Министерство транспорта Российской Федерации

Федеральное государственное автономное образовательное  
учреждение высшего образования  
«РУТ (МИИТ)»

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

КУРСОВОЙ ПРОЕКТ

по дисциплине

**«Основы построения защищенных баз данных»**

**на тему «Отдел кадров»**

Выполнили: ст. гр. ТКИ-441

Галыба Л.Д.

Поткин Д.А.

Проверил: доц., к.т.н.

Васильева М. А.

Москва 2024

**Оглавление**

[Цель курсового проекта …………………………………...3](#_Toc184207231)

[Задание на курсовой проект………………………………3](#_Toc184207232)

[UML-Диаграмма…………………………………………...4](#_Toc184207233)

[Код программы………………………………………….....4](#_Toc184207234)

[Прохождение тестов……………………………………..16](#_Toc184207235)

[Настройка миграции……………………………………..23](#_Toc184207236)

[ER-диаграмма………………………………………….....30](#_Toc184207237)

[Список литературы………………………………………30](#_Toc184207238)

# Цель курсового проекта

Изучить современные технологии ORM, разработать приложение с базой данных, умеющее отрабатывать операции CRUD.

Стек технологий: язык программирования C#, ORM – EF Core, NUnit, PostgreSQL. stylecop

# Задание на курсовой проект

1. Выбрать предметную область.

В соответствии с заданием (выбрать предметную область самостоятельно) описать предметную область, выделить основные сущности, формализовать задачу будущего приложения.

2. Разработать приложение с учетом выбранной технологии ORM. Покрыть все сущностные классы и классы отображения тестами.

3. Разработать репозиторий для работы с БД.

**Описание предметной области.**

База данных создаётся для информационного хранения информации о сеансах в кинотеатре. БД должна содержать информацию Работниках, их Должностях и Детях.

# UML-Диаграмма

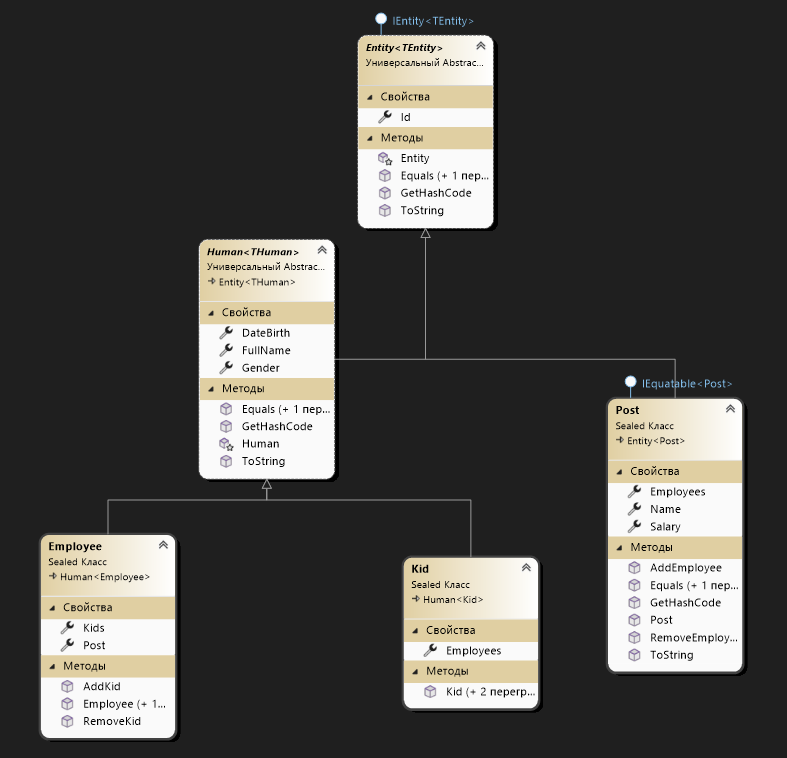


Рисунок 1 – UML – диаграмма

# Код программы

**Employee.cs**

// <copyright file="Employee.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using System;

using System.Collections.Generic;

/// <summary>

/// Класс работник.

/// </summary>

public sealed class Employee : Human<Employee>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Employee"/>.

/// </summary>

/// <param name="fullName"> Полное имя.</param>

/// <param name="dateBirth">День рождения.</param>

/// <param name="gender">Пол.</param>

/// <param name="post">Должность.</param>

public Employee(Name fullName, DateOnly dateBirth, Gender gender, Post? post = null)

: base(fullName, dateBirth, gender)

{

this.Post = post;

if (this.Post is not null)

{

\_ = this.Post.AddEmployee(this);

}

}

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Employee"/>.

/// </summary>

[Obsolete("For ORM only", true)]

private Employee()

: base(Name.Unknown)

{

}

/// <summary>

/// Работники с детьми.

/// </summary>

public ISet<Kid> Kids { get; } = new HashSet<Kid>();

/// <summary>

/// Добавление ребенка.

/// </summary>

/// <param name="kid">Ребенок.</param>

/// <returns>Ребенок добавлен.</returns>

public bool AddKid(Kid kid)

{

return kid is not null

&& this.Kids.Add(kid)

&& kid.Employees.Add(this);

}

/// <summary>

/// Удаление рребенка.

/// </summary>

/// <param name="kid">Ребенок.</param>

/// <returns>Ребенок удален.</returns>

public bool RemoveKid(Kid kid)

{

return kid is not null

&& this.Kids.Remove(kid)

&& kid.Employees.Remove(this);

}

/// <summary>

/// Должность.

/// </summary>

public Post Post { get; set; }

}

}

**Entity.cs**

// <copyright file="Entity.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using System;

/// <summary>

/// Базовая сущность.

/// </summary>

/// <typeparam name="TEntity"> Тип конкретной сущности. </typeparam>

public abstract class Entity<TEntity> : IEntity<TEntity>

where TEntity : class, IEntity<TEntity>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Entity{TEntity}"/>.

/// </summary>

protected Entity() => this.Id = Guid.Empty;

/// <inheritdoc cref="IEntity{TEntity}.Id"/>

public virtual Guid Id { get; protected set; }

/// <inheritdoc cref="object.ToString"/>

public override string ToString() => $"[{this.Id}]";

/// <inheritdoc cref="object.Equals(object?)"/>

public override bool Equals(object? obj)

{

return ReferenceEquals(this, obj)

|| (obj is TEntity entity && this.Equals(entity));

}

/// <inheritdoc/>

public virtual bool Equals(TEntity? other)

{

return other is not null

&& this.GetType() == other.GetType();

}

/// <inheritdoc/>

// @NOTE: В случае проблемы заменить на object.GetHashCode().

public override int GetHashCode() => base.GetHashCode();

}

}

**Gender.cs**

// <copyright file="Gender.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

/// <summary>

/// Перечисление полов.

/// </summary>

public enum Gender

{

/// <summary>

/// Мужской пол.

/// </summary>

Male,

/// <summary>

/// Женский пол.

/// </summary>

Female,

/// <summary>

/// Неизвестно.

/// </summary>

Unknow,

}

}

**Human.cs**

// <copyright file="Human.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using System;

using System.Text;

/// <summary>

/// Класс Человек.

/// </summary>

public abstract class Human<THuman> : Entity<THuman>

where THuman : Human<THuman>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Human{THuman}"/>.

/// </summary>

/// <param name="fullName"> Полное имя.</param>

/// <param name="dateBirth"> Дата рождения. </param>

/// <param name="gender"> Пол. </param>

/// <exception cref="ArgumentNullException">

/// Если Полное имя <see langword="null"/>.

/// </exception>

protected Human(

Name fullName,

DateOnly? dateBirth = null,

Gender gender = Gender.Unknow)

{

this.FullName = fullName ?? throw new ArgumentNullException(nameof(fullName));

this.DateBirth = dateBirth;

this.Gender = gender;

}

/// <summary>

/// Имя.

/// </summary>

public Name FullName { get; }

/// <summary>

/// Дата р

/// </summary>

public DateOnly? DateBirth { get; set; }

/// <summary>

/// Пол.

/// </summary>

public Gender Gender { get; }

/// <inheritdoc/>

public override bool Equals(object? obj)

{

return obj is THuman person && this.Equals(person);

}

/// <inheritdoc/>

public override bool Equals(THuman? other)

{

return base.Equals(other)

&& this.FullName == other.FullName

&& this.DateBirth == other.DateBirth;

}

/// <inheritdoc/>

public override int GetHashCode() => HashCode.Combine(this.FullName, this.DateBirth);

/// <inheritdoc/>

public override string ToString()

{

var buffer = new StringBuilder();

\_ = buffer.Append(this.FullName);

if (this.DateBirth is not null)

{

\_ = buffer.Append($" Год рождения: {this.DateBirth}");

}

return buffer.ToString();

}

}

}

**IEntity.cs**

// <copyright file="IEntity.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using System;

/// <summary>

/// Интерфейс базовой сущности.

/// </summary>

/// <typeparam name="TEntity"> Тип конкретной сущности. </typeparam>

public interface IEntity<TEntity> : IEquatable<TEntity>

where TEntity : class, IEntity<TEntity>

{

/// <summary>

/// Идентификатор.

/// </summary>

Guid Id { get; }

}

}

**Kid.cs**

// <copyright file="Kid.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using System;

using System.Collections.Generic;

/// <summary>

/// Класс ребенок.

/// </summary>

public sealed class Kid : Human<Kid>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Kid"/>.

/// </summary>

/// <param name="fullName"> Имя.</param>

/// <param name="dateBirth">Дата рорждения.</param>

/// <param name="employees">Работник.</param>

/// <param name="gender">Пол.</param>

/// <exception cref="ArgumentNullException">

/// Если Полное имя <see langword="null"/>.

/// </exception>

public Kid(Name fullName, DateOnly dateBirth, ISet<Employee> employees, Gender gender)

: base(fullName, dateBirth, gender)

{

this.Employees = employees ?? throw new ArgumentNullException(nameof(employees));

foreach (var employee in employees)

{

employee.AddKid(this);

}

}

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Kid"/>.

/// </summary>

/// <param name="fullName"> Имя.</param>

/// <param name="dateBirth">Дата рорждения.</param>

/// <param name="employees">Работник.</param>

/// <param name="gender">Пол.</param>

/// <exception cref="ArgumentNullException">Если название книги или код <see langword="null"/>.</exception>

/// <exception cref="ArgumentOutOfRangeException"> Если количество страниц меньше или равно нулю.</exception>

/// <exception cref="ArgumentOutOfRangeException"> Если полка <see langword="null"/>.</exception>

/// <exception cref="ArgumentOutOfRangeException"> Если авторы <see langword="null"/>.</exception>

public Kid(Name fullName, DateOnly dateBirth, Gender gender, params Employee[] employees)

: this(fullName, dateBirth, new HashSet<Employee>(employees), gender)

{

}

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Kid"/>.

/// </summary>

[Obsolete("For ORM only", true)]

private Kid()

: base(Name.Unknown)

{

}

/// <summary>

/// Работник.

/// </summary>

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

}

}

**Name.cs**

// <copyright file="Name.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using System;

using Extensions;

/// <summary>

/// Полное имя.

/// </summary>

public sealed class Name : IEquatable<Name>

{

/// <summary>

/// Имя-заглушка.

/// </summary>

public static readonly Name Unknown = new ("Неизвестно", "Неизвестно", "Неизвестно");

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Name"/>.

/// </summary>

/// <param name="familyName">Фамилия.</param>

/// <param name="firstName">Имя.</param>

/// <param name="patronicName">Отчество.</param>

/// <exception cref="ArgumentNullException">Если имя или фамилия <see langword="null"/>.</exception>

public Name(string familyName, string firstName, string patronicName)

{

this.FamilyName = familyName.TrimOrNull() ?? throw new ArgumentNullException(nameof(familyName));

this.FirstName = firstName.TrimOrNull() ?? throw new ArgumentNullException(nameof(firstName));

this.PatronicName = patronicName.TrimOrNull() ?? throw new ArgumentNullException(nameof(patronicName));

}

/// <summary>

/// Фамилия.

/// </summary>

public string FamilyName { get; }

/// <summary>

/// Имя.

/// </summary>

public string FirstName { get; }

/// <summary>

/// Отчество.

/// </summary>

public string PatronicName { get; }

public static bool operator == (Name? lha, Name? rha)

{

if (lha is null || rha is null)

{

return false;

}

return lha.Equals(rha);

}

public static bool operator !=(Name? lha, Name? rha) => !(lha == rha);

/// <inheritdoc/>

public bool Equals(Name? other)

{

return other is not null

&& this.FamilyName == other.FamilyName

&& this.FirstName == other.FirstName

&& this.PatronicName == other.PatronicName;

}

/// <inheritdoc/>

public override bool Equals(object? obj) => this.Equals(obj as Name);

/// <inheritdoc/>

public override int GetHashCode() =>

HashCode.Combine(this.FamilyName, this.FirstName, this.PatronicName);

/// <inheritdoc/>

public override string ToString()

{

return this.PatronicName is null

? $"{this.FamilyName} {this.FirstName}"

: $"{this.FamilyName} {this.FirstName} {this.PatronicName}";

}

}

}

**Post.cs**

// <copyright file="Post.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using Extensions;

using System;

using System.Collections.Generic;

/// <summary>

/// Класс должность.

/// </summary>

public sealed class Post : Entity<Post>, IEquatable<Post>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Post"/> class.

/// </summary>

/// <param name="name">Название должности.</param>

/// <param name="salary">Зароботная плата.</param>

public Post(string name, decimal salary)

{

ArgumentOutOfRangeException.ThrowIfNegativeOrZero(salary);

this.Salary = salary;

this.Name = name.TrimOrNull() ?? throw new ArgumentNullException(nameof(name));

}

/// <summary>

/// Работники.

/// </summary>

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

/// <summary>

/// Добавить работника.

/// </summary>

/// <param name="employee">работник.</param>

/// <returns>Должность.</returns>

public bool AddEmployee(Employee employee)

{

return employee is not null

&& this.Employees.Add(employee)

&& (employee.Post = this) is not null;

}

/// <summary>

/// Удалить работника.

/// </summary>

/// <param name="employee">Работник.</param>

/// <returns>Работник удален.</returns>

public bool RemoveEmployee(Employee employee)

{

return employee is not null

&& this.Employees.Remove(employee)

&& (employee.Post = this) is null;

}

/// <summary>

/// Название должности.

/// </summary>

public string Name { get; set; }

/// <summary>

/// Заработная плата.

/// </summary>

public decimal Salary { get; set; }

/// <inheritdoc/>

public bool Equals(Post? other)

{

return (other is not null) && (ReferenceEquals(this, other) && (this.Name == other.Name));

}

/// <inheritdoc/>

public override bool Equals(object? obj)

{

return this.Equals(obj as Post);

}

/// <inheritdoc/>

public override int GetHashCode()

{

return this.Name.GetHashCode() \* this.Salary.GetHashCode();

}

/// <inheritdoc/>

public override string ToString() => $"Должность - {this.Name}, Зарплата - {this.Salary}";

}

}

# Прохождение тестов

**PostTests.cs**

// <copyright file="Post.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoSc

{

using Extensions;

using System;

using System.Collections.Generic;

/// <summary>

/// Класс должность.

/// </summary>

public sealed class Post : Entity<Post>, IEquatable<Post>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Post"/> class.

/// </summary>

/// <param name="name">Название должности.</param>

/// <param name="salary">Зароботная плата.</param>

public Post(string name, decimal salary)

{

ArgumentOutOfRangeException.ThrowIfNegativeOrZero(salary);

this.Salary = salary;

this.Name = name.TrimOrNull() ?? throw new ArgumentNullException(nameof(name));

}

/// <summary>

/// Работники.

/// </summary>

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

/// <summary>

/// Добавить работника.

/// </summary>

/// <param name="employee">работник.</param>

/// <returns>Должность.</returns>

public bool AddEmployee(Employee employee)

{

return employee is not null

&& this.Employees.Add(employee)

&& (employee.Post = this) is not null;

}

/// <summary>

/// Удалить работника.

/// </summary>

/// <param name="employee">Работник.</param>

/// <returns>Работник удален.</returns>

public bool RemoveEmployee(Employee employee)

{

return employee is not null

&& this.Employees.Remove(employee)

&& (employee.Post = this) is null;

}

/// <summary>

/// Название должности.

/// </summary>

public string Name { get; set; }

/// <summary>

/// Заработная плата.

/// </summary>

public decimal Salary { get; set; }

/// <inheritdoc/>

public bool Equals(Post? other)

{

return (other is not null) && (ReferenceEquals(this, other) && (this.Name == other.Name));

}

/// <inheritdoc/>

public override bool Equals(object? obj)

{

return this.Equals(obj as Post);

}

/// <inheritdoc/>

public override int GetHashCode()

{

return this.Name.GetHashCode() \* this.Salary.GetHashCode();

}

/// <inheritdoc/>

public override string ToString() => $"Должность - {this.Name}, Зарплата - {this.Salary}";

}

}

**EmployeeTests.cs**

// <copyright file="EmployeeTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoTests

{

using DemoSc;

using System;

using System.Collections.Generic;

using NUnit.Framework;

[TestFixture]

internal class EmployeeTests

{

private static readonly Post Post = new ("Тестовое название", 10000);

private static readonly Name NameValue = new ("Толстой", "Лев", "Николаевич");

private static readonly Name OtherValue = new ("Пупукакин", "Чел", "Николаевич");

[Test]

public void Ctor\_ValidDate\_DoesNotThrow()

{

Assert.DoesNotThrow(() =>

\_ = new Employee(NameValue, new DateOnly(2002, 09, 08), Gender.Male, Post));

}

[Test]

public void Equals\_ValidDataDifferentName\_Success()

{

// Arrange

var employee1 = new Employee(NameValue, new DateOnly(1828, 09, 28), Gender.Male);

var employee2 = new Employee(OtherValue, new DateOnly(1799, 06, 06), Gender.Male);

// Act & Assert

Assert.That(employee1, Is.Not.EqualTo(employee2));

}

}

}

**KidTests.cs**

// <copyright file="KidTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DemoTests

{

using DemoSc;

using System;

using System.Collections.Generic;

using NUnit.Framework;

[TestFixture]

internal class KidTests

{

private static readonly Name NameValue = new ("Толстой", "Лев", "Николаевич");

private static readonly Name NameValue1 = new ("Галлер", "Кирилл", "Николаевич");

private static readonly Post Post = new ("Тестовое название", 10000);

private static readonly Employee Employee = new (NameValue, new DateOnly(2002, 09, 08), Gender.Male, Post);

[Test]

public void Ctor\_ValidDate\_DoesNotThrow()

{

Assert.DoesNotThrow(() =>

\_ = new Kid(NameValue, new DateOnly(2002, 09, 08), Gender.Male, Employee));

}

[Test]

public void Equals\_ValidDataDifferentName\_Success()

{

// Arrange

var kid1 = new Kid(NameValue, new DateOnly(2002, 09, 08), Gender.Male, Employee);

var kid2 = new Kid(NameValue1, new DateOnly(1973, 01, 08), Gender.Female, Employee);

// Act & Assert

Assert.That(kid1, Is.Not.EqualTo(kid2));

}

}

}

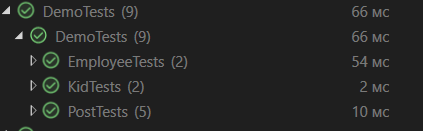


Рисунок 2 - Результат прохождения тестов

# Настройка миграции

**EmployeeConfiguration.cs**

// <copyright file="EmployeeConfiguration.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Configurations

{

using DemoSc;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

/// Конфигурация правил отображения сущности (<see cref="Employee"/>) в таблицу БД.

/// </summary>

internal sealed class EmployeeConfiguration : IEntityTypeConfiguration<Employee>

{

/// <inheritdoc/>

public void Configure(EntityTypeBuilder<Employee> builder)

{

\_ = builder.HasKey(employee => employee.Id);

\_ = builder.OwnsOne(

employee => employee.FullName,

nameBuilder =>

{

\_ = nameBuilder.Property(name => name.FamilyName)

.HasColumnName(nameof(Name.FamilyName))

.HasMaxLength(100)

.IsRequired()

.HasComment("Фамилия");

\_ = nameBuilder.Property(name => name.FirstName)

.HasColumnName(nameof(Name.FirstName))

.HasMaxLength(100)

.IsRequired()

.HasComment("Имя");

\_ = nameBuilder.Property(name => name.PatronicName)

.HasColumnName(nameof(Name.PatronicName))

.HasMaxLength(100)

.IsRequired()

.HasComment("Отчество");

});

\_ = builder.Property(employee => employee.DateBirth).HasMaxLength(50).IsRequired();

\_ = builder.Property(employee => employee.Gender).IsRequired();

\_ = builder.HasOne(employee => employee.Post).WithMany(post => post.Employees).IsRequired();

\_ = builder.HasMany(employee => employee.Kids).WithMany(kid => kid.Employees);

}

}

}

**KidConfiguration.cs**

// <copyright file="KidConfiguration.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Configurations

{

using DemoSc;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

/// Конфигурация правил отображения сущности (<see cref="Kid"/>) в таблицу БД.

/// </summary>

internal sealed class KidConfiguration : IEntityTypeConfiguration<Kid>

{

/// <inheritdoc/>

public void Configure(EntityTypeBuilder<Kid> builder)

{

\_ = builder.HasKey(kid => kid.Id);

\_ = builder.OwnsOne(

author => author.FullName,

nameBuilder =>

{

\_ = nameBuilder.Property(name => name.FamilyName)

.HasColumnName(nameof(Name.FamilyName))

.HasMaxLength(100)

.IsRequired()

.HasComment("Фамилия");

\_ = nameBuilder.Property(name => name.FirstName)

.HasColumnName(nameof(Name.FirstName))

.HasMaxLength(100)

.IsRequired()

.HasComment("Имя");

\_ = nameBuilder.Property(name => name.PatronicName)

.HasColumnName(nameof(Name.PatronicName))

.HasMaxLength(100)

.IsRequired()

.HasComment("Отчество");

});

\_ = builder.Property(kid => kid.DateBirth).HasMaxLength(50).IsRequired();

\_ = builder.HasMany(kid => kid.Employees).WithMany(employee => employee.Kids);

\_ = builder.Property(kid => kid.Gender).IsRequired();

}

}

}

**PostConfiguration.cs**

// <copyright file="PostConfiguration.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Configurations

{

using DemoSc;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

/// <summary>

/// Конфигурация правил отображения сущности (<see cref="Post"/>) в таблицу БД.

/// </summary>

internal sealed class PostConfiguration : IEntityTypeConfiguration<Post>

{

/// <inheritdoc/>

public void Configure(EntityTypeBuilder<Post> builder)

{

\_ = builder.HasKey(post => post.Id);

\_ = builder.Property(post => post.Name).IsRequired();

\_ = builder.Property(post => post.Salary).IsRequired();

}

}

}

# ER-диаграмма

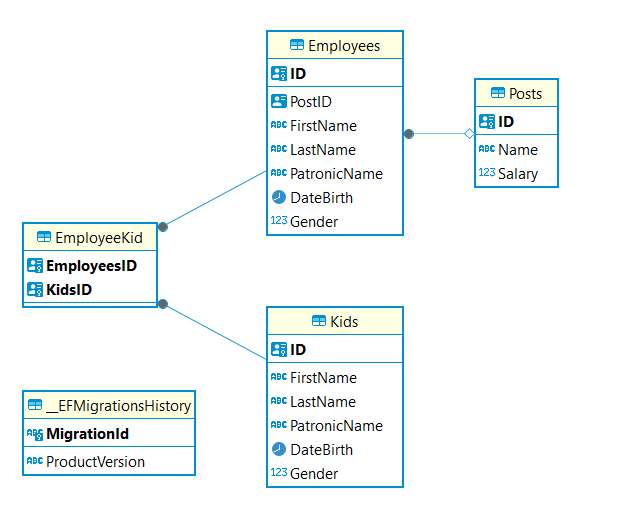


Рисунок 3 – ER-диаграмма

# Тесты на миграции

**BaseConfigurationTest**

// <copyright file="BaseConfigurationTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Tests

{

using System;

using DataAccessLayer;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Logging;

/// <summary>

/// Базовый тип для реализации модульных тестов конфигураций (<see cref="IEntityTypeConfiguration{TEntity}"/>).

/// </summary>

internal abstract class BaseConfigurationTests

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="BaseConfigurationTests"/>.

/// </summary>

/// <param name="minimumLogLevel"> Минимальный уровень логируемых сообщений. </param>

/// <exception cref="Exception">

/// В случае невозможности построения/получения контекста доступа к данным.

/// </exception>

protected BaseConfigurationTests(LogLevel minimumLogLevel = LogLevel.Debug)

{

this.DataContext = new ServiceCollection()

.AddDbContext<DataContext>(

options => options

.UseInMemoryDatabase($"InMemoryDB\_{Guid.NewGuid()}")

.EnableDetailedErrors()

.EnableSensitiveDataLogging()

.LogTo(Console.WriteLine, minimumLogLevel))

.BuildServiceProvider()

.GetService<DataContext>()

?? throw new Exception($"Cannot get {typeof(DataContext).FullName} from DI");

}

/// <summary>

/// Контекст доступа к данным.

/// </summary>

protected DataContext DataContext { get; }

}

}

**EmployeeConfigurationTests**

// <copyright file="EmployeeConfigurationTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Tests

{

using DemoSc;

using NUnit.Framework;

/// <summary>

/// Тесты для <see cref="DataAccessLayer.Configurations.EmployeeConfiguration"/>.

/// </summary>

[TestFixture]

internal sealed class EmployeeConfigurationTests : BaseConfigurationTests

{

[TearDown]

public void TearDown()

{

this.DataContext.ChangeTracker.Clear();

}

[Test]

public void AddEntityToDatabase\_Success()

{

// arrange

var name = new Name("Толстой", "Лев", "Николаевич");

var post = new Post("Тест", 1010010);

var employee = new Employee(name, new DateOnly(2002, 09, 08), Gender.Male, post);

// act

\_ = this.DataContext.Add(employee);

\_ = this.DataContext.SaveChanges(); // <-- если что-то плохо, то тут БУМ!

this.DataContext.ChangeTracker.Clear();

// assert

var result = this.DataContext.Find<Employee>(employee.Id);

Assert.That(result, Is.Not.Null);

Assert.That(result!.FullName, Is.EqualTo(employee.FullName));

}

}

}

**KidConfigirationTests**

// <copyright file="KidConfigurationTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Tests

{

using DemoSc;

using NUnit.Framework;

/// <summary>

/// Тесты для <see cref="DataAccessLayer.Configurations.KidConfiguration"/>.

/// </summary>

[TestFixture]

internal sealed class KidConfigurationTests : BaseConfigurationTests

{

[TearDown]

public void TearDown()

{

this.DataContext.ChangeTracker.Clear();

}

[Test]

public void AddEntityToDatabase\_Success()

{

// arrange

var name = new Name("Толстой", "Лев", "Николаевич");

var kidName = new Name("Бипкин", "Анатолий", "Васильевич");

var post = new Post("Тест", 1010010);

var employee = new Employee(name, new DateOnly(1977, 09, 08), Gender.Male);

var kid = new Kid(kidName, new DateOnly(2002, 09, 08), Gender.Male, employee);

// act

\_ = this.DataContext.Add(kid);

\_ = this.DataContext.SaveChanges(); // <-- если что-то плохо, то тут БУМ!

this.DataContext.ChangeTracker.Clear();

// assert

var result = this.DataContext.Find<Kid>(kid.Id);

Assert.That(result, Is.Not.Null);

Assert.That(result!.FullName, Is.EqualTo(kid.FullName));

}

}

}

**PostConfigurationsTest**

// <copyright file="PostConfigurationTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace DataAccessLayer.Tests

{

using DemoSc;

using NUnit.Framework;

/// <summary>

/// Тесты для <see cref="DataAccessLayer.Configurations.ShelfConfiguration"/>.

/// </summary>

[TestFixture]

internal sealed class PostConfigurationTests : BaseConfigurationTests

{

[TearDown]

public void TearDown()

{

this.DataContext.ChangeTracker.Clear();

}

[Test]

public void AddEntityToDatabase\_Success()

{

// arrange

var post = new Post("Тестовая", 100000);

// act

\_ = this.DataContext.Add(post);

\_ = this.DataContext.SaveChanges(); // <-- если что-то плохо, то тут БУМ!

this.DataContext.ChangeTracker.Clear();

// assert

var result = this.DataContext.Find<Post>(post.Id);

Assert.That(result, Is.Not.Null);

Assert.That(result!.Name, Is.EqualTo(post.Name));

}

}

}

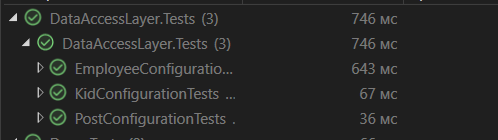


Рисунок 4 – Результат прохождения тестов

# Репозитории

**BaseRepository**

// <copyright file="PostRepositoryBase.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace Repository

{

using System;

using System.Linq;

using System.Linq.Expressions;

using DataAccessLayer;

using DemoSc;

/// <summary>

/// Базовый класс репозиториев.

/// </summary>

/// <typeparam name="TEntity"> Целевой тип сущности. </typeparam>

public abstract class BaseRepository<TEntity>

where TEntity : class, IEntity<TEntity>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="BaseRepository{TEntity}"/>.

/// </summary>

/// <param name="dataContext">Контекст доступа к данным.</param>

protected BaseRepository(DataContext dataContext)

{

this.DataContext = dataContext ?? throw new ArgumentNullException(nameof(dataContext));

}

/// <summary>

/// Контекст доступа к данным.

/// </summary>

public DataContext DataContext { get; }

/// <summary>

/// Создает сущность.

/// </summary>

/// <param name="entity">Сущность.</param>

/// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>

/// <returns>Контекст доступа к сущности.</returns>

public TEntity Create(TEntity entity, bool saveNow = true)

{

var result = this.DataContext.Add(entity).Entity;

\_ = this.Save(saveNow);

return result;

}

/// <summary>

/// Удаляет сущность.

/// </summary>

/// <param name="entity">Сущность.</param>

/// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>

/// <returns>Измененный контекст доступа к сущности.</returns>

public TEntity Delete(TEntity entity, bool saveNow = true)

{

var result = this.DataContext.Remove(entity).Entity;

\_ = this.Save(saveNow);

return result;

}

/// <summary>

/// Поиск множества сущностей по предикату (<paramref name="predicate"/>).

/// </summary>

/// <param name="predicate">Предикат, которому должна удовлетворять сушность.</param>

/// <returns>Множество (<see cref="IQueryable{TEntity}"/>) всех сущностей.</returns>

public IQueryable<TEntity> Filter(Expression<Func<TEntity, bool>> predicate) => this.GetAll().Where(predicate);

/// <summary>

/// Поиск сущности по предикату (<paramref name="predicate"/>).

/// </summary>

/// <param name="predicate">Предикат, которому должна удовлетворять сущность.</param>

/// <returns>Сущность или <see langword="null"/>.</returns>

public TEntity? Find(Expression<Func<TEntity, bool>> predicate) => this.GetAll().FirstOrDefault(predicate);

/// <summary>

/// Получение конкретной сущности по её идентификатору.

/// </summary>

/// <param name="id">Идентификатор сущности.</param>

/// <returns>Сушность.</returns>

public TEntity? Get(Guid id) => this.GetAll().SingleOrDefault(entity => entity.Id == id);

/// <summary>

/// Получение всех сущностей.

/// </summary>

/// <returns> Множество (<see cref="IQueryable{TEntity}"/>) всех сущностей.</returns>

public abstract IQueryable<TEntity> GetAll();

/// <summary>

/// Изменяет Сущность.

/// </summary>

/// <param name="entity">Сушность.</param>

/// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>

/// <returns>Измененный контекст доступа к сущности.</returns>

public TEntity Update(TEntity entity, bool saveNow = true)

{

var result = this.DataContext.Update(entity).Entity;

\_ = this.Save(saveNow);

return result;

}

/// <summary>

/// Сохраняет контекст в БД.

/// </summary>

/// <param name="saveNow">Надо ли сохранять сущность после изменения. </param>

/// <returns>Количество измененных сущностей.</returns>

private int Save(bool saveNow = true)

{

return saveNow

? this.DataContext.SaveChanges()

: 0;

}

}

}

**EmployeeRepository**

// <copyright file="EmployeeRepository.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace Repository

{

using System;

using System.Linq;

using DataAccessLayer;

using DemoSc;

using Microsoft.EntityFrameworkCore;

/// <summary>

/// Репозиторий для класса <see cref="DemoSc.Employee"/>.

/// </summary>

public sealed class EmployeeRepository : BaseRepository<Employee>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="EmployeeRepository"/>.

/// </summary>

/// <exception cref="ArgumentNullException">

/// В случае если <paramref title="dataContext"/> – <see langword="null"/>.

/// </exception>

public EmployeeRepository(DataContext dataContext)

: base(dataContext)

{

}

/// <summary>

/// Получает всех работников.

/// </summary>

/// <returns>Работники.</returns>

public override IQueryable<Employee> GetAll() => this.DataContext.Employees.Include(employee => employee.Post);

/// <summary>

/// Найти идентификатор работнка по его фамилии.

/// </summary>

/// <param name="familyName">Фамилия работнка.</param>

/// <returns>Идентификатор.</returns>

public Guid? GetId(string familyName)

{

return this.Find(employee => employee.FullName.FamilyName == familyName)?.Id;

}

/// <summary>

/// Получает список детей по идентификатору.

/// </summary>

/// <param name="id">Идентификатор работнка.</param>

/// <returns>Список детей.</returns>

public ISet<Kid>? GetKids(Guid id) => this.Find(employee => employee.Id == id)?.Kids;

}

}

**KidRepository**

namespace Repository

{

using System;

using System.Linq;

using DataAccessLayer;

using DemoSc;

using Microsoft.EntityFrameworkCore;

/// <summary>

/// Репозиторий для класса <see cref="DemoSc.Kid"/>.

/// </summary>

public sealed class KidRepository : BaseRepository<Kid>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="KidRepository"/>.

/// </summary>

/// <exception cref="ArgumentNullException">

/// В случае если <paramref title="dataContext"/> – <see langword="null"/>.

/// </exception>

public KidRepository(DataContext dataContext)

: base(dataContext)

{

}

/// <summary>

/// Получает всех детей.

/// </summary>

/// <returns>Дети.</returns>

public override IQueryable<Kid> GetAll() => this.DataContext.Kids;

/// <summary>

/// Найти идентификатор ребенка по его фамилии.

/// </summary>

/// <param name="familyName">Фамилия ребенка.</param>

/// <returns>Идентификатор.</returns>

public Guid? GetId(string familyName)

{

return this.Find(kid => kid.FullName.FamilyName == familyName)?.Id;

}

/// <summary>

/// Получает список работников по идентификатору.

/// </summary>

/// <param name="id">Идентификатор ребенка.</param>

/// <returns>Список работников.</returns>

public ISet<Employee>? GetEmployees(Guid id) => this.Find(kid => kid.Id == id)?.Employees;

}

}

**PostRepository**

// <copyright file="PostRepository.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace Repository

{

using System;

using System.Linq;

using System.Linq.Expressions;

using DataAccessLayer;

using DemoSc;

using Microsoft.EntityFrameworkCore;

/// <summary>

/// Репозиторий для класса <see cref="Domain.Shelf"/>.

/// </summary>

public sealed class PostRepository : BaseRepository<Post>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="PostfRepository"/>.

/// </summary>

/// <param name="dataContext"> Контекст доступа к данным.</param>

/// <exception cref="ArgumentNullException">

/// В случае если <paramref name="dataContext"/> – <see langword="null"/>.

/// </exception>

public PostRepository(DataContext dataContext)

: base(dataContext)

{

}

/// <inheritdoc/>

// @NOTE: IgnoreAutoIncludes()

public override IQueryable<Post> GetAll() => this.DataContext.Posts;

/// <summary>

/// Показать количество работников на этой должности.

/// </summary>

/// <param name="id">Идентификатор должности.</param>

/// <returns> Количество книг.</returns>

public int? GetCountEmployees(Guid id) => this.Get(id)?.Employees.Count;

/// <summary>

/// Показать количество работников на этой должности.

/// </summary>

/// <param name="name">Название должности.</param>

/// <returns>Количество книг.</returns>

public int? GetCountEmployees(string name)

{

var id = this.GetIdByName(name);

return id.HasValue

? this.GetCountEmployees(id.Value)

: null;

}

/// <summary>

/// Найти идентификатор по имени.

/// </summary>

/// <param name="name">Название должности.</param>

/// <returns>Идентификатор.</returns>

public Guid? GetIdByName(string name) => this.Find(post => post.Name == name)?.Id;

}

}

# Тесты на репозитории

**BaseRepositoryTests**

// <copyright file="BaseReposytoryTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace Repository.Tests

{

using System;

using DataAccessLayer;

using DemoSc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Logging;

/// <summary>

/// Базовый тип тестов для репозиториев.

/// </summary>

/// <typeparam name="TRepository"> Целевой тип тестируемого репозитория. </typeparam>

/// <typeparam name="TEntity"> Целевой тип сущности тестируемого репозитория. </typeparam>

internal abstract class BaseReposytoryTests<TRepository, TEntity>

where TRepository : BaseRepository<TEntity>

where TEntity : class, IEntity<TEntity>

{

private static readonly string ConnectionString = @"Data Source=.\tests.db";

private readonly ServiceProvider serviceProvider;

protected BaseReposytoryTests()

{

this.serviceProvider = new ServiceCollection()

.AddDbContext<DataContext>(

builder => builder.UseSqlite(ConnectionString)

.EnableDetailedErrors()

.EnableSensitiveDataLogging()

.LogTo(Console.WriteLine, LogLevel.Error))

.AddScoped<TRepository>()

.BuildServiceProvider();

}

protected DataContext DataContext

{

get => this.serviceProvider.GetService<DataContext>()

?? throw new Exception($"Cannot get {typeof(DataContext).Name}");

}

protected TRepository Repository

{

get => this.serviceProvider.GetService<TRepository>()

?? throw new Exception($"Cannot get {typeof(TRepository).Name}");

}

}

}

**EmployeeRepositoryTests**

// <copyright file="EmployeeRepositoryTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace Repository.Tests

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Xml.Linq;

using DemoSc;

using NUnit.Framework;

internal sealed class EmployeeRepositoryTests

: BaseReposytoryTests<EmployeeRepository, Employee>

{

[SetUp]

public void SetUp()

{

\_ = this.DataContext.Database.EnsureCreated();

}

[TearDown]

public void TearDown()

{

\_ = this.DataContext.Database.EnsureDeleted();

}

[Test]

public void Create\_ValidData\_Success()

{

// arrange

var name = new Name("Пупкин", "Василий", "Семенович");

var post = new Post("Тестовое название", 10000);

var employee = new Employee(name, new DateOnly(2002, 09, 08), Gender.Male, post);

// act

\_ = this.Repository.Create(employee);

// assert

var result = this.DataContext.Find<Employee>(employee.Id);

Assert.That(result, Is.EqualTo(employee));

}

[Test]

public void Update\_ValidData\_Success()

{

// arrange

var name = new Name("Пупкин", "Василий", "Семенович");

Post post = new ("Тестовое название", 10000);

var employee = new Employee(name, new DateOnly(2002, 09, 08), Gender.Male, post);

\_ = this.DataContext.Add(employee);

\_ = this.DataContext.SaveChanges();

// act

employee.DateBirth = new DateOnly(1828, 09, 28);

\_ = this.Repository.Update(employee);

// assert

var result = this.DataContext.Find<Employee>(employee.Id)?.DateBirth;

Assert.That(result, Is.EqualTo(employee.DateBirth));

}

[Test]

public void Delete\_ValidData\_Success()

{

// arrange

var name = new Name("Пупкин", "Василий", "Семенович");

Post post = new ("Тест", 10000);

var employee = new Employee(name, new DateOnly(2002, 09, 08), Gender.Male, post);

\_ = this.DataContext.Add(employee);

\_ = this.DataContext.SaveChanges();

// act

\_ = this.Repository.Delete(employee);

// assert

var result = this.DataContext.Find<Employee>(employee.Id);

Assert.That(result, Is.Null);

}

[Test]

public void GetIdByName\_FamilyName\_Success()

{

// arrange

var familyName = "Толстой";

Post post = new ("Тестовое название", 10000);

var employees = new[]

{

new Employee(new Name(familyName, "Лев", "Николаевич"), new DateOnly(2002, 09, 08), Gender.Male, post),

new Employee(new Name(familyName, "Алексей", "Константинович"), new DateOnly(2002, 09, 08), Gender.Male, post),

new Employee(new Name(familyName, "Алексей", "Николаевич"), new DateOnly(2002, 09, 08), Gender.Male, post),

};

this.DataContext.AddRange(employees);

\_ = this.DataContext.SaveChanges();

this.DataContext.ChangeTracker.Clear();

// act

var result = this.Repository.GetId(familyName);

// Act

Assert.That(

employees.Select(employee => employee.Id),

Has.One.EqualTo(result),

message: "Полученный идентификатор входит в множество идентификаторов целевых работников");

}

[Test]

public void GetKidsById\_ValidData\_Success()

{

var name = new Name("Толстой", "Лев", "Николаевич");

var post = new Post("Тест", 1010010);

var employee = new Employee(name, new DateOnly(1977, 09, 08), Gender.Male, post);

var kid = new Kid(new Name("Иванов", "Степан", "Анатольевич"), new DateOnly(2024, 09, 08), Gender.Male, employee);

var kids = new HashSet<Kid>();

\_ = kids.Add(kid);

\_ = this.DataContext.Add(employee);

\_ = this.DataContext.SaveChanges();

this.DataContext.ChangeTracker.Clear();

// act

var result = this.Repository.GetKids(employee.Id);

// assert

Assert.That(result, Is.EqualTo(kids));

}

}

}

**KidRepositoryTests**

using DemoSc;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Repository.Tests

{

internal class KidRepositoryTests

: BaseReposytoryTests<KidRepository, Kid>

{

[SetUp]

public void SetUp()

{

\_ = this.DataContext.Database.EnsureCreated();

}

[TearDown]

public void TearDown()

{

\_ = this.DataContext.Database.EnsureDeleted();

}

[Test]

public void Create\_ValidData\_Success()

{

// arrange

var name = new Name("Толстой", "Лев", "Николаевич");

var kidName = new Name("Бипкин", "Анатолий", "Васильевич");

var post = new Post("Тест", 1010010);

var employee = new Employee(name, new DateOnly(1977, 09, 08), Gender.Male, post);

var kid = new Kid(kidName, new DateOnly(2002, 09, 08), Gender.Male, employee);

// act

\_ = this.Repository.Create(kid);

// assert

var result = this.DataContext.Find<Kid>(kid.Id);

Assert.That(result, Is.EqualTo(kid));

}

[Test]

public void Delete\_ValidData\_Success()

{

// arrange

var name = new Name("Пупкин", "Василий", "Семенович");

Post post = new ("Тест", 10000);

var employee = new Employee(name, new DateOnly(1977, 09, 08), Gender.Male, post);

var kidName = new Name("Бипкин", "Анатолий", "Васильевич");

var kid = new Kid(kidName, new DateOnly(2002, 09, 08), Gender.Male, employee);

\_ = this.DataContext.Add(kid);

\_ = this.DataContext.SaveChanges();

// act

\_ = this.Repository.Delete(kid);

// assert

var result = this.DataContext.Find<Kid>(kid.Id);

Assert.That(result, Is.Null);

}

[Test]

public void GetIdByName\_FamilyName\_Success()

{

// arrange

var familyName = "Толстой";

var name = new Name("Бипкин", "Анатолий", "Васильевич");

var post = new Post("Тест", 1010010);

var employee = new Employee(name, new DateOnly(1977, 09, 08), Gender.Male, post);

var kids = new[]

{

new Kid(new Name(familyName, "Лев", "Николаевич"), new DateOnly(2002, 09, 08), Gender.Male, employee),

new Kid(new Name(familyName, "Алексей", "Константинович"), new DateOnly(2002, 09, 08), Gender.Male, employee),

new Kid(new Name(familyName, "Алексей", "Николаевич"), new DateOnly(2002, 09, 08), Gender.Male, employee),

};

this.DataContext.AddRange(kids);

\_ = this.DataContext.SaveChanges();

this.DataContext.ChangeTracker.Clear();

// act

var result = this.Repository.GetId(familyName);

// Act

Assert.That(

kids.Select(kid => kid.Id),

Has.One.EqualTo(result),

message: "Полученный идентификатор входит в множество идентификаторов целевых детей");

}

}

}

**PostRepositoryTests.cs**

// <copyright file="PostRepositoryTests.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace Repository.Tests

{

using DemoSc;

using NUnit.Framework;

/// <summary>

/// Модульные тесты для <see cref="ShelfRepository"/>.

/// </summary>

[TestFixture]

internal sealed class PostRepositoryTests

: BaseReposytoryTests<PostRepository, Post>

{

[SetUp]

public void SetUp()

{

\_ = this.DataContext.Database.EnsureCreated();

}

[TearDown]

public void TearDown()

{

\_ = this.DataContext.Database.EnsureDeleted();

}

[Test]

public void Create\_ValidData\_Success()

{

// arrange

var post = new Post("Уборщик", 10101001);

// act

\_ = this.Repository.Create(post);

// arrange

var result = this.DataContext.Find<Post>(post.Id);

Assert.That(result, Is.Not.Null);

Assert.That(result!.Name, Is.EqualTo(post.Name));

}

[Test]

public void Get\_ValidData\_Success()

{

// arrange

var shelf = new Post("Уборщик", 10101010);

this.DataContext.Add(shelf);

this.DataContext.SaveChanges();

// act

var result = this.Repository.Get(shelf.Id);

// assert

Assert.That(result, Is.Not.Null);

Assert.That(result!.Name, Is.EqualTo(shelf.Name));

}

[Test]

public void Update\_ValidData\_Success()

{

// arrange

var newName = "Новое имя";

var setSalary = 8994878438734;

var post = new Post("Тестовая", 1010101001);

this.DataContext.Add(post);

this.DataContext.SaveChanges();

// act

post.Name = newName;

post.Salary = setSalary;

var result = this.Repository.Update(post);

// assert

Assert.That(result, Is.Not.Null);

Assert.That(result!.Name, Is.EqualTo(newName));

}

[Test]

public void Delete\_ValidData\_Success()

{

// arrange

var post = new Post("Уборщик", 10101010);

this.DataContext.Add(post);

this.DataContext.SaveChanges();

// act

\_ = this.Repository.Delete(post);

// arrange

var result = this.DataContext.Find<Post>(post.Id);

Assert.That(result, Is.Null);

}

}

}

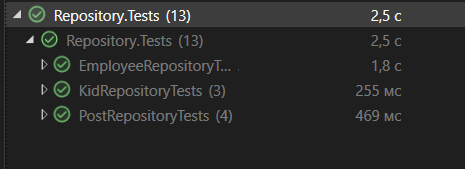
****

Рисунок 5 – Результаты выполнения тестов

# точка входа в программу

**Programm.cs**

// <copyright file="Program.cs" company="Гылыба Л.Д.">

// Copyright (c) Гылыба Л.Д.. All rights reserved.

// </copyright>

namespace Demo

{

using System;

using System.Linq;

using DataAccessLayer;

using DemoSc;

using Repository;

/// <summary>

/// Точка входа.

/// </summary>

internal static class Program

{

private static void Main(string[] args)

{

var dataContext = new DataContext();

Post post1 = new Post("Замазыватель трещен", 200000);

Employee employee1 = new Employee(new Name("Абраморвич", "Изя", "Шмуликович"), new DateOnly(1977, 12, 12), Gender.Male, post1);

try

{

var postRepository = new PostRepository(dataContext);

\_ = postRepository.Create(new Post("Уборщик", 10000000));

var employeeRepository = new EmployeeRepository(dataContext);

\_ = employeeRepository.Create(new Employee(new Name("Пупкин", "Вася", "Артемович"), new DateOnly(1984, 01, 01), Gender.Male, new Post("Сваязабиватель", 101010101)));

var kidRepository = new KidRepository(dataContext);

\_ = kidRepository.Create(new Kid(new Name("Жиганов", "Лимон", "Симотичнович"), new DateOnly(2000, 10, 10), Gender.Male, employee1));

foreach (var post in postRepository.GetAll())

{

Console.WriteLine($"{post.Id} --> {post.Name}");

}

foreach (var employee in employeeRepository.GetAll())

{

Console.WriteLine($"{employee.Id} --> {employee.FullName}");

}

foreach (var kid in kidRepository.GetAll())

{

Console.WriteLine($"{kid.Id} --> {kid.FullName}");

}

}

catch (Exception exception)

{

Console.Error.WriteLine(exception.Message);

}

}

}

}

**select** \* **from** "Library".public."Posts" *p*

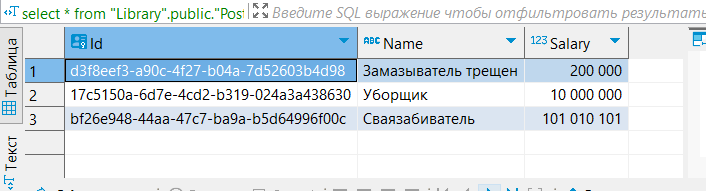


Рисунок 6 – Результат выполнения запроса

**select** \* **from** "Library".public."Employees" *e*

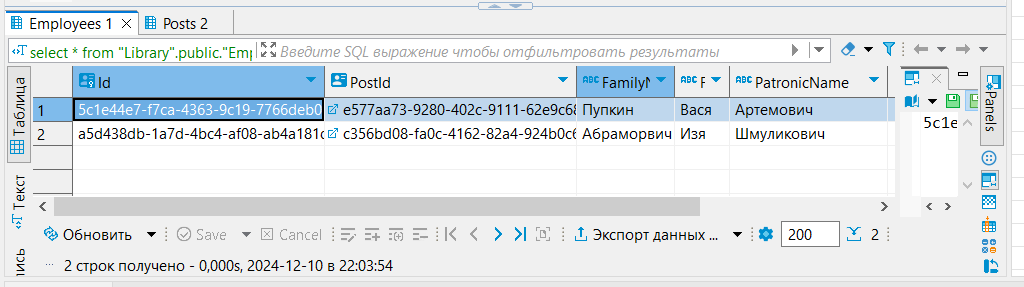
**

Рисунок 7 – Результат выполнения запроса

**select** \* **from** "Library".public."Kids" *k*

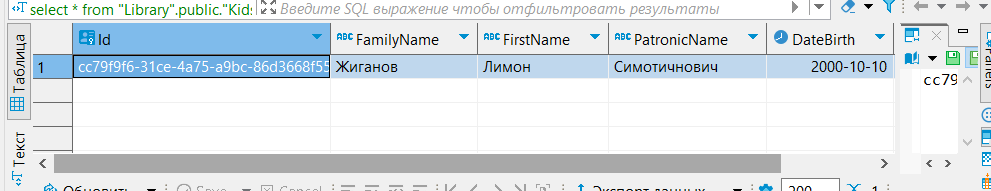


Рисунок 8 – Результат выполнения запроса

# Список литературы

* + - 1. Васильева М.А., Хобта Д.О., Фильтрация набора данных. Рекомендации по выполнению работы и перечень типовых заданий: Учебно-методическое пособие. Издание второе, исправленное и дополненное–М.:РУТ(МИИТ). 2023.–105с.
      2. Васильева М.А., Меркулов Д.А. Группировка и обобщение данных. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 46–с.
      3. Васильева М.А., Ракинцев Н.А. Соединение данных из множества таблиц. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 63–с.
      4. Балакина Е.П., Васильева М.А., Филипченко К.М. Информационное обеспечение систем управления. Методические указания к курсовому проектированию. Учебно-методическое пособие. Издание второе, исправленное и дополненное, 2023.102–с.
      5. SQLAlchemy [Электронный ресурс] // SQLAlchemy [сайт]. URL: <https://www.sqlalchemy.org/> (дата обращения 24.10.2023).
      6. PostgreSQL [Электронный ресурс] // PostgreSQL [сайт]. — URL: <https://www.postgresql.org/> (дата обращения 24.10.2023).