|  |  |
| --- | --- |
| https://aciee.ugal.ro/images/logo.png | **FACULTATEA DE AUTOMATICĂ, CALCULATOARE, INGINERIE ELECTRICĂ SI ELECTRONICĂ** UNIVERSITATEA ”DUNĂREA DE JOS” DIN GALAŢI Str. Științei nr. 2, Tel.: **+40 336 130 236**  800146 - Galaţi, România Fax: **+40 236 470 905**  [*https://aciee.ugal.ro*](https://aciee.ugal.ro) |

**Str. Ştiinţei nr. 2, Galaţi - 800146, România**

**UNIVERSITATEA DUNĂREA DE JOS GALAŢI**

**FACULTATEA DE AUTOMATICA, CALCULATOARE, INGINERIE ELECTRICĂ ȘI ELECTRONICĂ**

**SPECIALIZAREA: CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI**

Gestionarea stocurilor unui depozit

de materiale de constructii

Indrumator: s.l. Daniel Munteanu

Disciplina: Baze de Date II

Student: Dudu Florin grupa:22C31-A

1. **Tema**

Aplicația mea „Warehouse” reprezinta un sistem complex de gestionare al unui depozit cu materiale de constructii, conceput pentru a facilita și eficientiza procesele de stocare, gestionare al inventarului, procesarea comenzilor și administrarea relațiilor cu diversele entități implicate, cum ar fi angajații, clienții, furnizorii și transportatorii.

Scopul principal al aplicației este de a oferi o soluție integrată pentru urmărirea și administrarea eficientă a tuturor activităților care au loc în cadrul unui depozit.

1. **Analiza Problemei**
2. Realitatea Modelată

Aplicația reflectă o serie de procese și activități care au loc într-un depozit si anume:

* gestionarea Stocurilor: urmărirea materialelor și produselor stocate, inclusiv intrările și ieșirile din stoc.
* procesarea Comenzilor: Primirea, procesarea și expedierea comenzilor către clienți.
* administrarea Relațiilor: Interacțiunea cu angajații, clienții, furnizorii și transportatorii pentru a asigura o funcționare eficientă.

1. Entități și Interacțiuni

* Angajat: persoane care lucrează în depozit. Fiecare angajat are un rol specific (de exemplu, manager, lucrător depozit) și este responsabil pentru anumite activități (de exemplu, manipularea materialelor, procesarea comenzilor).
* Client: entități sau persoane care plasează comenzi la depozit. Relația cu clienții implică primirea comenzilor, procesarea acestora și asigurarea satisfacției clienților.
* Comandă: o solicitare de produse sau materiale. O comandă include detalii precum cantitatea, tipul de produse și termenul de livrare. Procesarea comenzilor implică pregătirea și expedierea articolelor către clienți.
* Evaluare: feedback primit de la clienți sau alte entități cu privire la serviciile depozitului. Evaluările pot influența îmbunătățirile în servicii și procese.
* Furnizor: Entități care furnizează materiale sau produse depozitului. Relația cu furnizorii include negocierea termenilor de achiziție și menținerea unui flux constant de materiale.
* Material: Articole sau produse stocate în depozit. Gestionarea eficientă a materialelor implică urmărirea stocurilor, asigurându-se că există suficiente resurse pentru a satisface cererea.
* transportator: Companii sau persoane responsabile pentru transportul materialelor sau produselor de la și către depozit. Colaborarea eficientă cu transportatorii este esențială pentru livrările prompte și sigure.

1. Acțiuni și Procese

Fiecare entitate desfășoară acțiuni specifice care contribuie la funcționarea generală a depozitului. De exemplu, angajații procesează comenzi, actualizează inventarul, furnizorii livrează materialele, iar transportatorii se ocupă de logistica de livrare. Toate aceste acțiuni sunt interconectate și necesită o coordonare atentă pentru a asigura eficiența și precizia în procesele depozitului.

1. **Specificarea cerintelor**

In momentul in care am primit tema si am incercat sa conturez scopul acestei aplicatii m-am gandit la rolul unui manager sau rolul unui administrator al unui depozit de materiale. Intentia mea a fost sa ma identific cu rolul administratorului acestui depozit si sa ma gandesc ce mi-as dori sa pot vizualiza si controla in cadrul acestei afaceri. Prin urmare am inceput sa implementez metodele CRUD (Create, Read, Update, Delete) metode ce sunt fundamentale pentru orice sistem de gestionare a unui depozit de materiale „Warehouse”. Aceste metode permit manipularea eficientă a datelor și sunt esențiale pentru interacțiunea cu baza de date.

In continuare voi prezenta cum fiecare dintre aceste operațiuni CRUD este utilizată în contextul diferitelor entități ale aplicației tale:

***3.1. Create (Creare)***

Angajat: Adăugarea unui nou angajat în sistem, cu toate detaliile relevante (id angajat, nume, funcție, date de contact).

Client: Înregistrarea unui nou client, inclusiv informații de contact și preferințe.

Comandă: Crearea unei noi comenzi de la un client, cu detalii despre produsele solicitate.

Evaluare : Primirea unei noi noi evaluari din partea clientilor.

Furnizor/Material: Adăugarea unui nou furnizor sau a unor noi materiale în inventar.

Transportatori: Introducerea unui nou transportator sau selectarea acestuia in momentul in care se efectueaza o comanda.

***3.2. Read (Citire)***

Angajat: Vizualizarea detaliilor despre angajați, inclusiv roluri și performanță.

Client: Accesarea istoricului comenzi și profilului clientului.

Comandă: Verificarea detaliilor și stadiului curent al unei comenzi.

Evaluare: Citirea feedback-ului primit de la clienți pentru îmbunătățirea serviciilor.

***3.3. Update (Actualizare)***

Angajat: Modificarea informațiilor unui angajat, cum ar fi schimbarea funcției sau actualizarea datelor de contact.

Client: Actualizarea informațiilor clientului sau a preferințelor acestuia.

Comandă: Modificarea detaliilor unei comenzi, cum ar fi schimbarea cantității sau a produselor comandate.

***3.4. Delete (Ștergere)***

Angajat: Înlăturarea unui angajat din sistem când acesta nu mai lucrează în depozit.

Client: Ștergerea unui client din baza de date, de exemplu, în cazul în care acesta renunță la servicii.

Comandă: Anularea unei comenzi înainte de procesarea sau expedierea acesteia.

Furnizor/Material: Înlăturarea unui furnizor sau a unor materiale din inventar.

Implementarea eficientă a acestor metode CRUD în aplicația mea asigură că toate operațiunile de gestionare a depozitului pot fi executate fluent și corect. Aceasta facilitează, de asemenea, menținerea datelor actualizate și preciziei în întregul sistem.

***3.5. Log In***

Sistemul de autentificare introdus oferă un acces securizat la aplicație, asigurând că numai utilizatorii autorizați pot vedea și modifica datele. Acesta este un aspect esențial pentru protejarea informațiilor confidențiale.

***3.6. Filtrare și Căutare***

Funcționalitatea de filtrare și căutare permite utilizatorilor să restrângă setul de date afișat pe baza unor criterii specifice. De exemplu, în imaginea încărcată, utilizatorii pot filtra angajații după ID, nume, funcție și date de contact. Acest lucru facilitează găsirea rapidă a informațiilor dorite fără a trebui să navighezi prin întreaga bază de date.

Toate aceste caracteristici împreună constituie o aplicație robustă, care nu numai că facilitează gestionarea eficientă a informațiilor dar și asigură securitatea și integritatea datelor în cadrul organizației tale.

In continuare voi prezenta organizarea codului în pachete și clase, așa cum se poate observa în aplicația mea Warehouse, aceasta oferă mai multe beneficii semnificative printre care:

1. **Pachetul dao (Data Access Object) –** are rolul de a interacționa direct cu

baza mea de date. Clasele precum AngajatDAO, ClientDAO etc., conțin metode pentru operațiile CRUD pentru fiecare entitate.

Beneficii: Separă logica de acces la baza de date de restul aplicației, facilitând modificările în structura bazei de date fără a afecta alte părți ale codului.

1. **Pachetul service** – are rolul de a oferi o interfață de nivel înalt pentru

operațiunile de afaceri. Clase precum AngajatService, ClientService etc., conțin logica aplicației.

Beneficii: Decuplează logica de afaceri de accesul la date, ceea ce permite schimbări în implementarea serviciilor fără a afecta alte părți ale codului.

1. **Pachetul model** - conține clasele entități precum Angajat, Client etc., care

reprezintă structurile de date utilizate în aplicație.

Beneficii: Oferă un mod structurat și centralizat de a gestiona datele, ușurând astfel înțelegerea și manipularea acestora.

1. **Pachetul factory** - conține clase care creează instanțe ale serviciilor, cum ar

fi ServiceFactory.

Beneficii: Simplifică crearea și gestionarea instanțelor de servicii, facilitând injecția de dependențe și gestionarea dependențelor între servicii.

1. **Pachetul util** - conține clase ajutătoare precum DatabaseConnection,DataUtil,

DialogUtil, UtilUI etc., care oferă funcționalități comune necesare în mai multe locuri din aplicație.

Beneficii: Evită duplicarea codului și centralizează logica comună, facilitând astfel mentenanța și dezvoltarea aplicației.

1. **Clasa Main** - reprezinta punctul de intrare în aplicație. Inițializează aplicația și stabilește configurația inițială.

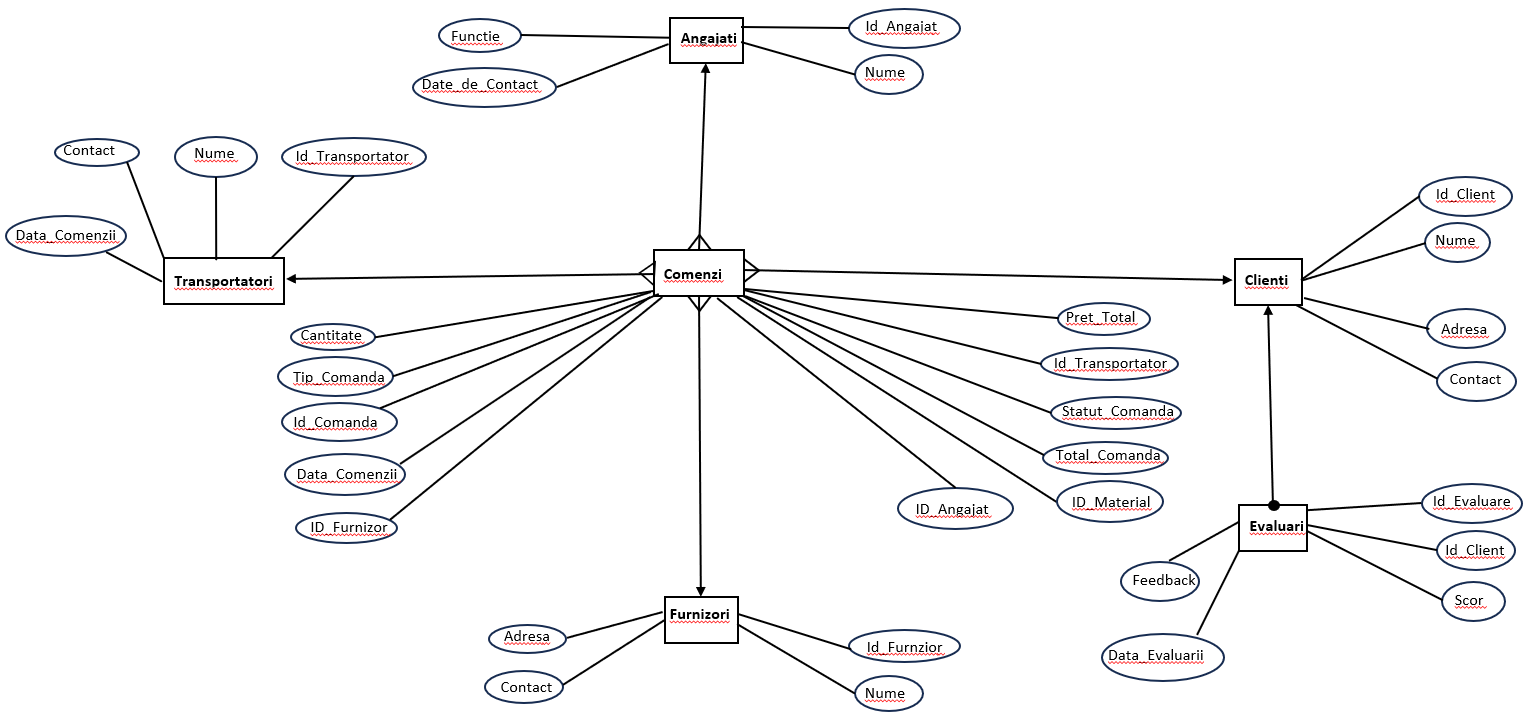
Beneficii: Clarifică locul de start al aplicației, oferind o vedere de ansamblu asupra modului de inițializare a acesteia.

Daca ar fi sa vorbim despre beneficiile globale ale organizării pe pachete și clase acestea aduc o serie de avantaje printre care:

* Claritate și Ușurință în Navigare: codul este mai ușor de navigat și înțeles. Fiecare clasă și pachet are un scop clar definit.
* Reutilizabilitate: clasele și metodele pot fi refolosite în diferite părți ale aplicației, reducând duplicarea codului.
* Mentenanță Ușoară: modificările într-o parte a sistemului au un impact minim asupra altor părți, facilitând astfel mentenanța și actualizările.
* Testare Facilitată: organizarea modulară permite testarea unitară și de integrare mai eficientă a componentelor individuale.
* Scalabilitate și Extensibilitate: Aplicația poate fi scalată și extinsă cu ușurință, adăugând noi pachete sau clase după necesități.

Această structură modulară și organizată pe pachete și clase este un principiu fundamental al designului software de calitate și este crucială pentru dezvoltarea eficientă și sustenabilă a aplicațiilor complexe.

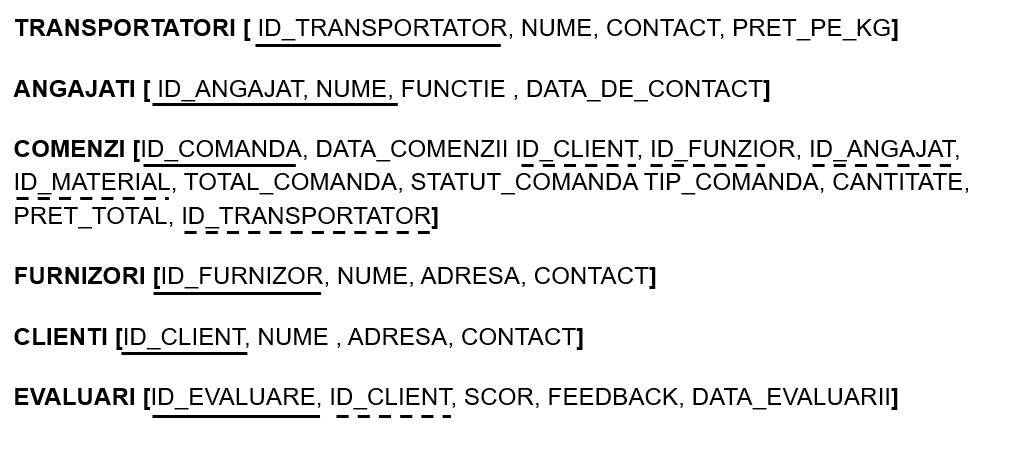
1. **Proiectarea la nivel conceptual (Diagrama Entitate Relatie)**

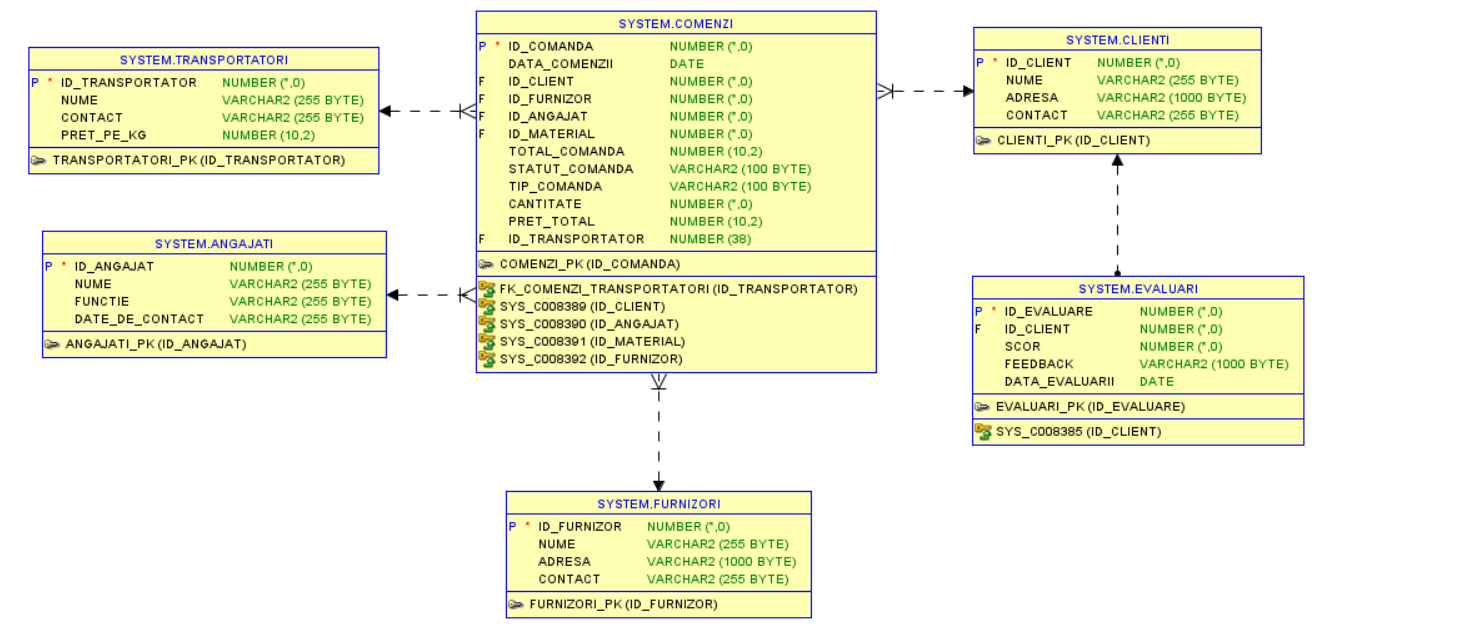


**Fig.4.1.** *Reprezentarea Modelului Conceptual al bazei de date Warehouse*

1. **Proiectarea la nivel logic (schema bazei de date relationale)**

Proiectarea la nivel logic a unei baze de date se referă la structurarea modelelor de tabele, relațiile dintre ele, tipurile de date pentru fiecare coloană și constrângerile aplicate.





**Fig.5.1.** *Reprezentarea modelului relational in SQL Developer*

1. **Proiectarea la nivel fizic**

Proiectarea fizică a bazei de date reprezintă etapa în care am stabilit modul concret

în care datele vor fi stocate în baza de date. Aceasta include definirea structurii tabelelor, a coloanelor, tipurilor de date, a dimensiunilor și a restricțiilor de integritate.

Fiecare tabel reprezintă o entitate din modelul nostru de date și conține informații

specifice, structurate în coloane cu diferite tipuri de date.

Pentru o înțelegere mai bună a structurii și a organizării datelor, fiecare tabel va fi prezentat detaliat cu specificațiile sale. Aceasta va include o descriere a fiecărei coloane, tipul de date, restricțiile impuse (de exemplu, dacă un câmp poate fi sau nu null, dacă este o cheie primară sau străină etc.), și o descriere succintă a rolului acelei coloane în cadrul tabelului.

Încep această secțiune cu prezentarea detaliată a tabelului Angajati:

1. **Tabelul Angajati**

Acest tabel stochează informații esențiale despre angajații companiei. Structura tabelului este descrisă mai jos:

ID\_ANGAJAT:

Tip de date: NUMBER(38,0)

Restricții: Nu poate fi null, Cheie primară

Descriere: Identificator unic pentru fiecare angajat, utilizat pentru a face referire la angajat în alte tabele și operații.

NUME:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Numele complet al angajatului.

FUNCTIE:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Funcția sau rolul angajatului în cadrul companiei.

DATE\_DE\_CONTACT:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Informații de contact ale angajatului, cum ar fi numărul de telefon sau adresa de e-mail.

Sintaxa SQL:

CREATE TABLE ANGAJATI (

ID\_ANGAJAT NUMBER(38,0) NOT NULL,

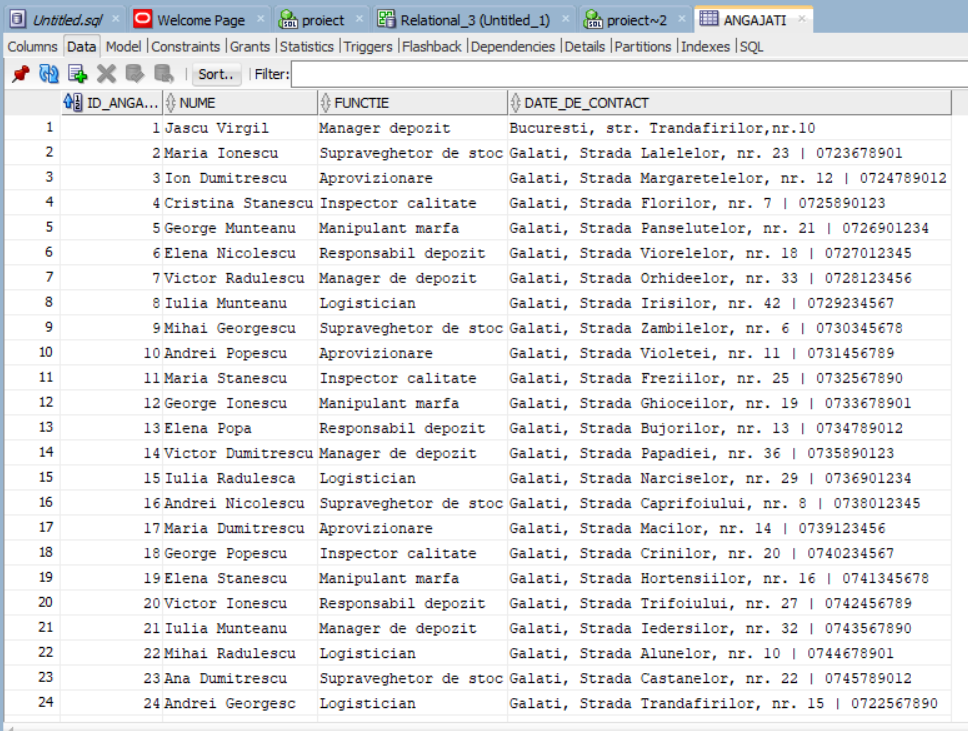
NUME VARCHAR2(255 BYTE),

FUNCTIE VARCHAR2(255 BYTE),

DATA\_DE\_CONTACT VARCHAR2(255 BYTE),

CONSTRAINT PK\_ANGAJATI PRIMARY KEY (ID\_ANGAJAT)

);



**Fig. 6.1.** *Captură de ecran cu tabelul Angajati în SQL*

1. **Tabelul Clienti**

Tabelul Clienti reprezintă o componentă esențială a sistemului, stocând informații despre clienții companiei. Această tabelă este structurată astfel:

ID\_CLIENT:

Tip de date: NUMBER(38,0)

Restricții: Nu poate fi null, Cheie primară

Descriere: Un identificator unic pentru fiecare client. Acest câmp este folosit pentru a asocia comenzi și alte activități specifice clientului.

NUME:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Numele sau denumirea clientului, care poate fi o persoană fizică sau o entitate juridică.

ADRESA:

Tip de date: VARCHAR2(1000 BYTE)

Restricții: Poate fi null

Descriere: Adresa fizică a clientului, utilă pentru livrări, facturare și alte nevoi logistice.

CONTACT:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Informații de contact ale clientului, cum ar fi numărul de telefon sau adresa de e-mail, esențiale pentru comunicare și servicii clienți.

Sintaxa SQL :

CREATE TABLE CLIENTI (

ID\_CLIENT NUMBER(38,0) NOT NULL,

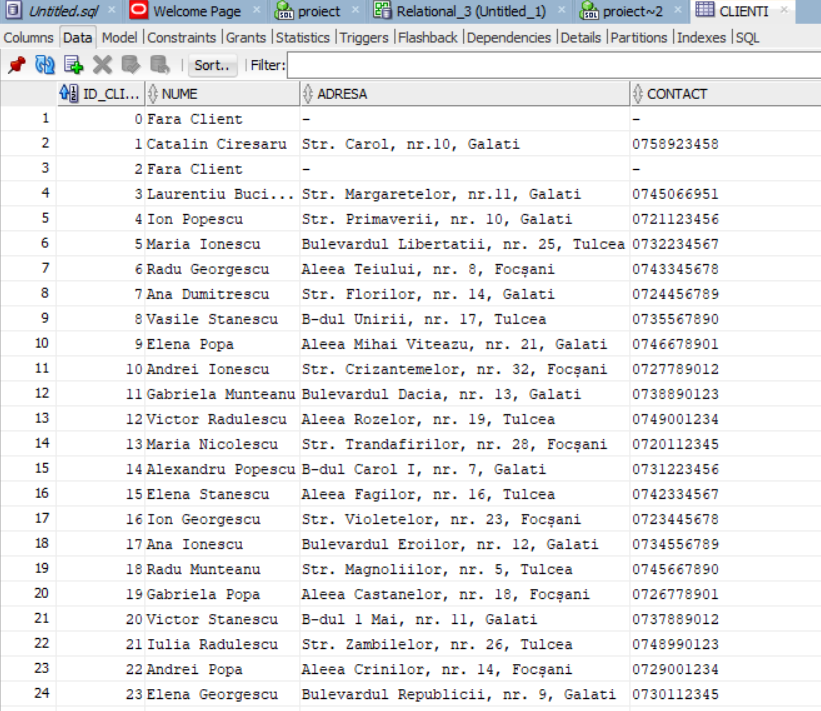
NUME VARCHAR2(255 BYTE),

ADRESA VARCHAR2(1000 BYTE),

CONTACT VARCHAR2(255 BYTE),

CONSTRAINT PK\_CLIENTI PRIMARY KEY (ID\_CLIENT)

);



**Fig.6.2.** *Captură de ecran cu tabelul Clienti în SQL*

1. **Tabelul Comenzi**

Acesta joacă un rol central în sistem, înregistrând informațiile legate de comenzi efectuate de clienți. Structura acestui tabel este următoarea:

ID\_COMANDA:

Tip de date: NUMBER(38,0)

Restricții: Nu poate fi null, Cheie primară

Descriere: Identificator unic al fiecărei comenzi, folosit pentru a urmări și gestiona comenzi specifice.

DATA\_COMENZII:

Tip de date: DATE

Restricții: Poate fi null

Descriere: Data la care comanda a fost plasată.

ID\_CLIENT:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Identificatorul clientului care a plasat comanda.

ID\_FURNIZOR:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Identificatorul furnizorului asociat comenzii.

ID\_ANGAJAT:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Identificatorul angajatului responsabil cu gestionarea comenzii.

ID\_MATERIAL:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Identificatorul materialului sau produsului comandat.

TOTAL\_COMANDA:

Tip de date: NUMBER(10,2)

Restricții: Poate fi null

Descriere: Valoarea totală a comenzii.

STATUT\_COMANDA:

Tip de date: VARCHAR2(100 BYTE)

Restricții: Poate fi null

Descriere: Statutul curent al comenzii (ex: în așteptare, livrată, anulată).

TIP\_COMANDA:

Tip de date: VARCHAR2(100 BYTE)

Restricții: Poate fi null

Descriere: Tipul comenzii (ex: online, directă).

CANTITATE:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Cantitatea produsului sau materialului comandat.

PRET\_TOTAL:

Tip de date: NUMBER(10,2)

Restricții: Poate fi null

Descriere: Prețul total pentru cantitatea comandată.

ID\_TRANSPORTATOR:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Identificatorul transportatorului responsabil pentru livrarea comenzii.

Sintaxa SQL :

CREATE TABLE Comenzi (

ID\_Comanda INT PRIMARY KEY,

Data\_Comenzii DATE,

ID\_Client INT,

ID\_Furnizor INT,

ID\_Angajat INT,

ID\_Material INT, -- Adaugam aceasta linie

Total\_Comanda DECIMAL(10, 2),

Statut\_Comanda VARCHAR2(100),

Tip\_Comanda VARCHAR2(100),

Cantitate INT,

Pret\_Total DECIMAL(10, 2),

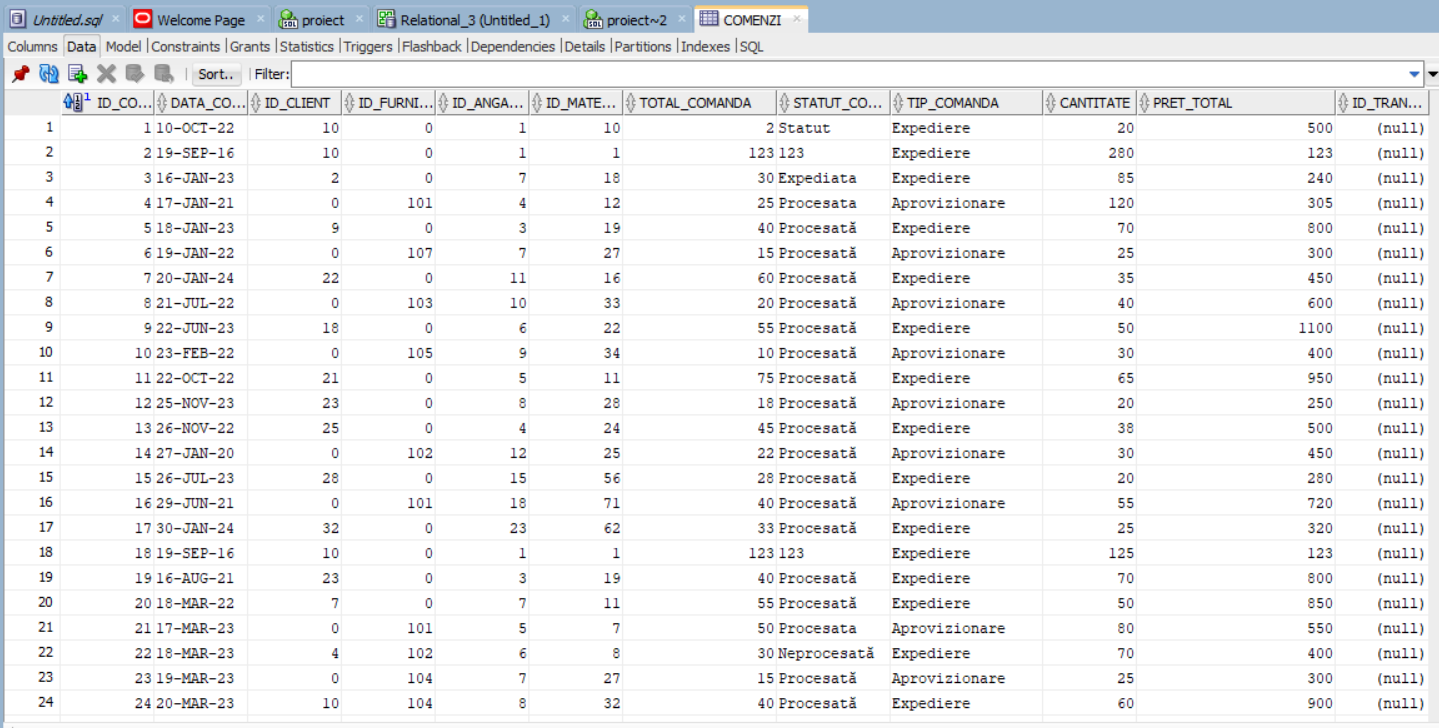
FOREIGN KEY (ID\_Client) REFERENCES Clienti(ID\_Client),

FOREIGN KEY (ID\_Angajat) REFERENCES Angajati(ID\_Angajat),

FOREIGN KEY (ID\_Material) REFERENCES Materiale(ID\_Material),

FOREIGN KEY (ID\_Furnizor) REFERENCES Furnizori(ID\_Furnizor)

);



**Fig.6.3.** *Captură de ecran cu tabelul Comenzi în SQL*

1. **Tabelul Evaluari**

Tabelul Evaluari este esențial pentru colectarea și analiza feedback-ului clienților. Iată structura acestui tabel:

ID\_EVALUARE:

Tip de date: NUMBER(38,0)

Restricții: Nu poate fi null, Cheie primară

Descriere: Identificator unic pentru fiecare evaluare, utilizat pentru a gestiona înregistrările individuale de feedback.

ID\_CLIENT:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Identificatorul clientului care a furnizat feedback-ul.

SCOR:

Tip de date: NUMBER(38,0)

Restricții: Poate fi null

Descriere: Scorul numeric acordat de client, reprezentând gradul de satisfacție.

FEEDBACK:

Tip de date: VARCHAR2(1000 BYTE)

Restricții: Poate fi null

Descriere: Textul feedback-ului oferit de client.

DATA\_EVALUARII:

Tip de date: DATE

Restricții: Poate fi null

Descriere: Data la care a fost înregistrat feedback-ul.

Sintaxa SQL:

CREATE TABLE EVALUARI (

ID\_EVALUARE NUMBER(38,0) NOT NULL,

ID\_CLIENT NUMBER(38,0),

SCOR NUMBER(38,0),

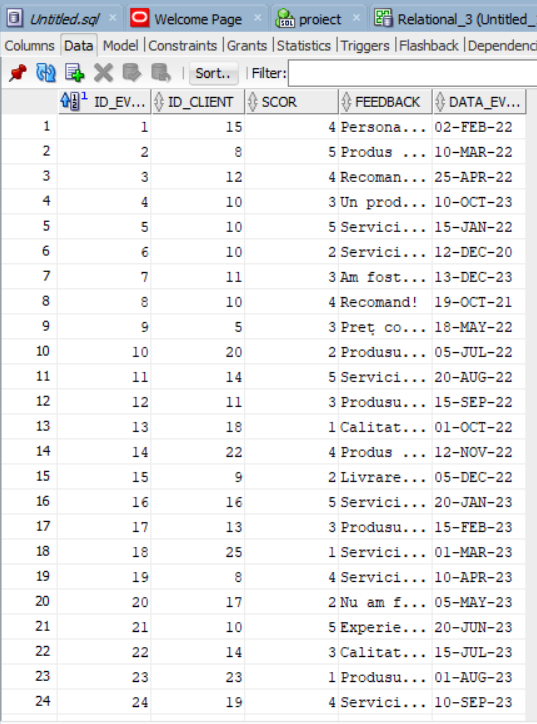
FEEDBACK VARCHAR2(1000 BYTE),

DATA\_EVALUARII DATE,

CONSTRAINT PK\_EVALUARI PRIMARY KEY (ID\_EVALUARE),

CONSTRAINT FK\_EVALUARI\_CLIENTI FOREIGN KEY (ID\_CLIENT) REFERENCES CLIENTI(ID\_CLIENT)

);



**Fig.6.4.** *Captură de ecran cu tabelul Evaluari în SQL*

1. **Tabelul Furnizori**

Tabelul furnizori este crucial pentru gestionarea relațiilor cu entitățile care furnizează bunuri sau servicii. Iată structura acestui tabel:

ID\_FURNIZOR:

Tip de date: NUMBER(38,0)

Restricții: Nu poate fi null, Cheie primară

Descriere: Identificator unic pentru fiecare furnizor, facilitând organizarea și urmărirea acestora.

NUME:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Numele furnizorului, esențial pentru identificarea și comunicarea cu acesta.

ADRESA:

Tip de date: VARCHAR2(1000 BYTE)

Restricții: Poate fi null

Descriere: Adresa completă a furnizorului, utilă pentru trimiterea corespondenței sau a documentelor oficiale.

CONTACT:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Datele de contact ale furnizorului (de exemplu, număr de telefon, e-mail), esențiale pentru comunicarea eficientă.

Sintaxa SQL :

CREATE TABLE FURNIZORI (

ID\_FURNIZOR NUMBER(38,0) NOT NULL,

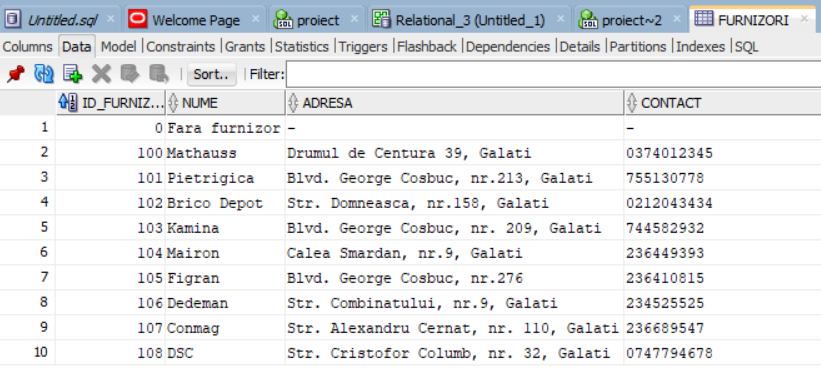
NUME VARCHAR2(255 BYTE),

ADRESA VARCHAR2(1000 BYTE),

CONTACT VARCHAR2(255 BYTE),

CONSTRAINT PK\_FURNIZORI PRIMARY KEY (ID\_FURNIZOR)

);



**Fig.6.5.** *Captură de ecran cu tabelul Furnizori în SQL*

1. **Tabelul Transportatori**

Acesta din urma joacă un rol esențial în managementul logistic al afacerii, oferind informații despre companiile sau persoanele care asigură transportul bunurilor. Iată detaliile structurii acestui tabel:

ID\_TRANSPORTATOR:

Tip de date: NUMBER(38,0)

Restricții: Nu poate fi null, Cheie primară

Descriere: Identificator unic pentru fiecare transportator, crucial pentru urmărirea și organizarea serviciilor de transport.

NUME:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Numele transportatorului sau al companiei de transport, important pentru identificarea și contactarea acestora.

CONTACT:

Tip de date: VARCHAR2(255 BYTE)

Restricții: Poate fi null

Descriere: Detaliile de contact ale transportatorului, cum ar fi numărul de telefon sau adresa de e-mail, vitale pentru coordonarea și comunicarea eficientă.

PRET\_PE\_KG:

Tip de date: NUMBER(10,2)

Restricții: Poate fi null

Descriere: Tariful perceput de transportator pentru fiecare kilogram transportat, esențial pentru calcularea costurilor de transport și pentru planificarea bugetară.

Sintaxa SQL :

CREATE TABLE TRANSPORTATORI (

ID\_TRANSPORTATOR NUMBER(38,0) NOT NULL,

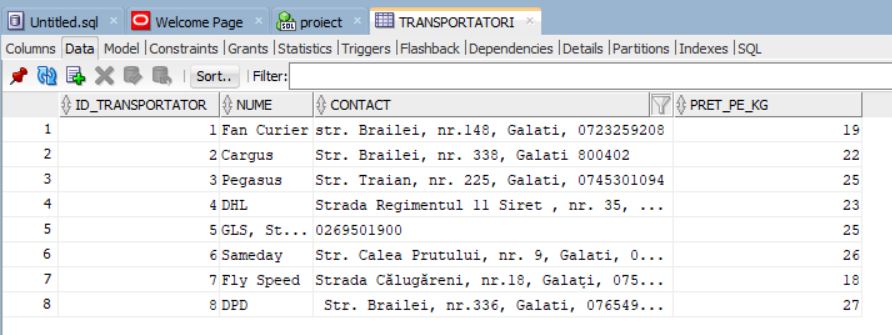
NUME VARCHAR2(255 BYTE),

CONTACT VARCHAR2(255 BYTE),

PRET\_PE\_KG NUMBER(10,2),

CONSTRAINT PK\_TRANSPORTATORI PRIMARY KEY (ID\_TRANSPORTATOR)

);



**Fig.6.6.** *Captură de ecran cu tabelul Transportatori în SQL*

1. **Limbaje de Programare**

Pentru edificarea și consolidarea structurii aplicației warehouse, am apelat la robustețea și versatilitatea limbajului Java, o adevărată coloană vertebrală în lumea dezvoltării software. Java, cu paradigma sa orientată pe obiecte și cu mecanismele sale bine înrădăcinate de gestionare a memoriei și de asigurare a securității, oferă un mediu de dezvoltare stabil și eficient, garantând performanță și scalabilitate în tratamentul și manipularea datelor complexe.

În tandem cu Java, limbajul SQL, acronim pentru Structured Query Language, servește drept instrument esențial în manipularea și interogarea minuțioasă a bazelor de date relaționale. Prin sintaxa sa clară și logică puternică, SQL facilitează extragerea, actualizarea, inserția și ștergerea datelor, permițându-ne să sculptăm în detaliu setul de date și să extragem informații vitale pentru procesele decizionale.

Împreună, Java și SQL constituie o sinergie tehnologică, împletind logică de programare avansată cu operațiuni de bază de date, pentru a crea o aplicație complexă, capabilă să servească cerințele intricate ale mediului de afaceri și să răspundă cu agilitate la dinamica nevoilor utilizatorilor.

Tehnologii și Librării

JavaFX: O bibliotecă pentru dezvoltarea de aplicații desktop în Java, folosită pentru a crea interfața grafică a aplicației tale. Permite construirea unei UI (User Interface - Interfață Utilizator) moderne și responsive.

JavaFX:

FXCollections: O clasă utilitară din JavaFX care oferă metode pentru crearea și gestionarea colecțiilor JavaFX, cum ar fi listele observabile.

ObservableList: O interfață din JavaFX care reprezintă o colecție care permite monitorizarea schimbărilor în lista de elemente.

Insets: O clasă din JavaFX folosită pentru a specifica mărimile marginilor pentru layout-urile JavaFX.

javafx.scene.chart.\*: Pachetul care conține clase și utilitare pentru crearea și gestionarea graficelor în JavaFX, cum ar fi graficele de linii, bare, cerc sau tort.

javafx.scene.control.\*: Pachetul care conține clase pentru diferite controale de interfață grafică, cum ar fi butoane, casete de text, meniuri, liste și multe altele.

javafx.scene.layout.HBox: O clasă din JavaFX folosită pentru a aranja nodurile copil într-o linie orizontală.

javafx.scene.layout.VBox: O clasă din JavaFX folosită pentru a aranja nodurile copil într-o coloană verticală.

Java Standard Library:

java.io.File: O clasă din pachetul java.io care reprezintă calea către un fișier sau un director din sistemul de fișiere.

java.io.PrintWriter: O clasă din java.io care oferă metode pentru a scrie caractere formate pe un flux de ieșire de text.

java.sql.SQLException: O clasă din java.sql care reprezintă o excepție care este generată atunci când apare o eroare în timpul accesării bazei de date sau în timpul comunicării cu aceasta.

java.util.\*: Pachetul larg care conține diferite clase și utilitare pentru lucrul cu colecții, date și alte structuri de bază în Java.

JDBC (Java Database Connectivity): O bibliotecă Java pentru conectarea la baze de date. Este folosită pentru a executa interogări SQL și a gestiona rezultatele acestora din Java.

java.util.\*: Pachetul standard Java care conține clase și interfețe pentru lucrul cu colecții, date și alte structuri de bază.

oracle.net.ano.AuthenticationService: Import pentru o clasă din Oracle care furnizează servicii de autentificare pentru rețea.

Oracle SQL Developer: Un IDE (Integrated Development Environment - Mediu de Dezvoltare Integrat) pentru lucrul cu baze de date SQL, utilizat pentru gestionarea bazei de date și executarea scripturilor SQL.

Alternative și Considerații

În loc de JavaFX, se putea folosi Swing pentru interfața grafică, dar JavaFX oferă un design mai modern și posibilități avansate de personalizare.

Pentru gestionarea conexiunilor la baza de date, se putea folosi un ORM (Object-Relational Mapping) precum Hibernate, care simplifică lucrul cu bazele de date prin maparea automată a obiectelor Java la tabele SQL.

Concluzie

In concluzie pot afirma faptul ca proiectul meu folosește o arhitectură bine structurată, bazată pe principii de design clar definite. Alegerea JavaFX și JDBC este potrivită pentru tipul aplicației, iar structurarea pe pachete oferă claritate și modularitate. Utilizarea validării datelor la nivelul interfeței și al back-endului asigură integritatea și corectitudinea datelor procesate.

Implementare

În cazul proiectul meu, validarea este implementată astfel:

JavaFX pentru UI: Folosirea de TextField cu validatori pentru a restricționa tipul și formatul datelor introduse (de exemplu, numai cifre pentru un ID).

JDBC și DAO: Înainte de a trimite o interogare la baza de date, se pot verifica datele pentru a asigura că respectă regulile de afaceri și formatul așteptat (de exemplu, verificarea că un ID nu este negativ).

Proceduri stocate SQL: În baza de date, se pot folosi proceduri stocate care includ logica de validare pentru a asigura integritatea datelor.

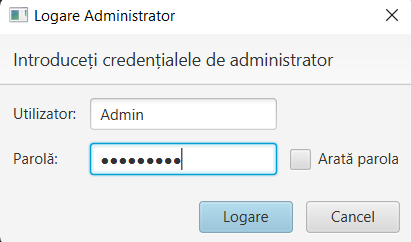
Alternative

ORM (Object-Relational Mapping) Tools: Unelte precum Hibernate pot automatiza unele validări și pot oferi un mod mai simplu de a gestiona validarea la nivel de baza de date.

Librării de validare: Există biblioteci dedicate validării datelor, cum ar fi Apache Commons Validator pentru Java, care pot fi folosite pentru a simplifica procesul de validare.

1. **Descrierea aplicatiei (manualul de utilizare)** - Descrierea tuturor functionalitatilor pe fiecare rol (folosind capturi de ecran)

Aplicatia in rolul de manager si utilizator functioneaza incepand cu logarea la programul de gestionare al stocurilor unui depozit.



**Fig.8.1.** *Reprezentare fereastra Login In pentru Admin*

Dupa logarea Adminului aplicatia imi permite accesul la un Meniul principal in care apar sectiunile sistemului de gestionare si permite alegerea fiecareia dintre acestea.



**Fig.8.2.** *Reprezentarea Meniului Aplicatiei Warehouse*

Odata cu accesarea meniului si alegerea uneia dintre sectiuni, in cadrul acestora sunt premise mai multe operatii printer care:

* Vizualizarea datelor existente;
* Introducerea de noi inregistrari;
* Posibilitatea de a edita o inregistrare efectuata deja;
* Stergerea unei inregistrari nadorite;
* Salvarea inregistrarilor si a modificarilor effectuate.

Pentru a extinde functionalitatea aplicatiei si pentru a putea manipula datele inregistrate in cadrul ei aceasta permite exportarea datelor dintr-o anumite sectiune de interes, exportul se poate face pe o selctie multipla doar a randurilor pe care le consideram reprezentative sau se pot exporta toate inregistrarile din categoria care ne intereseaza.



**Fig.8.3.** *Reprezentarea meniului angajati*



**Fig.8.4.** *Reprezentarea butoanelor de control si butonului de Export*

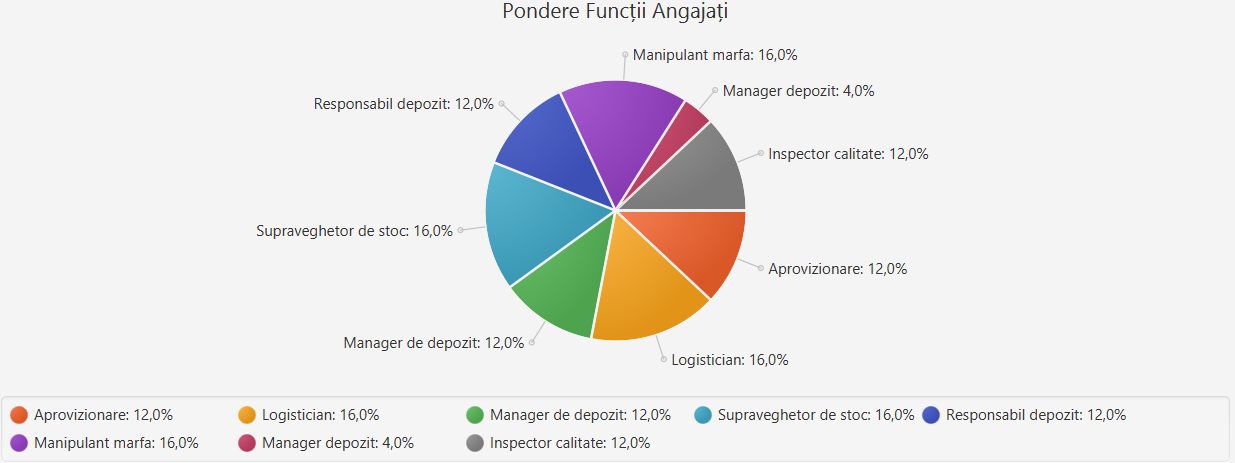
Pe langa operatiile mentionate anterior trebuie mentionat ca fiecare camp al tabelei permite sortare crescatoare sau descrescatoare ale inregistrarilor.

Pentru a facilita operativitatea utilizatorului aplicatiei si pentru a obtine informatiile in timp util aceasta iti permite sa realizezi o filtrare in campurile principale dupa criteriile dorite.

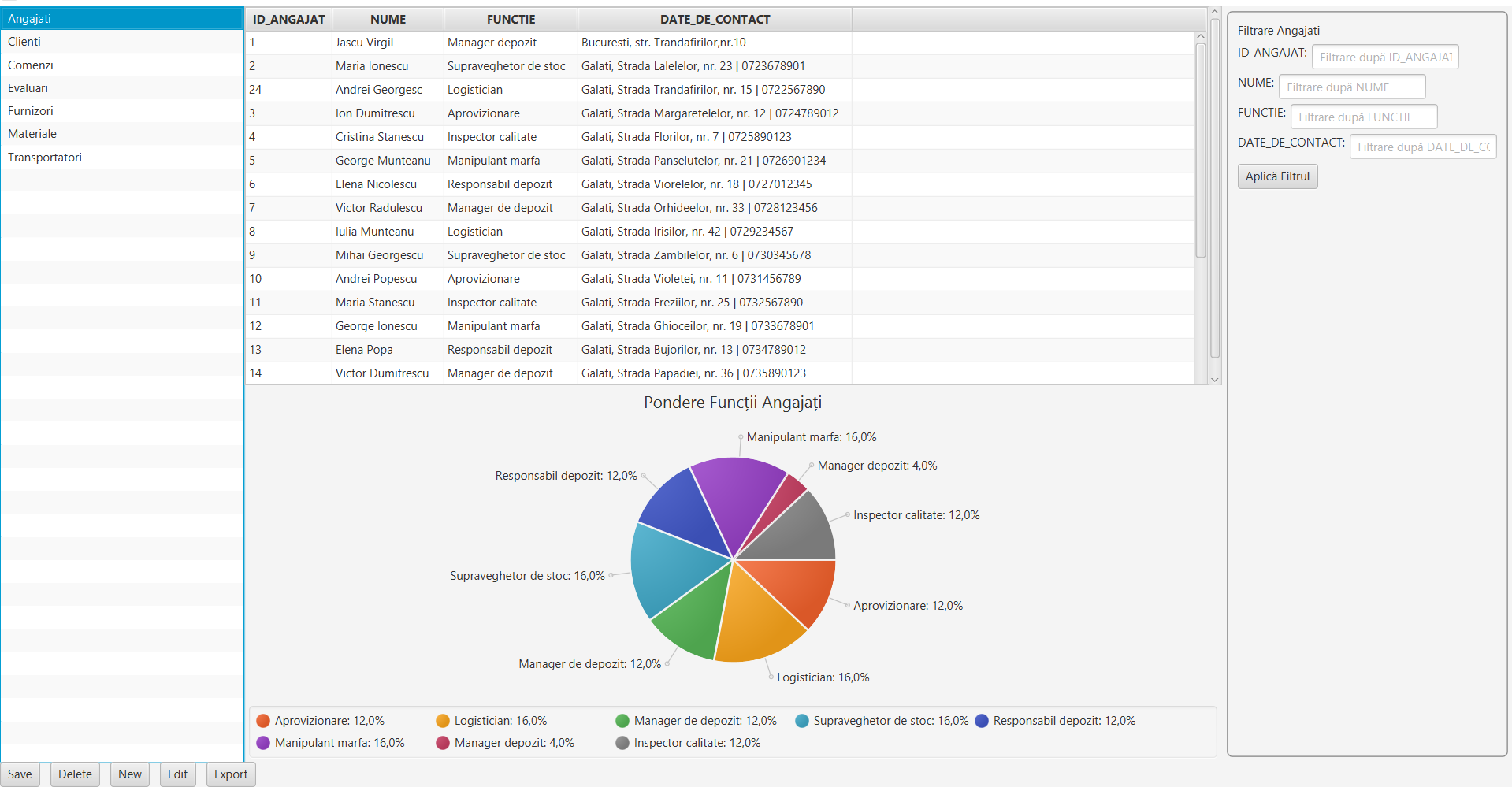


**Fig.8.5.** *Reprezentare sectiune filtrare Angajati*

De asemenea pentru a avea o imagine de ansamblu a categoriilor predominante de Angajati se poate observa o reprezentare a ponderii functiilor care ocupa anumite meserii in cadrul depozitului Warehouse.



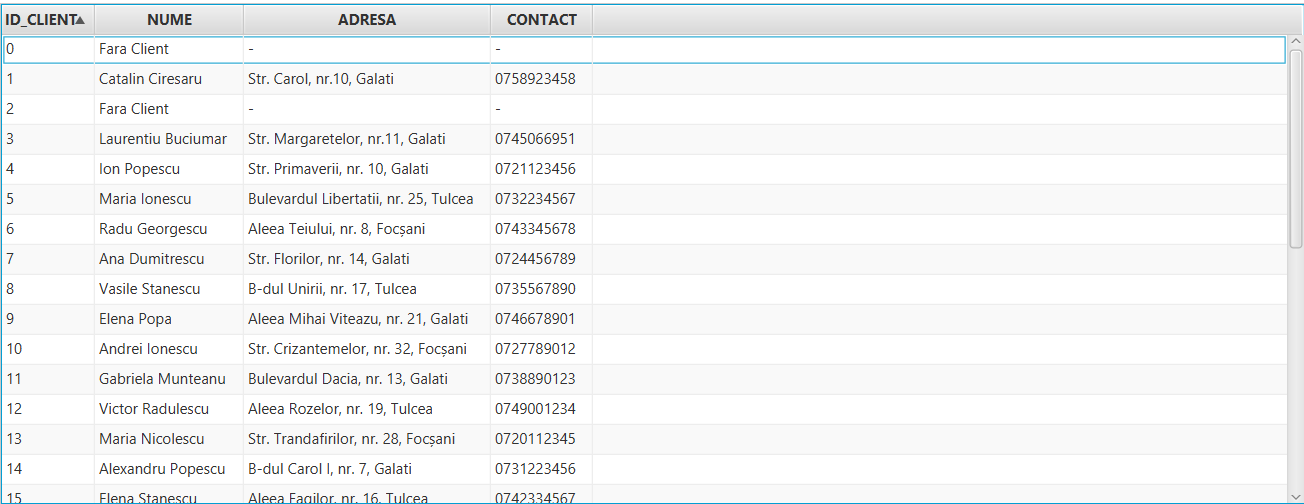
**Fig.8.6.** *Reprezentare ponderii functiilor din cadrul depozitului de materiale*



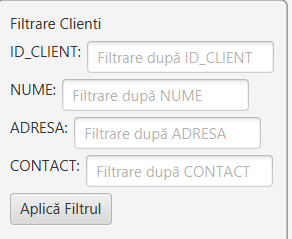
**Fig.8.7.** *Reprezentare meniului complet pentru sectiunea Angajati*



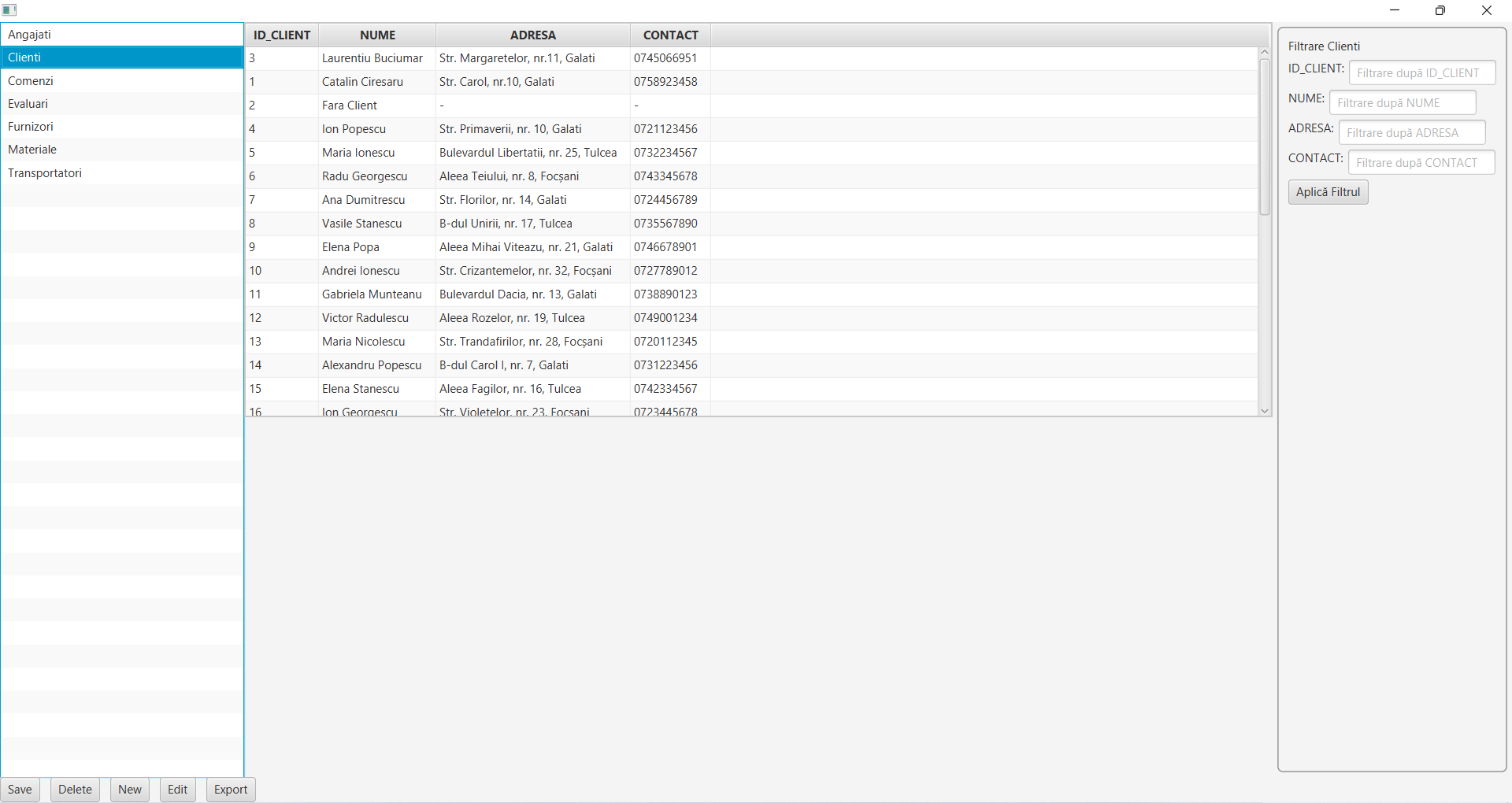
**Fig.8.8.** *Reprezentarea selectarii sectiunii Clienti*



**Fig.8.9.** *Reprezentare inregistrarilor pentru din sectiunea Clienti*



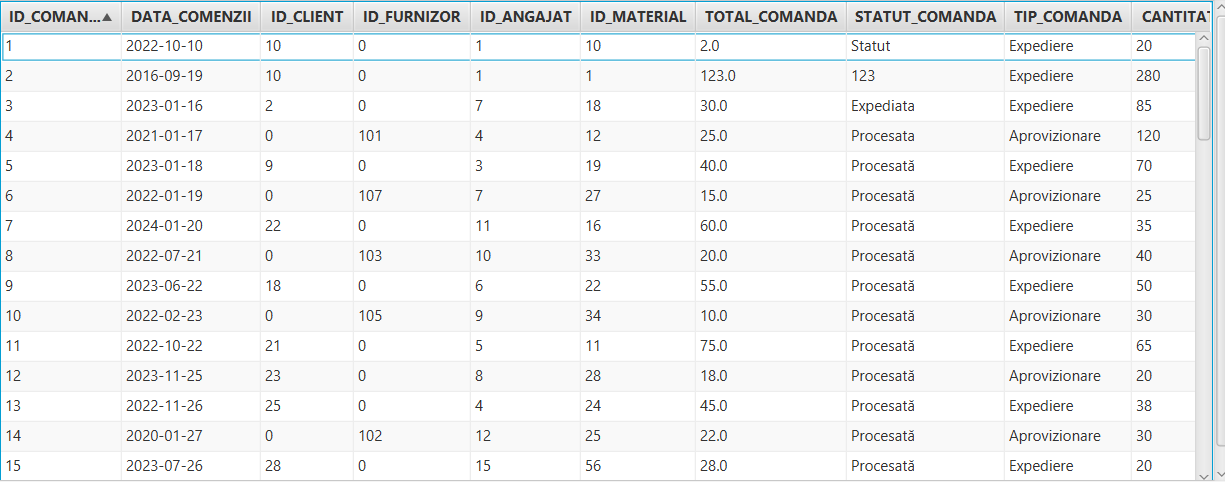
**Fig.8.10.** *Reprezentare sectiune filtrare Clienti*



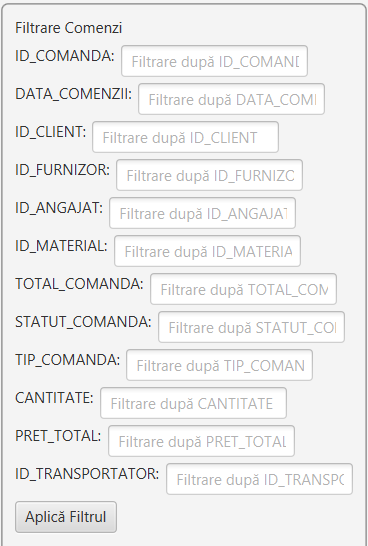
**Fig.8.11.** *Reprezentare meniului pentru sectiunea Clienti*



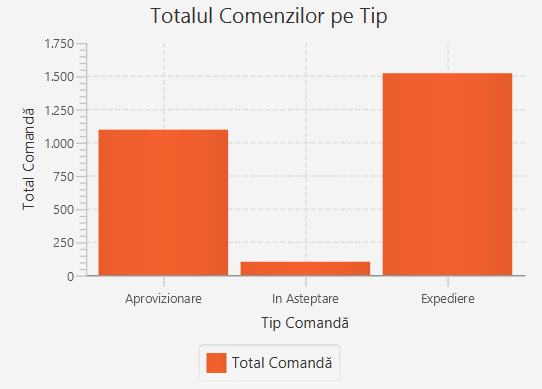
**Fig.8.12.** *Reprezentarea selectarii sectiunii Comenzi*



**Fig.8.13.** *Reprezentare inregistrarilor pentru din sectiunea Clienti*

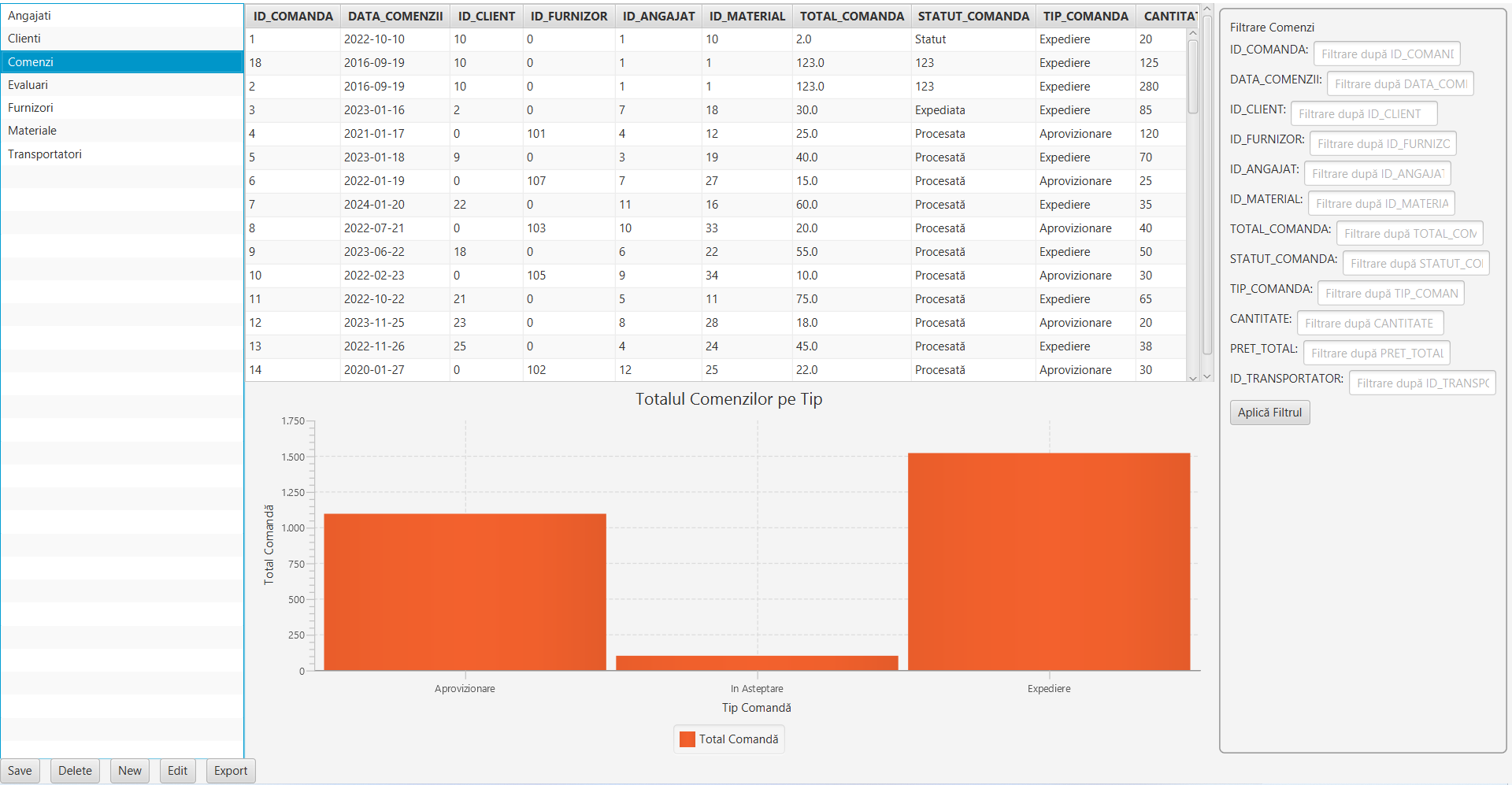


**Fig.8.14.** *Reprezentare sectiune filtrare Comenzi*



**Fig.8.15.** *Reprezentare grafica a comenzilor pentru*

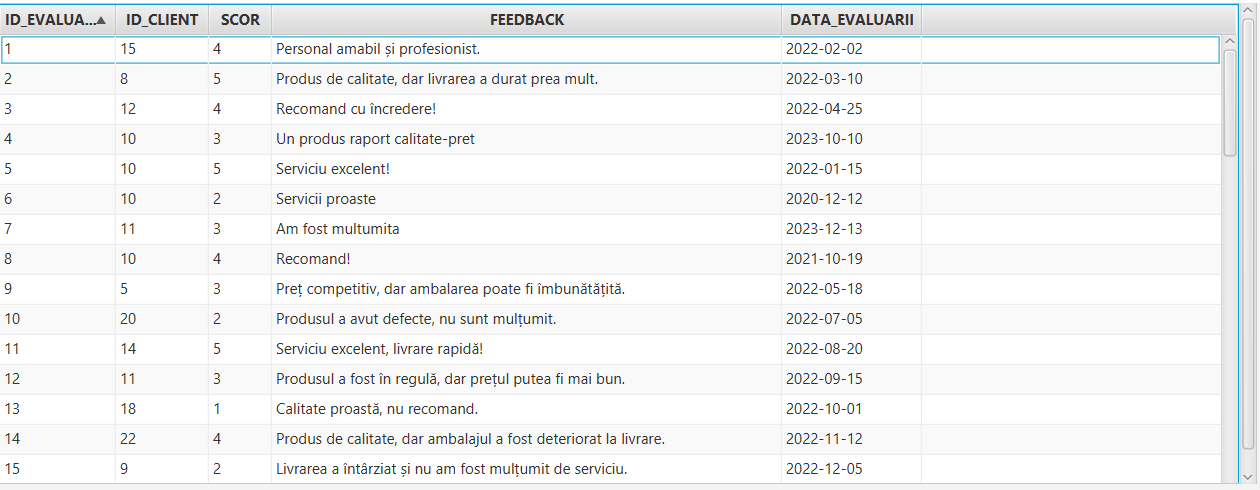
*Aprovizionare si Expediere*



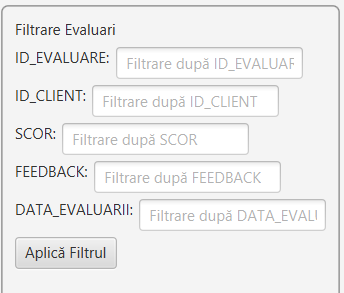
**Fig.8.16.** *Reprezentare meniuliu complet pentru Sectiunea Comenzi*



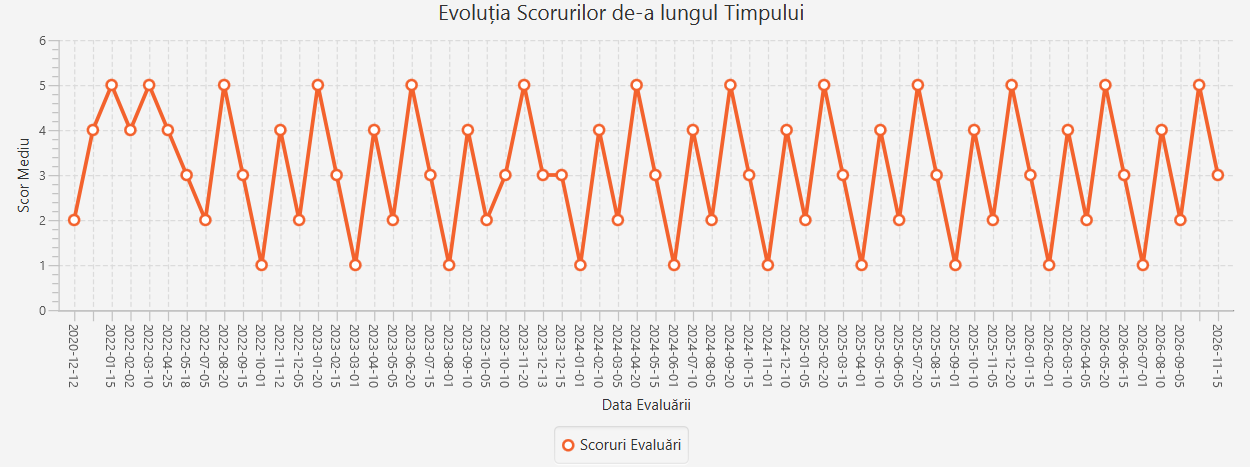
**Fig.8.17.** *Reprezentarea selectarii sectiunii Comenzi*



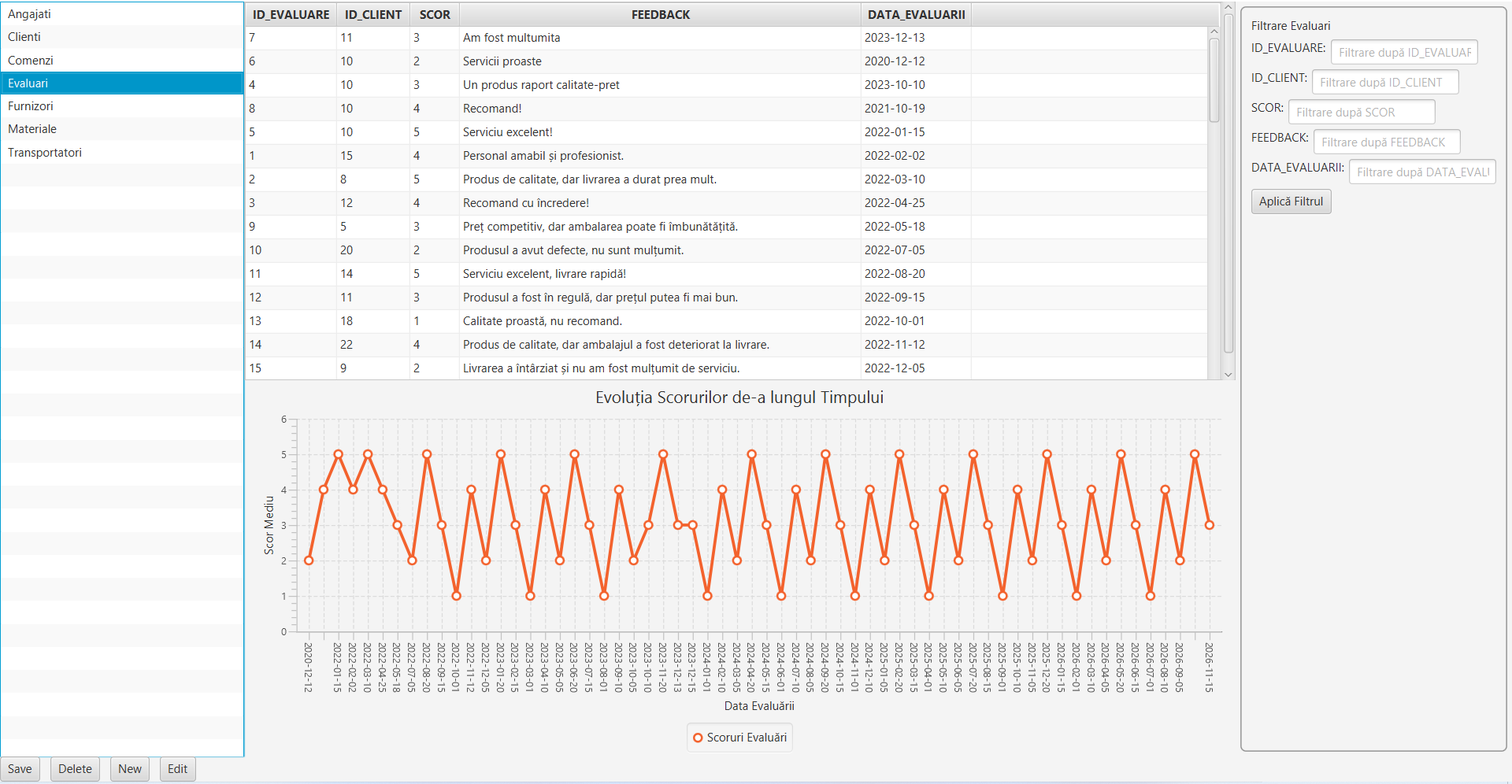
**Fig.8.18.** *Reprezentare inregistrarilor pentru din sectiunea Evaluari*



**Fig.8.19.** *Reprezentare sectiune filtrare Evaluari*



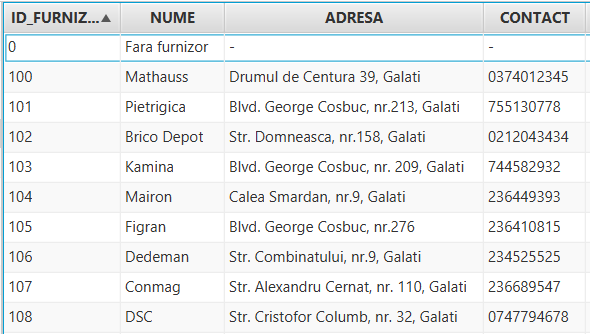
**Fig.8.20.** *Reprezentare a evolutiei Scorurilor in timp*



**Fig.8.21.** *Reprezentare meniului complet pentru Evaluari*



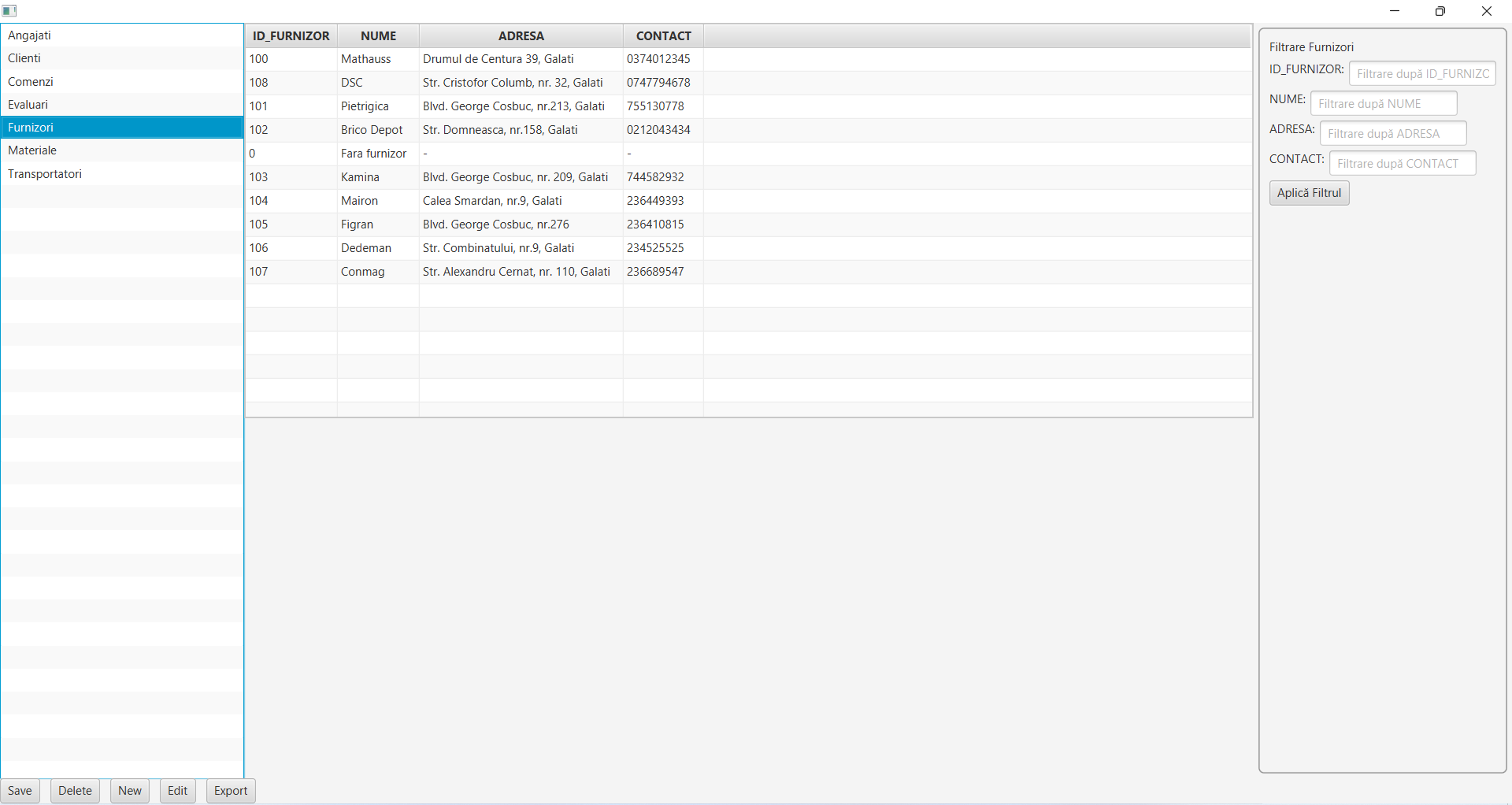
**Fig.8.22.** *Reprezentare sectiunii Furnizori*



**Fig.8.23.** *Reprezentare inregistrarilor pentru sectiunea Furnizori*



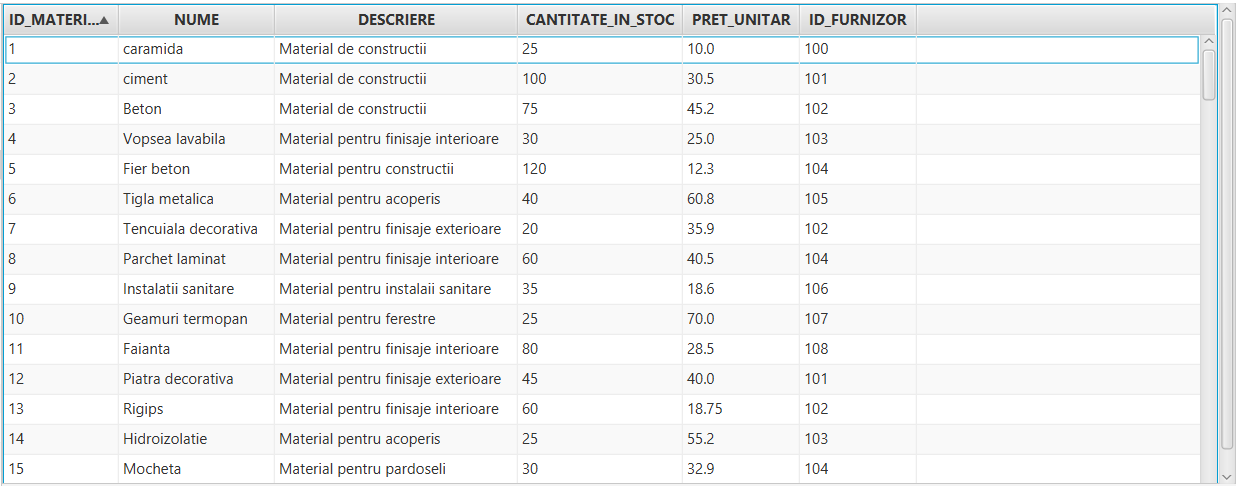
**Fig.8.24.** *Reprezentarea filtrelor pentru Furnizor*



**Fig.8.25.** *Reprezentarea meniului complet pentru Furnizori*



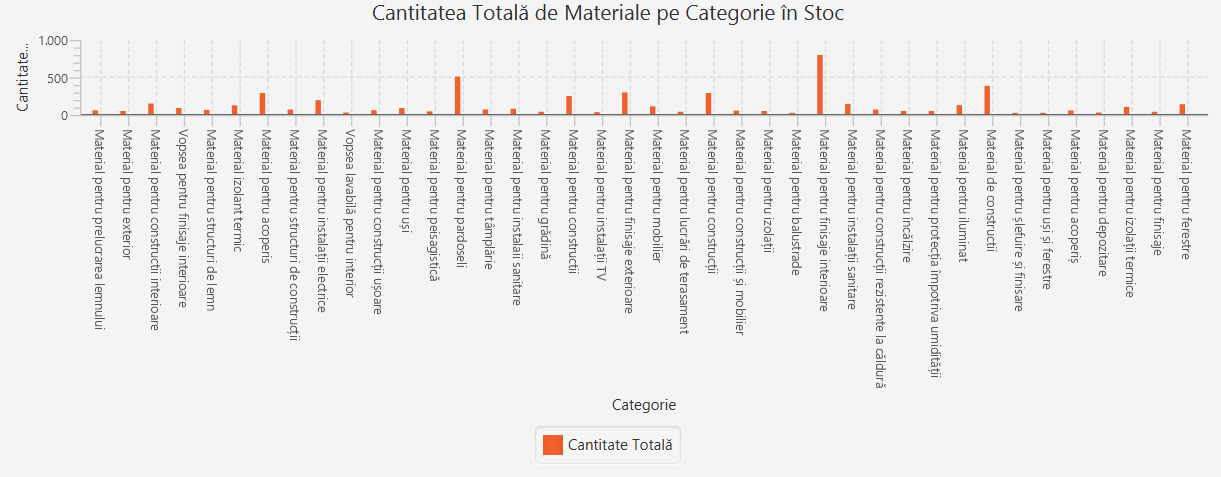
**Fig.8.26.** *Reprezentarea sectiunii Materiale*



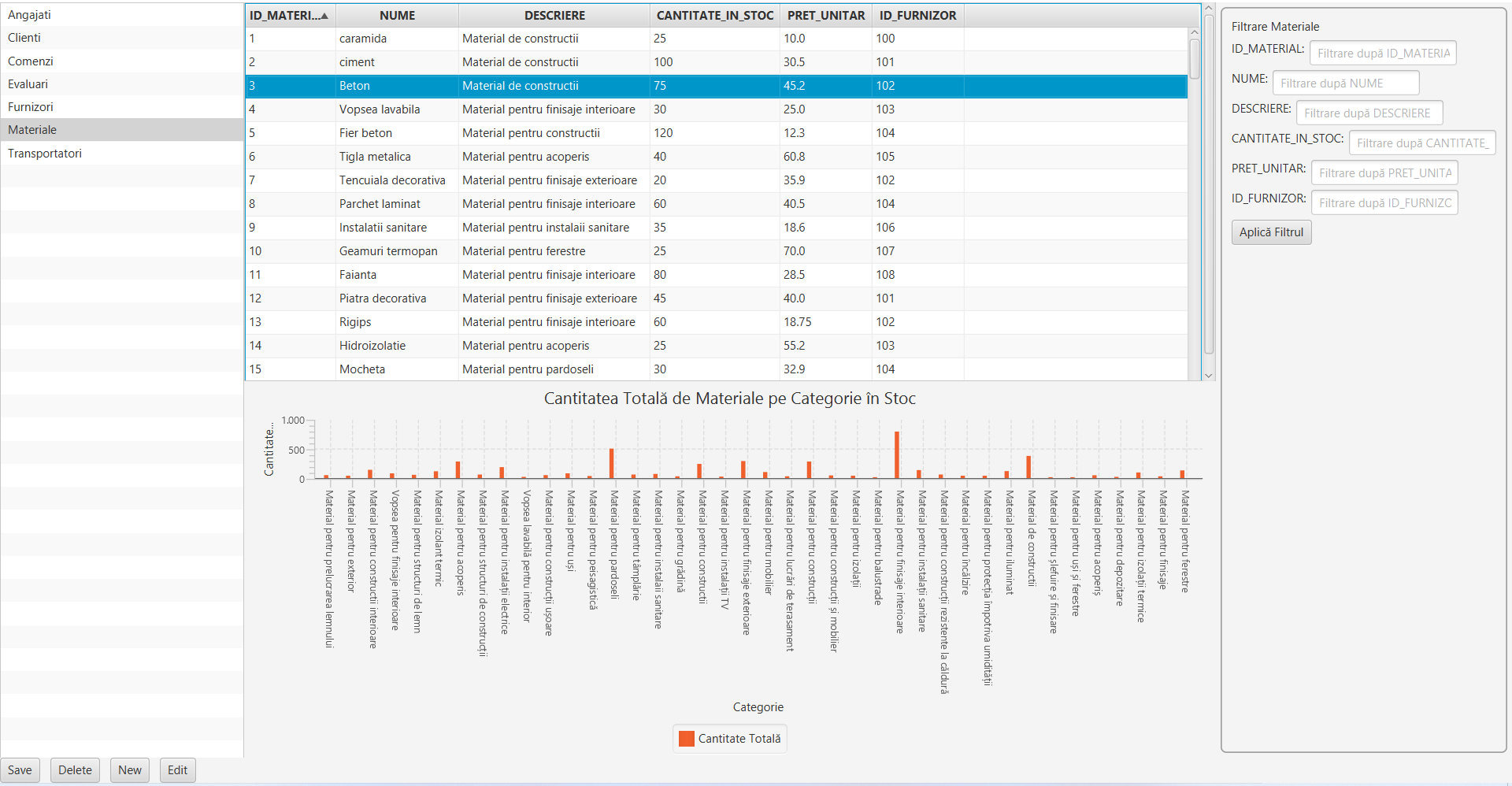
**Fig.8.27.** *Reprezentarea sectiunii de Materiale*



**Fig.8.28.** *Reprezentarea Filtrelor pentru sectiunea Materiale*



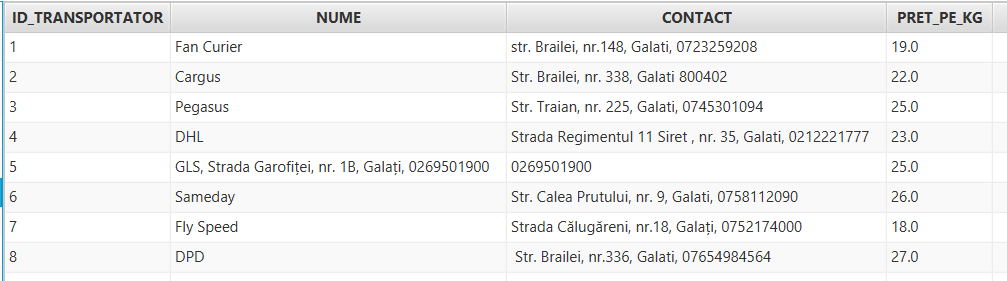
**Fig.8.29.** *Reprezentarea graficului pentru cantitatea si tipul de Materiale*



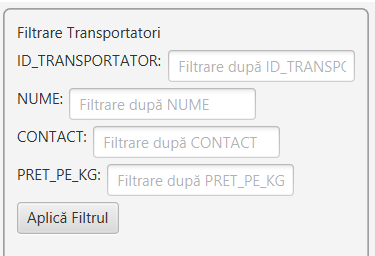
**Fig.8.30.** *Reprezentarea meniului complet pentru sectiunea Materiale*

**

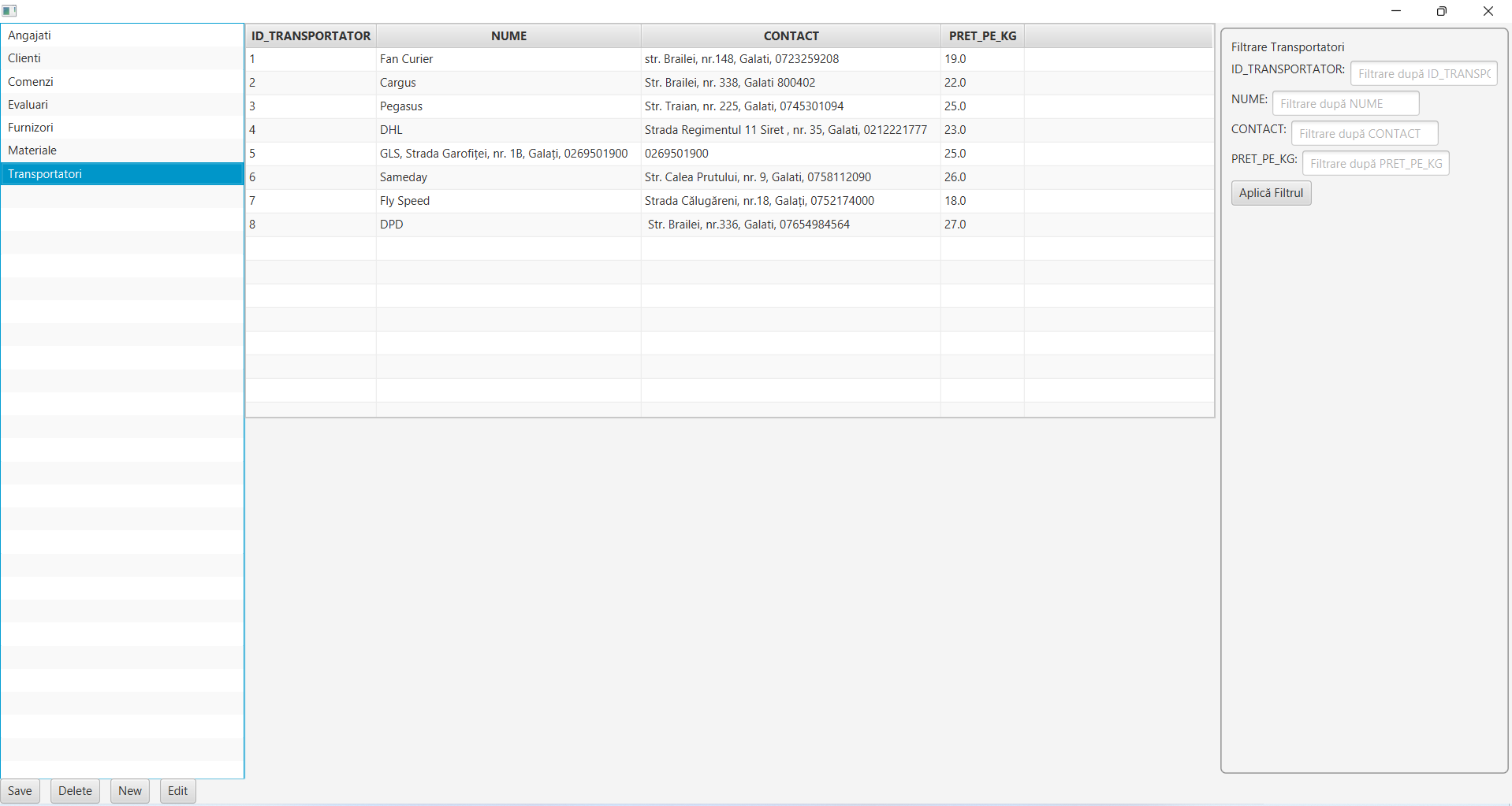
**Fig.8.31.** *Reprezentarea sectiunii Transportatori*

**

**Fig.8.32.** *Reprezentarea inregistrarilor pentru sectiunea Transportatori*



**Fig.8.33.** *Reprezentarea filtrelor pentru Transportatori*



**Fig.8.34.** *Reprezentarea meniului complet pentru Transportatori*