**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»**

Выполнил: ст. гр. ТКИ–511  
Несвижская Е.А.  
Проверил: Васильева М.А.

Москва 2021

# ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ

Разработать БД на языке C# с помощью сторонних библиотек.

# ЗАДАНИЕ ДЛЯ ПРАКТИЧЕСКОЙ РАБОТЫ

В данной практической работе ставится задача написать БД на языке С# в MicrosoftVisualStudio по заданной теме «Больница». База данных создаётся для информационного обслуживания медицинских работников и посетителей больницы. БД должна содержать информацию о лечащихся у них пациентах и нахождения пациентов по палатам.

# ЛИСТИНГ ПРОГРАММЫ

Hospital.Data.Access:

1. ChamberMap

namespace Hospital.DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Hospital.Domain;

/// <summary>

/// Класс, описывающий правила отображения <see cref=»Chamber»/> на таблицу в БД и наоборот.

/// </summary>

internal class ChamberMap : ClassMap<Chamber>

{

/// <summary>

/// Initializes a new instance of the <see cref=”ChamberMap”/> class.

/// </summary>

public ChamberMap()

{

this.Table(“Chambers”);

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

this.Map(x => x.Capacity);

this.Map(x => x.Number);

this.HasMany(x => x.Patients)

.Cascade.Delete()

.Not.Inverse();

}

}

}

1. PatientMap

namespace Hospital.DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using Hospital.Domain;

/// <summary>

/// Класс, описывающий правила отображения <see cref="Patient"/> на таблицу в БД и наоборот.

/// </summary>

internal class PatientMap : ClassMap<Patient>

{

/// <summary>

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

/// </summary>

public PatientMap()

{

this.Table("Patients");

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

this.Map(x => x.BirthDate);

this.References(x => x.Chamber);

this.Map(x => x.Diagnosis);

this.Map(x => x.FullName);

this.Map(x => x.Policy);

}

}

}

1. IRepository

namespace Hospital.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);

bool Save(TEntity entity);

IQueryable<TEntity> GetAll(ISession session);

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

}

}

1. ChamberRepository

namespace Hospital.DataAccess.Repositories

{

using System;

using System.Linq;

using System.Linq.Expressions;

using Hospital.DataAccess.Repositories.Abstraction;

using Hospital.Domain;

using NHibernate;

public class ChamberRepository : IRepository<Chamber>

{

private ISession session;

public Chamber Get(ISession session, int id) => session?.Get<Chamber>(id);

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

{

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

}

public IQueryable<Chamber> GetAll(ISession session) => session?.Query<Chamber>();

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

{

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

}

public bool Save( Chamber entity)

{

try

{

this.session?.Save(entity);

this.session.Flush();

return true;

}

catch

{

return false;

}

}

}

}

1. Configurator

using System;

using System.Collections.Generic;

using System.Text;

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);

}

}

}

1. Settings

namespace Hospital.DataAccess

{

using System;

using System.Configuration;

public sealed class Settings

{

//private const string ServerNameKey = "databaseServer";

//private readonly Configuration configuration;

//public Settings(Configuration configuration)

//{

// this.configuration = configuration ?? throw new ArgumentNullException(nameof(configuration));

//}

private string databaseServerName;

private string databaseName;

public void AddDatabaseServer(string serverName)

{

this.databaseServerName = serverName;

//this.configuration.AppSettings.Settings.Add(ServerNameKey, serverName);

}

public string GetDatabaseServer()

{

return this.databaseServerName;

//return this.configuration.AppSettings.Settings[ServerNameKey].Value;

}

public void AddDatabaseName(string databaseName)

{

this.databaseName = databaseName;

}

public string GetDatabaseName()

{

return this.databaseName;

}

}

}

1. TestsConfigurator

namespace Hospital.DataAccess

{

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using NHibernate;

using NHibernate.Cfg;

using NHibernate.Tool.hbm2ddl;

/// <summary>

/// Класс для настройки соединения для тестов.

/// </summary>

public class TestsConfigurator

{

private static Configuration configuration;

/// <summary>

/// Создание фабрики сессий.

/// </summary>

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

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

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

public static ISessionFactory GetSessionFactory(Assembly assembly = null, bool showSql = false)

{

var databaseConfiguration = SQLiteConfiguration.Standard.InMemory();

if (showSql)

{

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

}

return Fluently.Configure()

.Database(databaseConfiguration)

.Mappings(m => m.FluentMappings

.AddFromAssembly(assembly ?? Assembly.GetExecutingAssembly()))

.ExposeConfiguration(c => configuration = c)

.BuildSessionFactory();

}

/// <summary>

/// Генерируется сессия для модульных тестов.

/// </summary>

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

/// <returns> Сессия подключения к тестовой БД. </returns>

public static ISession BuildSessionForTest(bool showSql = true)

{

var session = GetSessionFactory(showSql: showSql).OpenSession();

new SchemaExport(configuration)

.Execute(true, true, false, session.Connection, null);

return session;

}

}

}

Hospital.Data.Access.Tests:

1. BookRepositoryTests

namespace Hospital.DataAccess.Tests.Repositories

{

using Hospital.DataAccess.Repositories;

using Hospital.Domain;

using NHibernate;

using NUnit.Framework;

/// <summary>

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

/// </summary>

[TestFixture]

public class ChamberRepositoryTests

{

[Test]

public void Get\_ValidId\_Success()

{

// arrange

var targetId = 1;

using var session = GetSession();

PrepareChamberInStorage(session, targetId);

var repository = GetRepository();

// act

var result = repository.Get(session, targetId);

// assert

Assert.IsNotNull(result);

Assert.AreEqual(targetId, result.Id);

}

private static void PrepareChamberInStorage(ISession session, int targetId)

{

var chamber = new Chamber(0, 1, 4);

session.Save(chamber);

session.Flush();

session.Clear();

}

private static ChamberRepository GetRepository() => new ChamberRepository();

private static ISession GetSession() => TestsConfigurator.BuildSessionForTest(showSql: true);

}

}

1. BaseMapTests:

using NUnit.Framework;

/// <summary>

/// Базовый класс для тестирования маппингов.

/// </summary>

public class BaseMapTests

{

protected ISession Session { get; private set; }

[SetUp]

public void SetUp()

{

this.Session = TestsConfigurator.BuildSessionForTest();

}

[TearDown]

public void TearDown()

{

this.Session?.Dispose();

}

}

}

1. ChamberMapTests:

using FluentNHibernate.Testing;

using Hospital.Domain;

using NUnit.Framework;

/// <summary>

/// Тесты на правила отображения <see cref="Hospital.DataAccess.Mappings.ChamberMap"/>.

/// </summary>

[TestFixture]

internal class ChamberMapTests : BaseMapTests

{

[Test]

public void PersistenceSpecification\_ValidData\_Success()

{

// arrange

var chamber = new Chamber(1, 1, 1);

// act & assert

new PersistenceSpecification<Chamber>(this.Session)

.VerifyTheMappings(chamber);

}

}

}

Hospital.Demo:

Program

namespace Hospital.Demo

{

using System;

//// using System.Configuration;

using System.Linq;

using System.Collections.Generic;

using Hospital.DataAccess;

using Hospital.DataAccess.Repositories;

using Hospital.Domain;

/// <summary>

/// Точка входа в программу.

/// </summary>

internal class Program

{

private static void Main()

{

int count = 400;

var chambers = new List<Chamber>();

for (int i = 0; i < count; i++)

{

int capacity = 1;

if (i % 100 == 0) { capacity += 2; }

chambers.Add(new Chamber(i, (int)i + 1, capacity));

}

DateTime dateOfBirth = new DateTime(1995, 5, 13);

var patient = new Patient(1, chambers[201], "Иванов Иван Николаевич", dateOfBirth, "ангина", 5643);

Console.WriteLine($"{chambers[201]} {patient}");

//// var connectionString = ConfigurationManager.ConnectionStrings["Default"].ConnectionString;

var settings = new Settings();

settings.AddDatabaseServer(@"DESKTOP-OJ4THAK\SQLEXPRESS");

settings.AddDatabaseName("securedHospital");

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

using (var session = sessionFactory.OpenSession())

{

chambers.ForEach(ch => session.Save(ch));

session.Save(patient);

session.Flush();

}

using (var session = sessionFactory.OpenSession())

{

var repo = new ChamberRepository();

// TODO: Для наглядности нужно много палат с разным пациентским составом!

Console.WriteLine("Results through repo:");

repo.Filter(session, ch => ch.Capacity == 4)

.Distinct()

.ToList()

.ForEach(Console.WriteLine);

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

}

using (var session = sessionFactory.OpenSession())

{

var tmpChamber = session.Query<Chamber>().First();

Console.WriteLine(tmpChamber);

}

using (var session = sessionFactory.OpenSession())

{

using (var transaction = session.BeginTransaction())

{

session.Clear();

var persistentPatient = session.Load<Patient>(1);

var newChamber = persistentPatient.Chamber;

if (newChamber is null)

{

throw new ArgumentNullException(nameof(newChamber));

}

var persistentChamber = session.Get<Chamber>(newChamber.Id);

session.Delete(persistentChamber);

transaction.Commit();

}

session.Flush();

}

}

}

}

Hospital.Demo.Tests:

1. ChamberTests

namespace Hospital.Demo.Tests

{

using System;

using NUnit.Framework;

using Hospital.Domain;

[TestFixture]

public class ChamberTests

{

[Test]

public void ToString\_ValidData\_Success()

{

// arrange

var chamber = new Chamber(0, 1, 2);

var expected = "1 2";

// act

var actual = chamber.ToString();

// assert

Assert.AreEqual(expected, actual);

}

[Test]

public void Constructor\_NegativeID\_Fail()

{

// act & assert

Assert.Throws<ArgumentException>(() => \_ = new Chamber(-4, 1, 3));

}

[Test]

public void Constructor\_ValidData\_Success()

{

// act & assert

Assert.DoesNotThrow(() => \_ = new Chamber(1, 5, 6));

}

}

}

1. PatientTests

using System;

using System.Collections.Generic;

using System.Text;

namespace Hospital.Demo.Tests

{

using System;

using NUnit.Framework;

using Hospital.Domain;

[TestFixture]

public class PatientTests

{

[Test]

public void Constructor\_ValidData\_Success()

{

// arrange

var chamber = new Chamber(1, 2, 3);

// act & assert

Assert.DoesNotThrow(() => \_ = new Patient(1, chamber, "Аркадий Игоревич Носов", DateTime.Now, "бронхит", 4));

}

[Test]

public void Constructor\_NegativeID\_Fail()

{

// arrange

var chamber = new Chamber(1, 2, 3);

// act & assert

Assert.Throws<ArgumentException>(() => \_ = new Patient(-1, chamber, "Аркадий Игоревич Носов", DateTime.Now, "бронхит", 4));

}

[Test]

public void ToString\_ValidData()

{

// arrange

var chamber = new Chamber(1, 2, 3);

var patient = new Patient(1, chamber, "Аркадий Игоревич Носов", DateTime.Now, "бронхит", 4);

var expected = "Аркадий Игоревич Носов";

// act

var result = patient.ToString();

// assert

Assert.AreEqual(expected, result);

}

}

}

Hospital.Domain:

1. Chamber

namespace Hospital.Domain

{

using System;

using System.Collections.Generic;

using System.Linq;

// using Hospital.Staff.Extensions;

/// <summary>

/// Палата.

/// </summary>

public class Chamber

{

/// <summary>

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

/// </summary>

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

/// <param name="number"> Номер палаты. </param>

/// <param name="capacity"> Вместимость палаты. </param>

public Chamber(int id, int number, int capacity)

{

if (id < 0)

{

throw new ArgumentException("ID cannot be negative.");

}

this.Id = id;

this.Number = number;

this.Capacity = capacity;

}

/// <summary>

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

/// </summary>

[Obsolete("For ORM", true)]

protected Chamber()

{

}

/// <summary>

/// Идентификатор.

/// </summary>

public virtual int Id { get; protected set; }

/// <summary>

/// Номер палаты.

/// </summary>

public virtual int Number { get; protected set; }

/// <summary>

/// Вместимость.

/// </summary>

public virtual int Capacity { get; protected set; }

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

public virtual bool AddPatient(Patient patient)

{

if (patient == null) throw new ArgumentNullException(nameof(Patient));

patient.Chamber = this;

return this.Patients.Add(patient);

}

/// <inheritdoc/>

public override string ToString() => $"{this.Number} {this.Capacity}".Trim();

}

}

1. Patient

namespace Hospital.Domain

{

using System;

using Hospital.Staff.Extensions;

/// <summary>

/// Пациент.

/// </summary>

public class Patient : IEquatable<Patient>

{

/// <summary>

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

/// </summary>

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

/// <param name="chamber"> Палата. </param>

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

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

/// <param name="diagnosis"> Диагноз. </param>

/// <param name="policy"> Номер полиса. </param>

/// <exception cref="ArgumentOutOfRangeException">

/// В случае если <paramref name="fullName"/> <see langword="null"/>, пустая строка

/// или строка, содержащая только пробельные символы.

/// </exception>

public Patient(int id, Chamber chamber, string fullName, DateTime birthDate, string diagnosis, int policy)

{

if (id < 0)

{

throw new ArgumentException("ID cannot be negative.");

}

this.Id = id;

this.Chamber = chamber;

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

this.BirthDate = birthDate;

this.Diagnosis = diagnosis;

this.Policy = policy;

}

/// <summary>

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

/// </summary>

[Obsolete("For ORM", true)]

protected Patient()

{

}

/// <summary>

/// Уникальный идентификатор.

/// </summary>

public virtual int Id { get; protected set; }

/// <summary>

/// Палата.

/// </summary>

public virtual Chamber Chamber { get; set; }

/// <summary>

/// ФИО.

/// </summary>

public virtual string FullName { get; protected set; }

/// <summary>

/// Дата рождения.

/// </summary>

public virtual DateTime BirthDate { get; protected set; }

/// <summary>

/// Диагноз.

/// </summary>

public virtual string Diagnosis { get; protected set; }

/// <summary>

/// Номер полиса.

/// </summary>

public virtual int Policy { get; protected set; }

/// <inheritdoc/>

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

/// <inheritdoc/>

public override bool Equals(object obj)

{

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

}

/// <inheritdoc cref="IEquatable{T}"/>

public virtual bool Equals(Patient other)

{

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

}

/// <inheritdoc/>

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

}

}

Hospital.Staff:

1. IEnumerableExtension

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);

}

}

1. StringExtensions

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();

return trimmedValue.IsNullOrEmpty()

? null

: trimmedValue;

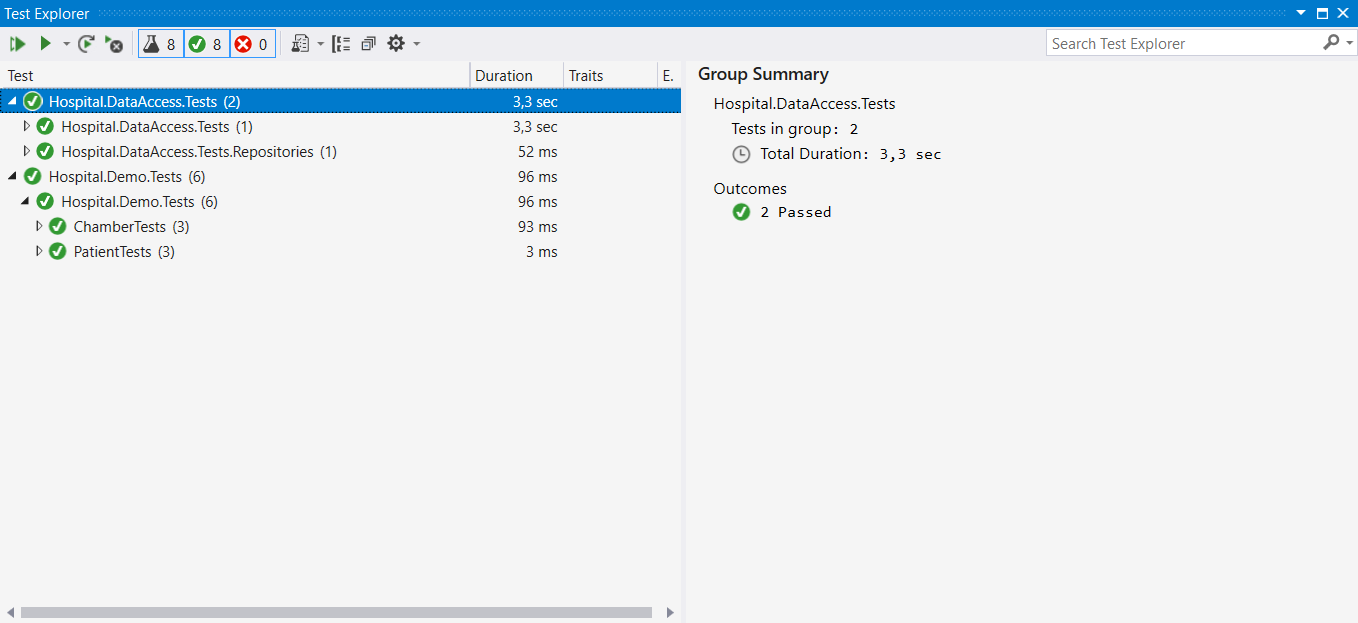
}

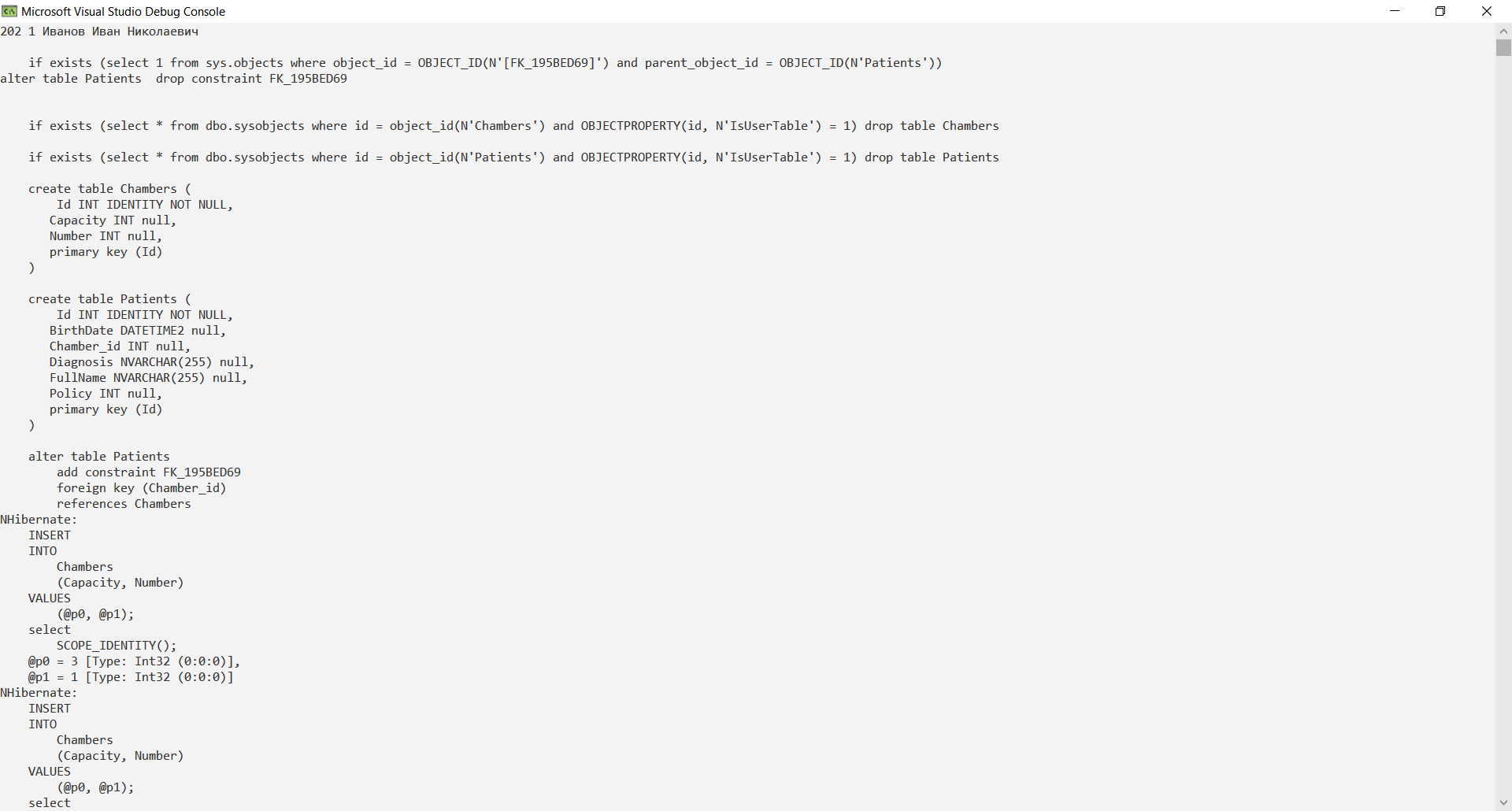
}

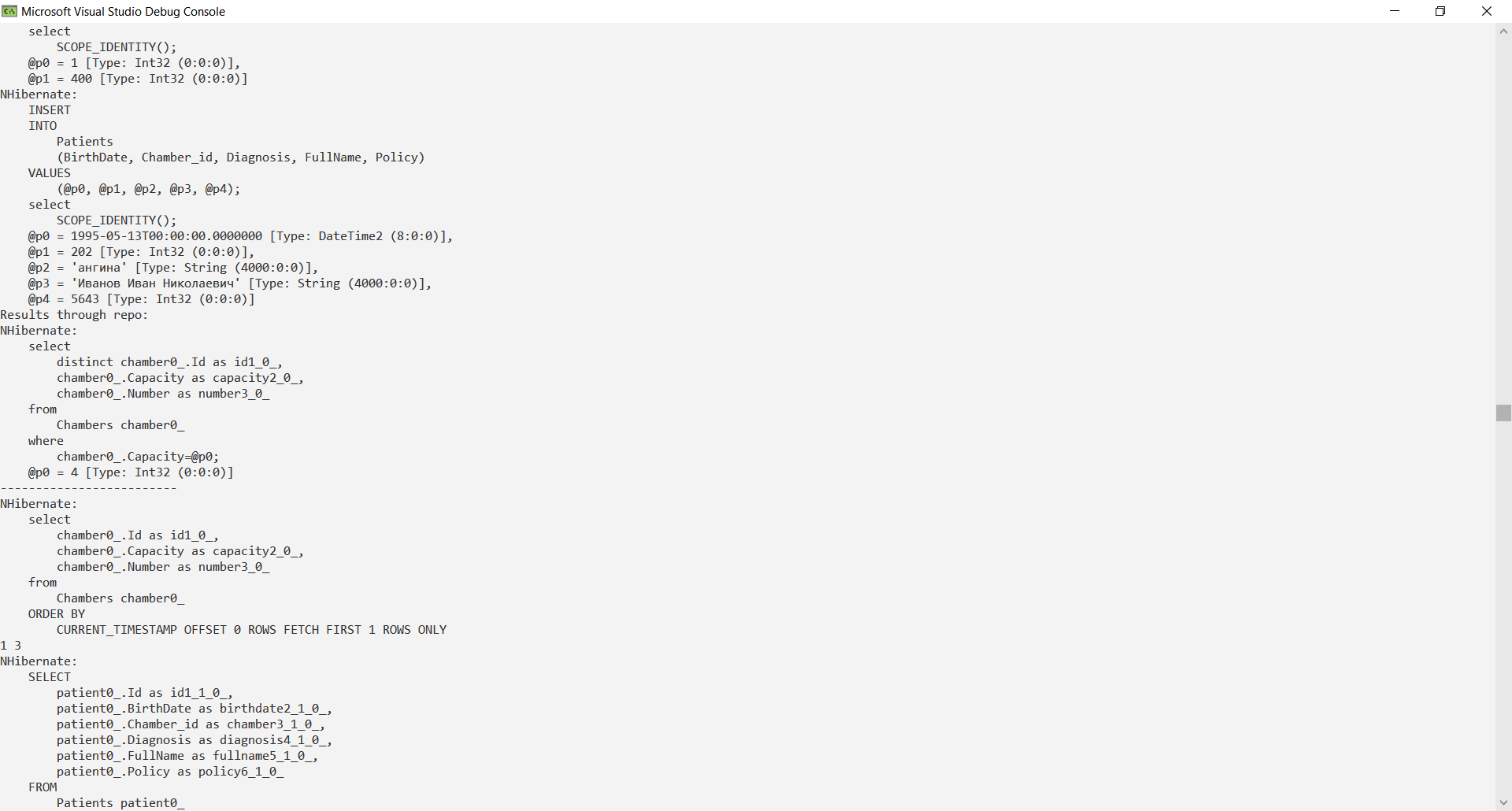
}

# ВЫПОЛНЕНИЕ ПРОГРАММЫ

Все тесты пройдены успешно







# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы мы разработали программу на языке C#.