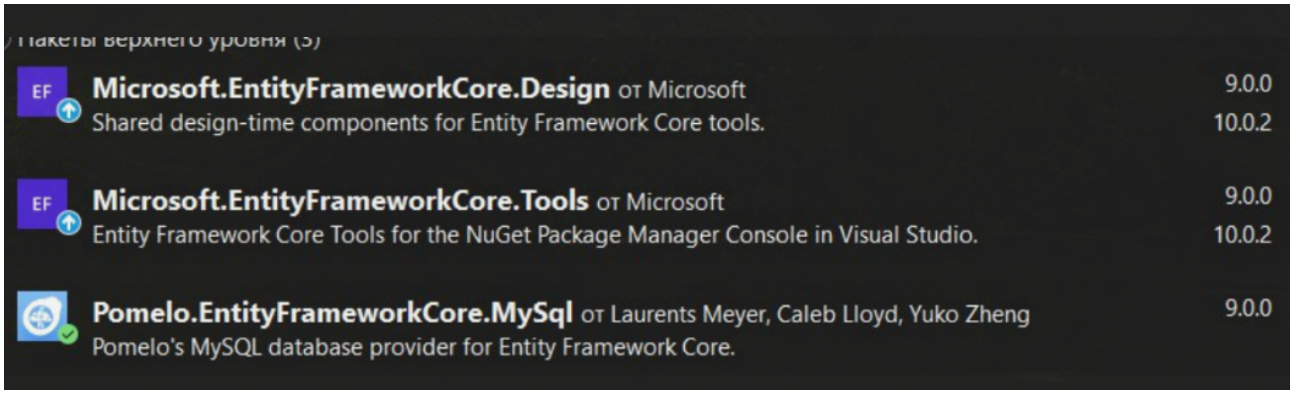
Подключение через Entity Framework и Database First

Сначала, для выполнения задания нужно создать новый проект в Visual Studio, а именно проект Windows Forms (Майкрософт). Требуется именно .NET (Core) версия форм, а не .NetFramework, иначе подключение пакетов не сработает.
Для подключения требуются следующие пакеты
- Microsoft.EntityFrameworkCore.Design 9 версия
- Microsoft.EntityFrameworkCore.Tools 9 версия
Для импорта в проект

Pomelo.EntityFrameworkCore.MySQL
Провайдер



Дальше, требуется найти консоль диспетчера пакетов, если консоли не обнаружено, то следует нажать на вкладку:
Вид > Консоль Диспетчера пакетов.

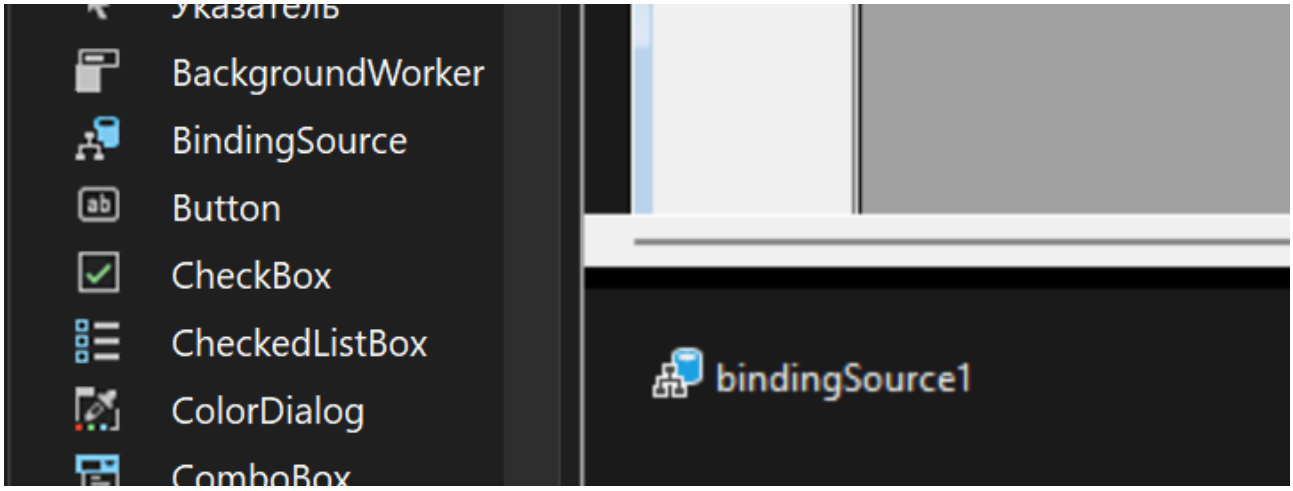
Далее, введите команду:
Scaffold-DbContext «Ваша строка подкоючения»
«Pomelo.EntityFrameworkCore.MySQL» 9 версия

Следующая команда: Scaffold-DbContext "server = cfif31.ru; username = username; password=youtpass; database=dbname" "Pomelo.EntityFrameworkCore.MySql" -OutputDir Models -f.

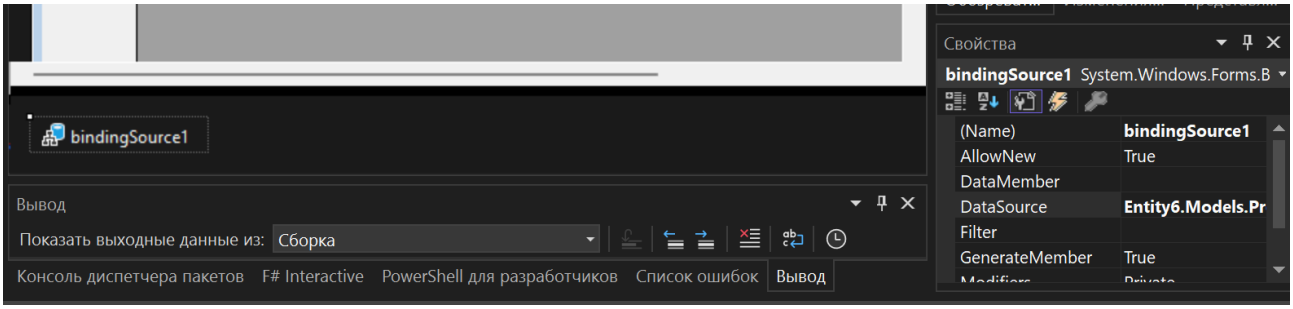
После успешного выполнения команды, выполните сборку проекта **Сборка > Собрать решение.** Это нужно, чтобы Visual Studio смогла получить информацию о типах данных, полученных при сборке.

CRUD через datagridview и Binding Source

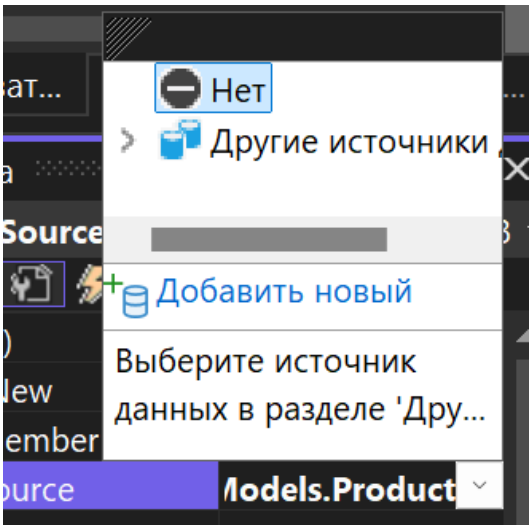
Добавьте на форму компонент BindingSource



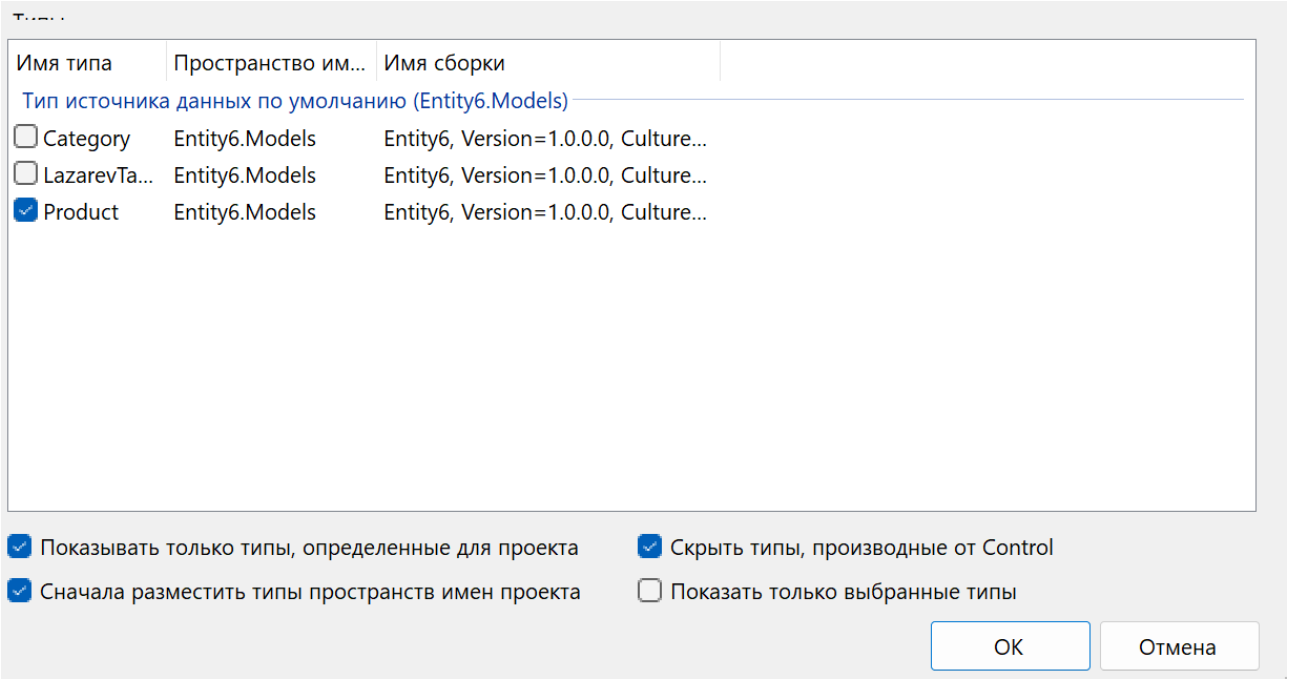
Далее, заходите в свойства у BindingSource



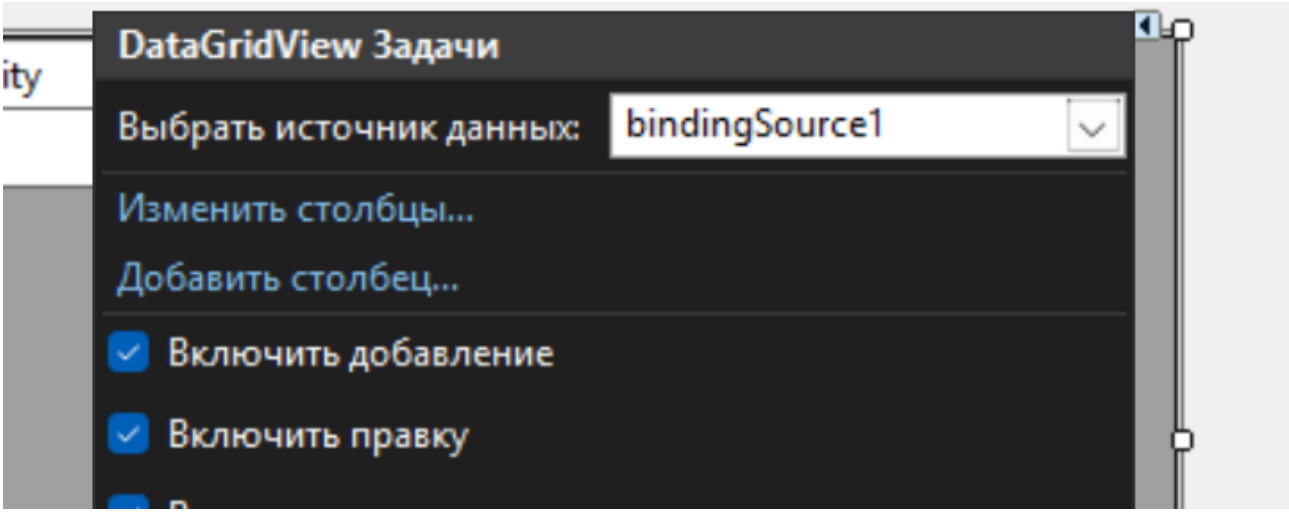
Затем требуется нажать на DataSource, а именно на стрелочку справа, чтобы раскрылось специальное меню



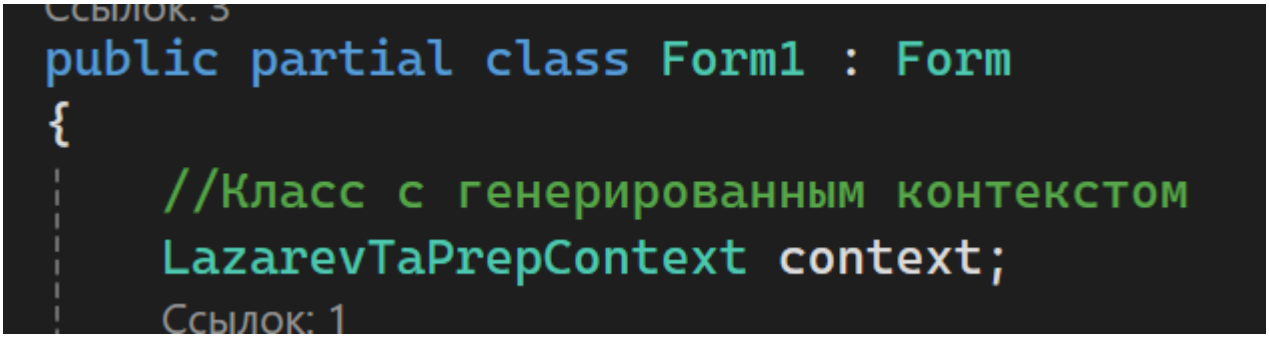
Затем выбрать ваши сущности, используемые для вывода



Требуется нажать ОК. Далее, разместите dataGridView на форме, нажмите стрелочку и выберите созданный источник данных:



Запишите созданный на предыдущих шагах класс контекста, состоящий из имени базы данных и слова Context



Затем требуется записать следующую логику в OnLoad().
В инициализации контекста требуется заменить контекст на импортированный.
В загрузке сущностей должен быть указан ваш сгенерированный класс.

```
Ссылок: 0
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);

    //инициализация контекста
    context = new LazarevTaPrepContext();

    //Загрузка сущностей из базы
    context.Products.Load();

    //Проверка на создание базы
    context.Database.EnsureCreated();

    //привязка данных
    bindingSource1.DataSource = context.Products.Local.ToBindingList();
}
```

Затем требуется создать кнопку и оформить следующее событие щелчка.

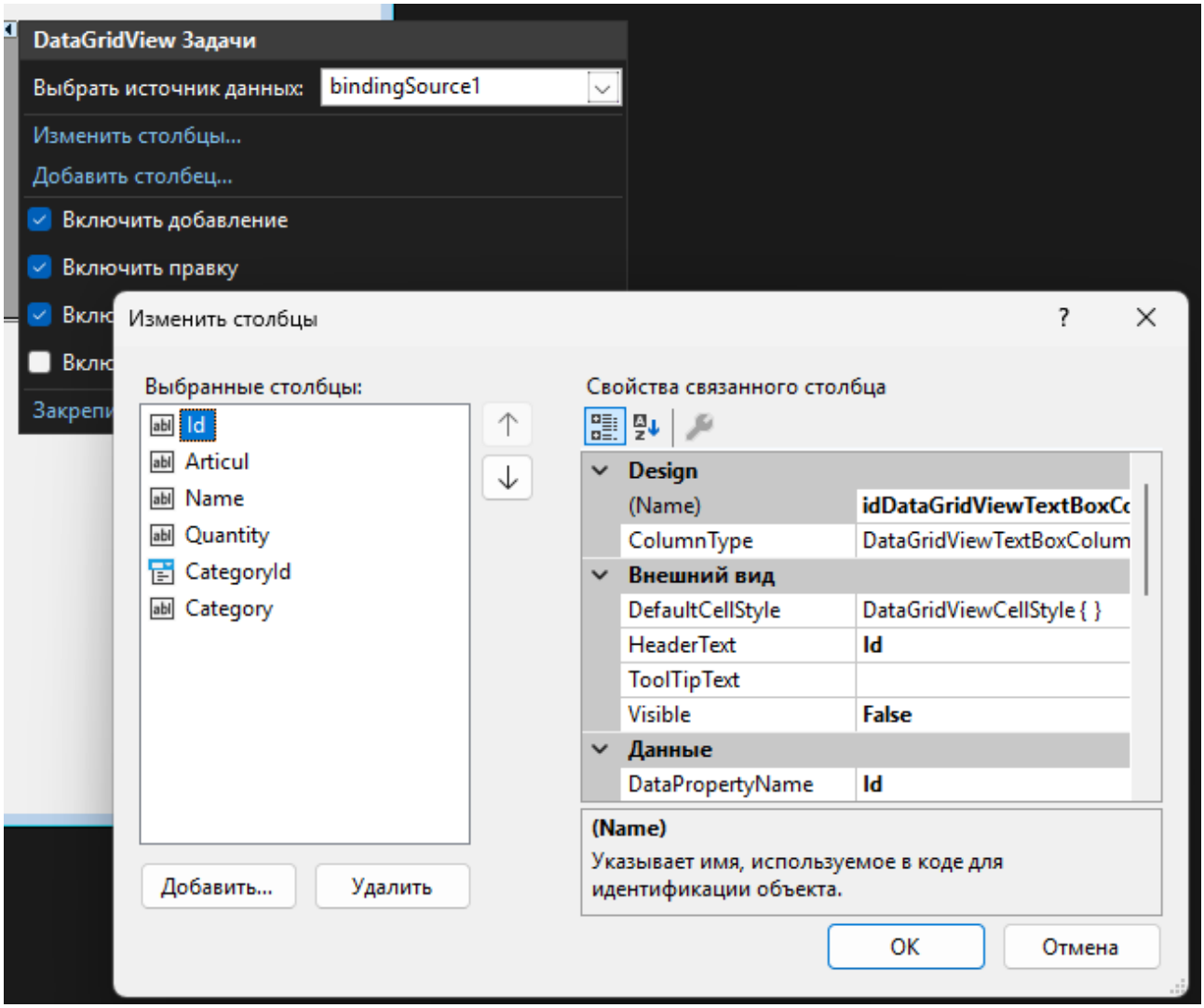
```
Ссылок: 1
private void saveButton_Click(object sender, EventArgs e)
{
    //Сохранение отслеживаемых сущностей в бд
    context.SaveChanges();
    dataGridView1.Refresh();
}
```

После сборки и запуска проекта можно убедиться, что записи в dataGridView можно добавлять, удалять, изменять без написания SQL запросов.

Редактирование внешнего вида колонок

Клиенту не нужно знать об идентификаторах, когда вы работаете с приложением.

Чтобы бы убрать ту или иную колонку, нужно нажать на Изменить столбцы.

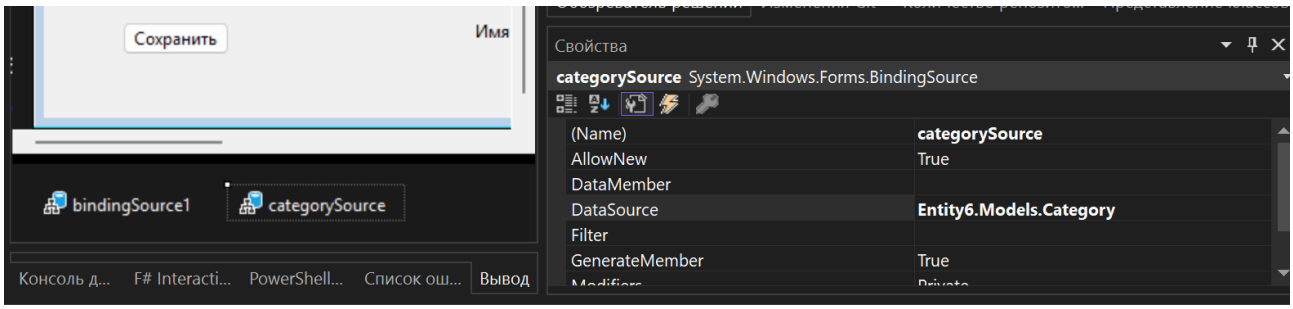


Здесь в категории Внешний вид, интересующими свойствами являются HeaderText и Visible, отвечающими соответственно за название колонки и видимость.

Смежные таблицы

Объединение через колонки Windows Forms

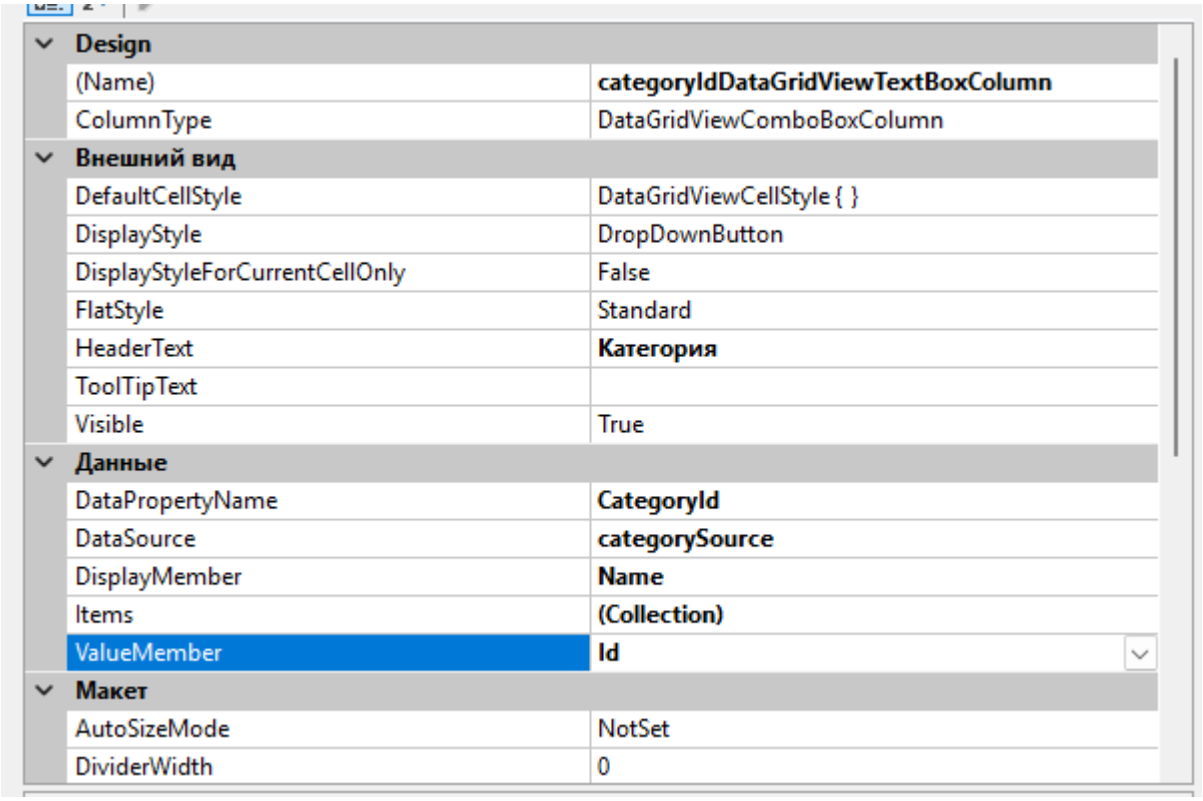
Требуется создать новый источник данных (bindingSource для категорий) и указать ваш класс категорий, который соответствует внешней, связанной сущности.



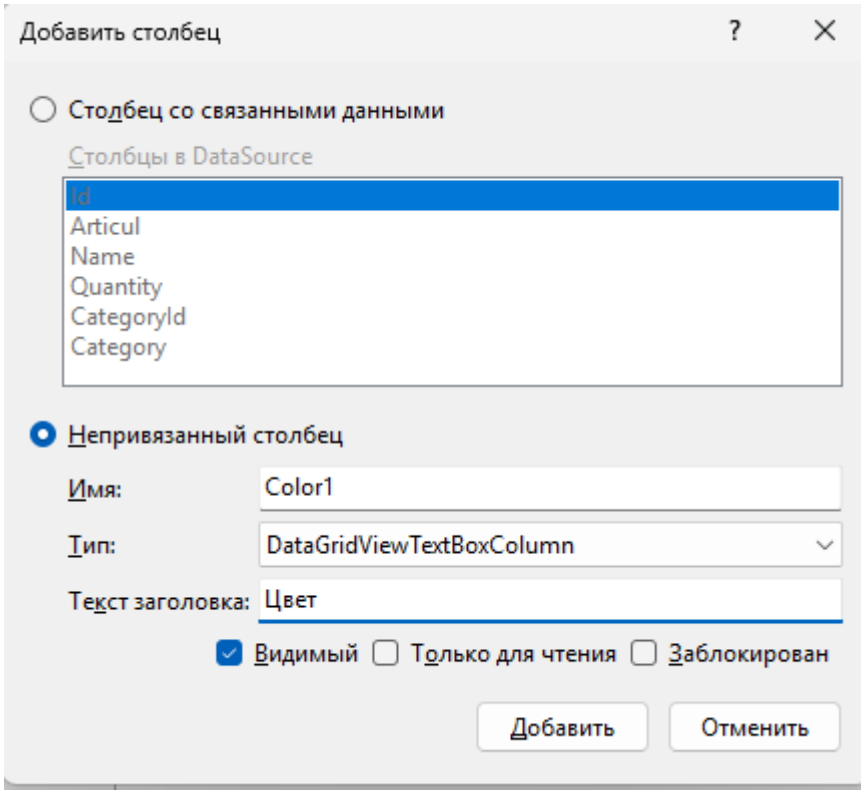
Следующая логика заполнения

```
{
    base.OnLoad(e);

    //инициализация контекста
    context = new LazarevTaPrepContext();

    //Загрузка сущностей из базы
    context.Products.Include(x => x.Category).Load();
    context.Categories.Load();

    //Проверка на создание базы
    context.Database.EnsureCreated();

    //привязка данных
    bindingSource1.DataSource = context.Products.Local.ToBindingList();
    categorySource.DataSource = context.Categories.Local.ToBindingList();
}
```

Далее перейдите в изменение колонок, выберите колонку идентификатором вашей связанной таблицы и измените columnType на указанный в скриншоте и измените колонку для того, чтобы вместо идентификатора показывалось имя товара.

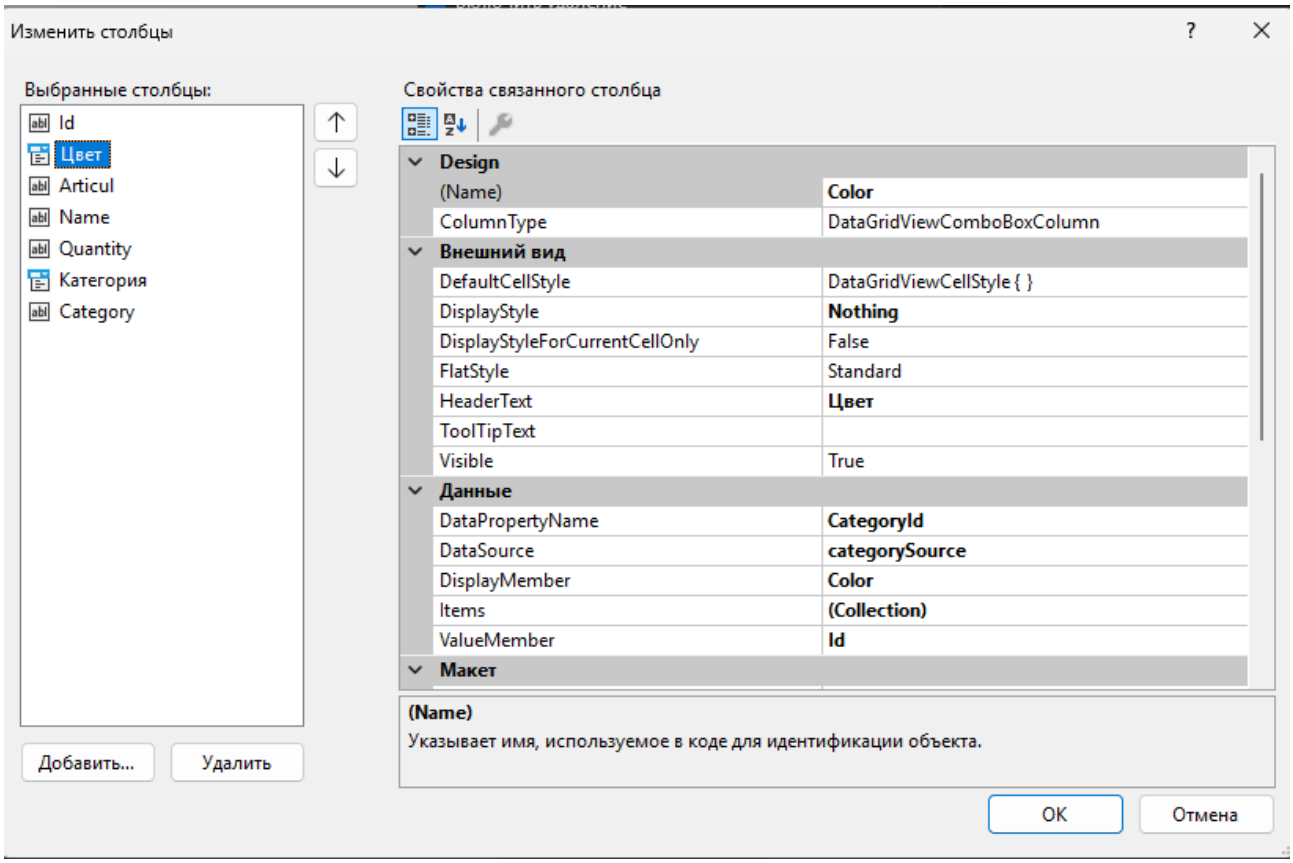


Для просмотра полей в связанной таблице, вы сначала должны добавить несвязанный столбец



Далее в ColumnType Нужно подставить DataGridViewComboBoxColumn

В настройках столбцах нужно указать в DataPropertyName идентификатор внешней сущности. В dataSource указать источник таблицы. В DisplayMember нужно подставить интересующую колонку. В ValueMember также указать её идентификатор.



Объединение через создание свойств

Требуется повторить шаг с Include в предыдущем пункте. Создайте отдельный cs файл в проекте и добавьте туда partial класс со свойством, которое получает значение из другого вложенного свойства или вычисляет новое значение:

```
namespace WinFormsAppEntity.Models
{
    Ссылок: 13
    public partial class Product
    {
        //Вычисление новой колонки
        [NotMapped]
        Ссылок: 0
        public string NameQuantity => Category.Name + ": " + Name + " " + Quantity.ToString() + " шт." ;


        // Получение вложенной колонки
        [NotMapped]
        Ссылок: 0
        public string CategoryColor => Category.Color;

    }
}
```

После этого требуется пересобрать проект и добавить новую колонку в конструкторе форм

Класс обязательно должен partial и иметь тот-же namespace, как и у сгенерированного ef Product.

Подробная инструкция, как присоединять таблицы через Include (аналог Join) в entity framework описана в следующей статье

https://learn.microsoft.com/ru-ru/ef/core/querying/related-data/eager

Поиск и фильтрация

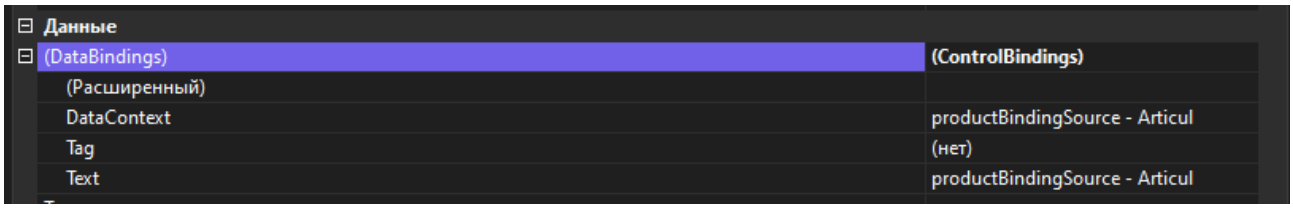
В следующем фрагменте продемонстрированы поиск и фильтрация по категории и продукту при нажатии на кнопку:

```
Ссылок: 1
private void applySearch_Click(object sender, EventArgs e)
{
    var product = productSearch.Text;  //получение строк для фильтрации из текстбоксов
    var category = categorySearch.Text;

    context.ChangeTracker.Clear();  //Очистка контекста от объектов
    context.Products
        .Include(x => x.Category) //joinФ
        .Where(x => x.Name.Contains(product) && x.Category.Name.Contains(category)) //Предикат
        .Load();


    productBindingSource.DataSource = context.Products.Local.ToBindingList(); //перепривязка
    dataGridView1.Refresh(); //Обновление
}
```

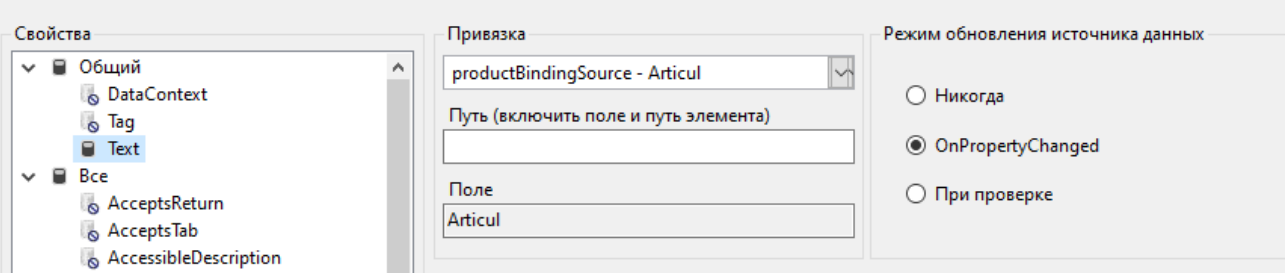
Можно организовать поиск в реальном времени, для этого надо привязать не к событию клика, а к событию TextChanged, для этого трубуется перейти в свойства у textbox и нажать на молнию, где найти событие textChanged и кликнуть по нему. После это создания нового метода, напишите в нем ту же логику фильтрации

Редактирование записи

Создайте новую форму и добавьте поля. Далее зайдите в один из текстбоксов и найдите категорию Данные, а в неё (DataBindings).



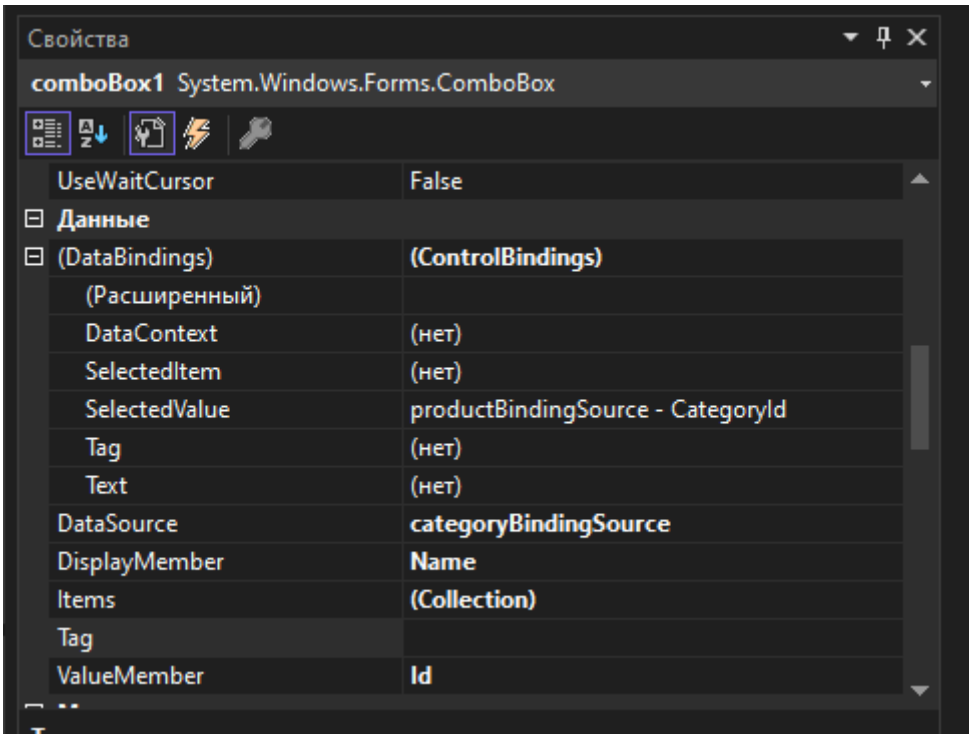
Затем нужно выбрать меню в поле расширенный. Если кликнуть на пустой слот, то появится троеточие, на которое нужно, чтобы открылось следующее меню.



Можно выбрать источник данных для каждого свойства слева, для вывода информации в текстбоксе следует выбрать Text

Если вы не создали источник данных для выводимой сущности, то можете выбрать его из существующих классов, а именно из **других источников данных**, где вы можете выбрать подходящий класс.

Для редактирования связанных сущностей следует воспользоваться отдельными источниками данных. Для настройки привязки требуется создать поле комбобокс и зайти в свойство Данные:



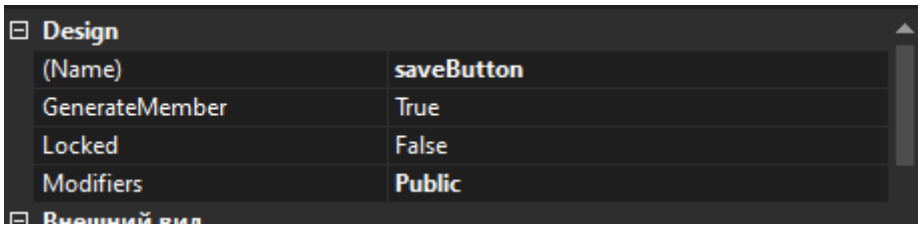
Внутри (DataBindings) поле Selected Value должно соответствовать ключу связанной сущности внутри редактируемой сущности.

DataSource соответствует источнику данных, из которого можно будет выбирать категории.

В DisplayMember можно выбрать атрибут у сущности по которому в комбобоксе будет выводится текст.

В ValueMember находится первичный ключ сущности, соответствующий внешнему ключу у редактируемой сущности.

Далее требуется создать на кнопку с модификатором public, чтобы можно было привязаться к её событиям из другой формы.



Модификатор Public, таким же образом, нужно выставить для всех источников данных в проекте

Затем перейдите на исходную форму, где отображаются все данные и добавьте логику для кнопки редактирования данных.

```
//обработка редактирования
Ссылок: 1
private void editButton_Click(object sender, EventArgs e)
{
    var form = new Form2();
    //Пробрасывание источника данных категорий в форму для редактирования
    form.categoryBindingSource.DataSource = categoryBindingSource.DataSource;
    //Пробрасывание выбранного элемента в источнике данных в форму
    form.productBindingSource.DataSource = bindingSource1.Current;
    //Переиспользование логики сохранения
    form.saveButton.Click += saveButton_Click;
    form.Show();
}

Ссылок: 2
private void saveButton_Click(object sender, EventArgs e)
{
    context.SaveChanges();
    dataGridView1.Refresh();
}
```

Добавление и удаление

Логика добавления и удаления следующая

```
Ссылок: 1
private void add_Click(object sender, EventArgs e)
{
    var form = new Form2();
    form.categoryBindingSource.DataSource = categoryBindingSource.DataSource;
    //одновременно добавляет элемент в источник данных и на форму
    form.productBindingSource.DataSource = bindingSource1.AddNew();
    form.saveButton.Click += saveButton_Click;
    form.Show();
}

Ссылок: 1
private void delete_Click(object sender, EventArgs e)
{
   //удаляет текущий элемент
   bindingSource1.RemoveCurrent();
   context.SaveChanges();
   dataGridView1.Refresh();
}
```

Валидация сущностей

Валидация происходит с использованием интерфейса IEditableObject, отвечающий за транзационное добавление сущностей в базу. Эту абстракцию использует BindingSource для добавления данных. В ней же можно и проводить валидацию сущностей.

Такой класс может выглядеть следующим образом:

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
namespace WinFormsAppEntity.Models;

Ссылок: 10
public partial class Product : IEditableObject
{
    LazarevTaPrepContext context;

    Ссылок: 0
    public void BeginEdit()
    {
        context = new();
        context.Update(this);
    }

    Ссылок: 0
    public void CancelEdit()
    {

        var entry = context.Entry(this);
        if (entry.State == Microsoft.EntityFrameworkCore.EntityState.Modified)
        {
            entry.Reload();
        }

    }

    Ссылок: 0
    public void EndEdit()
    {
        //Логика валидации
        if (string.IsNullOrEmpty(Name))
        {
            throw new ValidationException("Имя не должно быть пустым");
        }
        //Сохраняет измененную сущность
        context.SaveChanges();
    }
}
```

В BeginEdit инициализируется ваш контекст, который ничего не знает о текущем объекте, поэтому нужен context.Update, чтобы EF стал отслеживать его.

В CancelEdit присутствует Reload для загрузки сущности из баз данных.

CancelEdit вызывается напрямую из кода и из datagridview, когда строка с проблемой выходит из фокуса

```
Ссылок: 2
private void saveButton_Click(object sender, EventArgs e)
{
    try
    {
        bindingSource1.EndEdit();
    }
    catch (ValidationException error)
    {
        MessageBox.Show(error.Message);
        bindingSource1.CancelEdit();
    }


    dataGridView1.Refresh();
}

Ссылок: 1
private void add_Click(object sender, EventArgs e)
{
    var form = new Form2();
    form.categoryBindingSource.DataSource = categoryBindingSource.DataSource;
    form.productBindingSource.DataSource = bindingSource1.AddNew();
    form.saveButton.Click += (o, e) =>
    {
        try
        {
            bindingSource1.EndEdit();
        }
        catch (ValidationException error)
        {
            MessageBox.Show(error.Message);
        }
    };
    ;
    form.Show();
}
```

Если при добавлении выйдет исключение, то сущность не будет добавляться как в BindingSource, так и в базу данных

Авторизация

Для следующего примера требуется самостоятельно создать окно авторизации и привязать поля, как в примере с редактированием. Для авторизации может использоваться следующий код:

```
using System.Data;
using WinFormsAppEntity.Models;

namespace WinFormsAppEntity
{
    Ссылок: 3
    public partial class Auth : Form
    {
        User user = new();
        Ссылок: 1
        public Auth()
        {
            InitializeComponent();
        }

        Ссылок: 0
        protected override void OnLoad(EventArgs e)
        {
            base.OnLoad(e);
            //нам не нужно обращаться к textbox
            userBindingSource.DataSource = user;
        }

        Ссылок: 1
        private void enter_Click(object sender, EventArgs e)
        {
            var context = new LazarevTaPrepContext();
            //лямбда выражение для поиска
            var u = context.Users.Where(x => x.Login == user.Login && x.Pwd == user.Pwd).FirstOrDefault();
            if (u != null)
            {
                Hide();
                new Form1().Show();
            }
            else
            {
                MessageBox.Show("Пользователь не найден");
            }
        }
    }
}
```

Если присутствует разделение по ролям, то предположительно. В базе данных запись пользователя может выглядеть так:

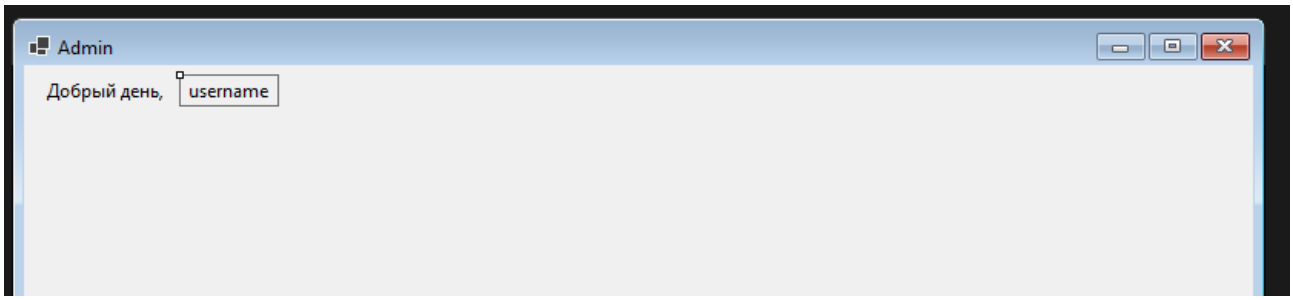
Result Grid | Filter Rows:

idusers	login	pwd	role
1	1	1	1
2	2	2	2
NULL	NULL	NULL	NULL

В таком случае, проверка на пользователя и вывод формы может выглядеть следующим образом:

```
//лямбда выражение для поиска
var u = context.Users.Where(x => x.Login == user.Login && x.Pwd == user.Pwd).FirstOrDefault();
if (u != null)
{
    Hide();
    if (u.Role == 1) //Если пользователь администратор...
        new Admin(u).Show(); //Открой форму для администратора
    if (u.Role == 2) //Если пользователь не администратор...
        new Form1(u).Show(); // Открой обычную форму
}
else
```

Предусмотрительно созданы две формы. Одна для привилегированного пользователя, другая для обычного пользователя. В форму был также передан параметр пользователя, чтобы можно было пользоваться информацией записанной в нём. Так, можно вывести любые данные о пользователе на форму



Так форма администратора может выглядеть следующим образом

```
Ссылок: 3
public partial class Admin : Form
{
    private User user;

    Ссылок: 1
    public Admin(User u)
    {
        user = u;// Мы передали параметр, но теперь нам нужно передать его в поле, чтобы получить достуа
        InitializeComponent();
    }

    //Всегда вызывается при загрузке формы
    Ссылок: 0
    protected override void OnLoad(EventArgs e)
    {
        base.OnLoad(e);
        userLabel.Text = user.Login; //Запись в свойство Text у Label логина пользователя
    }
}
```

Результатом будет выведение логина (или другой информации)

Создание карточек

Создание карточек включает в себя использование FlowLayoutPanel и пользовательского элемента управления. Данный элемент управления используется для автоматического упорядочивания дочерних элементов. Добавьте компонент на экран и проставьте следующие свойства:

Аттрибут	Значение	Комментарий
AutoScroll	true	Автоматическая прокрутка, если элементы не умещаются
FlowDirection	TopDown	Элементы идут сверху вниз
WrapContents	false	Элементы не перекидываются на следующую строку

Для создания пользовательского элемента нажмите правой кнопкой мыши на проект и выберите **Добавить > Пользовательский элемент управления.**

Далее, разместите элементы управления на пользовательском элементе по примеру из параграфа про редактирование на странице 9 и напишите следующую логику в форме.

```
// Заполнение новыми элементами
Ссылок: 2
private void Populate()
{
    flowLayoutPanel1.Controls.Clear(); // Очистка элементов управления
    foreach (var binding in mainBinding) // Получение каждого отдельного элемента связывания
    {
        var control = new CustomElem(); // Создание экземпляра кастомного элемента
        control.productBindingSource.DataSource = binding;
        control.categoryBindingSource.DataSource = catBinding;
        flowLayoutPanel1.Controls.Add(control);  // Добавление нового элемента
    }
}
```

Этот код можно вызывать при создании формы и сохранении данных.

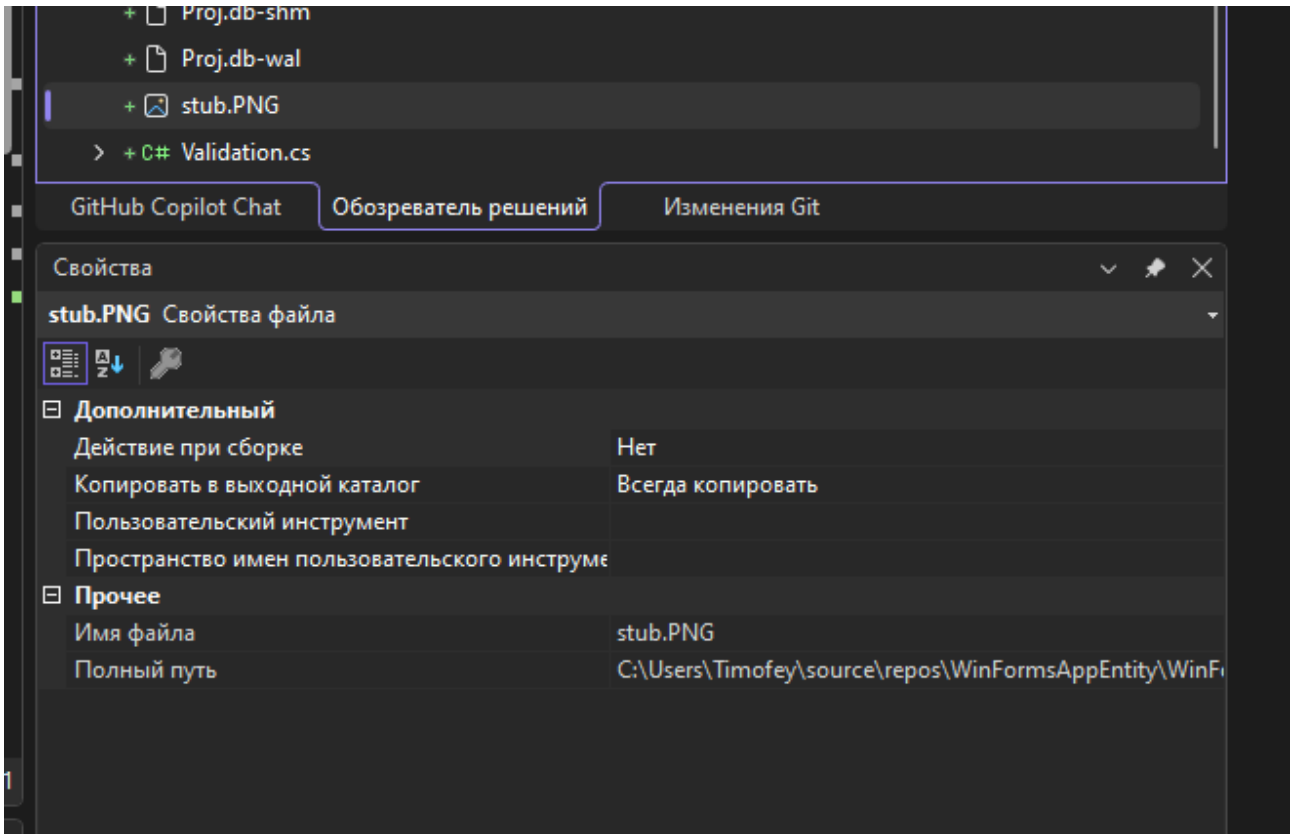
```
            mainBinding.DataSource = context.Products.Local.ToBindin
            catBinding.DataSource  = context.Categories.Local.ToBind
            Populate();
        }

        //Сохранение данных
        Ссылок: 1
        private void button1_Click(object sender, EventArgs e)
        {
            context.SaveChanges();
            dataGridView1.Refresh();
            Populate();
        }
    }
```

Обратите внимание, что для загрузки карточек при старте формы. Метод Populate пишется в OnLoad()

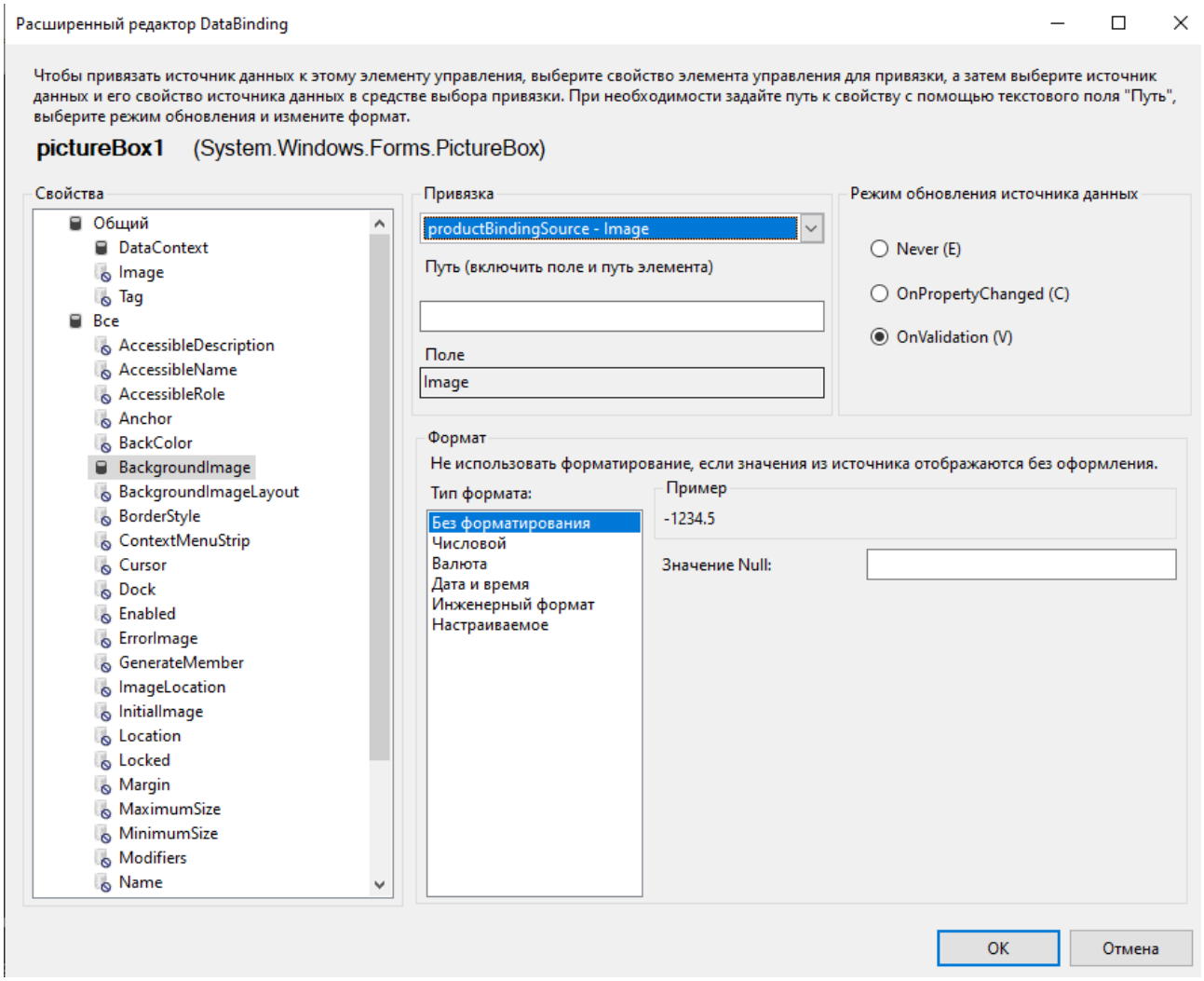
```
Ссылок: 0
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);

    context = new();

    context.Products.Include(x => x.Category).Load();
    context.Categories.Load();
    context.Database.EnsureCreated();
    bindingSource1.DataSource = list = context.Products.Local.ToBindingList();
    categoryBindingSource.DataSource = context.Categories.Local.ToBindingList();
    Populate();
}
```

Работа с изображениями

Для того, чтобы использовать изображение, которые вы вставили в проект. Нужно указать, чтобы оно копировалось при сборке

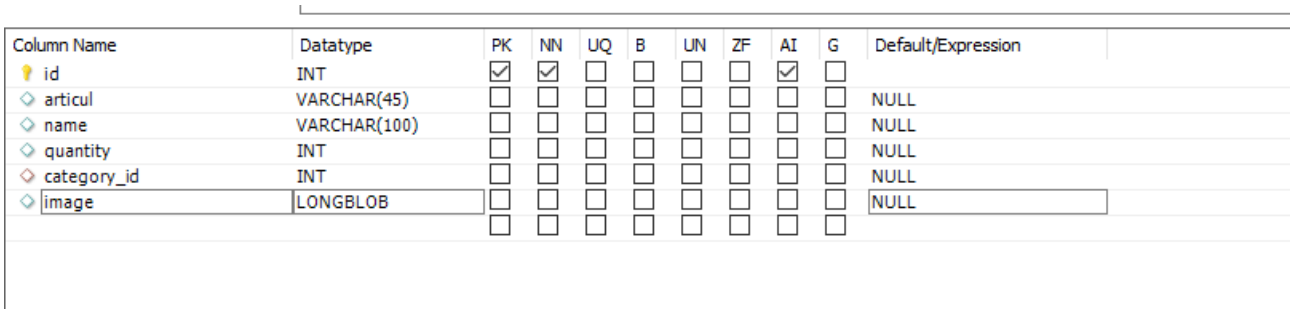


Для привязки к изображению используйте следующую конфигурацию



и в backgroundImageLayout укажите stretch.

Не забудьте, что в базе данных должен быть longblob



Код загрузки карточек можно изменить следующим образом. Но нужно не забыть сделать нужные члены карточки публичными, например кнопку для сохранения изображения

```
Ссылок: 4
private void Populate()
{
    flowLayoutPanel1.Controls.Clear();
    foreach (Product item in bindingSource1)
    {
        var control = new MyControl();
        control.productBindingSource.DataSource = item;
        control.categoryBindingSource.DataSource = categoryBindingSource;
        if (item.Image == null) //изображения нет? Вставляем заглушку!
            item.Image = image;
        control.saveImage.Click += (sender, e) =>
        {
            if (control.openFileDialog1.ShowDialog() == DialogResult.OK)
                item.Image = File.ReadAllBytes(control.openFileDialog1.FileName);
        }
        ;
        flowLayoutPanel1.Controls.Add(control);
    }
}
```

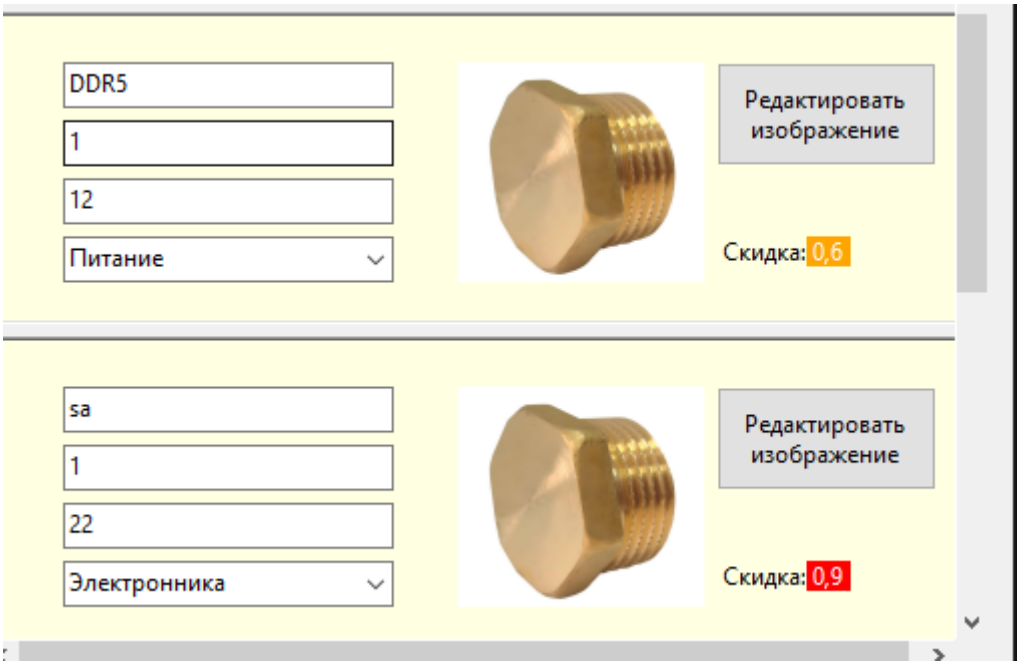
в переменной image находятся байты для заглушки

```
private byte[] image;
LazarevTaPrepContext context;
private BindingList<Product> list;

Ссылок: 1
public Form1(User u)
{
    InitializeComponent();
}

Ссылок: 0
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);
    image = File.ReadAllBytes("stub.png"); //Получаем изображение

    context = new();

    context.Products.Include(x => x.Category).Load();
    context.Categories.Load();
    context.Database.EnsureCreated();
    bindingSource1.DataSource = list = context.Products.Local.ToBindingList();
    categoryBindingSource.DataSource = context.Categories.Local.ToBindingList();
    Populate();
}
```

Изменение цветов

Изменение цвета происходит через свойство BackColor

```
Ссылок: 4
private void Populate()
{
    flowLayoutPanel1.Controls.Clear();
    foreach (Product item in bindingSource1)
    {
        var control = new MyControl();
        control.productBindingSource.DataSource = item;
        control.categoryBindingSource.DataSource = categoryBindingSource;
        if (item.Image == null) //изображения нет? Вставляем заглушку!
            item.Image = image;

        //У элементов управление есть свойство BackColor, Пользуйтесь им
        if (item.Discount > 0)
        {
            control.proc.ForeColor = Color.White;
            control.proc.BackColor = Color.GreenYellow;
        }
        if (item.Discount > 0.5)
            control.proc.BackColor = Color.Orange;
        if (item.Discount > 0.8)
            control.proc.BackColor = Color.Red;

        if(item.Discount == 0)
        {
            control.proc.Visible = false;
            control.label1.Visible = false;
        }
```



Обработка непредвиденных ошибок

```
/// </summary>
[STAThread]
Ссылок: 0
static void Main()
{
    // To customize application configuration such as set high DPI setti
    // see https://aka.ms/applicationconfiguration.
    AppDomain.CurrentDomain.UnhandledException += (o, e) =>
    {
        MessageBox.Show("Непредвиденная ошибка:\n\n" +
            (e.ExceptionObject as Exception).Message);
    };

    ApplicationConfiguration.Initialize();
    Application.Run(new Auth());
```

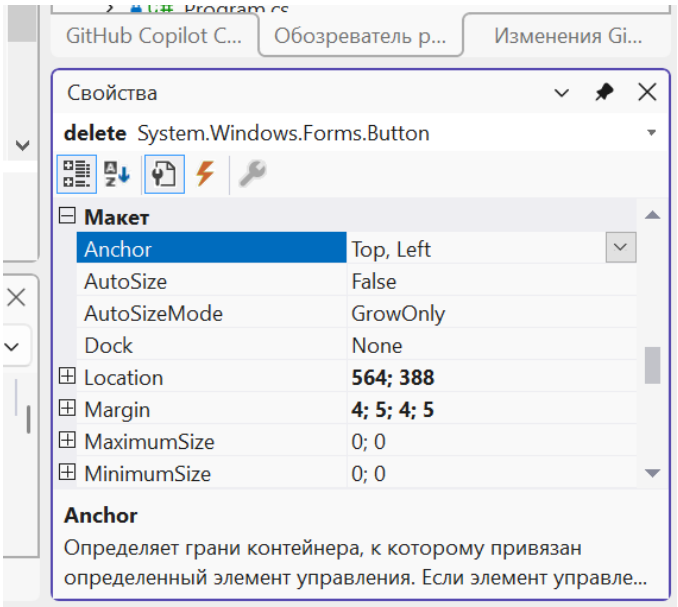
Якори элементов

Чтобы закреплять элементы по сторонам, воспользуйтесь свойством **Anchor**.

Можно закрепить элементы по верху, низу, левой и правой стороне.

Если закрепить слева и справа, то по увеличении окна по горизонтали, элементы будут не двигаться или оставаться на месте, а растягивать по горизонтали, тоже самое касается вертикальной оси. А если закрепить по всем сторонам, то элемент будет уменьшаться или увеличиваться согласно изменениям границ окна.

Если отметить только нижнюю или верхнюю, то элемент будет двигаться, если размер границ будет изменяться, то есть двигаться вместе с ними



Свойство **Dock** позволяет заполнят элементам заполнять одну определенную сторону или все пространство.

Чтобы Зафиксировать минимальный и максимальный рамер, воспользуйтесь свойствами MinimumSize или MaximumSize.

Если значения обеих свойств будут совпадать, то окна будет фиксированный, неизменяемый размер