# ЛАБОРАТОРНАЯ РАБОТА №3.
# РАБОТА С БД

## СОДЕРЖАНИЕ


Цель ..... 2

Задание ..... 2

Проектирование ..... 3

Реализация ..... 3

Контрольный пример ..... 15

Требования ..... 15

Порядок сдачи базовой части ..... 16

Контрольные вопросы к базовой части ..... 16

Усложненная лабораторная (необязательно) ..... 16

Порядок сдачи усложненной части ..... 17

Контрольные вопросы к усложненной части ..... 17

Варианты ..... 17

**Цель**

Изучить работу с базами данных. Подключаться к СУБД, извлекать, сохранять, удалять данные с применением ORM.

**Задание**

1. Создать ветку от ветки второй лабораторной.
2. *Требуется сделать приложение по учету изготовления изделий на заказ. Необходимо реализовать следующие требования:*
    a. *Ввод списка компонент, используемых при изготовлении изделий. Каждый компонент имеет уникальное имя, отличающее его от других компонент и цену за единицу компонента.*
    б. *Ввод списка производимых изделий. Для каждого изделия предусмотреть возможность указания компонент, из которых оно изготавливается и в каком количестве каждый компонент требуется при изготовлении изделия. Каждое изделие имеет стоимость, а также уникальное имя, отличающее его от других изделий. Стоимость изделия рассчитывается из суммы компонент, которые используются при его создании, и добавления определенного процента к этой сумме.*
    в. *Необходимо фиксировать дату создания заказа и дату выполнения.*
    г. *Заказ должен проходить ряд стадий: создание, изготовление, готов к выдачи, выдача заказа.*

    *Приложение должно быть оформлено в виде desktop-приложения. Требуется реализовать способ хранения данных в базе данных (использование СУБД для хранения данных и учет связанности данных при операциях удаления).*

3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

**Проектирование**

Для реализации хранения данных в файлах создадим новый проект **AbstractShopDatabaseImplement**. Этот проект будет относится к слою хранения данных и иметь те же зависимости, что и **AbstractShopListImplement** и **AbstractShopFileImplement**. По структуре также будет идентичен, 3 модели, 3 реализации и класс для работы с EntityFramework.

**Реализация**

Для работы с БД будем использовать Entity Framework Core. Подключаем его через NuGet пакеты. Для этого добавим в проект следующие пакеты (рисунок 3.1):

- Microsoft.EntityFrameworkCore;
- Microsoft.EntityFrameworkCore.SqlServer (данный пакет только для подключения в СУБД MS SQL, для подключения к другой СУБД потребуется ставить иные пакеты);
- Microsoft.EntityFrameworkCore.Tools;

А в проект **AbstractShopView** добавляем пакет Microsoft.EntityFrameworkCore.Design.

Добавляем модели. Так как на основе этих моделей будут созданы таблицы в базе данных, и они должны быть связанные, то потребуется настроить связи между таблицами в классах-моделях. Также, таблицы в БД не могут хранить списки, массивы и т.п., так что придется отказаться от словаря в модели «Изделие». Его придется заменить на отдельный класс-связь «Компонента» и «Изделия».

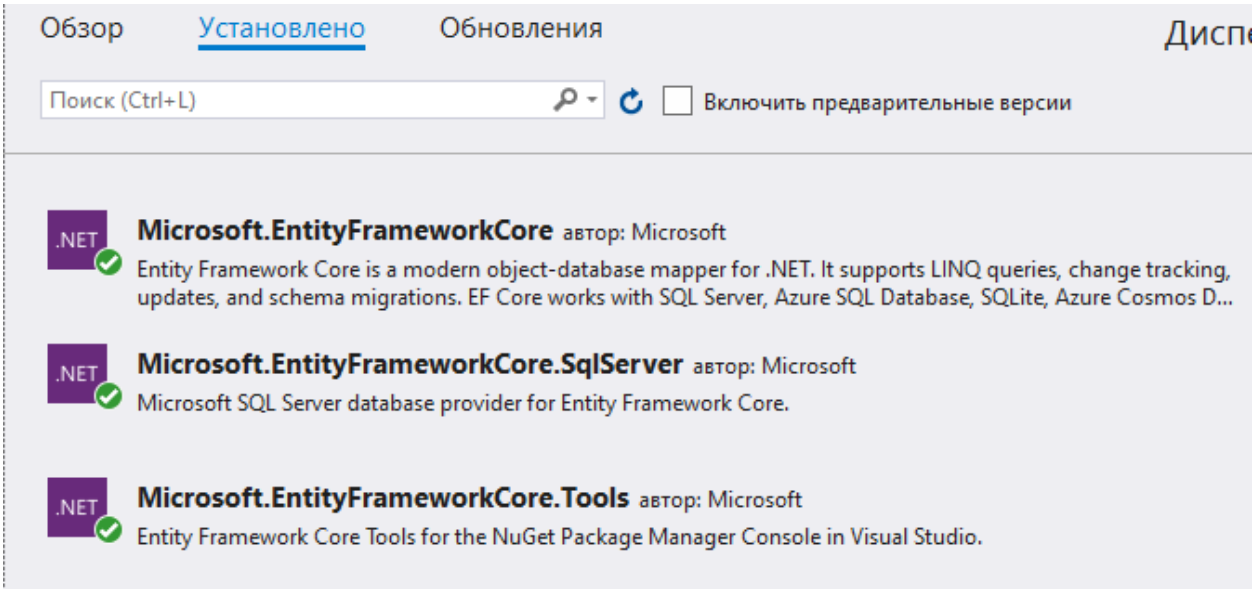


Рисунок 3.1 – Добавление EntityFrameworkCore

Сперва создадим все классы-модели на основе классов-моделей из проекта **AbstractShopListImplement**. Далее будем настраивать связи и ограничения.

Для класса-модели Компонента укажем, что название и цена обязательны к заполнению и что этот класс (таблица) будет связан с классом связи компонента и изделия типом связи один-ко-многим (по сути, у нас будет связь компонента и изделия по типу многие-ко-многим через отдельную сущность) (листинг 3.1).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Models;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace AbstractShopDatabaseImplement.Models
{
    public class Component : IComponentModel
    {
        public int Id { get; private set; }

        [Required]
        public string ComponentName { get; private set; } = string.Empty;

        [Required]
        public double Cost { get; set; }

        [ForeignKey("ComponentId")]
        public virtual List<ProductComponent> ProductComponents { get; set; } =
new();

        public static Component? Create(ComponentBindingModel model)
        {
            if (model == null)
            {
```

```
                return null;
            }
            return new Component()
            {
                Id = model.Id,
                ComponentName = model.ComponentName,
                Cost = model.Cost
            };
        }

        public static Component Create(ComponentViewModel model)
        {
            return new Component
            {
                Id = model.Id,
                ComponentName = model.ComponentName,
                Cost = model.Cost
            };
        }

        public void Update(ComponentBindingModel model)
        {
            if (model == null)
            {
                return;
            }
            ComponentName = model.ComponentName;
            Cost = model.Cost;
        }

        public ComponentViewModel GetViewModel => new()
        {
            Id = Id,
            ComponentName = ComponentName,
            Cost = Cost
        };
    }
}
```

Листинг 3.1 – Класс Component

Атрибут Required указывает на то, что поле обязательно к заполнению. Также указываем в качестве списка, что у нас есть связь один ко многим к таблице связи с изделиями. У списка указываем атрибут с внешним ключом (поле в таблице связи с изделиями). У класса-связи в таком случае, должно быть поле с таким названием (листинг 3.2).

```
using System.ComponentModel.DataAnnotations;

namespace AbstractShopDatabaseImplement.Models
{
    public class ProductComponent
    {
        public int Id { get; set; }

        [Required]
        public int ProductId { get; set; }

        [Required]
        public int ComponentId { get; set; }
```

```
        [Required]
        public int Count { get; set; }

        public virtual Component Component { get; set; } = new();

        public virtual Product Product { get; set; } = new();
    }
}
```

Листинг 3.2 – Класс ProductComponent

У класса-связи прописываем новое поле-ссылку на объект класса «Компонент». Таким образом, при создании таблиц в базе данных пропишется связь между этими таблицами.

У класса-модели «Изделие» будет 2 связи: с классом-связи (компонент-изделие) и классом-моделью заказа (один-ко-многим). У класса обязательно должны заполняться поля «Название» и «Цена». Как и для класс-модели «Изделия» в проекте **AbstractShopFileImplement** вводим отдельно свойство Components, только теперь оно будет в виде списка записей класса-связи (как раз для создания связи). Свойство ProductComponents будет заполняться, при обращении к GetViewModel. Также потребовалось отдельно прописать метод для обновления списка связей, так как теперь требуется выполнять несколько манипуляций: удаления неактуальных связей, обновление существующих и добавление новых. В начале метода придется сначала вытащить список всех связей по изделию и только потом фильтровать их по включению в список моделей, пришедших от пользователя. Это связано с тем, что метод ContainsKey словаря не сработает при запросе к БД (листинг 3.3).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Models;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace AbstractShopDatabaseImplement.Models
{
    public class Product : IProductModel
    {
        public int Id { get; set; }

        [Required]
        public string ProductName { get; set; } = string.Empty;

        [Required]
        public double Price { get; set; }

        private Dictionary<int, (IComponentModel, int)>? _productComponents =
null;
```

```
        [NotMapped]
        public Dictionary<int, (IComponentModel, int)> ProductComponents
        {
            get
            {
                if (_productComponents == null)
                {
                    _productComponents = Components
                            .ToDictionary(recPC => recPC.ComponentId, recPC =>
(recPC.Component as IComponentModel, recPC.Count));
                }
                return _productComponents;
            }
        }

        [ForeignKey("ProductId")]
        public virtual List<ProductComponent> Components { get; set; } = new();

        [ForeignKey("ProductId")]
        public virtual List<Order> Orders { get; set; } = new();

        public static Product Create(AbstractShopDatabase context,
ProductBindingModel model)
        {
            return new Product()
            {
                Id = model.Id,
                ProductName = model.ProductName,
                Price = model.Price,
                Components = model.ProductComponents.Select(x => new
ProductComponent
                {
                    Component = context.Components.First(y => y.Id == x.Key),
                    Count = x.Value.Item2
                }).ToList()
            };
        }

        public void Update(ProductBindingModel model)
        {
            ProductName = model.ProductName;
            Price = model.Price;
        }

        public ProductViewModel GetViewModel => new()
        {
            Id = Id,
            ProductName = ProductName,
            Price = Price,
            ProductComponents = ProductComponents
        };

        public void UpdateComponents(AbstractShopDatabase context,
ProductBindingModel model)
        {
            var productComponents = context.ProductComponents.Where(rec =>
rec.ProductId == model.Id).ToList();
            if (productComponents != null && productComponents.Count > 0)
            {   // удалили те, которых нет в модели
                context.ProductComponents.RemoveRange(productComponents.Where(rec
=> !model.ProductComponents.ContainsKey(rec.ComponentId)));
                context.SaveChanges();
                // обновили количество у существующих записей
                foreach (var updateComponent in productComponents)
```

```
                {
                    updateComponent.Count =
model.ProductComponents[updateComponent.ComponentId].Item2;
                    model.ProductComponents.Remove(updateComponent.ComponentId);
                }
                context.SaveChanges();
            }
            var product = context.Products.First(x => x.Id == Id);
            foreach (var pc in model.ProductComponents)
            {
                context.ProductComponents.Add(new ProductComponent
                {
                    Product = product,
                    Component = context.Components.First(x => x.Id == pc.Key),
                    Count = pc.Value.Item2
                });
                context.SaveChanges();
            }
            _productComponents = null;
        }
    }
}
```

Листинг 3.3 – Класс Product

Класс-модель «Заказ» должен быть связан с «Изделием». Обязательным к заполнению у него буду поля «Количество», «Сумма», «Статус» и «Дата создания». Прописать класс самостоятельно. Также для класса-модели «Изделие» прописать связь с классом-моделью «Заказ».

Теперь создадим класс, наследник от класса DbContext, в котором пропишем наборы от наших классов (на основе этого и создается база данных с таблицами) (листинг 3.4).

```
using AbstractShopDatabaseImplement.Models;
using Microsoft.EntityFrameworkCore;

namespace AbstractShopDatabaseImplement
{
    public class AbstractShopDatabase : DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            if (optionsBuilder.IsConfigured == false)
            {
                optionsBuilder.UseSqlServer(@"Data
Source=CHESHIR\SQLEXPRESS;Initial Catalog=AbstractShopDatabaseFull;Integrated
Security=True;MultipleActiveResultSets=True;;TrustServerCertificate=True");
            }
            base.OnConfiguring(optionsBuilder);
        }

        public virtual DbSet<Component> Components { set; get; }

        public virtual DbSet<Product> Products { set; get; }

        public virtual DbSet<ProductComponent> ProductComponents { set; get; }
```

```
        public virtual DbSet<Order> Orders { set; get; }
    }
}
```

Листинг 3.4 – Класс AbstractShopDatabase

В классе перегрузим метод OnConfiguring, где пропишем строку подключения к базе данных. Строка подключения состоит из нескольких настроек. Нас интересует 2 из них:

- Data Source – имя хоста (компьютера) и название СУБД, к которой хотим подключиться;
- Catalog – имя базы данных.

Данные настройки следует изменить для своего рабочего места и варианта задания. ВНИМАНИЕ! Данная настройка актуальна для подключения к СУБД MS SQL. Для подключения к иной СУБД потребуется иной формат строки подключения и вызова другого метода, а не UseSqlServer.

Далее через DbSet прописываются таблицы, которые следует создать в базе данных.

Следующий шаг – создание базы данных. Для этого нам понадобится «Консоль диспетчера пакетов». Ее можно открыть через «Средства -> Диспетчер пакетов NuGet». В открывшейся вкладке выбираем «Проект по умолчанию» проект с файлом от DbContext (**AbstractShopDatabaseImplement**). В нем будем прописывать команду создания миграции (рисунок 3.2).

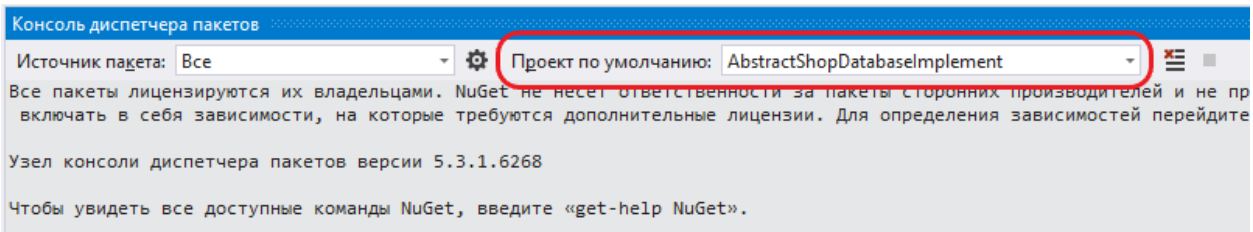


Рисунок 3.2 – Консоль диспетчера пакетов

В Консоли диспетчера пакетов прописываем команду создания миграции (листинг 3.5).

```
Add-Migration InitialCreate
```

Листинг 3.5 – Команда создания миграции

Здесь InitialCreate – название миграции (как правило для первой пишут, что это первая миграция или инициализация);

После успешного выполнения в проекте появится папка «Migrations». В ней будет хранится снимок базы данных и все миграции, создаваемые в ходе разработки (рисунок 3.3).

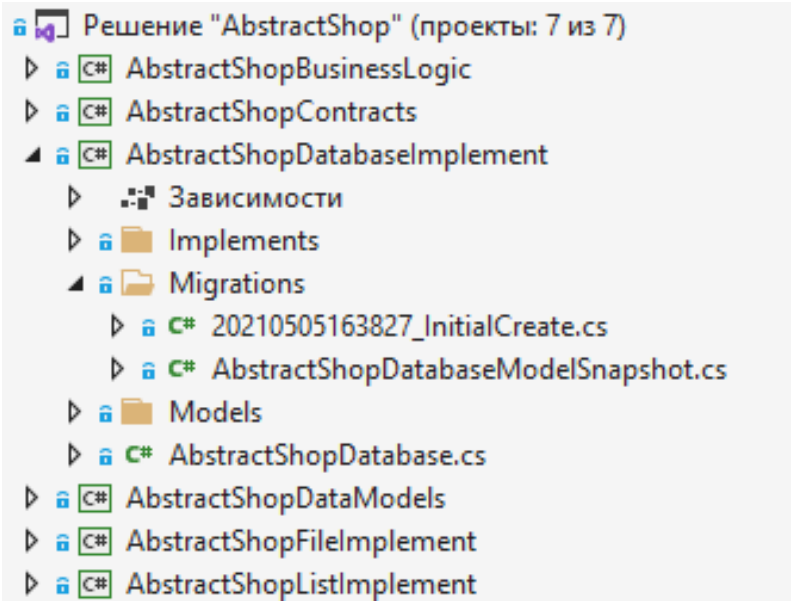


Рисунок 3.3 – Обозреватель решений

У нас есть снимок БД, но нет самой базы. Чтобы она появилась в консоли применить команду изменения базы данных (листинг 3.6).

Если все сделано верно, то в результате будет создана (или обновлена) база данных (рисунок 3.4).

```
Update-Database
```

Листинг 3.6 – Команда применения миграции

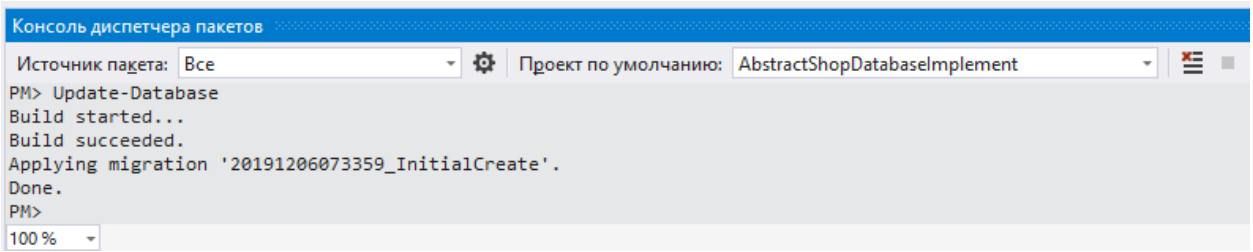


Рисунок 3.4 – Результат выполнения команды

Остается прописать реализации интерфейсов. Для начала, сделаем реализацию для компонента (листинг 3.7).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.StoragesContracts;
```

```
using AbstractShopContracts.ViewModels;
using AbstractShopDatabaseImplement.Models;

namespace AbstractShopDatabaseImplement.Implements
{
    public class ComponentStorage : IComponentStorage
    {
        public List<ComponentViewModel> GetFullList()
        {
            using var context = new AbstractShopDatabase();
            return context.Components
                    .Select(x => x.GetViewModel)
                    .ToList();
        }

        public List<ComponentViewModel> GetFilteredList(ComponentSearchModel
model)
        {
            if (string.IsNullOrEmpty(model.ComponentName))
            {
                return new();
            }
            using var context = new AbstractShopDatabase();
            return context.Components
                    .Where(x => x.ComponentName.Contains(model.ComponentName))
                    .Select(x => x.GetViewModel)
                    .ToList();
        }

        public ComponentViewModel? GetElement(ComponentSearchModel model)
        {
            if (string.IsNullOrEmpty(model.ComponentName) && !model.Id.HasValue)
            {
                return null;
            }
            using var context = new AbstractShopDatabase();
            return context.Components
                    .FirstOrDefault(x =>
(!string.IsNullOrEmpty(model.ComponentName) && x.ComponentName ==
model.ComponentName) ||
                                            (model.Id.HasValue && x.Id == model.Id))
                    ?.GetViewModel;
        }

        public ComponentViewModel? Insert(ComponentBindingModel model)
        {
            var newComponent = Component.Create(model);
            if (newComponent == null)
            {
                return null;
            }
            using var context = new AbstractShopDatabase();
            context.Components.Add(newComponent);
            context.SaveChanges();
            return newComponent.GetViewModel;
        }

        public ComponentViewModel? Update(ComponentBindingModel model)
        {
            using var context = new AbstractShopDatabase();
            var component = context.Components.FirstOrDefault(x => x.Id ==
model.Id);
            if (component == null)
            {
                return null;
```

```
            }
            component.Update(model);
            context.SaveChanges();
            return component.GetViewModel;
        }

        public ComponentViewModel? Delete(ComponentBindingModel model)
        {
            using var context = new AbstractShopDatabase();
            var element = context.Components.FirstOrDefault(rec => rec.Id ==
model.Id);
            if (element != null)
            {
                context.Components.Remove(element);
                context.SaveChanges();
                return element.GetViewModel;
            }
            return null;
        }
    }
}
```

Листинг 3.7 – Класс ComponentStorage

Рассмотрим, что здесь происходит. В каждом методе сперва подключаемся к БД. Можно сделать, чтобы у нас был один объект от AbstractShopDatabase и мы имели постоянное соединение с базой, но это не лучшее решение. Лучше, каждый раз подключатся, выполнять нужные действия и отключаться, позволяя другим клиентам подключаться к этой базе данных (так как количество подключений ограничено). Далее идут запросы к таблице базы данных (в данном случае к таблице «Components») с получением либо набора данных, либо конкретного значения. При добавлении нам не нужно теперь вычислять идентификатор. EntityFramework распознает поля с Id и сам применяет к ним необходимые параметры, в частности, подобное поле в БД помечается как первичный ключ и, если это число, оно будет автоинкрементным, так что при добавлении можно убрать логику вычисления maxId. При добавлении/редактировании/удалении последним действием будет вызов метода SaveChanges у context для сохранения изменений в БД (без этой команды все изменения будет сохраняться в копии БД в программе и не зафиксируются в реальной БД).

Теперь класс-реализация интерфейса IProductStorage. Во многом он будет идентичен классу ComponentStorage, но в методах добавления/редактирования потребуется добавить транзакции, чтобы либо

сохранялось все (и само изделие и связи с компонентами), либо был полный откат сохранения. При удалении удаляем только изделие. За счет создания связей в БД все записи в классе ProductComponent, связанные с этим изделием, удаляться автоматически. Также есть момент, связанный с чтением данных из БД. При чтении сначала придется выполнить запрос на получение данных (одной записи или всех) из БД (вызываем ToList после метода Where) и только потом преобразовывать через метод Select. Это связано с тем, что создание картежей (используется в значении в словаре) невозможно при запросе к БД. А для того, чтобы в классе Product при получении записей для ProductComponents всегда были у каждого элемента Component, надо в запросах прописать команды Include. Такие команды говорят, что вместе с исходной записью подтягивать зависимости, например, родительскую или дочерние. В SQL-запросе такой Include может представляться простым оператором Join (листинг 3.8).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.StoragesContracts;
using AbstractShopContracts.ViewModels;
using AbstractShopDatabaseImplement.Models;
using Microsoft.EntityFrameworkCore;

namespace AbstractShopDatabaseImplement.Implements
{
    public class ProductStorage : IProductStorage
    {
        public List<ProductViewModel> GetFullList()
        {
                    using var context = new AbstractShopDatabase();
                    return context.Products
                     .Include(x => x.Components)
                     .ThenInclude(x => x.Component)
                     .ToList()
                                 .Select(x => x.GetViewModel)
                                 .ToList();
             }

        public List<ProductViewModel> GetFilteredList(ProductSearchModel model)
             {
                    if (string.IsNullOrEmpty(model.ProductName))
                    {
                          return new();
                    }
                    using var context = new AbstractShopDatabase();
                    return context.Products
                     .Include(x => x.Components)
                     .ThenInclude(x => x.Component)
                     .Where(x => x.ProductName.Contains(model.ProductName))
                                 .ToList()
                                 .Select(x => x.GetViewModel)
```

```
                                .ToList();
            }

        public ProductViewModel? GetElement(ProductSearchModel model)
            {
                  if (string.IsNullOrEmpty(model.ProductName) &&
!model.Id.HasValue)
                  {
                        return null;
                  }
                  using var context = new AbstractShopDatabase();
                  return context.Products
                .Include(x => x.Components)
                .ThenInclude(x => x.Component)
                .FirstOrDefault(x => (!string.IsNullOrEmpty(model.ProductName) &&
x.ProductName == model.ProductName) ||
                                                      (model.Id.HasValue && x.Id ==
model.Id))
                        ?.GetViewModel;
            }

        public ProductViewModel? Insert(ProductBindingModel model)
            {
                  using var context = new AbstractShopDatabase();
                  var newProduct = Product.Create(context, model);
                  if (newProduct == null)
                  {
                        return null;
                  }
                  context.Products.Add(newProduct);
                  context.SaveChanges();
                  return newProduct.GetViewModel;
            }

        public ProductViewModel? Update(ProductBindingModel model)
        {
                  using var context = new AbstractShopDatabase();
                  using var transaction = context.Database.BeginTransaction();
                  try
                  {
                        var product = context.Products.FirstOrDefault(rec =>
rec.Id == model.Id);
                        if (product == null)
                        {
                              return null;
                        }
                        product.Update(model);
                        context.SaveChanges();
                        product.UpdateComponents(context, model);
                        transaction.Commit();
                        return product.GetViewModel;
                  }
                  catch
                  {
                        transaction.Rollback();
                        throw;
                  }
            }

        public ProductViewModel? Delete(ProductBindingModel model)
        {
                  using var context = new AbstractShopDatabase();
                  var element = context.Products
                        .Include(x => x.Components)
                        .FirstOrDefault(rec => rec.Id == model.Id);
```

```
                    if (element != null)
                    {
                        context.Products.Remove(element);
                        context.SaveChanges();
                        return element.GetViewModel;
                    }
                    return null;
                }
            }
}
```

Листинг 3.8 – Класс ProductStorage

Также реализовать интерфейс IOrderStorage самостоятельно.

Последний штрих. В проект AbstractShopView добавить ссылку на AbstractShopDatabaseImplement. И в классе Program поменять всего одну строку. Вместо

```
using AbstractShopFileImplement.Implements;
```

Вставить

```
using AbstractShopDatabaseImplement.Implements;
```

В результате приложение будет работать с другой реализацией и хранить данные в базе данных.

**Контрольный пример**

В СУБД будет новая база данных, в которых можно просмотреть записи, внесенные через приложение. При повторных запусках приложения данные будут сразу доступны пользователям.

**Требования**

1. Название проектов должны ОТЛИЧАТЬСЯ от названия проектов, приведенных в примере и должны соответствовать логике вашего задания по варианту.
2. Название форм, классов, свойств классов должно соответствовать логике вашего задания по варианту.
3. НЕ ИСПОЛЬЗОВАТЬ в названии класса, связанного с изделием слово «Product» (во вариантах в скобках указано название класса для изделия)!!!

4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Название базы данных должно соответствовать логике вашего задания по варианту.
6. Создать модель для заказа с правильной расстановкой связей.
7. Реализовать интерфейс IOrderStorage для хранения данных в БД.

**Порядок сдачи базовой части**

1. Запустить приложение (должны отображаться созданные до этого заказы).
2. Открыть справочник по компонентам (должны отобразиться имеющиеся компоненты).
3. Открыть справочник по изделиям (должны отобразиться имеющиеся изделия).
4. Ответить на вопрос преподавателя.

**Контрольные вопросы к базовой части**

1. Какой основной класс для взаимодействия ORM EntityFramework с проектом?
2. Из каких элементов состоит класс миграции и когда они вызываются?
3. Как задается связь «один-ко-многим» или «многие-ко-многим» через модели сущностей? Показать на примере проекта.

**Усложненная лабораторная (необязательно)**

1. Сохранять в базу информацию по магазинам и их загруженность.
2. Сделать реализацию интерфейса логики хранения данных для сущности «Магазин» с сохранением в базу данных (не должно быть 2-х магазинов с одним и тем же названием).

3. В логике для хранения данных «Магазин» метод проверки и продажи изделий из магазинов требуемом количестве реализовать с использованием транзакций (т.е., не должно быть проверки, а сразу идти продажа с откаткой в случае нехватки изделий).

**Порядок сдачи усложненной части**

1. Показать магазины (сколько изделий в них расположено).
2. Попытаться продать заведомо больше число изделий.
3. Пополнить магазины изделиями через заказы.
4. Продать изделие.
5. Показать магазины (сколько изделий в них расположено).
6. Ответить на вопрос преподавателя.

**Контрольные вопросы к усложненной части**

1. Какова логика проверки достаточности изделий для продажи?
2. Какова логика продажи изделий из магазина?
3. Как настроены связи сущности «Магазин»?

**Варианты**

1. Кондитерская. В качестве компонентов выступают различные виды шоколада и наполнители, типа орехов, изюма и т.п. Изделие – кондитерское изделие (pastry).
2. Автомастерская. В качестве компонентов выступают различные масла, смазки и т.п. Изделия – ремонт автомобиля (repair).
3. Моторный завод. В качестве компонентов выступают различные детали для производства двигателей. Изделия – двигатели (engine).
4. Суши-бар. В качестве компонентов выступают различные продукты для суши (рыба, водоросли, соусы). Изделия – суши (sushi).

5. Продажа компьютеров. В качестве компонентов выступают различные части для компьютеров (планки памяти, жесткие диски и т.п.). Изделия – компьютеры (computer).
6. Сборка мебели. В качестве компонентов выступают различные заготовки (ножки, спинки и т.п.). Изделия – мебель (furniture).
7. Рыбный завод. В качестве компонентов выступают различные виды рыб + дополнения к ним, типа соусов и т.п. Изделия – консервы (canned).
8. Установка ПО. В качестве компонентов выступают различное ПО. Изделия – пакеты установки, например, пакет установки офисных приложений, пакет разработчика и т.п. (package).
9. Ремонтные работы в помещении. В качестве компонентов выступают различные расходные материалы (клей, обои, краска, плитка, цемент и т.п.). Изделия – ремонтные работы в различных помещениях (repair).
10. Кузнечная мастерская. В качестве компонентов выступают различные болванки (заготовки), из которых изготавливаются подковы, кочерги и т.п. Изделия – кузнечные изделия (manufacture).
11. Пиццерия. В качестве компонентов выступают различные ингредиенты для пицц (тесто, соусы, паста и т.д.). Изделия – пиццы (pizza).
12. Завод ЖБИ. В качестве компонентов выступают различные виды бетона и металлоконструкций. Изделия – железобетонные изделия (reinforced).
13. Закусочная. В качестве компонентов выступают различные продукты для закусок (колбаса, сыр, хлеб и т.п.). Изделия – различные закуски (snack).
14. Пошив платьев. В качестве компонентов выступают различные ткани, нитки и т.п. Изделия – платья (dress).

15. Типография. В качестве компонентов выступают различные типы бумаг, тонер или чернила и т.п. Изделия – печатная продукция (листовки, брошюры, книги) (printed).
16. Автомобильный завод. В качестве компонентов выступают различные части для сборки автомобилей (кузов, двигатель, стекла и т.п.). Изделия – автомобили (car).
17. Юридическая фирма. В качестве компонентов выступают различные бланки для документов. Изделия – пакеты документов, например, для страховки или завещания (document).
18. Туристическая фирма. В качестве компонентов выступают различные условия поездки (отель проживания, туры в рамках поездок). Изделия – туристические путевки (travel).
19. Цветочная лавка. В качестве компонентов выступают различные цветы и украшения к ним. Изделия – цветочные композиции (flower).
20. Ювелирная лавка. В качестве компонентов выступают различные драгоценные камни и металлы. Изделия – драгоценности (jewel).
21. Авиастроительный завод. В качестве компонентов выступают различные части для сборки самолета (двигатели, крылья, фюзеляж и т.п.). Изделия – самолеты (plane).
22. Магазин подарков. В качестве компонентов выступают различные упаковочные материалы, ленты и подарки. Изделия – подарочные наборы (gift).
23. Система безопасности. В качестве компонентов выступают различные камеры, датчики и т.п. Изделия – базовые комплектации охраны, продвинутые, для предприятий, для частных и т.п. (secure).
24. Заказы еды. В качестве компонентов выступают различные блюда. Изделия – это наборы блюд (типа обеденный набор, или утренний набор, или набор для пикника) (dish).

25. Ремонт сантехники. В качестве компонентов выступают различные трубы, прокладки, смесители т.п. Изделия – замены смесителей, труб и т.п. (work).
26. Лавка с мороженым. В качестве компонентов выступают различные виды мороженного и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icecream).
27. Судостроительный завод. В качестве компонентов выступают различные части для сборки судов (корпуса, двигатели и т.п.). Изделия – суда (ship).
28. Столярная мастерская. В качестве компонентов выступают различные деревянные заготовки. Изделия – деревянные игрушки, утварь и т.п. (wood).
29. Бар. В качестве компонентов выступают различные ингредиенты для коктейлей. Изделия – коктейли (cocktail).
30. Швейная фабрика. В качестве компонентов выступают различные заготовки для штор, покрывал и т.п. (textile).