**UNIVERSITATEA „ARTIFEX” DIN BUCUREŞTI**

**FACULTATEA DE FINANȚE și CONTABILITATE**

**CONTABILITATE ŞI INFORMATICĂ DE GESTIUNE**

**LUCRARE DE LICENȚĂ**

**Coordonator științific:**

Prof. Univ. Dr. Alexandru MANOLE

**Student**

Silviu Pompiliu SERIGEANU

**BUCUREŞTI, 2018**

**UNIVERSITATEA „ARTIFEX” DIN BUCUREŞTI**

**FACULTATEA DE FINANȚE și CONTABILITATE**

**CONTABILITATE ŞI INFORMATICĂ DE GESTIUNE**

**REALIZAREA UNUI CATALOG ELECTRONIC FOLOSIND O BAZĂ DE DATE SQL**

**Coordonator științific:**

Prof. Univ. Dr. Alexandru MANOLE

**Student**

Silviu Pompiliu SERIGEANU

**BUCUREŞTI, 2018**

CUPRINS

|  |  |
| --- | --- |
| 1. Introducere | 4 |
| 1. Proiectarea aplicației | 5 |
| * 1. Prezentarea tehnologiilor folosite | 5 |
| * 1. Proiectarea bazei de date | 5 |
| * 1. Prezentarea bazei de date | 8 |
| * 1. Diagrama UML | 14 |
| 1. Realizarea aplicației | 16 |
| * 1. Conectarea la baza de date | 16 |
| * 1. Realizarea interfeței grafice | 17 |
| * 1. Proceduri folosite | 18 |
| * + 1. Interogarea bazei de date și extragerea datelor | 18 |
| * + 1. Actualizarea datelor afișate | 21 |
| * + 1. Adăugarea, ștergerea și modificarea datelor din baza de date | 23 |
| * + 1. Resetarea câmpurilor modificabile | 25 |
| * + 1. Exportarea datelor | 26 |
| * + 1. Crearea listelor drop-down | 29 |
| * + 1. Crearea și ștergerea asocierilor | 30 |
| 1. Manual de utilizare | 38 |
| * 1. Popularea inițială și adăugarea de informații în baza de date | 38 |
| * 1. Modificarea informațiilor din baza de date | 38 |
| * 1. Ștergerea informațiilor din baza de date | 39 |
| * 1. Butonul Reset | 39 |
| * 1. Butonul Export | 39 |
| 1. Concluzii și dezvoltări ulterioare | 40 |
| 1. Bibliografie | 41 |

Capitolul 1

INTRODUCERE

Lucrarea de fata prezinta modul de realizare și funcționare al unei aplicații Windows care își propune organizarea unui catalog electronic, folosind o colecție de informații înregistrate într-o bază de date SQL.

Aplicația a fost realizată conectând o interfață grafica de tip Windows, la o bază de date SQL ce conține informații despre entitățile bazei de date. Aceasta combinație a fost aleasă deoarece legătura dintre interfață și baza de date se poate face cu ușurință folosind același set de instrumente ce se regăsește în mediul de dezvoltare Visual Studio.

În Capitolul 2 vor fi prezentate detaliile despre proiectarea aplicației, începând cu prezentarea tehnologiilor folosite apoi trecând prin procesul de proiectare a bazei de date, prezentarea elementelor bazei de date și încheind cu diagrama UML.

Capitolul 3 va descrie în detaliu realizarea efectivă a aplicației, urmărind conectarea la baza de date, procedurile folosite de-a lungul aplicației și realizarea interfeței grafice.

Capitolul 4 va conține instrucțiunile de utilizare a aplicației, iar în capitolul 5 vor fi prezentate concluziile și dezvoltările ulterioare.

Aceasta aplicație poate înlocui cu ușurință modelul clasic al cataloagelor folosite în sistemul

de învățământ și prezintă mai multe avantaje:

* Organizează datele într-o structura logica, ușor de înțeles
* Toate datele pot fi găsite într-un singur loc
* Asigura persistenta datelor înregistrate
* Economisește timp și spațiu pentru utilizatori
* Prezinta un grad ridicat de flexibilitate în modificare datelor existente

Funcționalități dorite:

* Înregistrarea cu ușurință a datelor specifice unui catalog (adăugare/modificare/ștergere)
* Organizarea tuturor listelor de entități folosite (studenți, profesori, materii etc.)
* Afișarea datelor într-o singura fereastra
* Exportarea datelor într-un format ușor de folosit

Capitolul 2

PROIECTAREA APLICATIEI

2.1 Prezentarea tehnologiilor folosite

Pentru gestionarea bazei de date s-a folosit limbajul **SQL** (Structured Query Language). Acces limbaj a apărut la începutul anilor ’70 fiind dezvoltat la IBM de către Donald D. Chamberlain și Raymond F. Boyce după studierea modelului relațional conceput de către Ted Codd în 1968. SQL are la baza tabele de date legate prin relații, iar din interogarea acestor tabele rezulta șiruri de date ce îndeplinesc cerințele interogării.

Pentru realizarea interfeței grafice s-a folosit sistemul grafic **WPF** (Windows Presentation Foundation) ca parte integranta a .NET Framework 3.0, iar pentru definirea elementelor grafice s-a folosit limbajul **XAML** (Extensible Application Markup Language). S-a ales aceasta varianta deoarece aplicațiile realizate în sistemul WPF sunt compatibile cu orice versiune de Windows, iar limbajul XAML permite un grad mare de personalizare a interfeței grafice.

Pentru legăturile dintre elementele grafice și baza de date SQL s-a folosit limbajul de programare **C#,** dezvoltat de Microsoft. Fiind un limbaj orientat pe obiect, permite definirea cu ușurință a claselor de obiecte folosite de către aplicație precum și construirea algoritmilor necesari rezolvării problemelor legate de interogarea bazelor de date. Toate aceste instrumente se regăsesc în mediul de programare Visual Studio. Versiunea folosita pentru realizarea aplicației este Microsoft Visual Studio Community 2017 15.7.2.

2.2 Proiectarea bazei de date

**Modelul relațional al unei baze de date**, formulat de Ted Codd, presupune existenta a trei elemente componente:

1. **Structuri de date**. Acestea sunt alcătuite din relații de date bidimensionale și se refera la:

1. **Atribute și domenii**. Plecând de la unul din avantajele fundamentale ale modelului relațional, omogenitatea, care presupune așezarea datelor în tabele cu linii și coloane care au același format, **atributul** reprezintă o caracteristica a unei entități din baza de date și este reprezentat de valoarea ce se regăsește la intersecția dintre o linie și o coloana, iar **domeniul** unui atribut reprezintă mulțimea finita de valori de același tip pe care le poate primi un atribut.
2. **Tuplul** reprezintă mulțimea de valori aflata pe o linie dintr-o relație tabelara, adică mulțimea valorilor atributelor atașate unei entități din baza de date.
3. **Relația** este o mulțime finita de tupluri și de regula ea reprezintă numele tabelei din baza de date. **Schema** relației reprezintă suma proprietăților comune ale tuplurilor conținute în relație.

2. **Restricțiile de integritate** reprezintă un set de cerințe ce trebuie îndeplinite pentru a asigura corectitudinea și coerența datelor din baza de date. **Cheia** unui tuplu reprezintă un identificator unic, reprezentat de valoarea unui atribut. în acest caz ea se mai numește și **cheie simplă**. în cazul în care identificatorul este format din mai multe atribute, atunci **cheia** este **compusă**. **Cheia externa** reprezintă un atribut dintr-o schema definit pe același domeniu ca și cheie primară a altei scheme.

3. **Operații asupra bazei de date**. Principalele modificări operate asupra unei baze de date presupun:

- **adăugarea** unui tuplu nou într-o relație a bazei de date

- **modificarea** valorilor atributelor unui tuplu existent într-o relație

- **ștergerea** unui tuplu dintr-o relație

Pentru proiectarea bazei de date a aplicației de față, se va lucra cu următoarele entități:

* Studenți
* Profesori
* Materii
* Specializări
* Grupe

,precum și cu înregistrări privind examenele (notă, dată, etc.) ce se vor regăsi sub schema **Catalog**.

**Lista de studenți** va stoca informațiile relevante despre studenți. Fiecare student va avea un număr matricol care va fi un identificator (cheie) unic, nume, prenume și posibilitatea de a fi asociat unei grupe de studenți. Pot fi adăugați studenți în aceasta tabelă, fără a fi asociați cu o grupă, presupunând ca sunt înscriși de curând și nu au fost încă alocați unei grupe sau nu s-a creat încă grupa unde vor fi alocați. Un student poate fi alocat unei singure grupe, dar o grupa poate conține mai mulți studenți (asociere de tipul 1:M).

**Lista de profesori** va conține informațiile relevante despre profesori. Fiecare profesor va avea un număr de identificare (cheie) unic, nume, prenume și materiile pe care le preda. Se vor putea adăuga profesori fără a menționa ce materii vor preda, aceasta informație putând fi adăugată ulterior.

**Tabela de materii** va conține o lista de nume de materii, un identificator unic pentru fiecare materie, precum și specializarea sau specializările asociate. Se vor putea adăuga materii fără a menționa cărei specializare ii este asociată, aceasta informație putând fi adăugată ulterior.

Informații similare for fi stocate și în **lista de specializări**:nume specializare și identificator unic.

**Lista de grupe** trebuie sa conțină numele fiecărei grupe, id-ul grupei (care va fi și identificator unic) și legătura cu o specializare. O grupa va putea fi asociată doar cu o specializare dar o specializare poate avea asociate mai multe grupe (asociere de tipul 1:M). Nu se va putea crea o grupa fără ca aceasta să fie asociată cu o specializare.

Pentru a face legătura intre materii și profesori, adică ce materii predă fiecare profesor, vom crea tabela de legătură **Materii-Profesori**, ce va conține pe lângă un id unic de identificare și id-urile profesorului și a materiei care vor fi asociate. Unui profesor i se vor putea asocia mai multe materii (de exemplu un profesor poate preda atât limba Engleză cat și limba Franceză), iar o materie va putea fi asociata cu mai mulți profesori (asociere de tipul M:N).

Legătura dintre materii și specializări, adică ce materii se vor parcurge la fiecare specializarese va face folosind tabela **Materii-Specializări.** Drept urmare, o materie se va putea asocia cu mai multe specializări (de exemplu materia limba Engleză poate fi predată la toate specializările facultății), iar unei specializări i se vor putea asocia mai multe materii (asociere de tipul M:N).

Cea mai importanta tabela va fi cea de **Catalog** care va face legăturaîntre majoritateatabelelor deja existente și va conține înregistrări specifice examenelor: data, numele și prenumele studentului, materia, nota, numele și prenumele profesorului precum și un identificator unic pentru fiecare înregistrare din tabela.

Identificatorul unic din fiecare din cele 6 tabele va fi folosit drept cheie primara în tabela proprie, dar vor fi și chei externe pentru realizarea legăturilor dintre entități. Un alt avantaj pentru folosirea de chei primare în fiecare tabela este și selectarea și ștergerea/modificarea cu ușurință a înregistrărilor din tabele.

2.3 Prezentarea bazei de date

Folosind datele proiectate în capitolul anterior, s-au creat următoarele tabele cu atributele respective.

Tabela **ListaStudenti** cu atributele:

* **NumarMatricol**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **NumeStudent**
  + Tip de data *nvarchar*
  + Lungime maxima 50
  + Nu accepta valori nule
* **PrenumeStudent**
  + Tip de data *nvarchar*
  + Lungime maxima 50
  + Nu accepta valori nule
* **IdGrupa**
  + Tip de data *int*
  + Este cheie străină din tabela **ListaGrupe**
  + Permite valori nule
  + Face legătura intre un student și o grupă
  + Unui student i se poate asocia o singură grupă

Tabela **ListaProfesori** cu atributele:

* **IdProfesor**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **NumeProfesor**
  + Tip de data *nvarchar*
  + Lungime maxima 50
  + Nu accepta valori nule
* **PrenumeProfesor**
  + Tip de data *nvarchar*
  + Lungime maxima 50
  + Nu accepta valori nule

Tabela **ListaMaterii** cu atributele:

* **IdMaterie**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **NumeMaterie**
  + Tip de data *nvarchar*
  + Lungime maxima 50
  + Nu accepta valori nule

Tabela **ListaSpecializari** cu atributele:

* **IdSpecializare**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **NumeSpecializare**
  + Tip de data *nvarchar*
  + Lungime maxima 50
  + Nu accepta valori nule

Tabela **ListaGrupe** cu atributele:

* **IdGrupa**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **NumeGrupa**
  + Tip de data *nvarchar*
  + Lungime maxima 50
  + Nu accepta valori nule
* **IdSpecializare**
  + Tip de data *int*
  + Este cheie străină din tabela **ListaSpecializari**
  + Nu accepta valori nule
  + Face legătura intre o grupă și o specializare
  + Unei grupe i se poate asocia o singură specializare

Tabela **MateriiProfesori** cu atributele:

* **ID**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **IdMaterie**
  + Tip de data *int*
  + Este cheie străină din tabela **ListaMaterii**
  + Nu accepta valori nule
  + Face legătura intre o materie și un profesor
  + Unei materii i se pot asocia mai mulți profesori
* **IdProfesor**
  + Tip de data *int*
  + Este cheie străină din tabela **ListaProfesori**
  + Nu accepta valori nule
  + Face legătura intre un profesor și o materie
  + Unui profesor i se pot asocia mai multe materii

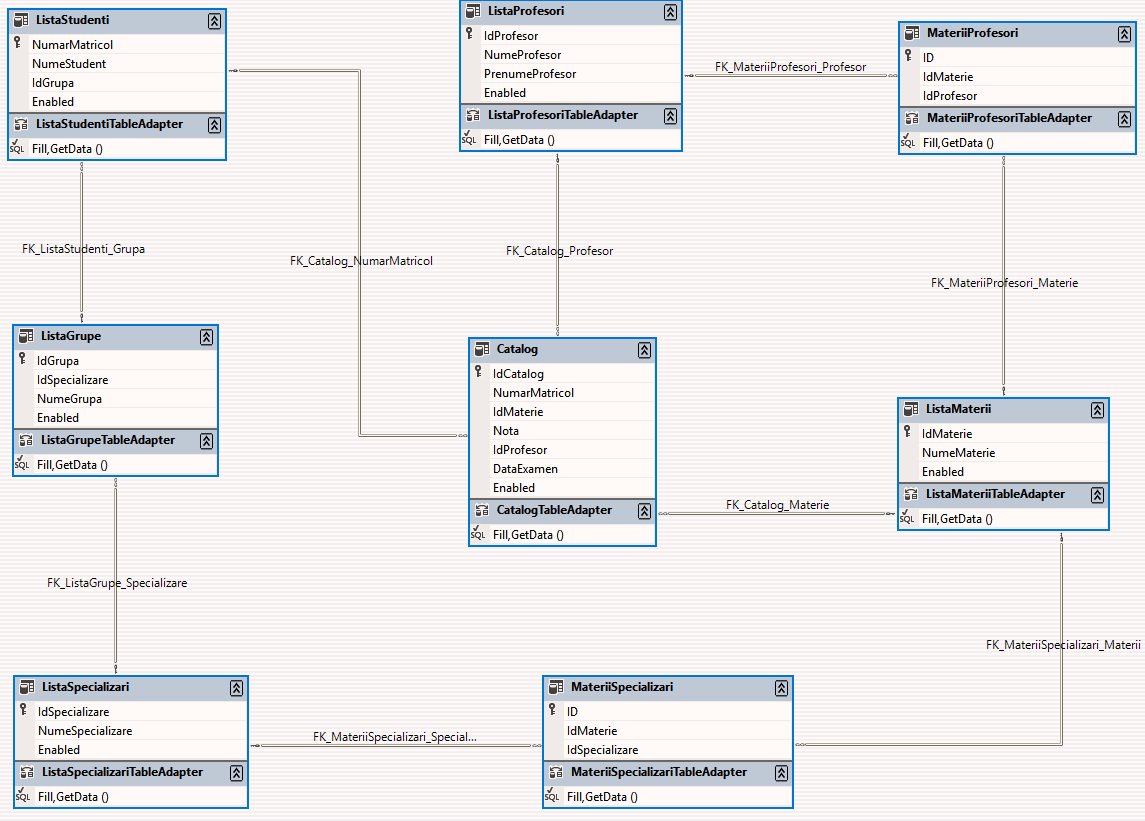
Tabela **MateriiSpecializari** cu atributele:

* **ID**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **IdMaterie**
  + Tip de data *int*
  + Este cheie străină din tabela **ListaMaterii**
  + Nu accepta valori nule
  + Face legătura intre o materie și un profesor
  + Unei materii i se pot asocia mai multe specializări
* **IdSpecializare**
  + Tip de data *int*
  + Este cheie străină din tabela **ListaSpecializari**
  + Nu accepta valori nule
  + Face legătura intre o specializare și o materie
  + Unei specializări i se pot asocia mai multe materii

Tabela **Catalog** cu atributele:

* **IdCatalog**
  + Tip de data *int*
  + Este cheie primara
  + Nu accepta valori nule
  + Valoare inițială = 1
  + Valoare este crescuta cu 1 automat, la crearea unui tuplu nou
  + Valoare nu poate fi modificata de utilizator
* **NumarMatricol**
  + Tip de data *int*
  + Este cheie externa din tabela **ListaStudenti**
  + Nu accepta valori nule
  + Definește pentru ce student s-a înregistrat nota din catalog
* **IdMaterie**
  + Tip de data *int*
  + Este cheie externa din tabela **ListaMaterii**
  + Nu accepta valori nule
  + Definește pentru ce materie s-a înregistrat nota din catalog
* **Nota**
  + Tip de data *int*
  + Accepta valori intregi de la 1 la 10
  + Nu accepta valori nule
  + Definește nota asociata înregistrării din catalog
* **IdProfesor**
  + Tip de data *int*
  + Este cheie externa din tabela **ListaProfesori**
  + Nu accepta valori nule
  + Definește profesorul care a acordat nota ce s-a înregistrat în catalog
* **DataExamen**
  + Tip de data *date*
  + Nu accepta valori nule
  + Definește data examenului înregistrat în catalog

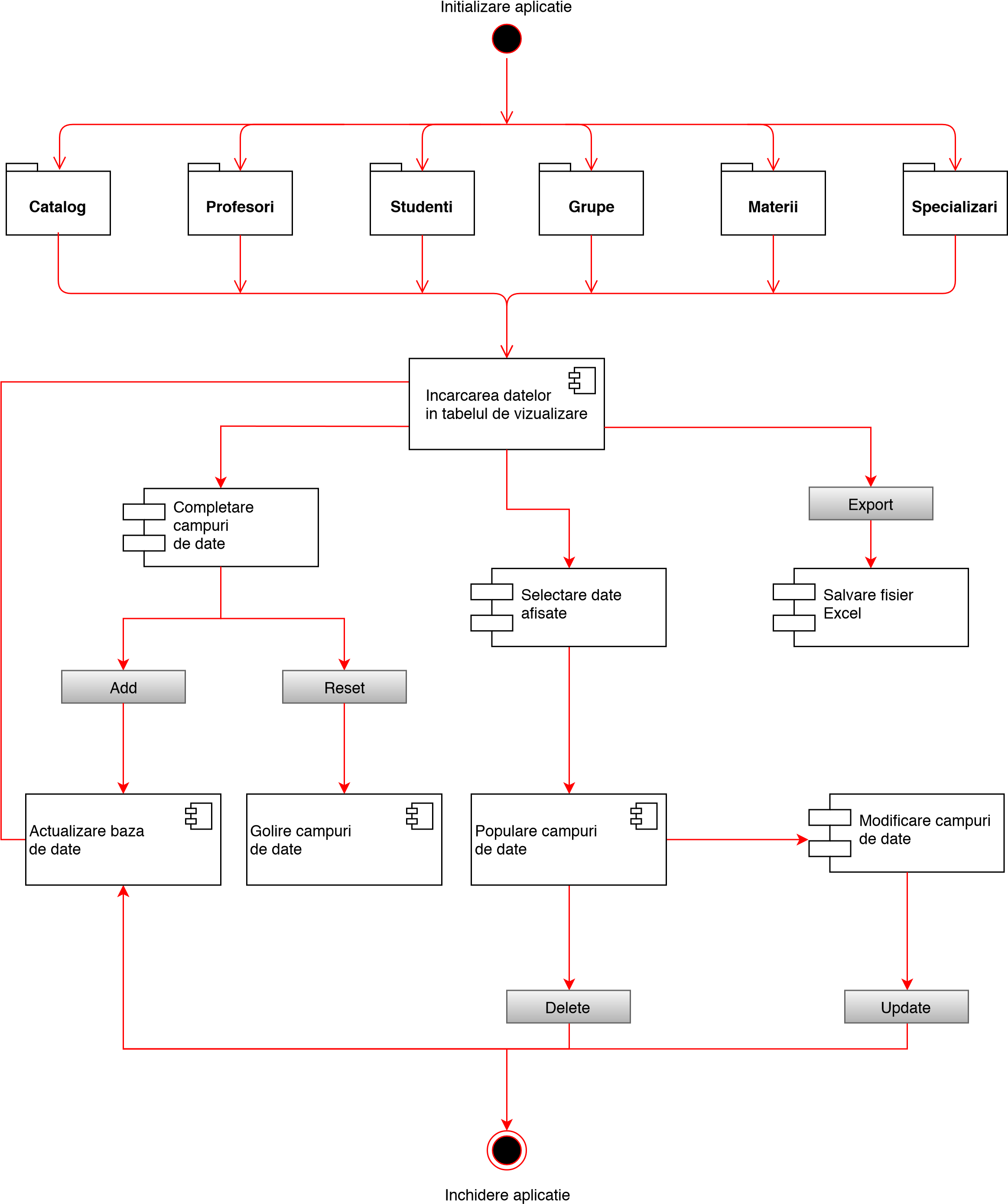
Legăturile dintre tabele se pot observa cu ușurință și în figura nr.1:

****

**Figura nr.1 Legăturile dintre tabelele bazei de date**

**2.4 Diagrama UML**

Diagrama UML (**Unified Modeling Language**) prezinta acțiunile pe care le poate executa utilizatorul în cadrul acestei aplicații și modul în care aplicația prelucrează datele.



La pornirea aplicației este încărcata fereastra principală și tab-ul **Catalog** este afișat (vezi figura nr.2 Elementele interfeței grafice), iar datele specifice sunt afișate în secțiunea 1 a interfeței.

Oricare din celelalte tab-uri (**Studenți**, **Profesori** etc.) sunt disponibile pentru a fi selectate din secțiunea 2.

La încărcarea celor 6 tab-uri, informațiile specifice sunt automat extrase din baza de date și afișate în secțiunea 1.

Pentru toate tab-urile se păstrează elementele 1 și 4 ale interfeței grafice, diferențele apărând în secțiunea 3 unde sunt afișate câmpurile specifice entității tab-ului selectat (de ex. pentru **Profesori** vom avea afișate câmpurile: *Nume profesor*, *Prenume profesor*, *Materie* și *Id profesor*; iar pentru **Studenți**: *Nume student*, *Prenume student*, *Număr matricol* și *Grupa*) .

După încărcarea datelor în tabelul de vizualizare, utilizatorul are 3 variante de a continua:

* Exportarea datelor afișate, într-un fișier Excel, folosind butonul
* Selectarea unei linii de date din tabelul afișat în elementul 1.
  + După selectare, câmpurile din secțiunea 3 sunt populate automat cu datele înregistrării selectate
  + Aceste date pot fi modificate și apoi trimise către baza de date pentru a fi actualizate prin intermediul butonului
  + sau înregistrarea aleasa poate fi ștearsa cu ajutorul butonului
* Completarea câmpurilor cu datele specifice tab-ului curent
  + După completare, datele pot fi trimise către baza de date pentru actualizare, folosind butonul
  + sau datele completate pot fi șterse cu ușurință cu ajutorul butonului

acesta golind concomitent toate câmpurile completate. Acest buton poate fi folosit și pentru a goli câmpurile populate după selectarea unei înregistrări din tabelul de vizualizare a datelor

După fiecare actualizare a bazei de date, informațiile sunt reîncărcate în tabelele din secțiunea 1 a interfeței, din fiecare tab.

In orice moment, aplicația poate fi oprita folosind butonul de închidere

Capitolul 3

REALIZAREA APLICAȚIEI

**3.1 Conectarea la baza de date**

Pentru realizare aplicației Windows s-a folosit limbajul C# și primul pas a fost definirea modalității de conectare la baza de date creată în capitolul anterior. în acest sens s-a definit variabila ConnStringde tip *ConnectionString* cu valoarea:Data Source=(LocalDB)\MSSQLLocalDB; AttachDbFilename=|DataDirectory|\PriectDiploma.mdf; IntegratedSecurity=True; ConnectTimeout=30, unde Data Source reprezinta locația bazei de date, AttachDbFilename este numele bazei de date, IntegratedSecurity este metoda de securitate folosită pentru autentificare la baza de date și ConnectTimeout este durata pentru care se va relua conectarea la baza de date în cazul unei încercări eșuate.

Aceasta variabilă va fi utilă de fiecare dată când conexiunea la baza de date SQL va fi deschisă, folosind codul:

SqlConnection conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

.

.

conn.Close();

,cu alte cuvinte se va declara și instanția o noua variabila conn de tipul SqlConnection și i se va atribui proprietății conn.ConnectionString valoarea stocată în ConnString. Baza de date se va deschide folosind funcția conn.Open(), iar după efectuarea operațiilor se va închide cu conn.Close().

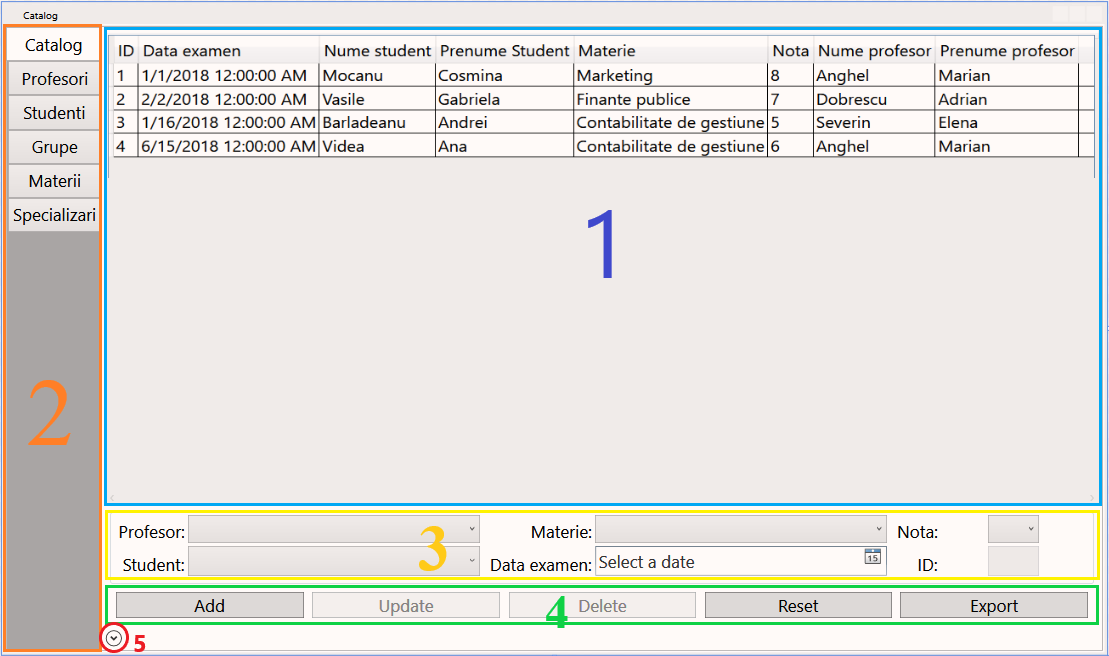
După fiecare deschidere a conexiunii către baza de date este foarte importantă și închiderea acesteia pentru a preveni accesul neautorizat la datele existente.

Accesul la baza de date este realizat de fiecare data când au loc următoarele operații:

* Adăugarea unei informații noi în baza de date (student nou, profesor nou etc.)
* Modificarea unei informații existente (schimbarea unui nume, corectarea unei note etc.)
* Ștergerea unei informații existente (ștergerea unei înregistrări eronate, a duplicatelor etc.)

**3.2 Realizarea interfeței grafice**

Interfața grafică a fost concepută astfel încât sa cuprindă toate informațiile într-o singură fereastră, fără a fi nevoie sa se navigheze printr-o structură de ferestre succesive.



**Figura nr. 2 Elementele interfeței grafice**

In figura nr.2 se pot observa principalele elemente ale interfeței grafice:

1. Primul element reprezinta zona (tabelul) de vizualizare a datelor, după ce acestea au fost încărcate din baza de date.
2. Din coloana din stânga se poate alege entitatea (tab-ul) cu care se va lucra. în funcție de alegerea făcută, se vor afișa informațiile specifice entității în tabelul (1) și în câmpurile de manipulare a datelor (3).
3. Câmpurile de manipulare a datelor conțin atributele specifice fiecărei entități de lucru.
4. Butoanele de interacțiune permit efectuarea operațiilor asupra bazei de date precum și exportul informațiilor. Aceste butoane sunt identice pentru fiecare tab.
5. Panourile (3) și (4) pot fi ascunse folosind butonul (5) pentru a permite o mai buna vizualizare a datelor. Ulterior panourile poate fi afișate folosind același buton (5).

Fereastra principala este un obiect de tip *Grid*, în care sunt amplasate toate obiectele interfeței.

In interiorul acestui *Grid* este un obiect de tip *TabControl* (Figura 2, elementul 2), aliniat la marginea din stânga a ferestrei, care conține mai multe obiecte de tip *TabItem*, ordonate de sus în jos*.* Fiecare dintre aceste obiecte stochează informațiile privind entitățile bazei de date (Catalog, Profesori, Studenți etc.).

In fiecare obiect *TabItem* exista un element de tip *DockPanel* ce conține la rândul sau două obiecte:

* Un *Expander* în partea de jos unde sunt aranjate toate butoanele și listele din secțiunile (3) și (4) conform figurii 2. Folosind butonul (5) aceste liste și butoane pot fi ascunse pentru a permite o vizualizare mai cuprinzătoare asupra elementului *DataGrid*
* Un *DataGrid* (1) unde sunt afișate datele preluate din baza de date sub formă de linii și coloane.

**3.3 Proceduri folosite**

**3.3.1 Procedura de interogare a bazei de date și extragerea datelor**

Principala procedura folosita în întreaga aplicație este cea de conectare la baza de date, interogare și obținerea unor informații. Acest lucru se realizează prin intermediul unei variabile de tip *SqlCommand* căruia i se trimit doi parametri de tipul *string* și *SqlConnection.* Parametrul de tip *string* conține instrucțiunile în limbajul SQL, pentru interogarea bazei de date, iar cel de tip *SqlConnection* conține datele necesare conectării la baza de date. Datele rezultate după interogare sunt apoi stocate în variabila de tip *SqlDataReader*.

Aceasta procedura este folosită în următoarele cazuri:

1. Crearea listelor de entități necesare popularii câmpurilor de tip drop-down din tab-urile:
   * Catalog
     + Lista de profesori cuprinsa în cbxCatalogProfesor.Items
     + Lista de studenți cuprinsa în cbxCatalogStudent.Items
     + Lista de materii cbxCatalogMaterie.Items
   * Profesori
     + Lista de materii ce pot fi asociate cu profesorii, cuprinsa în cbxProfesorMaterii.Items
   * Studenți
     + Lista de grupe ce pot fi asociate cu studenții, cuprinsa în cbxStuGrupa.Items
   * Grupe
     + Lista de specializări de pot fi asociate cu fiecare grupa, cuprinsa în cbxGruSpecializare.Items
   * Materii
     + Lista de specializări ce pot fi asociate cu fiecare materie, cuprinsa în cbxMatSpecializare.Items
2. Popularea tabelului (1) de tip *DataGrid* cu informațiile specifice fiecărui tab.
3. Modificarea conținutului bazei de date (adăugarea, actualizarea sau ștergerea informațiilor)
4. Actualizarea tabelului (1) de fiecare data când au loc modificări ale conținuturi bazei de date

Exemplu pentru crearea listei de profesori ce este afișată în drop-down-ul din tab-ul Catalog:

private void creazaListaProfesori()

{

SqlConnection conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

string query = "SELECT \* FROM ListaProfesori ORDER BY NumeProfesor, PrenumeProfesor ";

SqlCommand cmd = new SqlCommand(query, conn);

SqlDataReader dataReader = cmd.ExecuteReader();

string s;

cbxCatalogProfesor.Items.Clear();

while (dataReader.Read())

{

s = dataReader["NumeProfesor"].ToString() + " " + dataReader["PrenumeProfesor"].ToString();

cbxCatalogProfesor.Items.Add(s);

valoriIdProfesori.Add(Convert.ToInt32(dataReader["IdProfesor"].ToString()));

}

dataReader.Close();

conn.Close();

}

Urmărind pas cu pas exemplul de mai sus au loc următoarele etape la apelarea metodei creazaListaProfesori():

* + Declararea și instanțierea unei variabile conn de tip *SqlConnection*
  + Se atribuie valoarea stocata în Properties.Settings.Default.ConnString proprietății conn.ConnectionString
  + Deschiderea conexiunii către baza de date apelând conn.Open()
  + Declararea unei variabile query de tip *string și* instanțierea ei cu valoarea "SELECT \* FROM ListaProfesori ORDER BY NumeProfesor, PrenumeProfesor " care în limbaj SQL interoghează tabela ListaProfesori din baza de date și extrage toate informațiile (Id, Nume, Prenume) despre profesorii existenți, ordonându-i mai întâi după NumeProfesor apoi după PrenumeProfesor
  + Declararea și instantierea unei variabile cmd de tip *SqlCommand* care primește parametrii query și conn, definiți mai sus. Aceasta comanda folosește datele definite de parametrul conn ca sa se conecteze la baza de date și datele din parametrul query ca sa extragă informații
  + Declararea unei variabile dataReader de tip *SqlDataReader* în care vom stoca informațiile obținute din baza de date, prin trimiterea comenzii cmd
  + Declararea unei variabile s de tip *string* în care vom stoca valorile ce vor fi adăugate în lista cbxCatalogProfesor.Items
  + Golirea listei cbxCatalogProfesor.Items, pentru a evita dublarea datelor
  + Folosind o bucla de tip *while* se urmărește fiecare înregistrare (linie) din dataReader și în variabila s se atribuie valorile NumeProfesor și PrenumeProfesor, de pe linia respectiva, alipite
  + In lista cbxCatalogProfesor.Items se adaugă valoarea lui s
  + Id-ul profesorului adăugat în lista de mai sus se adaugă într-o lista diferita (valoriIdProfesori). Aceasta lista va ajuta mai târziu în a asocia profesorii cu materiile pe care le pot preda.
  + Se trece la următoarea înregistrare (linie) din bucla *while*
  + Când toate liniile au fost parcurse se iese din bucla
  + dataReader-ul se închide
  + Conexiunea conn se închide

Aceasta metoda este similară pentru toate celelalte entități, singura diferență fiind tabela din baza de date din care se extrag informațiile și lista unde vor fi stocate aceste informații.

**3.3.2 Procedura de actualizare a tabelului (1) de tip DataGrid**

Aceasta funcție updateDataGrid() este apelata în următoarele cazuri pentru fiecare tab:

* + La inițializarea aplicației când sunt încărcate cele 6 tab-uri
  + La modificarea conținutului undei tabele din baza de date
    - Adăugare
    - Modificare
    - Ștergere
  + La modificarea asocierii dintre doua entități
    - Profesori-Materii
    - Materii-Specializări

Exemplu pentru popularea tabelului de date din tab-ul Studenți:

private void updateDataGrid\_Studenti()

{

SqlConnection conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

string query = "SELECT NumarMatricol as [Numar matricol], NumeStudent AS Nume, PrenumeStudent AS Prenume, NumeGrupa AS Grupa FROM ListaStudenti LEFT JOIN ListaGrupe ON ListaGrupe.IdGrupa=ListaStudenti.IdGrupa ORDER BY NumeStudent";

SqlCommand cmd = new SqlCommand(query, conn);

SqlDataReader dataReader = cmd.ExecuteReader();

DataTable dataTable = new DataTable();

dataTable.Load(dataReader);

dgr\_studenti.ItemsSource = dataTable.DefaultView;

dataReader.Close();

conn.Close();

}

Pasii urmați sunt similari cu metoda creazaListaProfesori(), diferențele apar la datele extrase din baza de date și la folosirea lor:

* + Pentru a interoga baza de date se va construi *string*-ul querry care va extrage numărul matricol, numele și prenumele studenților din tabela ListaStudenti, precum și grupa din tabela ListaGrupe, folosind asocierea făcută pe baza cheii de legătura IdGrupa.
  + Pentru a folosi informațiile preluate de dataReader, se va declara și instanția o variabila dataTable de tip *DataTable*, pe care o vom încărca cu datele din Reader dataTable.Load(dataReader);
  + Apoi obiectul de tip *DataGrid* preia și afișează aceste informații din dataTable

dgr\_studenti.ItemsSource = dataTable.DefaultView;

Similar se procedează pentru fiecare tab (Catalog, Profesori, etc.). Diferă doar tabela din care se preiau informațiile și dataGrid-ul unde sunt afișate.

**3.3.3 Procedura AUD (Add, Update, Delete)**

Aceasta funcție este construita pentru fiecare entitate de lucru, și reglementează procedura de adăugare, modificare și ștergere de informații din baza de date. Funcția accepta 3 parametri de tip *int*  numiți pOperatie:

* + 0 pentru a adăuga informații în baza de date **(Add)**
  + 1 pentru a modifica o informație din baza de date **(Update)**
  + 2 pentru a șterge informații din baza de date **(Delete)**

Funcția este apelata cu parametrul 0 când este apăsat butonul din secțiunea 4 a interfeței, cu parametrul 1 la apăsarea butonului , și cu parametrul 2 la apăsarea butonului .

Aceasta funcție include cele doua proceduri descrise anterior, pentru conectarea la baza de date și de actualizare a tabelului de vizualizare de date.

Exemplu pentru entitatea Grupe:

private void AUDGrupe(int pOperatie)

{

SqlConnection conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

String msg = "";

String sql = "";

switch (pOperatie)

{

case 0:

sql = "INSERT INTO ListaGrupe (NumeGrupa, IdSpecializare) " + "VALUES('" + txtGruNumeGrupa.Text + "', '" + idSpecializareAleasa + "')";

msg = "Datele despre grupa au fost adaugate cu suuces.";

break;

case 1:

sql = "UPDATE ListaGrupe SET NumeGrupa='" + txtGruNumeGrupa.Text + "', IdSpecializare='" +

idSpecializareAleasa + "' WHERE IdGrupa=" + txtGruIdGrupa.Text;

msg = "Datele despre grupa au fost actualizate.";

break;

case 2:

sql = "DELETE FROM ListaGrupe WHERE IdGrupa=" + txtGruIdGrupa.Text;

msg = "Datele despre grupa au fost sterse.";

break;

}

SqlCommand cmd = conn.CreateCommand();

cmd.CommandText = sql;

cmd.CommandType = CommandType.Text;

try

{

int n = cmd.ExecuteNonQuery();

if (n > 0)

{

MessageBox.Show(msg);

updateDataGrid\_Grupe();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

conn.Close();

resetGrupe();

creazaListaGrupe();

}

}

Descrierea procedurii:

* + Procedura începe prin a deschide o conexiune la baza de date.
  + Apoi se declara și instanțiază doua variabile de tip *string* msg și sql
  + Folosind o construcție de tip *switch* se vor testa cele 3 valori ale parametrului pOperatie
  + Pentru cazul 0 string-ul sql va prelua valoarea instrucțiunii de adăugare în tabela ListaGrupe a valorilor prezente în câmpurile din secțiunea 3 pentru a le stoca în NumeGrupa, IdSpecializare
  + String-ului msg i se atribuie valoarea "Datele despre grupa au fost adaugate cu suuces." Acest mesaj va fi afișat după adăugarea cu succes a datelor în baza de date.
  + Pentru cazul 1 string-ul sql va prelua valoarea instrucțiunii de modificare a numelui grupei și a specializării atașate înregistrării căreia ii corespunde id-ul afișat în câmpul txtGruIdGrupa
  + String-ul msg va afișa în acest caz un mesaj de actualizare cu succes a datelor
  + Pentru cazul 2 va avea loc o operație de ștergere din baza de date a înregistrării cu id-ul afișat în câmpul txtGruIdGrupa, după ce a fost selectata anterior o înregistrare din tabelul de afișare.
  + In acest caz string-ul msg va prelua o valoare ce va notifica utilizatorul despre ștergerea datelor din baza de date
  + După alegerea cazului din construcția *switch* se declara și instanțiază o comanda cmd de tip *SqlCommand*, care va folosi string-ul sql pentru a trimite instrucțiunile alese, către baza de date
  + Cu ajutorul unei construcții de tip *try/catch* vom incerca sa executam comanda cmd
    - In *try* declaram o variabila n de tip *int* care se instanțiază cu valoarea rândurilor modificate în baza de date de comanda cmd
    - Folosind un *if* verificam daca aceasta valoare este mai mare de 0, adică daca s-au efectuat modificări în baza de date
    - In caz afirmativ, se afișează mesajul msg și se actualizează tabelul de afișare folosind funcția updateDataGrid\_Grupe()
    - In caz negativ, nu se întâmpla nimic
    - In *catch* afișam mesajul de eroare ex.message în cazul în care operația asupra bazei de data a eșuat
    - Indiferent de rezultatele din *try* și *catch*, în *finnaly* închidem conexiunea la baza de date

**3.3.4 Procedura reset()**

 Aceasta procedura golește câmpurile completate în secțiunea 3 a interfeței.

Ea este apelata la apăsarea butonului , după orice modificare făcută în baza de date prin intermediul procedurii AUD descrise anterior și atunci când este ales un tab pentru afișare.

Exemplu pentru entitatea Materii:

private void resetMaterii()

{

txtMatNumeMaterie.Text = "";

txtMatIdMaterie.Text = "";

cbxMatSpecializare.Text = "";

btnAddMaterie.IsEnabled = true;

btnUpdateMaterie.IsEnabled = false;

btnDeleteMaterie.IsEnabled = false;

}

Descrierea procedurii:

* + Se atribuie pe rând un string gol ca valoare pentru câmpurile specifice tab-ului **Materii:**
    - Câmpul de nume al materiei
    - Câmpul de id al materiei
    - Câmpul de specializare asociata materiei
  + Apoi este activat butonul
  + Si sunt dezactivate butoanele și
  + Acești ultimi pași sunt necesari deoarece operațiile de adăugare sau ștergere sunt permise doar daca este selectată o înregistrare din tabelul de afișare a datelor, caz în care butonul de adăugare este dezactivat (nu se poate adăuga o înregistrare deja existenta în baza de date).

**3.3.5 Procedura Export()**

Aceasta procedura permite exportarea informatiilor afișate în secțiunea 1 a interfeței, salvând datele într-un fișier Excel, la o locație aleasă de utilizator.

Procedura accepta ca parametru de intrare un obiect de tip *DataGrid*, ca sursa a datelor ce vor fi exportate.

Ea este apelata la apăsarea butonului , aflat în fiecare tab, iar în funcție de tab-ul în care ne aflam, se trimite funcției, obiectul de tip *DataGrid* al tab-ului respectiv ca parametru (de ex. Export(dgr\_profesori);)

Funcția Export:

private void Export (DataGrid dgr)

{

Microsoft.Office.Interop.Excel.Application excelApplication = new Microsoft.Office.Interop.Excel.Application();

Microsoft.Office.Interop.Excel.Workbook excelWorkBook = excelApplication.Workbooks.Add(Type.Missing);

Microsoft.Office.Interop.Excel.Worksheet excelWorkSheet= null;

try

{

excelWorkSheet = excelWorkBook.ActiveSheet;

int cellRowIndex = 1;

int cellColumnIndex = 1;

DataTable dtExport = new DataTable();

dtExport = ((DataView)dgr.ItemsSource).ToTable();

for (int i = 0; i <= dtExport.Rows.Count; i++)

{

for (int j = 0; j < dtExport.Columns.Count; j++)

{

if (cellRowIndex == 1)

{

excelWorkSheet.Cells[cellRowIndex, cellColumnIndex] = dtExport.Columns[j].ColumnName;

}

else

{

excelWorkSheet.Cells[cellRowIndex, cellColumnIndex] = dtExport.Rows[i-1][j].ToString();

}

cellColumnIndex++;

}

cellColumnIndex = 1;

cellRowIndex++;

}

System.Windows.Forms.SaveFileDialog saveDialog = new System.Windows.Forms.SaveFileDialog();

saveDialog.Filter = "Excel files (\*.xlsx)|\*.xlsx|All files (\*.\*)|\*.\*";

saveDialog.FilterIndex = 1;

if (saveDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

excelWorkBook.SaveAs(saveDialog.FileName);

MessageBox.Show("Export Successful");

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

excelApplication.Quit();

excelWorkBook = null;

excelApplication = null;

}

}

Descrierea procedurii:

* Pentru a putea lucra cu fișiere Excel se declară și instanțiază 3 variabile:
  + excelApplication de tip *Excel.Applciation*
  + excelWorkBook de tip *Excel.Workbook*
  + excelWorkSheet de tip *Excel.Worksheet*
* Deschidem un bloc *try* și declaram fila activa a fișierului Excel
* Declaram doua variabile de tip *int* cellRowIndex și cellColumnIndex, atribuindu-le valoarea 1. Ele ne vor ajuta sa parcurgem tabelul cu înregistrări din baza de date
* Declaram și instanțiem o variabila dtExport de tip *DataTable*
* Apoi ii atribuim valorile din parametrul de intrare dgr
* Deschidem o bucla de tip *for* care parcurge toate liniile și coloanele din dtExport și atribuie valorile găsite, celulelor corespondente din fișierul Excel
* Declaram și instanțiem o variabilă saveDialog de tip *Windows.Forms.SaveFileDialog*

care va crea fereastra de salvare a fișierului de export

* In aceasta fereastra stabilim tipul de fișiere de export ca fiind „Excel files”
* Deschidem un bloc condițional *if* și daca se apasă butonul Ok fișierul de export va fi salvat cu numele dat și mesajul "Export realizat cu succes!" va fi afișat
* Închidem blocul *try*
* Deschidem un bloc *catch* care va afișat mesajul erorii intampinate, daca este cazul
* Închidem blocul *catch*
* Deschidem blocul *finally*
* Închidem aplicația Excel și atribuim valoarea nulă variabilelor excelWorkBook și excelWorkSheet

**3.3.6 Procedurile de creare a listelor de tip drop-down**

Aceasta procedura este folosită pentru a popula cu informații din baza de date, listele de tip drop-down cu entități:

* In tab-ul **Catalog**:
  + Lista de profesori (in aceasta lista se grupează numele și prenumele pentru a fi afișate împreuna)
  + Lista de materii
  + Lista de studenți (in aceasta lista se grupează numele și prenumele pentru a fi afișate împreuna)
  + Lista de note folosește o procedura diferita dar simplificata (un loop *for* ce parcurge un sir de 10 numere și le adaugă pe rând în lista)
* In tab-ul **Profesori**:
  + Lista de materii
* In tab-ul **Studenți**:
  + Lista de grupe
* In tab-ul **Grupe**:
  + Lista de specializări
* In tab-ul **Materii**:
  + Lista de specializări

Exemplu pentru crearea listei de specializări existente în baza de date:

List<int> valoriIdSpecializari = new List<int>();

private void creazaListaSpecializari()

{

SqlConnection conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

string query = "SELECT \* FROM ListaSpecializari ORDER BY NumeSpecializare";

SqlCommand cmd = new SqlCommand(query, conn);

SqlDataReader dataReader = cmd.ExecuteReader();

string s;

cbxMatSpecializare.Items.Clear();

cbxGruSpecializare.Items.Clear();

while (dataReader.Read())

{

s = dataReader["NumeSpecializare"].ToString();

cbxMatSpecializare.Items.Add(s);

cbxGruSpecializare.Items.Add(s);

valoriIdSpecializari.Add(Convert.ToInt32(dataReader["IdSpecializare"].ToString()));

}

dataReader.Close();

conn.Close();

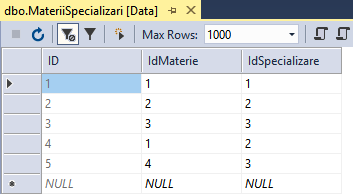
}

Descrierea procedurii:

* Declaram și instanțiem o listă de valori List<int> valoriIdSpecializari de tip *int* care va stoca id-urile specializărilor adăugate în lista drop-down
* Folosind procedura 3.3.1 extragem toate informațiile din tabela ListaSpecializari (numele specializarilor și id-ul lor). Aceste informații se vor găsi în *dataReader*
* Declaram o variabila s de tip *string* ce va stoca numele specializării adăugate în lista drop-down
* Înainte de a adăuga valori în listele drop down din tab-urile **Materii** și **Specializări**,golim mai întâi conținutul lor pentru a ne asigura ca nu adăugăm date duplicate, de fiecare când apelam aceasta funcție
* Folosind o bucla *while*, parcurgem fiecare linie din dataReader
* Informatiile din coloana NumeSpecializare sunt salvate în string-ul s, apoi adăugate în listele drop-down din tab-urile **Materii** și **Specializări**
* Informatia de pe coloanaIdSpecializare este adăugată în lista List<int> valoriIdSpecializari. Aceasta lista ava fi folosita ulterior și pentru procedura 3.2.7

**3.3.7 Procedurile de adăugare și ștergere a asocierilor dintre entități**

Aceste doua proceduri sunt printre cele mai complexe proceduri folosite în aceasta aplicație. Prin intermediul lor se realizează sau se șterge asocierea dintre entitățile materii-profesori și materii-specializări. Aceste asocieri se regăsesc în tabelele **MateriiProfesori** și **MateriiSpecializari** sub forma de perechi de chei externe (exemplificat în figura nr.3).



**Figura nr.3 Tabela MateriiSpecializari**

Adăugarea și ștergerea asocierilor se realizează folosind butoanele din dreptul câmpurilor:

* Materie în tab-ul **Profesori**
* Specializare în tab-ul **Materii**

Acesta funcționalitate se regăsește doar în tab-urile menționate mai sus.

Exemplu pentru adăugarea unde asocieri intre o materie și o specializare:

int idSpecializareAleasa = 0;

int idMaterieAleasa = 0;

List<int> valoriIdSpecializari = new List<int>();

private void btnAdaugaSpecializare\_Click(object sender, RoutedEventArgs e)

{

if (cbxMatSpecializare.Text == "")

{

MessageBox.Show("Nu ati ales nici o specializare!");

return;

}

else

{

int index = 0;

index = cbxMatSpecializare.SelectedIndex;

idSpecializareAleasa = valoriIdSpecializari[index];

}

DataRowView dataRowView = dgr\_materii.SelectedItem as DataRowView;

if (dataRowView == null)

{

MessageBox.Show("Nu ati ales o materie!");

return;

}

else

{

idMaterieAleasa = Convert.ToInt32(txtMatIdMaterie.Text);

}

SqlConnection conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

string query = "SELECT \* FROM MateriiSpecializari WHERE IdMaterie=" + idMaterieAleasa.ToString() + " AND IdSpecializare=" + idSpecializareAleasa.ToString();

SqlCommand cmd = new SqlCommand(query, conn);

SqlDataReader dataReader = cmd.ExecuteReader();

if (dataReader.HasRows)

{

MessageBox.Show("Aceasta specializare este deja adaugata!");

dataReader.Close();

conn.Close();

return;

}

else

{

dataReader.Close();

conn.Close();

query = "INSERT INTO MateriiSpecializari (IdMaterie, IdSpecializare) VALUES (" + idMaterieAleasa.ToString() + ", "

+ idSpecializareAleasa.ToString() + ")";

}

conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

cmd = new SqlCommand(query, conn);

int rezultat = cmd.ExecuteNonQuery();

if (rezultat > 0)

{

MessageBox.Show("Datele despre specializare au fost adaugate cu succes.");

}

else

{

MessageBox.Show("Datele nu au fost inserate!");

}

dataReader.Close();

conn.Close();

updateDataGrid\_Materii();

resetMaterii();

}

Descrierea procedurii:

* Funcția este apelată doar la apăsarea butonului din tab-ul **Materii**. Similar funcționează și funcția de adaugare din tab-ul **Profesori**.
* Declaram o variabila idSpecializareAleasa de tip *int* și o instanțiem cu valoarea 0; Ea ne va ajuta sa stocam id-ul specializării ce va trebui asociata
* Declaram o variabila idMaterieAleasa de tip *int* și o instanțiem cu valoarea 0; Ea ne va ajuta sa stocam id-ul materiei ce va trebui asociata
* Declaram și instanțiem o lista de valori List<int> valoriIdSpecializari de tip *int* care va stoca id-urile tuturor specializărilor existente în baza de date
* Folosim un block if – else pentru a verifica mai întâi daca a fost sau nu selectata o intrare din lista de specializări
* Daca nu a fost aleasa nici o intrare, se va afișa mesajul „Nu ați ales nici o specializare!”
* Daca a fost aleasa o specializare, declaram o variabila index de tip *int* și o instanțiem cu valoarea poziției specializării alese din lista drop-down
* Id-ul specializării alese din listă, se obține din lista List<int> valoriIdSpecializari și se atribuie variabilei idSpecializareAleasa cu valoarea stocata pe poziția valoriIdSpecializari[index]
* Id-ul specializării este necesar pentru a stabili asocierea cu un id de materie
* Id-ul materiei pentru asociere este preluat din câmpul Id Materie
* Daca id-ul nu este prezent în acest câmp înseamnă ca nu a fost aleasă o linie din tabelul de vizualizare și mesajul ”Nu ați ales o materie” va fi afișat.
* Folosim procedura 3.3.1 sa interogam baza de date pentru a afla daca exista deja o înregistrare în tabela **MateriiSpecializari** pentru o asociere dintre cele doua id-uri obținute
* Se deschide apoi un bloc if – else
* Daca exista deja asocierea, mesajul „Aceasta specializare este deja adăugată” va fi afișat
* Daca asocierea nu exista, atribuim o noua valoare string-ului querry și folosind procedura 3.3.3 adăugam în baza de date o noua înregistrare în tabela **MateriiSpecializari** cu asocierea dintre id-urile respective
* Se declara și instanțiază o variabilă rezultat de tip *int* cu valoarea numărului de rânduri schimbate de procedura 3.3.3
* Daca aceasta valoare este mai mare ca 0, adică s-a modificat cel puțin un rând din baza de date, mesajul „Datele despre specializare au fost adăugate cu succes” va fi afișat.
* Daca valoarea este 0, înseamnă ca nu a fost produsa nici o modificare în baza date și mesajul „Datele nu au fost inserate!” va fi afișat
* Se închide dataReader-ul
* Închidem conexiunea
* Apelam procedura 3.3.2 updateDataGrid\_Materii(); pentru a actualiza datele afișate în tabelul de vizualizare
* Apelam procedura 3.3.4 resetMaterii(); pentru a goli câmpurile populate în secțiunea 3 și a pregăti interfața pentru o operație noua

Exemplu pentru ștergerea unde asocieri intre o materie și un profesor:

int idProfesorAles = 0;

int idMaterieAleasa = 0;

List<int> valoriIdMaterii = new List<int>();

private void btnStergeMaterie\_Click(object sender, RoutedEventArgs e)

{

if (cbxProfesorMaterii.Text == "")

{

MessageBox.Show("Nu ati ales nici un profesor!");

return;

}

else

{

int index = 0;

index = cbxProfesorMaterii.SelectedIndex;

idMaterieAleasa = valoriIdMaterii[index];

}

DataRowView dataRowView = dgr\_profesori.SelectedItem as DataRowView;

if (dataRowView == null)

{

MessageBox.Show("Nu ati ales un profesor!");

return;

}

else

{

idProfesorAles = Convert.ToInt32(txtId.Text);

}

SqlConnection conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

string query = "SELECT \* FROM MateriiProfesori WHERE IdProfesor=" + idProfesorAles.ToString() + " AND IdMaterie=" + idMaterieAleasa.ToString();

SqlCommand cmd = new SqlCommand(query, conn);

SqlDataReader dataReader = cmd.ExecuteReader();

if (dataReader.HasRows)

{

query = "DELETE FROM MateriiProfesori WHERE IdProfesor = " + idProfesorAles.ToString() + " AND " + "IdMaterie= " + dMaterieAleasa.ToString();

dataReader.Close();

conn.Close();

}

else

{

MessageBox.Show("Asociere inexistenta!");

dataReader.Close();

conn.Close();

return;

}

conn = new SqlConnection();

conn.ConnectionString = Properties.Settings.Default.ConnString;

conn.Open();

cmd = new SqlCommand(query, conn);

int rezultat = cmd.ExecuteNonQuery();

if (rezultat > 0)

{

MessageBox.Show("Asocierea dintre profesor și materie a fost stearsa cu succes.");

}

else

{

MessageBox.Show("Datele nu au fost sterse!");

}

dataReader.Close();

conn.Close();

updateDataGrid\_Profesori();

resetProfesori();

}

Descrierea procedurii:

* Funcția este apelata doar la apăsarea butonului din tab-ul **Profesori**. Similar funcționează și funcția de ștergere din tab-ul **Materii**.
* Declaram o variabila idProfesorAleas de tip *int* și o instanțiem cu valoarea 0; Ea ne va ajuta sa stocam id-ul profesorului ce va trebui asociat
* Declaram o variabila idMaterieAleasa de tip *int* și o instanțiem cu valoarea 0; Ea ne va ajuta sa stocam id-ul materiei ce va trebui asociata
* Declaram și instanțiem o lista de valori List<int> valoriIdMaterii de tip *int* care va stoca id-urile tuturor materiilor existente în baza de date
* Folosim un block if – else pentru a verifica mai întâi daca a fost sau nu selectata o intrare din lista de materii
* Daca nu a fost aleasa nici o intrare, se va afișa mesajul „Nu ați ales nici o materie!”
* Daca a fost aleasa o materie, declaram o variabila index de tip *int* și o instanțiem cu valoarea poziției materiei alese din lista drop-down
* Id-ul materiei alese din lista, se obține din lista List<int> valoriIdMaterii și se atribuie variabilei idMaterieAleasa cu valoarea stocata pe poziția valoriIdMaterii[index]
* Id-ul materiei este necesar pentru a stabili asocierea cu un id de profesor
* Id-ul profesorului pentru asociere este preluat din câmpul Id profesor
* Daca id-ul nu este prezent în acest câmp înseamnă ca nu a fost aleasa o linie din tabelul de vizualizare și mesajul ”Nu ați ales un profesor” va fi afișat.
* Folosim procedura 3.3.1 sa interogam baza de date pentru a afla daca exista deja o înregistrare în tabela **MateriiProfesori** pentru o asociere dintre cele doua id-uri obținute
* Se deschide apoi un bloc if – else
* Daca asocierea exista, atribuim o noua valoare string-ului querry și folosind procedura 3.3.3 ștergem din baza de date înregistrarea din tabela **MateriiProfesori** cu asocierea dintre id-urile respective
* Daca nu exista asocierea, mesajul „Asociere inexistenta!” va fi afișat
* Se declara și instanțiază o variabila rezultat de tip *int* cu valoarea numărului de rânduri schimbate de procedura 3.3.3
* Daca aceasta valoare este mai mare ca 0, adică s-a modificat cel puțin un rând din baza de date, mesajul „Asocierea dintre profesor și materie a fost ștearsă cu succes.” va fi afișat.
* Daca valoarea este 0, înseamnă ca nu a fost produsa nici o modificare în baza date și mesajul „Datele nu au fost șterse!” va fi afișat
* Se închide dataReader-ul
* Închidem conexiunea
* Apelam procedura 3.3.2 updateDataGrid\_Profesori(); pentru a actualiza datele afișate în tabelul de vizualizare
* Apelam procedura 3.3.4 resetProfesori(); pentru a goli câmpurile populate în secțiunea 3 și a pregăti interfața pentru o operație noua

Capitolul 4

MANUAL DE UTILIZARE

Pentru a putea folosi aplicația în mod optim este necesara o populare inițială a bazei de date cu informații pentru fiecare entitate de lucru, folosind următorii pași:

1. Adăugarea profesorilor în tabela ListaProfesori folosind tab-ul **Profesori**
2. Adăugarea studenților în tabela ListaStudenti folosind tab-ul **Studenți**
3. Adăugarea specializărilor în tabela ListaSpecializari folosind tab-ul **Specializări**
4. Adăugarea grupelor în tabela ListaGrupe folosind tab-ul **Grupe**
5. Actualizarea tabelei ListaStudenti cu grupa fiecărui student folosind tab-ul **Studenți**
6. Adăugarea materiilor în tabela ListaMaterii folosind tab-ul **Materii**
7. Adăugarea legăturilor dintre materii și profesorii ce le predau, în tabela de legătură **MateriiProfesori**  folosind tab-ul **Profesori**  și butoanele specifice
8. Adăugarea legăturilor dintre materii și specializările unde se predau, în tabela de legătură **MateriiSpecializari** folosindtab-ul **Materii și** butoanele
9. Adăugarea de înregistrări în tabela **Catalog** folosind tab-ul cu același nume

**4.1 Instrucțiuni pentru adaugarea de inregistrari noi**

Acest lucru se poate realiza folosind câmpurile din secțiunea 3. Completarea tuturor câmpurilor este obligatorie, excepție făcând următoarele tab-uri:

* **Profesori**: este permisa adăugarea unui profesor nou fără a fi selectata o materie
* **Studenți**: este permisa adăugarea unui student nou fără a fi selectata o grupa
* **Materii**: este permisa adăugarea unei materii noi fără a fi selectata o specializare

După completare, informațiile pot fi adăugate în baza de date cu butonul

**4.2 Modificarea unei înregistrări**

Mai întâi trebuie selectata o linie din tabelul de vizualizare (1). Datele înregistrării respective vor fi preluate de câmpurile din secțiunea 3. Aceste câmpuri pot fi modificate, apoi noile informații pot fi actualizate în baza de date folosind butonul . Toate câmpurile obligatorii trebuie sa conțină informații pentru a fi permisa actualizarea.

**4.3 Ștergerea unei înregistrări**

Pentru a șterge o înregistrare existenta, ea trebuie aleasa din tabelul de vizualizare, apoi apăsat butonul .

**Notă:** Cu toate ca funcționalitatea de a șterge o înregistrare din baza de date este implementată, din cauza constrângerilor de integritate ce leagă diferite tabele prin intermediul cheilor străine, pot apărea complicații atunci se dorește ștergerea unei înregistrări din baza de date. Pentru a fi permisă, dacă atributele înregistrării alese pentru ștergere sunt chei primare, trebuie șterse și înregistrările din tabelele în care aceste atribute sunt chei străine. Cu alte cuvinte daca dorim sa ștergem un profesor din tabela Profesori, unde Id profesor este cheie primara, va trebui sa ștergem mai întâi toate înregistrările din tabela MateriiProfesori, unde Id profesor este cheie străină și face legătura intre o materie și profesorul care o predă, pe baza cheilor entităților Materie și Profesor. în concluzie, argumentez împotriva ștergerii înregistrărilor de acest tip, singura excepție fiind tab-ul **Catalog**, unde cheia primara – Id – nu este cheie străină în nici o altă tabelă.

1. 

**4.4 Butonul**

Folosește la golirea câmpurilor din secțiunea 3, după ce acestea au fost populate cu informații, fie prin selectarea unei linii din tabelul de vizualizare, sau prin alegerea unei intrări din lista prezenta sau completarea directa a datelor.



**4.5 Butonul**

Permite exportarea datelor afișate în tabelul de vizualizare intre-un fișier Excel. După apăsare, este deschisa o fereastra de dialog unde utilizatorul poate alege locatia și numele fișierului de export. Daca exportul este realizat cu succes un mesaj va fi afișat pe ecran.

**Capitolul 5**

**CONCLUZII și DEZVOLTĂRI ULTERIOARE**

Aplicația de fata reușește sa îndeplinească cerințele propuse în capitolul introductiv:

* Permite administrarea cu ușurință a entităților specifice unui catalog electronic, întreaga interacțiune realizând-se într-o singura fereastra
* Modificările specifice pot fi realizare prin intermediul interfeței intuitive
* Date stocate în baza de date pot fi exportate într-un format ușor de folosit pentru imprimare sau prelucrări ulterioare

Cu ajutorul unor dezvoltări suplimentare, aceasta aplicație ar putea fi folosita într-un cadru real într-un mod foarte eficient.

Îmbunătățiri propuse:

* Implementarea unei interfețe web
* Modul de importare a datelor inițiale
* Interfața de autentificare pentru diferite tipuri de utilizatori (administrator, student, profesor)
* Jurnal de modificări ale bazei de date
* Modul de realizare a unei copii de rezerva a bazei de date
* Aplicație pentru telefonul mobil

Așa cum este exemplificat în capitolul 2, proiectând într-o maniera structurată baza de date, se pot realiza nenumărate aplicații similare, care sa permită administrarea cu ușurință a oricărui tip de informație. Combinând acest lucru cu o interfață ușor de folosit, utilitatea unei astfel de aplicații este mai mult decât evidenta.

**Bibliografie:**

**M. Velicanu, I. Lungu, C. Bodea, C. Ioniță, G. Bădescu (2000). *Sisteme de gestiune a bazelor de date.* București: Editura Petrion**

**I. Lungu, C. Bodea, G. Bădescu, C. Ioniță (1995). *Baze de date – Organizare, proiectare și implementare.* București: Editura ALL**

**L. Maciaszek (2005). *Requirements analysis and system design - developing Information systems with UML* (Ediția a 2-a). Essex, UK: Editura Addison Wesley**

**V. Cotelea (2003). *Baze de date relaționale.* Chișinău: 2003 – accesat pe** [**http://www.math.md/studlib/informatica/bd\_relationale.html**](http://www.math.md/studlib/informatica/bd_relationale.html) **accesat la data de 13.02.2018**

**C# programming guide – accesat pe** [**https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/**](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/) **accesat la data de 20.02.2018**