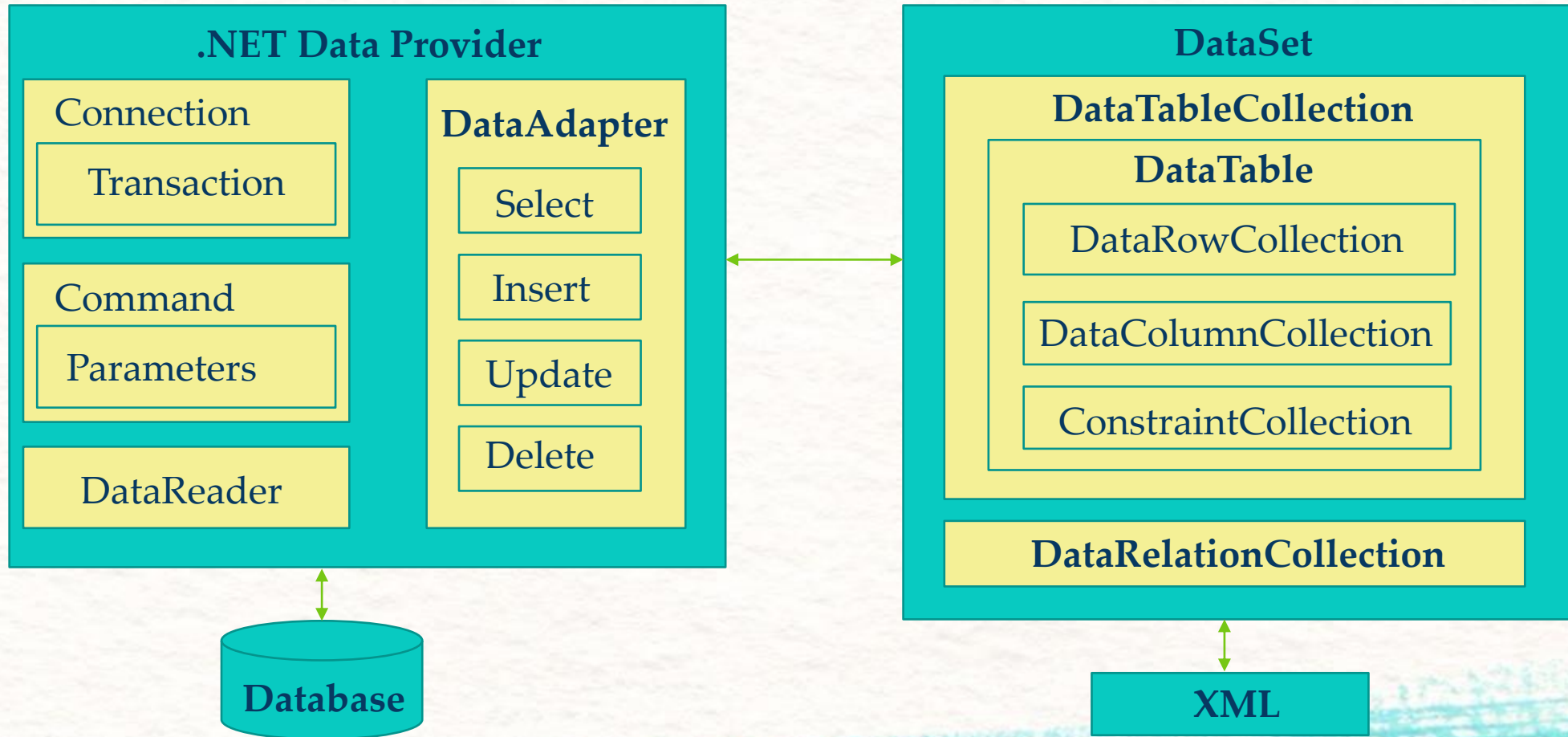


# ADO.NET

Seminar 2

# Arhitectura ADO.NET





## Data Binding – Windows Forms

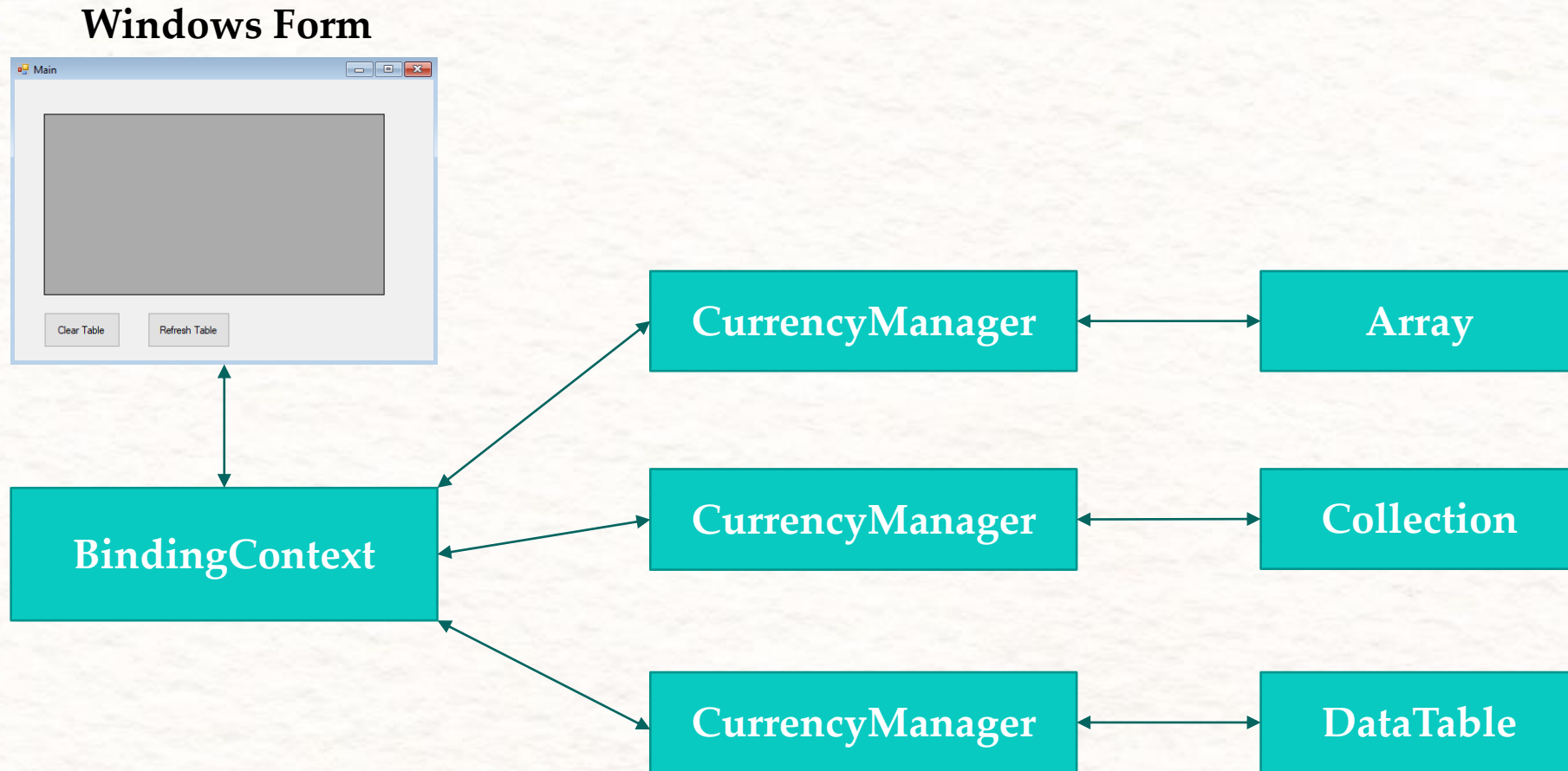
- **Data binding**-ul este o modalitate automată de a seta orice proprietate accesibilă la *runtime* a oricărui control din form
- În **Windows Forms**, **data binding**-ul :
  - oferă modalități de a **afișa** și de a **modifica** informații care **provin** din **surse de date** în **controalele** din **form**
  - permite **accesarea** datelor din diferite **surse de date** (aproape orice **structură** care conține **date** poate fi o **sursă de date**: *Array*, *Collection*, *DataTable*, etc.)

# Data Binding – Windows Forms

- Există **două** tipuri de **data binding**:
  - **Simple data binding**: legarea unui **control** (*TextBox*, *Label*, etc.) la un **singur data element**, cum ar fi o **valoare** dintr-o **coloană** a unui *DataTable* dintr-un *DataSet*
  - **Complex data binding**: legarea unui **control** (*DataGridView*, *ListBox*, *ComboBox*, etc.) la mai **multe data elements**, în general la mai multe **înregistrări** dintr-o bază de date



# Data Binding – Windows Forms



## Consumatori de date în .NET

- **BindingContext** gestionează colecția de obiecte *BindingManagerBase* pentru orice obiect care derivă din clasa *Control*
- Fiecare **Windows Form** are cel puțin un obiect **BindingContext** care gestionează colecția de obiecte *BindingManagerBase* pentru form



## Consumatori de date în .NET

- **BindingManagerBase** este o clasă **abstractă**, deci tipul returnat al proprietății *Item[Object]* poate fi **CurrencyManager** sau **PropertyManager**
- Pentru fiecare **sursă de date** asociată cu un **Windows Form**, există un singur obiect **CurrencyManager** sau **PropertyManager**

## Consumatori de date în .NET

- Dacă **sursa de date** este un **obiect** care poate returna doar o **singură proprietate** (în locul unei liste de obiecte), tipul va fi **PropertyManager**
- Dacă **sursa de date** este un **obiect** care implementează **interfața *IList* sau *IBindingList***, tipul va fi **CurrencyManager**



# Consumatori de date în .NET

- **PropertyManager**

- Menține legătura dintre proprietatea unui **obiect** și proprietatea *data-bound* a unui **control**
- Derivă din clasa *BindingManagerBase*

## Consumatori de date în .NET

- **CurrencyManager**

- Menține controalele legate la date **sincronizate** între ele (administrează o **listă** de obiecte *Binding*)
- Derivă din clasa *BindingManagerBase*

- **Proprietăți:**

- **Current** – returnează elementul curent din listă
- **Position** – returnează sau setează poziția curentă a tuturor controalelor conectate la același *CurrencyManager*



# Controale data-bound

- **BindingSource**

- Conectează controalele din **form** la un *DataTable* din *DataSet*
- **Simplifică** legarea **controalelor** din form la **date** oferind *currency management, change notification* și alte servicii
- **Sursele de date** care sunt **legate** la o componentă **BindingSource** pot fi **parcurse** și **administrate** cu *BindingNavigator*

## Controale data-bound

- **BindingNavigator**

- Folosit pentru **parcurgerea** înregistrărilor din tabel
- Interfața utilizator a unui **BindingNavigator** este compusă dintr-o serie de butoane *ToolStrip*, *text boxes* și *static text elements* pentru cele mai comune acțiuni asupra datelor, cum ar fi **adăugarea**, **ștergerea** și **parcurgerea** înregistrărilor



## Surse de date în .NET

- O listă trebuie să **implementeze** interfața *IList* pentru a putea îndeplini funcția de **sursă de date**
- ADO.NET furnizează structuri de date **potrivite** pentru *binding*:
- **DataColumn**
  - Este componenta **fundamentală** în construirea structurii unui *DataTable* (structura tabelului este construită prin adăugarea unui obiect sau a mai multor obiecte  *DataColumn*  unei  *DataColumnCollection* )
  - Are o proprietate numită *DataType* care determină tipul de date

# Surse de date în .NET

- **DataTable**

- Este un obiect central în biblioteca ADO.NET și poate fi folosit de obiectele *DataSet* și *DataRowView*
- Reprezintă un **tabel** cu **date stocate** în **memorie** și conține:
  - o colecție de obiecte  *DataColumn*
  - o colecție de obiecte  *DataRow*
  - o colecție de obiecte  *Constraint*



# Surse de date în .NET

- **DataView**

- Este un **view personalizat** pentru **sortare, filtrare, căutare, editare și navigare** a unui *DataTable* și permite *data binding*
- Permite crearea mai **multor** view-uri diferite a datelor stocate într-un *DataTable*
- Oferă **vizualizarea dinamică** a unui **singur set de date**, asupra căruia se pot aplica diferite **criterii de sortare și filtrare** (asemănător unui view dintr-o bază de date)
- Poate fi personalizat pentru a afișa doar o **parte** a datelor din *DataTable* (acest lucru permite ca două controale legate la același *DataTable* să afișeze două versiuni diferite a datelor)

## Surse de date în .NET

- **DataView**

- **Nu** poate fi tratat ca un tabel și **nu** poate conține date din mai multe tabele
- **Nu** poate exclude coloane care există în tabelul sursă, nici **nu** poate adăuga coloane care **nu** există în tabelul sursă (cum ar fi coloanele calculate)
- **Nu** stochează date, ci doar reprezintă un view conectat al *DataTable*-ului corespunzător



# Surse de date în .NET

- **DataSet**

- Reprezintă un cache în memorie al datelor
- Este compus din tabele, relații și constrângeri

- **DataViewManager**

- Reprezintă un view personalizat al unui *DataSet*
- Poate fi folosit pentru a administra setările de vizualizare a tuturor tabelelor dintr-un *DataSet*
- Oferă o modalitate convenabilă de a administra setările implicite de vizualizare pentru fiecare tabel
- Conține un *DataViewSettingCollection* implicit pentru fiecare *DataTable* din *DataSet*

## Popularea DataSet-urilor cu date

- Un *DataSet* **nu** conține date în mod implicit
- Tabelele sunt **populate** cu date prin **execuția** interogărilor *TableAdapter* sau prin **execuția** comenzilor *DataAdapter* (*SqlDataAdapter*)



# Popularea DataSet-urilor cu date

- Exemplu *DataSet* tipizat:

```
categoriiFloriTableAdapter.Fill(florarieDataSet.CategoriiFlori);
```

- Exemplu *DataSet* netipizat:

```
categoriiProduseSqlDataAdapter.Fill(magazinDataSet, "CategoriiProduse");
```

# Salvarea datelor dintr-un DataSet

- Exemplu *DataSet* tipizat:

```
categoriiFloriTableAdapter.Update(florarieDataSet.CategoriiFlori);
```

- Exemplu *DataSet* netipizat:

```
categoriiProduseSqlDataAdapter.Update(magazinDataSet, "CategoriiProduse");
```



## Salvarea datelor dintr-un DataSet

- Metoda **Update** examinează **valoarea** proprietății *RowState* pentru a **determina** care sunt înregistrările ce urmează să fie **salvate** și care **comandă specifică** trebuie **invocată** (*InsertCommand*, *UpdateCommand* sau *DeleteCommand*)

## Accesarea înregistrărilor

- Fiecare tabel expune o colecție de înregistrări
- Ca în orice colecție, înregistrările se pot accesa folosind **indexul colecției** sau utilizând **instrucțiuni specifice colecției** în limbajul de programare utilizat



# Accesarea înregistrărilor

- Exemplu *DataSet* tipizat:

```
textBox1.Text = florarieDataSet.CategoriiFlori[1].nume;
```

- Exemplu *DataSet* netipizat:

```
string numeCategorie = (string)  
magazinDataSet.Tables["CategoriiProduce"].Rows[0]["nume"];
```

## Tabele asociate și obiecte *DataRelation*

- Informațiile din tabelele aflate într-un *DataSet* pot fi inter-relaționate
- Crearea obiectelor *DataRelation* permite descrierea **relațiilor** dintre tabelele care se află în *DataSet*
- Se poate folosi un obiect *DataRelation* pentru a localiza înregistrări asociate prin apelarea metodei **GetChildRows** pe un *DataRow* din tabelul părinte (această metodă returnează un *array* de înregistrări copil asociate)
- Se poate apela metoda **GetParentRow** a unui *DataRow* din tabelul copil (această metodă returnează un singur *DataRow* din tabelul părinte)



# Returnarea înregistrărilor copil a unei înregistrări părinte

- Exemplu *DataSet* tipizat:

```
//Se va afișa numărul de comenzi ale clientului cu ID-ul "ALFKI"  
string custID = "ALFKI";  
  
NorthwindDataSet.OrdersRow[] orders;  
  
orders = (NorthwindDataSet.OrdersRow[])  
northwindDataSet.Customers.  
FindByCustomerID(custID).GetChildRows  
("FK_Orders_Customers");  
  
MessageBox.Show(orders.Length.ToString());
```

# Returnarea înregistrărilor copil a unei înregistrări părinte

- Exemplu *DataSet* netipizat:

```
//Se va afișa numărul de produse pentru înregistrarea aflată pe  
poziția 2 din DataTable "CategoriiProduse"
```

```
DataRow[] produse;
```

```
produse = magazinDataSet.Tables["CategoriiProduse"].Rows[2].
```

```
    GetChildRows("FK_CategoriiProduse_Produse");
```

```
MessageBox.Show(produse.Length.ToString());
```



# Returnarea înregistrării părinte a unei înregistrări copil

- Exemplu *DataSet* tipizat:

```
//Se va afișa numele companiei pentru clientul care a făcut  
comanda cu ID-ul 10707
```

```
int orderID = 10707;
```

```
NorthwindDataSet.CustomersRow customer;
```

```
customer = (NorthwindDataSet.CustomersRow)  
northwindDataSet.Orders.FindByOrderID(orderID).
```

```
GetParentRow("FK_Orders_Customers");
```

```
MessageBox.Show(customer.CompanyName);
```

# Returnarea înregistrării părinte a unei înregistrări copil

- Exemplu *DataSet* netipizat:

```
//Se va afișa numele categoriei pentru înregistrarea  
aflată pe poziția 1 din DataTable "Produse"
```

```
DataRow categorieProduse;
```

```
categorieProduse = magazinDataSet.Tables["Produse"].
```

```
Rows[1].GetParentRow("FK_CategoriiProduse_Produse");
```

```
MessageBox.Show(categorieProduse["nume"].ToString());
```



## Constrângeri

- Constrângerile pot fi folosite pentru a impune **restricții** asupra datelor stocate în *DataTable* (pentru a menține integritatea datelor) și sunt adăugate în *ConstraintCollection* a obiectului *DataTable*
- O constrângere este o **regulă** care se aplică în mod **automat** unei coloane sau unui grup de coloane și determină **modul de acțiune** în momentul în care valoarea unei înregistrări este modificată în vreun fel

## Constrângeri

- Constrângerile sunt aplicate când valoarea proprietății *EnforceConstraints* a unui obiect *DataSet* este **True** (în mod implicit este setată pe **True**)
- Două tipuri de constrângeri sunt disponibile în **ADO.NET** și sunt create în mod automat atunci când un obiect *DataRelation* este adăugat unui *DataSet*:
  - **ForeignKeyConstraint**
  - **UniqueConstraint**



## Constrângeri

- **ForeignKeyConstraint** impune reguli privind propagarea actualizărilor și ștergerilor în tabelele asociate
- Proprietățile **UpdateRule** și **DeleteRule** ale *ForeignKeyConstraint* definesc acțiunea care va avea loc atunci când utilizatorul actualizează sau șterge o înregistrare dintr-un tabel asociat

## Constrângeri

- Proprietățile **UpdateRule** și **DeleteRule** pot avea următoarele valori:
  - **Cascade** – Actualizează sau șterge înregistrările asociate
  - **SetNull** – Setează valorile din înregistrările asociate pe *DBNull*
  - **SetDefault** – Setează valorile din înregistrările asociate pe valoarea implicită
  - **None** – Nicio acțiune nu are loc asupra înregistrărilor asociate
- În mod implicit, valorile proprietăților **UpdateRule** și **DeleteRule** sunt setate pe **None**



# Constrângeri

- **UniqueConstraint**

- Asigură **unicitatea** la nivel de înregistrare a valorilor din coloana sau coloanele pe care este definită
- Poate fi atribuită unei singure coloane sau unui *array* de coloane dintr-un *DataTable* și se poate specifica dacă această coloană (sau coloane) formează o **cheie primară**

# Constrângeri

- Exemplu de creare a unui *UniqueConstraint* cu proprietatea **IsPrimaryKey** setată pe **True** :

```
DataTable Cadouri = new DataTable("Cadouri");  
Cadouri.Columns.Add("cod_cadou", Type.GetType("System.Int32"));  
Cadouri.Columns.Add("descriere", Type.GetType("System.String"));  
UniqueConstraint uq = new UniqueConstraint("unique_constraint",  
    Cadouri.Columns["cod_cadou"], true);  
Cadouri.Constraints.Add(uq);  
MessageBox.Show("Valoarea proprietății IsPrimaryKey este "+  
    uq.IsPrimaryKey.ToString());
```



# Exemplu de aplicație C# Windows Forms - DataRelation

- În SQL Server, vom crea o bază de date numită "SGBDIR"
- După ce baza de date a fost creată, vom crea două tabele noi:

```
CREATE TABLE Categorii  
(  
    cod_categorie INT PRIMARY KEY IDENTITY,  
    nume_categorie VARCHAR(100)  
);
```

# Exemplu de aplicație C# Windows Forms - DataRelation

```
CREATE TABLE Produse  
(  
    cod_produș INT PRIMARY KEY IDENTITY,  
    nume_produș VARCHAR(100),  
    pret REAL,  
    cod_categorie INT FOREIGN KEY REFERENCES Categori  
    (cod_categorie)  
);
```



# Exemplu de aplicație C# Windows Forms - DataRelation

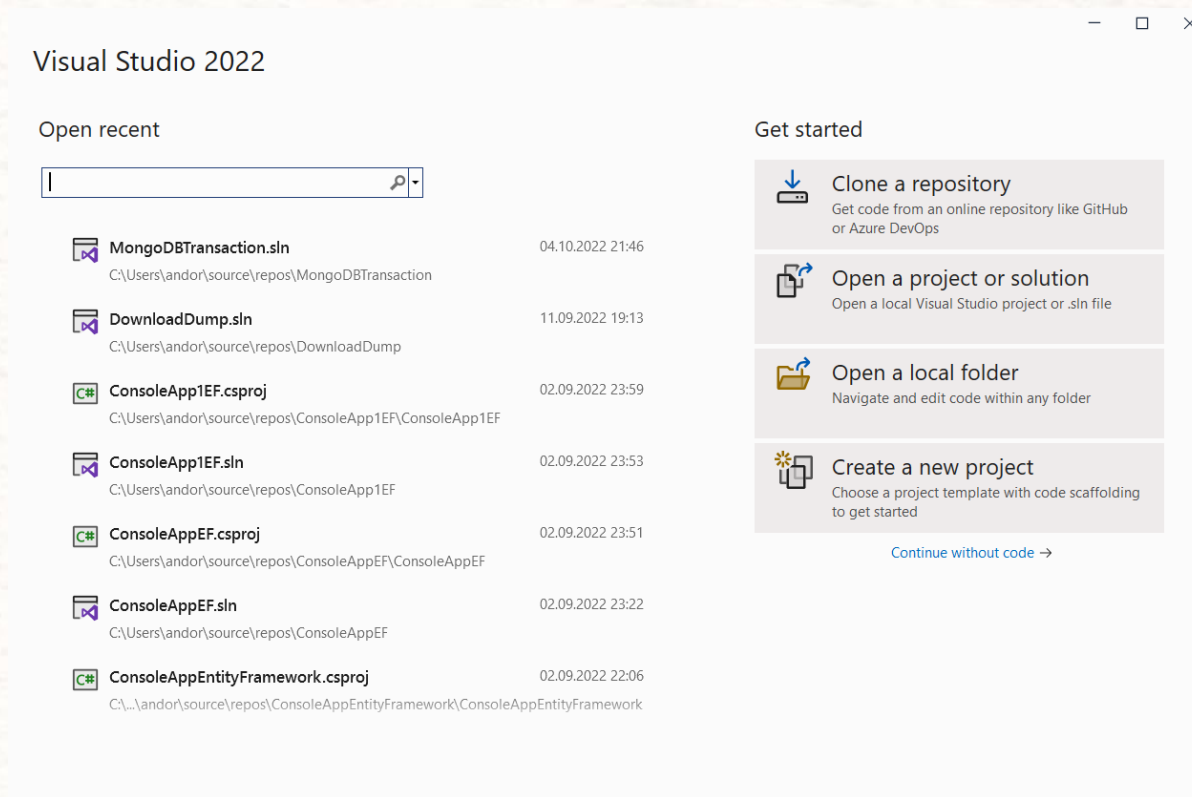
- Apoi, vom adăuga câteva înregistrări în fiecare tabel:

```
INSERT INTO Categori (nume_categorie) VALUES  
( 'dulciuri'), ('haine');
```

```
INSERT INTO Produse (nume_produs, pret, cod_categorie)  
VALUES ( 'Milka', 3, 1), ( 'Oreo', 2.5, 1),  
( 'Tricou', 56, 2), ( 'Blugi', 100, 2);
```

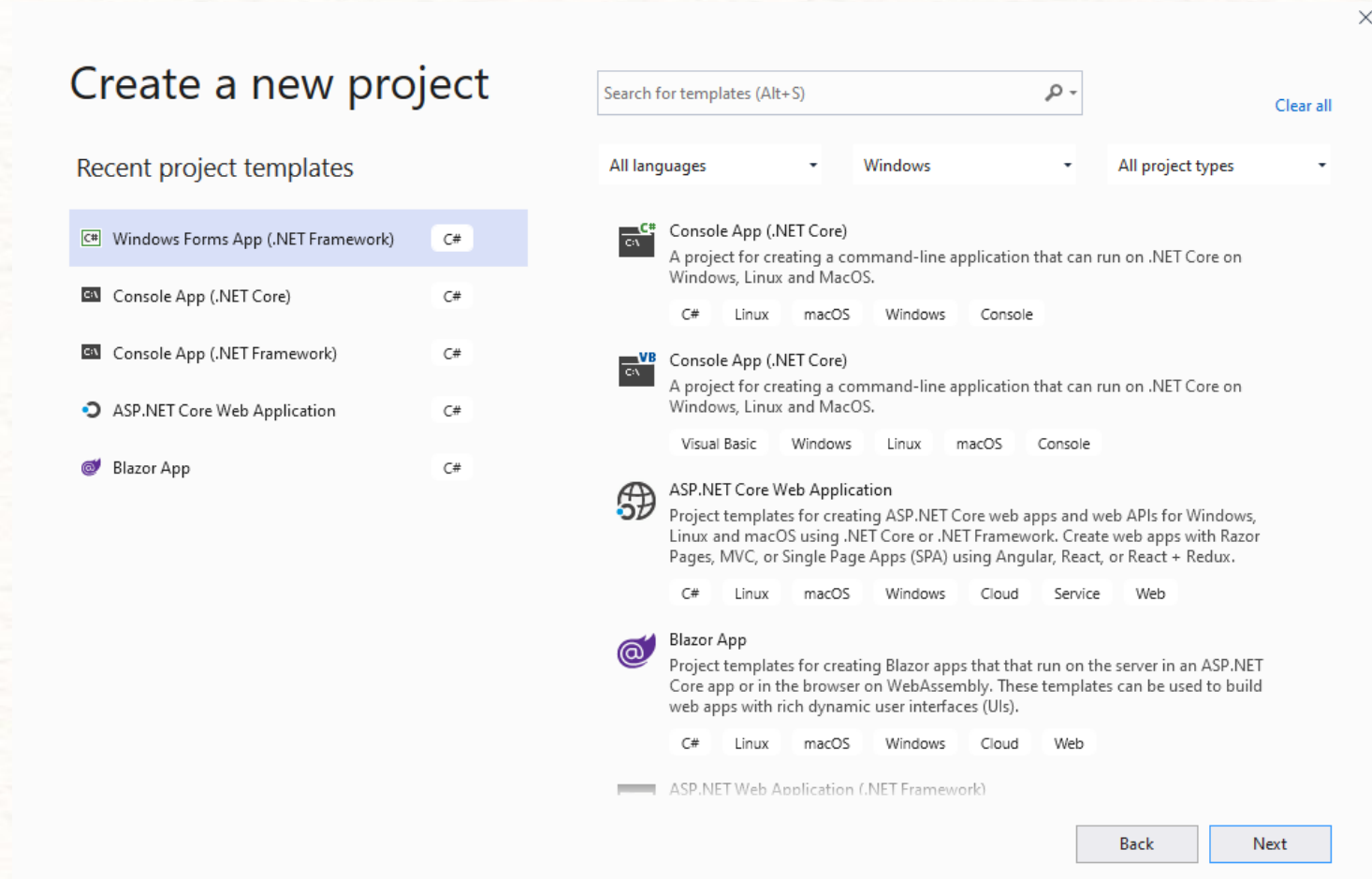
# Exemplu de aplicație C# Windows Forms - DataRelation

- În Visual Studio, vom crea un nou proiect folosind template-ul Windows Forms App disponibil în lista de template-uri a Visual C#:

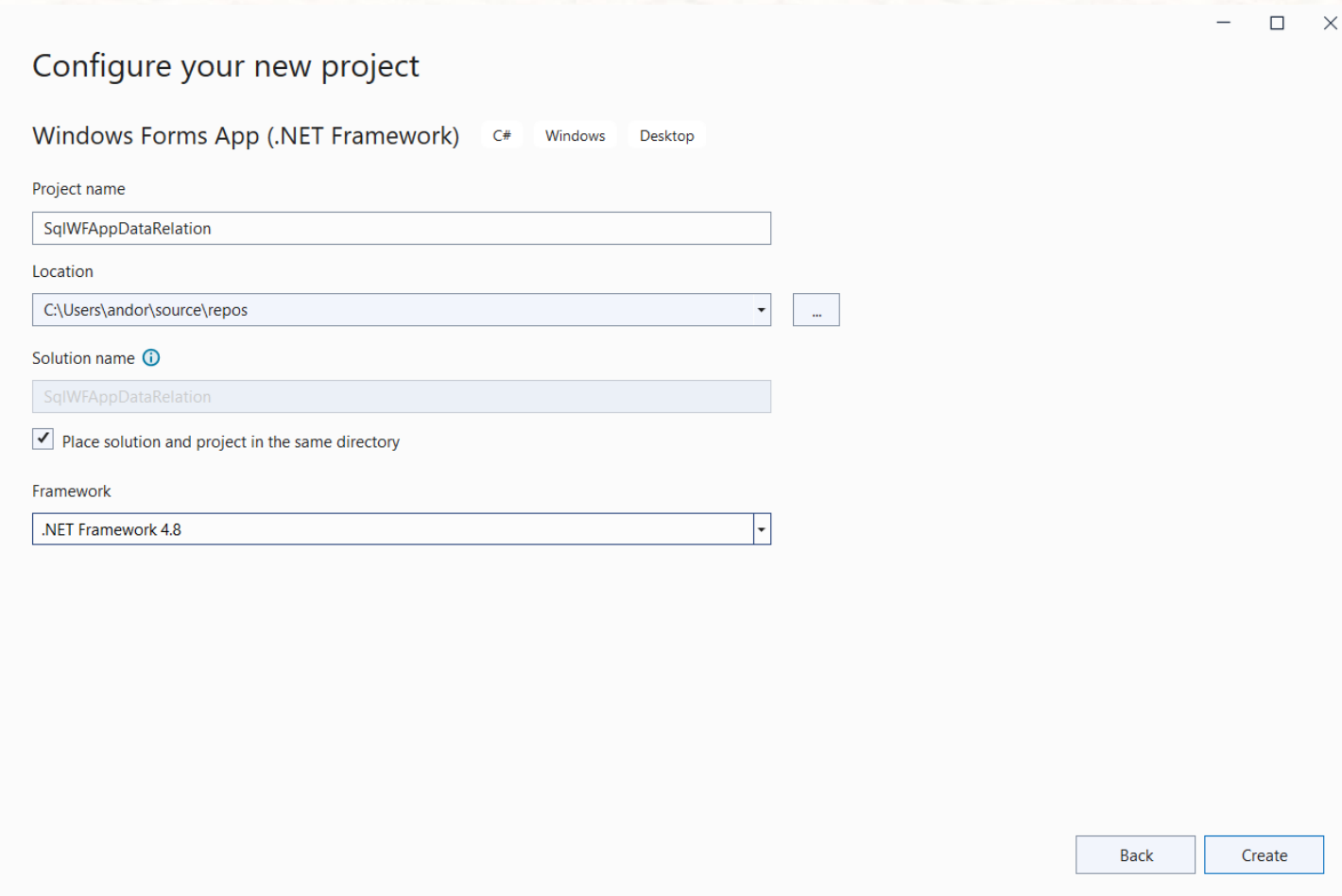




# Exemplu de aplicație C# Windows Forms - DataRelation



# Exemplu de aplicație C# Windows Forms - DataRelation



Configure your new project

Windows Forms App (.NET Framework) C# Windows Desktop

Project name

SqlWfAppDataRelation

Location

C:\Users\andor\source\repos

Solution name ⓘ

SqlWfAppDataRelation

☒ Place solution and project in the same directory

Framework

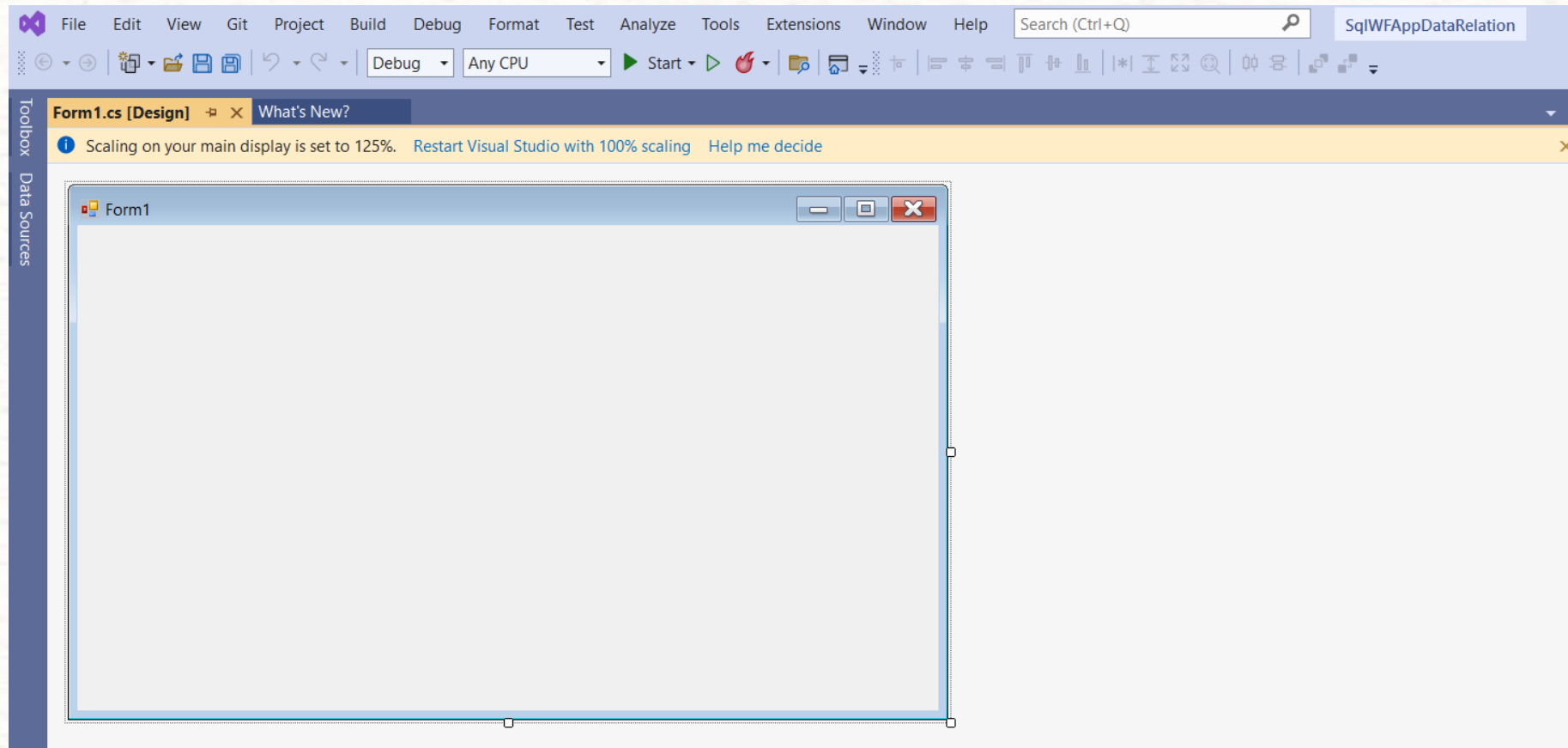
.NET Framework 4.8

Back Create



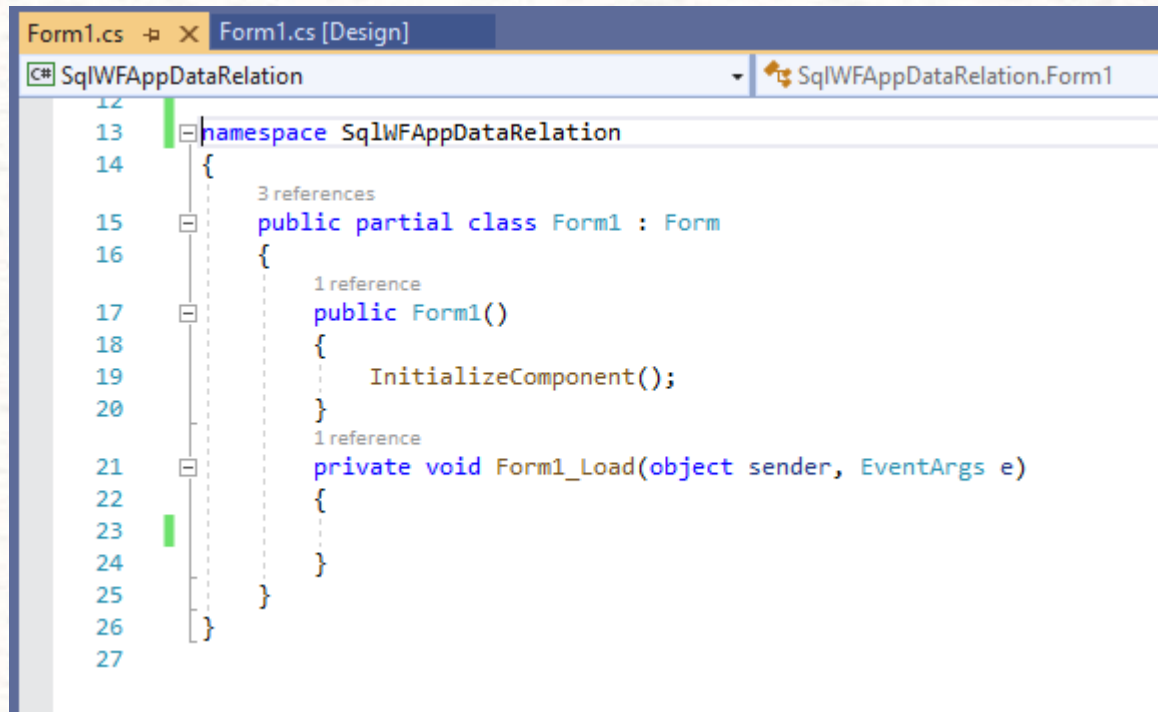
# Exemplu de aplicație C# Windows Forms - DataRelation

- După ce proiectul a fost creat, apare primul *Form* al aplicației, denumit *Form1*:



# Exemplu de aplicație C# Windows Forms - DataRelation

- După un dublu click pe *Form1*, se va crea event handler-ul *Form1\_Load* (vizibil în fișierul **Form1.cs**):



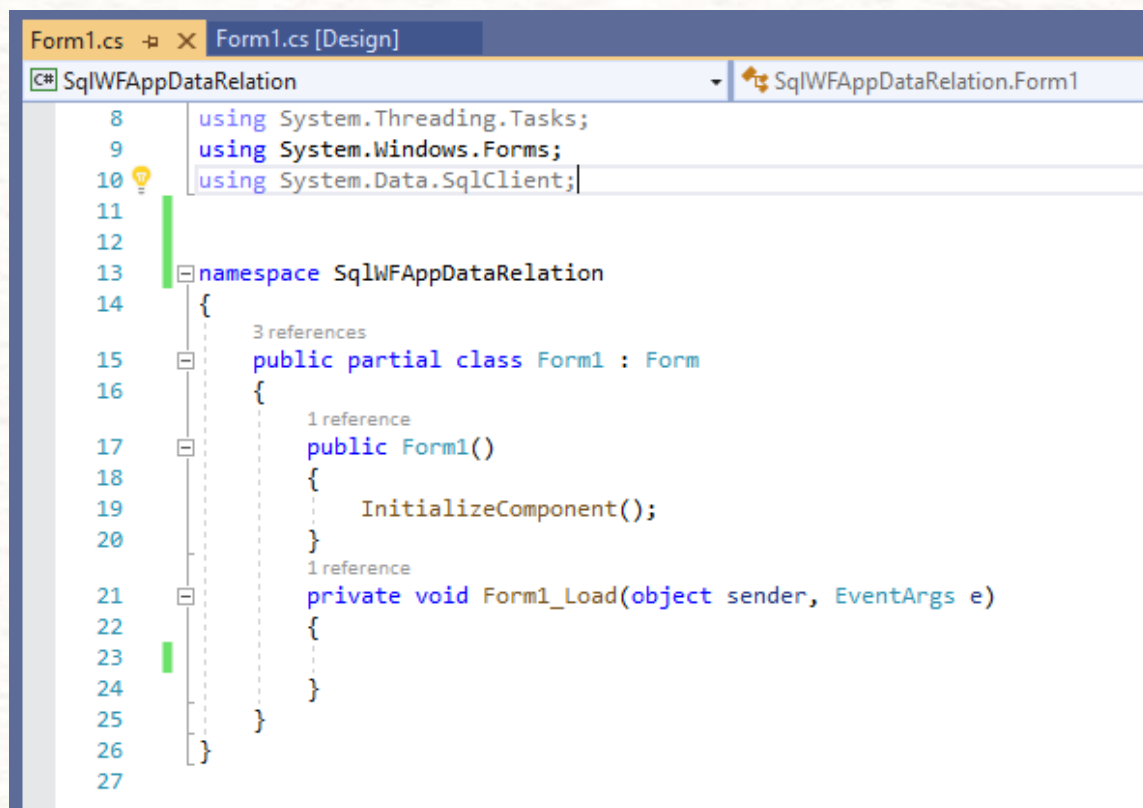
The screenshot shows the Visual Studio IDE with the 'Form1.cs [Design]' window active. The code editor displays the following C# code:

```
12
13 namespace SqlWFApDataRelation
14 {
15     3 references
16     public partial class Form1 : Form
17     {
18         1 reference
19         public Form1()
20         {
21             InitializeComponent();
22         }
23         1 reference
24         private void Form1_Load(object sender, EventArgs e)
25         {
26         }
27     }
```



# Exemplu de aplicație C# Windows Forms - DataRelation

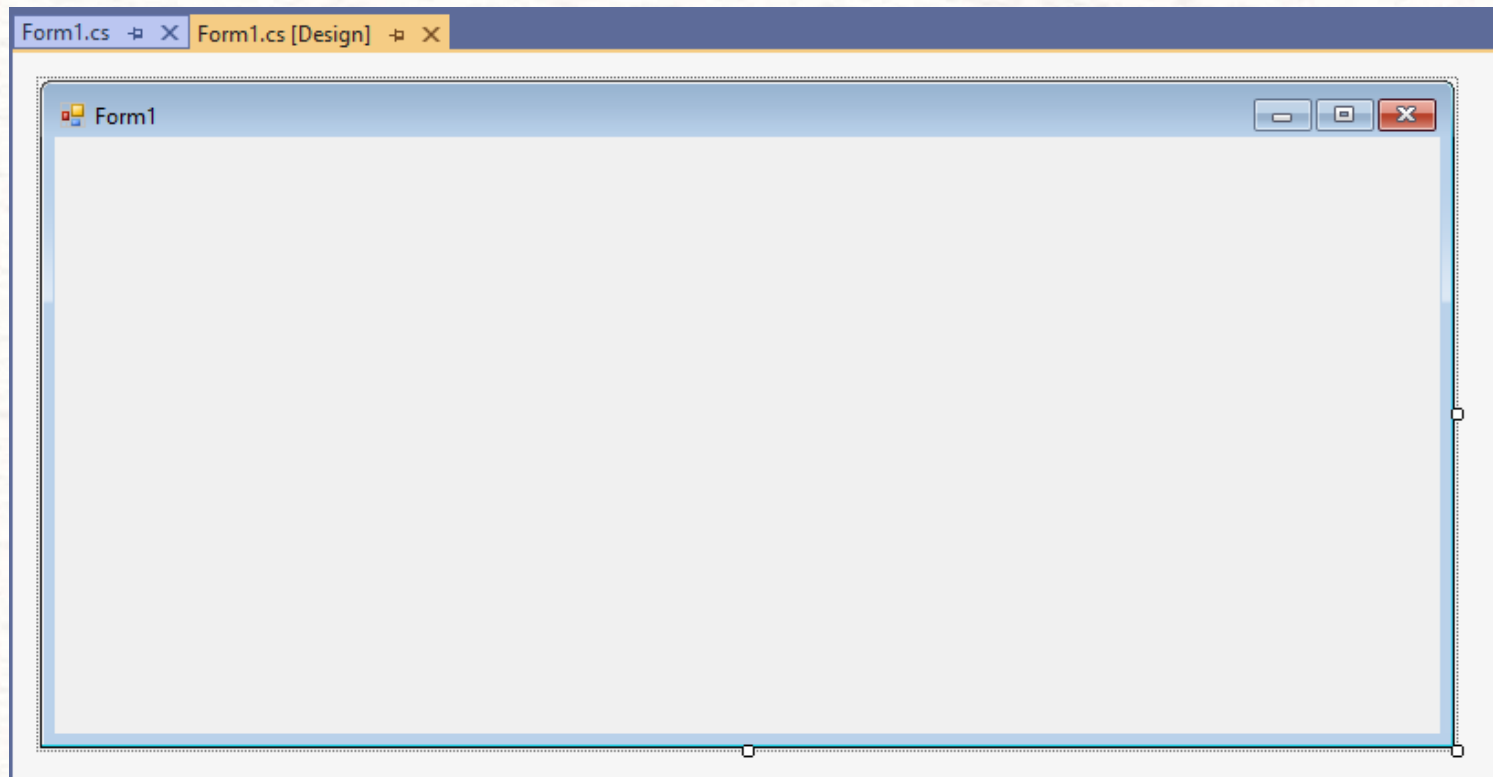
- După aceea, vom include în fișierul **Form1.cs** namespace-ul **System.Data.SqlClient** care este .NET Data Provider pentru SQL Server:



```
Form1.cs [Design]
SqlWfAppDataRelation
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10 using System.Data.SqlClient;
11
12
13 namespace SqlWfAppDataRelation
14 {
15     3 references
16     public partial class Form1 : Form
17     {
18         1 reference
19         public Form1()
20         {
21             InitializeComponent();
22         }
23         1 reference
24         private void Form1_Load(object sender, EventArgs e)
25         {
26         }
27     }
28 }
```

# Exemplu de aplicație C# Windows Forms - DataRelation

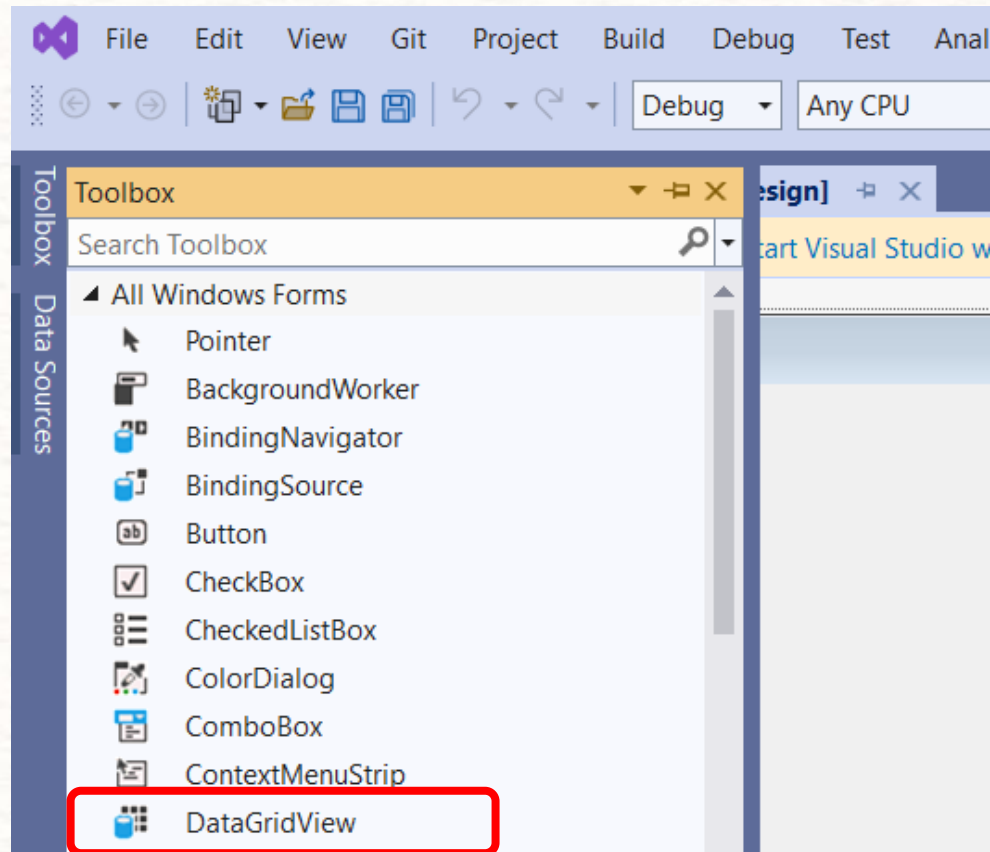
- Deoarece avem nevoie de mai mult spațiu pentru a plasa două controale *DataGridView* pe *Form*, îl vom extinde:





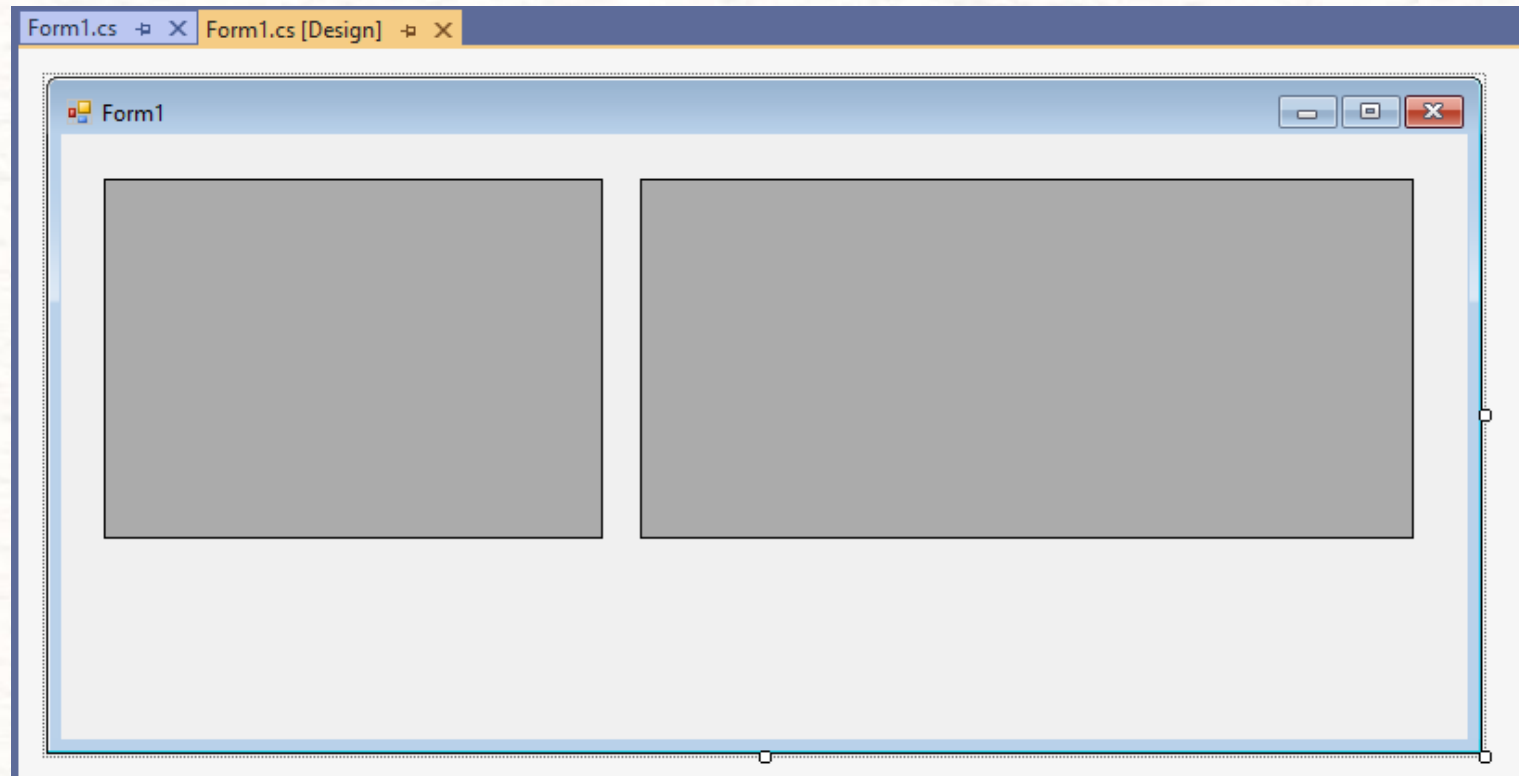
# Exemplu de aplicație C# Windows Forms - DataRelation

- Din **Toolbox**, vom plasa în interiorul *Form*-ului două controale *DataGridView*:



# Exemplu de aplicație C# Windows Forms - DataRelation

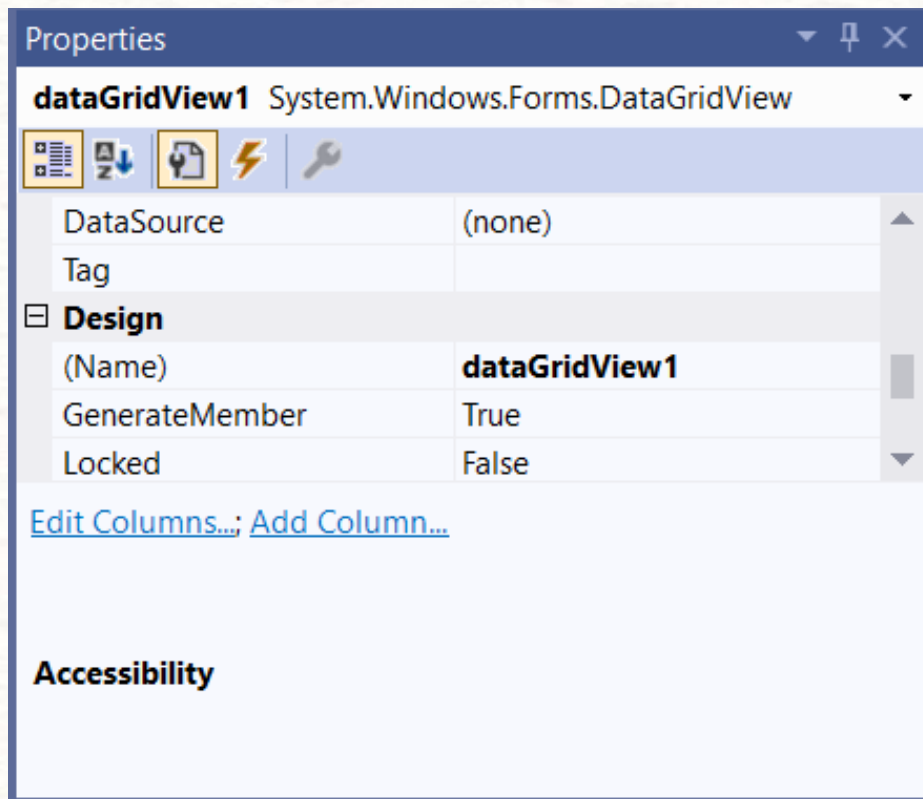
- După plasarea celor două controale *DataGridView*, *Form*-ul arată în modul următor:





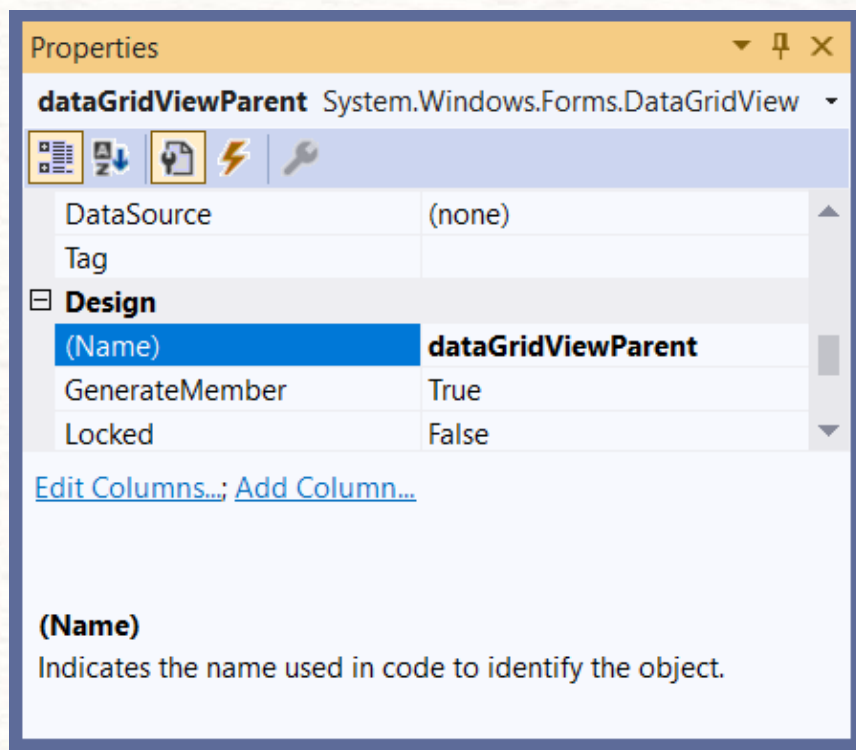
# Exemplu de aplicație C# Windows Forms - DataRelation

- După un click pe primul *DataGridView*, putem vedea în fereastra **Properties** că proprietatea **Name** este setată pe “dataGridView1”:



# Exemplu de aplicație C# Windows Forms - DataRelation

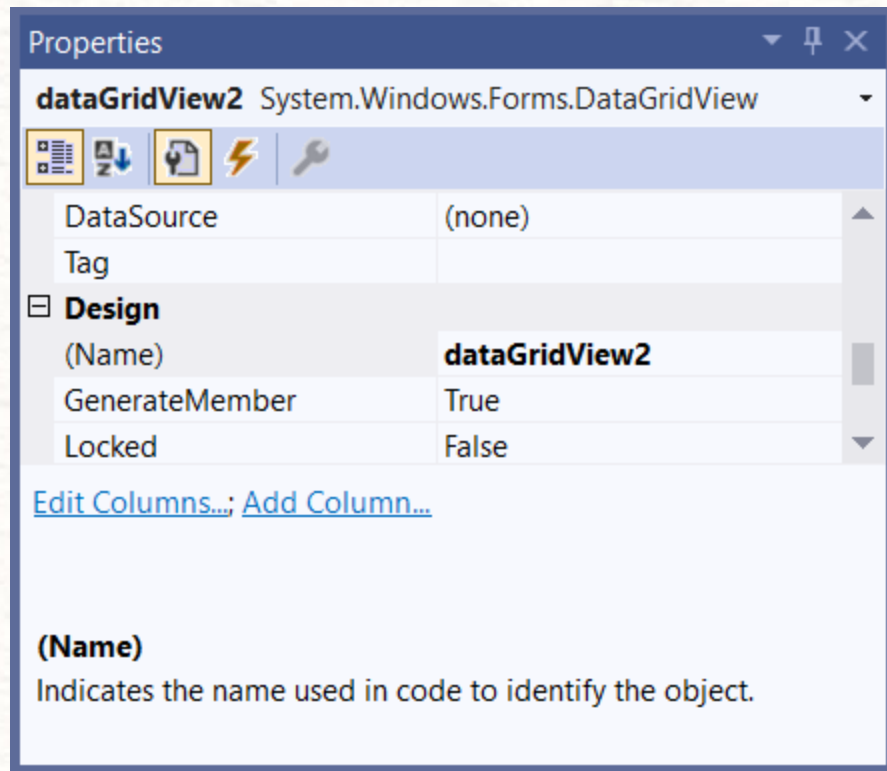
- Deoarece în primul *DataGridView* vor fi afișate datele stocate în tabelul părinte (tabelul “Categorii”), vom schimba valoarea proprietății **Name** din “dataGridView1” în “dataGridViewParent”:





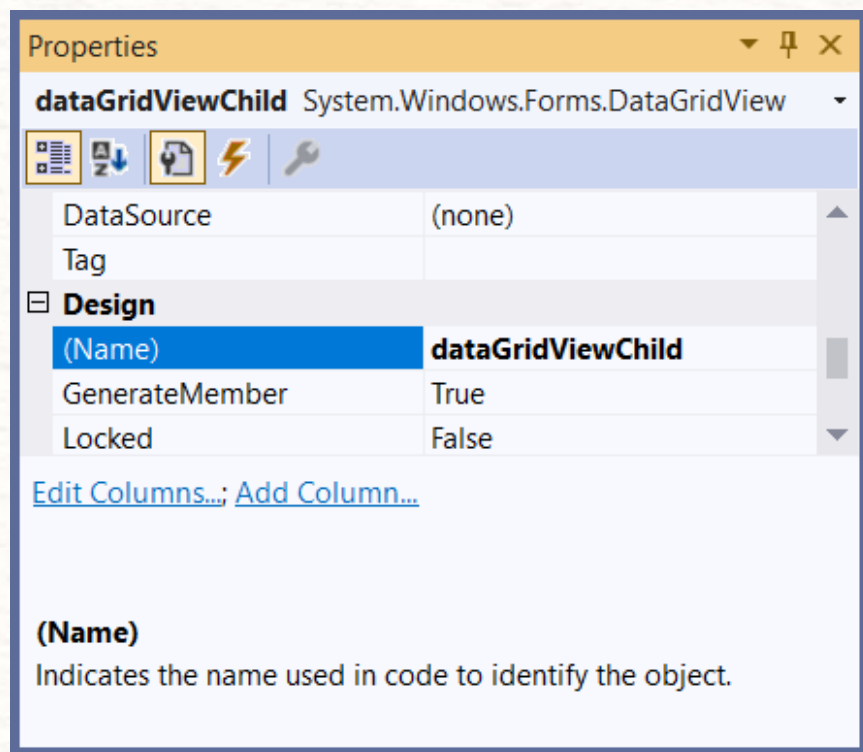
# Exemplu de aplicație C# Windows Forms - DataRelation

- După un click pe al doilea *DataGridView*, putem vedea în fereastra **Properties** că proprietatea **Name** este setată pe “dataGridView2”:



# Exemplu de aplicație C# Windows Forms - DataRelation

- Deoarece în al doilea *DataGridView* vor fi afișate datele stocate în tabelul copil (tabelul “Produce”), vom schimba valoarea proprietății **Name** din “dataGridView2” în “dataGridViewChild”:





## Exemplu de aplicație C# Windows Forms - DataRelation

- Următoarea secvență de cod (inclusă în fișierul **Form1.cs**) deschide o conexiune la baza de date pentru a crea și a popula două *DataTables* ("Categorii" și "Produse") dintr-un *DataSet* folosind *SqlDataAdapters*
- Un *BindingSource* leagă *DataTable*-ul "Categorii" (tabelul părinte) de *DataGridView*-ul corespunzător
- Între cele două *DataTables* se va crea un *DataRelation*
- *DataRelation*-ul este folosit pentru a afișa doar acele înregistrări din tabelul copil care sunt asociate înregistrării selectate din tabelul părinte
- Înregistrările din tabelul copil sunt afișate în *DataGridView*-ul corespunzător *DataTable*-ului copil ("Produse")

# Exemplu de aplicație C# Windows Forms - DataRelation

```
using System;  
  
using System.Collections.Generic;  
  
using System.ComponentModel;  
  
using System.Data;  
  
using System.Drawing;  
  
using System.Linq;  
  
using System.Text;  
  
using System.Threading.Tasks;  
  
using System.Windows.Forms;  
  
using System.Data.SqlClient;
```



# Exemplu de aplicație C# Windows Forms - DataRelation

```
namespace SqlWFAppDataRelation
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

# Exemplu de aplicație C# Windows Forms - DataRelation

```
private void Form1_Load(object sender, EventArgs e)
{
    string connectionString =
    "Server=ACERASPIRE;Database=SGBDIR;Integrated Security=true;
    TrustServerCertificate=true;";

    try
    {
        using (SqlConnection connection = new
        SqlConnection(connectionString))
        {
            //Deschide conexiunea

            connection.Open();

            MessageBox.Show("Starea conexiunii: " +
            connection.State.ToString());
        }
    }
}
```



# Exemplu de aplicație C# Windows Forms - DataRelation

```
//Crearea DataSet-ului
```

```
DataSet dataset = new DataSet();
```

```
//Crearea celor două SqlDataAdapter's pentru tabelele părinte și copil
```

```
SqlDataAdapter parentAdapter = new SqlDataAdapter("SELECT * FROM Categoriile;", connection);
```

```
SqlDataAdapter childAdapter = new SqlDataAdapter("SELECT * FROM Produse;", connection);
```

```
//Crearea și popularea DataTable-ului părinte și a DataTable-ului copil
```

```
parentAdapter.Fill(dataset, "Categoriile");
```

```
childAdapter.Fill(dataset, "Produse");
```

# Exemplu de aplicație C# Windows Forms - DataRelation

```
//Crearea celor două BindingSources pentru DataTable-ul părinte și  
pentru DataTable-ul copil
```

```
BindingSource parentBS = new BindingSource();
```

```
BindingSource childBS = new BindingSource();
```

```
//Afișarea tuturor înregistrărilor din DataTable-ul părinte în  
dataGridViewParent
```

```
parentBS.DataSource = dataset.Tables["Categorii"];
```

```
dataGridViewParent.DataSource = parentBS;
```



# Exemplu de aplicație C# Windows Forms - DataRelation

//Crearea și adăugarea în DataSet a DataRelation-ului dintre DataTable-ul părinte și DataTable-ul copil

```
DataColumn parentPK =  
dataset.Tables["Categorii"].Columns["cod_categorie"];
```

```
DataColumn childFK =  
dataset.Tables["Produse"].Columns["cod_categorie"];
```

```
DataRelation relation = new DataRelation("fk_parent_child",  
parentPK, childFK);
```

```
dataset.Relations.Add(relation);
```

## Exemplu de aplicație C# Windows Forms - DataRelation

//Afișarea în dataGridViewChild a înregistrărilor copil care aparțin înregistrării părinte selectate

```
childBS.DataSource = parentBS;
```

```
childBS.DataMember = "fk_parent_child";
```

```
dataGridViewChild.DataSource = childBS;
```

```
}
```

```
}
```

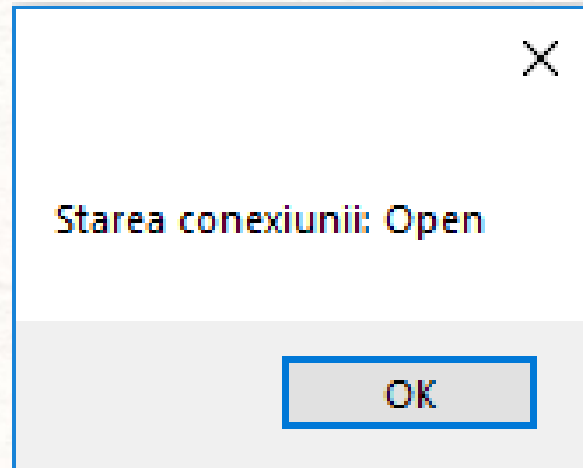


# Exemplu de aplicație C# Windows Forms - DataRelation

```
catch (Exception err)
{
    MessageBox.Show(err.Message.ToString());
}
}
}
}
```

# Exemplu de aplicație C# Windows Forms - DataRelation

- După ce pornim aplicația, apare un **MessageBox** care afișează starea conexiunii:





# Exemplu de aplicație C# Windows Forms - DataRelation

- După ce apăsăm butonul **OK**, apare *Form*-ul pe care se pot vedea cele două controale *DataGridView* în care sunt afișate date:

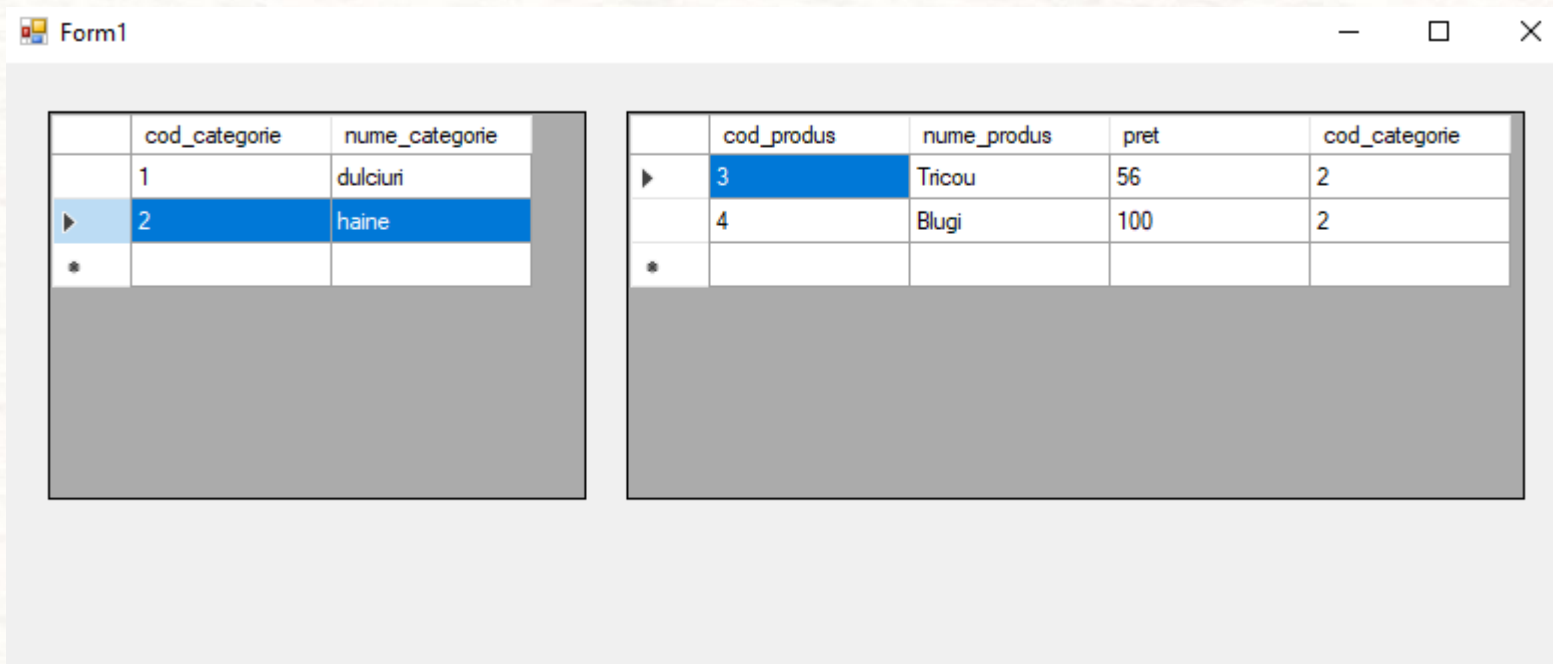
The screenshot shows a Windows Forms application window titled "Form1". Inside the window, there are two *DataGridView* controls side-by-side. The left *DataGridView* has two columns: "cod\_categorie" and "nume\_categorie". It contains three rows: a header row with a right-pointing triangle icon, a row with "1" and "dulciuri", and a row with "2" and "haine". The right *DataGridView* has four columns: "cod\_produș", "nume\_produș", "pret", and "cod\_categorie". It contains three rows: a header row with a right-pointing triangle icon, a row with "1", "Milka", "3", and "1", and a row with "2", "Oreo", "2.5", and "1". Both *DataGridView* controls have a grey background and a white border.

	cod_categorie	nume_categorie
▶	1	dulciuri
	2	haine
*		

	cod_produș	nume_produș	pret	cod_categorie
▶	1	Milka	3	1
	2	Oreo	2.5	1
*				

# Exemplu de aplicație C# Windows Forms - DataRelation

- Dacă selectăm a doua înregistrare părinte, înregistrările copil asociate vor fi afișate:



The screenshot displays a Windows Forms application window titled "Form1". It contains two data tables side-by-side. The left table has columns "cod\_categorie" and "nume\_categorie". The right table has columns "cod\_produs", "nume\_produs", "pret", and "cod\_categorie".

	cod_categorie	nume_categorie
	1	dulciuri
▶	2	haine
*		

	cod_produs	nume_produs	pret	cod_categorie
▶	3	Tricou	56	2
	4	Blugi	100	2
*				



# Bibliografie

- <https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-architecture>
- <https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/dataset-datatable-dataview/>
- <https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/dataset-datatable-dataview/adding-datarelations?source=recommendations>
- <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/bindingsource-component?source=recommendations&view=netframeworkdesktop-4.8>
- <https://www.c-sharpcorner.com/article/data-binding-in-net/>
- <https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/dataset-datatable-dataview/datatable-constraints>
- <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/bindingsource-component-architecture?view=netframeworkdesktop-4.8>

# Bibliografie

- <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/bindingnavigator-control-overview-windows-forms?view=netframeworkdesktop-4.8>
- <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/data/overview?view=netdesktop-7.0>
- <https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.bindingsource?view=windowsdesktop-7.0>
- <https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.bindingcontext?view=windowsdesktop-7.0>
- <https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.datagridview?view=windowsdesktop-7.0>
- <https://www.codeproject.com/articles/24656/a-detailed-data-binding-tutorial>