

**Krzysztof Bieniasz**

## **Sprawozdanie z laboratorium nr 2 z przedmiotu Bazy Danych: .Net, Entity Framework**

### **1. Stworzenie dodatkowych klas.**

Poza klasami napisanymi na zajęciach, zdecydowałem się utworzyć dwie dodatkowe: klasę Order oraz klasę OrderDetails.

Kod klasy Customer:

```
public class Customer
{
    [Key]
    public string CompanyName { get; set; }
    public string Descriptoin { get; set; }
    public string Password { get; set; }
    public List<Order> Orders { get; set; }
}
```

Kod klasy Category:

```
public class Category
{
    public int CategoryID { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public List<Product> Products { get; set; }
}
```

Kod klasy Product:

```
public class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
    public int CategoryID { get; set; }
    [Column("UnitPrice", TypeName = "money")]
    public decimal Unitprice { get; set; }
    public List<OrderDetails> OrderDetails { get; set; }
}
```

Kod klasy Order:

```
public class Order
{
    public int OrderID { get; set; }
    [ForeignKey("Customer")]
    public string CompanyName { get; set; }
    public Customer Customer { get; set; }
    public List<OrderDetails> orderDetails { get; set; }
    public Boolean paid { get; set; }
}
```

Użyłem tutaj adnotacji ForeignKey, bo właściwość mogła sobie nie pozwolić zadziałać konwencji z utworzeniem zależności jeden do wielu między encją Orders a encją Customers.

Kod klasy OrderDetails:

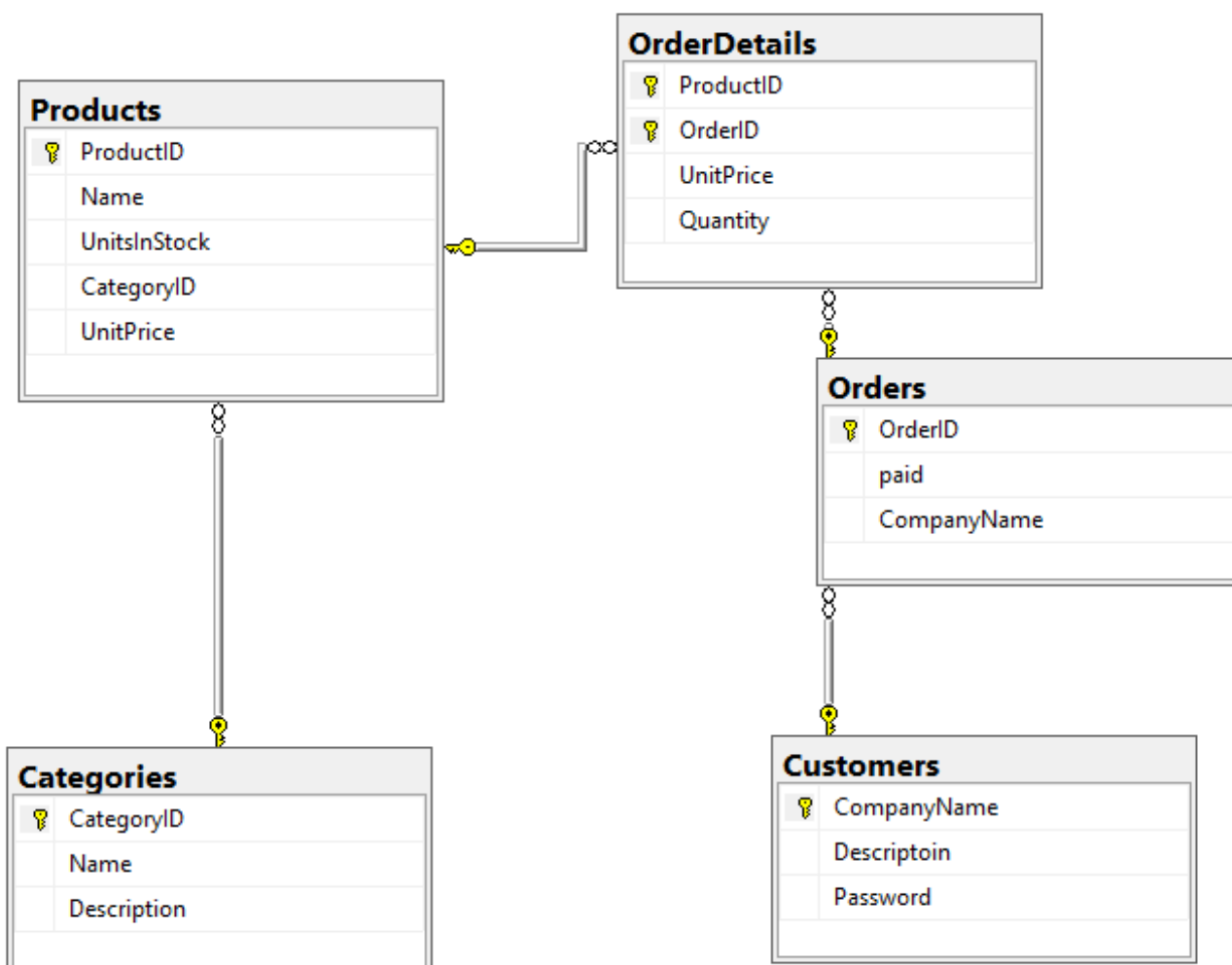
```
public class OrderDetails
{
    [Key]
    [Column(Order=1)]
    public int ProductID { get; set; }
    [Key]
    [Column(Order = 2)]
    public int OrderID { get; set; }
    public Product product { get; set; }
    public Order order { get; set; }
    [Column("UnitPrice", TypeName = "money")]
    public decimal Unitprice { get; set; }
    [Range(0.0, Double.MaxValue)]
    public int Quantity { get; set; }
}
```

Utworzyłem klucz kompozytowy, na który złożyły się ProductID oraz OrderID. Adnotacja nad polem Quantity zapewnia, iż to pole nie będzie miało ujemnych wartości. Wartość zero dopuszczam z powodu logiki biznesowej aplikacji.

W powyższych klasach w celu stworzenia relacji między encjami umieszczałem property 'wskazujący' na inny obiekt lub kolekcja obiektów w zależności od krotności relacji.

## 2. Diagram bazy danych

Na podstawie napisanych klas diagram bazy danych wygenerował się w następujący sposób:



## 3. Aplikacja oparta na formularzach.

### a) formularz startowy

Po uruchomieniu aplikacji wyświetla się formularz startowy. Aby móc dokonać zakupu użytkownik powinien wpisać poprawnie swój login oraz hasło. W przypadku braku wpisania poprawnego hasła lub loginu wyświetla się komunikat o błędach. Użytkownik nie mający konta w systemie może się zarejestrować klikając przycisk „sign up”. Dostępny jest też przycisk „admin mode”, który pozwala dostać się do trybu aplikacji, w której można swobodnie modyfikować dane. Domyślnie jest on dostępny tylko dla administratora.

Kod formularza StartForm:

```
public partial class StartForm : Form
{
    ProdContext context = new ProdContext();
    public StartForm()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        SigUpForm sigUpForm = new SigUpForm();
        sigUpForm.ShowDialog();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        AdminForm adminForm = new AdminForm();
        adminForm.ShowDialog();
    }

    private void button3_Click(object sender, EventArgs e)
    {
        string login = textBox1.Text;
        string password = textBox2.Text;

        Customer user = (from c in context.Customers
                        where c.CompanyName == login && c.Password == password
                        select c)
                        .DefaultIfEmpty(null)
                        .First();

        if (user == null)
        {
            MessageBox.Show("Please check if You put right login and password.");
        }
        else
        {
            ClientForm clientForm = new ClientForm(user);
        }
    }
}
```

```

        clientForm.ShowDialog();
    }
}
}

```

Funkcje `button*` odpowiadają za reakcję na zdarzenia naciśnięcia odpowiednich przycisków. W przypadku obsługi przycisku „sign in” musimy sprawdzić czy wprowadzone dane są zgodne z danymi w bazie danych. W przypadku, gdy w bazie danych nie ma rekordu o wprowadzonych parametrach wyświetla się okienko informujące, o tym żeby sprawdzić czy wprowadziło się poprawne dane.

## b) formularz zakładania konta

Kod formularza SignUpForm:

```

public partial class SigUpForm : Form
{
    ProdContext context = new ProdContext();

    public SigUpForm()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        string companyName = textBox1.Text;
        string password = textBox2.Text;
        string passwordControl = textBox3.Text;
        string description = richTextBox1.Text;
        if(!password.Equals(passwordControl))
        {
            return;
            /*
             ewentualnie okienko z komunikatem
            */
        }
        Customer customer = new Customer
        {
            CompanyName = companyName, Description = description,
            Password = password };
        ProdContext context = new ProdContext();
        context.Customers.Add(customer);
    }
}

```

```

context.SaveChanges();
ClientForm clientForm = new ClientForm(customer);
clientForm.ShowDialog();

    }
}

```

Klasa zawiera w sobie obsługę przycisku potwierdzenia wprowadzenia danych. Pobieram wartości pól tekstowych i na ich podstawie tworzę nowy obiekt Customer, który dodaje do kontekstu. Po wywołaniu context.SaveChanges() klient zostaje dodany do bazy danych.

### c) formularz wyboru produktów

W tym formularzu klient może wybrać z jakiej kategorii chce zobaczyć produkty wybierając odpowiednią nazwę kategorii z comboboxa i wciskając przycisk search. W przypadku chęci zobaczenia wszystkich produktów może kliknąć przycisk „show all products”. Bo dodaniu zaznaczonego produktu do koszyka aktualizuje się pole pokazujące aktualną wartość koszyka. Ponadto użytkownik klikając przycisk „check Your previous orders” może sprawdzić swoje poprzednie zamówienia. Ponadto w dolnej części formularza znajduje się Warto skomentować, iż użytkownik nie ma możliwości edycji danych w formularzu, może on jednak zaznaczać wiersze oraz sortować produkty według ceny, nazwy, id oraz dostępnej ilości w magazynie.

The screenshot shows a Windows application window titled "ClientForm". The interface has a light blue background. At the top left, it says "Welcome Johan !". To the right of this are two buttons: "check Your previous orders" and "go to bucket and finalize order". Below the welcome message is a "select category" label above a dropdown menu currently showing "Basketball". To the right of the dropdown are three buttons: "search", "show all products", and "add to bucket". Further right, it says "Current bucket value: 0.00". At the bottom, there is a table with 6 columns: ProductID, Name, UnitsInStock, CategoryID, and Unitprice. The first row is highlighted in blue.

	ProductID	Name	UnitsInStock	CategoryID	Unitprice
▶	13	Jordan shoes	5	13	299,0000
	14	Spalding ball	10	13	99,0000
	15	LAL jersey	3	13	150,0000

Kod formularza ClientForm:

```
public partial class ClientForm : Form
{
    private Customer customer;
    private List<Product> bucket = new List<Product>();
    private decimal currentPrice = 0;
    ProdContext context = new ProdContext();

    public ClientForm(Customer customer)
    {
        InitializeComponent();
        this.customer = customer;

        var cat = from Category in context.Categories select Category;
        comboBox1.DataSource = cat.ToList();
        comboBox1.DisplayMember = "Name";
        comboBox1.ValueMember = "CategoryID";
        label2.Text = "Welcome " + this.customer.CompanyName + "!";
    }

    protected override void OnLoad(EventArgs e)
    {
        base.OnLoad(e);
        this.categoryBindingSource.DataSource =
            context.Categories.Local.ToBindingList();
        context.Categories.Load();
        this.orderDetailsBindingSource.DataSource =
            context.OrderDetails.Local.ToBindingList();
        context.OrderDetails.Load();
        this.ordersBindingSource.DataSource = context.Orders.Local.ToBindingList();
        context.Orders.Load();
        this.productsBindingSource.DataSource = (from p in context.Products where
                                                p.UnitsInStock > 0
                                                select p).ToList();
        context.Products.Load();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        Category cat= (Category) comboBox1.SelectedItem;
        if (cat != null)
        {
            this.productsBindingSource.DataSource = (from p in context.Products
                                                    where p.CategoryID ==
                                                            cat.CategoryID
                                                    select p).ToList();
        }
    }

    // dodanie produktu do koszyka
    private void button2_Click(object sender, EventArgs e)
    {
        Product product = (Product )productsDataGridView.SelectedRows[0].DataBoundItem;
        bucket.Add(product);
        currentPrice += product.Unitprice;
        label3.Text = "Current bucket value: " + currentPrice;
    }
}
```

```
// wyświetlenie wszystkich produktów
private void button3_Click(object sender, EventArgs e)
{
    this.productsBindingSource.DataSource = context.Products.Local.ToBindingList();
}

//przejsie do poprzednich zamówień
private void button4_Click(object sender, EventArgs e)
{
    PrevoiusOrdersForm prevoiusOrdersForm = new PrevoiusOrdersForm(this.customer);
    prevoiusOrdersForm.ShowDialog();
}

//przejsie do finalizacji zamówienia
private void button5_Click(object sender, EventArgs e)
{
    BucketForm bucketForm = new BucketForm(this.customer, this.bucket);
    bucket = new List<Product>();
    currentPrice = 0;
    bucketForm.ShowDialog();
}
}
```

Warto skomentować, iż opcje combobox-a są tworzone dynamicznie przy tworzeniu całego okienko poprzez pobranie z kontekstu wszystkich dostępnych w danym momencie kategorii i ustawieniu pole „DisplayMember” na „Name”, czyli nazwę produktu. Podczas przypisywania źródła danych dla źródła bindowania produktów, sprawdzam warunek czy choć jeden produkt jest dostępny. W przypadku kliknięcia przycisku finalizacji zamówienia tworzę nowy formularz, do którego przekazuję klienta oraz listę wybranych przezeń produktów (bucket).

#### d) formularz pokazujący poprzednie zamówienia

W okienku wyświetlają się dwa formularze, jeden zawierający dane pochodzące z tabeli Orders, a drugi dane z tabeli OrderDetails, oczywiście powiązane tylko z danym użytkownikiem. Po wybraniu rzędu z górnego formularza i kliknięciu na OrderID, w dolnym formularzu pokazują się tylko szczegóły dotyczące danego zamówienia. Należało zaznaczyć opcje zaznaczania całego rzędu.

OrderID	paid
1	<input checked="" type="checkbox"/>
2	<input type="checkbox"/>
3	<input type="checkbox"/>
4	<input type="checkbox"/>
5	<input type="checkbox"/>

ProductID	OrderID	Unitprice	Quantity
14	1	99,0000	2
15	1	150,0000	2



Kod formularza PreviousOrdersForm;

```
public partial class PrevoiusOrdersForm : Form
{
    private Customer customer;
    ProdContext context = new ProdContext();

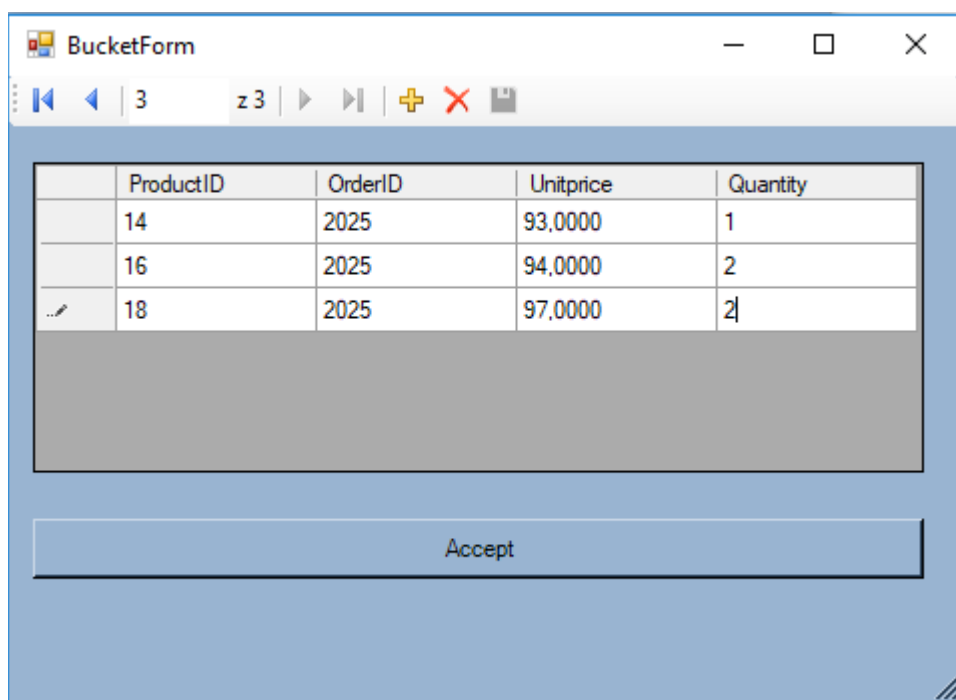
    public PrevoiusOrdersForm(Customer customer)
    {
        InitializeComponent();
        this.customer = customer;
    }

    protected override void OnLoad(EventArgs e)
    {
        base.OnLoad(e);
        context = new ProdContext();
        this.ordersBindingSource.DataSource = new BindingList<Order>
            (context.Orders.Where(o => o.CompanyName ==
                customer.CompanyName).ToList());
        context.Orders.Load();
        this.orderDetailsBindingSource.DataSource =
            (from od in context.OrderDetails
             join o in context.Orders
             on od.OrderID equals o.OrderID
             where o.CompanyName == customer.CompanyName select od).ToList();
        context.OrderDetails.Load();
    }

    private void ordersDataGridView_CellContentClick(object sender,
        DataGridViewCellEventArgs e)
    {
        Order order = (Order)this.ordersDataGridView.SelectedRows[0].DataBoundItem;
        if (order != null)
        {
            this.orderDetailsBindingSource.DataSource = new BindingList<OrderDetails>
                (context.OrderDetails.Where(od => od.OrderID ==
                    order.OrderID).ToList());
        }
    }
}
```

### e) formularz finalizacji zakupów

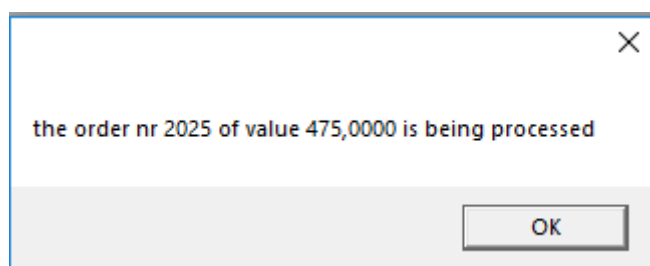
Podczas ładowanie tego formularza są tworzone jest tworzony obiekt Order oraz obiekty OrderDetails, początkowo we wszystkich utworzonych szczegółach zamówienia ilość zamówionych sztuk wynosi 1 oraz dodaje je do kontekstu. Ustawiłem możliwość edycji kolumny Quantity, dzięki czemu użytkownik może wybrać ilość produktów, którą chce kupić. Po kliknięciu przycisku „accept” sprawdzam w bazie danych czy jest dostępna odpowiednia ilość każdego produktu. W przypadku braku takiej ilości, pojawia się komunikat w okienku o braku danego produktu w magazynie. W przypadku dostępności wszystkich produktów zapisywane są zmiany w ilości zamówienia oraz zmieniana jest wartość UnitsInStock w odpowiednim produkcie.



	ProductID	OrderID	Unitprice	Quantity
	14	2025	93,0000	1
	16	2025	94,0000	2
✎	18	2025	97,0000	2

Accept

Okienko informujące o pozytywnym zakończeniu składania zamówienia:



the order nr 2025 of value 475,0000 is being processed

OK

Przykład pokazujący, iż po złożeniu zamówienia zmniejszyła się ilość dostępnych produktów. Można zwrócić uwagę na przykład na produkt o id=18 i nazwie Adidas 11pro. W pierwszym okienku jest dostępnych 10 sztuk, natomiast w drugim 8. A więc zmieniło się o tyle ile użytkownik wprowadził na okienku przedstawiającym BucketForm.

ClientForm

Welcome Johan !

check Your previous orders

go to bucket and finalize order

select category

Basketball

search

show all products

Current bucket value: 284,0000

add to bucket

	ProductID	Name	UnitsInStock	CategoryID	Unitprice
	13	Jordan shoes	5	13	297,0000
▶	14	Spalding ball	10	13	93,0000
	15	LAL jersey	3	13	150,0000
	16	Nike Ordem V ball	10	14	94,0000
	17	Nike control360	3	14	149,0000
	18	Adidas 11pro	10	14	97,0000
	19	Manchester Unit...	3	14	299,0000

ClientForm

Welcome Johan !

check Your previous orders

go to bucket and finalize order

select category

Basketball

search

show all products

Current bucket value: 0.00

add to bucket

	ProductID	Name	UnitsInStock	CategoryID	Unitprice
▶	13	Jordan shoes	5	13	297,0000
	14	Spalding ball	9	13	93,0000
	15	LAL jersey	3	13	150,0000
	16	Nike Ordem V ball	8	14	94,0000
	17	Nike control360	3	14	149,0000
	18	Adidas 11pro	8	14	97,0000
	19	Manchester Unit...	3	14	299,0000

Kod klasy BucketForm:

```
public partial class BucketForm : Form
{
    private Customer customer;
    private List<Product> bucket = new List<Product>();
    ProdContext context = new ProdContext();
    private List<OrderDetails> orderDetailsList = new List<OrderDetails>();
    private Order order;
    private int globalOrderID = 0;

    public BucketForm(Customer customer, List<Product> bucket)
    {
        InitializeComponent();
        this.customer = customer;
        this.bucket = bucket;
        this.order = new Order();
        order.CompanyName = customer.CompanyName;
        context.Orders.Add(order);
        context.SaveChanges();
        this.orderDetailsBindingSource.DataSource =
            context.OrderDetails.Local.ToBindingList();
        context.OrderDetails.Load();

        var orders_ids = from orders in context.Orders
                        orderby orders.OrderID descending
                        select orders.OrderID;

        int max = 1;
        foreach (var item in orders_ids)
        {
            if((int) item > max)
            {
                max = (int)item;
            }
        }
        int order_id = max;
        globalOrderID = order_id;

        foreach(Product product in bucket)
        {
            OrderDetails orderDetails = new OrderDetails
            {
                ProductID = product.ProductID,
                OrderID = order_id,
                Quantity = 1,
                Unitprice = product.Unitprice
            };

            context.OrderDetails.Add(orderDetails);
            context.SaveChanges();
        }

        this.orderDetailsBindingSource.DataSource = (from od in context.OrderDetails
                                                    where od.OrderID == order_id
                                                    select od).ToList();
        this.orderDetailsDataGridView.Refresh();
    }

    private void button1_Click(object sender, EventArgs e)
    {

```

```

context.SaveChanges();
var finalOrderDetails = from od in context.OrderDetails
                        where od.OrderID == globalOrderID
                        select od;
foreach (OrderDetails details in finalOrderDetails)
{
    Product result = context.Products.SingleOrDefault
                    (p => p.ProductID == details.ProductID);

    if (result != null)
    {
        if (result.UnitsInStock - details.Quantity < 0)
        {
            MessageBox.Show("There is no this amount of product with ID " +
                            result.ProductID + " in stock");
            return;
        }
    }
}

decimal total_price = 0;
foreach (OrderDetails details in finalOrderDetails)
{
    ProdContext prodContext2 = new ProdContext();
    Product result = prodContext2.Products.SingleOrDefault
                    (p => p.ProductID == details.ProductID);

    if (result != null)
    {
        result.UnitsInStock -= details.Quantity;
        total_price += details.Quantity * details.Unitprice;
        prodContext2.SaveChanges();
    }
}

MessageBox.Show("the order nr " + globalOrderID + " of value " + total_price + "
                is being processed");
this.Close();
}
}

```

f) tryb admina

Utworzyłem też okno, w którym można dowolnie modyfikować każdą z tabel. Oczywiście w normalnej aplikacji nie byłoby to wskazane, jednak tutaj było użyteczne chociażby do szybkiej modyfikacji danych.

CategoryID	Name	Description
13	Basketball	Wonderful game
14	Soccer	Beautiful game
15	Tennis	Hard game

```
public partial class AdminForm : Form
{
    ProdContext context = new ProdContext();

    public AdminForm()
    {
        InitializeComponent();
    }

    protected override void OnLoad(EventArgs e)
    {
        base.OnLoad(e);
        context = new ProdContext();
        this.categoryBindingSource.DataSource = context.Categories.Local.ToBindingList();
        context.Categories.Load();
        this.orderDetailsBindingSource.DataSource = context.OrderDetails.Local.ToBindingList();
        context.OrderDetails.Load();
        this.ordersBindingSource.DataSource = context.Orders.Local.ToBindingList();
        context.Orders.Load();
        this.customerBindingSource.DataSource = context.Customers.Local.ToBindingList();
        context.Customers.Load();
        this.productsBindingSource.DataSource = context.Products.Local.ToBindingList();
        context.Products.Load();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        if(MessageBox.Show("Are You sure to save changes", "!",
        MessageBoxButtons.YesNo)==DialogResult.Yes)
        {

```

```

        context.SaveChanges();
        this.categoryDataGridView.Refresh();
    }
}

private void button2_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Are You sure to save changes", "!", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
    {
        context.SaveChanges();
        ordersDataGridView.Refresh();
    }
}

private void button3_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Are You sure to save changes", "!", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
    {
        context.SaveChanges();
        customerDataGridView.Refresh();
    }
}

private void button4_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Are You sure to save changes", "!", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
    {
        context.SaveChanges();
        productsDataGridView.Refresh();
    }
}

private void button5_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Are You sure to save changes", "!", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
    {
        context.SaveChanges();
        orderDetailsDataGridView.Refresh();
    }
}
}

```

Okienko jest podzielone na 5 zakładek, z których każda składa się z formularza odpowiadającego jednej tabeli i przyciskowi „save”, po którego wciśnięciu zapisywany jest aktualny stan oraz odświeżany widok.