

Contents

CAPITOLUL 3.....	2
Baze de date.....	2
1. Introducere	2
1.1 Clasificarea bazelor de date	3
1.2 Structura bazelor de date	4
1.2.1 Relații între tabele.....	4
1.3 Clasele utilizate pentru accesul la baza de date	4
1.3.1 DbConnection	4
1.3.2 DbCommand	5
1.3.2 DbDataReader	5
1.3.4 DbDataAdapter	5
1.3.5 DataTable	5
1.3.6 Data Relation.....	6
1.3.7 DataSet.....	6
2. Crearea unei aplicații Win Form App cu bază de date.....	6
3. Exerciții.....	14

CAPITOLUL 3

Baze de date

În acest capitol este prezentat modul de lucru cu baze de date relaționale, create local în mediul de dezvoltare Visual Studio.

1. Introducere

ADO.NET (Active Data Objects for .NET) este o componentă .NET Framework care permite conectarea la diferite surse de date, recuperare și manipulare de date. De obicei, sursa de date poate fi o bază de date, dar poate fi și un fișier text, Excel, XML sau Access.

În bazele de date tradiționale, conexiunea este menținută pe întreaga durată a procesului. Folosind ADO.NET, lucrul cu bazele de date poate fi efectuat în timp ce este conectat sau deconectat. Deconectat înseamnă că aplicațiile se pot conecta la baza de date numai pentru recuperarea sau actualizarea datelor. În acest caz, numărul conexiunilor deschise simultan poate fi redus. În figura de mai jos putem observa arhitectura ADO.NET.

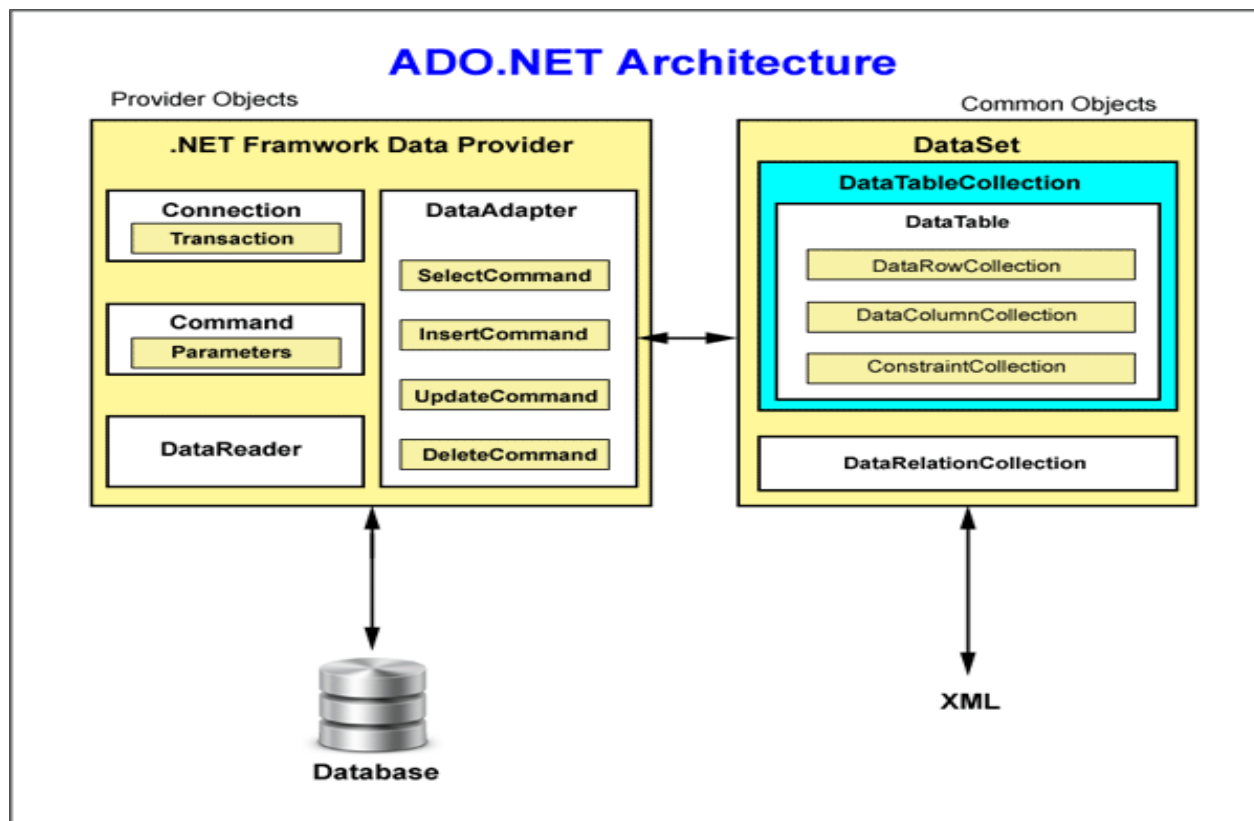


Fig. 1 Arhitectura ADO.NET

O bază de date este o colecție organizată de informații, capabilă să stocheze, să actualizeze și să regăsească informația. Practic este un mod de stocare electronică a informațiilor. Pentru ca o bază de date să fie actualizată, modificată, utilizată în obținerea unor statistici și rapoarte este nevoie de un ansamblu software interpus între utilizatori și baza de date, care poartă numele de Sistem de Gestiune a Bazelor de Date (SGBD). În figura ce urmează este prezentată arhitectura SGBD.

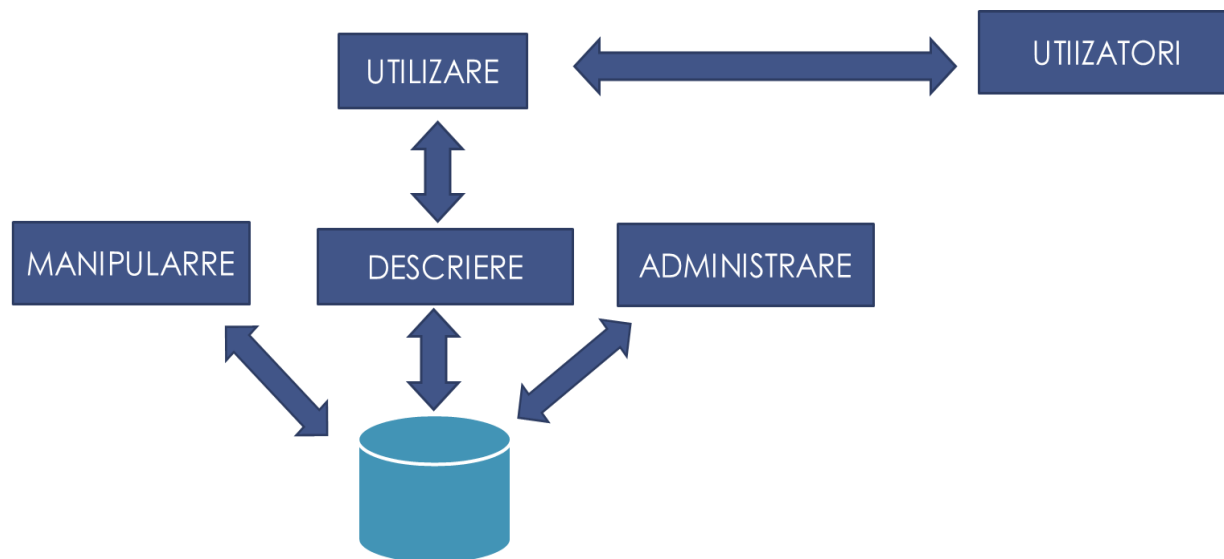


Fig.2 Arhitectura SGBD

1.1 Clasificarea bazelor de date

Bazele de date pot fi clasificate în funcție de modul de organizare a datelor, precum urmează:

- **Baze de date ierarhice** – legăturile dintre date sunt ordonate unic, accesul se face numai prin vârful ierarhiei, un subordonat nu poate avea decât un singur superior direct și nu se poate ajunge la el decât pe o singură cale;
- **Baze de date în rețea** – datele sunt reprezentate ca într-o mulțime de ierarhii, în care un membru al ei poate avea oricâți superiori, iar la un subordonat se poate ajunge pe mai multe căi;
- **Baze de date distribuite** – sunt rezultatul integrării tehnologiei bazelor de date cu cea a rețelelor de calculatoare. Sunt baze de date logic integrate, dar fizic distribuite pe mai multe sisteme de calcul;
- **Modele primitive** – datele sunt organizate la nivel logic în fișiere, structura de bază este înregistrarea, mai multe înregistrări fiind grupate în structuri de tip fișier;
- **Baze de date relaționale** – structura de bază a datelor este aceea de relație – tabelă, limbajul SQL (Structured Query Language) este specializat în comenzi de manipulare la nivel de tabelă.

1.2 Structura bazelor de date

Informațiile stocate într-o bază de date sunt organizate în mai multe **tabele**, cu diferite legături(relații) între ele. O linie dintr-un tabel poartă denumirea de **înregistrare**.

Pentru a putea crea legături între tabelele unei baze de date se definesc **chei de identificare**, care sunt de două tipuri:

- **Cheie primară (PRIMARY KEY)** – aceasta este utilizată pentru a identifica în mod *unic* fiecare înregistrare stocată într-un tabel.
Notă!: Fiecare tabel poate să aibă o singură cheie primară.
- **Cheie secundară/straină (FOREIGN KEY)** – aceasta este utilizată pentru a stabili o relație între coloana din tabelul curent și coloana tabelii de referință. Valorile din tabela curentă sunt identice cu cele din tabelul de referință, pentru câmpurile din cheie.
Notă!: Un tabel poate să aibă una sau mai multe chei secundare.

1.2.1 Relații între tabele

Relațiile sunt folosite pentru a putea conecta datele "înrudite" stocate în tabele diferite. **O relație** se definește ca fiind o conexiune logică între două sau mai multe tabele și specifică câmpurile pe care le au în comun cele două tabele.

Relațiile dintre tabele sunt de trei feluri, și anume:

- **Relatii 1:1 (One to one)** – O relație de tipul 1 la 1 apare în cazul în care unei linii dintr-un tabel îi corespunde o singură linie în tabelul cu care acesta este în legătură;
- **Relatii 1:n (One to many)** – O relație de tipul 1 la n apare în cazul în care unei linii dintr-un tabel îi corespund mai multe linii în tabelul cu care acesta este în legătură;
- **Relatii n:n (Many to many)** – O relație de tipul n la n apare în cazul în care într-o relație bidirecțională fiecărei linii dintr-un tabel îi corespund mai multe linii în tabelul cu care acesta este în legătură.

1.3 Clasele utilizate pentru accesul la baza de date

ADO.NET este o tehnologie introdusă de Microsoft utilizată pentru accesul la baze de date. Aceasta dispune de 7 clase de bază și anume:

- | | |
|---------------------------------|-------------------------------|
| ➤ <code>SqlConnection</code> ; | ➤ <code>DataTable</code> ; |
| ➤ <code>SqlCommand</code> ; | ➤ <code>DataRelation</code> ; |
| ➤ <code>SqlDataReader</code> ; | ➤ <code>DataSet</code> . |
| ➤ <code>SqlDataAdapter</code> ; | |

În cele ce urmează sunt prezentate cele șapte clase de bază.

1.3.1 SqlConnection

Această clasă este utilizată pentru crearea unei conexiuni la baza de date. Tipurile de conexiuni sunt:

- obiectul **SqlConnection**, conceput pentru SQL Server 7.0 sau o versiune ulterioară;
- obiectul **OleDbConnection**, conceput pentru Microsoft Access și Oracle.

Observație!: Clasa `DbConnection` expune metodele **`Open()`** și **`Close()`**, utilizate pentru a conecta și deconecta programul de la baza de date.

1.3.2 DbCommand

Clasa `DbCommand` implementează metode de interacțiune primară cu baza de date. Obiectele `DbCommand` pot executa interogări SQL, proceduri de stocare, etc.

Pentru executarea unei comenzi prin intermediul `DbCommand`, avem trei opțiuni și anume:

- Metoda `DbCommand.ExecuteNonQuery()` – nu returnează nici un rezultat;
- Metoda `DbCommand.ExecuteScalar()` – returnează un singur rezultat;
- Metoda `DbCommand.ExecuteReader()` – returnează date reprezentative pe mai multe randuri.

1.3.3 DbDataReader

Această clasă permite citirea datelor dintr-un set de rezultate și expune un număr relativ mare de proprietăți și metode care permit examinarea liniei curent citite.

Câteva dintre aceste metode sunt:

- `DbDataReader.FieldCount` – va conține numărul de coloane din linia citită;
- `DbDataReader.Read()` – permite citirea primei linii dintr-un set de date;
- `DbDataReader.HasRows` – verifică dacă setul de date rezultat are mai multe linii.

1.3.4 DbDataAdapter

Clasa `DbDataAdapter` permite interschimbarea datelor între cursoare (data set) și baza de date și pune la dispoziția programatorului cele patru comenzi de bază pentru lucrul cu baze de date:

- | | |
|--|--|
| ➤ Select (<code>SelectCommand</code> – regasirea datelor), | ➤ Update (<code>UpdateCommand</code> – editarea datelor) |
| ➤ Insert (<code>InsertCommand</code> – adaugarea datelor), | ➤ Delete (<code>DeleteCommand</code> – stergerea datelor). |

Această clasă expune un set de metode, printre care cele mai utilizate sunt:

- `DbDataAdapter.Fill()` - realizează încărcarea obiectului;
- `DbDataAdapter.Update()` - realizează transmiterea înspre baza de date.

1.3.5 DataTable

Clasa `DataTable` permite stocarea tabelelor de date. Obiectele `DataTable` expun o serie de evenimente, precum:

- **DataRow** – Clasa `DataRow` este utilizată pentru a stoca date dintr-o linie a obiectului `DataTable`;
- **DataColumn** – Clasa `DataColumn` stochează toate informațiile necesare pentru definirea completă a unei coloane dintr-un obiect `DataTable`;
- **Constraint** – Clasa `Constraint` stochează toate informațiile care nu sunt conținute în coloane.

1.3.6 Data Relation

Această clasă implementează relații între tabele. Mai multe obiecte `DataRelation` pot fi grupate împreună într-un obiect `DataRelationCollection`.

Un obiect `DataRelation` este, de asemenea, utilizat pentru a crea și impune următoarele constrângeri:

- *constrângere unică*, aceasta garantează că o coloană din tabel nu conține duplicate;
- *constrângere cu cheie străină*, care poate fi utilizată pentru a menține integritatea referențială între o masă părinte și o masă copil într-un set de date.

1.3.7 DataSet

Clasa `DataSet` este utilizată pentru manipularea datelor și poate să conțină o colecție de obiecte `DataTable` și `DataRelation`.

2. Crearea unei aplicații Win Form App cu baze de date

În continuare vom crea o aplicație de tip "Windows Form Application", și vor adăuga o bază de date relationala utilizând mediul de dezvoltare Visual Studio. După crearea proiectului (Fig. 3), cu click dreapta pe numele proiectului -> Add -> New Item... vom putea adăuga Service-based Database precum în fig. 4.

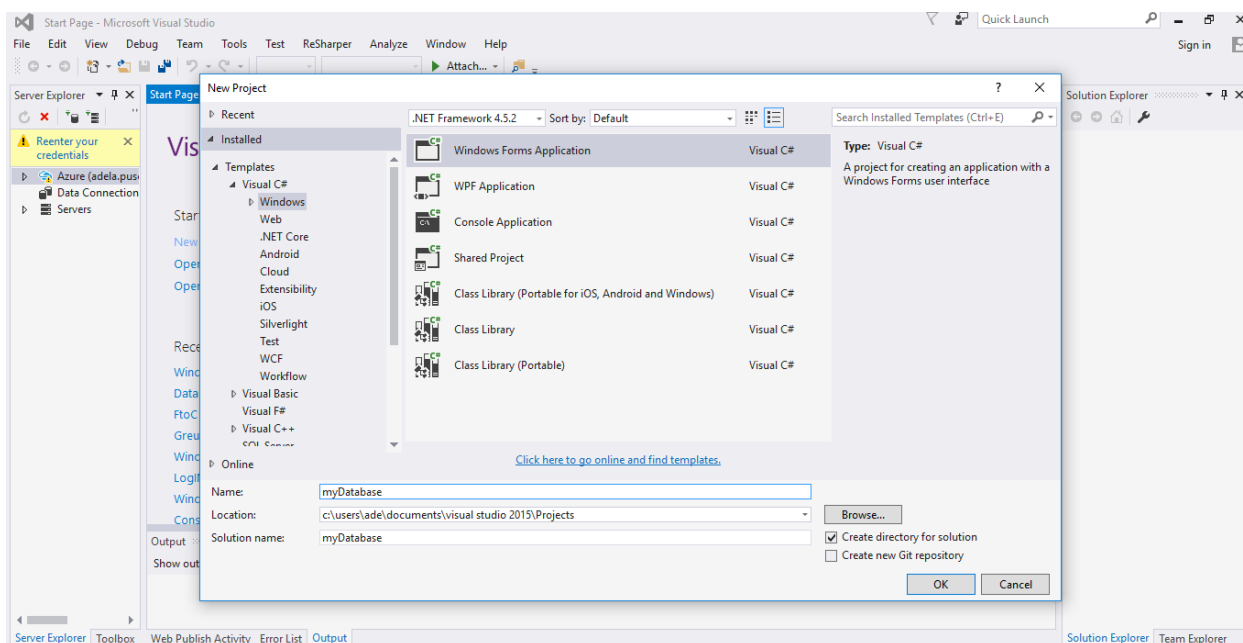


Fig. 3 Crearea unui proiect WFApp

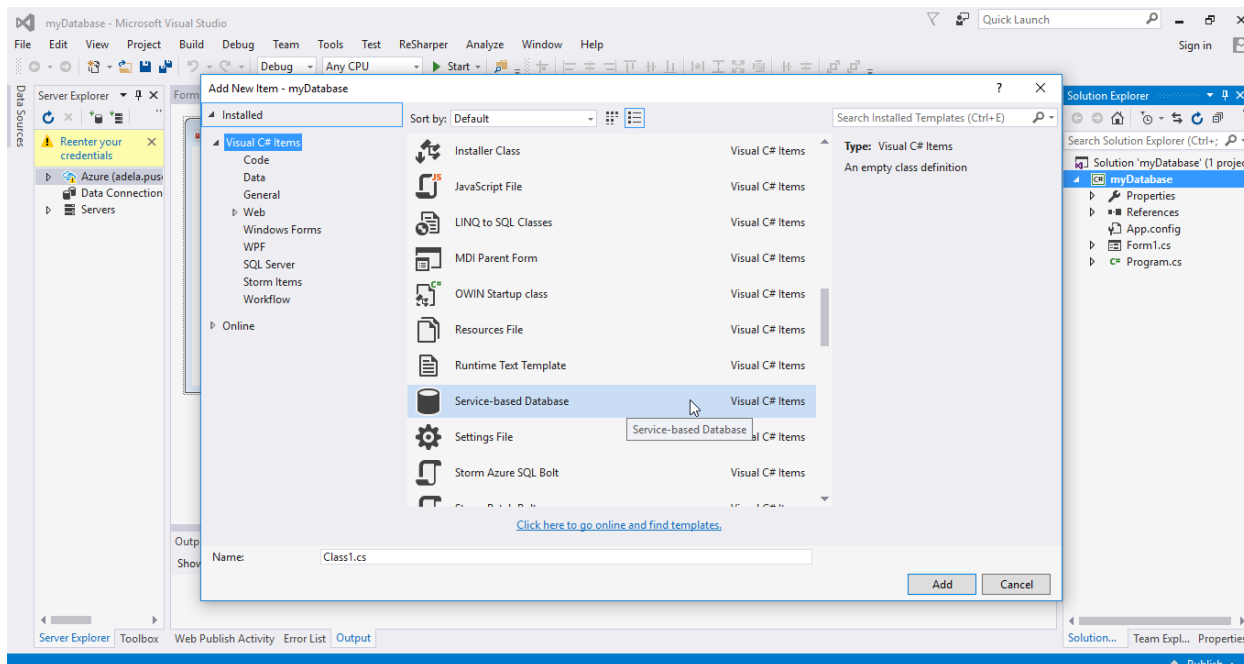


Fig. 4 Adaugare Service-based Database

După adăugarea bazei de date în proiect, acționând dublu click pe numele bazei de date din Solution Explorer, aceasta va apărea și în Server Explorer. Pentru a putea adăuga tabelele necesare se acționează click dreapta pe folderul Tables, precum în figura 5.

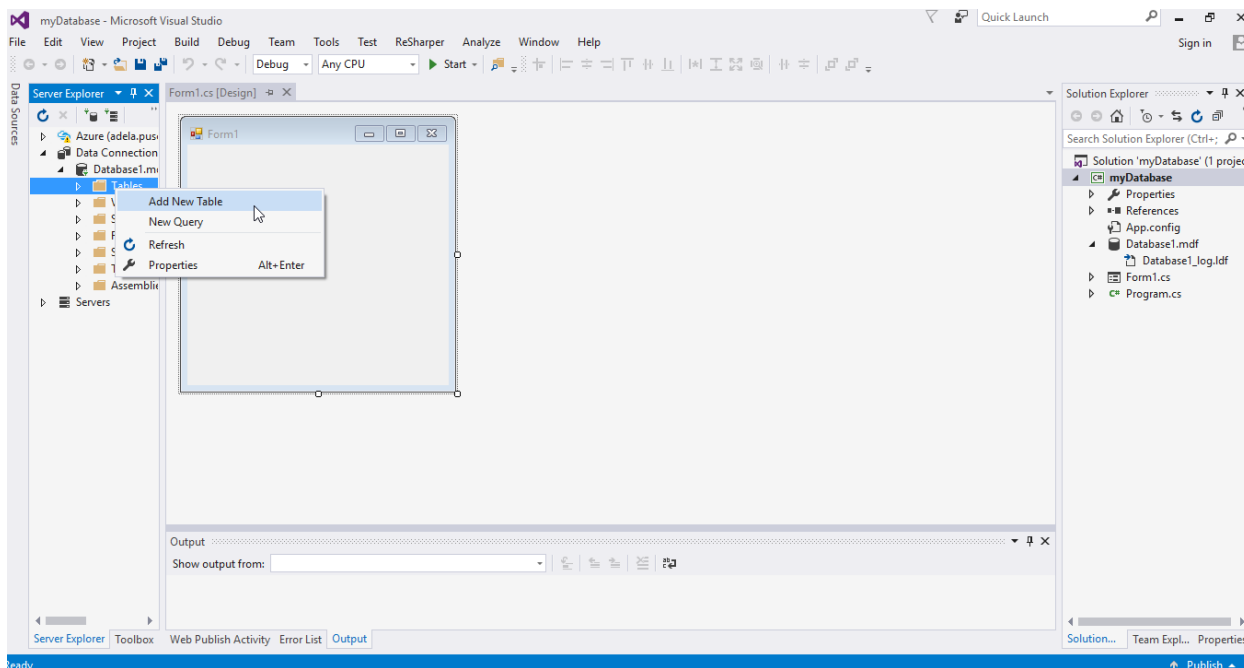


Fig. 5 Adaugare unei noi Tabele

În continuare vom adăuga doua tabele proiectului creat și anume : Tabela Universitati și tabela Facultati. Acestea vor avea următoarele câmpuri:

Tabela **Univeristati** (Fig. 6a): Id, NameUniv, City, Code

Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
NameUniv	text	<input type="checkbox"/>
City	text	<input type="checkbox"/>
Code	int	<input type="checkbox"/>

Fig. 6a Universitati

Tabela **Facultati** (Fig. 6b): Id, Code, NameFac

Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Code	int	<input type="checkbox"/>
NameFac	text	<input type="checkbox"/>

Fig. 6b Facultati

După cum putem observa câmpul **Code** apare în ambele tabele. Deoarece o Universitate poate avea mai multe Facultăți, vom seta câmpul Code din tabela Universitați cheie primară. În figura de mai jos putem observa cum se adăuga o cheie primară din Design, dar se poate adăuga și din partea de cod (T-SQL).

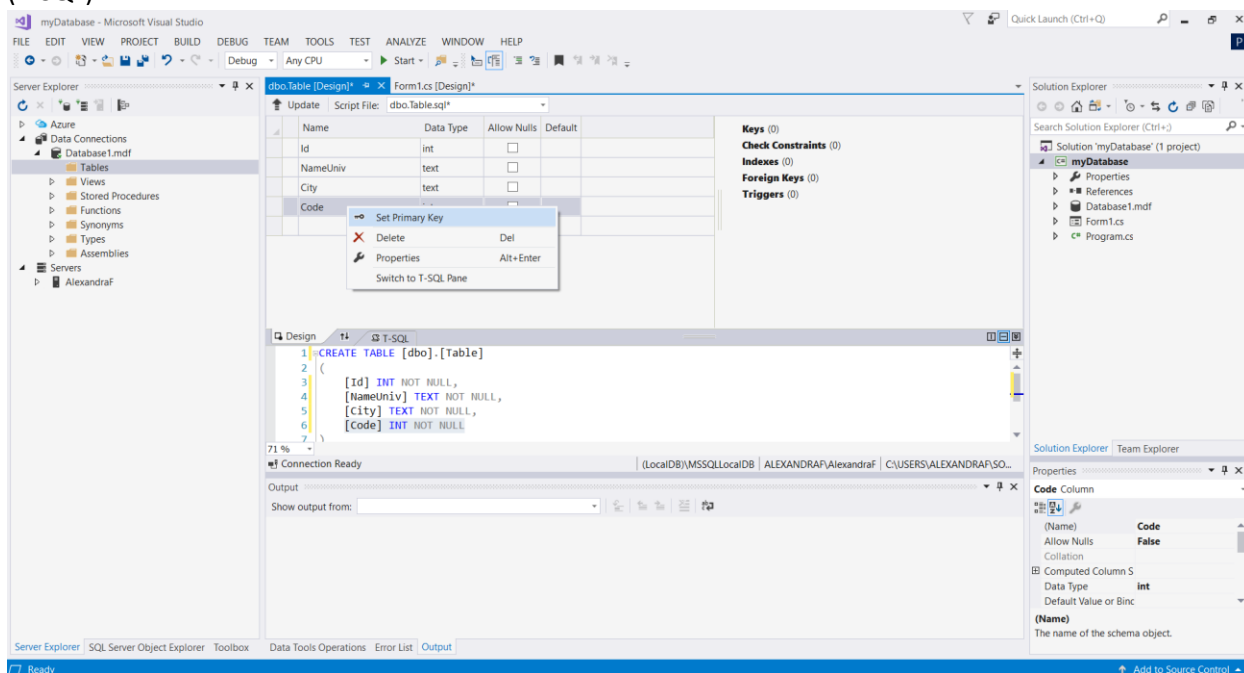



Fig. 7 Adăugarea unei chei primare

După definirea tabelului, din partea de cod (T-SQL) adăugam numele tabelului (Fig. 8), iar apoi la apăsarea butonului Update( Update) aceasta este adăugată bazei de date (Fig. 9).

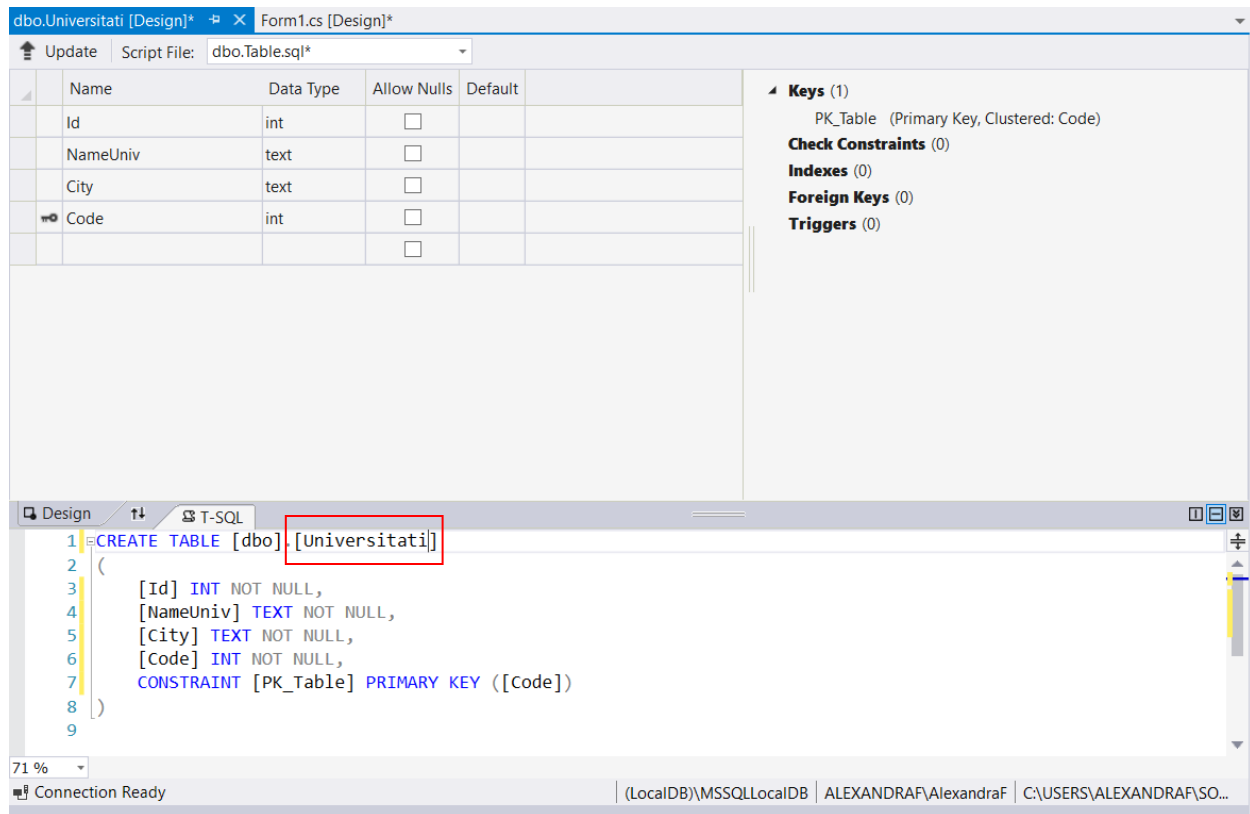


Fig. 8 Schimbarea numelui unei tabele

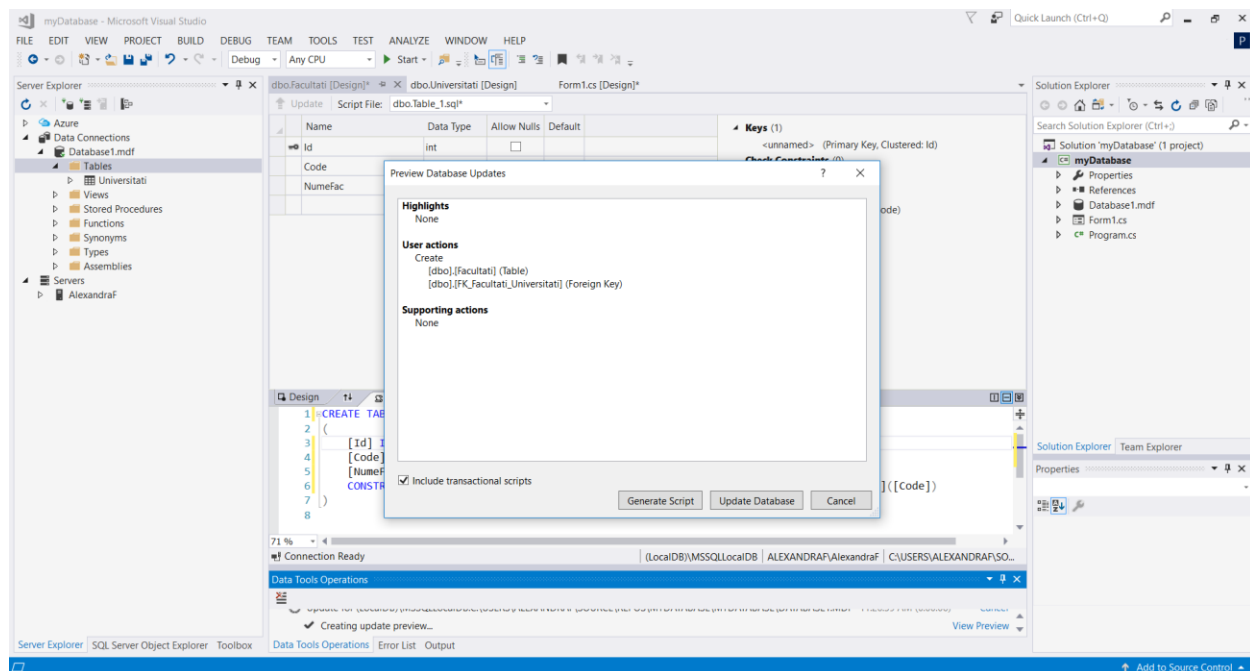


Fig. 9 Încărcarea tabelii în baza de date

În continuare vom seta câmpul Code din tabela Facultati ca și cheie straină pentru a face legătura dintre cele doua tabele, după cum putem observa în figura 10 și figura 11.

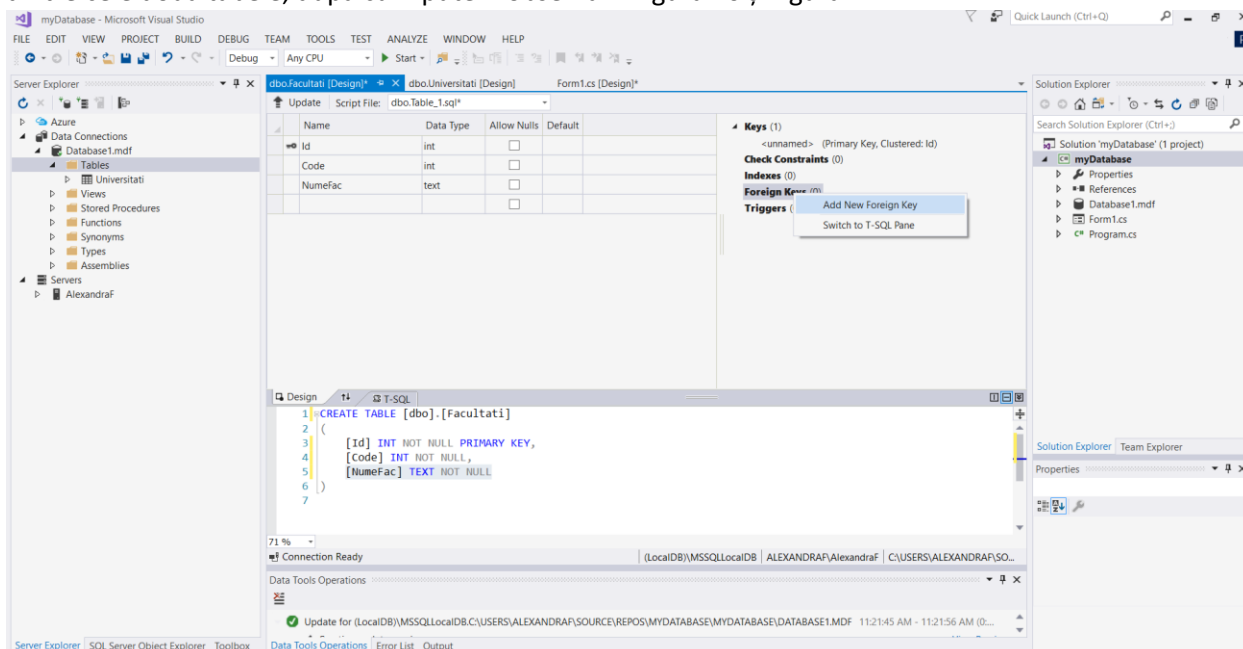


Fig. 10 Adăugarea unei chei straine

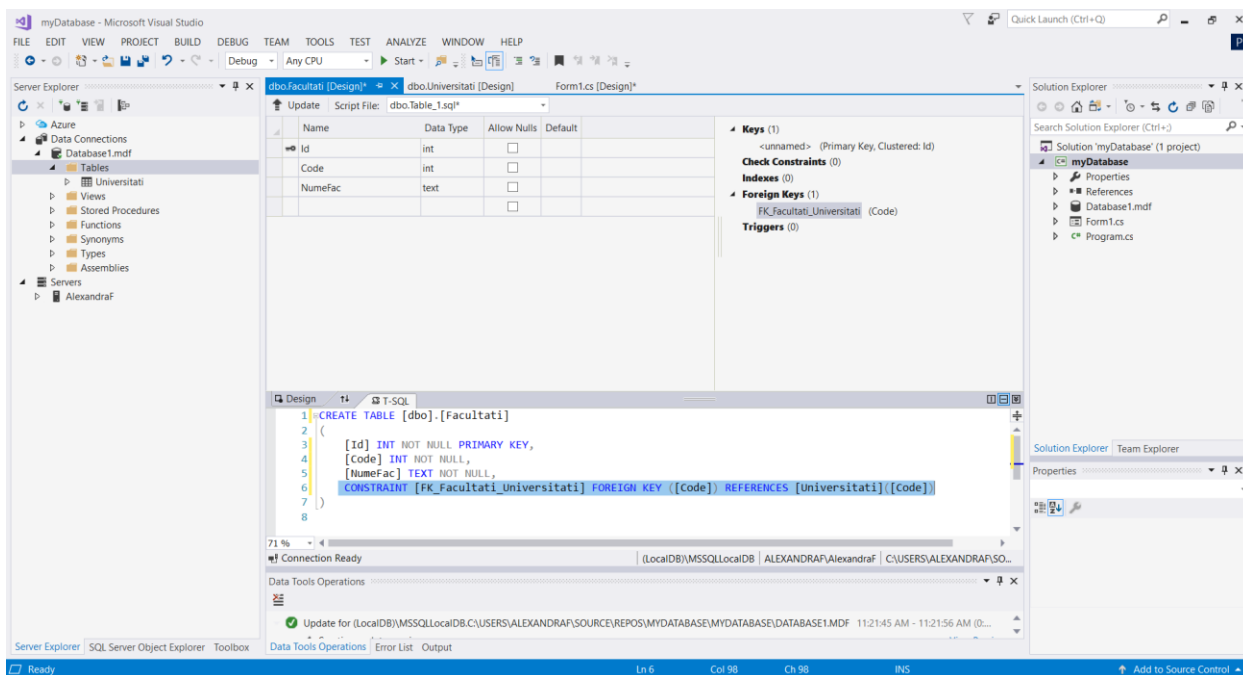


Fig. 11 Adăugarea unei chei straine

După proiectarea tabelor, putem să stocăm informațiile dorite. Acest lucru se poate face cu click dreapta pe tabela -> Show Table Data (Fig. 12).

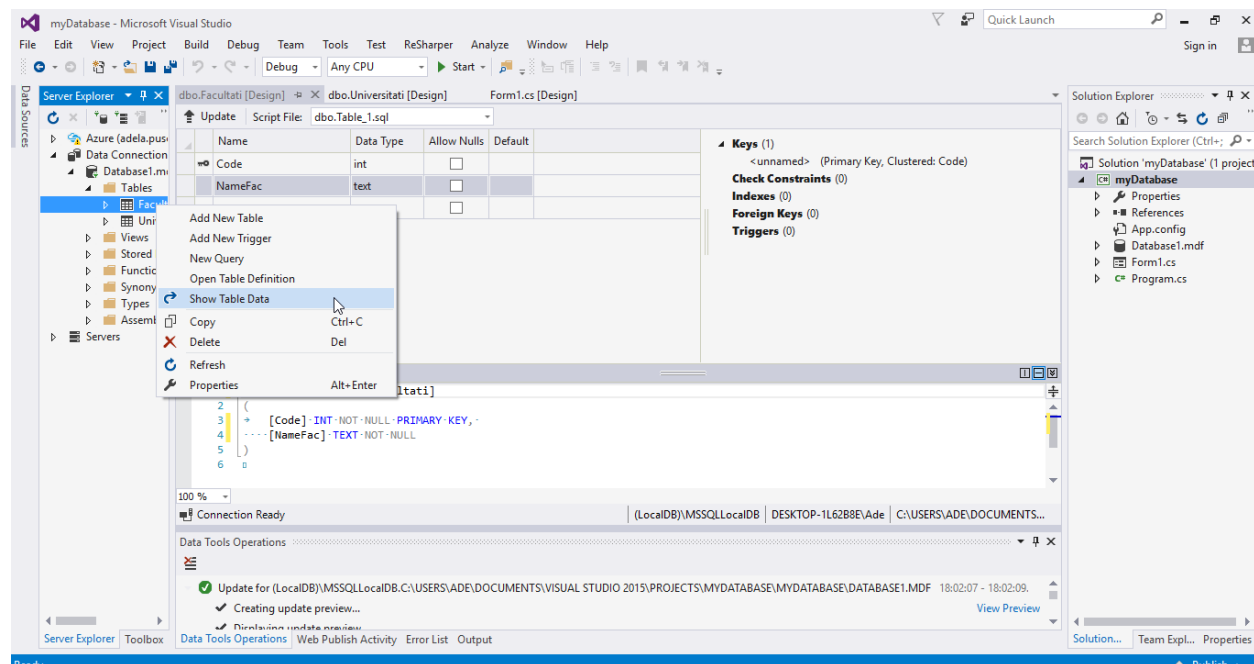


Fig. 12 Popularea unei tabele

Pentru a putea utiliza informațiile stocate trebuie să conectăm aplicație dezvoltată la baza de date. Acest lucru se poate face prin instanțierea clasei **SqlConnection**. Cu ajutorul proprietății **ConnectionString** specificăm calea de acces la baza de date, care se poate realiza prin două moduri:

- 1) **@ "calea"**

```
SqlConnection myCon = new SqlConnection();
myCon.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;
AttachDbFilename=|DataDirectory|\Database1.mdf;Integrated Security=True";
```
- 2) **"calea\calea\..."**

```
SqlConnection myCon = new SqlConnection();
myCon.ConnectionString = "Data Source=(LocalDB)\MSSQLLocalDB;
AttachDbFilename=|DataDirectory|Database1.mdf;Integrated
Security=True";
```

Pentru a accesa **ConnectionString** din **Project** selectăm **Add New Data Source...** și copiem stringul (Fig. 13).

Observatie!: Pentru a putea utiliza clasele specifice bazei de date trebuie să adăugăm:
using System.Data.SqlClient ;

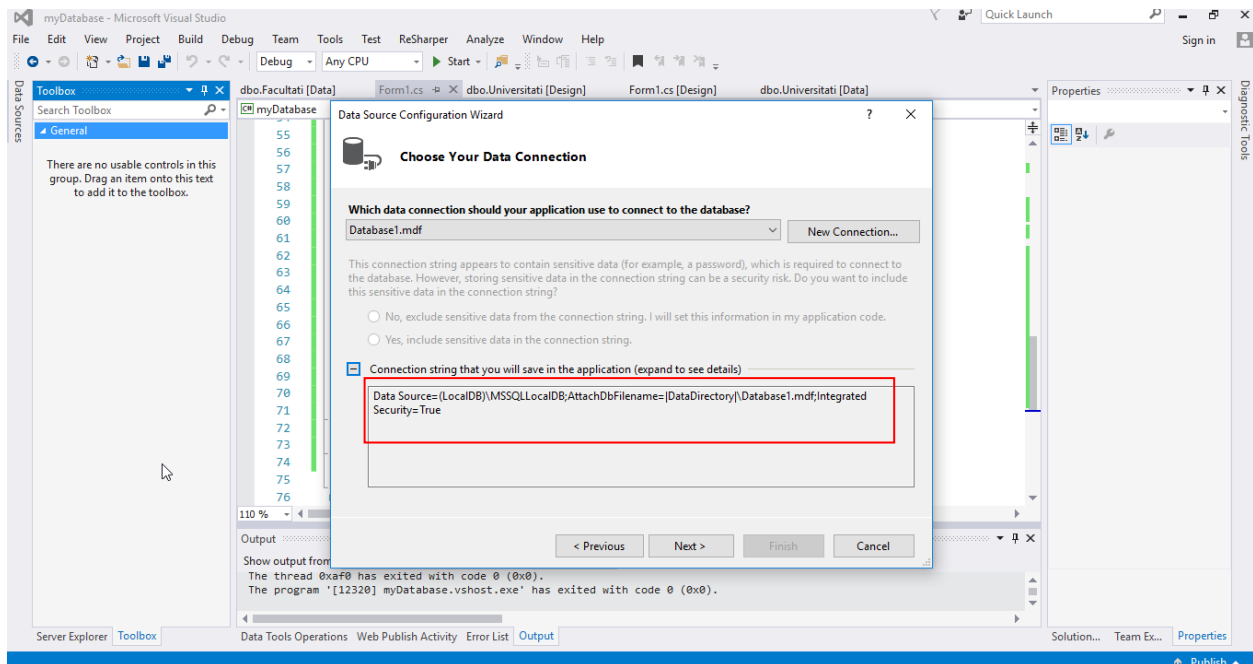
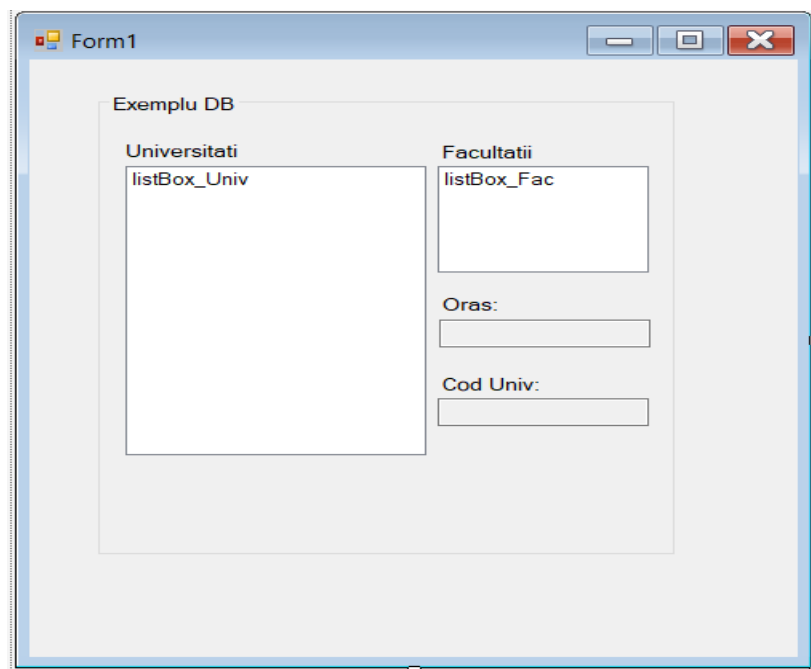


Fig. 13 Popularea unei tabele

Exemplu: Să se creeze o aplicație care să preia din baza de date, lista Universitatilor, și să se afișeze. Pentru fiecare Universitate selectată, se va afișa orașul, codul Universității și Facultățile aferente.



Observație!: Pentru a se salva informațiile introduse din aplicație, trebuie să setăm din proprietățile bazei de date : **Copy if newer**, precum în figura de mai jos:

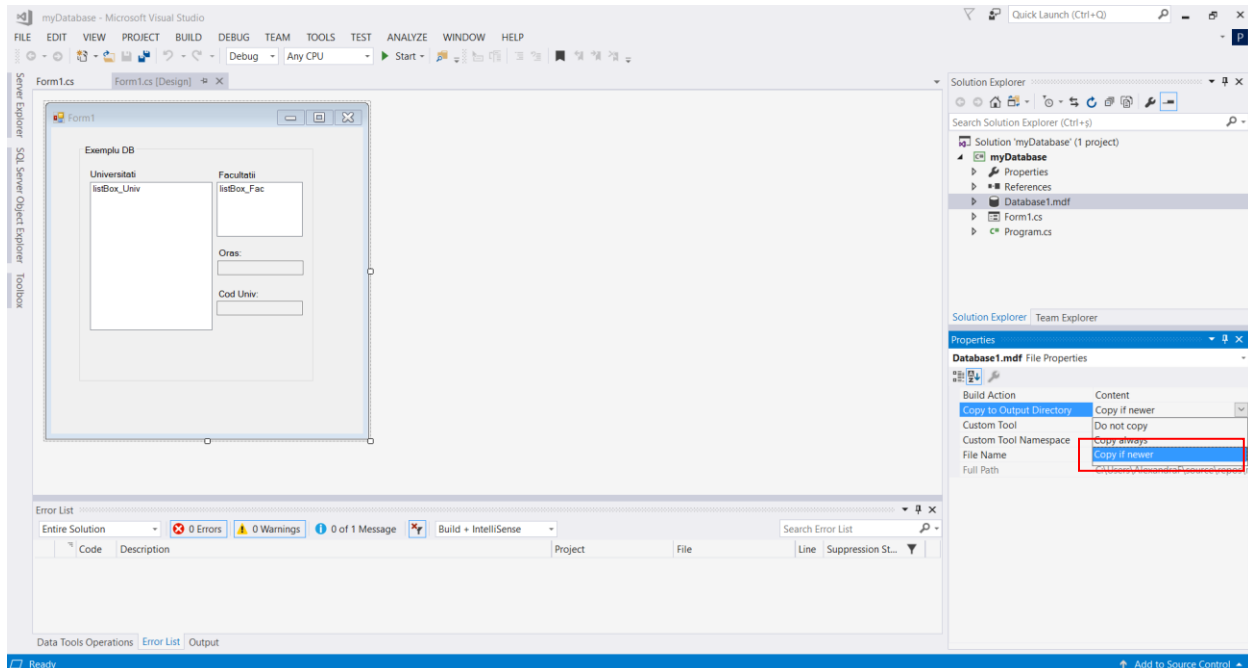


Fig. 14 Proprietăți - baza de date

```

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

namespace myDatabase
{
    public partial class Form1 : Form
    {
        SqlConnection myCon = new SqlConnection();
        DataSet dsUniv;
        DataSet dsFac;

        public Form1()
        {
            InitializeComponent();
            myCon.ConnectionString = @"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Database1.mdf;Integrated
Security=True";
            myCon.Open();
            dsUniv = new DataSet();
            dsFac = new DataSet();

            SqlDataAdapter daUniv = new SqlDataAdapter("SELECT * FROM Universitati", myCon);
            daUniv.Fill(dsUniv, "Universitati");
        }
    }
}

```

```

        SqlDataAdapter daFac = new SqlDataAdapter("SELECT * FROM Facultati", myCon);
        daFac.Fill(dsFac, "Facultati");
        foreach (DataRow dr in dsUniv.Tables["Universitati"].Rows)
        {
            String name = dr.ItemArray.GetValue(1).ToString();
            listBox_Univ.Items.Add(name);
        }
        myCon.Close();
    }

    private void listBox_Univ_SelectedIndexChanged(object sender, EventArgs e)
    {
        listBox_Fac.Items.Clear();
        textBox_City.Clear();
        int code = 0;
        String UnivSelected = listBox_Univ.SelectedItem.ToString();
        foreach (DataRow dr in dsUniv.Tables["Universitati"].Rows)
        {
            if (UnivSelected == dr.ItemArray.GetValue(1).ToString())
            {
                textBox_City.Text = dr.ItemArray.GetValue(2).ToString();
                code = Convert.ToInt16(dr.ItemArray.GetValue(3));
                textBox_CodeUniv.Text = code.ToString();
            }
        }

        foreach (DataRow dr in dsFac.Tables["Facultati"].Rows)
        {
            if (code == Convert.ToInt16(dr.ItemArray.GetValue(1)))
            {
                String nameFac = dr.ItemArray.GetValue(2).ToString();
                listBox_Fac.Items.Add(nameFac);
            }
        }
    }
}

```

3. Exerciții

1. Să se extindă aplicația precedentă, astfel încât să se poată efectua operații de inserare, ștergere și update pentru tabela Universitați. Informațiile să poată fi introduse de la tastatură.
2. Să se adauge în proiect un dataGridView cu ajutorul căruia să se introducă date în tabela Facultății, iar la selectarea unui rând acesta să poată fi șters.