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 №8

Школа

Выполнил:

студент группы ИТИ-31, Демиденко А. С. 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/soManyVoices/RPBDIS-LABS

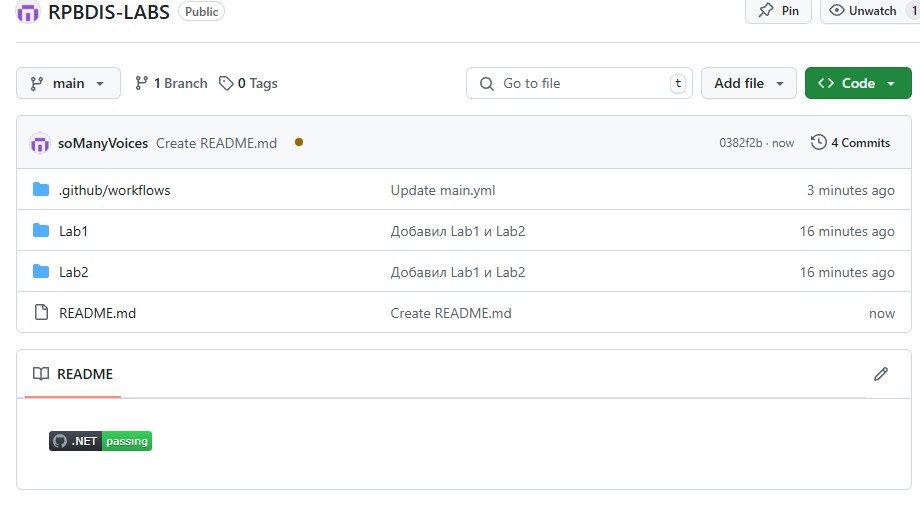


Рисунок 1 – *GitHub* Репозиторий

На рисунке 2 представлена строка подключения конфигурационного *json*-

файла.



Рисунок 2 – Строка подключения

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

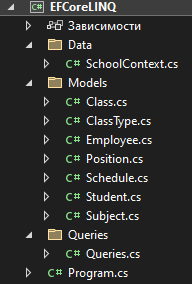


Рисунок 3 – Модели и контекст данных

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

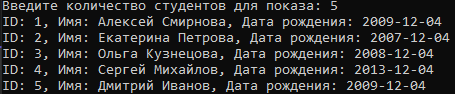


Рисунок 4 – Выборка данных 1

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

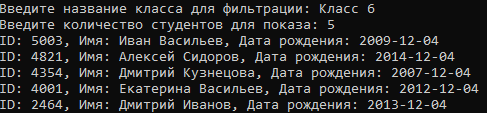


Рисунок 5 – Выборка данных 2

На рисунке 6 представлена выборку данных таблицы «Student», сгруппированных по классу 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.

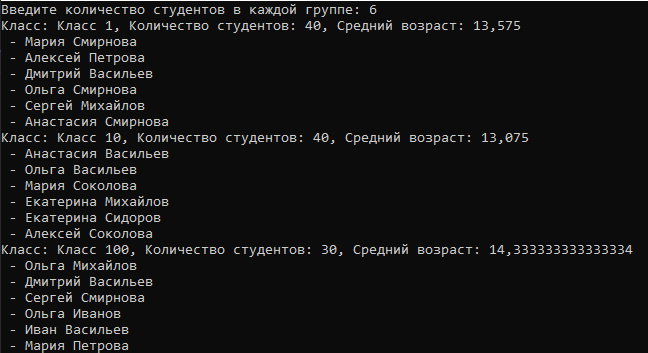


Рисунок 6 – Выборка данных 3

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

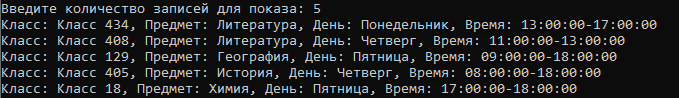


Рисунок 7 – Выборка данных 4

На рисунке 8 представлена выборка данных таблицы «Employee» и отфильтрованным по должности, налагающему ограничения на значения одного или нескольких полей – 1 шт.

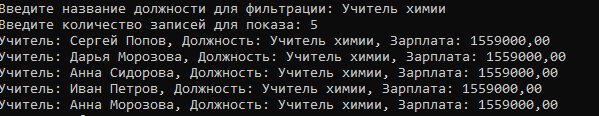


Рисунок 8 – Выборка данных 5

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

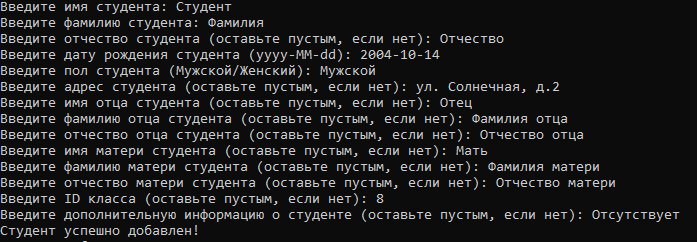


Рисунок 9 – Вставка данных 1

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



Рисунок 10 – Вставка данных 2

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



Рисунок 11 – Удаление данных 1

На рисунке 12 представлено удаление данных из таблицы «Subject», 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 из- менении в репозитории (Приложение Г).

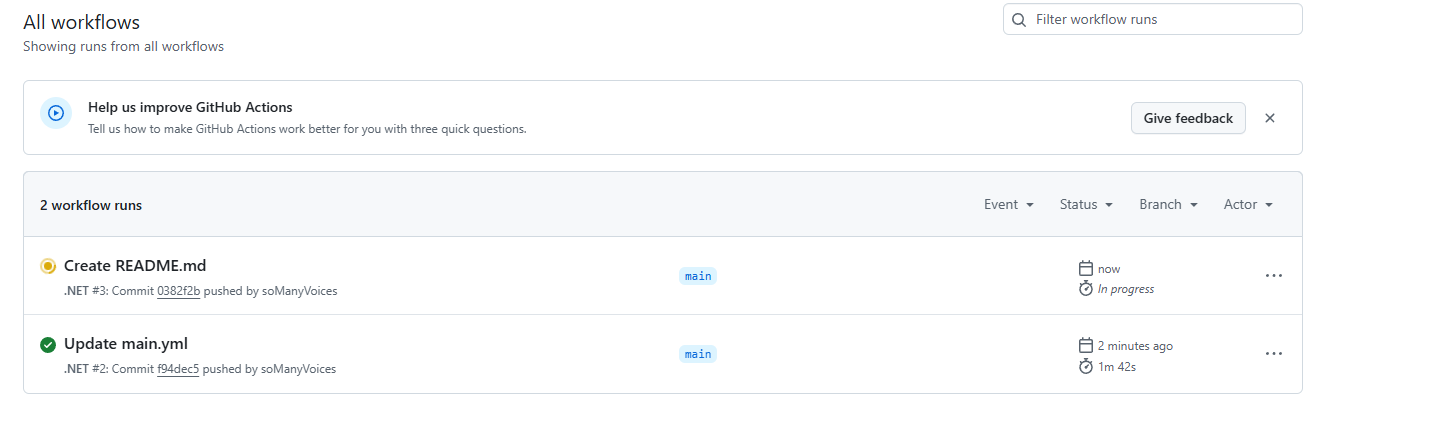


Рисунок 14 – Рабочий процесс *GitHub Actions*

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.

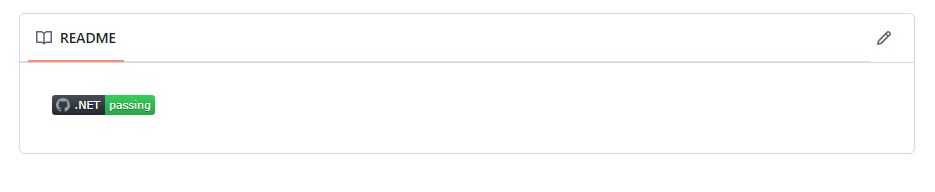


Рисунок 15 – Эмблема состояния

**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**

**Class.cs:**

using System;

using System.Collections.Generic;

namespace EFCoreLINQ.Models;

public class Class

{

public int ClassId { get; set; }

public string Name { get; set; }

public string ClassTeacher { get; set; }

public int ClassTypeId { get; set; }

public int StudentCount { get; set; }

public int YearCreated { get; set; }

public ClassType ClassType { get; set; }

public ICollection<Student> Students { get; set; } = new HashSet<Student>();

public ICollection<Schedule> Schedules { get; set; } = new HashSet<Schedule>();

}

**ClassType.cs:**

using System;

using System.Collections.Generic;

namespace EFCoreLINQ.Models;

public class ClassType

{

public int ClassTypeId { get; set; }

public string Name { get; set; }

public string? Description { get; set; }

public ICollection<Class> Classes { get; set; } = new HashSet<Class>();

}

**Employee.cs**

using System;

using System.Collections.Generic;

namespace EFCoreLINQ.Models;

public class Employee

{

public int EmployeeId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string? MiddleName { get; set; }

public int PositionId { get; set; }

public Position Position { get; set; }

public ICollection<Subject> Subjects { get; set; } = new HashSet<Subject>();

}

**Position.cs**

using System;

using System.Collections.Generic;

namespace EFCoreLINQ.Models;

public class Position

{

public int PositionId { get; set; }

public string Name { get; set; }

public string? Description { get; set; }

public decimal? Salary { get; set; }

public ICollection<Employee> Employees { get; set; } = new HashSet<Employee>();

}

**Schedule.cs**

using System;

using System.Collections.Generic;

namespace EFCoreLINQ.Models;

public class Schedule

{

public int ScheduleId { get; set; }

public DateTime Date { get; set; }

public string DayOfWeek { get; set; }

public int ClassId { get; set; }

public int SubjectId { get; set; }

public TimeSpan StartTime { get; set; }

public TimeSpan EndTime { get; set; }

public Class Class { get; set; }

public Subject Subject { get; set; }

}

**Student.cs**

using System;

using System.Collections.Generic;

namespace EFCoreLINQ.Models;

public class Student

{

public int StudentId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string? MiddleName { get; set; }

public DateTime DateOfBirth { get; set; }

public string Gender { get; set; }

public string? Address { get; set; }

public string? FatherFirstName { get; set; }

public string? FatherLastName { get; set; }

public string? FatherMiddleName { get; set; }

public string? MotherFirstName { get; set; }

public string? MotherLastName { get; set; }

public string? MotherMiddleName { get; set; }

public int? ClassId { get; set; }

public string? AdditionalInfo { get; set; }

public Class? Class { get; set; }

}

**Subject.cs**

using System;

using System.Collections.Generic;

namespace EFCoreLINQ.Models;

public class Subject

{

public int SubjectId { get; set; }

public string Name { get; set; }

public string? Description { get; set; }

public int EmployeeId { get; set; }

public Employee Employee { get; set; }

public ICollection<Schedule> Schedules { get; set; } = new HashSet<Schedule>();

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Текст программы**

## Appsettings.json

{

"ConnectionStrings": {

"SQLConnection": "Server=HOME-PC;Database=School; Trusted\_Connection =True; TrustServerCertifi- cate=True"

}

}

# ПРИЛОЖЕНИЕ В

(обязательное)

**Текст программы**

**SchoolContext.cs**

using System;

using EFCoreLINQ.Models;

using System.Security.Claims;

using Microsoft.EntityFrameworkCore;

namespace SchoolDatabase

{

public partial class SchoolContext : DbContext

{

public SchoolContext()

{

}

public SchoolContext(DbContextOptions<SchoolContext> options)

: base(options)

{

}

public virtual DbSet<ClassType> ClassTypes { get; set; }

public virtual DbSet<Class> Classes { get; set; }

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

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

public virtual DbSet<Subject> Subjects { get; set; }

public virtual DbSet<Student> Students { get; set; }

public virtual DbSet<Schedule> Schedules { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

=> optionsBuilder.UseSqlServer("Server=HOME-PC;Database=School;Trusted\_Connection=True;TrustServerCertificate=True");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<ClassType>(entity =>

{

entity.HasKey(e => e.ClassTypeId);

entity.HasIndex(e => e.Name).IsUnique();

entity.Property(e => e.Name).HasMaxLength(100).IsRequired();

entity.Property(e => e.Description).HasMaxLength(255);

});

modelBuilder.Entity<Class>(entity =>

{

entity.HasKey(e => e.ClassId);

entity.HasIndex(e => e.Name).IsUnique();

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

entity.Property(e => e.ClassTeacher).HasMaxLength(100).IsRequired();

entity.Property(e => e.YearCreated).IsRequired();

entity.Property(e => e.StudentCount).HasDefaultValue(0);

entity.HasOne(e => e.ClassType)

.WithMany(ct => ct.Classes)

.HasForeignKey(e => e.ClassTypeId)

.OnDelete(DeleteBehavior.Cascade);

});

modelBuilder.Entity<Position>(entity =>

{

entity.HasKey(e => e.PositionId);

entity.HasIndex(e => e.Name).IsUnique();

entity.Property(e => e.Name).HasMaxLength(100).IsRequired();

entity.Property(e => e.Description).HasMaxLength(255);

entity.Property(e => e.Salary).HasColumnType("decimal(10, 2)");

});

modelBuilder.Entity<Employee>(entity =>

{

entity.HasKey(e => e.EmployeeId);

entity.Property(e => e.FirstName).HasMaxLength(50).IsRequired();

entity.Property(e => e.LastName).HasMaxLength(50).IsRequired();

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

entity.HasOne(e => e.Position)

.WithMany(p => p.Employees)

.HasForeignKey(e => e.PositionId)

.OnDelete(DeleteBehavior.Cascade);

});

modelBuilder.Entity<Subject>(entity =>

{

entity.HasKey(e => e.SubjectId);

entity.HasIndex(e => e.Name).IsUnique();

entity.Property(e => e.Name).HasMaxLength(100).IsRequired();

entity.Property(e => e.Description).HasMaxLength(255);

entity.HasOne(e => e.Employee)

.WithMany(emp => emp.Subjects)

.HasForeignKey(e => e.EmployeeId)

.OnDelete(DeleteBehavior.Cascade);

});

modelBuilder.Entity<Student>(entity =>

{

entity.HasKey(e => e.StudentId);

entity.Property(e => e.FirstName).HasMaxLength(50).IsRequired();

entity.Property(e => e.LastName).HasMaxLength(50).IsRequired();

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

entity.Property(e => e.Gender).HasMaxLength(10).IsRequired();

entity.Property(e => e.Address).HasMaxLength(255);

entity.Property(e => e.AdditionalInfo).HasMaxLength(255);

entity.HasOne(e => e.Class)

.WithMany(cls => cls.Students)

.HasForeignKey(e => e.ClassId)

.OnDelete(DeleteBehavior.SetNull);

});

modelBuilder.Entity<Schedule>(entity =>

{

entity.HasKey(e => e.ScheduleId);

entity.Property(e => e.DayOfWeek).HasMaxLength(20).IsRequired();

entity.Property(e => e.StartTime).IsRequired();

entity.Property(e => e.EndTime).IsRequired();

entity.HasOne(e => e.Class)

.WithMany(cls => cls.Schedules)

.HasForeignKey(e => e.ClassId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(e => e.Subject)

.WithMany(sub => sub.Schedules)

.HasForeignKey(e => e.SubjectId)

.OnDelete(DeleteBehavior.Cascade);

});

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 System;

using System.Linq;

using EFCoreLINQ.Models;

using SchoolDatabase;

namespace EFCoreLINQ.Queries

{

public class Queries

{

private readonly SchoolContext \_context;

public Queries(SchoolContext context)

{

\_context = context;

}

public IQueryable<Student> GetAllStudents(int recordsNumber)

{

return \_context.Students.Take(recordsNumber);

}

public IQueryable<Student> FilterStudentsByClass(string className, int recordsNumber)

{

return \_context.Students

.Where(s => s.Class != null && s.Class.Name == className)

.Take(recordsNumber);

}

public IQueryable<object> GroupStudentsByClass(int recordsNumber)

{

return \_context.Students

.GroupBy(s => s.Class.Name)

.Select(g => new

{

ClassName = g.Key,

StudentsCount = g.Count(),

MinAge = g.Min(s => (DateTime.Now.Year - s.DateOfBirth.Year)),

MaxAge = g.Max(s => (DateTime.Now.Year - s.DateOfBirth.Year)),

AvgAge = g.Average(s => DateTime.Now.Year - s.DateOfBirth.Year),

Students = g.Take(recordsNumber).ToList()

});

}

public IQueryable<object> GetClassSchedules(int recordsNumber)

{

return (from schedule in \_context.Schedules

join subject in \_context.Subjects

on schedule.SubjectId equals subject.SubjectId

join classInfo in \_context.Classes

on schedule.ClassId equals classInfo.ClassId

select new

{

ClassName = classInfo.Name,

SubjectName = subject.Name,

DayOfWeek = schedule.DayOfWeek,

StartTime = schedule.StartTime,

EndTime = schedule.EndTime

}).Take(recordsNumber);

}

public IQueryable<object> FilterTeachersByPosition(string positionName, int recordsNumber)

{

return (from employee in \_context.Employees

join position in \_context.Positions

on employee.PositionId equals position.PositionId

where position.Name.Contains(positionName)

select new

{

TeacherName = employee.FirstName + " " + employee.LastName,

Position = position.Name,

Salary = position.Salary

}).Take(recordsNumber);

}

public void AddStudent(Student student)

{

\_context.Students.Add(student);

\_context.SaveChanges();

}

public Employee? GetEmployeeById(int employeeId)

{

return \_context.Employees.SingleOrDefault(e => e.EmployeeId == employeeId);

}

public void AddSubject(Subject subject)

{

\_context.Subjects.Add(subject);

\_context.SaveChanges();

}

public void DeleteStudent(string firstName, string lastName)

{

var student = \_context.Students

.FirstOrDefault(s => s.FirstName.ToLower() == firstName.ToLower() &&

s.LastName.ToLower() == lastName.ToLower());

if (student != null)

{

\_context.Students.Remove(student);

\_context.SaveChanges();

}

else

{

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

}

}

public void DeleteSubject(int subjectId)

{

var subject = \_context.Subjects.Find(subjectId);

if (subject != null)

{

\_context.Subjects.Remove(subject);

\_context.SaveChanges();

}

}

public void UpdateTeacherSalaries(decimal salaryIncrease)

{

var teachers = \_context.Employees.ToList();

foreach (var teacher in teachers)

{

var position = \_context.Positions.Find(teacher.PositionId);

if (position != null && position.Salary != null)

{

position.Salary += salaryIncrease;

}

}

\_context.SaveChanges();

}

}

}