UNIVERSITATEA TEHNICĂ “GH ASACHI” IAŞI

FACULTATEA AUTOMATICĂ ŞI CALCULATOARE

SPECIALIZAREA CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

DISCIPLINA: BAZE DE DATE – PROIECT

GESTIUNEA UNUI TEATRU CU RECUZITĂ

Coordonator,

Prof. Mironeanu Cătălin

Student,

Gorie Ioana

Grupa 1307A

Iași, 2022

Titlul proiectului: Gestiunea unui teatru cu recuzită

Analiza, proiectarea și implementarea unei baze de date și a aplicației aferente care să modeleze activitatea gestionării unui teatru cu recuzită.

Descrierea cerinței și a modului de organizare al proiectului

Cerința: Dorim o bază de date care să se ocupe cu gestiunea sălilor, pieselor, recuzitei și a distribuției de actori.

Ne dorim ca personalul care se ocupă de pregătirea decorului pentru o piesă care va urma să fie jucată, să știe exact în ce zi, la ce oră și în ce sală trebuie să pregătească scena. De asemenea, el trebuie să cunoască și din ce tematică face parte decorul pe care trebuie să-l pregătească, pentru a așeza pe scenă toate obiectele de recuzită de care este nevoie pentru a putea recrea decorul respectiv.

Este necesar să cunoaștem date despre actorii noștri precum mărimea la haine pentru a le oferi costume din recuzită care să li se potrivească. Fiecare actor deține o cabină proprie. Contabilii noștri au nevoie să cunoacă valoarea salariului fiecăruia pentru a le asigura lunar plata aferentă. În acest scop ne dorim să existe o fișă pentru fiecare actor cu toate datele despre el , pentru ca personalul auxiliar să obțină ușor informațiile de care are nevoie pentru a-și îndeplini atribuțiile pe care le are față de fiecare actor în parte. Totodată, este necesar ca fiecare actor să semneze un contract pentru fiecare piesă în care joacă.

Clădirea în care ne desfășurăm activitatea dispune de spațiu pentru cabinele de probă la etajele 1,2 și 3, iar sălile în care se țin spectacolele se află de la parter până la etajul 4.

Programul teatrului este de la 8:00 la 22:00 de luni până duminică.

Modului de organizare al proiectului:

Entități

În modelarea bazei de date pentru gestiunea teatrului de recuzită s-a urmărit crearea a 9 entități după generearea modelului relațional .

Recuzita : se ocupă de înregistrarea obiectelor de recuzită de care dispune teatrul nostru. Se reține materialul din care este fabricat obiectul respectiv.

Decor : prevede spațiile/locurile diverse în care poate fi deghizată scena.

Decor-Recuzită : asociază fiecărui tip de decor toate obiectele de recuzită necesare pentru a aranja scena astfel încât să redea un loc anume. Se reține

cantitatea de obiecte de recuzită necesare. Spre exemplu pentru recreerea unui restaurant nu este suficient un singur scaun și o singură masă pentru a sugera un astfel de loc, ci este nevoie de cel puțin 8 scaune și 2 mese.

Cabina : se ocupă de înregistrarea spațiilor din teatru de tip cabină, reținând etajul fiecărei cabine. Clădirea dispune de cabine pentru actori doar pe etajele 1,2 și 3.

Sala : înregistrează sălile de care dispune teatrul, indicând etajul la care se află sala respectică și capacitatea ei.

Actor : prevede toți angajații teatrului cu postul de actor, pentru care e nevoie să se rețină data angajării pentru un eventual calcul al vechimii lui și o majorare de salariu în cazul atingerii anumitor praguri (salariu care este și el reținut tot într-un câmp al entității) , telefonul la care poate fi contactat dar și id-ul cabinei care îi aparține. Opțional actorii pot oferi și detalii despre email-ul lor pentru a putea primi pe email programarea reprezentatiilor unei piese în care joacă sau despre mărimea lor pentru a se asigura ca vor primi din prima costume pe măsura lor.

Piesă : înregistrează toate piesele din repertoriul teatrului, reținând numele piesei, autorul, genul , durata și decorul necesar punerii ei în scenă. În cazul unor piese se poate specifica și vârsta minimă pe care trebuie să o aibă spectatorii pentru a avea voie să vizioneze piesa, dar aceasta nu este obligatoriu întrucât la acest teatru se joacă și piese destinate copiilor.

Contract : în momentul în care un actor ia audiția pentru rolul într-o anumită piesă, acesta va semna un contract prin care declară că își ia obligația să participe la reprezentatiile ei.

Reprezentație : reține planificarea reprezentației unei piese, prin evidența zilei, orei de început, orei de sfârșit , a sălii în care se dorește să fie jucată și a numelui piesei.

Relații

Între entitățile regăsite în modelul logic , 7 la număr, fără entitatea Decor-Recuzită și Piesă-Actor, adică Contract, (care au apărut abia la generarea modelului relațional) despre care vom vorbi mai jos , s-au stabilit 6 relații (4 relații one-to-many, o relație many-to-many și o relație one to one).

Relația Decor-Recuzită : relație many-to-many deoarece un obiect de recuzită poate aparține mai multor decoruri și în același timp un decor este compus din mai multe piese de recuzită. Această relație many-to-many a fost rezolvată la generarea modelului relațional prin apariția unei entități suplimentare numită

Decor_Recuzită, în urma căreia este scindată relația dintre cele două tabele inițiale și apar două conexiuni noi de tip one-to-many de la cele două entități Decor și Recuzită către noua entitate.

Relația Piesă-Decor : relație one-to-many ținând cont de faptul că același decor poate fi folosit la mai multe piese, însă la reprezentația unei piese scena poate fi aranjată într-un singur fel (un singur decor).

Relația Actor-Cabină: singura relație de tip one-to-one justificată de faptul că fiecare actor are propria lui cabină și că nu este posibil ca doi sau mai mulți actori să împartă aceeași cabină.

Relația Piesă-Actor: la bază o relație many-to-many dar rezolvată încă din modelul logic prin apariția unei tabele intermediare numită Contract , ce implică ștergerea legăturii inițiale și formarea a două relații noi de tip one-to-many :

Piesă-Contract și **Actor-Contract.** Aceste două relații se ocupă cu asocierea unui actor cu o anumită piesă prin intermediul unui contract. Un actor poate avea mai multe contracte precum și o piesă poate fi inclusă în contractele a mai mulți actori, însă același contract nu poate fi încheiat între mai mulți actori cu o piesă, ci e nevoie de un contract pentru fiecare, cum nici mai multe piese nu pot fi trecute pe contractul aceluiași actor.

Relația Piesă-Sală: gândită inițial ca o relație de tip many-to-many a fost și ea rezolvată însă abia la generarea modelului relațional al bazei de date prin inserarea unei entități numite Reprezentație, care ca și la relația discutată inițial (Decor-Recuzită) produce apariția a două noi relații de tip one-to-many :

Piesă-Reprezentație și **Sală-Reprezentație.** Prin intermediul noii tabele administratorul bazei de date poate ține evidența tuturor reprezentațiilor, aceasta stocând detalii despre data și ora când va avea loc reprezentația, dar și numele piesei și sala în care se va juca aceasta. O piesă poate fi jucată de mai multe ori (one-to-many de la piesă către reprezentație) , dar o reprezentație este specifică unei singure piese, deoarece conține detalii unice despre data și timpul la care se vor desfășura piesa respectivă. O sală poate fi folosită la mai multe reprezentații însă o reprezentație se poate desfășura într-o singură sală și nu în mai multe simultan.

Descrierea constrângerilor

Entitățile și atributele lor au fost create pe baza a mai multor constrângeri ce au rolul de a forța regulile la nivel de tabelă și de a preveni ștergerea unei tabele sau a datelor din ea dacă existe dependențe. Dintre constrângerile utilizate vom

descrie câteva în cele ce urmează, discutându-le la nivelul fiecărei tabele deoarece fiecare conține constrângeri.

Ca un aspect general la nivelul fiecărui tabel din proiect, la crearea tuturor câmpurilor obilgatorii s-a folosit constrângerea NOT NULL ce impune completarea lor cu o valoare, nefiind permisă opțiunea de a le lăsa necompletate. Totodată, fiecare tabel deține o cheie de tip PRIMARY-KEY identificată prin id-ul specific fiecărei înregistrări dintr-o tabelă de orice tip. Spre exemplu o înregistrare a tabelei decor va fi identificată prin id-ul ei propriu generat prin autoincrement, la fel ca și id-urile celorlalte tabele, mai puțin a tabelelor Decor_Recuzită și Contract care au rezultat din ruperea legăturilor de tip many-to-many și se identifică print primary-key-uri compuse din id-urile tabelelor între care a existat inițial legătura many-to-many.

Câmpul *nume* al tabelei **Decor** poate conține doar anumite valori prestabilite precum “bâlci” sau “parc”, fapt pentru care utilizăm o constrângere de tip CHECK care nu va permite altor tipuri de decoruri nepermise să fie introduse în tabelă, ci doar cele care se găsesc în lista de valori acceptate.

Același tip de constrângere , CHECK, îl folosim și la câmpurile *nume* și *material* ale tabelei **Recuzită** întrucât pentru ambele se acceptă doar anumite valori , cum ar fi ‘armura cavaler’ sau ’valiza’ pentru *nume* și ’caramida’ sau ’lemn’ pentru *material*.

Tabela **Actor** conține constrângeri de tip CHECK pentru validarea datelor despre fiecare actor, cum ar fi lungimea numelui și prenumelui să respecte limita minimă de 3 caractere , numărul de telefon să respecte formatul standard cu 0 la început, email-ul în cazul în care există să conțină caracterul @ și încheierea de 2,3 sau 4 litere precedate de punct, iar mărimea la haine, tot opțională, să se afle între valorile existente de mărimi, de la XS până la XXXXXL inclusiv. Proprii tabelei Actor sunt constrângerile de tip UNIQUE de la nivelul atributelor telefon și email, justificate de faptul că este nevoie ca ele să fie unice pentru a putea contacta actorul în cauză. Spre exemplu dacă doi actori sunt rude și locuiesc în aceeași casă, nu va fi permis să se introducă pentru amândoi în baza de date numărul de telefon fix pe care îl au acasă, ci este nevoie de numărul lor de telefon mobil. Cum fiecare actor deține o cabină proprie, va fi nevoie de o constrângere FOREIGN KEY ce va referi coloana id_cabina a tabelei *Cabină.*

Tabela **Cabină** are o constrângere de tip CHECK care impune ca etajul la care se află cabina să fie între 1 și 3, pentru că la celelalte etaje nu pot exista cabine.

Pentru tabela **Sală** au fost necesare trei constrângeri CHECK , două dintre ele de tipul BETWEEN...AND pentru câmpurile *etaj* și *capacitate* pentru a marca

limitile între care trebuie să se încadreze etajul sălii (teatrul dispunând de spațiu pentru săli de specatol de la parter la etajul 4) și numărul de locuri care poate exista într-o sală (de la 10 la 500), și una de tip IN pentru *nume*, oferind accesul doar anumitor denumiri să intre în baza de date.

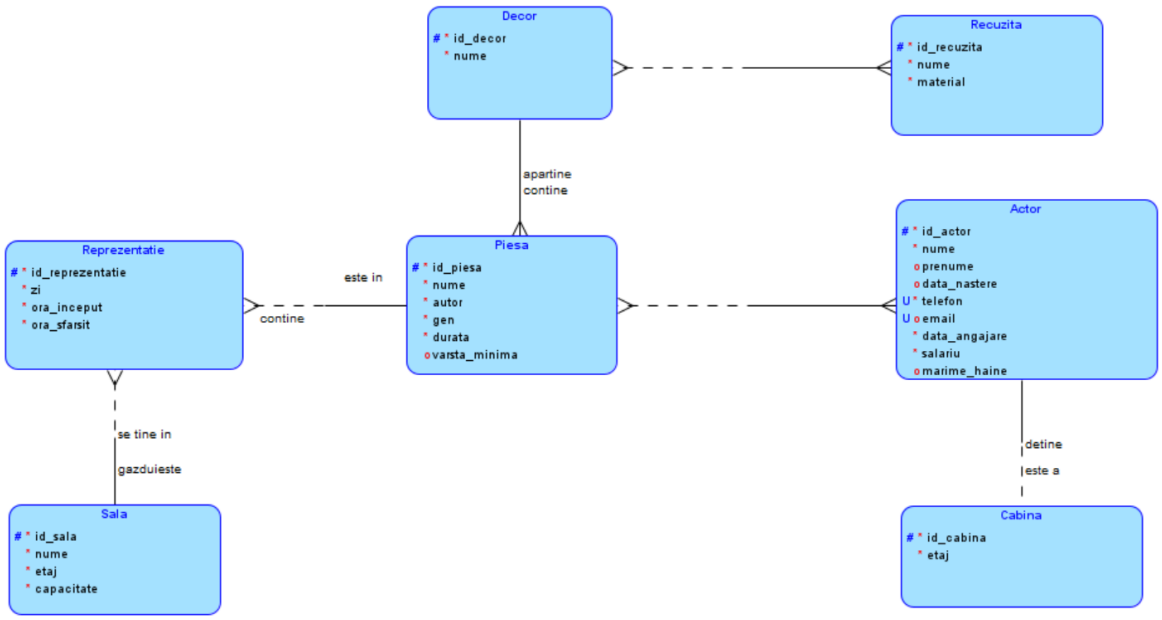
Tabela **Piesă** conține numeroase constrângeri de tip CHECK, folosite fie pentru setarea unei lungimi minime a unei valori ce va urma să fie introdusă în câmpurile *nume* și *autor*, fie pentru oferirea accesului doar a anumite valori dintr-o listă predefinită pentru câmpurile *gen* și *varsta minima* și nu în cele din urmă pentru oferirea unui interval de valori pentru *durata* unei piese. Este necesar un FOREIGN KEY pentru a realiza legătura cu tabela *Decor*, fiecărei piesă fiindu-i desemnat un decor prin intermediul unui id.

Fiind un intermediar între tabela *Actor* și *Piesă*, tabela **Contract** are două constrângeri de tip FOREIGN KEY către coloanele id ale acestor două entități.

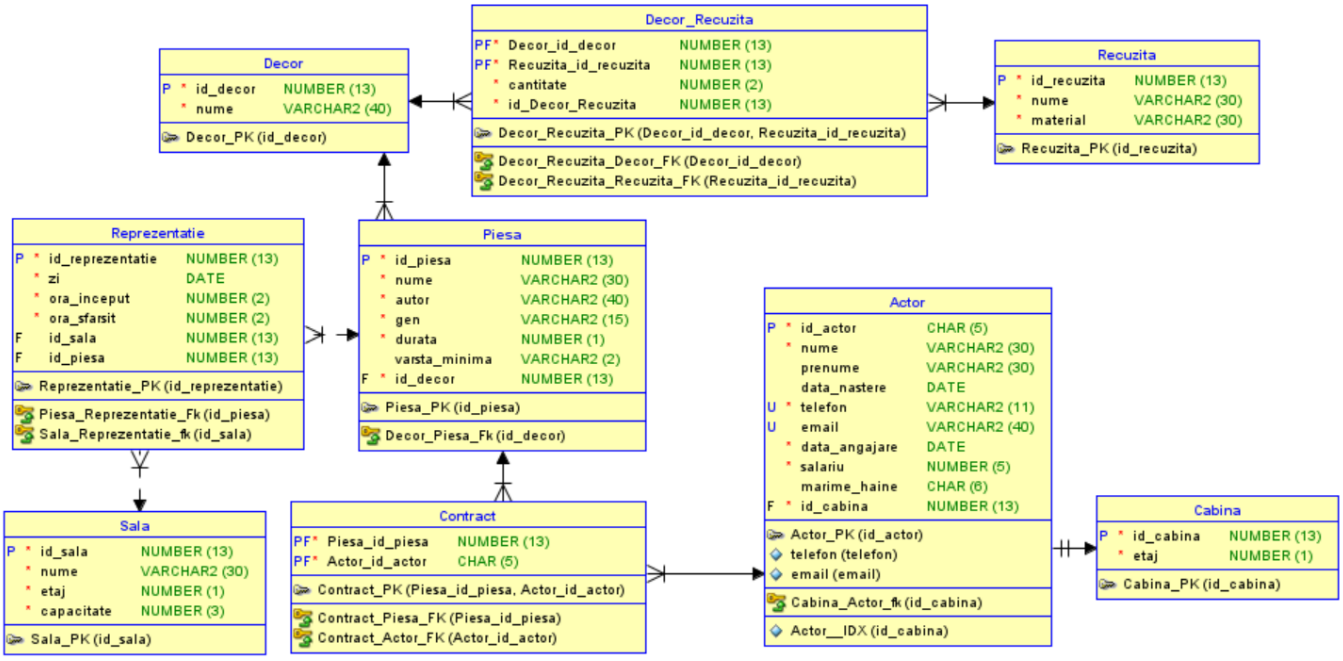
Tabela **Reprezentație** are două constrângeri de tip FOREIGN KEY către id-urile din tabelele Piesă și Sală, scopul ei fiind tocmai rezolvarea relației de tip many-to-many dintre acestea. Pentru câmpurile ce se ocupă cu înregistrarea orei de *început* și de *sfârșit* a unei piese se folosesc constrângeri de tip CHECK ce oferă două intervale de ore comform programului teatrului : de la 8:00 la 22:00.

Structura si inter-relaționarea tabelelor

Modelul Logic



Modelul relațional



Normalizarea Tabelelor

Tabela **Actor** respectă proprietățile următoarelor forme normale:

- Prima formă normală (câmpurile nu conțin valori care se repetă) ex: în *telefon* nu pot există mai multe numere de telefon pentru un actor, la fel ca și *mărime* , pentru că un actor nu poate avea mai multe mărimi simultan, mereu este înregistrată mărimea actuală.
- A doua formă normală : atributele depind de identificatorul unic al entității, de id_actor
- A treia formă normală : toate atributele non-cheie sunt direct dependente de singura cheie primară existentă, id_actor, și nu tranzitiv (prin intermediul altei chei)
- A patra formă normală : nu există dependențe multi-valoare
- A cincea formă normală : join-urile sunt făcute pe baza id_actor

Respectarea proprietăților acestor forme normale face ca tabela **Actor** să se încadreze în FN5.

Pentru celelalte tabele se va specifica doar forma în care se încadrează, fiecare formă implicându-le pe toate dinaintea ei și încă o proprietatea în plus.

În FN5 se mai încadrează tabelele **Sală** , **Cabină**, **Decor** , **Recuzită**, **Piesă, Contract, Reprezentație.**

În FN1 se încadrează **Decor_Recuzită** (FN2 nu este respectată deoarece atributul cantitate depinde doar de cheia-primară Recuzita_id_recuzita nu și de Decor_id_decor, referind cantitatea de obiecte de recuzită necesare pentru realizarea unui decor).

Interfața Aplicației

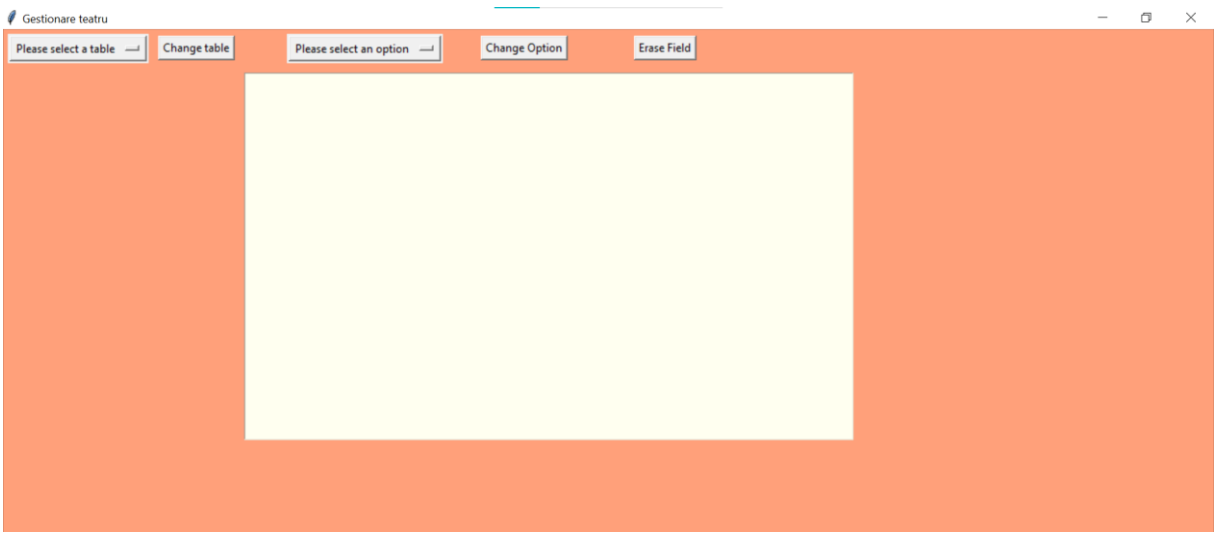
Aplicația este dezvoltată în limbajulu de programare Python, cu ajutorul uneltelor regăsite în pachetul **Tkinter**, ce se ocupă de aspectele ce țin de Graphic User Interface ale unei aplicații Python.

Pentru a realiza acest lucru, s-a importat în mediul de programare pentru limbajul Python librăria **cx_oracle** care facilitează realizarea conexiunii dintre interfață și baza de date Oracle.

Pentru a obține această conexiune, se inițiază un client de tip Oracle folosind librăria **instantclient** și se introduc datele de conectare. Acestea sunt username-ul, password-ul și data source name-ul (dsn).

După ce conexiunea a fost realizată, pentru a executa operații CRUD (create, read, update, delete) folosindu-ne de datele bazei de date la care suntem conectați, se va utiliza un cursor. După ce este construit query-ul , acesta va fi exacutat prin intermediul funcției **execute** apelată cu ajutorul cursorului. Dacă vorbim despre un query ce are ca scop modificarea structurii sau datelor bazei de date, se va executa și comanda *commit* pentru a asigura o schimbare permanentă a bazei de date. Tot în cazul unui query ce modifică baza de date se efectuează și o verificare a execuției acestuia, care informează utilizatorul dacă s-a produs o eroare în momentul în care a încercat să aducă schimbări bazei de date sau dacă comanda lui s-a desfășurat cu succes. O eroare poate apărea atunci când prin execuția query-ului se încalcă o regulă de constrângere impusă de dezvoltatlorul bazei de date sau dacă la scrierea query-ului există erori de sintaxă.

Pagina inițială și menu-ul aplicației

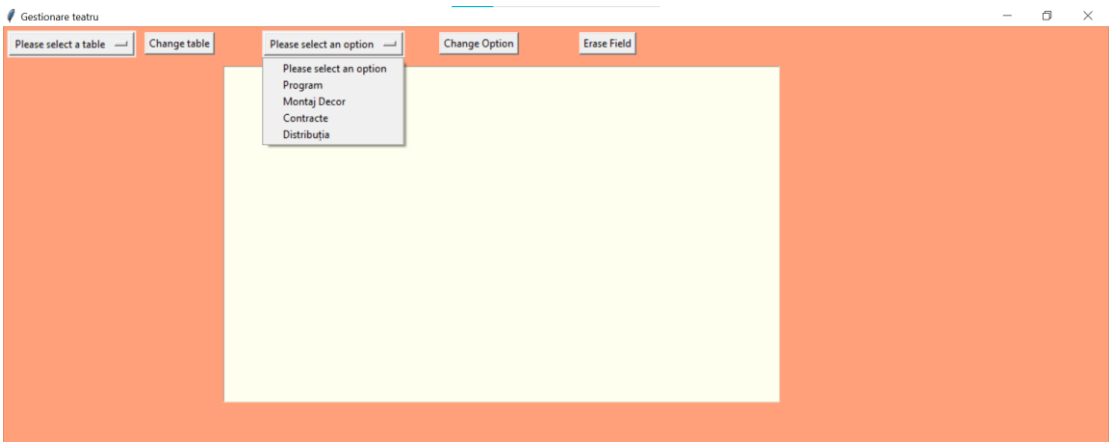


Pagina inițială a interfeței conține field-ul în care se va realiza inserția tuturor query-urilor de tip SELECT regăsite în aplicație.

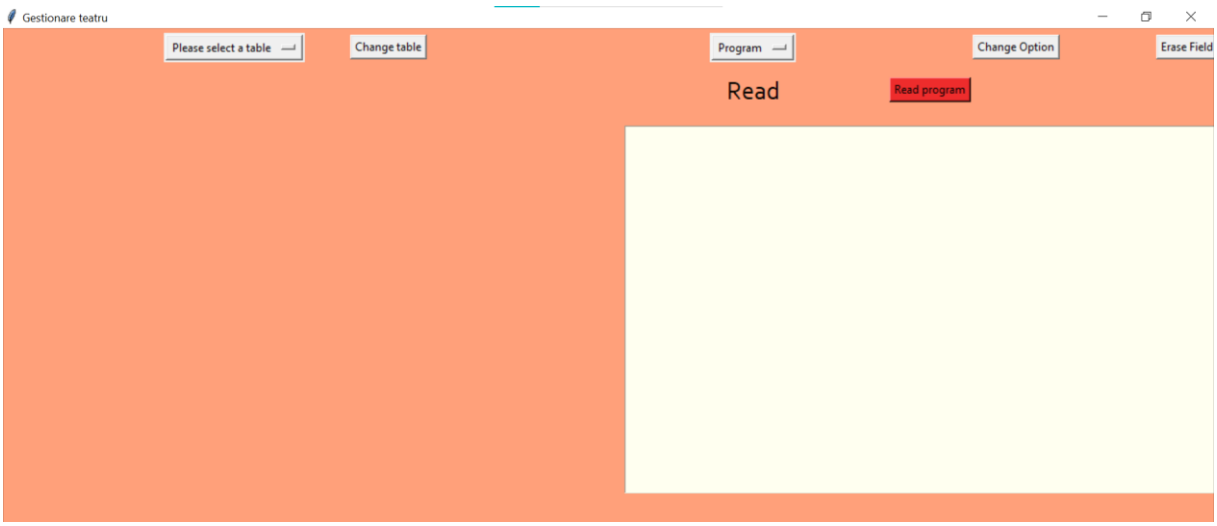
Conținutul acestuia va fi șters de fiecare dată când se "pictează" o pagină diferită a interfeței prin apăsarea butoanelor **Change table** după ce s-a selectat un nou tabel din lista **Please select a table** de tip drop-down list, sau prin apăsarea butonului **Change option** după alegerea unei opțiuni din drop-down list-ul **Please select an option**.

Dacă utilizatorul consideră că e prea aglomerat text field-ul la un moment dat, îl poate goli complet apăsând pe butonul din dreapta sus **Erase Field** care va rămâne pe interfață pe toată durata de viață a programului, ca și restul butoanelor regăsite în pagina de deschidere a aplicației, ce constituie bara de menu a aplicației.

Pagină opțiune



Când se va face click pe butonul de tip drop-down list-ul **Please select an option** utilizatorului i se va afișa o listă cu 4 opțiuni : **Program, Montaj Decor, Contracte, Distribuția**. După ce utilizatorul face click pe opțiunea dorită trebuie să apese butonul **Change Option** pentru ca interfața să ia forma specifică execuției opțiunii dorite.



Pentru opțiunea de tip **Program** interfața va arăta astfel.

Pe lângă menu-ul și chenarul folosit pe post de text box discutate anterior care vor fi prezente pe interfață pe tot parcursul rulării aplicației, se remarcă apariția unui buton nou **Read program.**

Apăsarea butonului va apela funcția **read_program(event)** care va realiza printr-un cursor execuția query-ului de tip **SELECT** ce va face o selecțe a datelor regăsite în tabela reprezentație în funcție de id-ul sălilor și al piesei, afișând în text boxul cu rol de afiș un text care anunță date despre reprezentațiile teatrului, precum data, numele piesei sau sala în care se va ține.

Această funcție de află în interiorul funcției **paint_program ()** (utilizată pentru a proiecta interfața în conformitate cu necesitățile realizării unei ferestre de tip program) .

```
def paint_program():
    print('Programul')
    label_terminal = Label(root, text="Read", bg='lightsalmon', font='Corbel 20')
    label_terminal.grid(row=1, column=5, padx=5, pady=5)

    def read_program(event):
        curs = conn.cursor()
        field.insert(END, "\n\t\t\tProgram: \n\n")
        curs.execute(
            '''SELECT 'Reprezentatia nr. ' || id_reprezentatie || ':' || ' Pe data de ' || r.zi || ' de la ora '||
             r.ora_inceput || ' pana la ' || r.ora_sfarsit || ' se va juca piesa ' || p.nume || ' scrisa de ' ||
             p.autor || '. Spectacolul va avea loc in sala ' || s.nume || ' pe care o gasiti la etajul ' || s.etaj ||
              ' in cladirea teatrului nostru.' FROM reprezentatie r, sala s, piesa p WHERE r.id_piesa= p.id_piesa AND
              r.id_sala=s.id_sala order by zi''')
        res = curs.fetchall()
        for row in res:
            field.insert(END, row)
            field.insert(END, "\n\n\n")
        curs.close()

    btn_read_program = Button(root, text="Read program", bg="firebrick2")
    btn_read_program.grid(row=1, column=6, padx=5, pady=5, sticky=W)
    btn_read_program.bind("<Button>", read_program)
```

Celelalte opțiuni vor fi activate în acelasi mod, iar interfața pentru ele va arăta la fel cu cea a opțiunii *Program*, exceptând butonul **Read program** care va fi diferit, fiecare opțiune în parte având funcțiile **paint_opțiune(), read_opțiune(event)** și butonul **read_opțiune** proprii, dar care la bază se ocupă cu acelasi lucru: afișarea pe interfață a unor date ce țin de programul teatrului , de montajul decorului , de distribuția actorilor sau de contractele acestora.

Pagină Tabel

Când se va face click pe butonul de tip drop-down list-ul **Please select a table** utilizatorului i se va afișa o listă cu 5 opțiuni : **Actor, Cabina Decor, Piesa, Recuzita**. După ce utilizatorul face click pe opțiunea dorită trebuie să apese butonul **Change table** pentru ca interfața să ia forma specifică prelucrării tabelului dorit.

Pentru tabelul **Actor** interfața va arăta astfel.

Pe lângă menu-ul și chenarul folosit pe post de text box discutate anterior care vor fi prezente pe interfață pe tot parcursul rulării aplicației, se remarcă apariția a 4 butoane noi, cele roșii care vor apela funcții ce vor realiza execuția query-urilor de tip INSERT, SELECT, UPDATE și DELETE asupra datelor regăsite în tabela **Actor**.

Pe coloana intitulată **Create** se pot observa mai multe label-uri care vor prelua valorile pe care utilizatorul dorește să le înregistreze în baza de date pentru un actor nou. Se poate observa cheia de tip OptionMenu din dreptul etichetei Id cabină care va permite utilizatorului să aleagă din lista de cabine existentă în baza de date pe cea pe care o dorește să i-o atribuie actorului pe care dorește să-l adauge în sistem.

Acest lucru se realizează prin funcția **update_dp_actor_create()** care lucrează cu un dicționar de cabine și folosește o listă de cabine realizând un Dropdown menu prin intermediul funcției **OptionMenu().** Prin intermediul dicționarului se mapează informatiile cabinei cu id-ul acesteia. Cheia este reprezentată de informatiile cabinei, iar valoarea este id-ul. Când se execută un query ce are nevoie de un foreign key, se găsește valoarea acestuia în dicționarul specific de unde va fi preluată.

```
def update_dp_actor_create():
    global dp_create_cabina_actor
    global variable_create_cabina_actor
    list_cabine = list()
    dictionar_cabine.clear()
    curs = conn.cursor()
    curs.execute("SELECT id_cabina, etaj FROM cabina")
    res = curs.fetchall()
    string_cabina = "New dress room"
    dictionar_cabine[string_cabina] = -1
    list_cabine.append(string_cabina)
    for row in res:
        string_cabina = f"id: {str(row[0])} etaj {str(row[1])}"
        dictionar_cabine[string_cabina] = int(row[0])
        list_cabine.append(string_cabina)
    variable_create_cabina_actor = StringVar(root)
    variable_create_cabina_actor.set(list_cabine[0])
    linie_actor = 10
    paint_label("Id cabina", linie_actor, 0)
    dp_create_cabina_actor = OptionMenu(root, variable_create_cabina_actor, *list_cabine)
    dp_create_cabina_actor.grid(row=linie_actor, column=1, padx=5, pady=5)
```

După ce utilizatorul va completa toate labelurile not null și va selecta o cabină din OptionMenu-ul Id cabina, va putea apăsa butonul **Insert actor.** Acesta va apela o funcție care va realiza introducerea actorul în baza de date dacă datele sunt complete și corecte afișând în text box-ul de pe ecran un mesaj de reușită, sau în caz contrat, la apariția unei erori, va înștiința utilizatorul că datele nu sunt valide.

Coloana **Update** va avea aceeași structura cu a coloanei **Create**, exceptând apariția unui nou OptionMenu **Id actor** ce permite utilizatorului să selecteze actorul pentru care dorește să modifice date. După ce va completa câmpurile valorilor pe care dorește să le modifice, utilizatorul apasă butonul **Update actor** și astfel datele actorului vor fi actualizate dacă sunt valide. În aceeași manieră ca și la inserarea unui nou actor, utilizatorului îi va fi confirmată printr-un text modificarea cu succes a unei înregistrări, sau eșecul acestei acțiuni.

Deasupra textboxului se află un buton **Read actor** care va realiza afișarea tuturor înregistrărilor aflate în prezent în tabela Actor din baza de date.

În partea de jos a textboxului se află un alt OptionMenu care îi permite utilizatorului să selecteze actorul pe care dorește să îl elimine din baza de date, dar și un buton **Delete actor** care apelează funcția **delete_actor** ce va executa query-ul de tip delete pentru actorul selectat în căsuța de deasupra. Când se realizează o functie de stergere sau de editare, trebuie modificate drop down list-urile pentru ca informatia sa fie cea corecta. Dupa un query de update sau delete, se distrug toate drop down-urile și se realizează un select nou pentru a lua informatiile bune. Această metodă face ca utilizatorul să nu fie nevoit să apese pe butonul de **change tabel** ca să vizualizeze informațiile corecte pe interfață si pentru a nu îl deruta.

```
#Delete
global btn_delete_actor
update_dp_actor_delete()
def delete_actor(event):
    curs = conn.cursor()
    try:
        print(dictionar_actori[variable_delete_actors.get()])
        querry = "DELETE FROM actor where id_actor = '%s'" % (dictionar_actori[variable_delete_actors.get()])
        curs.execute(querry)
        curs.execute('commit')
        field.insert(END, "[SUCCES]: Actorul a fost sters!\n")
    except Exception as e:
        print(e)
        field.insert(END, "[EROARE]: Id-ul nu este corect. Verificati sa existe\n")
    dp_delete_actor.destroy()
    dp_update_cabina_actor.destroy()
    dp_create_cabina_actor.destroy()
    update_dp_actor_delete()
    update_dp_actor_create()
    update_dp_actor_update()
btn_delete_actor = Button(root, text="Delete actor", bg="firebrick2")
btn_delete_actor.grid(row=linie_actor - 1, column=6, padx=5, pady=5, sticky=W)
btn_delete_actor.bind("<Button>", delete_actor)
```

Pentru celelalte tabele structura interfeței va fi aceeași :

Coloana **Create** care va conține căsuțe de tip *label* pentru valori obișnuite și *OptionMenu* pentru foreign_key-uri și un buton **Insert nume_tabel**

Coloana **Update** care imită coloana Create, având în plus un *OptionMenu* ce permite selecția id-ului înregistrării pentru care se dorește să se realizeze actualizarea datelor și un buton **Update nume_tabel** cu o funcționalitate diferită

Butonul **Read nume_tabel** de desupra text box-ului care va afișa în acesta toate înregistrările din prezent a tabelei respective.

Butonul **Delete nume_tabel** de sub text box care va realiza ștergerea din tabela curentă a înregistrării selectate în *OptionMenu-ul* din dreapta butonului.