MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT

disciplina

Introducere in Baze de Date

Sistem de gestiune a unei platforme de studiu

An academic: 2022-2023

Grupa: 30221

PROIECT DE SEMESTRU
Studenți: *Buda Andreea și Filip Iarina*
Catedra de calculatoare
Disciplina: Introducere în Baze de Date
Coordonator: Cosmina Ivan
Data: 18.01.2023

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Cuprins

I. INTRODUCERE

1. FORMULAREA PROBLEMEI

Se doreste implementarea unui sistem informatic destinat gestiunii unei platforme de studiu. Aplicaţia va putea fi accesată, pe baza unui proces de autentificare, de către mai multe tipuri de utilizatori: studenti, profesori, administratori. Pentru fiecare tip de utilizator se vor reține informații precum CNP, nume, prenume, adresa, număr de telefon, email, cont IBAN, numărul de contract. Fiecare utilizator își va putea vizualiza datele personale imediat după ce va accesa sistemul informatic, fără a avea însă posibilitatea de a le modifica. Utilizatorul de tip administrator poate adăuga, modifica şi şterge informaţii în baza de date, informații legate de utilizatori. Exista și un rol de super-administrator care poate opera inclusiv asupra utilizatorilor de tip administrator. Administratorii pot sa caute utilizatorii dupa nume si ii pot filtra dupa tip, pot asigna profesorii la cursuri si pot face cautare dupa numele cursului. La cautarea unui curs se afiseaza si numele profesorilor de la acel curs si un buton care permite vizualizarea tuturor studentilor inscrisi la cursul respectiv. Pentru un utilizator de tip profesor se vor retine si cursurile predate, numarul minim si numarul maxim de ore pe care le poate preda si departamentul din care face parte. Pentru un utilizator de tip student se va retine si anul de studiu si numarul de ore pe care trebuie sa le sustina.

Aplicatia permite gestiunea cu usurinta a activitatilor didactice si astfel a interacțiunilor dintre studenti si profesori. Cursurile sunt predate de mai multi profesori și au una sau mai multe tipuri de activitati (curs, seminar, laborator), o descriere, si un numar maxim de studenti participanti. Studentii se pot inscrie la cursuri si sunt asignati profesorului cu cei mai putini studenti la data inscrierii. Acestia sunt evaluati cu note pentru fiecare tip de activitate si primesc o nota finala ca medie ponderata intre tipurile de activitati. Profesorul stabileste din interfata grafica impartirea procentuala pe tipurile de activitati. Fiecare activitate se desfasoara recursiv intre doua date, pe o anumita perioada de timp. La asignarea unui profesor la un curs se vor alege tipurile de activitati. Ulterior, profesorul poate programa activitatile (curs, seminar, laborator, colocviu, examen) intr-un calendar, pe zile si ore, specificand si numarul maxim de participanti. Activitatile pot fi programate doar in viitor. Profesorii pot accesa un catalog, unde pot filtra studentii dupa cursuri si le pot adauga note. Cataloagele pot fi descarcate sub forma de fisier.

La logare, studentii si profesorii pot sa isi vada activitatile din ziua curenta sau pot accesa o pagina cu toate activitatile la care sunt asignati / inscrisi. Studentii se pot inscrie la cursuri, pot renunta la cursuri si isi pot vedea notele. Acestia trebuie sa aleaga activitatile la care vor sa participe si pot participa la ele doar daca mai sunt locuri sau nu exista o suprapunere cu o alta activitate. Totodata, studentii se pot inscrie in grupuri de studiu pentru o anumita materie, daca sunt inscrisi la materia respectiva. Acestia pot sa vada toti membrii grupului si sa lase mesaje. Pe grup, studentii pot adauga activitati si sa defineasca un numar minim de participanti si o perioada in care ceilalti studenti pot sa anunte participarea. Daca numarul minim nu este atins, activitatea se anuleaza.

2. SCOPUL APLICATIEI

Aplicatia va folosi un sistem de gestiune pentru baze de date MySQL, iar interactiunea cu aceasta va fi realizata doar prin interfata grafica. Functionalitatile pe care le va oferi programul

vizeaza operatii ce tin de gestiunea studentilor, profesorilor si administrarea operatiilor curente din cadrul unor programe de studiu.

II. ANALIZA CERINTELOR UTILIZATORULUI

1. IPOTEZE SPECIFICE DOMENIULUI ALES PENTRU PROIECT

- Majoritatea studenților vor folosi sistemul pentru a-și vizualiza notele și pentru a se înscrie la cursuri.
- Profesorii vor folosi în primul rând sistemul pentru a programa activități și pentru a atribui note elevilor.
- Administratorii vor utiliza în primul rând sistemul pentru a gestiona informațiile utilizatorilor și alocările profesorilor la cursuri.
- Grupurile de studiu vor fi utilizate în principal de către studenți pentru a colabora și a discuta materialele cursului în afara clasei.
- Sistemul va trebui să se ocupe de un număr mare de utilizatori concurenți în timpul orelor de vârf (de exemplu, la începutul unui semestru, când are loc înscrierea și programarea).
- Sistemul va trebui să asigure integritatea și securitatea datelor, în special atunci când manipulează informații personale sensibile și note.
- Sistemul va trebui să poată gestiona o cantitate mare de date, inclusiv informații despre studenți, informații despre curs și programe de activitate.
- Sistemul va trebui să ofere o interfață intuitivă și ușor de utilizat pentru toate tipurile de utilizatori.
- Sistemul va trebui să fie capabil să gestioneze diferite tipuri de activități (cursuri, seminarii, laboratoare) și să le atribuie diferite ponderi pentru calculul notelor finale.
- Sistemul va trebui să fie capabil să gestioneze conflictele de programare și să se asigure că elevii nu se pot înscrie în activități suprapuse.
- Sistemul va trebui să poată gestiona activitățile de grup, în cazul în care este necesar un număr minim de participanți și se stabilește o perioadă de timp pentru participare.

2. ORGANIZAREA STRUCTURATA A CERINTELOR

Pentru ca obiectivele impuse sa fie atinse am creat urmatoarele tabele cu scopul stocarii informatiilor specifice. Structura acestora poate fi vazuta mai jos:

- utilizator (id_user, CNP, nume, prenume, adresa, email, parola, nr_telefon, cont_iban, nr_cont, nr_contact, tip);
- profesor (id_profesor, nr_min_ore, nr_maxim_ore,cursuri, departament, id_profesor);
- materie (id_materie, profesor_id, denumire_materie, profesor_id, curs, seminar, laborator);
- grup_studiu (id_grup, materie_id, materie_id);
- student (id_student, an_studiu, grup_id, id_student, grup_id);
- administrator (id_administrator, tip, id_administrator);
- inscriere (student_id, student_id, materie_id, materie_id);

- activitate (id_activitate, materie_id, tip_activitate, cologviu, examen, procent, nr_maxim_participanti);
- inscriere_grup (student_id, grup_id);
- mesaj (id_mesaj, continut, student_id, grup_id);
- activitate_grup (id_activitate_grup, descriere_activitate, data_activitate, ora_inceput_activitate, durata_activitate, nr_min_participanti, durata_exp_accept, student_id, grup_id);
- note_activitate (nota, student_id, activitate_id);
- note_finale (nota_finala, student_id, materie_id);

3. DETERMINAREA SI CARACTERIZAREA DE PROFILURI DE UTILIZATORI

Student utilizator: Acest utilizator va utiliza în primul rând sistemul pentru a vizualiza notele lor, înscrieți-vă la cursuri și participa la grupuri de studiu. Aceștia vor avea o capacitate limitată de a modifica informații, cum ar fi înscrierea la cursuri și activități, dar nu vor putea să-și modifice informațiile personale sau informațiile altor utilizatori.

Profesor utilizator: Acest utilizator va folosi în primul rând sistemul pentru a programa activități, pentru a atribui note elevilor și pentru a vizualiza elevii înscriși la cursurile lor. Ei vor putea gestiona activitățile pe care le predau, dar nu vor avea capacitatea de a modifica informațiile utilizatorilor sau de a atribui profesori la cursuri.

Utilizator administrator: Acest utilizator va avea acces deplin la sistem, inclusiv capacitatea de a gestiona informațiile utilizatorilor, de a atribui profesori la cursuri și de a gestiona activitățile și grupurile de studiu. Aceștia vor putea vizualiza și modifica toate informațiile din sistem, dar nu vor avea acces la informații personale sensibile.

Super-administrator utilizator: Acest utilizator va avea același acces ca și administratorul, dar va avea, de asemenea, posibilitatea de a gestiona alți utilizatori administrator.

III. MODELUL DE DATE SI DESCRIEREA ACESTUIA

1. ENTITATI SI ATRIBUTELE LOR

Codul SQL de creare a entitatilor mentionate mai sus este urmatorul:

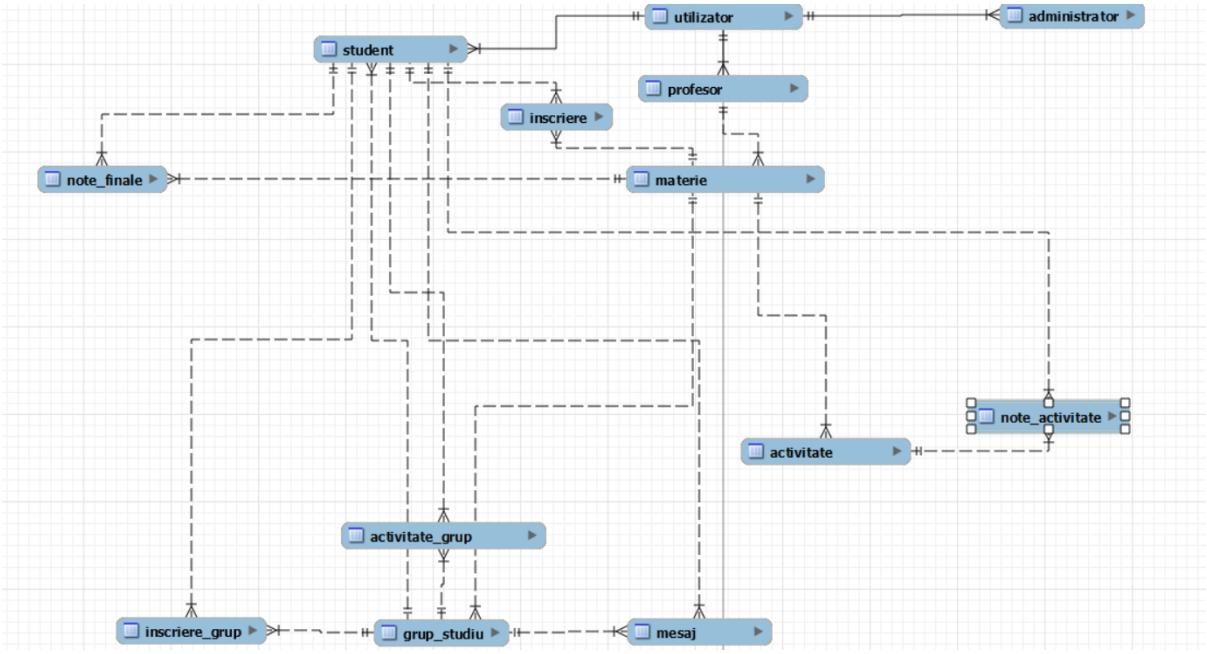
```
DROP TABLE IF EXISTS utilizator;
CREATE TABLE IF NOT EXISTS utilizator (
        id_user int not null primary key,
   CNP varchar(14) NOT NULL,
   nume varchar(30) NOT NULL,
   prenume varchar(30) NOT NULL,
```

```
    adresa varchar(50) NOT NULL,
    email varchar(30) NOT NULL,
    parola varchar(30) NOT NULL,
    nr_telefon varchar(30) NOT NULL,
    cont_iban varchar(30) NOT NULL,
    nr_cont varchar(30) NOT NULL,
    nr_contact varchar(30) NOT NULL,
    tip varchar(30) NOT NULL
);

DROP TABLE IF EXISTS profesor;
CREATE TABLE IF NOT EXISTS profesor (
            id_profesor int not null primary key,
    nr_min_ore int,
    nr_maxim_ore int,
    cursuri varchar(30),
    departament varchar(30),
    FOREIGN KEY (id_profesor) REFERENCES utilizator(id_user)
);

DROP TABLE IF EXISTS materie;
CREATE TABLE IF NOT EXISTS materie (
            id_materie int not null primary key,
            profesor_id int,
    denumire_materie varchar(20),
    FOREIGN KEY (profesor_id) REFERENCES profesor(id_profesor),
    curs BOOLEAN,
    seminar BOOLEAN,
    laborator BOOLEAN
);

DROP TABLE IF EXISTS grup_studiu;
CREATE TABLE IF NOT EXISTS grup_studiu (
            id_grup int not null primary key,
    materie_id int,
    FOREIGN KEY (materie_id) REFERENCES materie(id_materie)
);

DROP TABLE IF EXISTS student;
CREATE TABLE IF NOT EXISTS student (
            id_student int not null primary key,
    an_studiu varchar(10) NOT NULL,
    grup_id int,
    FOREIGN KEY (id_student) REFERENCES utilizator(id_user),
    FOREIGN KEY (grup_id) REFERENCES grup_studiu(id_grup)
);

DROP TABLE IF EXISTS administrator;
CREATE TABLE IF NOT EXISTS administrator (
            id_administrator int not null primary key,
    tip varchar(20),
    FOREIGN KEY (id_administrator) REFERENCES utilizator(id_user)
```

```
);

DROP TABLE IF EXISTS inscriere;
CREATE TABLE IF NOT EXISTS inscriere (
        student_id int,
   FOREIGN KEY (student_id) REFERENCES student(id_student),
   materie_id int,
        FOREIGN KEY (materie_id) REFERENCES materie(id_materie)
);

DROP TABLE IF EXISTS activitate;
CREATE TABLE IF NOT EXISTS activitate (
        id_activitate int not null primary key,
        materie_id int,
        FOREIGN KEY (materie_id) REFERENCES materie(id_materie),
   tip_activitate varchar(20) NOT NULL,
   cologviu BOOLEAN,
   examen BOOLEAN,
   procent varchar(20) NOT NULL,
   nr_maxim_participanti int
);

DROP TABLE IF EXISTS inscriere_grup;
CREATE TABLE IF NOT EXISTS inscriere_grup (
        student_id int,
   grup_id int,
   FOREIGN KEY (student_id) REFERENCES student(id_student),
   FOREIGN KEY (grup_id) REFERENCES grup_studiu(id_grup)
);

DROP TABLE IF EXISTS mesaj;
CREATE TABLE IF NOT EXISTS mesaj (
        id_mesaj int not null primary key,
   continut varchar(50) NOT NULL,
        student_id int,
   grup_id int,
   FOREIGN KEY (student_id) REFERENCES student(id_student),
   FOREIGN KEY (grup_id) REFERENCES grup_studiu(id_grup)
);

DROP TABLE IF EXISTS activitate_grup;
CREATE TABLE IF NOT EXISTS activitate_grup (
        id_activitate_grup int not null primary key,
   descriere_activitate varchar(50) NOT NULL,
   data_activitate date,
   ora_inceput_activitate time,
   durata_activitate time,
   nr_min_participanti int,
   durata_exp_accept time,
        student_id int,
   grup_id int,
   FOREIGN KEY (student_id) REFERENCES student(id_student),
```

```
    FOREIGN KEY (grup_id) REFERENCES grup_studiu(id_grup)
);

DROP TABLE IF EXISTS note_activitate;
CREATE TABLE IF NOT EXISTS note_activitate(
            nota float,
            student_id int,
    activitate_id int,
    FOREIGN KEY (student_id) REFERENCES student(id_student),
    FOREIGN KEY (activitate_id) REFERENCES activitate(id_activitate)
);

DROP TABLE IF EXISTS note_finale;
CREATE TABLE IF NOT EXISTS note_finale(
            nota_finala float,
            student_id int,
    materie_id int,
    FOREIGN KEY (student_id) REFERENCES student(id_student),
    FOREIGN KEY (materie_id) REFERENCES materie(id_materie)
);
```

2. DIAGRAMA UML PENTRU MODELUL DE DATE COMPLETE



3. NORMALIZAREA DATELOR

Normalizarea este ramura teoriei relaționale care oferă informații despre proiectare. Este procesul de determinare a câtă redundanță există într-un tabel. Obiectivele normalizării sunt:

-Să fie capabilă să caracterizeze nivelul de redundanță într-o schemă relațională

-Să furnizeze mecanisme pentru transformarea schemelor pentru a elimina redundanța

Teoria normalizării se bazează puternic pe teoria dependențelor funcționale. Teoria normalizării definește șase forme normale (normal forms, NF). Fiecare formă normală implică un set de proprietăți de dependență pe care o schemă trebuie să le îndeplinească și fiecare formă normală oferă garanții cu privire la prezența și / sau absența anomaliilor de actualizare. Aceasta înseamnă că formele normale mai ridicate au o redundanță mai redusă și, ca urmare, mai puține probleme de actualizare.

O bază de date bine proiectată nu permite ca datele să fie redundante, adică aceeaşi informaţie să se găsească în locuri diferite. De asemenea nu se memorează informaţii care se pot deduce din alte informaţii retinute în baza de date.

In 1970 – 1971 Edgar Codd a definit primele trei forme normale 1NF, 2NF şi 3NF. Ulterior s-au mai definit formele normale 4NF, 5NF, 6NF care însă sunt rar folosite în proiectarea bazelor de date.

Prima formă normală

O entitate se găseşte în prima formă normală dacă şi numai dacă:

- nu există atribute cu valori multiple;

- nu există atribute sau grupuri de atribute care se repetă.

Cu alte cuvinte toate atributele trebuie să fie atomice, adică să conţină o singură informaţie.

A doua formă normală

O entitate se găseşte în a doua formă normală dacă şi numai dacă se găseşte în prima formă normală şi în plus orice atribut care nu face parte din UID (unique identifier) va depinde de întregul UID nu doar de o parte a acestuia.

A treia formă normală

O entitate se găseşte în a treia formă normală dacă şi numai dacă se găseşte în a doua formă normală şi în plus nici un atribut care nu este parte a UID-ului nu depinde de un alt atribut non-UID. Cu alte cuvinte nu se acceptă dependenţe tranzitive, adică un atribut să depindă de UID în mod indirect.

IV. DETALII DE IMPLEMENTARE

1. TEHNOLOGII UTILIZATE

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems la începutul anilor ’90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor.

Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java (engleză Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

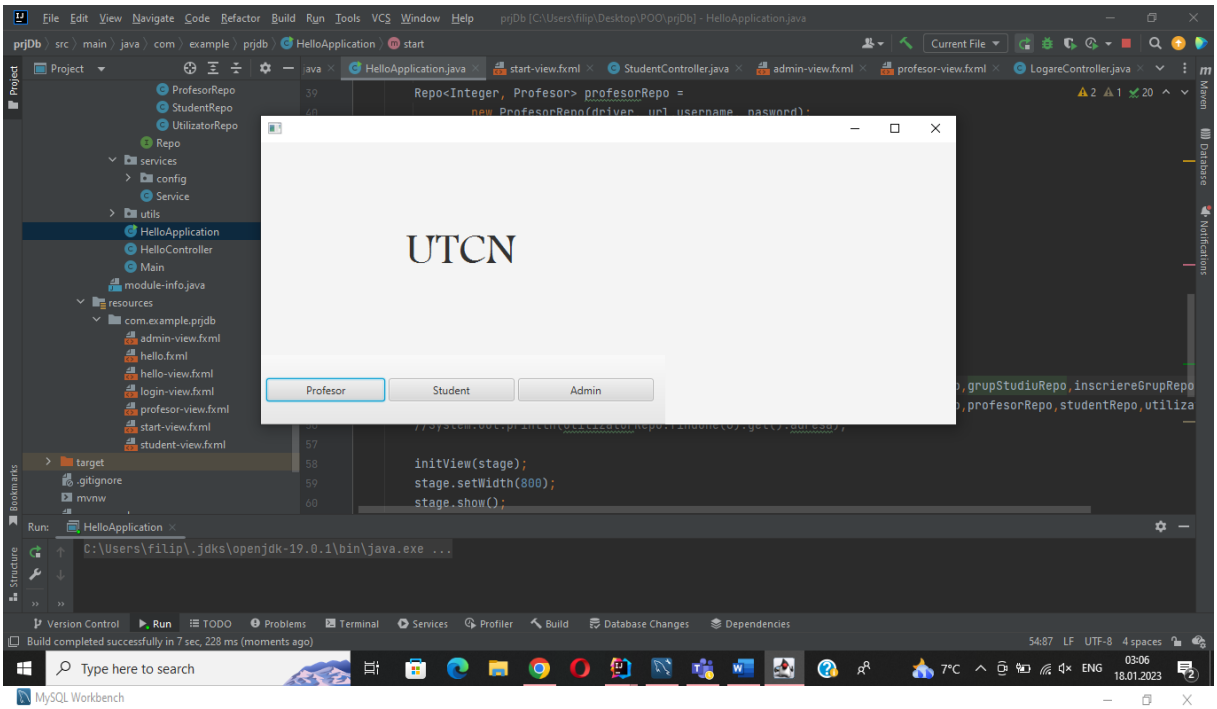
IntelliJ IDEA este un mediu de dezvoltare integrat (IDE) scris în Java pentru dezvoltarea de software de calculator scris în Java, Kotlin, Groovy și alte limbi bazate pe JVM. Acesta este dezvoltat de JetBrains (cunoscut anterior ca IntelliJ) și este disponibil ca o ediție comunitară autorizată Apache 2, și într-o ediție comercială proprietară. Ambele pot fi utilizate pentru dezvoltarea comercială.

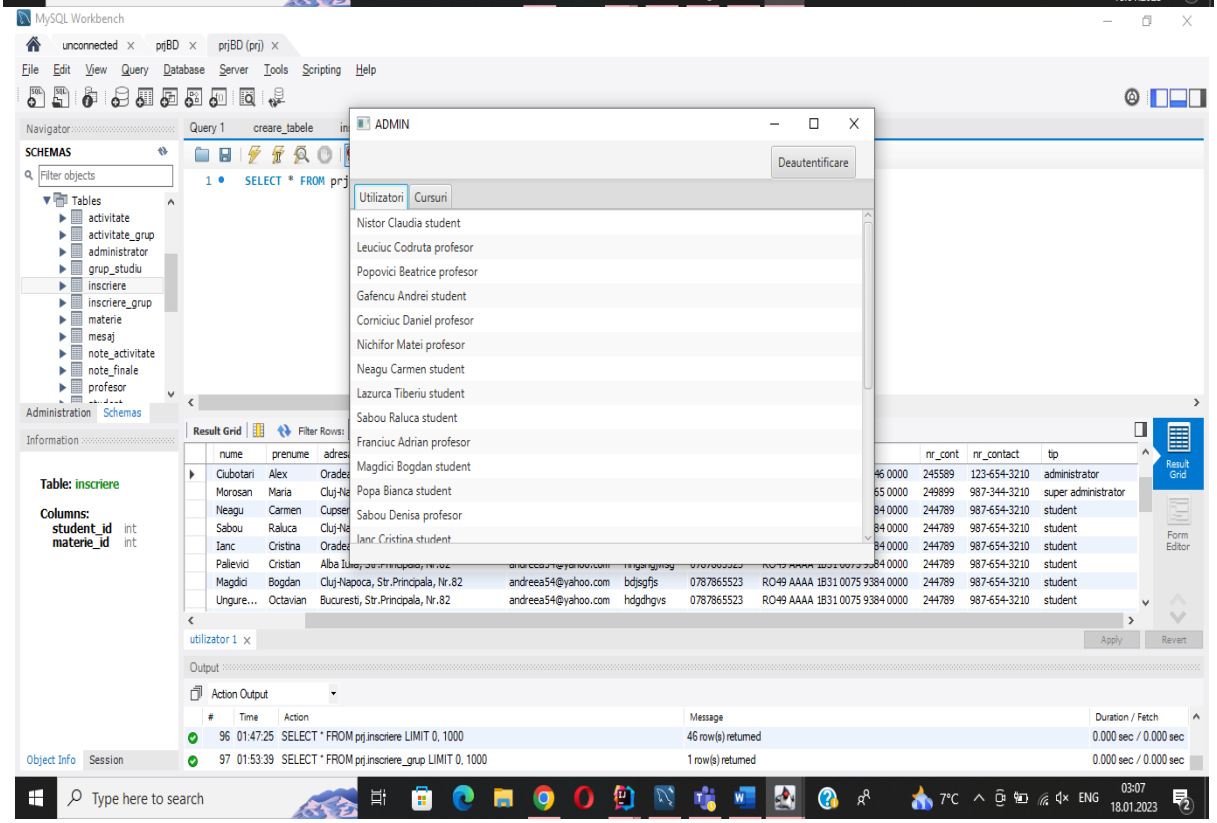
2. INTERFATA GRAFICA

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

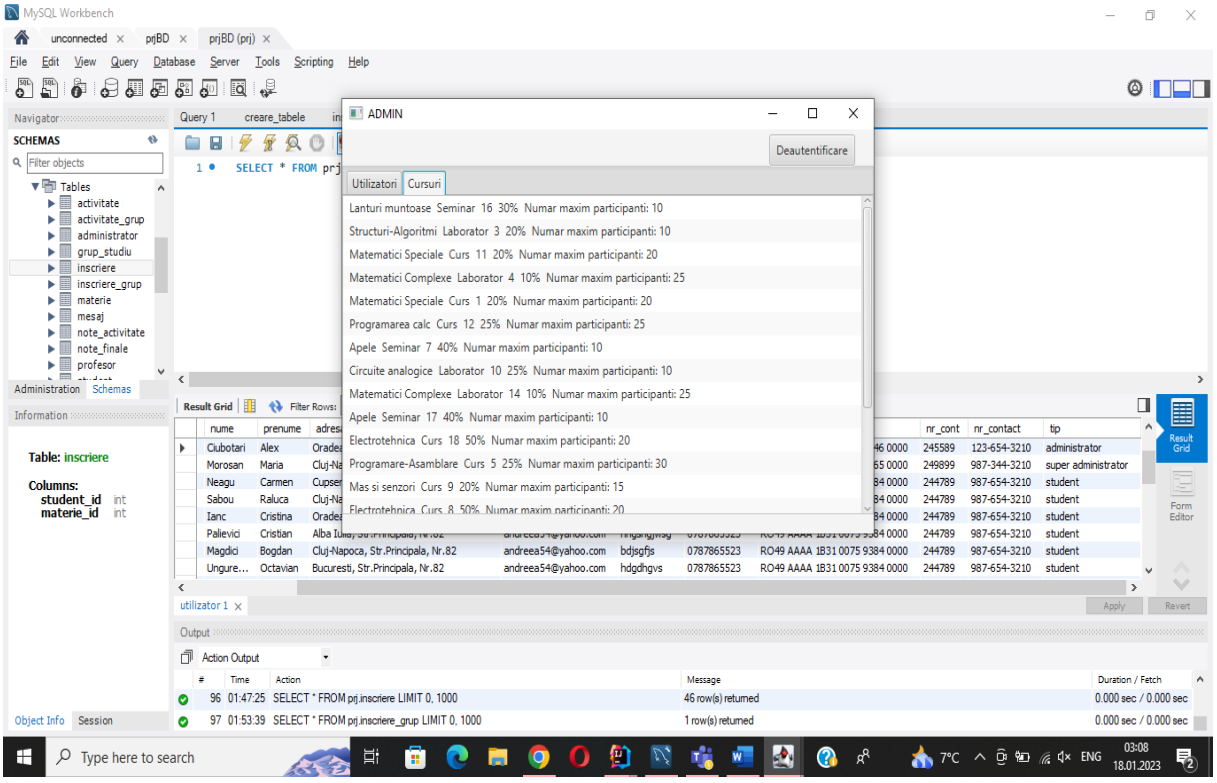


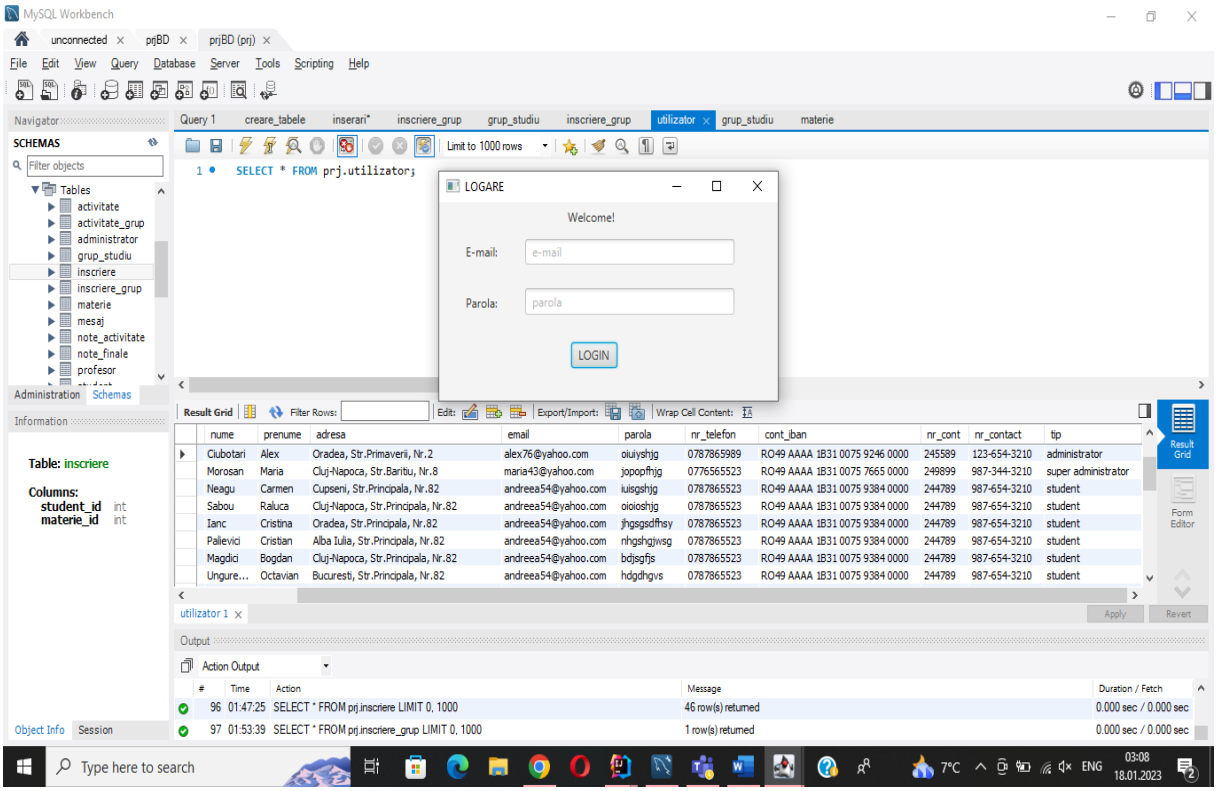


MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

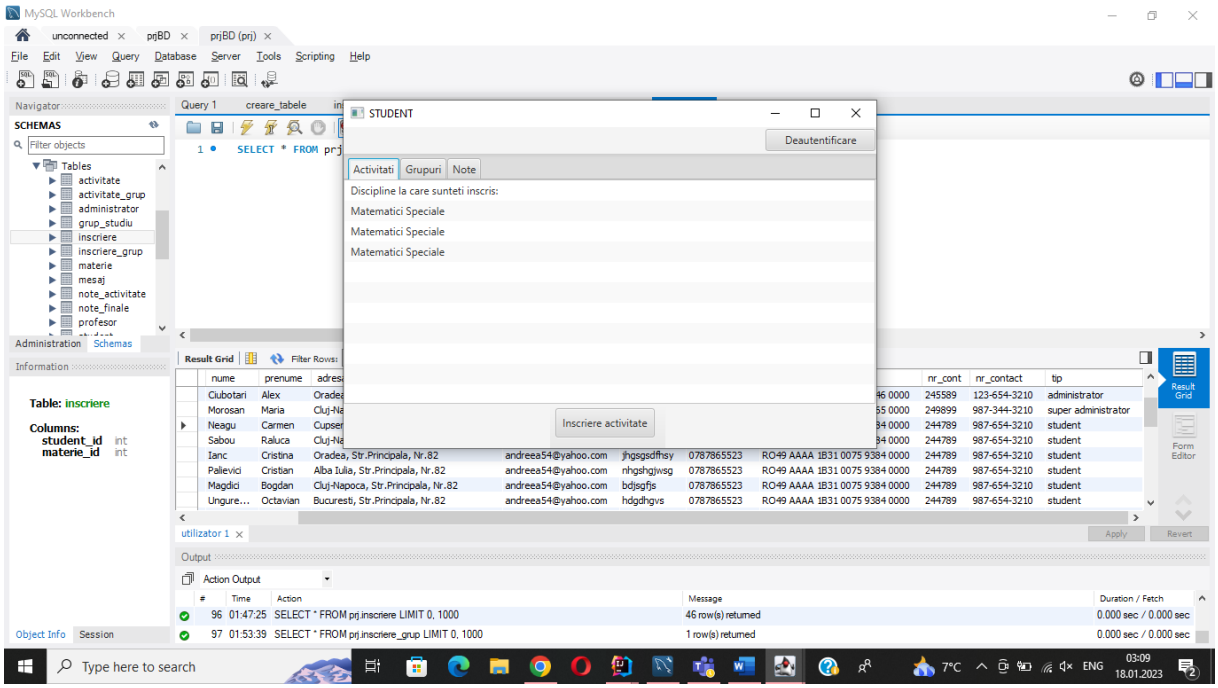


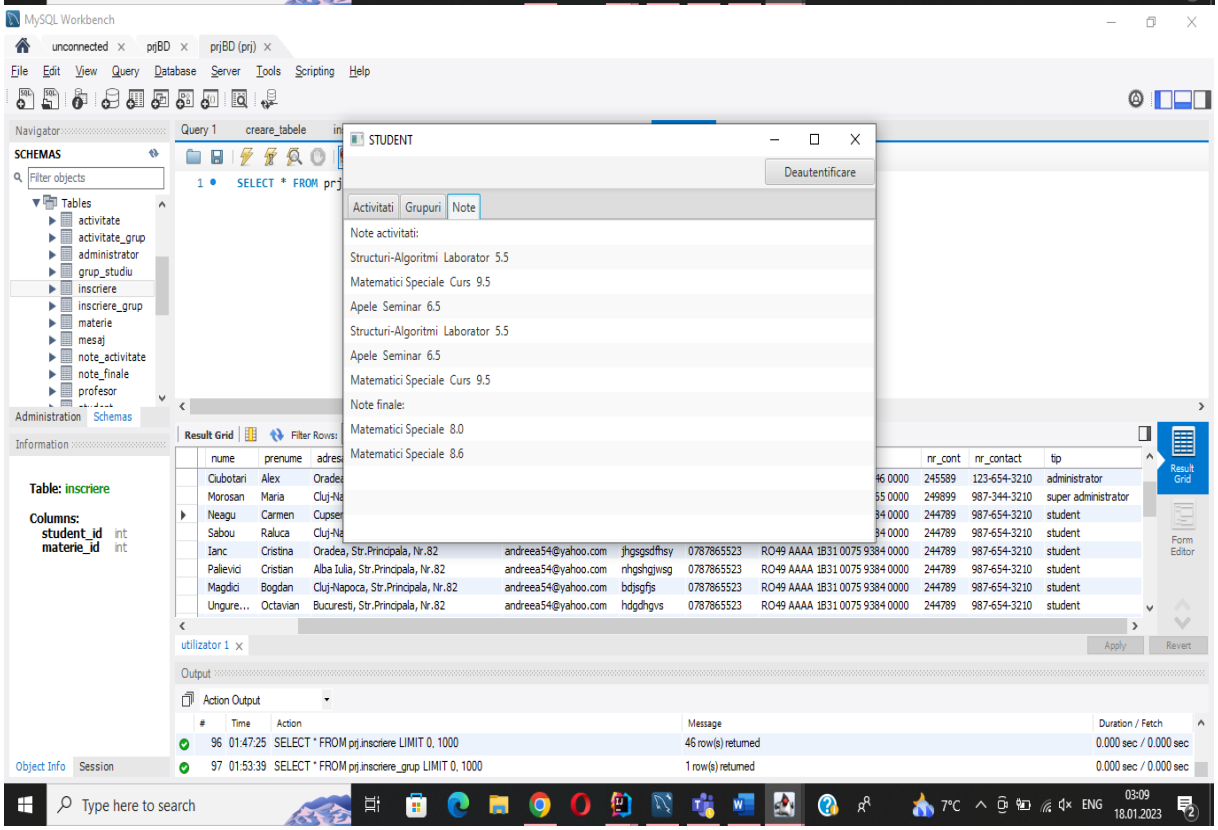


MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

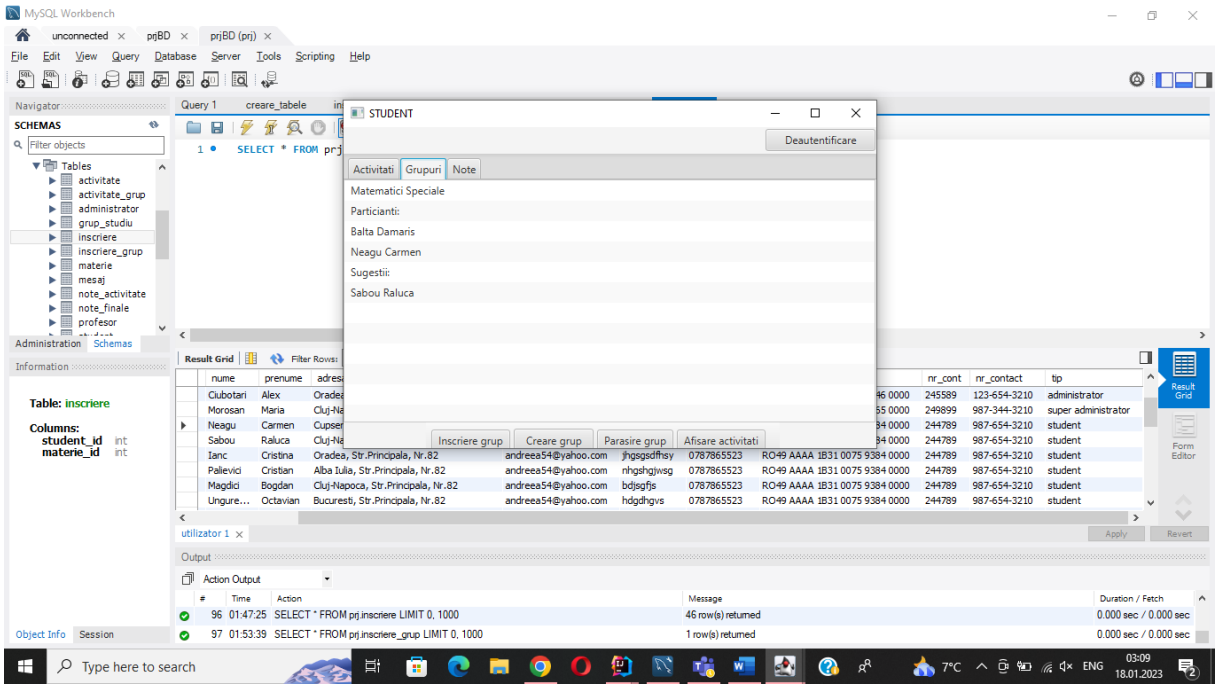


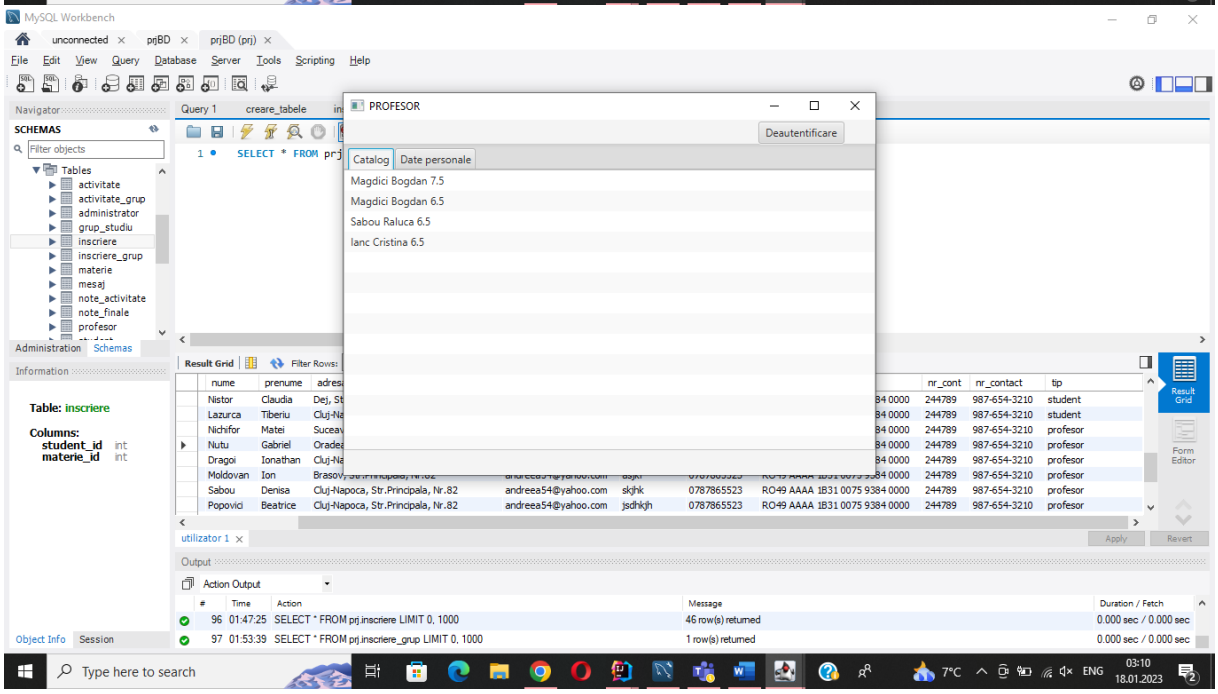


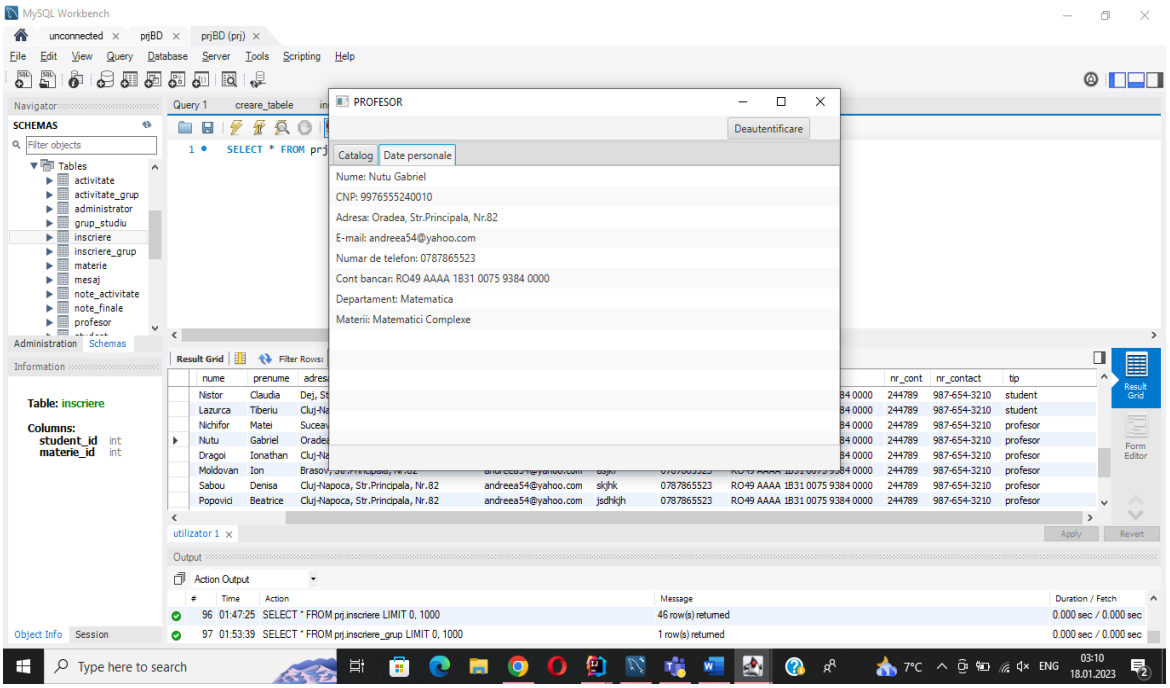
MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE







3. CONCLUZII, LIMITARI SI DEZVOLTARI ULTERIOARE

Concluzii:

Acest program poate fi utilizat pentru managementul activitatilor si inscrierilor studentilor la cursuri si activitati extracurriculare, precum si pentru gestionarea notelelor acestora.

De asemenea, poate fi utilizat pentru crearea si gestionarea grupurilor de studiu si pentru comunicarea intre studenti si profesori.

Limitari:

Programul este limitat la managementul activitatilor si inscrierilor pentru universitati si poate avea dificultati in a fi adaptat pentru alte tipuri de institutii de invatamant sau alte domenii.

Nu exista un sistem de plata integrat pentru taxele de scolarizare sau alte costuri asociate cu activitatile si cursurile.

Programul poate avea probleme cu managementul si afisarea informatiilor in cazul in care sunt prea multe activitati sau studenti.

Dezvoltari ulterioare:

Adaugarea unui sistem de plata pentru taxele de scolarizare si alte costuri asociate cu activitatile si cursurile.

Implementarea unui sistem de notificari pentru studenti si profesori pentru a fi la curent cu activitatile si inscrierile.

V. BIBLIOGRAFIE SI RESURSE

https://ftp.utcluj.ro/pub/users/civan/IBD/3.Evaluari/Proiect/Proiect_IBD_TeamMax_v2.pdf

https://www.telework.ro/ro/normalizarea-bazelor-de-date-forme-normale/

https://atestatinfo.blogspot.com/2010/01/formele-normale-ale-bazelor-de-date.html

https://www.scribd.com/document/554485846/Referat-Despre-Formele-Normale-Ale-Bazei-de-Date#:~:text=Formele%20normale%20ale%20bazei%20de%20date%20sunt%3A%201,4.%20Normalizarea%20bazelor%20de%20date%20-%20bazededatememementoadit%20%28google.com%29

https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)

https://en.wikipedia.org/wiki/IntelliJ_IDEA

VI. ANEXA

1. TRIGGERE

```
DROP TRIGGER IF EXISTS user_delete;
DELIMITER //
CREATE TRIGGER user_delete AFTER DELETE ON utilizator
FOR EACH ROW BEGIN
   DELETE FROM utilizator WHERE id_user = OLD.id_user;
   DELETE FROM autentificare WHERE id_user_autentificare = OLD.id_user;
   DELETE FROM profesor WHERE id_profesor = OLD.id_user;
   DELETE FROM student WHERE id_student = OLD.id_user;
   DELETE FROM administrator WHERE id_administrator = OLD.id_user;
   DELETE FROM inscriere WHERE student_id = OLD.id_user;
   DELETE FROM inscriere_grup WHERE student_id = OLD.id_user;
END; //
DELIMITER ;

DROP TRIGGER IF EXISTS profesor_delete;
DELIMITER //
CREATE TRIGGER profesor_delete AFTER DELETE ON utilizator
FOR EACH ROW BEGIN
  DELETE FROM profesor WHERE id_profesor = OLD.id_user;
END; //
DELIMITER ;

DROP TRIGGER IF EXISTS materie_delete;
DELIMITER //
CREATE TRIGGER  materie_delete AFTER DELETE ON materie
FOR EACH ROW BEGIN
        DELETE FROM materie WHERE id_materie = OLD.id_materie;
END; //
DELIMITER ;

DROP TRIGGER IF EXISTS grup_studiu_delete;
DELIMITER //
CREATE TRIGGER  grup_studiu_delete AFTER DELETE ON materie
FOR EACH ROW BEGIN
  DELETE FROM grup_studiu WHERE materie_id = OLD.id_materie;
END; //
```

```
DELIMITER ;

DROP TRIGGER IF EXISTS student_delete;
DELIMITER //
CREATE TRIGGER student_delete AFTER DELETE ON student
FOR EACH ROW BEGIN
   DELETE FROM student WHERE id_student = OLD.id_student;
END; //
DELIMITER ;


DROP TRIGGER IF EXISTS administrator_delete;
DELIMITER //
CREATE TRIGGER  administrator_delete AFTER DELETE ON utilizator
FOR EACH ROW BEGIN
   DELETE FROM administrator WHERE id_administrator = OLD.id_user;
END; //

DROP TRIGGER IF EXISTS inscriere_delete;
DELIMITER //
CREATE TRIGGER  inscriere_delete AFTER DELETE ON student
FOR EACH ROW BEGIN
  DELETE FROM inscriere WHERE student_id = OLD.id_student;
END; //

DROP TRIGGER IF EXISTS activitate_delete;
DELIMITER //
CREATE TRIGGER  activitate_delete AFTER DELETE ON materie
FOR EACH ROW BEGIN
  DELETE FROM activitate WHERE materie_id = OLD.id_materie;
END; //

DROP TRIGGER IF EXISTS inscriere_grup_delete;
DELIMITER //
CREATE TRIGGER inscriere_grup_delete AFTER DELETE ON student
FOR EACH ROW BEGIN
   DELETE FROM inscriere_grup WHERE student_id = OLD.id_student;
END; //
DELIMITER ;

DROP TRIGGER IF EXISTS mesaj_delete;
DELIMITER //
CREATE TRIGGER mesaj_delete AFTER DELETE ON mesaj
FOR EACH ROW BEGIN
  DELETE FROM mesaj WHERE id_mesaj = OLD.id_mesaj;
END; //

DROP TRIGGER IF EXISTS activitate_grup_delete;
DELIMITER //
CREATE TRIGGER  activitate_grup_delete AFTER DELETE ON activitate_grup
FOR EACH ROW BEGIN
            DELETE FROM activitate_grup WHERE id_activitate_grup = OLD.id_activitate_grup;
END; //
DELIMITER ;

DROP TRIGGER IF EXISTS note_activitate_delete;
DELIMITER //
CREATE TRIGGER  note_activitate_delete AFTER DELETE ON note_activitate
FOR EACH ROW BEGIN
  DELETE FROM note_activitate WHERE student_id = OLD.student_id AND activitate_id = OLD.activitate_id;
END; //
```

```
DROP TRIGGER IF EXISTS note_finale_delete;
DELIMITER //
CREATE TRIGGER note_finale_delete AFTER DELETE ON note_finale
FOR EACH ROW BEGIN
  DELETE FROM note_finale WHERE student_id = OLD.student_id AND materie_id = OLD.materie_id;
END; //
DELIMITER ;
```

2. PROCEDURI STOCATE

```
DROP PROCEDURE IF EXISTS cautareDupaNume;
DELIMITER //
CREATE PROCEDURE cautareDupaNume (nume1 varchar(30), prenume1 varchar(30))
BEGIN
	SELECT* FROM utilizator WHERE nume = nume1 AND prenume = prenume1;
END;//

CALL cautareDupaNume('Pop', 'Andreea');

DROP PROCEDURE IF EXISTS filtrareDupaTip;
DELIMITER //
CREATE PROCEDURE filtrareDupaTip (tip1 varchar(30)) #tipul de utilizator
BEGIN
	SELECT * FROM utilizator WHERE tip = tip1;
END; //

CALL filtrareDupaTip('student');

DROP PROCEDURE IF EXISTS cautareCursDupaNume;
DELIMITER //
CREATE PROCEDURE cautareCursDupaNume (denumire_materie1 varchar(20)) #din tabela materie
BEGIN
	SELECT * FROM materie WHERE denumire_materie = denumire_materie1 AND curs = TRUE;
END; //

CALL cautareCursDupaNume('Matematica');

DROP PROCEDURE IF EXISTS studentiInscrisiCurs;
DELIMITER //
CREATE PROCEDURE studentiInscrisiCurs (tip_activitate1 varchar(20)) #din tabela activitate + inscriere
BEGIN
	SELECT u.nume, u.prenume
	FROM utilizator u
	INNER JOIN student s ON u.id_user = s.id_student
	INNER JOIN inscriere i ON s.id_student = i.student_id
	INNER JOIN materie m ON i.materie_id = m.id_materie
	INNER JOIN activitate a ON m.id_materie = a.materie_id
	WHERE a.tip_activitate = tip_activitate1;
END; //

#CALL studentiInscrisiCurs('curs');

DROP PROCEDURE IF EXISTS noteStudent;
DELIMITER //
CREATE PROCEDURE noteStudent (nume1 varchar(30), prenume1 varchar(30))
BEGIN
	SELECT note_activitate.*, materie.denumire_materie FROM note_activitate
```

```
        INNER JOIN materie ON note_activitate.materie_id = materie.id_materie
        WHERE note_activitate.nume = nume1 AND note_activitate.prenume = prenume1;
END; //

#CALL noteStudent('Buda', 'Andreea');

DROP PROCEDURE IF EXISTS noteStudentMaterie;
DELIMITER //
CREATE PROCEDURE noteStudentMaterie (nume1 varchar(30), prenume1 varchar(30), denumire_materie1 varchar(20))
BEGIN
        SELECT nota_finala FROM note_finale WHERE nume = nume1 AND prenume = prenume1 AND
denumire_materie = denumire_materie1;
END; //

#CALL noteStudentMaterie('Buda', 'Andreea', 'Matematica');

DROP PROCEDURE IF EXISTS noteFinaleMaterie;
DELIMITER //
CREATE PROCEDURE noteFinaleMaterie (nume_prof1 varchar(30), prenume_prof1 varchar(20), denumire_materie1
varchar(20))
BEGIN
        SELECT student.nume, student.prenume, note_activitate.nota FROM student
        INNER JOIN inscriere ON student.id_student = inscriere.student_id
        INNER JOIN materie ON inscriere.materie_id = materie.id_materie
        INNER JOIN profesor ON materie.profesor_id = profesor.id_profesor
        INNER JOIN note_activitate ON student.id_student = note_activitate.student_id
        WHERE profesor.nume = nume_prof1 AND profesor.prenume = prenume_prof1 AND materie.denumire_materie
= denumire_materie1;
END; //

#CALL noteFinaleMaterie('Buda', 'Andreea', 'Matematica');

DROP PROCEDURE IF EXISTS  nrCursuriProfesori;
DELIMITER //
CREATE PROCEDURE nrCursuriProfesori (nume_prof1 varchar(30), prenume_prof1 varchar(20))
BEGIN
        SELECT COUNT(*) as nr_cursuri FROM materie
        INNER JOIN profesor ON materie.profesor_id = profesor.id_profesor
        WHERE profesor.nume = nume_prof1 AND profesor.prenume = prenume_prof1;
END; //

#CALL nrCursuriProfesori('Buda', 'Andreea');

DROP PROCEDURE IF EXISTS nrActivitatiProgramateGrup;
DELIMITER //
CREATE PROCEDURE nrActivitatiProgramateGrup (denumire_materie1 varchar(20))
BEGIN
        SELECT COUNT(*) as nr_activitati FROM grup_studiu
        INNER JOIN activitate_grup ON grup_studiu.id_grup = activitate_grup.grup_id
        WHERE materie.denumire_materie=denumire_materie1;
END; //

#CALL nrActivitatiProgramateGrup ('Matematica');

DROP PROCEDURE IF EXISTS studentiDinGrup;
DELIMITER //
CREATE PROCEDURE studentiDinGrup (denumire_materie1 varchar(20))
```

```
BEGIN
        SELECT nume, prenume FROM student
   INNER JOIN grup_studiu ON student.grup_id = grup_studiu.id_grup
   WHERE materie.denumire_materie=denumire_materie1;
END; //

#CALL studentiDinGrup ('Matematica');

DROP PROCEDURE IF EXISTS mesajeGrup;
DELIMITER //
CREATE PROCEDURE mesajeGrup (denumire_materie1 varchar(20))
BEGIN
        SELECT continut FROM mesaj
   INNER JOIN grup_studiu ON mesaj.grup_id = grup_studiu.id_grup
   WHERE materie.denumire_materie=denumire_materie1;
END; //
```