**Министерство транспорта Российской Федерации**

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

**Институт транспортной техники и систем управления**

**Кафедра «Управление и защита информации»**

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

**«Основы построения защищенных баз данных»**

**на тему  
«Создание ORM для реляционной БД «Школа»»**

Выполнил: ст. гр. ТКИ–511  
Фаюстов Н.С.  
Проверил: Васильева М.А.

Москва 2022

# Задание

Разработать ORM на языке C# для хранения экземпляров классов базы данных школы. База данных создается для информационного обслуживания администрации школы. Реализовать принцип CRUD (Create, Rewrite, Update, Delete).

# Листинг программы

1) Demo.Program:

{

using System;

using System.Linq;

using DataAccess;

using DataAccess.Repositories;

using Domain;

internal class Program

{

private static void Main()

{

var teacher = new Teacher(1, "pasnumber 0000 00001", "contnumber 001");

var teacher1 = new Teacher(2, "pasnumber 0000 00002", "contnumber 002");

var teacher2 = new Teacher(3, "pasnumber 0000 00003", "contnumber 003");

var teacher3 = new Teacher(4, "pasnumber 0000 00004", "contnumber 004");

var class1 = new Class(1, "1-A", teacher);

var class2 = new Class(1, "2-Б", teacher1);

var class3 = new Class(1, "3-В", teacher2);

var class4 = new Class(1, "4-Г", teacher3);

Console.WriteLine($"{class1} {teacher}");

var settings = new Settings();

settings.AddDatabaseServer(@"DESKTOP-1U2HID6\SQLEXPRESS");

settings.AddDatabaseName("ClassServer");

using var sessionFactory = Configurator.GetSessionFactory(settings, showSql: true);

using (var session = sessionFactory.OpenSession())

{

session.Save(class1);

session.Save(class2);

session.Save(class3);

session.Save(class4);

session.Save(teacher);

session.Save(teacher1);

session.Save(teacher2);

session.Save(teacher3);

session.Flush();

}

using (var session = sessionFactory.OpenSession())

{

var repoClass = new ClassRepository();

Console.WriteLine("All classes:");

repoClass.GetAll(session)

.ToList().ForEach(Console.WriteLine);

Console.WriteLine(new string('-', 25));

var repoTeacher = new TeacherRepository();

Console.WriteLine("All teachers:");

repoTeacher.GetAll(session)

.ToList().ForEach(Console.WriteLine);

Console.WriteLine(new string('-', 25));

}

}

}

}

2) Domain.Teacher:

namespace Domain

{

using System;

using System.Collections.Generic;

using Staff.Extensions;

public class Teacher : IEquatable<Teacher>

{

public Teacher(int id, string passport, string employmentContractNumber)

{

this.ID = id;

this.Passport = passport.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(passport));

this.EmploymentContractNumber = employmentContractNumber.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(employmentContractNumber));

}

[Obsolete("For ORM", true)]

protected Teacher()

{

}

public virtual int ID { get; protected set; }

public virtual string Passport { get; protected set; }

public virtual string EmploymentContractNumber { get; protected set; }

public virtual string FullInfo => $"{this.Passport} {this.EmploymentContractNumber}".Trim();

public virtual ISet<Class> Classes { get; protected set; } = new HashSet<Class>();

public virtual bool AddClass(Class class1)

{

return class1 == null

? throw new ArgumentNullException(nameof(class1))

: this.Classes.Add(class1);

}

public override string ToString() => this.FullInfo;

public override bool Equals(object obj)

{

return !ReferenceEquals(null, obj) && (ReferenceEquals(this, obj) || this.Equals(obj as Teacher));

}

public virtual bool Equals(Teacher other)

{

return !ReferenceEquals(null, other) && (ReferenceEquals(this, other) || this.ID == other.ID);

}

public override int GetHashCode() => this.ID;

}

}

3) Domain.Class

namespace Domain

{

using System;

using System.Collections.Generic;

using System.Linq;

using Staff.Extensions;

public class Class

{

public Class(int id, string className, params Teacher[] teachers)

: this(id, className, new HashSet<Teacher>(teachers))

{

}

public Class(int id, string className, ISet<Teacher> teachers = null)

{

this.ID = id;

this.ClassName = className.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(className));

foreach (var teacher in teachers ?? Enumerable.Empty<Teacher>())

{

this.Teachers.Add(teacher);

teacher.AddClass(this);

}

}

[Obsolete("For ORM", true)]

protected Class()

{

}

public virtual int ID { get; protected set; }

public virtual string ClassName { get; protected set; }

public virtual ISet<Teacher> Teachers { get; protected set; } = new HashSet<Teacher>();

/// <inheritdoc/>

public override string ToString() => $"{this.ClassName} {this.Teachers.Join()}".Trim();

}

}

4) DataAccess.Configurator:

namespace DataAccess

{

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using NHibernate;

using NHibernate.Tool.hbm2ddl;

public static class Configurator

{

private static FluentConfiguration fluentConfiguration;

public static ISessionFactory GetSessionFactory(

Settings settings,

Assembly assembly = null,

bool showSql = false)

{

return GetConfiguration(settings, assembly ?? Assembly.GetExecutingAssembly(), showSql)

.BuildSessionFactory();

}

private static FluentConfiguration GetConfiguration(

Settings settings,

Assembly assembly,

bool showSql = false)

{

if (fluentConfiguration is null)

{

var databaseConfiguration = MsSqlConfiguration.MsSql2012.ConnectionString(

x => x

.Server(settings.GetDatabaseServer())

.Database(settings.GetDatabaseName())

.TrustedConnection());

if (showSql)

{

databaseConfiguration = databaseConfiguration.ShowSql().FormatSql();

}

fluentConfiguration = Fluently.Configure()

.Database(databaseConfiguration)

.Mappings(m => m.FluentMappings.AddFromAssembly(assembly))

.ExposeConfiguration(BuildSchema);

}

return fluentConfiguration;

}

private static void BuildSchema(NHibernate.Cfg.Configuration configuration)

{

new SchemaExport(configuration).Execute(true, true, false);

}

}

}

5) DataAccess.Settings:

namespace DataAccess

{

using System;

using System.Configuration;

public sealed class Settings

{

private string databaseServerName;

private string databaseName;

public void AddDatabaseServer(string serverName)

{

this.databaseServerName = serverName;

}

public string GetDatabaseServer()

{

return this.databaseServerName;

}

public void AddDatabaseName(string databaseName)

{

this.databaseName = databaseName;

}

public string GetDatabaseName()

{

return this.databaseName;

}

}

}

6) DataAccess.Mappings.ClassMap:

namespace DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Domain;

internal class ClassMap : ClassMap<Class>

{

public ClassMap()

{

this.Table("Classes");

this.Id(x => x.ID);

this.Map(x => x.ClassName)

.Not.Nullable();

this.HasManyToMany(x => x.Teachers)

.Cascade.Delete()

.Inverse();

}

}

}

7) DataAccess.Mappings.TeacherMap:

namespace DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Domain;

internal class TeacherMap : ClassMap<Teacher>

{

public TeacherMap()

{

this.Table("Teachers");

this.Id(x => x.ID);

this.Map(x => x.Passport)

.Not.Nullable();

this.Map(x => x.EmploymentContractNumber)

.Not.Nullable();

this.HasManyToMany(x => x.Classes)

.Cascade.Delete();

}

}

}

8) DataAccess.Repositories.ClassRepository:

namespace DataAccess.Repositories

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using DataAccess.Repositories.Abstraction;

using Domain;

using NHibernate;

public class ClassRepository : IRepository<Class>

{

public Class Get(ISession session, int id) =>

session?.Get<Class>(id);

public Class Find(ISession session, Expression<Func<Class, bool>> predicate)

{

return this.GetAll(session).FirstOrDefault(predicate);

}

public IQueryable<Class> GetAll(ISession session) =>

session?.Query<Class>();

public IQueryable<Class> Filter(ISession session, Expression<Func<Class, bool>> predicate)

{

return this.GetAll(session).Where(predicate);

}

public bool Create<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Delete<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Update<TEntity>(TEntity oldEntity, TEntity newEntity)

{

throw new NotImplementedException();

}

}

}

9) DataAccess.Repositories.TeacherRepository:

namespace DataAccess.Repositories

{

using System;

using System.Collections.Generic;

using System.Linq;

using DataAccess.Repositories.Abstraction;

using Domain;

using NHibernate;

public class TeacherRepository : IRepository<Teacher>

{

public bool Create<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Delete<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public IQueryable<Teacher> Filter(ISession session, System.Linq.Expressions.Expression<Func<Teacher, bool>> predicate)

{

return this.GetAll(session)

.Where(predicate);

}

public Teacher Find(ISession session, System.Linq.Expressions.Expression<Func<Teacher, bool>> predicate)

{

return this.GetAll(session)

.FirstOrDefault(predicate);

}

public Teacher Get(ISession session, int id) =>

session?.Get<Teacher>(id);

public IQueryable<Teacher> GetAll(ISession session) =>

session?.Query<Teacher>();

public bool Update<TEntity>(TEntity oldEntity, TEntity newEntity)

{

throw new NotImplementedException();

}

}

}

10) DataAccess.Repositories.Abstraction.IRepository:

namespace DataAccess.Repositories.Abstraction

{

using System;

using System.Linq;

using System.Linq.Expressions;

using NHibernate;

public interface IRepository<TEntity>

{

TEntity Get(ISession session, int id);

TEntity Find(ISession session, Expression<Func<TEntity, bool>> predicate);

IQueryable<TEntity> GetAll(ISession session);

IQueryable<TEntity> Filter(ISession session, Expression<Func<TEntity, bool>> predicate);

bool Create<TEntity>(TEntity entity);

bool Delete<TEntity>(TEntity entity);

bool Update<TEntity>(TEntity oldEntity, TEntity newEntity);

}

}

11) Staff.Extension.IEnumerable:

namespace Staff.Extensions

{

using System.Collections.Generic;

public static class IEnumerableExtension

{

public static string Join<T>(this IEnumerable<T> collection, string separator = ", ") => string.Join(separator, collection);

}

}

12) Staff.Extension.StringExtensions:

namespace Staff.Extensions

{

public static class StringExtensions

{

public static bool IsNullOrEmpty(this string value) => string.IsNullOrEmpty(value);

public static string TrimOrNull(this string value)

{

var trimmedValue = value?.Trim(' ', '\n', '\t', '\r', '\0');

return trimmedValue.IsNullOrEmpty()

? null

: trimmedValue;

}

}

}

# Результаты работы программы

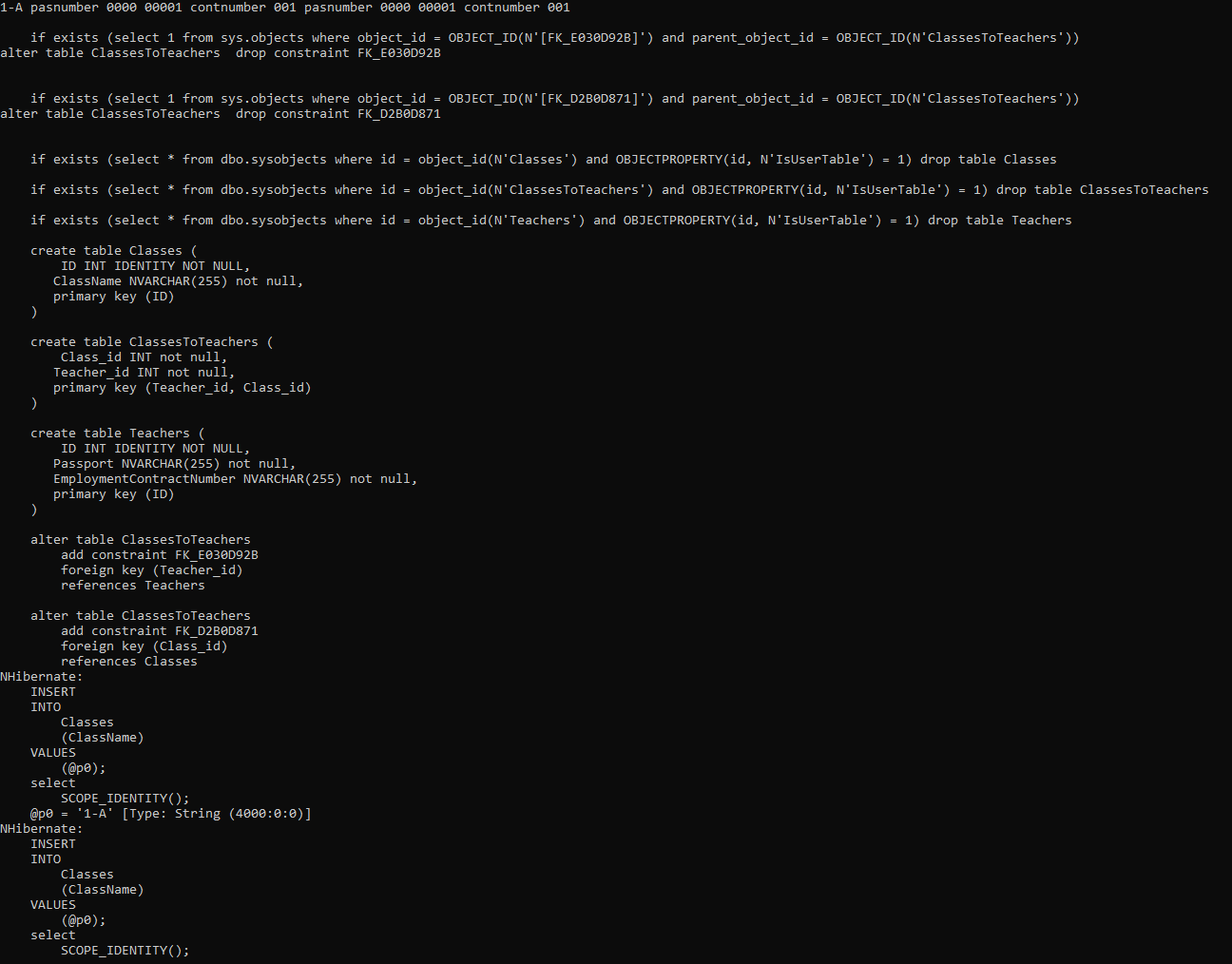


Рисунок 1

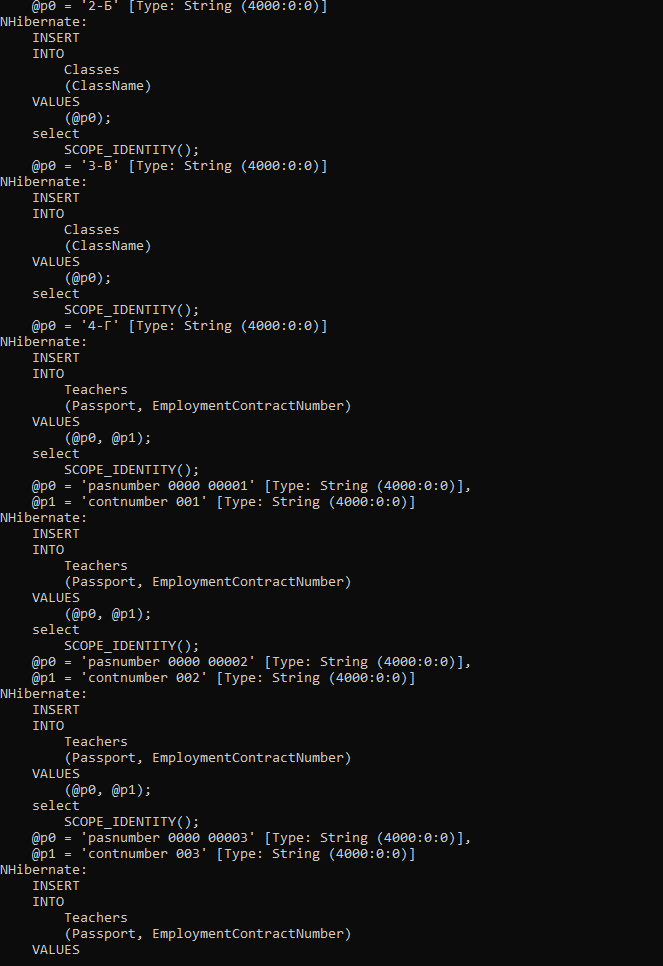


Рисунок 2

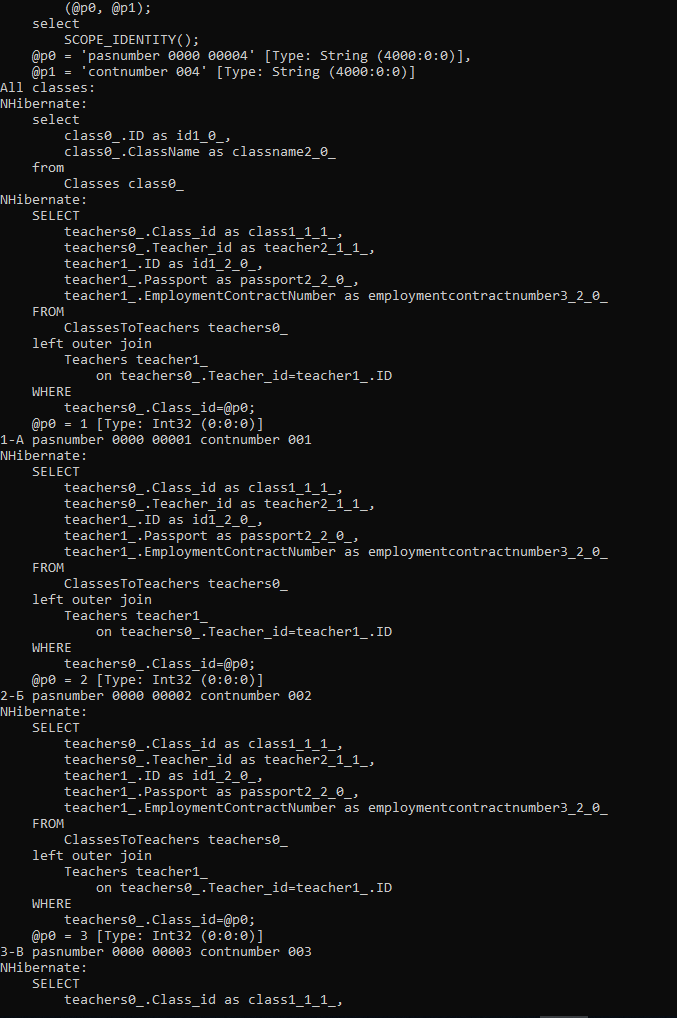


Рисунок 3

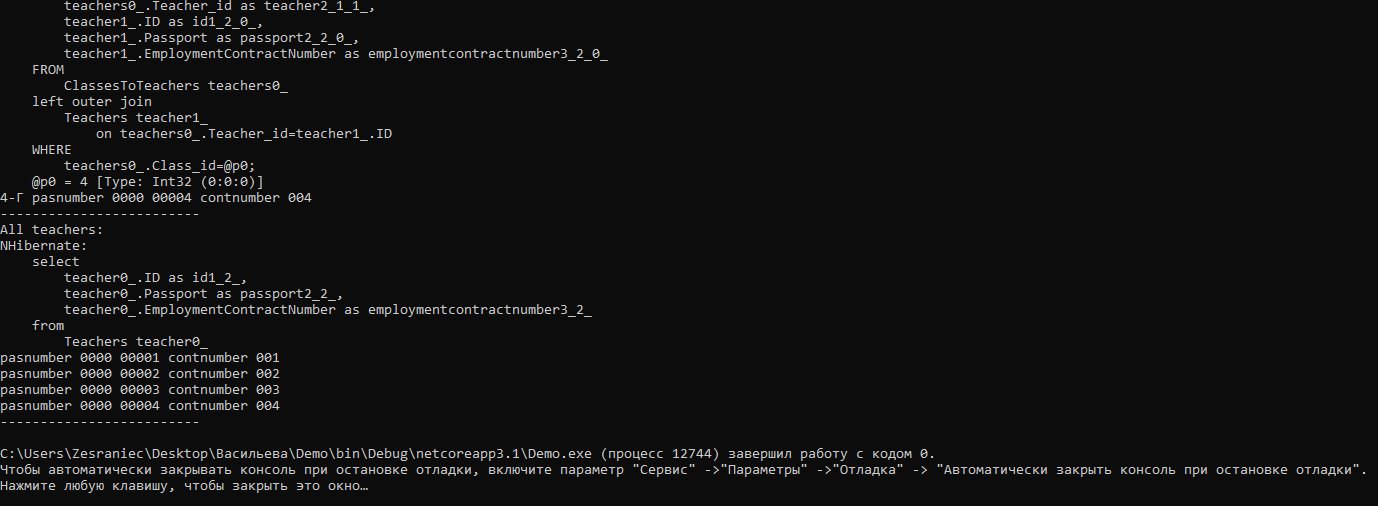


Рисунок 4

# Результаты прохождения тестов

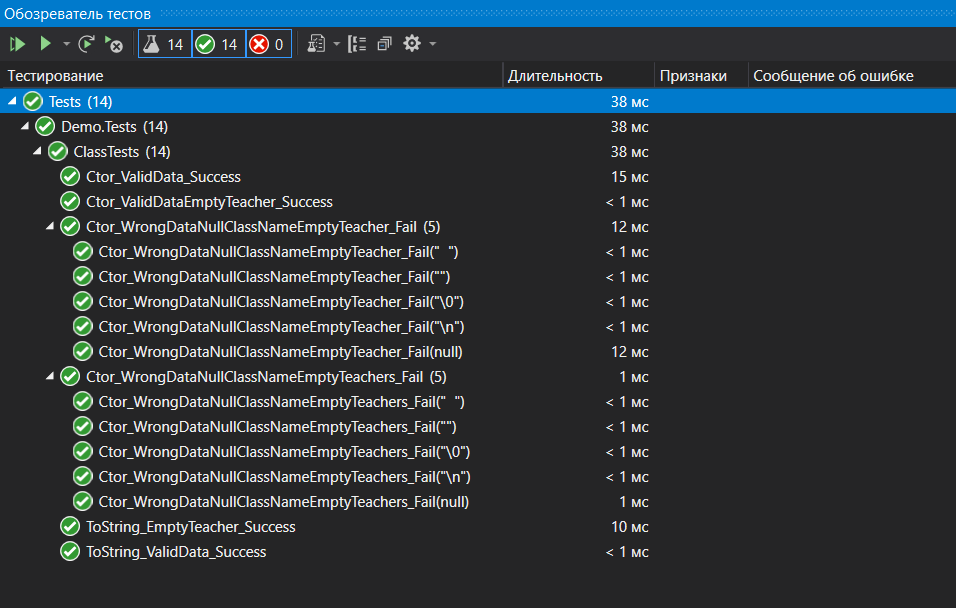


Рисунок 5