
   public class Queries
   {
      private readonly MarriageAgencyContext _context;

      /// <summary>
      /// Инициализирует новый экземпляр класса <see cref="Queries"/> с указанным контекстом базы данных.
      /// </summary>
      /// <param name="context">Контекст базы данных.</param>
      public Queries(MarriageAgencyContext context)
      {
         _context = context;
      }

      /// <summary>
      /// Получает всех клиентов с ограничением на количество записей.
      /// </summary>
      /// <param name="recordsNumber">Максимальное количество записей для выборки.</param>
      /// <returns>Запрос, содержащий всех клиентов.</returns>
      public IQueryable<Client> GetAllClients(int recordsNumber)
      {
         return _context.Clients.Take(recordsNumber);
      }

      /// <summary>
      /// Фильтрует клиентов по профессии с ограничением на количество записей.
      /// </summary>
      /// <param name="profession">Профессия для фильтрации.</param>
      /// <param name="recordsNumber">Максимальное количество записей для выборки.</param>
      /// <returns>Запрос, содержащий отфильтрованных клиентов.</returns>
      public IQueryable<Client> FilterClientsByProfession(string profession, int recordsNumber)
      {
         return _context.Clients
            .Where(c => c.Profession == profession)
            .Take(recordsNumber);
      }

      /// <summary>
      /// Группирует клиентов по профессии с расчетом статистики по росту.
      /// </summary>
      /// <param name="recordsNumber">Максимальное количество клиентов для выборки.</param>
      /// <returns>Запрос, содержащий группы клиентов и статистику по росту.</returns>
      public IQueryable<object> GroupClientsByProfession(int recordsNumber)
      {
         return _context.Clients
            .GroupBy(c => c.Profession)
            .Select(g => new
            {
               Profession = g.Key,
               ClientsCount = g.Count(),
               MinHeight = g.Min(c => c.PhysicalAttribute.Height),
```

```
            MaxHeight = g.Max(c => c.PhysicalAttribute.Height),
            AvgHeight = g.Average(c => c.PhysicalAttribute.Height),
            Clients = g.Take(recordsNumber).ToList() // Выбираем только recordsNumber клиентов
        });
}

/// <summary>
/// Получает услуги клиентов с ограничением на количество записей.
/// </summary>
/// <param name="recordsNumber">Максимальное количество записей для выборки.</param>
/// <returns>Запрос, содержащий услуги клиентов.</returns>
public IQueryable<object> GetClientServices(int recordsNumber)
{
    return (from service in _context.Services
            join additionalService in _context.AdditionalServices
            on service.AdditionalServiceId equals additionalService.AdditionalServiceId
            select new
            {
                ClientName = service.Client.FirstName + " " + service.Client.LastName,
                ServiceName = additionalService.Name,
                service.Cost,
                service.Date
            }).Take(recordsNumber);
}

/// <summary>
/// Фильтрует клиентов по национальности с ограничением на количество записей.
/// </summary>
/// <param name="nationality">Название национальности для фильтрации.</param>
/// <param name="recordsNumber">Максимальное количество записей для выборки.</param>
/// <returns>Запрос, содержащий отфильтрованных клиентов.</returns>
public IQueryable<object> FilterClientsByNationality(string nationality, int recordsNumber)
{
    return (from client in _context.Clients
            join nat in _context.Nationalities
            on client.NationalityId equals nat.NationalityId
            where nat.Name.Contains(nationality)
            select new
            {
                ClientName = client.FirstName + " " + client.LastName,
                Nationality = nat.Name,
                client.Profession
            }).Take(recordsNumber);
}

/// <summary>
/// Добавляет нового клиента в базу данных.
/// </summary>
/// <param name="client">Клиент для добавления.</param>
public void AddClient(Client client)
{
    _context.Clients.Add(client);
    _context.SaveChanges();
}

/// <summary>
/// Добавляет новую услугу в базу данных.
/// </summary>
/// <param name="service">Услуга для добавления.</param>
public void AddService(Service service)
{
    _context.Services.Add(service);
    _context.SaveChanges();
}

/// <summary>
/// Удаляет клиента по имени и фамилии.
```

```
        /// </summary>
        /// <param name="firstName">Имя клиента.</param>
        /// <param name="lastName">Фамилия клиента.</param>
        /// <returns>Удаленный клиент или null, если клиент не найден.</returns>
        public Client DeleteClient(string firstName, string lastName)
        {
            var client = _context.Clients
                .FirstOrDefault(c => c.FirstName == firstName && c.LastName == lastName); // Простой поиск по имени
и фамилии

            if (client != null)
            {
                _context.Clients.Remove(client); // Если клиент найден, удаляем
                _context.SaveChanges(); // Сохраняем изменения
            }

            return client; // Возвращаем удаленного клиента или null, если не найден
        }

        /// <summary>
        /// Удаляет услугу по идентификатору.
        /// </summary>
        /// <param name="serviceId">Идентификатор услуги.</param>
        /// <returns>Удаленная услуга или null, если услуга не найдена.</returns>
        public Service DeleteService(int serviceId)
        {
            var service = _context.Services
                .Include(s => s.AdditionalService) // Загружаем дополнительную услугу
                .FirstOrDefault(s => s.ServiceId == serviceId); // Находим услугу по ID

            if (service != null)
            {
                _context.Services.Remove(service); // Если услуга найдена, удаляем
                _context.SaveChanges(); // Сохраняем изменения
            }

            return service; // Возвращаем удаленную услугу или null, если не найдена
        }

        /// <summary>
        /// Обновляет записи клиентов, удовлетворяющих заданному порогу возраста.
        /// </summary>
        /// <param name="ageThreshold">Порог возраста для обновления.</param>
        /// <returns>Запрос с обновленными записями клиентов.</returns>
        public IQueryable<Client> UpdateClientRecords(int ageThreshold)
        {
            // Получаем клиентов, удовлетворяющих условию
            var clientsToUpdate = _context.Clients
                .Where(c => c.PhysicalAttribute.Age > ageThreshold)
                .ToList(); // Загружаем в память

            // Обновляем записи
            foreach (var client in clientsToUpdate)
            {
                client.Profession = "Обновленная профессия"; // Пример обновления
            }

            // Сохраняем изменения в базе данных
            _context.SaveChanges();

            // Возвращаем обновленные записи
            return _context.Clients.Where(c => c.PhysicalAttribute.Age > ageThreshold);
        }
    }
}
```