Student: Hofman Sebastian

Grupa: 1309A

Disciplină: Baze de date

Coordonator: Cătălin Mironeanu

08 January 2023

World Cup 2022 database

Aplicația a fost creată cu scopul de a ține gestiunea bazei de date a unui turneu de fotbal, cel mondial care s-a terminat acum în Decembrie. Baza de date conține patru tabele:

- **Teams** - cele 32 echipe prezente la cupa mondiala
- **Players** - toți jucătorii prezenți la cupa mondiala
- **Player Stats** - statisticile jucătorilor: goluri marcate, pase de gol, cartonașe galbene, etc.
- **Stadiums** - stadioanele pe care s-au jucat meciurile

Prin această aplicație se pot modifica vizual, prin intermediul interfeței cu utilizatorul, principalele funcții ale unei baze de date. Așadar, se pot **vizualiza, modifica** și **șterge** date din tabele doar cu un simplu click pe niște butoane, programul folosind în partea de backend funcții precum *SELECT, INSERT, DELETE, UPDATE*.

Pentru realizarea aplicației s-a folosit limbajul de programare **Java** și următoarele librării externe:

- Oracle JDBC Driver (v11) - pentru accesarea bazei de date Oracle
- JavaFX - pentru realizarea interfeței grafice cu utilizatorul
- Apache commons-lang - pentru câteva funcții folosite în cadrul aplicației (ex: StringUtils)

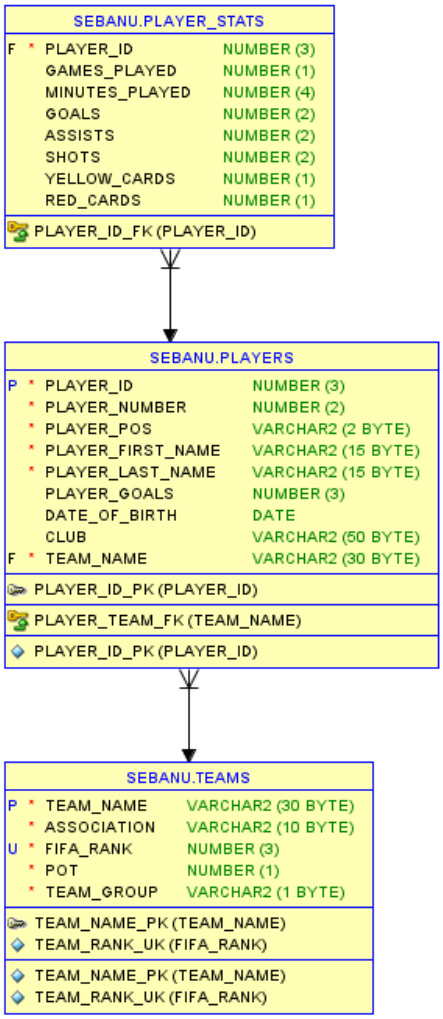
JavaFX si **Apache commons-lang** au fost adaugate în proiect prin intermediul fișierului *pom.xml* din cadrul Maven.

```
<dependencies>
   <dependency>
       <groupId>org.openjfx</groupId>
       <artifactId>javafx-controls</artifactId>
       <version>11</version>
   </dependency>
   <dependency>
       <groupId>org.openjfx</groupId>
       <artifactId>javafx-fxml</artifactId>
       <version>11</version>
   </dependency>
   <dependency>
       <groupId>org.apache.commons</groupId>
       <artifactId>commons-lang3</artifactId>
       <version>3.8.1</version>
   </dependency>
</dependencies>
```

Oracle JDBC Driver a fost adăugat manual prin adaugarea fișierului **.jar** în structura proiectului urmând pașii:

1. File > Project Structure > Libraries
2. Apăsați butonul + și selectați Java.
3. Localizați fișierul .jar în calculatorul dvs. și adăugați-l.
4. Apăsați Apply și apoi OK.

Diagrama ER


SEBANU.PLAYER_STATS
F * PLAYER_ID NUMBER (3)
GAMES_PLAYED NUMBER (1)
MINUTES_PLAYED NUMBER (4)
GOALS NUMBER (2)
ASSISTS NUMBER (2)
SHOTS NUMBER (2)
YELLOW_CARDS NUMBER (1)
RED_CARDS NUMBER (1)
PLAYER_ID_FK (PLAYER_ID)
SEBANU.PLAYERS
P * PLAYER_ID NUMBER (3)
* PLAYER_NUMBER NUMBER (2)
* PLAYER_POS VARCHAR2 (2 BYTE)
* PLAYER_FIRST_NAME VARCHAR2 (15 BYTE)
* PLAYER_LAST_NAME VARCHAR2 (15 BYTE)
PLAYER_GOALS NUMBER (3)
DATE_OF_BIRTH DATE
CLUB VARCHAR2 (50 BYTE)
F * TEAM_NAME VARCHAR2 (30 BYTE)
PLAYER_ID_PK (PLAYER_ID)
PLAYER_TEAM_FK (TEAM_NAME)
PLAYER_ID_PK (PLAYER_ID)
SEBANU.TEAMS
P * TEAM_NAME VARCHAR2 (30 BYTE)
* ASSOCIATION VARCHAR2 (10 BYTE)
U * FIFA_RANK NUMBER (3)
* POT NUMBER (1)
* TEAM_GROUP VARCHAR2 (1 BYTE)
TEAM_NAME_PK (TEAM_NAME)
TEAM_RANK_UK (FIFA_RANK)
TEAM_NAME_PK (TEAM_NAME)
TEAM_RANK_UK (FIFA_RANK)


SEBANU.STADIUMS
P * STADIUM_ID NUMBER (1)
* STADIUM_NAME VARCHAR2 (30 BYTE)
* LOCATION VARCHAR2 (30 BYTE)
* CAPACITY NUMBER (6)
* GAMES NUMBER (10)
STADIUM_ID_PK (STADIUM_ID)
STADIUM_ID_PK (STADIUM_ID)

Descrierea constrângerilor

- **Teams**
    - **Primary Key** (team_name) - Naționalele sunt identificate după numele lor, numele fiind unul unic. (ex. nu pot exista două echipe cu numele Argentina la mondial)
    - **Unique Key** (fifa_rank) - Locul ocupat în clasamentul fifa este unic, nu pot fi două sau mai multe echipe pe un anumit loc.
    - **Check** (pot si team_group) - Pot-ul din care face parte națională poate conține doar o valoare între 1 și 4. De asemenea, grupa trebuie să fie un caracter între A și H.
    - **NOT NULL** (toate) - Fiecare coloană trebuie să conțină o valoare, așadar nu există valori nule.
- **Players**
    - **Primary Key** (player_id) - Fiecare jucător este identificat după un id. **Modalitate de calcul a id-ului**: În aplicație, fiecărei echipe îi este asignată o valoare între 0 și 31 (să-i zicem value) după ordinea de aici, iar id-ul jucătorului este calculat după formula *value*26 + number* (26 reprezintă numărul de jucători ai fiecărei echipe, iar number reprezintă coloana următoare din tabel).
    - **Foreign Key** (team_name) - Echipa jucătorului poate fi aleasă doar din tabela **Teams**. În aplicație, când vrem să adăugăm un jucător, este folosit un **dropdown box** din care se poate alege o echipă existentă în tabela **Teams**.

    - **CHECK** (player_number și player_pos) - Numărul jucătorului poate fi doar între 1 și 26, iar poziția acestuia poate fi doar GK, DF, MF, FW.
    - **NOT NULL** (toate cu excepția a 3) - golurile pot fi nule dacă sunt 0, data nașterii poate fi necunoscută, iar jucătorul poate să nu fie la niciun club. (liber de contract)
- **Player Stats**
    - **Foreign Key / NOT NULL** (player_id) - Jucătorul poate fi ales doar din tabela **Players.**
- **Stadiums**
    - **Primary Key** (stadium_id) - Fiecare stadion este unic identificat după un id. Id-ul se calculează prin incrementare cu 1 la adăugare sau decrementare cu 1 la ștergere.
    - **NOT NULL** (toate) - Nu pot exista coloane cu valori nule, fiecare stadion are un nume, o locație, o capacitate și un număr de meciuri jucate.

Software folosit în cadrul aplicației

- IntelliJ IDEA - IDE-ul folosit pentru Java
- Scene Builder - pentru a se realiza mai ușor interfața cu utilizatorul.
- SQL Developer - pentru vizualizarea, crearea si popularea tabelelor

Baza de date în aplicație

Toate funcțiile ce țin de operațiile de bază a unei baze de date sunt conținute în clasa **Database.**

Conectarea la baza de date

Database.java

```
public static Connection getConnection(String username, String password) throws
SQLException {
   return DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/orclpdb",
username, password);
}
```

username și *password* sunt valorile introduse în caseta de login.

Obținerea datelor din tabelele bazei de date

Database.java

```
public static LinkedList<LinkedList<Object>> getData(String table) throws
SQLException {
   LinkedList<LinkedList<Object>> rows = new LinkedList<>();
   String sql = "SELECT * FROM " + table;
   Statement statement = conn.createStatement();
   ResultSet result = statement.executeQuery(sql);

   ResultSetMetaData metaData = result.getMetaData();
   int columnCount = metaData.getColumnCount();

   try {
       while (result.next()) {
           LinkedList<Object> row = new LinkedList<>();
           for (int i = 1; i <= columnCount; i++) {
               Object res = result.getObject(i);
               if (res instanceof BigDecimal) {
                   res = ((BigDecimal) res).intValue();
               } else if (res instanceof Integer) {
                   res = (Integer) res;
               }
               row.add(res);
           }
```

```
            rows.add(row);
        }
    } finally {
        result.close();
        statement.close();
    }
    return rows;
}
```

Datele sunt salvate într-o listă de liste. Listele conținute în lista părinte conțin datele de pe un singur rând.

Exemplu: `[Morocco, CAF, 22, 3, F]`

Așa ar arăta o listă conținută în lista părinte dacă s-ar extrage date din table Teams.

Modificarea și ștergerea datelor din tabele

Database.java

```
public static void modifyData(String table, String setClause, String whereClause)
throws SQLException {
    String sql = "UPDATE " + table + " SET " + setClause + " WHERE " + whereClause;
    Statement statement = conn.createStatement();
    try {
        statement.executeUpdate(sql);
    } finally {
        statement.close();
    }
}
```

Se folosește funcția *UPDATE*, iar clauzele *SET* și *WHERE* sunt introduse ca parametrii în funcție de tabela cu care se lucrează.

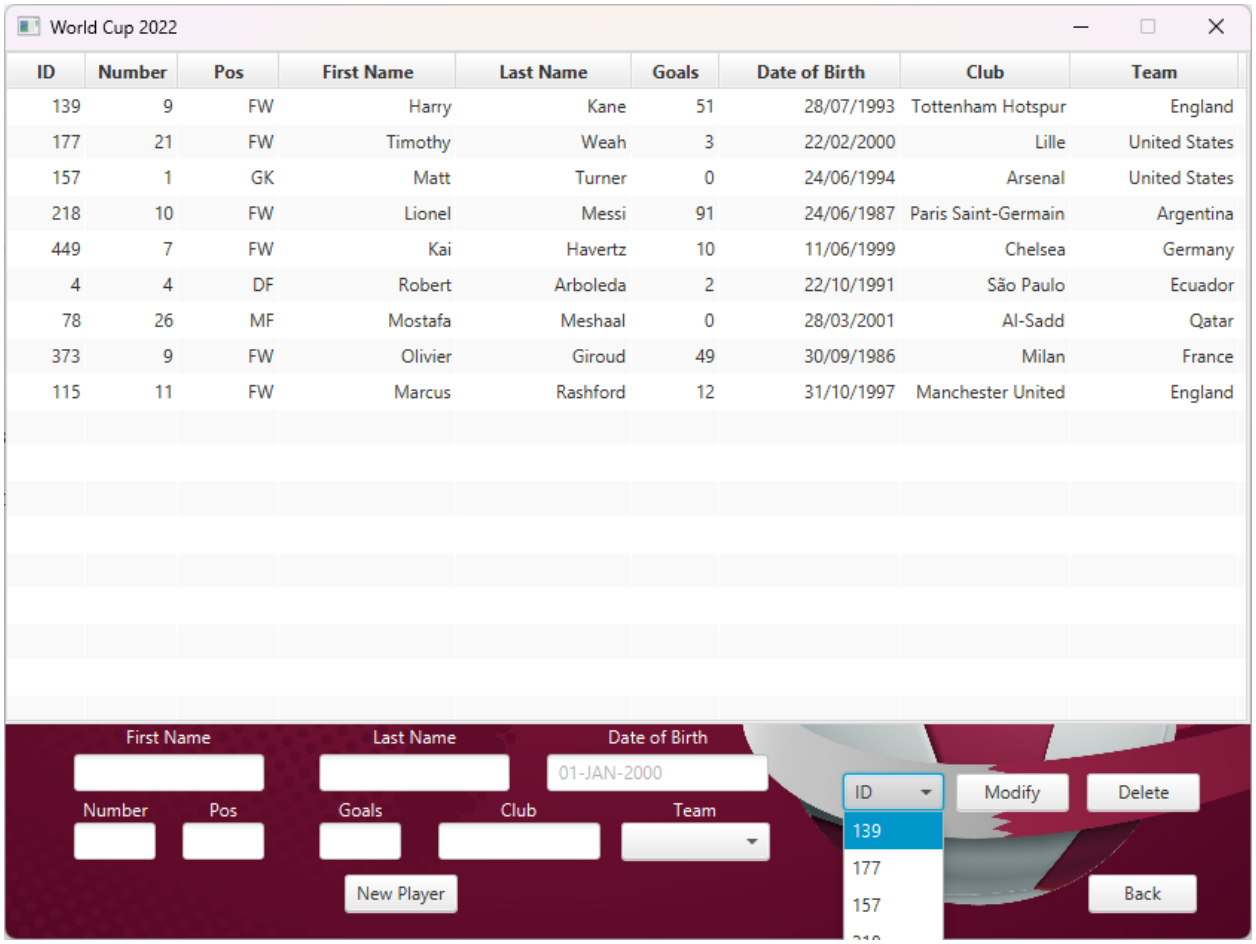
Exemplu de utilizare a funcției:

PlayerScreenController.java

```
public void modifyButton() throws SQLException, IOException {
    Integer idAsInt = deleteField.getValue();
    String id = (idAsInt != null) ? idAsInt.toString() : null;
    String number = numberField.getText();
    String pos = posField.getText();
    String firstName = firstNameField.getText();
    String lastName = lastNameField.getText();
    String goals = goalsField.getText();
    String dob = dobField.getText();
    String club = clubField.getText();
    String team = teamField.getValue();

    if (id == null || id.isEmpty())
        JOptionPane.showMessageDialog(null, "Please provide the id of the row
you want to modify!", "Error", JOptionPane.ERROR_MESSAGE);
    else {
        Database.modifyData("players", "player_id=" + id, "player_id=" + id);
        if (!number.isEmpty())
            Database.modifyData("players", "player_number=" + number,
"player_id=" + id);
        if (!pos.isEmpty())
            Database.modifyData("players", "fifa_rank=" + "'" + pos + "'",
"player_id=" + id);
        if (!firstName.isEmpty())
            Database.modifyData("players", "player_first_name=" + "'" +
firstName + "'", "player_id=" + id);
        if (!lastName.isEmpty())
            Database.modifyData("players", "player_last_name=" + "'" +
lastName + "'", "player_id=" + id);
        if (!goals.isEmpty())
            Database.modifyData("players", "player_goals=" + goals,
"player_id=" + id);
        if (!dob.isEmpty())
            Database.modifyData("players", "date_of_birth=" + "'" + dob + "'",
"player_id=" + id);
        if (!club.isEmpty())
            Database.modifyData("players", "club=" + "'" + club + "'",
"player_id=" + id);
        if (team != null && !team.isEmpty())
            Database.modifyData("players", "team=" + "'" + team + "'",
"player_id=" + id);
        tableView.setItems(getPlayers()); // update gui table
        updateGUI(); // update other gui stuff
    }
}
```

Datele sunt modificate doar dacă s-au introdus date noi în casetele fiecărei coloane, altfel acestea sunt ignorant. E obligatoriu ca doar id-ul (primary_key) să fie selectat din dropdown.



Dropdown-ul pentru ID este folosit doar pentru butoanele **Modify** și **Delete**. Pentru **Modify** se alege id-ul rândului care trebuie modificat și se inserează în coloanele din stânga date acolo se dorește modificarea, și se apasă pe butonul **Modify**. Pentru **Delete** se alege id-ul, după care se apasă butonul **Delete**.

Ștergerea datelor din tabele

Database.java

```
public static void deleteData(String table, String whereClause) {
   try {
       String sql = "DELETE FROM " + table + " WHERE " + whereClause;
       Statement statement = conn.createStatement();
       statement.executeUpdate(sql);
       statement.close();
   } catch(SQLException sqlException) {
       JOptionPane.showMessageDialog(null, "Could not delete the row. ID is being
used in another table!", "Error", JOptionPane.ERROR_MESSAGE);
   }
}
```

Se folosește funcția *DELETE*, iar clauza *WHERE* este dată ca parametrul.

Exemplu de utilizare a funcției:

PlayerScreenController.java

```
public void deleteButton() throws SQLException, IOException {
   Integer deleteText = deleteField.getValue();
   if (deleteText.equals(null))
       JOptionPane.showMessageDialog(null, "Please provide the id of the row you
want to delete!", "Error", JOptionPane.ERROR_MESSAGE);
   else {
       Database.deleteData("players", "player_id=" + deleteText); // delete player
stats by id
       tableView.setItems(getPlayers()); // update gui table
       updateGUI(); // update other gui stuff
   }
}
```

Se șterge jucătorul după id, iar apoi este actualizată interfața.

Adăugarea date noi în tabele

Database.java

```
public static void addData(String table, String... objects) throws SQLException {
   String sql = "INSERT INTO " + table + " VALUES(" + StringUtils.repeat("?", ",",
objects.length) + ")";
   PreparedStatement statement = conn.prepareStatement(sql);
   try {
       for (int i = 0; i < objects.length; i++) {
           statement.setString(i + 1, objects[i]);
       }
       statement.executeUpdate();
   } finally {
       statement.close();
   }
}
```

Se folosește funcția *INSERT* pentru a se insera date în tabelă. Ca parametrii este dată tabela în care se dorește introducerea datelor și un număr variat de obiecte ce urmează a fi introduse, în funcție de numărul de coloane a tabelei.

Exemplu utilizare funcție:

PlayerScreenController.java

```
public void newTeamButton() throws SQLException, IOException {
   String number = numberField.getText();
   String pos = posField.getText();
   String firstName = firstNameField.getText();
   String lastName = lastNameField.getText();
   String goals = goalsField.getText();
   String dob = dobField.getText();
   String club = clubField.getText();
   String team = teamField.getValue();
```

```
   int id = teamsById.get(team) * 26 + Integer.parseInt(number);

   if (number.isEmpty() || pos.isEmpty() || firstName.isEmpty() ||
lastName.isEmpty() || goals.isEmpty() || dob.isEmpty() || club.isEmpty() ||
team.isEmpty())
       JOptionPane.showMessageDialog(null, "All fields are required!", "Error",
JOptionPane.ERROR_MESSAGE);
   else {
       Database.addData("players", Integer.toString(id), number, pos, firstName,
lastName, goals, dob, club, team); // add values to database table
       tableView.setItems(getPlayers()); // update gui table
       updateGUI(); // update other gui stuff
   }
}
```

Un mic dezavantaj al acestui cod este că fiecare casetă de text trebuie umplută pentru a se adăuga valori în tabelă. Din această cauză nu pot fi introduce date nule.

Exemplu adăugare pe interfață: