**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 «Кондитерская фабрика»»**

Выполнила: ст. гр. ТКИ–511  
Хобта Д.О.  
Проверил: Васильева М.А.

Москва 2021

# Задание

Разработать ORM на языке C# для хранения экземпляров классов базы данных кондитерской фабрики. Фабрика изготавливает кондитерские изделия. Для изготовления товара требуются продукты. Реализовать принцип CRUD.

# Описание

Было создано одно общее решение ConfectioneryFactory, в составе которого имеются проекты. Проект Demo, содержащий главный метод, через который запускается решение (является консольным приложением). Проект Domain необходим для реализации классов сущностей реализуемой базы данных. Выбраны сущности *ингредиент* (ingredient) и *продукт* (product). DataAccess проект, содержащий классы маппинга, репозитория и фабрики сессий. В Staff представлены классы для определения и сокращения пустых строк. В завершении были добавлены модульные тесты Demo.Tests на сущностный класс.

# Код программы

* 1. ConfectioneryFactory.Demo.Program:

namespace ConfectioneryFactory.Demo

{

using System;

using System.Linq;

using ConfectioneryFactory.DataAccess;

using ConfectioneryFactory.DataAccess.Repositories;

using ConfectioneryFactory.Domain;

/// <summary>

/// Точка входа в программу.

/// </summary>

internal class Program

{

static void Main(string[] args)

{

var ingredient1 = new Ingredient(1, "Мука", 70);

var ingredient2 = new Ingredient(2, "Сахар", 40);

var ingredient3 = new Ingredient(3, "Молоко", 100);

var ingredient4 = new Ingredient(4, "Масло", 200);

var product1 = new Product(1, "Сладкоежка", "Пончик", 100, ingredient1, ingredient2, ingredient4);

var product2 = new Product(2, "Зима", "Эклер", 150, ingredient1, ingredient2, ingredient3);

var product3 = new Product(3, "Карамель", "Леденец", 150);

Console.WriteLine($"{product1}");

Console.WriteLine($"{product2}");

Console.WriteLine($"{product3}");

var settings = new Settings();

settings.AddDatabaseServer(@"DESKTOP-REIQ8I1\SQLEXPRESS");

settings.AddDatabaseName("SecureConFactory");

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

using (var session = sessionFactory.OpenSession())

{

session.Save(ingredient1);

session.Save(ingredient2);

session.Save(ingredient3);

session.Save(ingredient4);

session.Save(product1);

session.Save(product2);

session.Flush();

}

using (var session = sessionFactory.OpenSession())

{

var repoProduct = new ProductRepository();

var repoIngredient = new IngredientRepository();

Console.WriteLine("All ingredients");

repoIngredient.GetAll(session)

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

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

}

}

}

}

* 1. ConfectioneryFactory.Domain.Ingredient

namespace ConfectioneryFactory.Domain

{

using System;

using System.Collections.Generic;

using ConfectioneryFactory.Staff.Extensions;

public class Ingredient : IEquatable<Ingredient>

{

public Ingredient(int id, string name, int price)

{

this.Id = id;

var trimmName = name.TrimOrNull();

this.Name = trimmName ?? throw new ArgumentOutOfRangeException(nameof(name));

this.Price = price;

}

[Obsolete("For ORM", true)]

protected Ingredient()

{

}

public virtual int Id { get; protected set; }

public virtual string Name { get; protected set; }

public virtual int Price { get; protected set; }

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

public virtual bool AddProduct(Product product)

{

return product == null ? throw new ArgumentNullException(nameof(product)) : this.Products.Add(product);

}

public override string ToString() => $"{Id}. {Name} - {Price}р";

public override bool Equals(object obj)

{

if (ReferenceEquals(null, obj)) return false;

if (ReferenceEquals(this, obj)) return true;

return Equals((Ingredient)obj);

}

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

public virtual bool Equals(Ingredient other)

{

if (ReferenceEquals(null, other)) return false;

if (ReferenceEquals(this, other)) return true;

return this.Id == other.Id;

}

}

}

* 1. ConfectioneryFactory.Domain.Product

namespace ConfectioneryFactory.Domain

{

using System;

using System.Collections.Generic;

using System.Linq;

using ConfectioneryFactory.Staff.Extensions;

public class Product

{

public Product(int id, string name, string type, int price, params Ingredient[] ingredients)

: this(id, name, type, price, new HashSet<Ingredient>(ingredients))

{

}

public Product(int id, string name, string type, int price, ISet<Ingredient> ingredients = null)

{

this.Id = id;

var trimmName = name.TrimOrNull();

var trimmType = type.TrimOrNull();

this.Name = trimmName ?? throw new ArgumentOutOfRangeException(nameof(name));

this.Type = trimmType ?? throw new ArgumentOutOfRangeException(nameof(type));

this.Price = price;

foreach (var ingredient in ingredients ?? Enumerable.Empty<Ingredient>())

this.Ingredients.Add(ingredient);

}

[Obsolete("For ORM", true)]

protected Product()

{

}

public virtual int Id { get; protected set; }

public virtual string Name { get; protected set; }

public virtual string Type { get; protected set; }

public virtual int Price { get; protected set; }

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

public override string ToString()

{

string result = $"{Id}) {Type} <<{Name}>> - {Price}р";

if (Ingredients.Count != 0)

{

result += "(Ингредиенты:";

foreach (var ingredient in Ingredients)

result += $" {ingredient}";

result += ")";

}

return result;

}

}

}

* 1. ConfectioneryFactory.Staff.StringExtensions

namespace ConfectioneryFactory.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;

}

}

}

* 1. ConfectioneryFactory.DataAccess.Mappings

.IngredientMap

namespace ConfectioneryFactory.DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using ConfectioneryFactory.Domain;

/// <summary>

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

/// </summary>

class IngredientMap : ClassMap<Ingredient>

{

public IngredientMap()

{

this.Table("Ingredients");

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

this.Map(x => x.Name)

.Not.Nullable();

this.Map(x => x.Price)

.Not.Nullable();

this.HasManyToMany(x => x.Products)

.Cascade.Delete();

}

}

}

* 1. ConfectioneryFactory.DataAccess.Mappings

.ProductMap

namespace ConfectioneryFactory.DataAccess.Mappings

{

using FluentNHibernate.Mapping;

using ConfectioneryFactory.Domain;

/// <summary>

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

/// </summary>

internal class ProductMap : ClassMap<Product>

{

/// <summary>

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

/// </summary>

public ProductMap()

{

this.Table("Products");

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

this.Map(x => x.Name)

.Not.Nullable();

this.Map(x => x.Type)

.Not.Nullable();

this.Map(x => x.Price)

.Not.Nullable();

this.HasManyToMany(x => x.Ingredients)

.Cascade.Delete()

.Inverse();

}

}

}

* 1. ConfectioneryFactory.DataAccess.Repositories

.Abstraction.IRepository

namespace ConfectioneryFactory.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);

bool Save(TEntity entity, ISession session);

}

}

* 1. ConfectioneryFactory.DataAccess.Repositories

.IngredientRepository

namespace ConfectioneryFactory.DataAccess.Repositories

{

using System;

using System.Collections.Generic;

using System.Linq;

using ConfectioneryFactory.DataAccess.Repositories.Abstraction;

using ConfectioneryFactory.Domain;

using NHibernate;

public class IngredientRepository : IRepository<Ingredient>

{

public bool Create<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

public bool Delete<TEntity>(TEntity entity)

{

throw new NotImplementedException();

}

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

{

return this.GetAll(session)

.Where(predicate);

}

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

{

return this.GetAll(session)

.FirstOrDefault(predicate);

}

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

session?.Get<Ingredient>(id);

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

session?.Query<Ingredient>();

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

{

throw new NotImplementedException();

}

public bool Save(Ingredient entity, ISession session)

{

try

{

session?.Save(entity);

session?.Flush();

return true;

}

catch

{

return false;

}

}

}

}

* 1. ConfectioneryFactory.DataAccess.Repositories

.ProductRepository

namespace ConfectioneryFactory.DataAccess.Repositories

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using ConfectioneryFactory.DataAccess.Repositories.Abstraction;

using ConfectioneryFactory.Domain;

using NHibernate;

public class ProductRepository : IRepository<Product>

{

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

session?.Get<Product>(id);

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

{

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

}

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

session?.Query<Product>();

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

}

public IQueryable<Ingredient> IngredientsWithoutCollaborates(ISession session)

{

return this.Filter(

session,

b => b.Ingredients.Count < 2)

.SelectMany(b => b.Ingredients)

.Distinct();

}

public bool Save(Product entity, ISession session)

{

try

{

session?.Save(entity);

session?.Flush();

return true;

}

catch

{

return false;

}

}

}

}

* 1. ConfectioneryFactory.DataAccess.Configurator

namespace ConfectioneryFactory.DataAccess

{

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using NHibernate;

using NHibernate.Cfg;

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(Configuration configuration)

{

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

}

}

}

* 1. ConfectioneryFactory.DataAccess.Settings

using System;

namespace ConfectioneryFactory.DataAccess

{

public 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;

}

}

}

* 1. ConfectioneryFactory.DataAccess.TestConfigurator

namespace ConfectioneryFactory.DataAccess

{

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using NHibernate;

using NHibernate.Cfg;

using NHibernate.Tool.hbm2ddl;

public class TestConfigurator

{

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;

}

}

}

* 1. ConfectioneryFactory.DataAccess.Tests

.Repositories.ProductRepositoryTests

namespace ConfectioneryFactory.DataAccess.Tests.Repositories

{

using ConfectioneryFactory.DataAccess.Repositories;

using ConfectioneryFactory.Domain;

using NHibernate;

using NUnit.Framework;

[TestFixture]

public class ProductRepositoryTests

{

[Test]

public void Get\_ValidId\_Success()

{

var targetId = 1;

using var session = GetSession();

PrepareProductInStorage(session, targetId);

var repository = GetRepository();

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

Assert.IsNotNull(result);

Assert.AreEqual(targetId, result.Id);

}

private static void PrepareProductInStorage(ISession session, int targetId)

{

var product = new Product(1, "Тестовое название", "Тестовый тип", 100);

session.Save(product);

session.Flush();

session.Clear();

}

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

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

}

}

* 1. ConfectioneryFactory.DataAccess.Tests

.BaseMapTests

namespace ConfectioneryFactory.DataAccess.Tests

{

using NHibernate;

using NUnit.Framework;

public class BaseMapTests

{

protected ISession Session { get; private set; }

[SetUp]

public void Setup()

{

this.Session = TestConfigurator.BuildSessionForTest();

}

[TearDown]

public void TearDown()

{

this.Session?.Dispose();

}

}

}

* 1. ConfectioneryFactory.DataAccess.Tests

.ProductMapTests

namespace ConfectioneryFactory.DataAccess.Tests

{

using FluentNHibernate.Testing;

using ConfectioneryFactory.Domain;

using NUnit.Framework;

[TestFixture]

internal class ProductMapTest : BaseMapTests

{

[Test]

public void PersistenceSpecification\_ValidData\_Success()

{

var product = new Product(1, "Тестовое имя", "Тестовый тип", 100);

new PersistenceSpecification<Product>(this.Session)

.VerifyTheMappings(product);

}

}

}

* 1. ConfectioneryFactory.Demo.Tests.ProductTests

namespace ConfectioneryFactory.Demo.Tests

{

using System;

using NUnit.Framework;

using ConfectioneryFactory.Domain;

[TestFixture]

public class ProductTests

{

[Test]

public void ToString\_ValidData\_Success()

{

var ingredient = new Ingredient(1, "Сахар", 40);

var product = new Product(1, "Сладкоежка", "Пончик", 100, ingredient);

var expected = $"1) Пончик <<Сладкоежка>> - 100р(Ингредиенты: 1. Сахар - 40р)";

var actual = product.ToString();

Assert.AreEqual(expected, actual);

}

[Test]

public void ToString\_EmptyIngredient\_Success()

{

var book = new Product(1, "Карамель", "Леденец", 40);

var expected = "1) Леденец <<Карамель>> - 40р";

var actual = book.ToString();

Assert.AreEqual(expected, actual);

}

[Test]

public void Ctor\_ValidDataEmptyIngredients\_Success()

{

Assert.DoesNotThrow(() => \_ = new Product(1, "Карамель", "Леденец", 40));

}

[Test]

[TestCase(null)]

[TestCase("")]

[TestCase(" ")]

[TestCase("\0")]

[TestCase("\n")]

[TestCase("\r")]

[TestCase("\t")]

public void Ctor\_WrongDataNullTypeEmptyIngredients\_Fail(string wrongType)

{

Assert.Throws<ArgumentOutOfRangeException>(() => \_ = new Product(1, "Карамель", wrongType, 40));

}

[Test]

public void Ctor\_ValidData\_Success()

{

var ingredient = new Ingredient(1, "Сахар", 40);

Assert.DoesNotThrow(() => \_ = new Product(1, "Сладкоежка", "Пончик", 100, ingredient));

}

}

}

# Результаты работы программы

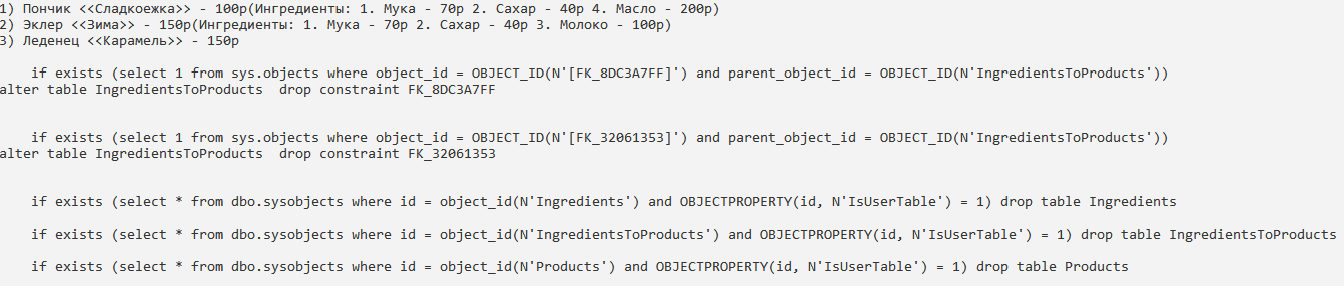


Рисунок 1

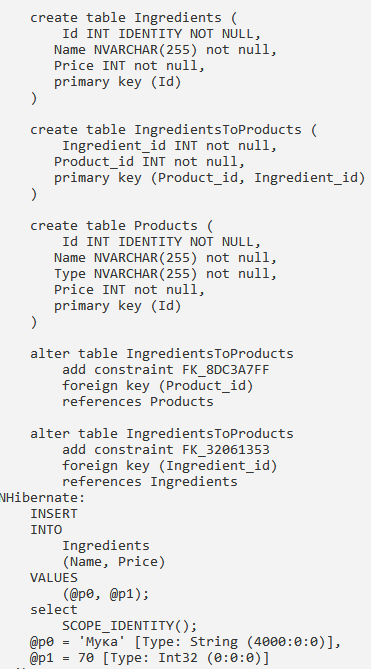


Рисунок 2

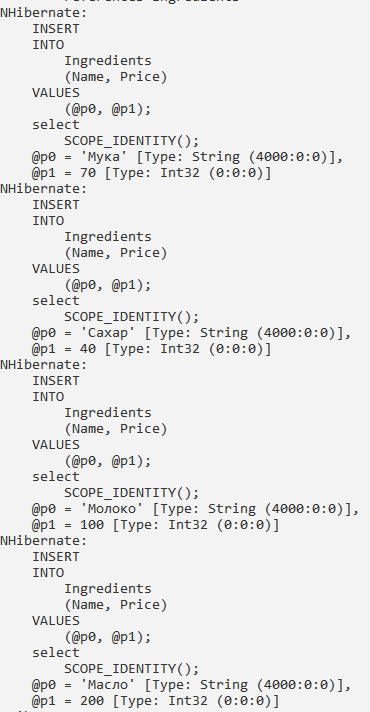


Рисунок 3

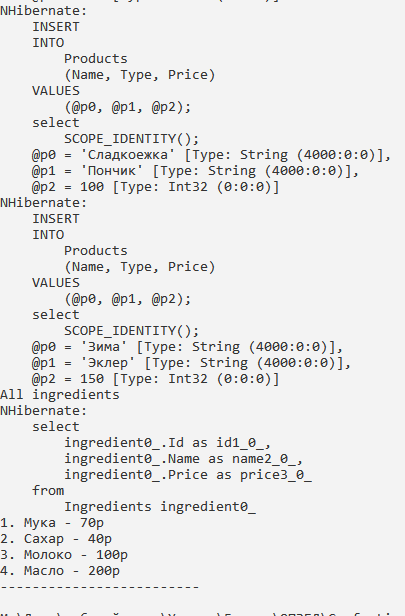


Рисунок 4

# Результаты прохождения тестов

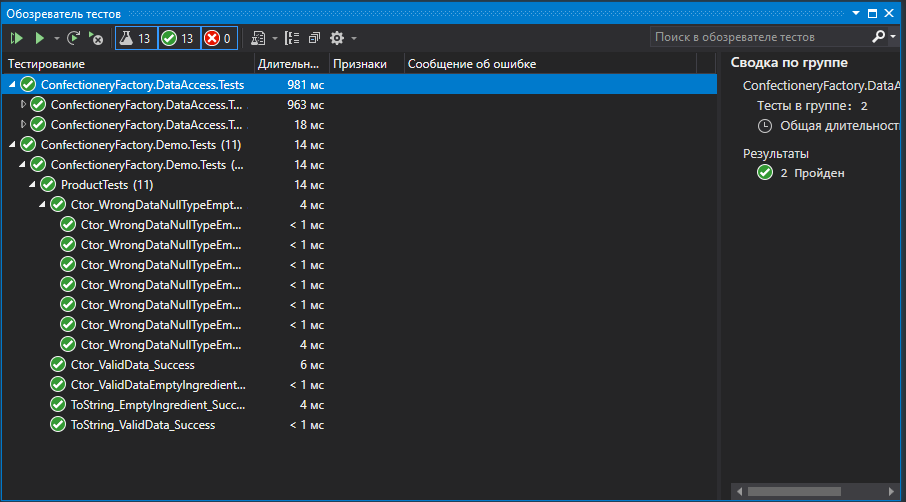


Рисунок 5

# ERD-диаграмма

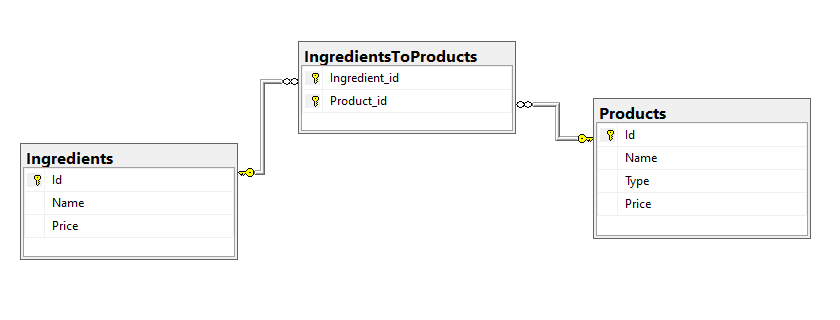


Рисунок 6

# Вывод

В данной работе было разработано ORM на языке C# для хранения экземпляров классов базы данных кондитерской фабрики, а также был реализован принцип CRUD.