Piotr Wróbel

Entity Framework – sprawozdanie

Spis treści

1. Wstęp

Wykonywanie ćwiczenia na laboratorium zakończono na etapie osadzania tabeli Category na formularzu CategoryForm, ustawienia pola CategoryID jako tylko do odczytu i wykonaniu programu (punkt IVg instrukcji), stąd opis w niniejszym sprawozdaniu rozpoczęto od następnego w kolejności punktu.

2. Bindowanie danych do komórek

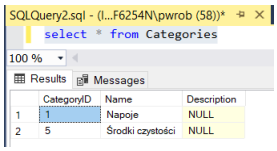
Po uruchomieniu programu na etapie punktu IVf instrukcji nie zaobserwowano ładowania danych z tabeli do osadzonej na formularzu kontrolki. Aby to zmienić, zdefiniowano metodę Load formularza CategoryForm w następujący sposób:

```
1 private void CategoryForm_Load(object sender, EventArgs e)
2 {
3   base.OnLoad(e);
4
5   prodContext = new ProdContext();
6   prodContext.Categories.Load();
7   this.categoryBindingSource.DataSource = prodContext.Categories.Local.ToBindingList();
8 }
```

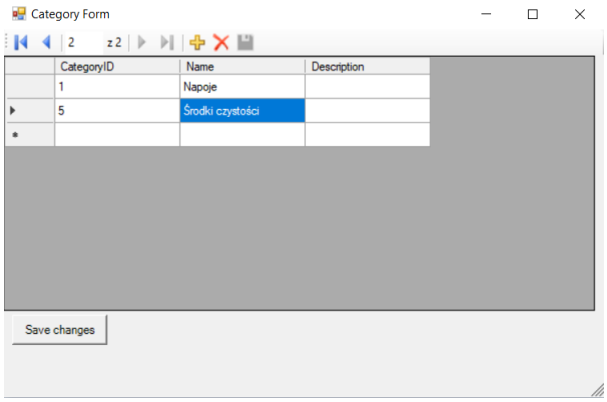
Po ponownym uruchomieniu programu po modyfikacji zaobserwowano poprawne ładowanie danych do kontrolki. Następnie dodano do formularza przycisk służący do zapisywania zmian i przypisano zapisywanie danych do bazy do zdarzenia obsługującego jego kliknięcie:

```
1 private void saveButton_Click(object sender, EventArgs e)
2 {
3   prodContext.SaveChanges();
4
5   this.categoryDataGridView.Refresh();
6 }
```

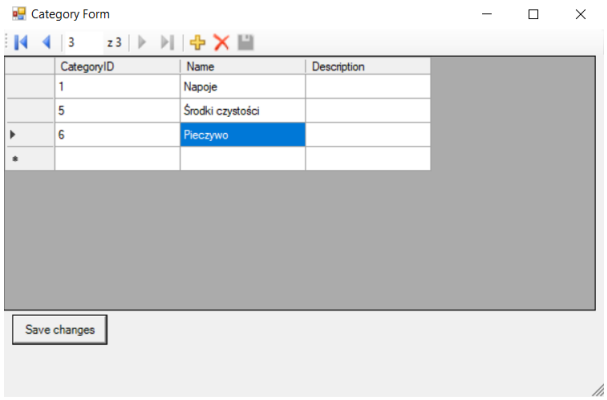
Poprawność działania zweryfikowano wywołując polecenie select na tabeli Categories przed i po zapisaniu zmian w formularzu (Rysunki 1-4).



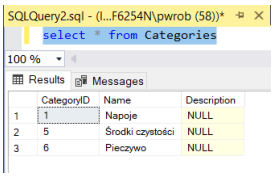
Rysunek 1. Stan tabeli Categories przed modyfikacją



Rysunek 2. Formularz CategoryForm przed modyfikacją



Rysunek 3. Formularz CategoryForm po modyfikacji



Rysunek 4. Po wciśnięciu przycisku Save changes zaobserwowano zapisanie zmian do bazy

W analogiczny sposób jak dla tabeli Categories, do formularza dodano kontrolkę wyświetlającą zawartość tabeli Products. W celu zapewnienia wyświetlania tylko produktów z aktualnie zaznaczonej kategorii, a także poprawnego działania zapisywania nowych danych do bazy (dla przefiltrowanych danych kliknięcie przycisku Save changes nie zadziałałoby), wprowadzono na formularz przycisk przełączający między trybem przeglądania i zapisywania danych (aktualny tryb jest rozpoznawany dzięki globalnej zmiennej boolowskiej isSavingMode). Dodatkowo zmienna globalna currentCategoryID przechowuje ID aktualnie zaznaczonej kategorii. Do zdarzeń CellContentClick i RowEnter kontrolki categoryDataGridView przypisano metodę:

```
private void handleCategorySelection(object sender, DataGridViewCellEventArgs e)
{
  int selectedRow = e.RowIndex;
  Category selectedCategory = (Category)categoryDataGridView.Rows[selectedRow].DataBoundItem;

  if (selectedCategory != null)
  {
    currentCategoryID = selectedCategory.CategoryID;
    if (!this.isSavingMode)
    {
      // query based syntax
      this.productBindingSource.DataSource = (from prod in prodContext.Products
                        where prod.CategoryID == currentCategoryID
                                              select prod).ToList();

      // method based syntax
      //this.productBindingSource.DataSource =
            //prodContext.Products.Where(prod => prod.CategoryID == currentCategoryID)
            //.ToList();
    }
  }
}
```

A do zdarzenia obsługującego kliknięcie przycisku switchButton:

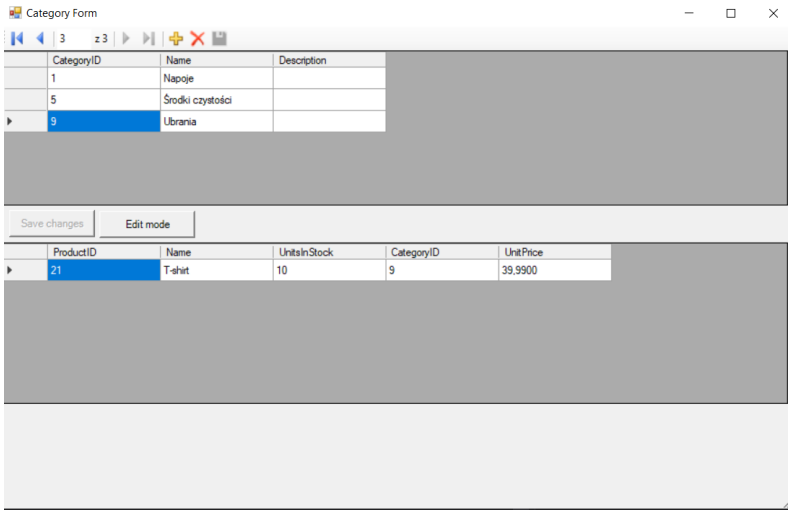
```
private void switchModeButton_Click(object sender, EventArgs e)
{
  if(isSavingMode)
  {
    this.switchModeButton.Text = "Edit mode";

    this.productBindingSource.DataSource =
      prodContext.Products.Where(prod => prod.CategoryID == currentCategoryID).ToList();
  }
  else
  {
    this.switchModeButton.Text = "Read mode";

    productBindingSource.DataSource = prodContext.Products.Local.ToBindingList();
  }

  this.saveButton.Enabled = !this.saveButton.Enabled;

  this.productDataGridView.AllowUserToAddRows =
    !this.productDataGridView.AllowUserToAddRows;
  this.productDataGridView.AllowUserToDeleteRows =
    !this.productDataGridView.AllowUserToDeleteRows;
```

```
23
24      this.categoryDataGridView.AllowUserToAddRows =
25          !this.categoryDataGridView.AllowUserToAddRows;
26      this.categoryDataGridView.AllowUserToDeleteRows =
27          !this.categoryDataGridView.AllowUserToDeleteRows;
28
29      isSavingMode = !isSavingMode;
30  }
```

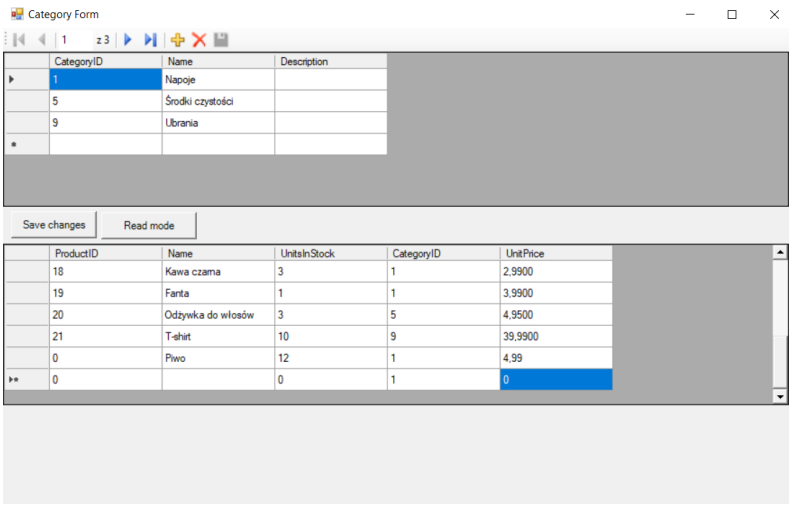
W celu zapewnienia poprawności dodawanych nowych danych, zaimplementowano metodę DefaultValuesNeeded kontrolki productDataGridView:

```
1  private void productDataGridView_DefaultValuesNeeded(object sender, DataGridViewRowEventArgs e)
2  {
3      e.Row.Cells["prodDGVCategoryID"].Value = currentCategoryID;
4  }
```

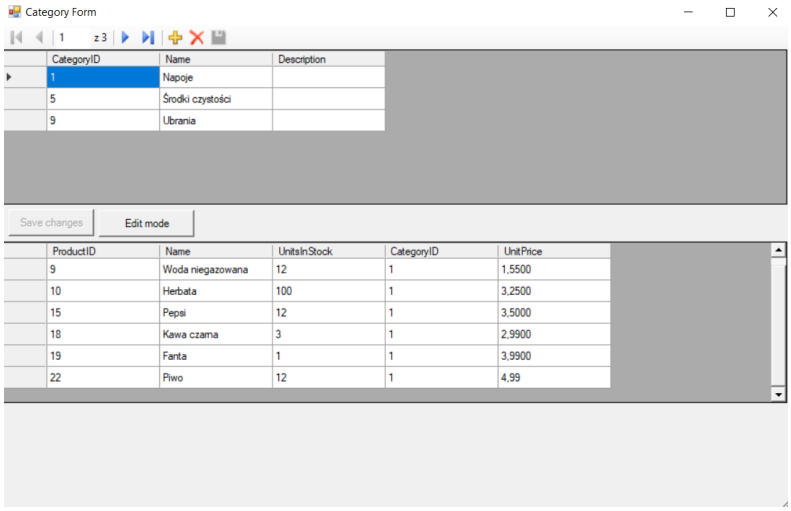
Działanie programu po dodaniu modyfikacji przedstawiono na screenach (Rysunek 5-7).



Rysunek 5. Tryb przeglądania danych



Rysunek 6. Tryb modyfikacji danych



Rysunek 7. Po zapisaniu nowych danych i kliknięciu odpowiedniej kategorii

3. Method syntax vs query syntax

W konsolowej części aplikacji zaimplementowano wypisywanie wszystkich kategorii korzystając z method based syntax bez wymuszania automatycznej ewaluacji zapytania:

```
1  // punkt V
2  // deferred query execution
3  using(ProdContext prodContext = new ProdContext())
4  {
5    IQueryable<String> categoryNames = prodContext.Categories.Select(c => c.Name);
6
7    Console.WriteLine("Category names:");
8
9    foreach(String categoryName in categoryNames)
10   {
11     Console.WriteLine(categoryName);
12   }
13   Console.ReadLine();
14 }
```

Ustawiając odpowiednie breakpointy zaobserwowano w SQL Server Profilerze, że zapytanie jest wykonywane dopiero w momencie wejścia do pętli foreach.

Następnie zmodyfikowano kod, tak aby zapytanie było wywoływane natychmiastowo:

```
1  // immediate query execution
2
3  using (ProdContext prodContext = new ProdContext())
4  {
5    List<String> categoryNames = prodContext.Categories.Select(c => c.Name).ToList();
6
7    Console.WriteLine("Category names:");
8
9    foreach (String categoryName in categoryNames)
10   {
11     Console.WriteLine(categoryName);
12   }
13   Console.ReadLine();
14 }
```

Tym razem zaobserwowano, że zapytanie select jest wykonywane już w momencie definicji zapytania, jeszcze przed wejściem do pętli.

Zdefiniowano funkcje wypisujące kategorie i produkty, w wersji query based syntax:

```
// query based syntax - categories and products (join, lazy loading)

using (ProdContext prodContext = new ProdContext())
{
  IQueryable<Category> categories = prodContext.Categories;
  IQueryable<Product> products = prodContext.Products;

  var query =
    from category in categories
    join product in products
    on category.CategoryID equals product.CategoryID
    select new
    {
      CategoryID = category.CategoryID,
      CategoryName = category.Name,
      CategoryDescription = category.Description,
      ProductID = product.ProductID,
      ProductName = product.Name,
      UnitsInStock = product.UnitsInStock,
      UnitPrice = product.UnitPrice
    };

  foreach (var category in query)
  {
    Console.WriteLine("{0}\t{1}\t{2}\t\t\t{3}\t{4}\t{5}\t{6}",
          category.CategoryID,
            category.CategoryName,
            category.CategoryDescription,
            category.ProductID,
            category.ProductName,
            category.UnitsInStock,
            category.UnitPrice
    );
  }
  Console.Read();
}
```

i w wersji method based syntax:

```
// method based syntax (navigation property, eager loading)

using (ProdContext prodContext = new ProdContext())
{
  var categoryQuery = prodContext.Categories.Include("Product").Select(cat => new
  {
    CategoryID = cat.CategoryID,
    CategoryName = cat.Name,
    CategoryDescription = cat.Description,
    CategoryProducts = cat.Products
  });

  foreach (var category in categoryQuery)
  {
    Console.WriteLine("{0}\t{1}\t{2}",
      category.CategoryID,
      category.CategoryName,
      category.CategoryDescription
    );
```

```
    foreach (Product product in category.CategoryProducts)
    {
      Console.WriteLine("\t\t\t\t{0}\t{1}\t{2}\t{3}",
        product.ProductID,
        product.Name,
        product.UnitsInStock,
        product.UnitPrice);
    }
  }
  Console.Read();
}
```

Ponadto zaimplementowano funkcję zliczającą ilość produktów w poszczególnych kategoriach, w wersji query based syntax:

```
// products number
// query based syntax (lazy loading)

using (ProdContext prodContext = new ProdContext())
{
  var query = from category in prodContext.Categories
    select new
    {
      CategoryID = category.CategoryID,
      Name = category.Name,
      Description = category.Description,
      ProdCount = category.Products.Count()
    };

  foreach (var category in query)
  {
    Console.WriteLine("{0}\t{1}\t{2}\t{3}",
      category.CategoryID,
      category.Name,
      category.Description,
      category.ProdCount);
  }
  Console.Read();
}
```

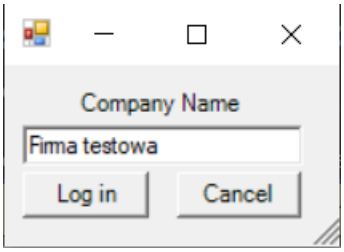
i w wersji method based syntax:

```
// method based syntax (navigation property, lazy loading)

using (ProdContext prodContext = new ProdContext())
{
  var categoryQuery = prodContext.Categories.Include("Product").Select(cat => new
  {
    CategoryID = cat.CategoryID,
    CategoryName = cat.Name,
    CategoryDescription = cat.Description,
    CategoryProducts = cat.Products
  });

  foreach (var category in categoryQuery)
  {
    int productsCount = 0;
    foreach (Product product in category.CategoryProducts)
    {
      productsCount++;
    }

```

```
21      Console.WriteLine("{0}\t{1}\t{2}\t{3}",
22        category.CategoryID,
23        category.CategoryName,
24        category.CategoryDescription,
25        productsCount);
26    }
27
28    Console.Read();
29  }
```

4. Zadanie domowe

Aby możliwe było złożenie zamówienia przez konkretnego użytkownika, dodano nowy formularz służący do logowania (podania nazwy użytkownika znajdującej się w bazie) – Rysunek 8.



Rysunek 8. Formularz logowania

Jeżeli podano nazwę znajdującą się w tabeli Customers, to otwierany jest formularz zamówienia (odpowiednio rozbudowany w stosunku do punktu IV), kod odpowiedzialny za sprawdzanie zgodności nazwy został przypisany do obsługi kliknięcia przycisku Log in. Ponadto do formularza CategoryForm dodano atrybut będący nazwą zalogowanego użytkownika przekazywany w konstruktorze.

```
1  private void loginButton_Click(object sender, EventArgs e)
2  {
3    String companyName = this.companyNameTextBox.Text;
4    bool companyExists = prodContext.Customers.Any(c => c.CompanyName == companyName);
5
6    if(!companyExists)
7    {
8      MessageBox.Show("Non-valid company name", "Logging error", MessageBoxButtons.OK);
9    }
10   else
11   {
12     CategoryForm categoryForm = new CategoryForm(companyName);
13     categoryForm.ShowDialog();
14     this.Close();
15   }
16 }
```

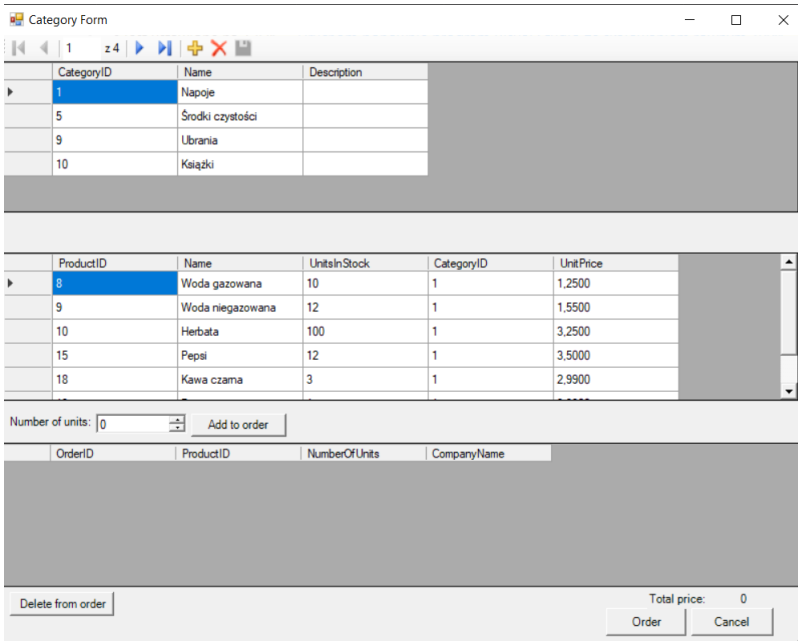
W aplikacji istnieje specjalny użytkownik Admin, który ma możliwość modyfikacji pól dotyczących produktów i kategorii. Kwestia, czy bieżącemu użytkownikowi można przypisać takie uprawnienia, jest sprawdzana przy ładowaniu CategoryForm:

```
if (companyName == "Admin")
{
  this.saveButton.Enabled = false;
  this.switchModeButton.Text = "Edit mode";
}
else
{
  this.saveButton.Visible = false;
  this.switchModeButton.Visible = false;
}
```

Do obsługi zamówień, dodano nową klasę Order:

```
public class Order
{
  public int OrderID { get; set; }
  [ForeignKey("Product")]
  public int ProductID { get; set; }

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

  [ForeignKey("Customer")]
  public String CompanyName { get; set; }
  public virtual Customer Customer { get; set; }
  public virtual Product Product { get; set; }
}
```

Zmodyfikowano formularz CategoryForm, dodając do niego pola służące do dodawania i usuwania pozycji do/z zamówienia. Ponadto umieszczono etykietę zawierającą całkowitą wartość zamówienia, a także przyciski Order – zapisujący dane zamówienia do bazy i Cancel – czyszczący wszystkie dane o zamówieniu z formularza. Nowy wygląd formularza przedstawia Rysunek 9.



Rysunek 9. Formularz zamawiania

O ile w bazie na etapie ładowania formularza istnieją jakieś wpisy w tabeli Order, to przy jego ładowaniu jest sumowana wartość wszystkich produktów z wykorzystaniem NavigationProperty:

```
this.orderBindingSource.DataSource = prodContext.Orders.Local.Where(ord =>
  ord.CompanyName == companyName).ToList();
this.totalPrice = 0;

foreach (Order order in (List<Order>) orderBindingSource.DataSource)
{
  Product currentProduct = order.Product;
  this.totalPrice += currentProduct.UnitPrice * order.NumberOfUnits;
}

this.totalPriceLabel.Text = Convert.ToString(this.totalPrice);
```

Przy dodawaniu pozycji do zamówienia sprawdzane jest czy podana ilość jest niezerowa (mniejsze wartości są wykluczone na etapie odpowiedniego ustawienia kontrolki) oraz czy nie przekracza ilości zadeklarowanej w tabeli Products. W przeciwnym wypadku tworzona jest nowa pozycja zamówienia, dodawana do kontekstu oraz pomniejszana jest dostępna ilość w tabeli Products, a także aktualizowana zawartość etykiety wyświetlającej cenę:

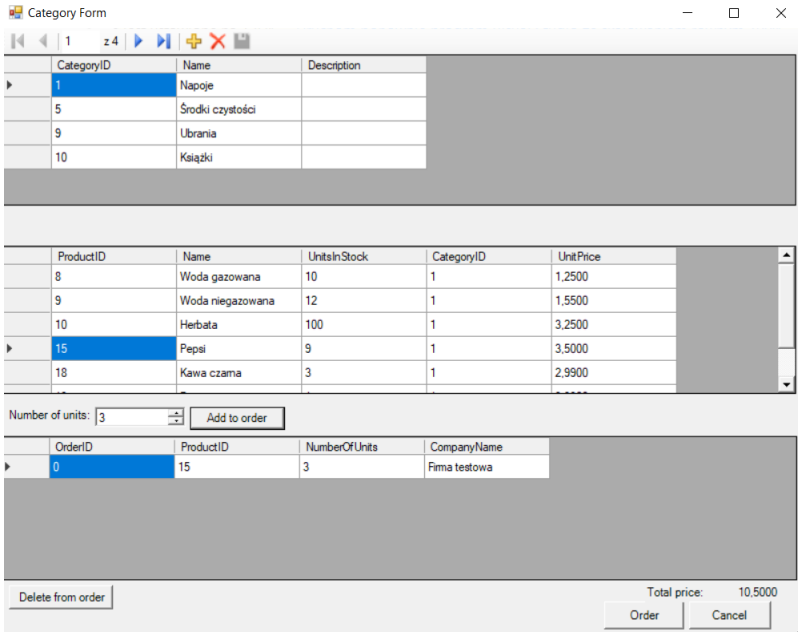
```
private void addToOrderButton_Click(object sender, EventArgs e)
{
    int selectedProductRowIndex = productDataGridView.SelectedCells[0].RowIndex;
    DataGridViewRow selectedProductRow = productDataGridView.Rows[selectedProductRowIndex];
    int maxUnits = Convert.ToInt32(selectedProductRow.Cells["prodDGVUnitsInStock"].Value);

    if(this.numberOfUnitsUpDown.Value == 0 || this.numberOfUnitsUpDown.Value > maxUnits)
    {
      MessageBox.Show("Incorrect unit number!", "Error", MessageBoxButtons.OK);
    }
    else
    {
    Order newOrder = new Order();
    newOrder.CompanyName = this.companyName;
    newOrder.NumberOfUnits = (int) this.numberOfUnitsUpDown.Value;
    newOrder.ProductID = Convert.ToInt32(selectedProductRow.Cells["prodDGVProductID"].Value);
    prodContext.Orders.Add(newOrder);

    orderBindingSource.DataSource = prodContext.Orders.Local.Where(ord =>
      ord.CompanyName == companyName).ToList();

    Product productToModify = prodContext.Products.First(prod =>
      prod.ProductID == newOrder.ProductID);
    productToModify.UnitsInStock -= newOrder.NumberOfUnits;
    productDataGridView.Refresh();

    this.totalPrice += productToModify.UnitPrice * newOrder.NumberOfUnits;
    this.totalPriceLabel.Text = Convert.ToString(this.totalPrice);
  }
}
```

Analogicznie zaimplementowano usuwanie pozycji z zamówienia:

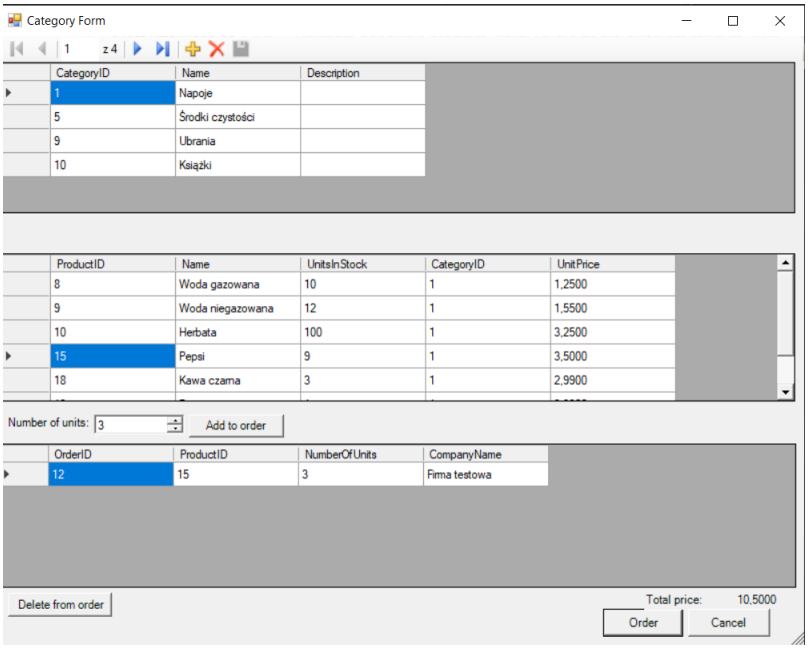
```
private void deleteFromOrder_Click(object sender, EventArgs e)
{
    int selectedOrderRowIndex = orderDataGridView.SelectedCells[0].RowIndex;
    DataGridViewRow selectedOrderRow = orderDataGridView.Rows[selectedOrderRowIndex];
    int productToChangeID = Convert.ToInt32(selectedOrderRow.Cells["orderDGVProductID"].Value);
```

```
6       int cancelledUnits = Convert.ToInt32(selectedOrderRow.Cells["orderDGVNumberOfUnits"].Value);
7       int cancelledOrderID = Convert.ToInt32(selectedOrderRow.Cells["orderDGVOrderID"].Value);
8
9       Order orderToRemove = prodContext.Orders.Local.First(ord =>
10        (ord.ProductID == productToChangeID)  &&
11        ((ord.ProductID == productToChangeID) &&
12        (ord.NumberOfUnits == cancelledUnits) &&
13        (String.Compare(ord.CompanyName, this.companyName) == 0)));
14
15      Product productToModify = prodContext.Products.First(prod => prod.ProductID == productToChangeID);
16      productToModify.UnitsInStock += cancelledUnits;
17      productDataGridView.Refresh();
18
19      this.totalPrice -= cancelledUnits * productToModify.UnitPrice;
20      this.totalPriceLabel.Text = Convert.ToString(this.totalPrice);
21
22      prodContext.Orders.Local.Remove(orderToRemove);
23      orderBindingSource.DataSource = prodContext.Orders.Local.Where(ord =>
24      ord.CompanyName == companyName).ToList();
25  }
```

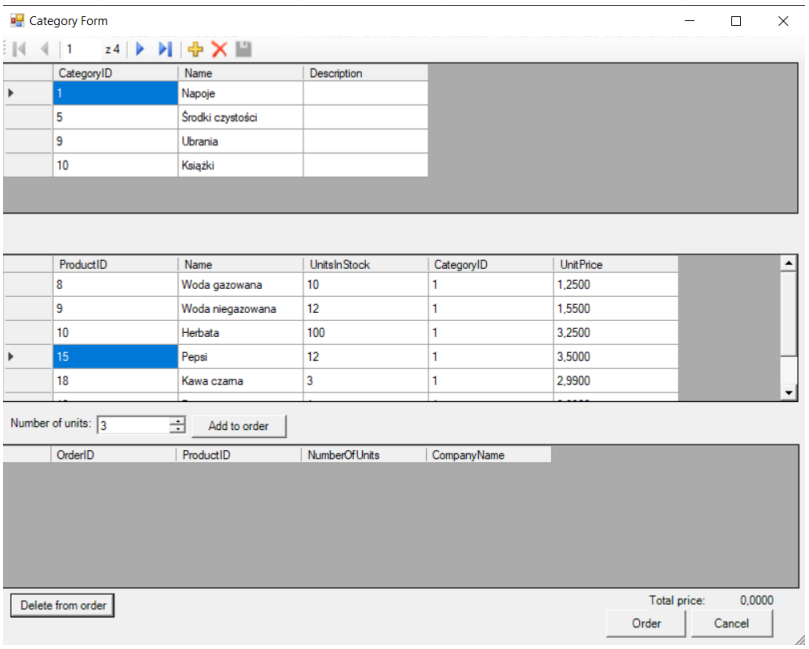
Przykładowe działanie przedstawiono na Rysunkach 10-12.



Rysunek 10. Dodanie nowego zamówienia



Rysunek 11. Zapisanie zmian w kontekście



Rysunek 12. Usuwanie zamówienia