Universitatea Tehnică “Gheorghe Asachi” din Iași

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Gestiunea unei orchestre

Baze de date – Proiect

Student/ă : Voineag Diana-Ioana

Grupa: 1309B

Coordonator: Mironeanu Cătălin

Cuprins

1. Cerințele aplicației

Titlu proiect: Gestionarea unei orchestra

Analiza, proiectarea și implementarea unei baze de date care să stocheze și să modeleze informații despre muzicienii, concertele orchestrei, piesele din repertoriul concertelor, instrumentele necesare și gestiunea asignării acestora, având ca scop ușurarea procesului de a organiza un concert atât din perspectiva comisiei de organizare cât și a membrilor orchestrei.

Descrierea cerințelor și modelul de organizare al proiectului

Baza de date propusă dorește automatizarea procesului organizării unui concert, urmărind îndeaproape îndeplinirea tuturor condițiilor pentru ca evenimentul să se desfășoare conform (necesarul de muzicieni, instrumente, date calendaristice și locații este îndeplinit), cât și automatizarea și gestionarea asignării de instrumente muzicienilor pentru evenimentele create sau pentru altele (nu se ține cont de concerte în vederea asignării instrumentelor). Astfel, în orice moment se poate ști care exemplar dintr-un anumit tip de instrument a fost asignat și cui (dintre membrii orchestrei, desigur). Pe lângă acest lucru, se pot afla diverse informații precum repertoriul de piese muzicale din cadrul unui concert, cât de pregătiți sunt muzicienii și pe ce instrumente profesează, numărul de exemplare din fiecare tip de instrument și informații privind disponibilitatea și starea de conservare a acestora.

Aplicația funcționează în baza următoarelor constrângeri:

• Un instrument poate fi asignat doar dacă este disponibil (ori nu apare în înregistrările asignărilor ori data de final a asignării e înaintea datei de astăzi) și într-o stare de conservare bună sau foarte bună.

• Un instrument poate fi asignat doar la un muzician o dată.

• Nu pot fi ținute 2 concerte în aceeași locație și la aceeași dată.

• Necesarul de pregătire a fiecărui muzician este de minim 5 ani.

• Data de încheiere a asignării trebuie să fie după data de începere și nu se gestionează repercusiunile nerespectării acestor date.

• Numărul de unități alocate (numărul de exemplare din Inventarul de Instrumente) va fi mereu mai mic sau egal cu numărul de unități general disponibile.

• Nu pot exista 2 sau mai multe piese cu același titlu.

• E-mail-ul personal este principala sursă de informații de contact a muzicienilor, deci acesta va fi unic.

• Avem un număr limitat de instrumente din categorii bine definite (Strings, Percussion, WoodWinds și Brass)

• Starea de conservare a unui exemplar dintr-un tip de instrument poate fi bună (Good), foarte bună (New) sau rea/de nefolosit (Not_Usable), iar cele de nefolosit nu pot fi asignate.

Informațiile de care avem nevoie sunt legate de:

- Piese muzicale: Cod, Titlu, Autor/i
- Muzicieni: ID, Nume, Anii de studii, E-mail, Data nașterii
- Instrumente - fictive: Tip, Categorie, Unități disponibile, Unități alocate în inventar
- Instrumente – fizice/exemplare: ID, Stare de conservare, disponibilitate (se poate deduce)
- Concerte: Id, Data, Locație, necesar

2.Soluția propusă

-Descriere Tehnică-

Dupa analiza cerintelor clientului, s-a hotarat crearea bazei de date precum si testarea operatiilor ce trebuie implementate cu ajutorul urmatoarelor produse Software puse la dispozitie de Oracle: SQL Developer si SQL Data Modeler.

Din textul problemei, reies 8 entități cu următoarele atribute:

1. Entitatea **Musicians (Muzicieni)** cu atributele:
   - *ID* -> cheie primară (tip NUMERIC(3))
   - *Name* -> obligatoriu (tip VARCHAR(30))
   - *Year_Of_Studies* -> numărul de ani de studiu în domeniul muzicii-> trebuie saă fie mai mare egal ca 5 (tip NUMERIC(2))

2. Entitatea **Musician_Details (Detalii muzician)** cu atributele:
   - *Contact_Email* -> obligatoriu, respectă formatul de e-mail, unic (tip VARCHAR(30))
   - *Birth_Date* -> opțional (tip Date)

3. Entitatea **Musical_Pieces (Piese muzicale)** cu atributele:
   - *Code* - > cheie primară (tip NUMERIC(3))
   - *Title* -> obligatoriu și unic (tip VARCHAR(25))
   - *Name_of_Authors* -> opțional (tip VARCHAR(25))

4. Entitatea **Instruments (Instrumente)** cu atributele:
   - *Type* -> cheie primară (tip VARCHAR(20))
   - *Number_Of_Units_Available* -> obligatoriu și mai mare ca 0 (tip NUMERIC(2))
   - *Number_Of_Units_Allocated* - > crește odată cu inserarea unui instrument în inventar (Inventory), are rolul de a verifica depășirea numărului de *Number_Of_Units_Available* (tip NUMERIC(2))

- *Category* -> obligatoriu și valoare luată din lista de parametri (Strings, Percussion, Brass, Woodwinds) (tip VARCHAR(25))

5. Entitatea **Inventory (Inventar)** cu atributele:
   - *Instrument_ID - >* cheie primară (tip NUMERIC(3))
   - *Conservation_State ->* obligatoriu și poate lua valori din lista de valori {New, Good, Not_Usable} (tip VARCHAR(20))

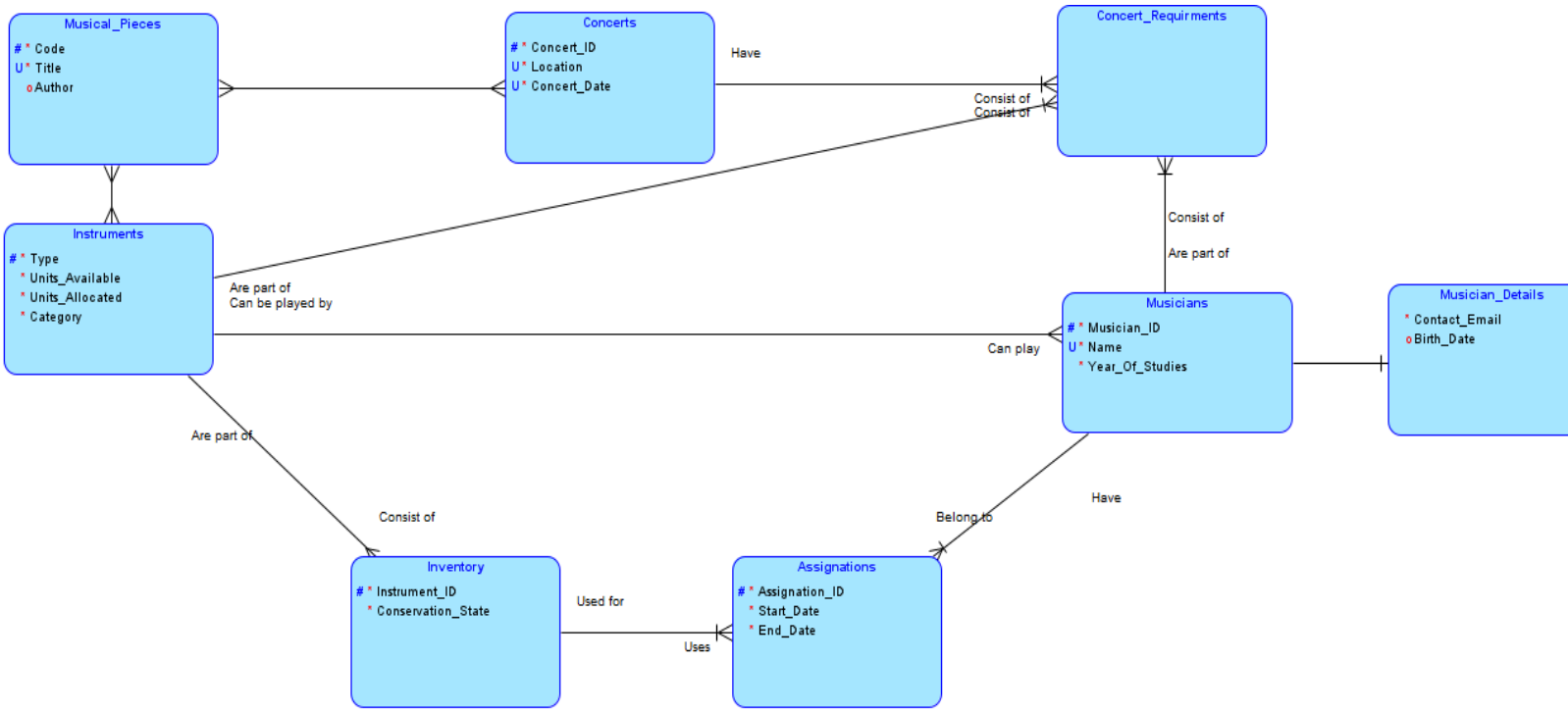
6. Entitatea **Assignations (Asignări)** cu atributele:
   - *Asignment_ID* ->cheie primară (tip NUMERIC(3))
   - *Start_Date ->* data de început a asignării, obligatorie (tip DATE)
   - *End_Date ->* data de sfârșit a asignării, obligatorie, mai mare ca *Start_Date* (tip DATE)

7. Entitatea **Concerts (Concerte)** cu atributele:
   - *Concert_ID* -> cheie primară (tip NUMERIC(3))
   - *Location* (tip VARCHAR(20))
   - *Date* -> Location + Date->combinație unică, ambele atribute sunt obligatorii (tip DATE)

8. Entitatea **Concert_Requirments (Necesare concert)**
   - Reunește pentru fiecare concert, fiecare muzician și fiecare instrument care intră în compoziția acelui concert

Toate aceste entități, precum și legăturile dintre ele pot fi vizualizate prin Diagrama Modelului Logic din capitolul următor.

3.Diagrama Model Logic



4. Descriere relații dintre entități si mod de aparitie tabele

În proiectarea aplicației s-au identificat relații de 1:1, 1:n, n:1 și n:m.

Între entitătile **Musicians** și **Musician_Details** există o relație de 1:1, deoarece unui muzician îi este asociat un singur set de informații (astfel se păstreaza integritatea informațiilor respective). De asemenea, un set de informații este identificat în mod unic de un muzician.

Între entitătile **Instruments** și **Inventory** există o relație de one-to-many, deoarece pot exista mai multe exemplare din același tip de instrument în inventar, dar nu poate exista un exemplar de mai multe tipuri.

Între entitătile **Inventory** și **Assignations** este o relație de one-to-many, deoarece același exemplar de instrument din inventar poate fi asignat de mai multe ori (nu in același timp), dar o asignare corespunde unui singur exemplar (în combinație cu un singur muzician) în mod unic.

Aici merită mentionată și relația dintre **Assignations** și **Musicians** de many-to-one care descrie proprietatea asignărilor de a fi multiple pentru fiecare muzician in parte, dar un muzician identifică în mod unic (în combinatie cu un exemplar de instrument) o asignare anumită.

Între entitătile **Concerts** și **Instruments** ar fi trebuit să existe o relație de many-to-many, deoarece un concert poate avea un necesar de mai multe tipuri de instrumente și fiecare tip de instrument poate intra în compoziția mai multor concerte. De asemenea, între entitățile **Muzicians** si **Concerts** ar fi trebuit să existe o relație de many-to-many, deoarece la un concert canta mai multi muzicieni, iar același muzician poate cânta la mai multe concerte.

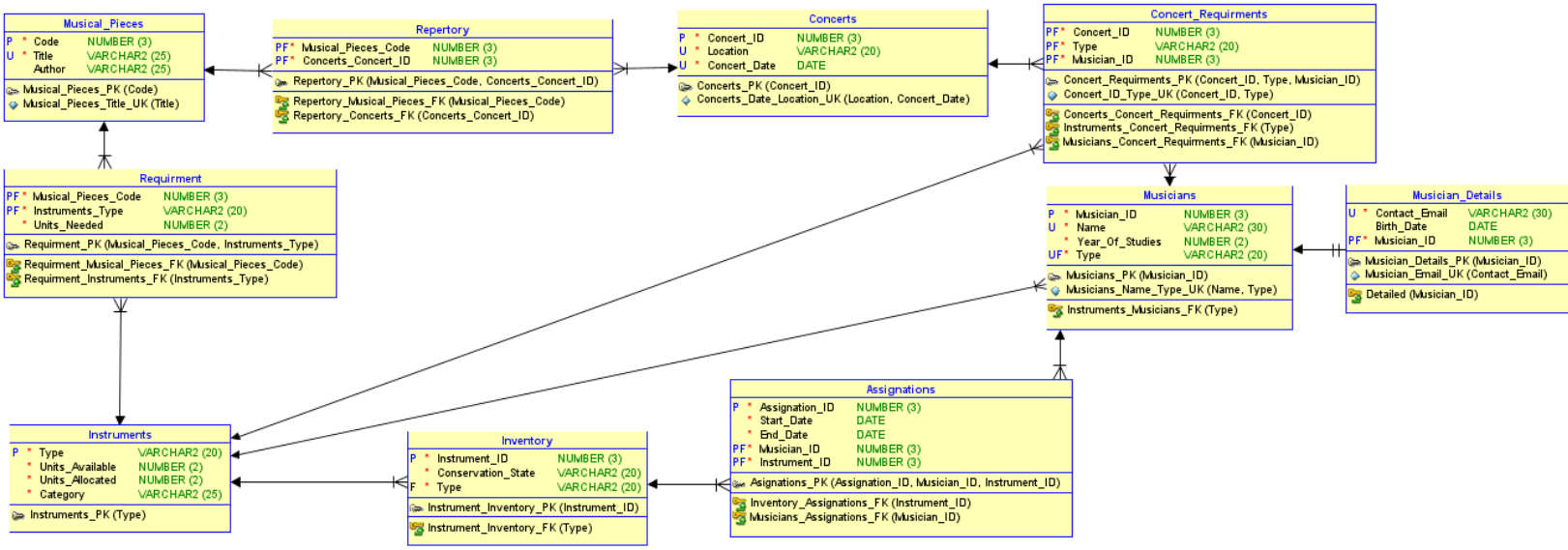
Astfel, pentru a nu mai crea două entități suplimentare ca urmare a transformărilor legăturilor many-to-many, a aparut tabela **Concert_Requirments** care conține informatii legate de necesarul unui concert atât din punct de vedere instrumental cât și al muzicienilor.

Între entitătile **Musical_Pieces** și **Instruments** exista o relatie de many-to-many, deoarece o piesa poate avea un necesar de mai multe tipuri de instrumente si fiecare tip de instrument poate intra in compozitia mai multor piese musicale, rezultă în modelul fizic tabela **Requirment** care conține informații referitoare la necesarul de instrumente al fiecărei piese muzicale.

Între entitătile **Musical_Pieces** si **Concerts** e o relație de many-to-many, deoarece în compoziția repertoriului unui concert intră mai mult de o piesă, iar aceeasi piesa poate fi cantata la mai multe concerte, rezultă, în modelul fizic, tabela **Repertory** care conține informații despre compoziția repertoriului fiecărui concert în parte.

5. Diagrama Model Fizic

În modelul fizic am implementat auto-incrementări pentru cheile primare ale tabelelor Musical_Pieces, Musicians, Concerts, Inventory și Assignations. De asemenea am adăugat un trigger pe câmpurile Concert_Date si Start_Date ale tabelelor Concerts, respectiv Assignations care verifică dacă data la care se încearcă să se organizeze concertul/asigneze un instrument unui muzician este după data de azi.



6. Normalizarea Tabelelor

O relație este în **prima formă normală** dacă satisface următoarele condiții:

- un atribut conține valori atomice din domeniul său (și nu grupuri de astfel de valori);

- nu conține grupuri care se repetă.

Baza de date proiectată se află în prima formă normală deoarece fiecare atribut conține valori atomice, atribute precum email-ul de contact care este considerat a fiind unic.

O relație este în **a doua formă normală** dacă satisface următoarele condiții:

- este în prima formă normală;

- toate atributele non-cheie depind în totalitate de TOATE cheile candidat.

Baza de date proiectată se află în a doua formă normală deoarece fiecare cheie este unică și este direct legată de celelalte atribute și, conform celor de mai sus, baza de date se află și în prima formă normală.

O relație este în a treia formă normală dacă satisface următoarele condiții:

- este în a doua formă normală;

- toate atributele non-cheie sunt direct (non-tranzitiv) dependente de TOATE cheile candidat.

Baza de date proiectată se află în a treia formă normală deoarece se află în primele două forme normale, iar fiecare atribut non-cheie este dependent de câte o cheie candidat unică, iar între toate atributele non-cheie și cheie există o dependență.

7.Conectarea la baza de date

Conexiunea cu baza de date este realizata prin intermediul modulului ‘oracledb’ din nodejs. In acest sens, am salvat toate informatiile de conectare la baza de date intr-o variabila ‘dbConfig’ care are urmatoarele campuri:

```
const dbConfig = {
    user: 'bd116',
    password: 'bd116',
    connectString: 'bd-dc.cs.tuiasi.ro:1539/orcl',
    poolMax: 10,
    poolMin: 2,
    poolIncrement: 2,
    poolTimeout: 60
};
```

Parametrii precum user, password, și connectString sunt specifici bazei de date Oracle. Acești parametri sunt utilizați pentru a configura conexiunea. Campurile care contin terminologia ‘pool’ sunt configuratii pentru pool-ul de conexiuni realizat pentru a permite reutilizarea conexiunilor, fapt care eficientizeaza accesul la baza de date.

```
let pool;

async function init() {
    pool = await oracledb.createPool(dbConfig);
}

init();
```

Functia ‘init’ este apelata pentru a initializa acest pool.

De asemenea, pentru fiecare ruta express care necesita access la baza de date se utilizeaza functia din modulul oracledb, ‘oracledb.getConncection()’ pentru a obtine o conexiune din pool si ‘connection.close()’ pentru a o elibera.

8.Tehnologii

Pe partea de backend a aplicatiei am folosit framework-ul de JavaScript NodeJs, mai exact am folosit framework-ul Express.js. Am ales acest framework deoarece acesta pune la dispozitie o modalitate simplista de a crea aplicatii Web.

Mai jos se poate observa cum importam modulele necesare aplicatiei si facem ca server-ul sa asculte pe portul 3000.

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const oracledb = require('oracledb');

const app = express();
const port = 3000;

// Setez folderul de unde se vor lua fisierele statice
app.use(express.static(path.join(__dirname, 'views')))

// Set the view engine to EJS
app.set('view engine', 'ejs');

// Use body-parser middleware
app.use(bodyParser.urlencoded({ extended: true }));
```

-import module utilizate-

```
// Start the server
app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});
```

-start server port 3000-

Pentru fiecare pagina s-a creat in partea de backend o ruta /GET care realizeaza query-uri de select pe tabelele din baza de date si trimite datele catre frontend, unde paginile .ejs le manipuleaza pentru a fi vizualizate de catre utilizator.

In poza de mai jos se poate observa cum se creeaza o conexiune la baza de date, se face un query pentru a selecta datele referitoare la concerte, se trimit catre pagina ejs ‘concerts.ejs’, iar apoi se inchide conexiunea.

```
app.get('/concerts', async (req, res) => {
    try {
        connection = await oracledb.getConnection();
        query = 'SELECT * FROM concerts';
        result = await connection.execute(query);
        concerts = result.rows.map(concert => {
            return [concert[0], concert[1], formatDateTime(concert[2])]
        });

        res.render('concerts', { concerts });
    } catch (error) {
        console.error('Error fetching concerts:', error);
        res.status(500).send('Internal Server Error');
    } finally {
        connection.close();
    }
  });
```

Pentru operatiile de inserare s-au create rute ‘/POST’ care sa preia datele de la formularele din partea de frontend si care sa execute query-uri de inserare sau actualizare in baza de date. In poza de mai jos se poate observa cum este tratata actualizarea unor informatii din tabela ‘Inventory’.

```
app.post('/submitUpdateConservation', async (req, res) => {
  const instrumentId = req.body.instrumentId;
  const newConservationState = req.body.newConservationState;
  const connection = await oracledb.getConnection(dbConfig);
  try {

    const result = await connection.execute(
      `UPDATE inventory SET conservation_state = :newConservationState WHERE instrument_id = :instrumentId`,
      { newConservationState, instrumentId }
    );
    await connection.commit();
    res.render('updateState', { instrumentId, successMessage: 'State altered successfully!', errorMessage: null });
  } catch (error) {
    console.error('Error changing state:', error);
    res.render('updateState', {instrumentId ,errorMessage: 'State could not be altered.', successMessage: null });
  } finally{
    await connection.close();
  }
});
```

In partea de frontend folosim pagini ejs pentru a putea accesa variabilele trimis de pe partea de backend. De exemplu, in poza de mai jos, se poate observa cum accesam variabila concerts care contine informatii despre concerte, pentru a le afisa sub forma unui tabel. Practic, paginile ejs sunt pagini html in care putem introduce secvente de cod JavaScript.

```
<body>
  <h1>Concerts List</h1>
  <table>
    <thead>
      <tr>
        <th>Location</th>
        <th>Time</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <% concerts.forEach(concert => { %>
        <tr class="clickable-row" onclick="redirectToDetails('<%= concert[0] %>')">
          <td><a href="#"><%= concert[1] %></a></td>
          <td><%= concert[2] %></td>
          <td>
            <a href="/updateConcert/<%= concert[0] %>"><button>Delay</button></a>
            <a href="/cancelConcert/<%= concert[0] %>"><button>Cancel</button></a>
          </td>
        </tr>
      <% }); %>
    </tbody>
  </table>
```

In paginile ejs in care facem inserari/actualizari folosim form-uri pentru a permite utilizatorului sa introduca date. De exemplu, in poza de mai jos se pot observa campurile formularului corespunzator adaugarii unui nou concert.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add Concert</title>
  <link rel="stylesheet" type="text/css" href="/styles/form.css">
</head>
<body>
  <h1>Add Concert</h1>
  <% if (successMessage) { %>
    <div class="alert success">
      <%= successMessage %>
    </div>
  <% } %>
  <form action="/submitConcert" method="post">
    <label for="location">Location:</label>
    <input type="text" id="location" name="location" required>

    <label for="concertDate">Concert Date:</label>
    <input type="date" id="concertDate" name="concertDate" required>

    <button type="submit" class="button-submit">Submit</button>
  </form>
  <a href="/concerts" class="button-go-back">Go Back to Concerts List</a>
</body>
</html>
```

9.Operatia de tranzactie

Operatia de tranzactie este realizata pe tabelele ‘Inventory’ si ‘Instruments’ si are ca rol validarea inserarilor din aceastea prin intermediul atributelor ‘Units_Available’ si ‘Units_Allocated’. Astfel, la fiecare inserare in tabela Inventory se incrementeaza si atributul ‘Units_Allocated’ pentru acel tip de instrument din tabela ‘Instruments’ si se verifica sa nu depaseasca valoarea atributului ‘Units_Available’. Daca o depaseste se afiseaza un mesaj de eroare si se efectueaza rollback, iar daca nu se face update pe tabela ‘Instruments’ cu units_allocated incrementat cu 1 si se insereaza in tabela ‘Inventory’ exemplarul dorit. O tranzactie asemanatoare s-a realizat si la operatia de stergere a unui exemplar din tabela

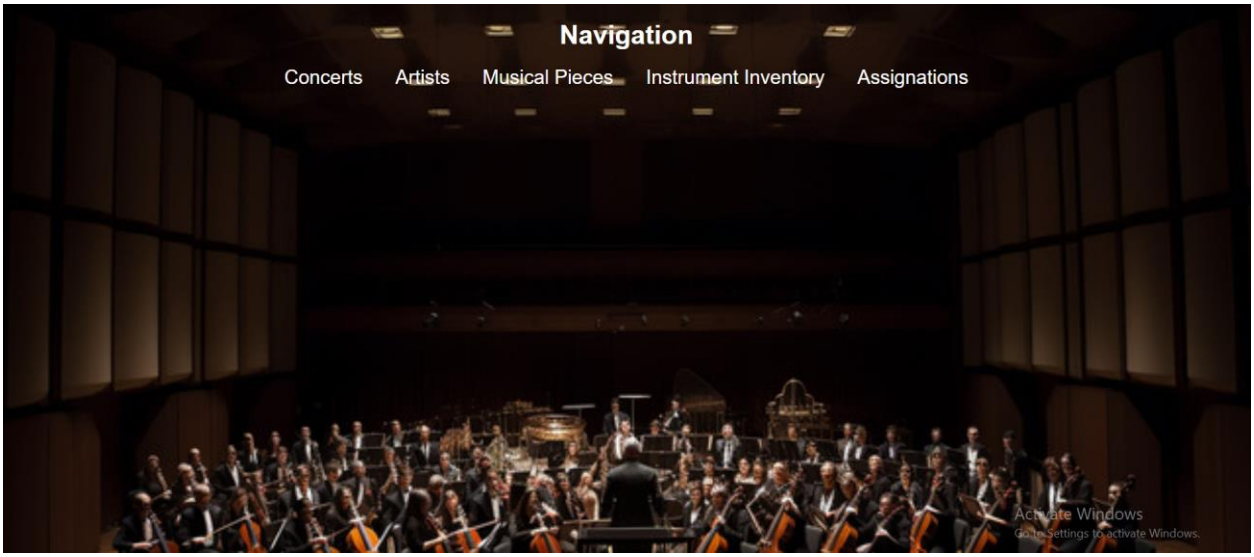
‘Inventory’ intrucat atributul ‘Units_Allocated’ trebuie decrementat. Aceasta decrementare poate avea loc numai daca atributul este nenul. Se poate observa, in poza de mai jos , codul SQL ce realizeaza tranzactia pentru operatia de inserare in tabela ‘Inventory’:

```
DECLARE
  v_instrument_type VARCHAR2(20) := :type;
  v_units_available NUMBER;
  v_units_allocated NUMBER;
  v_error_message VARCHAR2(500);
  v_success_message VARCHAR2(500);
BEGIN
  SELECT units_available, units_allocated
  INTO v_units_available, v_units_allocated
  FROM instruments
  WHERE type = v_instrument_type
  FOR UPDATE;
  v_units_allocated := v_units_allocated + 1;

  IF v_units_allocated > v_units_available THEN
    RAISE_APPLICATION_ERROR(-20001, 'Transaction failed: The number of units allocated exceeds the number of units available for this instrument');
  ELSE
    UPDATE instruments
    SET units_allocated = v_units_allocated
    WHERE type = v_instrument_type;
    COMMIT;
    v_success_message := 'Instrument added to inventory successfully.';
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'No data found for the specified instrument type');
  WHEN OTHERS THEN
    v_error_message := 'Internal Server Error: ' || SQLERRM;
    RAISE_APPLICATION_ERROR(-20003, v_error_message);
:successMessage := v_success_message;
:errorMessage := v_error_message;
END;
```

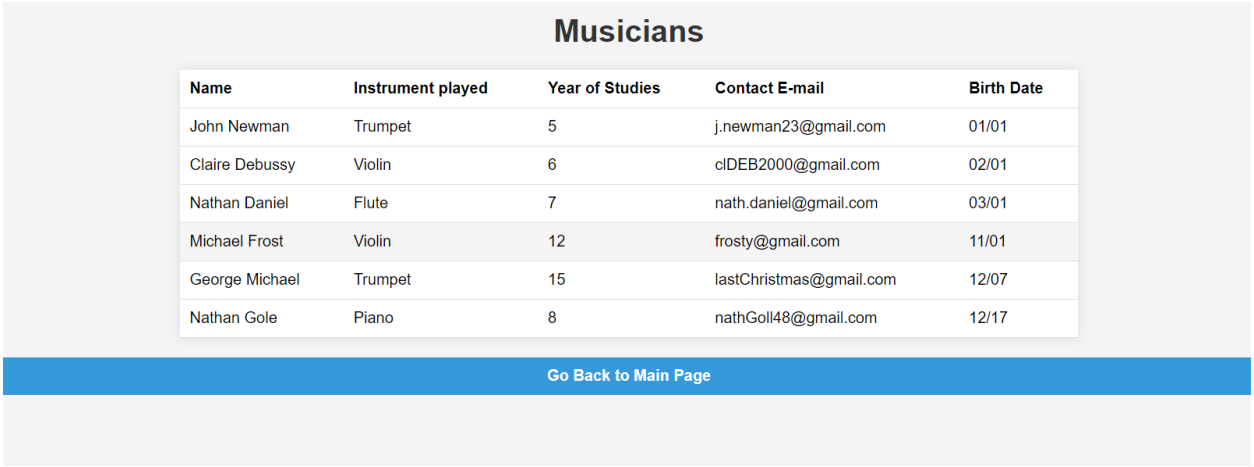
10.Interfata Grafica

Interfata grafica este impartita in mai multe pagini Web. Pe pagina principala se afla un meniu de navigare din care se poate ajunge la pagini de interes precum: vizualizare Concerte, piese musicale, instrumente, asignari si inventar instrumente.



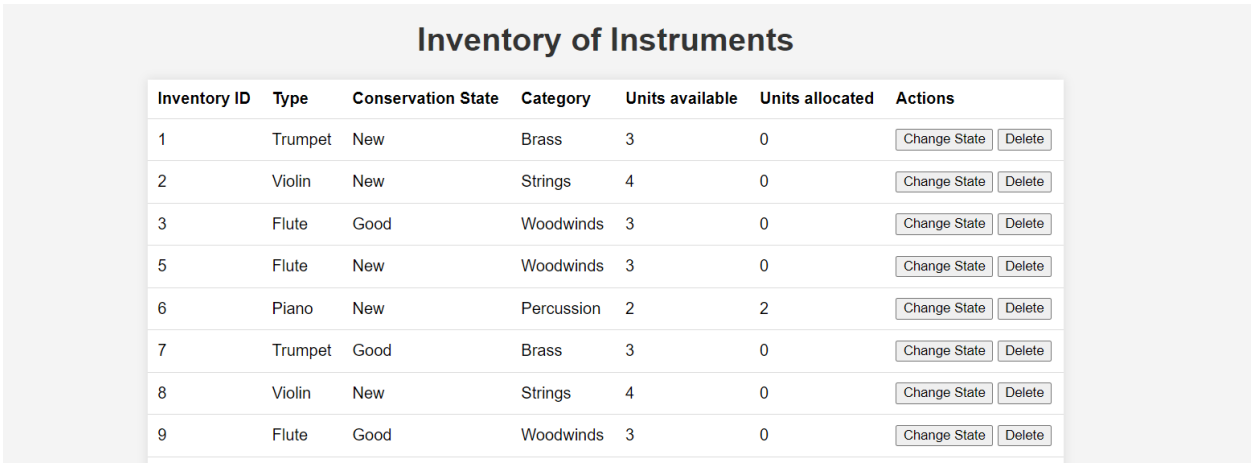
-pagina principala a aplicatiei-

Daca navigam in submeniul de muzicieni (‘Musicians’) se poate observa o lista cu muzicienii din baza de date si detaliile acestora.



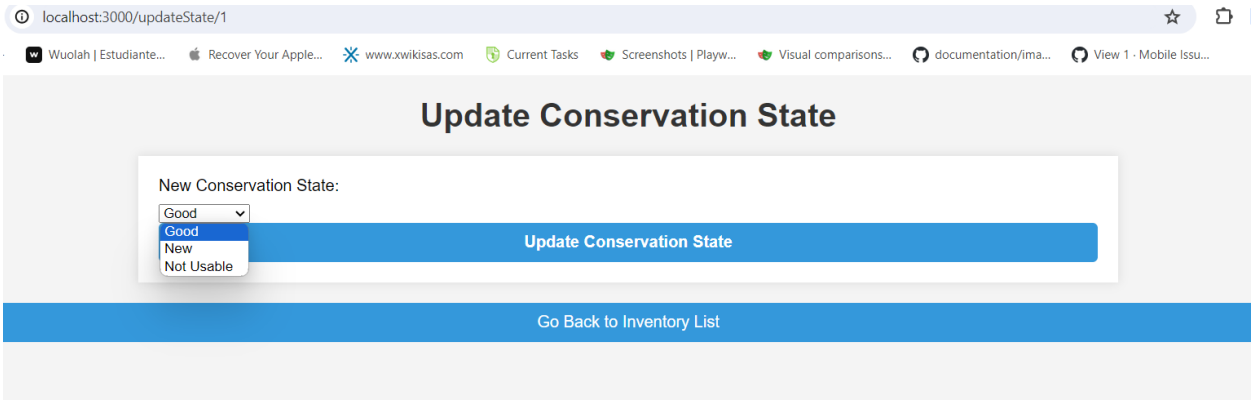
-pagina vizualizare muzicieni-

De asemenea, daca navigam in submeniul Inventar (‘Inventory’) se poate vedea o lista cu mai multe exemplare, posibil din acelasi tip de instrument, fiecare avand id-ul sau si starea sa de conservare. Este nevoie de afisarea id-ului deoarece pot exista 2 instrumente din acelas tip cu aceeasi stare de conservare, dar care totusi sa fie exemplare diferite. In coloanal ‘Actions’ se pot observa 2 butoane, unul de actualizare a starii de conservare (‘Change State’) si unul de stergere a exemplarului din inventar (‘Delete’).

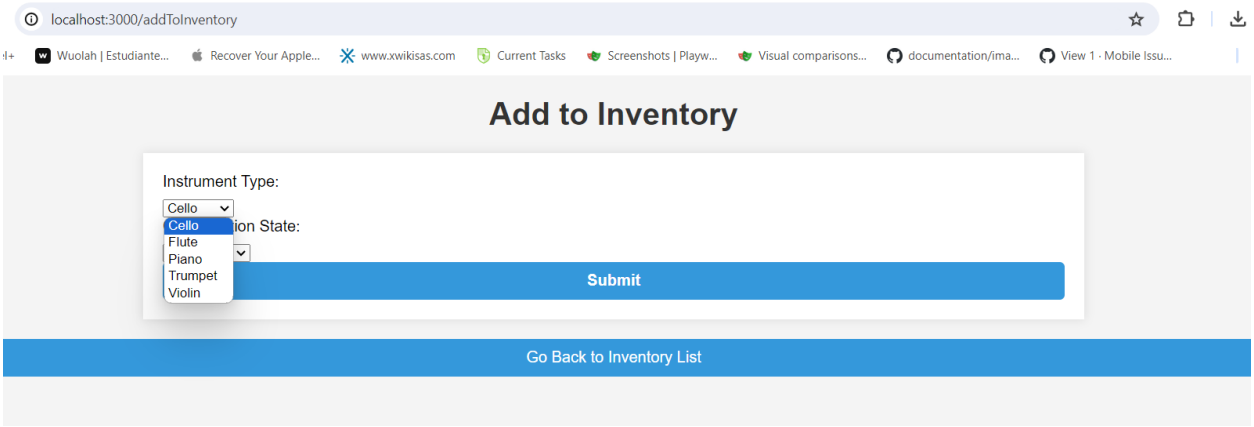


Atunci cand apasam pe butonul de ‘Change State’ din cadrul unei linii se navigheaza spre o pagina de unde putem modifica starea de conservare prin intermediul

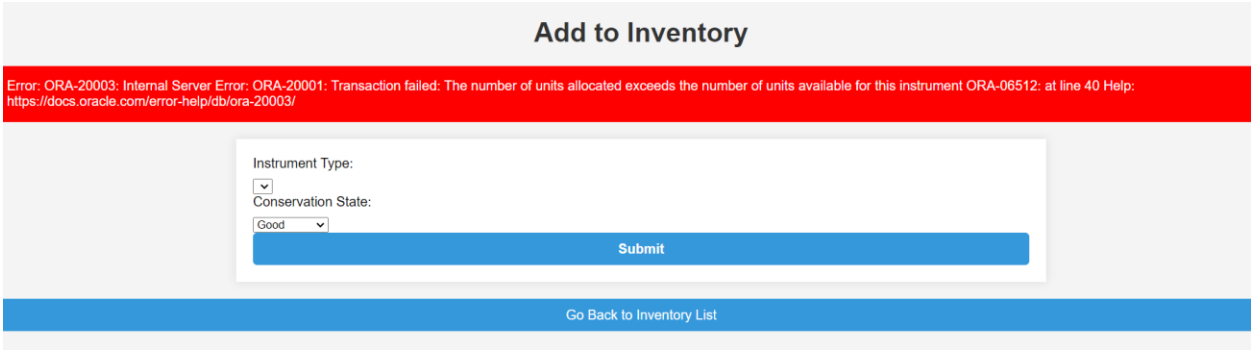
unui dropdown, conform constraint-urilor de check din baza de date. Se mai poate observa in url ca id-ul exemplarului din inventar se transmite automat din vechea pagina.



Pentru inserarea in tabela ‘Inventory’ tipul instrumentului si starea acestuia de conservare se aleg dintr-un dropdown pentru a nu fi posibil ca utilizatorul sa introduce valori ce nu exista in baza de date.



Interfata grafica trateaza, de asemenea, si mesajul de eroare aruncat de esuarea tranzactiei in baza de date, astfel daca incercam sa introducem in inventar mai multe exemplare decat sunt disponibile, vom vedea acest mesaj de eroare:



11. Bibliografie

[1] Suport de curs Baze de date

[2] Documentație Oracle – docs.oracle.com

[3] Oracle SQL Developer, Packt Publishing