**Министерство транспорта Российской Федерации**

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

**Институт транспортной техники и систем управления**

**Кафедра «Управление и защита информации»**

Создание ORM для реляционной БД «Поликлиника»»

Выполнил: ст. гр. ТКИ–511  
Газизов Ф.М.  
Проверил: Васильева М.А.

Москва 2021

Оглавление

[Задание 3](#_Toc96718707)

[Листинг программы 3](#_Toc96718708)

[Реализация программы 12](#_Toc96718709)

[Тесты 17](#_Toc96718710)

# Задание

На языке C# разработать ORM для хранения базы данных поликлиники. Поликлиника обеспечивает прием пациентов врачами. Реализовать принцип CRUD.

# Листинг программы

**Hospital.Demo.program**

namespace Hospital.Demo

{

using System;

using System.Linq;

using Hospital.DataAccess;

using Hospital.DataAccess.Repositories;

using Hospital.Domain;

using NHibernate.Util;

internal class Program

{

/// <summary>

/// Главный метод

/// </summary>

/// <param name="args"></param>

private static void Main()

{

var doctor = new Doctor(1, "Быстряков", "Павел", "терапевт", "Валерьевич");

var doctor1 = new Doctor(2, "Кулакова", "Анна", "стоматолог");

var doctor2 = new Doctor(3, "Кусяка", "Дмитрий", "невролог", "Валентинович");

var doctor3 = new Doctor(4, "Черный", "Александр", "лор", "Максимович");

var pacient = new Pacient(1, "Иванов", "Сергей", "12345", "Анатольевич", doctor);

var pacient1 = new Pacient(2, "Петров", "Георгий", "67890", "Владимирович", doctor1);

var pacient2 = new Pacient(3, "Одинцов", "Михаил", "54321", "Олегович", doctor2);

Console.WriteLine($"{pacient} {doctor}");

var settings = new Settings();

settings.AddDatabaseServer(@"DESKTOP-SMJ2IL0\SQLEXPRESS");

settings.AddDatabaseName("HospitalServer");

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

using (var session = sessionFactory.OpenSession())

{

session.Save(pacient);

session.Save(pacient1);

session.Save(pacient2);

session.Save(doctor);

session.Save(doctor1);

session.Save(doctor2);

session.Save(doctor3);

session.Flush();

}

using (var session = sessionFactory.OpenSession())

{

var repoPacient = new PacientRepository();

Console.WriteLine("All pacients:");

repoPacient.GetAll(session)

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

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

//Console.WriteLine($"{0} {1} {2} {3}");

var repoDoctor = new DoctorRepository();

Console.WriteLine("All doctors:");

repoDoctor.GetAll(session)

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

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

}

}

}

}

**Hospital.Domain.Doctor**

namespace Hospital.Domain

{

using System;

using System.Collections.Generic;

using Hospital.Staff.Extensions;

public class Doctor : IEquatable<Doctor>

{

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Doctor"/>

/// </summary>

/// <param name="id"> Идентефикатор </param>

/// <param name="lastName"> Фамилия </param>

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

/// <param name="MiddleName"> Отчество </param>

/// <param name="specialization"> Специализация </param>

/// <exception cref="ArgumentOutOfRangeException">

/// В случае если <paramref name="lastName"/> или <paramref name="firstName"/> или <paramref name="specialization"/> <see langword="null"/>,

/// пустая строка или строка, содержащая только пробельные символы.

/// </exception>

public Doctor(int id, string lastName, string firstName, string specialization, string middleName = null)

{

this.ID = id;

this.LastName = lastName.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(lastName));

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

this.Specialization = specialization.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(specialization));

this.MiddleName = middleName.TrimOrNull();

}

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Doctor"/>

/// </summary>

[Obsolete("For ORM", true)]

protected Doctor()

{

}

/// <summary>

/// Уникальный идентефикатор

/// </summary>

public virtual int ID { get; protected set; }

/// <summary>

/// Фамилия

/// </summary>

public virtual string LastName { get; protected set; }

/// <summary>

/// Имя

/// </summary>

public virtual string FirstName { get; protected set; }

/// <summary>

/// Специализация

/// </summary>

public virtual string Specialization { get; protected set; }

/// <summary>

/// Отчество

/// </summary>

public virtual string MiddleName { get; protected set; }

/// <summary>

/// Полная информация о враче

/// </summary>

public virtual string FullInfDoctor => $"{this.LastName} {this.FirstName} {this.Specialization} {this.MiddleName}.".Trim();

/// <summary>

/// Множество пациентов

/// </summary>

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

/// <summary>

/// Метод, добавляющий пациента доктору.

/// </summary>

/// <param name="pacient"> Добавляем пациента. </param>

/// <returns>

/// Флаг успешности выполнения операции:

/// <see langword="true"/> – пациент был успешно добавлен,

/// <see langword="false"/> в противном случае.

/// </returns>

/// <exception cref="ArgumentNullException">

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

/// </exception>

public virtual bool AddPacient(Pacient pacient)

{

return pacient == null

? throw new ArgumentNullException(nameof(pacient))

: this.Pacients.Add(pacient);

}

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

public override bool Equals(object obj)

{

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

}

public virtual bool Equals(Doctor other)

{

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

}

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

}

}

**Hospital.Domain.Pacient**

namespace Hospital.Domain

{

using System;

using System.Collections.Generic;

using System.Linq;

using Hospital.Staff.Extensions;

/// <summary>

/// Пациент

/// </summary>

public class Pacient

{

/// <summary>

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

/// </summary>

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

/// <param name="lastName"> Фамилия </param>

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

/// <param name="MiddleName"> Отчество </param>

/// <param name="policie"> Номер полиса </param>

/// <param name="doctors"> . </param>

public Pacient(int id, string lastName, string firstName, string policie, string middleName, params Doctor[] doctors)

: this(id, firstName, lastName, policie, middleName, new HashSet<Doctor>(doctors))

{

}

/// <summary>

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

/// </summary>

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

/// <param name="lastName"> Фамилия </param>

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

/// <param name="MiddleName"> Отчество </param>

/// <param name="policie"> Номер полиса </param>

/// <param name="doctors"> Множество врачей. </param>

public Pacient(int id, string lastName, string firstName, string policie, string middleName = null, ISet<Doctor> doctors = null)

{

this.ID = id;

this.LastName = lastName.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(lastName));

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

this.Policie = policie.TrimOrNull() ?? throw new ArgumentOutOfRangeException(nameof(policie));

this.MiddleName = middleName.TrimOrNull();

foreach (var doctor in doctors ?? Enumerable.Empty<Doctor>())

{

this.Doctors.Add(doctor);

doctor.AddPacient(this);

}

}

/// <summary>

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

/// </summary>

[Obsolete("For ORM", true)]

protected Pacient()

{

}

/// <summary>

/// Идентификатор.

/// </summary>

public virtual int ID { get; protected set; }

/// <summary>

/// Фамилия

/// </summary>

public virtual string LastName { get; protected set; }

/// <summary>

/// Имя

/// </summary>

public virtual string FirstName { get; protected set; }

/// <summary>

/// Специализация

/// </summary>

public virtual string Policie { get; protected set; }

/// <summary>

/// Отчество

/// </summary>

public virtual string MiddleName { get; protected set; }

/// <summary>

/// Абоненты

/// </summary>

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

/// <inheritdoc/>

public override string ToString() => $"{this.FirstName} {this.LastName} {this.Policie} {this.MiddleName} {this.Doctors.Join()}".Trim();

}

}

**Hospital.DataAccess.Configurator**

namespace Hospital.DataAccess

{

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using NHibernate;

using NHibernate.Tool.hbm2ddl;

/// <summary>

/// Настройщик подключения к БД и поставщик фабрики сессий.

/// </summary>

public static class Configurator

{

/// <summary>

/// Конфигурация.

/// </summary>

private static FluentConfiguration fluentConfiguration;

/// <summary>

/// Генерирует фабрику сессий (<see cref="ISessionFactory"/>).

/// </summary>

/// <param name="settings"> Настройки. </param>

/// <param name="assembly"> Целевая сборка. </param>

/// <param name="showSql"> Показывать генерируемый SQL-код. </param>

/// <returns> Фабрику сессий. </returns>

public static ISessionFactory GetSessionFactory(

Settings settings,

Assembly assembly = null,

bool showSql = false)

{

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

.BuildSessionFactory();

}

/// <summary>

/// Возвращаем конфигурацию по правилам.

/// </summary>

/// <param name="settings"> Установки названия сервера БД и имени БД. </param>

/// <param name="assembly"> Целевая сборка. </param>

/// <param name="showSql"> Показывать генерируемый SQL-код. </param>

/// <returns> конфигурацию по правилам. </returns>

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;

}

/// <summary>

/// Метод, порождающий таблицы (если их не было в БД) по конфигурации.

/// </summary>

/// <remarks> Необходимо только для создания схемы БД из ничего. </remarks>

/// <param name="configuration"> Конфигурация ORM, содержащая правила отображения. </param>

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

{

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

}

}

}

**Hospital.DataAccess.Settings**

namespace Hospital.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;

}

}

}

**Hospital.DataAccess.Mappings.DoctorMap**

namespace Hospital.DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Hospital.Domain;

internal class DoctorMap : ClassMap<Doctor>

{

public DoctorMap()

{

this.Table("Doctors");

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

this.Map(x => x.LastName)

.Not.Nullable();

this.Map(x => x.FirstName)

.Not.Nullable();

this.Map(x => x.Specialization)

.Not.Nullable();

this.Map(x => x.MiddleName)

.Nullable();

this.HasManyToMany(x => x.Pacients)

.Cascade.Delete();

}

}

}

**Hospital.DataAccess.Mappings.PacientMap**

namespace Hospital.DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Hospital.Domain;

internal class PacientMap : ClassMap<Pacient>

{

public PacientMap()

{

this.Table("Pacients");

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

this.Map(x => x.LastName)

.Not.Nullable();

this.Map(x => x.FirstName)

.Not.Nullable();

this.Map(x => x.Policie)

.Not.Nullable();

this.Map(x => x.MiddleName)

.Nullable();

this.HasManyToMany(x => x.Doctors)

.Cascade.Delete()

.Inverse();

}

}

}

**Hospital.DataAccess.Repositories.DoctorRepository**

namespace Hospital.DataAccess.Repositories

{

using System;

using System.Linq;

using Hospital.DataAccess.Repositories.Abstraction;

using Hospital.Domain;

using NHibernate;

public class DoctorRepository : IRepository<Doctor>

{

public bool Create<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Delete<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

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

{

return this.GetAll(session)

.Where(predicate);

}

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

{

return this.GetAll(session)

.FirstOrDefault(predicate);

}

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

session?.Get<Doctor>(id);

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

session?.Query<Doctor>();

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

{

throw new NotImplementedException();

}

}

}

**Hospital.DataAccess.Repositories.PacientRepository**

namespace Hospital.DataAccess.Repositories

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using Hospital.DataAccess.Repositories.Abstraction;

using Hospital.Domain;

using NHibernate;

public class PacientRepository : IRepository<Pacient>

{

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

session?.Get<Pacient>(id);

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

{

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

}

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

session?.Query<Pacient>();

public IQueryable<Pacient> Filter(ISession session, Expression<Func<Pacient, 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();

}

}

}

**Hospital.Staff.Extensions.lEnumerableExtention**

namespace Hospital.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);

}

}

**Hospital.Staff.Extensions.StringExtention**

namespace Hospital.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

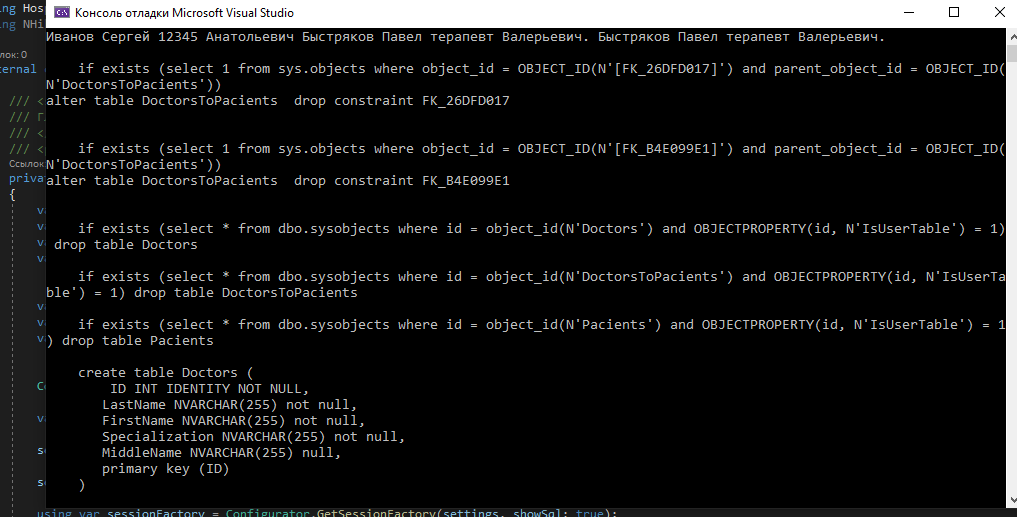
: trimmedValue;

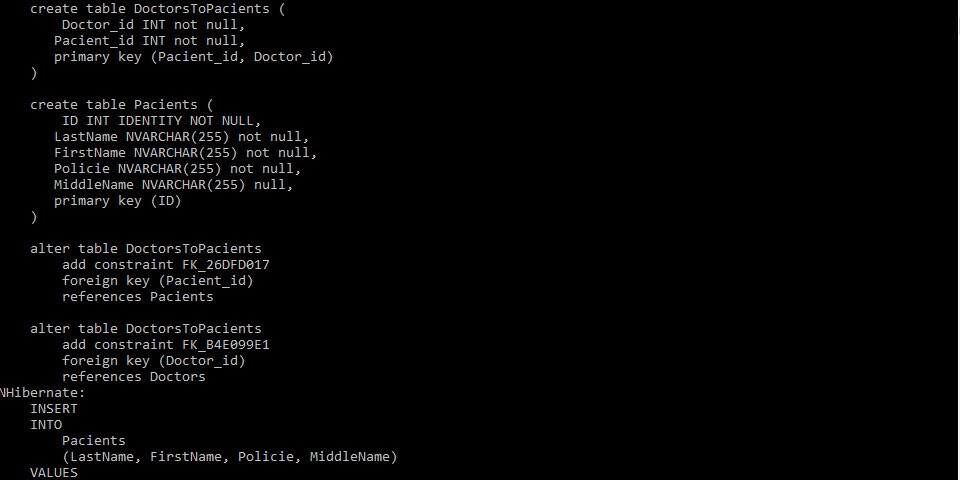
}

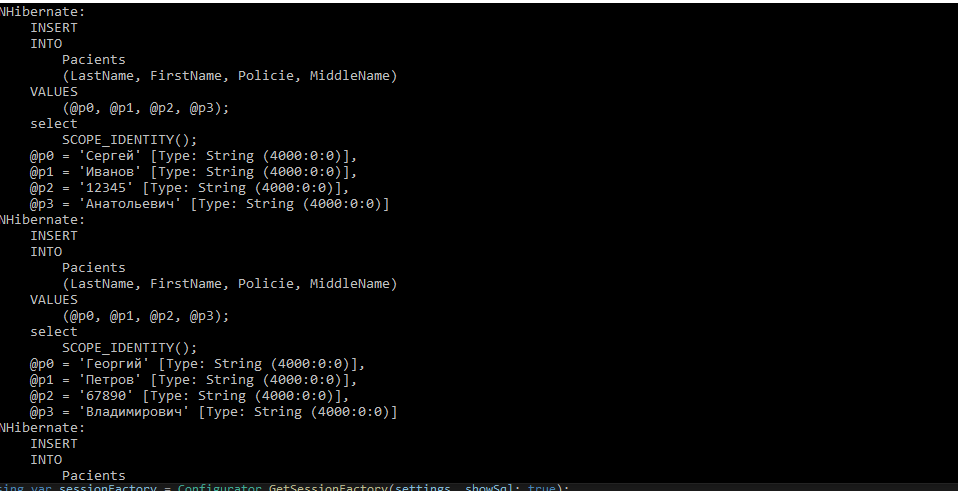
}

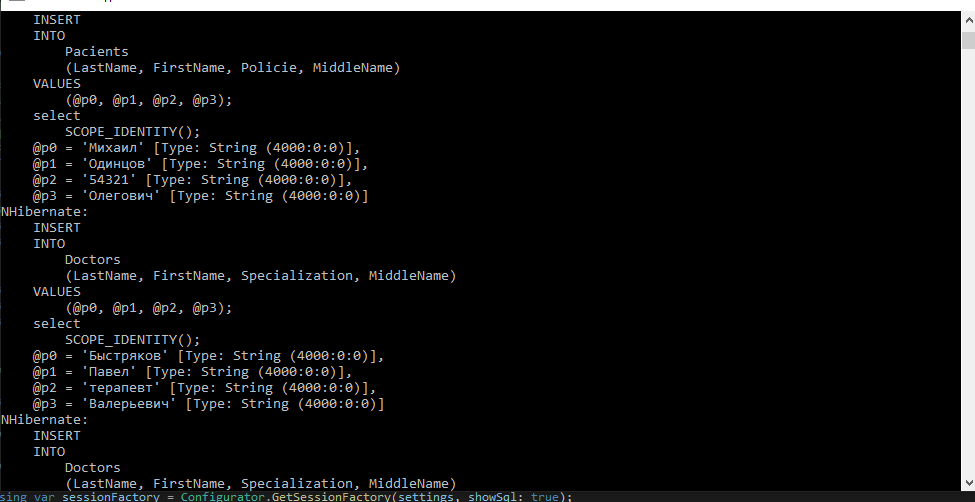
}

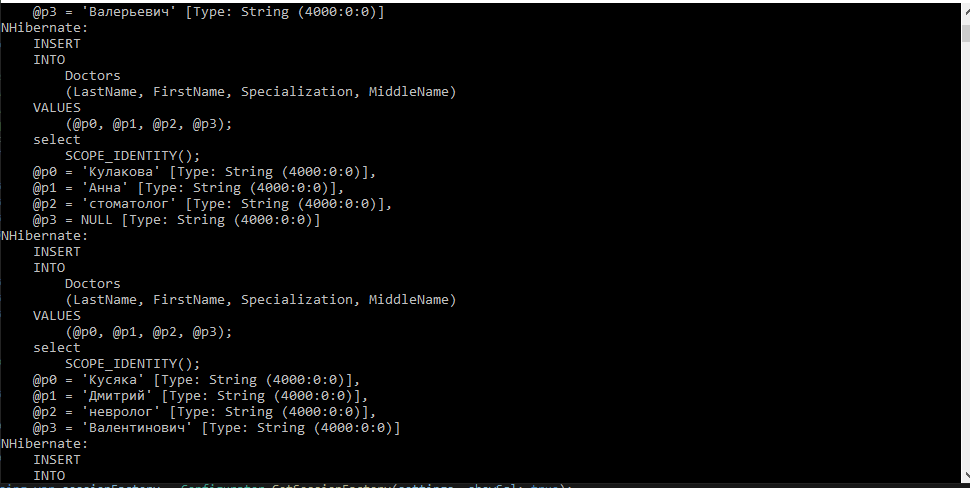
# Реализация программы

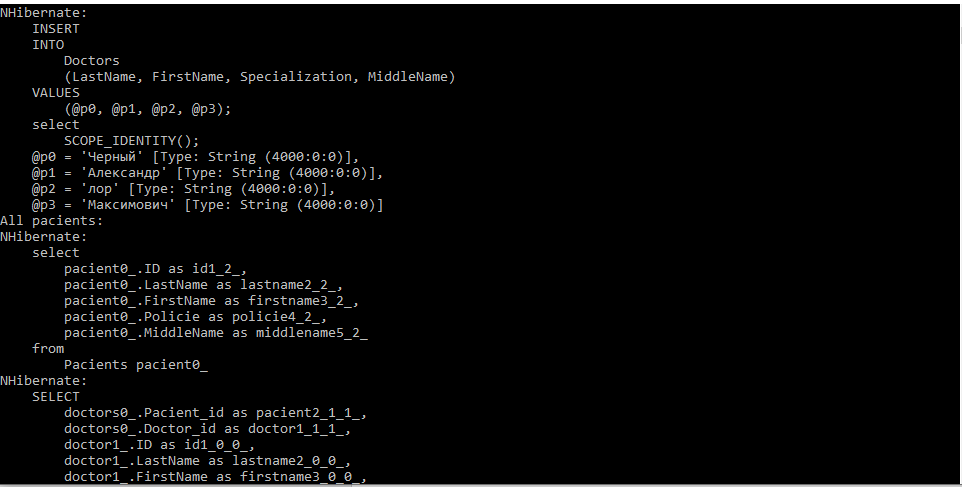


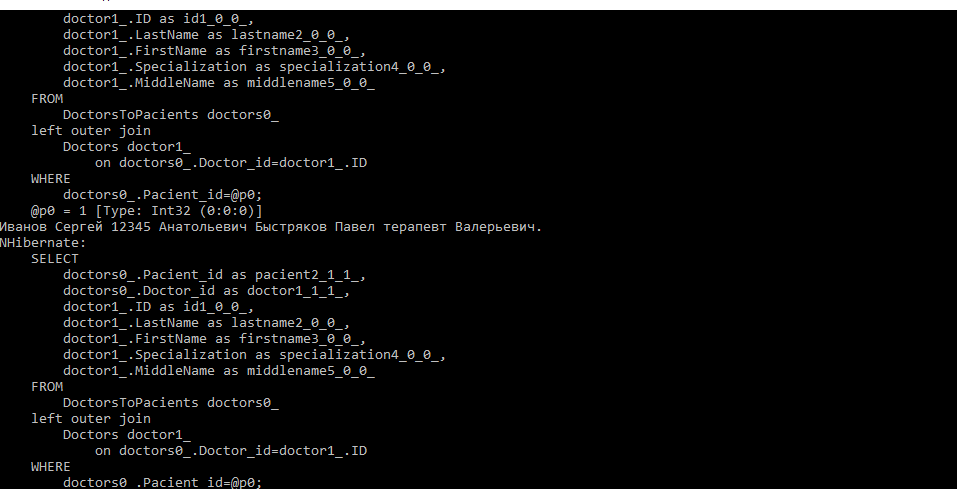


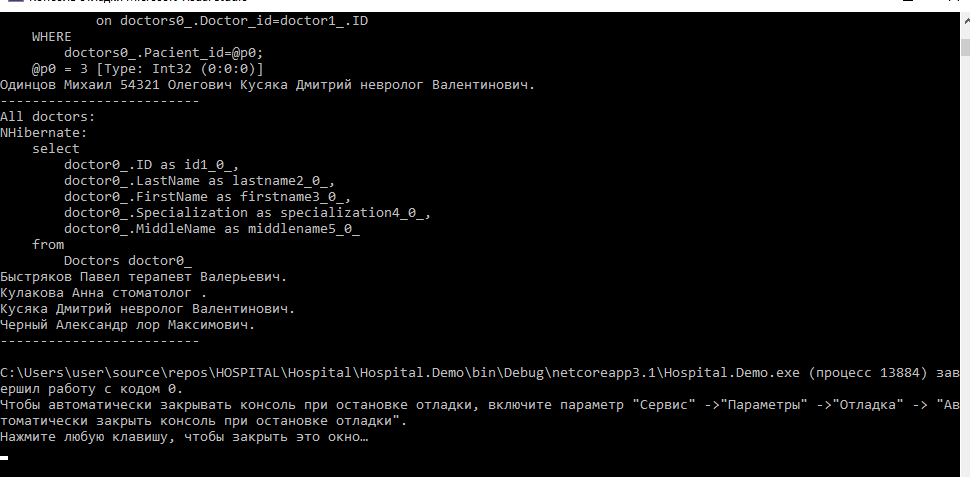












# Тесты

