**Proiect echivalent examen scris**

**Grigoraș Ana-Maria**

**10LF312**

**Informatică Aplicată**

*Cerința proiectului:*

Pornind de la o bază de date deja construită în FN3, creați un program care permite utilizatorului să facă operații CRUD pe baza de date mai sus menționată (inserare, ștergere, editare, vizualizare). Pentru această cerință se pot folosi adnotari JPA, iar pentru conectarea bazei de date la IntelliJ se va folosi PostgreSQL. La final scrieți pașii de rulare a aplicatiei pentru oricine ar dori sa testeze proiectul.

*Documentație și legendă:*

* Baza de date a fost creată cu scopul de a gestiona eficient un cinematograf. Aceasta oferă informații despre derularea filmelor, sălile cinematografului, bilete, clienți, angajați și departamente.
* Filmele sunt descrise prin date precum id, titlu, gen, data apariției și regizor, iar în fiecare zi sunt difuzate un număr fix de filme în anumite intervale de timp ale zilei. Unui film îi pot fi asociate mai multe derulări. Un film poate avea mai multe genuri și mai mulți regizori. De asemenea, un regizor poate regiza mai multe categorii de filme.
* Fiecare bilet prezintă informații despre sală, rând, loc și preț, având asociat id-ul unei derulări. Prețul unui bilet diferă în funcție de zi, de ora de rulare, de formatul acestuia și categoria biletului achiziționat. Copiii sub 7 ani, elevii, studenții și pensionarii beneficiază de reduceri la achiziționarea biletului, pe baza unui act de identitate și / sau a carnetului de note/de student.
* Fiecare persoană are un CNP, nume, prenume și număr de telefon. Un angajat este o persoană căreia i se adaugă un salariu, fiind repartizată într-un departament. Un client este o persoană care cumpără un bilet în funcție de categoria sa (copil, elev, student, adult, pensionar).
* Fiecare sală de cinematograf are un id, numărul sălii și numărul de locuri. Unui angajat îi pot fi repartizate una sau mai multe săli. Pentru fiecare loc din sală este asociat un bilet, iar fiecare derulare se petrece în câte o sală.
* <https://www.jetbrains.com/idea/guide/tutorials/your-first-spring-application/creating-spring-boot-application/> (pentru crearea aplicatiei):
* <https://www.jetbrains.com/help/idea/postgresql.html> (pentru legarea bazei de date de IntelliJ);
* <https://www.bezkoder.com/spring-boot-postgresql-example> (pentru dependente si structura proiectului, respectiv clase repository, controller, model, data to object-DTO).
* <https://medium.com/javarevisited/hibernate-vs-jpa-vs-spring-data-jpa-ff4485aaa780> (hibernate)

*Principalele elemente de programare utilizate:*

* Am configurat proeictul de tip *Spring* pentru a seta dependentele si a avea acces la Jakarta Persistence, dar si pentru alte utilitati (@ Autowired-injecteaza instanta portrivita in obiectul care solicita dependenta, @Service-depedentele sunt injectate mai usor, @Repository-furnizeaza metode si operatii pentru a manipula datele, gestionare automata a exceptiilor);
* Am utilizat framework-ul *Hibernate* pentru a scrie mai usor operati CRUD, interogari asupra bazei de date si pentru a lucra direct cu obiectele din Java.
* Am folosit Jakarta Persistence (JPA) pentru stocarea, accesul si gestionarea datelor intre obiectele din Java si baza de date din PostgreSQL (ex: @ Entity, @Column, @ld etc);
* Ca si plug-in am utilizat Lombok - genereaza automat getteri, setteri, constructori, metoda *toString()* ca sa afisam informatiile personalizat.

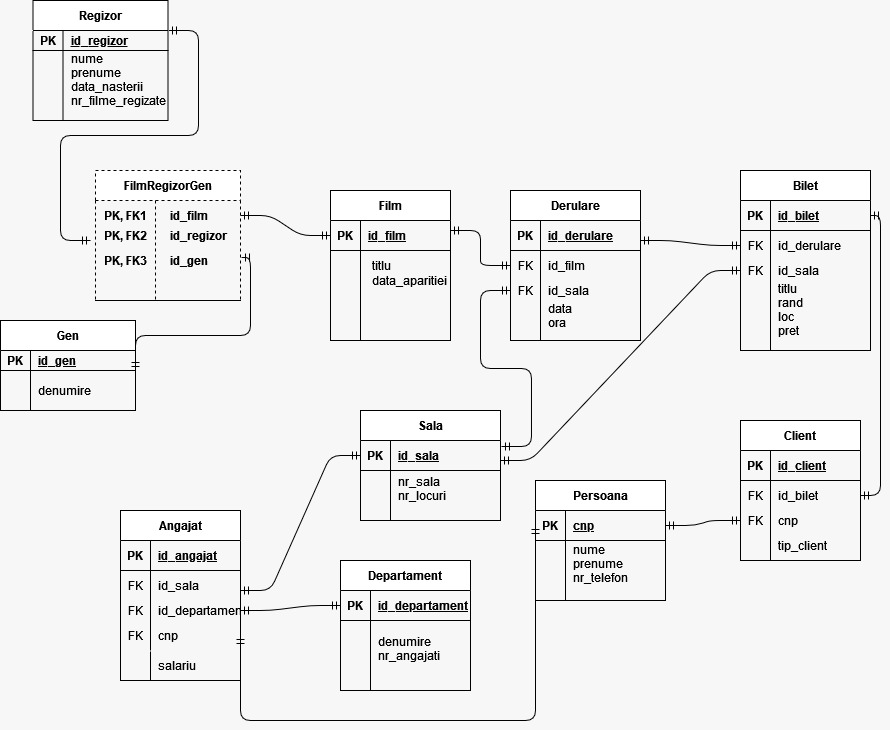
*Entități:*

* + Film,
  + Regizor,
  + Gen,
  + Film\_Regizor\_Gen,
  + Bilet,
  + Derulare,
  + Persoană,
  + Angajat,
  + Departament,
  + Client,
  + Sală.

*Relații, determinarea cardinalității și a participării în tipurile de relații în baza de date:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | **Participare** | |
| **Tip de entitate** | **Tip de relație** | **Tip de entitate** | **Cardinalitate** | **Directă** | **Inversă** |
| Film | *are* | Derulare | C:M | parțială | totală |
| *are* | Regizor | MC:N | parțială | totală |
| *are* | Gen | M:N | totală | totală |
| Regizor | *regizează* | Gen | MC:N | parțială | totală |
| Derulare | *are* | Bilet | 1:M | totală | totală |
| Sală | *are* | Derulare | C:1 | parțială | totală |
| *are* | Bilet | C:M | parțială | totală |
| *este întreținută de* | Angajat | M:1 | totală | totală |
| Client | *cumpără* | Bilet | 1:MC | totală | parțială |
| *este* | Persoană | 1:C | totală | parțială |
| Angajat | *este* | Persoană | 1:C | totală | parțială |
| Departament | *conține* | Angajat | 1:M | totală | totală |

*Diagrama bazei de date:*

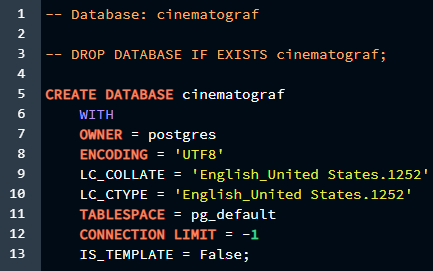


*Cod folosit pentru crearea bazei de date oferit în scopul ajutorului cititorului la implementarea proiectului pe calculatorul propriu:*

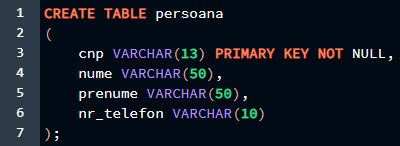
Pentru crearea / restabilirea bazei de date se urmează pașii indicați de următorul articol: <https://hevodata.com/learn/pgadmin-backup-database/>

În cazul în care cititorul dorește să implementeze baza de date de la 0, acesta este codul necesar:

1. Cod SQL pentru baza de date Cinematograf:
   1. Crearea bazei de date:



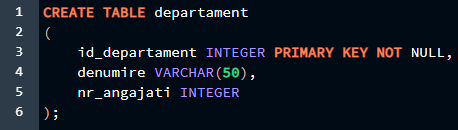
* 1. Crearea tabelei persoană:



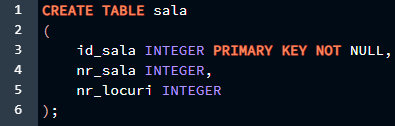
De asemenea, am adaugat o constrângere la tabela persoană care sa garanteze ca CNP-ul să fie de lungime 13:



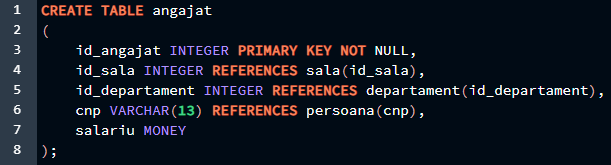
* 1. Crearea tabelei departament:



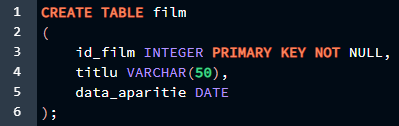
* 1. Crearea tabelei sala:



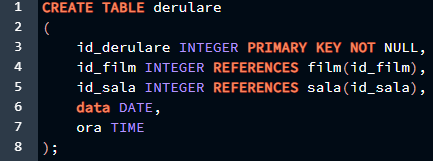
* 1. Crearea tabelei angajat:



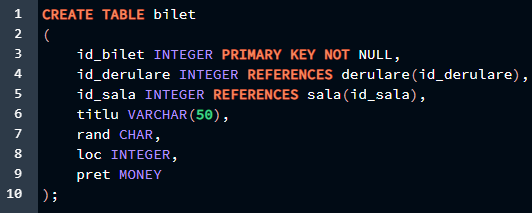
* 1. Crearea tabelei film:



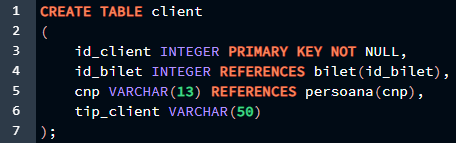
* 1. Crearea tabelei derulare:



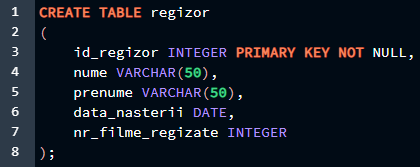
* 1. Crearea tabelei bilet:



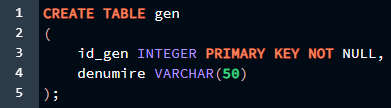
* 1. Crearea tabelei client:



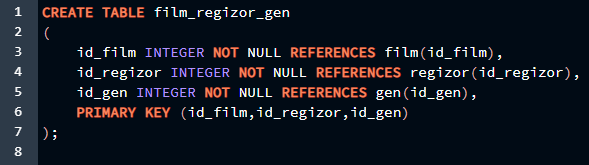
* 1. Crearea tabelei regizor:



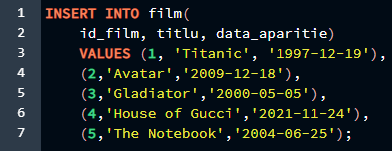
* 1. Crearea tabelei gen:

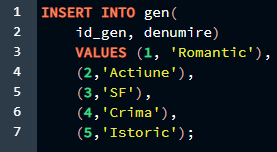


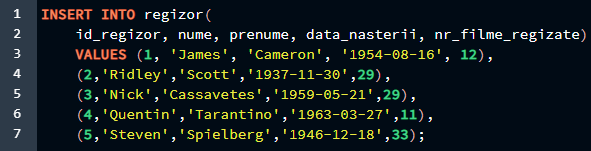
* 1. Crearea tabelei film\_regizor\_gen:

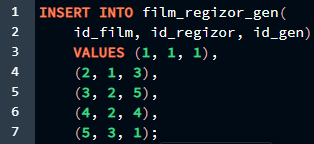


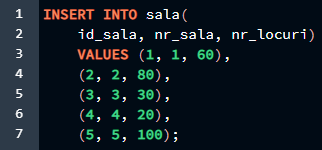
* 1. Inserarea de date în tabele:

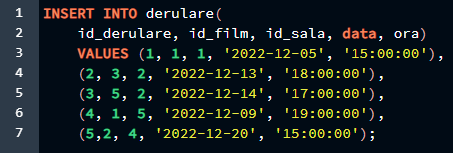


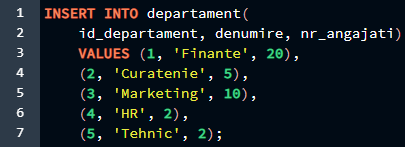


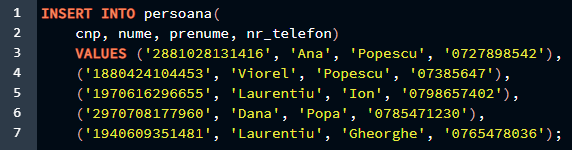


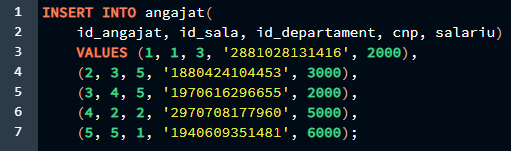




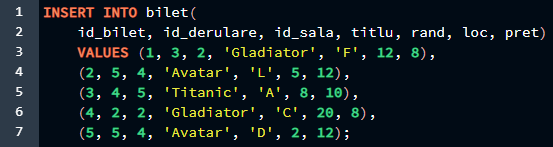


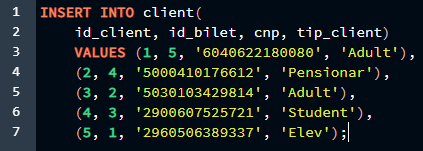






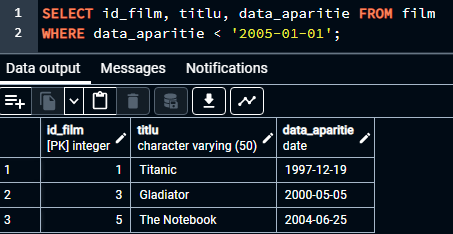




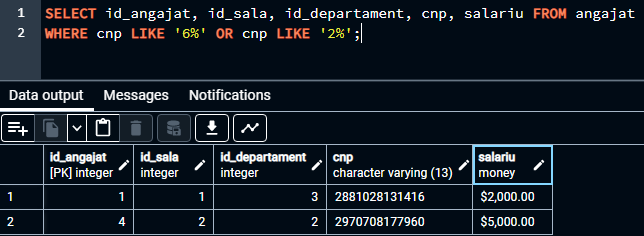


*În continuare se vor prezenta cateva operații efectuate asupra bazei de date (interogări simple/ folosind JOIN, subinterogări, proceduri, triggere și funcții):*

* 1. Interogări simple:
     1. Afișarea filmelor care au aparut înainte de 2005:



* + 1. Afișarea angajaților de sex feminin:



* + 1. Afișarea biletelor în ordine crescătoare după preț:



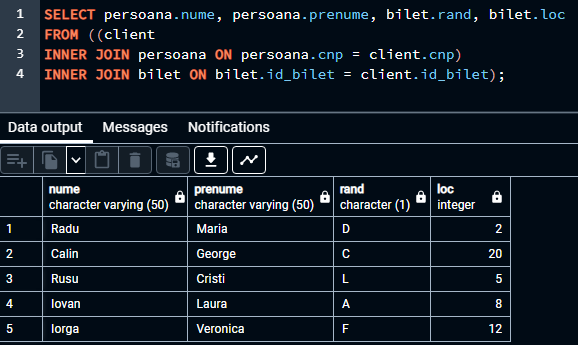
* 1. Interogări cu JOIN:
     1. Afișarea numelui, prenumelui și a tipului de client:



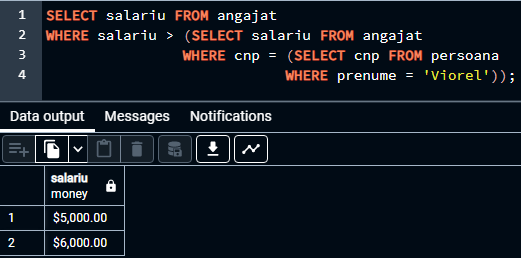
* + 1. Aflarea salariului fiecărui angajat:



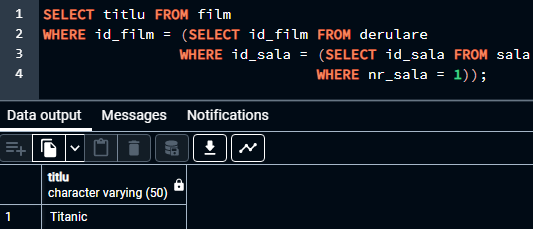
* + 1. Aflarea numelui și al prenumelui persoanelor care au achiziționat bilete:



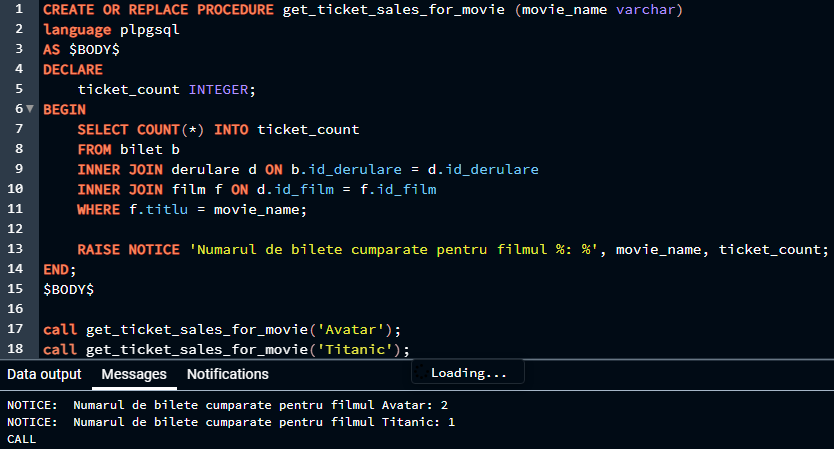
* 1. Subinterogări:
     1. Afișarea salariilor mai mari decât al lui Viorel:



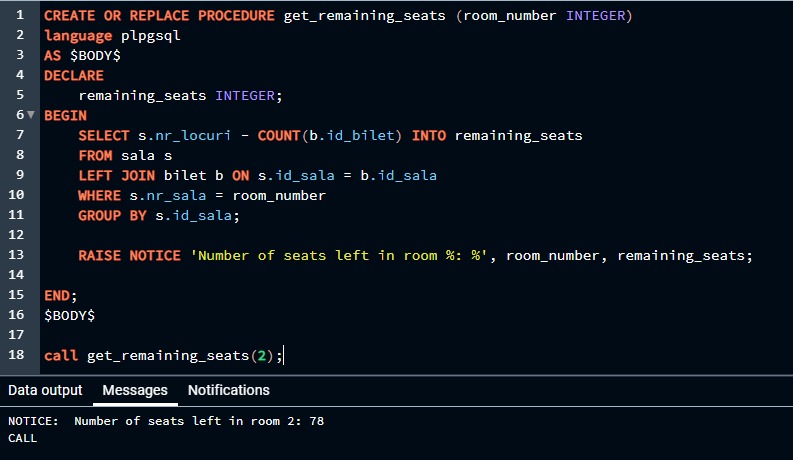
* + 1. Afișarea filmelor care se difuzează în sala numărul 1:



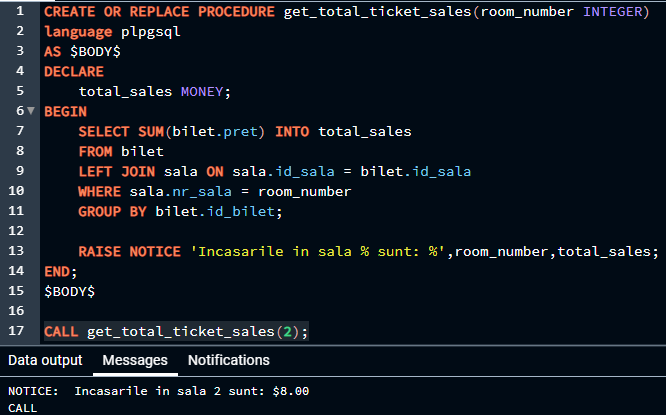
* 1. Proceduri:
     1. Procedură creată pentru afișarea numărului de bilete vândute pentru fiecare film:



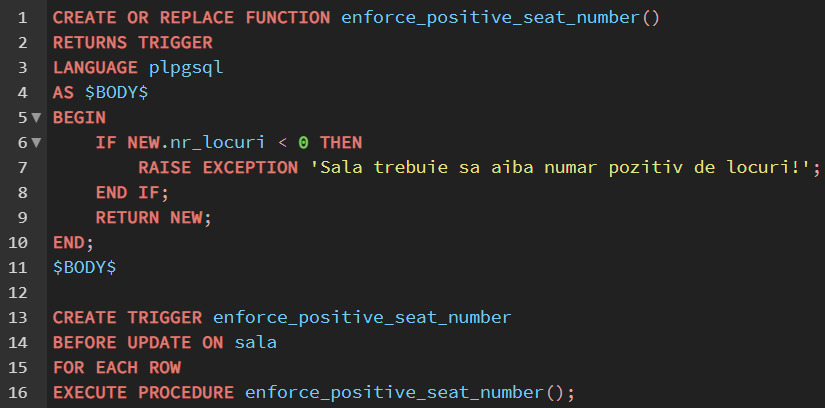
* + 1. Procedură creată pentru a afișa numărul de locuri rămase libere dintr-o anumită sală:

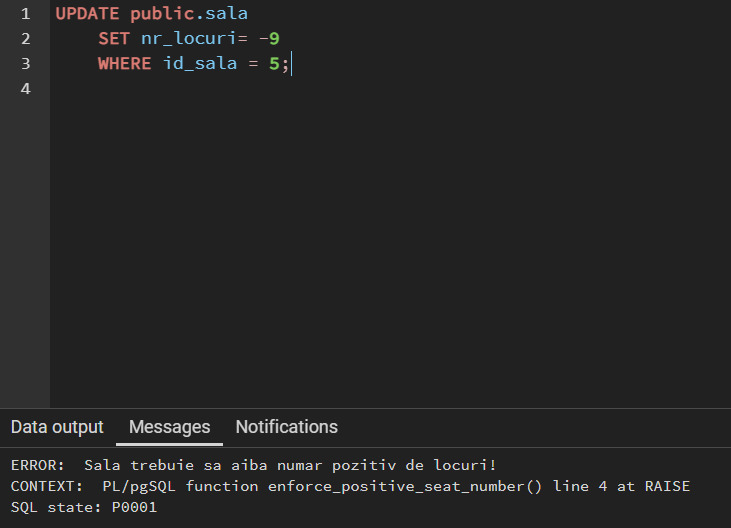


* + 1. Procedură pentru a afișa încasările dintr-o anumită sală:



* 1. Triggere:
     1. Funție ce verifică dacă numărul de locuri al sălii este pozitiv înainte de a face UPDATE:





* + 1. Funție care crește numărul de angajați de fiecare dată cand se adauga un angajat nou:

