UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAŞI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DOMENIUL: Calculatoare şi tehnologia informaţiei
SPECIALIZAREA: Tehnologia informaţiei

Aplicație gestionare ligi de fotbal

Proiect la disciplina
Baze de date – temă (BD)

Profesor coordonator
Sorin Avram

Student
Mircea-Constantin Dobreanu
Grupa 1307B

Iaşi, 2018

Cuprins

Introducere

Mi-am propus pentru tema la disciplina Baze de date să realizez o aplicație pentru gestionarea ligilor de fotbal (implicit și a clasamentelor). Am numit-o microbe (apropiat de microbist în română). Pentru realizarea proiectului m-am folosit de următoarele limbaje/unelte/biblioteci:

- Java 8u171,
- Eclipse,
- Maven (pentru configurarea proiectului),
- Git,
- Tomcat 9.0.6 (server-ul),
- Jersey (implementare server JAX-RS),
- HTML, CSS, JavaScript cu biblioteca jQuery pe partea de front-end,
- MariaDB v10.1.29,
- SQL.

Capitolul 1. Descrierea generală a aplicațiilor și a deciziilor luate

Am luat decizia de a folosi chei primare auto generate de către MariaDB pentru a îmi ușura munca pe partea de back-end; astfel, toate entitățile au chei primare numere întregi auto-generate.

Aplicația se folosește de sistemul de gestionare de baze de date MariaDB. În baza de date microbe sunt definit 4 tabele care ar putea și ar trebui să fie extinse într-o aplicație menită spre a fi folosită la scară largă sau comercializată. Am ales să nu adaug informații relevante dar care nu sunt critice. De aceea tabelele coaches (antrenori) și leagues (ligi) sunt doar perechi de chei și nume.

În continuare voi atașa codul DDL (vezi și Figura 1.1) pentru crearea tabelelor (care poate fi găsit în fișierul database_scripts/create_tables.sql, care mai conține și inserările):

```
CREATE DATABASE IF NOT EXISTS microbe;
USE microbe;

CREATE TABLE IF NOT EXISTS leagues (
 leagueID INT UNSIGNED AUTO_INCREMENT,
 leagueName VARCHAR(50) NOT NULL,

 CONSTRAINT pk_league PRIMARY KEY (leagueID),
 CONSTRAINT uk_league UNIQUE KEY (leagueName)
);

CREATE TABLE IF NOT EXISTS coaches (
 coachID INT UNSIGNED AUTO_INCREMENT,
 coachName VARCHAR(50) NOT NULL,

 CONSTRAINT pk_coach PRIMARY KEY (coachID)
);

CREATE TABLE IF NOT EXISTS teams (
 teamID INT UNSIGNED AUTO_INCREMENT,
 teamName VARCHAR(50) NOT NULL,
 leagueID INT UNSIGNED NOT NULL,
 coachID INT UNSIGNED,

 CONSTRAINT pk_team PRIMARY KEY (teamID),
 CONSTRAINT uk_team UNIQUE KEY (teamName),
 CONSTRAINT uk_coach_team UNIQUE KEY(coachID),
 CONSTRAINT fk_league_team FOREIGN KEY (leagueID) REFERENCES
leagues(leagueID),
 CONSTRAINT fk_coach_team FOREIGN KEY (coachID) REFERENCES
coaches(coachID)
);

CREATE TABLE IF NOT EXISTS matches (
 matchID  INT UNSIGNED AUTO_INCREMENT,
 homeTeamID INT UNSIGNED NOT NULL,
 awayTeamID INT UNSIGNED NOT NULL,
```

```
  homeGoals INT UNSIGNED NOT NULL DEFAULT 0,
  awayGoals INT UNSIGNED NOT NULL DEFAULT 0,
  playDate DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP(),

  CONSTRAINT pk_match PRIMARY KEY (matchID),
  CONSTRAINT uk_match UNIQUE KEY (homeTeamID, awayTeamID, playDate),
  CONSTRAINT ck_match CHECK (homeTeamID != awayTeamID),
  CONSTRAINT fk_homeTeam_match FOREIGN KEY(homeTeamID) REFERENCES
teams(teamID),
  CONSTRAINT fk_awayTeam_match FOREIGN KEY(awayTeamID) REFERENCES
teams(teamID)
);
```

Cheile străine asigură integritatea datelor și le consider în mare parte evidente (marea majoritate a lor au și condiția de a fi diferite de nul, cu excepția câmpului coachID din tabela teams, pentru că ar putea exista situația în care un club de fotbal tocmai a dat afară un antrenor).

Cheile unice stabilesc că obiectele/câmpurile sunt unice în spațiul problemei:

- două echipe nu pot avea același antrenor.
- două echipe nu pot disputa mai multe meciuri în aceeași zi calendaristică (dacă aș fi pus cheie unică doar pe perechea id-urilor echipelor ar fi însemnat că două echipe nu pot juca una împotriva celeilalte decât o singură dată).

De asemenea, am folosit o constrângere de tip check pentru a verifica că echipele sunt distinctie (nu are sens ca o echipă să joace împotriva sa într-o ligă; meciurile de antrenament nu au ce căuta în spațiul problemei).

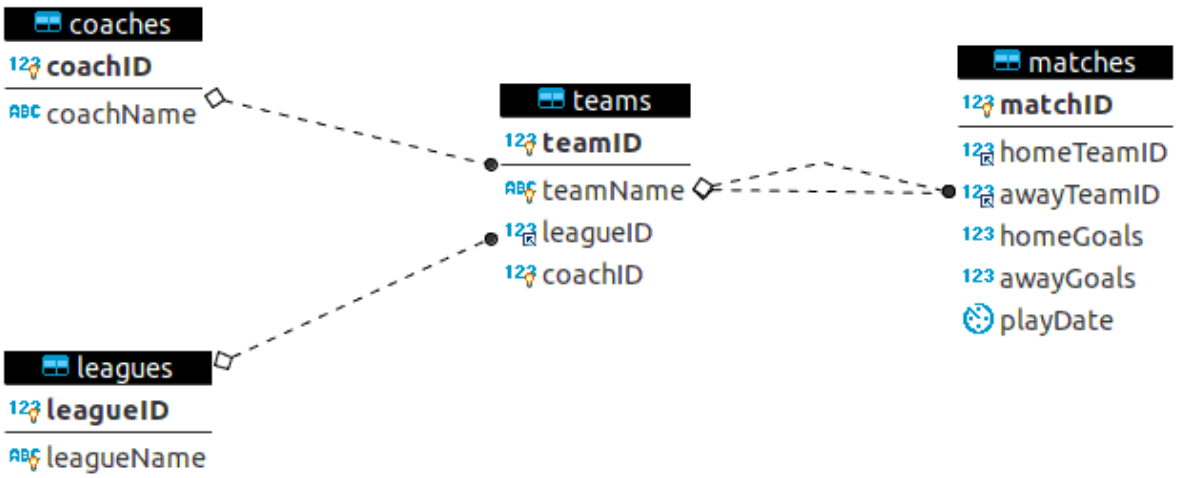


Figura 1.1. Diagrama ER a soluției propuse

1.1.1. Conectarea la baza de date

După cum am amintit și mai sus, m-am folosit de SGBD-ul MariaDB, un fork al popularului MySQL. Pentru conectare am folosit un driver JDBC (Java Database Connector) special conceput pt SGBD-ul ales. Fiecare obiect (echipă, ligă, antrenor, meci) este creat/adaugat/editat/șters prin intermediul secvențelor de cod SQL. Pe back-end am implementat un server de servicii web (folosind modelul REST = Representantional State Transfer, care reprezintă apropae perfect operațiunile enumerate mai sus). Fiecare obiect are atașat (în mod logic) o clasă DAO (Database access object) care realizează design pattern-ul singleton și a cărei

responsabilitate este gestionarea tuturor operațiilor legate de acel obiect. În constructorul clasei abstracte AbstractDatabaseObject (părinte a tuturor DAO-urilor menționate mai sus) este realizată conectarea efectivă la baza de date:

```
protected AbstractDatabaseAccessObject() {
    try {

          String rootPath = Thread.currentThread()
                      .getContextClassLoader().getResource("").getPath();


          Properties dbProps = new Properties();
          dbProps.load(new FileInputStream(rootPath + "db.properties"));

          sql = new Properties();
          sql.load(new FileInputStream(rootPath + "queries.properties"));

          Class.forName(dbProps.getProperty("jdbcDriver"));
          conn = DriverManager.getConnection(dbProps.getProperty("url"),
                                    dbProps.getProperty("username"),
                                    dbProps.getProperty("password"));
    } catch (Exception e) {
          e.printStackTrace();
    }
 }
```

După cum se poate observa, m-am folosit de fișiere de proprietăți pentru a nu trebui recompila codul dacă aleg să fac o schimbare a codului SQL sau să schimb SGBD-ul. Fișierul db.properties conține informații legate despre SGBD:

- cum se cheamă baza de date?
- cum se realizează conexiunea? (driver-ul)
- care sunt datele de logare în SGBD?

Fișierul queries.properties conține toate declarațiile pregătite (din engleză prepared statements), adică acele secvențe de cod SQL care sunt parametrizabile. În continuare, voi arăta codul pentru o resursă din soluția propusă, anume cea pentru tabela matches:

```
## Matches
getMatchesByTeam      =     SELECT * \
                            FROM matches \
                            WHERE homeTeamID = ? OR awayTeamID = ?;

getMatch              =     SELECT * \
                            FROM matches \
                            WHERE matchID = ?;

createMatch           =     INSERT INTO matches \
                            VALUES(default, ?, ?, ?, ?, ?);

replaceMatch          =     UPDATE matches \
                            SET homeTeamID = ?, awayTeamID = ?, homeGoals
= ?, awayGoals = ?, playDate = ?;
```

```
deleteMatch             =     DELETE FROM matches \
                              WHERE matchID = ?;
```

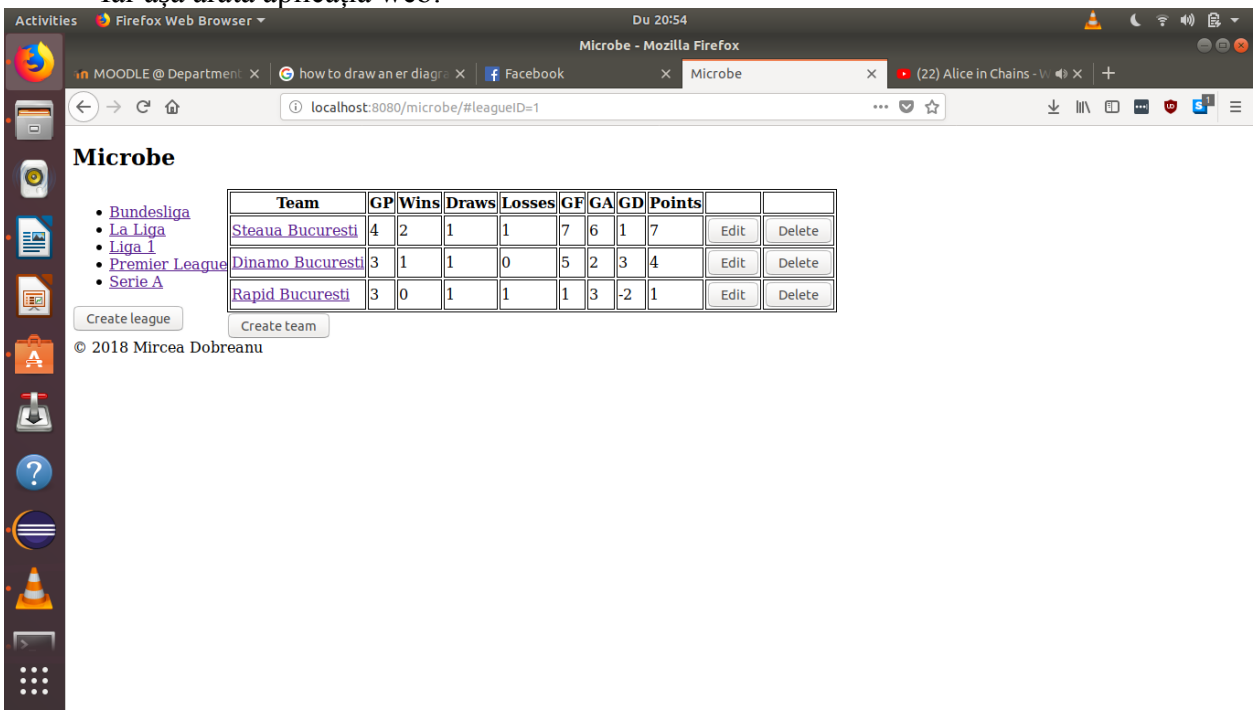
Caracterele \ sunt doar pentru a semnala că șirul de caractere continuă pe următoarea linie, iar caracterele ? reprezintă parametrii declarației SQL. Toate resursele urmează în mare parte cam același tipar evidențiat mai sus. Iar în continuare avem codul sursă Java care se folosește de operațiile SQL de mai sus pentru a implementa opeațiile de vizualizare:

```
public class MatchDAO extends AbstractDatabaseAccessObject {
 private static final MatchDAO instance = new MatchDAO();

 public static MatchDAO getInstance() {
    return instance;
 }

 private MatchDAO() {
    super();
 }

 public List<Match> getMatches(int teamID) {
    List<Match> matches = new ArrayList<>();
    try {
          statement =
conn.prepareStatement(sql.getProperty("getMatchesByTeam"));
          statement.setInt(1, teamID);
          statement.setInt(2, teamID);
          ResultSet rs = statement.executeQuery();
          while (rs.next()) {
                Match m = copyOf(rs).build();
                matches.add(m);
          }
    } catch (SQLException e) {
          e.printStackTrace();
    }
    return matches;
 }

 public Match getMatch(int matchID) {
    Match m = null;
    try (ResultSet rs = getResourceById("getMatch", matchID)) {
          if (rs.next())
                m = copyOf(rs).build();

    } catch (SQLException e) {
          e.printStackTrace();
    }
    return m;
 }
 ...
```

Mai apoi codul este expus ca un serviciu web. Am optat pentru o astfel de variantă pentru că nu contează tehnologia în care este implementat clientul (nu este neapărat să fie implementat tot în Java) fiindcă comportamentul serviciilor web este în mare parte standard. Mai jos atașez codul care face posibilă consumarea serviciului web (în codul ce urmează este prezentat doar partea de vizualizare, restul aflându-se în fișierul sursă).

```
@Path("matches")
public class MatchResource extends AbstractResource {
 @GET
 @Path("query")
 @Produces(MediaType.APPLICATION_JSON)
 public List<Match> getMatches(@QueryParam("teamID") int teamID) {
    return MatchDAO.getInstance().getMatches(teamID);
 }

 @GET
 @Path("{matchID}")
 @Produces(MediaType.APPLICATION_JSON)
 public Match getMatch(@PathParam("matchID") int matchID) {
    return MatchDAO.getInstance().getMatch(matchID);
 }
...
```

Iar așa arată aplicația web:



Bibliografie