**UNIVERSITATEA „ARTIFEX” DIN BUCUREŞTI**

**FACULTATEA DE FINANȚE ŞI 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 ŞI 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
2. Proiectarea aplicației 5
   1. Prezentarea tehnologiilor folosite 5
   2. Proiectarea bazei de date 5
   3. Prezentarea bazei de date 8
   4. Diagrama UML 14
3. Realizarea aplicației 15
   1. Conectarea la baza de date 15
   2. Realizarea interfeței grafice 16
   3. Proceduri folosite 12
4. Manual de utilizare 13
5. Concluzii și dezvoltări ulterioare 14
6. Bibliografie 15

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 si 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 si încheind cu diagrama UML.

Capitolul 3 va descrie în detaliu realizarea efectiva a aplicației, urmărind conectarea la baza de date, algoritmii folosiți de-a lungul aplicației si realizarea interfeței grafice.

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

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

de învățământ și prezinta 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 legaturile 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 in 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 si se refera la:

1. **Atribute si domenii**. Plecând de la unul din avantajele fundamentale ale modelului relațional, omogenitatea, care presupune așezarea datelor în tabele cu linii si coloane care au același format, **atributul** reprezintă o caracteristica a unei entități din baza de date si este reprezentat de valoarea ce se regăsește la intersecția dintre o linie si 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 si de regula ea reprezintă numele tabelei din baza de date. **Schema** relației reprezintă suma proprietăților comune ale tuplurilor conținute in relație.

2. **Restricțiile de integritate** reprezintă un set de cerințe ce trebuie îndeplinite pentru a asigura corectitudinea si coerența datelor din baza de date. **Cheia** unui tuplu reprezintă un identificator unic, reprezentat de valoarea unui atribut. In acest caz ea se mai numește si **cheie simplă**. In cazul in 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 si 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 si posibilitatea de a fi asociat unei grupe de studenți. Pot fi adăugați studenți in aceasta tabelă, fără a fi asociați cu o grupă, presupunând ca sunt înscriși de curând si 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 si 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 si 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 si in **lista de specializări**:nume specializare si identificator unic.

**Lista de grupe** trebuie sa conțină numele fiecărei grupe, id-ul grupei (care va fi si identificator unic) si 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 si profesori, adică ce materii predă fiecare profesor, vom crea tabela de legătură **Materii-Profesori**, ce va conține pe lângă un id unic de identificare si 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 si 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 in tabela proprie, dar vor fi si 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 in 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 si o materie
  + Unui profesor i se pot asocia mai multe materii

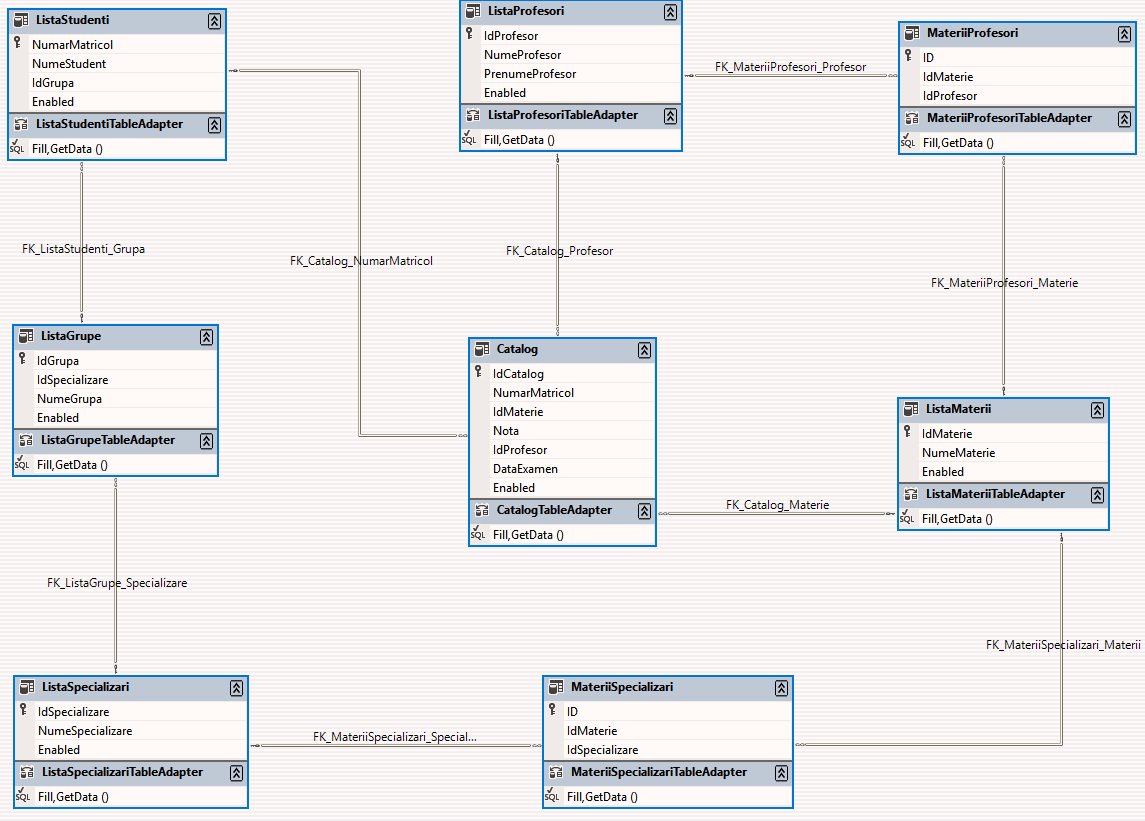
Tabela **MateriiSpecializari** cu atributele:

* **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 si 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 in catalog
* **DataExamen**
  + Tip de data *date*
  + Nu accepta valori nule
  + Definește data examenului înregistrat in catalog

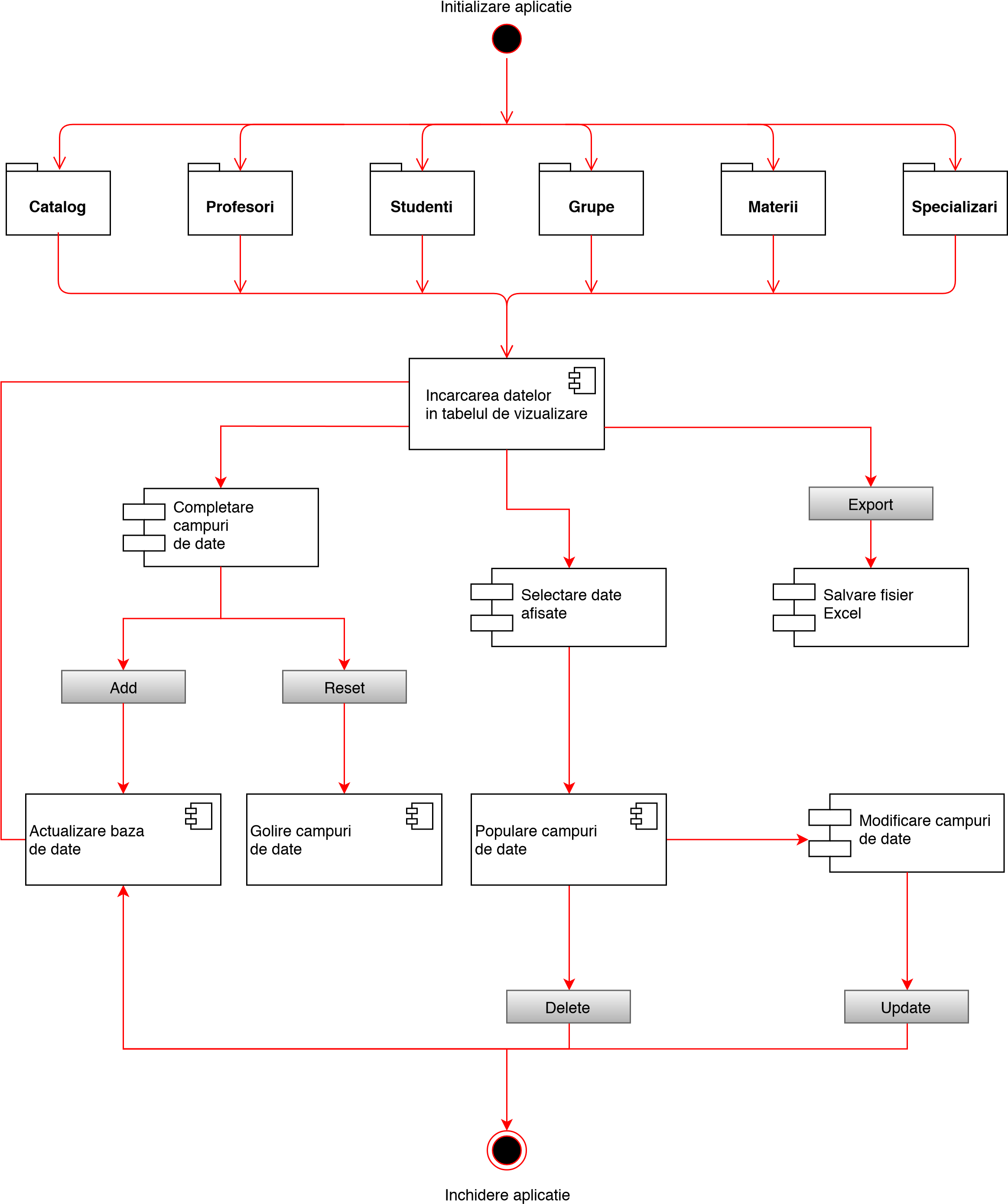
Legăturile dintre tabele se pot observa cu ușurință si in figura nr.1:

****

**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 in cadrul acestei aplicații si modul in 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 in 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 si afișate in secțiunea 1.

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

După încărcarea datelor in 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 in elementul 1.
  + După selectare, câmpurile din secțiunea 3 sunt populate automat cu datele înregistrării selectate
  + Aceste date pot fi modificate si 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 si 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 in 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# si primul pas a fost definirea modalității de conectare la baza de date creată in capitolul anterior. In 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 si ConnectTimeout este durata pentru care se va relua conectarea la baza de date in 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 si instanția o noua variabila conn de tipul SqlConnection si i se va atribui proprietății conn.ConnectionString valoarea stocată in ConnString. Baza de date se va deschide folosind funcția conn.Open(), iar după efectuarea operațiilor se va închide cu conn.Close().

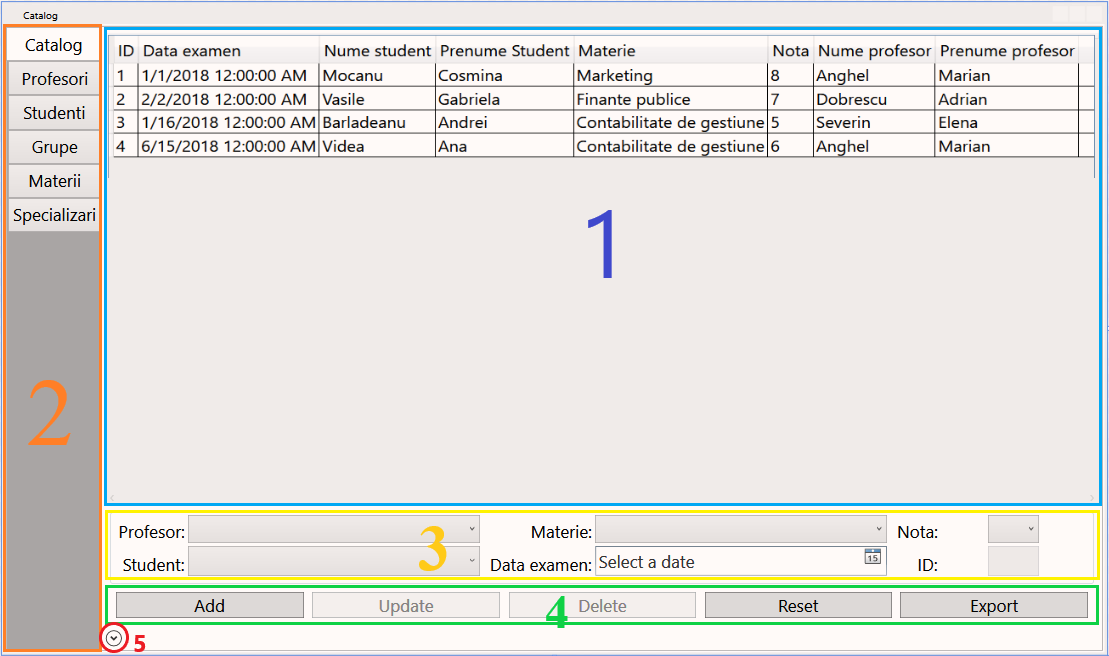
După fiecare deschidere a conexiunii către baza de date este foarte importantă si închiderea acesteia pentru a preveni accesul neautorizat la datele existente.

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

* Adăugarea unei informații noi in 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. In funcție de alegerea făcută, se vor afișa informațiile specifice entității in tabelul (1) si in 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 si exportul informațiilor. Aceste butoane sunt identice pentru fiecare tab.
5. Panourile (3) si (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*, in 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 in 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* in partea de jos unde sunt aranjate toate butoanele si listele din secțiunile (3) si (4) conform figurii 2. Folosind butonul (5) aceste liste si 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 si coloane.

**3.3 Proceduri folosite**

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

Principala procedura folosita in întreaga aplicație este cea de conectare la baza de date, interogare si obținerea unor informații. Acest lucru se realizează prin intermediul unei variabile de tip *SqlCommand* căruia i se trimit doi parametri de tipul *string* si *SqlConnection.* Parametrul de tip *string* conține instrucțiunile in 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 in variabila de tip *SqlDataReader*.

Aceasta procedura este folosită in următoarele cazuri:

1. Crearea listelor de entități necesare popularii câmpurilor de tip drop-down din tab-urile:
   * Catalog
     + Lista de profesori cuprinsa in cbxCatalogProfesor.Items
     + Lista de studenți cuprinsa in cbxCatalogStudent.Items
     + Lista de materii cbxCatalogMaterie.Items
   * Profesori
     + Lista de materii ce pot fi asociate cu profesorii, cuprinsa in cbxProfesorMaterii.Items
   * Studenți
     + Lista de grupe ce pot fi asociate cu studenții, cuprinsa in cbxStuGrupa.Items
   * Grupe
     + Lista de specializări de pot fi asociate cu fiecare grupa, cuprinsa in cbxGruSpecializare.Items
   * Materii
     + Lista de specializări ce pot fi asociate cu fiecare materie, cuprinsa in 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 in 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 si instanțierea unei variabile conn de tip *SqlConnection*
  + Se atribuie valoarea stocata in Properties.Settings.Default.ConnString proprietății conn.ConnectionString
  + Deschiderea conexiunii către baza de date apelând conn.Open()
  + Declararea unei variabile query de tip *string* si instanțierea ei cu valoarea "SELECT \* FROM ListaProfesori ORDER BY NumeProfesor, PrenumeProfesor " care in limbaj SQL interoghează tabela ListaProfesori din baza de date si extrage toate informațiile (Id, Nume, Prenume) despre profesorii existenți, ordonându-i mai întâi după NumeProfesor apoi după PrenumeProfesor
  + Declararea si instantierea unei variabile cmd de tip *SqlCommand* care primește parametrii query si conn, definiți mai sus. Aceasta comanda folosește datele definite de parametrul conn ca sa se conecteze la baza de date si datele din parametrul query ca sa extragă informații
  + Declararea unei variabile dataReader de tip *SqlDataReader* in care vom stoca informațiile obținute din baza de date, prin trimiterea comenzii cmd
  + Declararea unei variabile s de tip *string* in care vom stoca valorile ce vor fi adăugate in 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 si in variabila s se atribuie valorile NumeProfesor si PrenumeProfesor, concatenate
  + In lista cbxCatalogProfesor.Items se adaugă valoarea lui s
  + Id-ul profesorului adăugat in lista de mai sus se adaugă într-o lista diferita (valoriIdProfesori). Aceasta lista va ajuta mai târziu in 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 si lista unde vor fi stocate aceste informații.

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

Aceasta funcție updateDataGrid() este apelata in 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 si la folosirea lor:

* + Pentru a interoga baza de date se va construi *string*-ul querry care va extrage numărul matricol, numele si prenumele studenților din tabela ListaStudenti, precum si grupa din tabela ListaGrupe, folosind asocierea făcută pe baza cheii de legătura IdGrupa.
  + Pentru a folosi informațiile preluate de dataReader, se va declara si 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 si 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 si dataGrid-ul unde sunt afișate.

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

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

* + 0 pentru a adăuga informații in 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 , si cu parametrul 2 la apăsarea butonului .

Aceasta funcție include cele doua proceduri descrise anterior, pentru conectarea la baza de date si 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 si instanțiază doua variabile de tip *string* msg si 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 in tabela ListaGrupe a valorilor prezente in câmpurile din secțiunea 3 pentru a le stoca in 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 in baza de date.
  + Pentru cazul 1 string-ul sql va prelua valoarea instrucțiunii de modificare a numelui grupei si a specializării atașate înregistrării căreia ii corespunde id-ul afișat in câmpul txtGruIdGrupa
  + String-ul msg va afișa in 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 in 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 si 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 in baza de date de comanda cmd
    - Folosind un *if* verificam daca aceasta valoare este mai mare de 0, adică daca s-au efectuat modificări in baza de date
    - In caz afirmativ, se afișează mesajul msg si se actualizează tabelul de afișare folosind funcția updateDataGrid\_Grupe()
    - In caz negativ, nu se întâmpla nimic
    - In *catch* afișam mesajul de eroare ex.message in cazul in care operația asupra bazei de data a eșuat
    - Indiferent de rezultatele din *try* si *catch*, in *finnaly* închidem conexiunea la baza de date

**3.2.4 Procedura reset()**

 Aceasta procedura golește câmpurile completate in secțiunea 3 a interfeței.

Ea este apelata la apăsarea butonului , după orice modificare făcută in baza de date prin intermediul procedurii AUD descrise anterior si 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 si
  + 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 in care butonul de adăugare este dezactivat (nu se poate adăuga o înregistrare deja existenta in baza de date).

**3.2.4 Procedura Export()**

Aceasta procedura permite exportarea informatiilor afișate in 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 in fiecare tab, iar in funcție de tab-ul in care ne aflam, se trimite funcției obiectul de tip *DataGrid* al tab-ului respectiv ca parametru (de ex. Export(dgr\_profesori);)

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 si instanțiază 3 variabile:
  + excelApplication de tip *Excel.Applciation*
  + excelWorkBook de tip *Excel.Workbook*
  + excelWorkSheet de tip *Excel.Worksheet*
* Deschidem un bloc *try* si declaram fila activa a fișierului Excel
* Declaram doua variabile de tip *int* cellRowIndex si cellColumnIndex, atribuindu-le valoarea 1. Ele ne vor ajuta sa parcurgem tabelul cu înregistrări din baza de date
* Declaram si 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 si coloanele din dtExport si atribuie valorile găsite, celulelor corespondente din fișierul Excel
* Declaram si 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* si daca se apasă butonul Ok fișierul de export va fi salvat cu numele dat si 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 si atribuim valoarea nula lui excelWorkBook si excelWorkSheet

Capitolul 4

MANUAL DE UTILIZARE

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

1. Adăugarea profesorilor in tabela ListaProfesori folosind tab-ul **Profesori**
2. Adăugarea studenților in tabela ListaStudenti folosind tab-ul **Studenți**
3. Adăugarea specializărilor in tabela ListaSpecializari folosind tab-ul **Specializări**
4. Adăugarea grupelor in tabela ListaGrupe folosind tab-ul **Grupe**
5. Actualizarea tabelei ListaStudenti cu grupa fiecărui student folosind tab-ul **Studenți**
6. Adăugarea materiilor in tabela ListaMaterii folosind tab-ul **Materii**
7. Adăugarea legăturilor dintre materii si profesorii ce le predau in tabela de legătură **MateriiProfesori**  folosind tab-ul **Profesori**  si butoanele specifice
8. Adăugarea legăturilor dintre materii si specializările unde se predau in tabela de legătură **MateriiSpecializari** folosindtab-ul **Materii** si butoanele
9. Adăugarea de înregistrări in 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 in 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 in 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 si înregistrările din tabelele in 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 si face legătura intre o materie si profesorul care o predă, pe baza cheilor entităților Materie si Profesor. In concluzie, argumentez împotriva ștergerii înregistrărilor de acest tip, singura excepție fiind tab-ul **Catalog**, unde cheia primara – Id – nu este cheie străină in 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.



**4.4 Butonul**

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

**Capitolul 5**

**CONCLUZII SI DEZVOLTĂRI ULTERIOARE**

Aplicatia de fata reuseste sa indeplineasca cerintele propuse in capitolul introductiv:

* Permite administrarea cu usurinta a entitatilor specifice unui catalog electronic, intreaga interactiune realizandu-se intr-o singura fereastra
* Modificarile specifice pot fi realizare prin intermediul interfetei intuitive
* Date stocate in baza de date pot fi exportate intr-un format usor de folosit pentru imprimare sau prelucrari ulterioare

Cu ajutorul unor dezvoltari suplimentare, aceasta aplicatie ar putea fi folosita intr-un cadru real intr-un mod foarte eficient.

Imbunatatiri propuse:

* Implementarea unei interfete web
* Modul de importare a datelor initiale
* Interfata de autentificare pentru diferite tipuri de utilizatori (administrator, student, profesor)
* Jurnal de modificari ale bazei de date
* Modul de realizare a unei copii de rezerva a bazei de date
* Aplicatie pentru telefonul mobil

Asa cum este exemplificat in capitolul 2, proiectand intr-o maniera structurata baza de date, se pot realiza nenumarate aplicatii similare, care sa permita administrarea cu usurinta a oricarui tip de informatie. Combinand acest lucru cu o interfata usor de folosit, utilitatea unei astfel de aplicatii este mai mult decat evidenta.

**Bibliografie:**