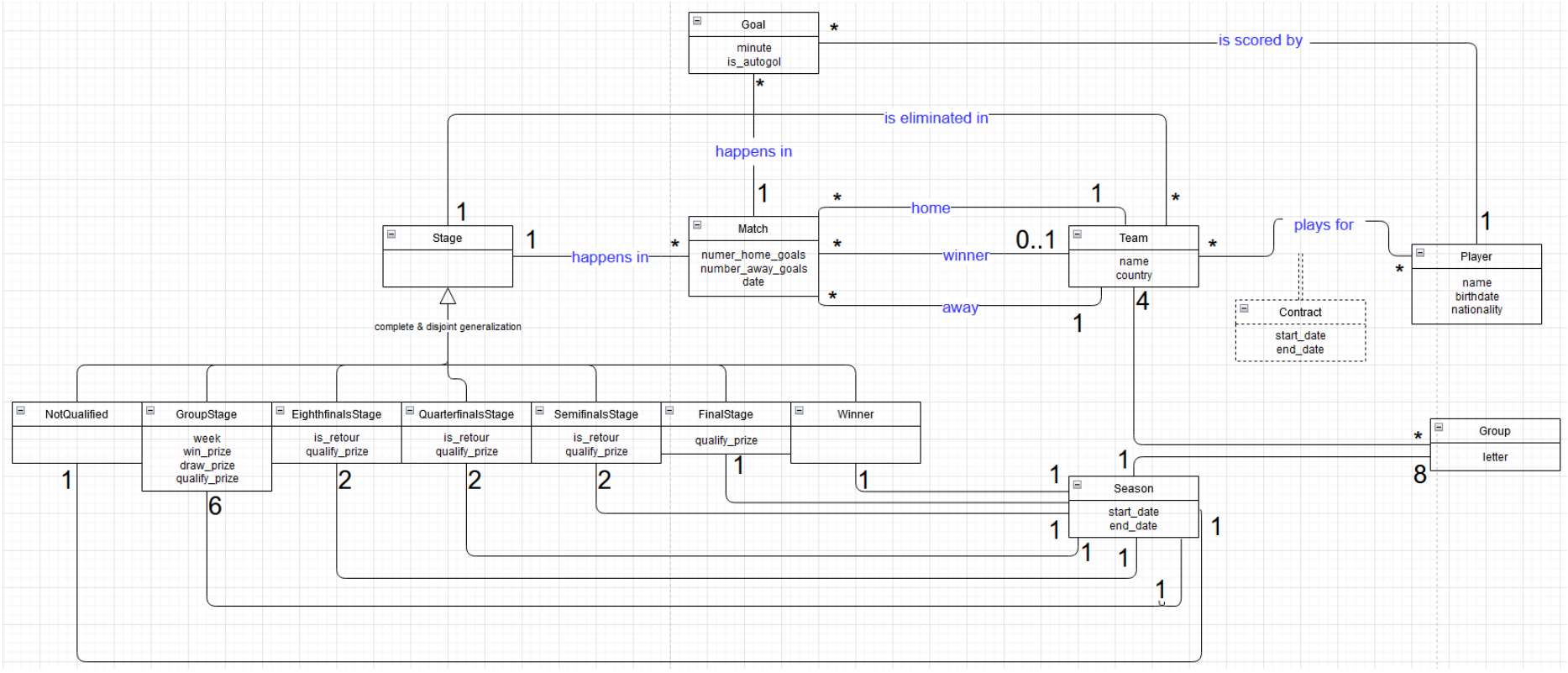
Champions League Results Database
2022 - 2023

Students from Class 2LEIC13:
Raducanu Alexandru - up202202991
Serban Emilia-Bianca - up202202992
Gabriela Dias Salazar Neto Silva - up202004443

Summary

A. Definition of Conceptual Model



Description of the project

The main objective of our project was to develop a database that can manage the results of **Champions League Tournament** across multiple seasons, from the group stage to the grand final. It is also required to keep track of the goal scorers and of the financial prizes that each team deserves.

Based on the statement given by the teachers, we started to design the **Conceptual Mode**l of our Database. We created our main classes: **Player, Team, Match, Season, Group, Goal and Stage**, and their attributes.

Most of the associations and multiplicities presented on the Conceptual Model are easily-understandable (for example: a player can score multiple goals, but a goal is scored by 1 and only 1 player), but there are some restrictions that helped us keep the model as close to the reality as possible:

1. A group must have exactly 4 teams. A team can be in multiple groups across the seasons.
2. A season has exactly 8 groups, with letters from ‘A’ to ‘H’. An instance of the Group class can only be linked with 1 season.
3. A season is linked with: 6 GroupStage instances, 2 EighthFinals instances, 2 QuarteFinals instances, 2 Semifinals instances and 1 FinalStage instance. A season also has 1 NotQualified instance (for the teams in our database that did not qualify in the groups in a specific season) and 1 Winner instance (to clearly differentiate the winner of the competition)
4. A Stage instance can have multiple Matches (for example, one GroupStage instance must be linked to exactly 16 matches). A match happens in 1 stage.
5. The particular stages are grouped together into a Stage class using the generalisation. This generalisation is complete and disjoint, because an instance of the Stage class must be an instance of 1 and only 1 subclass.
6. A team is eliminated in exactly 1 stage per season (Winner stage if the team won the tournament, NotQualified if the team did not qualified to the group stage in a specific year, the others are intuitive)
7. A match has exactly 1 home team and 1 away team. A match can have a winner or not (if it ends in a draw).
8. The association between Player and Team is a qualified association, as we need to have a contract between the player and team. Because our database manages the Champions League across multiple seasons, we need to keep track of the start date and end date of every contract.
9. The financial prizes are given for each win and draw in the group stage, as well as for qualifying to further stages. In our database, this feature is implemented through the attributes of specific stages.

B. Definition of Relational Schema

a) Relational Model

- Player (idPlayer , name, birthdate, nationality)
- Team (idTeam, name, country)
- Contract (idPlayer → Player, idTeam → Team, start_date, end_date)
- Goal (minute, idPlayer → Player, idMatch → Match, is_autogol)
- Match (idMatch, date, number_home_goals, number_away_goals, { idHome, idAway, idWinner} → Team, idStage → Stage)
- Elimination (idTeam → Team, idSeason → Season, idStage → Stage)
- Group (idGroup, letter , idSeason → Season)
- GroupTeam (idGroup → Group , idTeam → Team)
- Season (idSeason , start_date, end_date)
- Stage (idStage, type, week, idSeason → Season , qualify_prize, win_prize, draw_prize, is_retour)

The **primary key** for every relationship is the underlined attribute(s). The **foreign keys** are specified in every relation by the " → " character.

Notice that for the generalisation in the Conceptual Model (at the Stage class), we used the **"Use nulls" approach**, as we believe it fits the best with our problem. Even if it is recommended to make the "Object-Oriented" approach on complete and disjoint generalisation (like the one we are discussing about), it would add 5 more attributes to each relation that has the "idStage → Stage" foreign key.

This is because we would have to add "idGroupStage → GroupStage" , "idFinal → Final", "idSemifinal → Semifinal", "idQuarterfinal → Quarterfinal" and "idEigthfinal → EigthFinal" as foreign keys to replace the "idStage → Stage" foreign key. Also, 4 out of 5 foreign keys for each instance would have the value "null". That would not be efficient in terms of memory usage.

b) Restrictions of the Relational Model

As we wish to have a database that manages the result of the Champions League competition as realistic as possible, we have to introduce a couple of restrictions to our model. The restrictions are:

1) The teams that are part of a group have to be from different countries, as this is the rule in UEFA Champions League. This way, there will not be any match between 2 teams from the same country in the Group Stage.
2) If a match is in the “Eighth-finals Stage”, the opponents cannot be from the same country. This restriction is not applicable to Quarter-Finals, Semi-Finals and the Final (in these stages, two teams from the same country can meet in a match).
3) The prizes for a season of Champions League are decided at the beginning of the season. There are **fixed** amounts of money paid to the clubs for:
    a) Winning a match in the group stage
    b) Drawing a match in a group stage
    c) Qualifying from the group stage to the Round of 16 (Eighth of Finals)
    d) Qualifying in the Quarter Finals
    e) Qualifying in the Semi-Finals
    f) Qualifying to the Final
    g) Winning the Final

    Based on the matches and on the round a team has been eliminated in, we can come up with a formula to compute what amount of money the team deserves for that season.
4) A given season must have exactly 8 groups, each group having a different letter from A to H
5) A given season must have exactly one Final, two Semifinals, two QuarterFinals, two Eighth-Finals (only one of each type with the field “is_retour” activated) and exactly 6 GroupStage instances. These 6 GroupStages must have a different “week” (numbered from 1 to 6) and the same amount of money as prizes.
6) A team must be eliminated from the competition once and only once each season. An elimination can occur in a stage only if the “week” is 6 , “is_retour” is activated or it is the Final Stage (, NotQualified stage or Winner stage).
7) If a match ends in a draw, the “idWinner” field must be NULL
8) The “minute” field of a goal can have values between 1 and 120

9) The GroupTeam relation uniquely identifies a link between a Team and its Group. Thus, there must be exactly 4 instances of GroupTeam that share the same “idGroup → Group” attribute.

C. Functional Dependencies and Normal Form Analysis

Now, we will analyse the Functional Dependencies and the Normal Form of each relation from our model.

- Player (idPlayer , name, birthdate, nationality)
  - idPlayer → name, birthdate, nationality

  This relation is in the Boyce-Codd Normal Form, as the only non-trivial Functional Dependency has a superkey on the left side.
  The relation is in the 3rd Normal Form, as the only non-trivial Functional Dependency has a key on the left side.

- Team (idTeam, name, country)
  - idTeam → name, country
  - name, country → idTeam

  This relation is in the Boyce-Codd Normal Form. All the non-trivial FDs have on the left side a superkey.
  This relation is in the 3rd Normal Form. All non-trivial FDs have on the left side a key ({name, country} and {idTeam} are candidate keys).

- Contract (idPlayer → Player, idTeam → Team, start_date, end_date)
  - idPlayer, idTeam, start_date → end_date
  - idPlayer, idTeam, end_date → start_date
  - idPlayer, start_date, end_date → idTeam

  This relation is in the Boyce-Codd Normal Form. Each non-trivial FD has on the left side a superkey (candidate keys).
  This relation is in the 3rd Normal Form thanks to the same reason.

- Goal (minute, idPlayer → Player, idMatch → Match, is_autogol)
  - minute , idPlayer, idMatch → is_autogol

This relation is in the Boyce-Codd Normal Form, as the only non-trivial Functional Dependency has a superkey on the left side.

This relation is in the 3rd Normal Form as the only non-trivial FD has a key on the left side.

- Elimination (idTeam → Team, idSeason → Season, idStage → Stage)
  - idTeam, idSeason -> idStage

  This relation is in the Boyce-Codd Normal Form, as the only non-trivial Functional Dependency has a superkey on the left side.

  The relation is in the 3rd Normal Form thanks to the same reason.

- Group (idGroup, letter , idSeason → Season)
  - idGroup → letter, idSeason
  - idSeason , letter → idGroup

  This relation is in the Boyce-Codd Normal Form and in the 3rdNormal Form.

- GroupTeam (idGroup → Group , idTeam → Team)

  This relation is in the Boyce-Codd Normal Form and in the 3rdNormal Form.

- Season (idSeason , start_date, end_date)
  - idSeason → start_date, end_date
  - Start_date, end_date → idSeason

  This relation is in the Boyce-Codd Normal Form and in the 3rd Normal Form, as the left side of each non-trivial Functional Dependency is a superkey.

- Stage (idStage, type, week, idSeason → Season , qualify_prize, win_prize, draw_prize, is_retour)
  - idStage → type , week , idSeason , qualify_prize, win_prize, draw_prize, is_retour

    This relation is in the Boyce-Codd Normal Form and in the 3rd Normal Form, because the only non-trivial FD has a key on the left side.

- Match (idMatch, date,number_home_goals, number_away_goals, { idHome, idAway, idWinner} → Team, idStage → Stage)

- idMatch → number_home_goals, number_away_goals, idHome, idAway, idWinner, idStage, date
- idHome, idAway , number_away_goals, number_home_goals → idWinner

This relation is **NOT** in the Boyce-Codd Normal Form, because the left side of the second FD is not a superkey. Following the Boyce-Codd Normal Form Decomposition Algorithm, we get 2 relations that both satisfy the BCNF and 3NF. These relations are:

- Match1 ({idHome, idAway, idWinner} → Team , number_home_goals, number_away_goals)
- Match2 (idMatch, number_home_goals, number_away_goals, { idHome, idAway} → Team, idStage → Stage , date)

We will not use this decomposition in our database, as it makes everything a lot more complex. We will operate with the initial relation.

D & E. Creating the database in SQLite and Adding constraints

- **Deleting and creating a new database**

  In order to allow the user to delete already existing data in the database and to create an empty database, we made sure to delete all existing tables:

```
 6  DROP TABLE IF EXISTS GroupTeam;
 7  DROP TABLE IF EXISTS [Group];
 8  DROP TABLE IF EXISTS Elimination;
 9  DROP TABLE IF EXISTS Goal;
10  DROP TABLE IF EXISTS [Match];
11  DROP TABLE IF EXISTS Stage;
12  DROP TABLE IF EXISTS Contract;
13  DROP TABLE IF EXISTS Team;
14  DROP TABLE IF EXISTS Player;
15  DROP TABLE IF EXISTS Season;
```

  Important note: The deletion takes place in a reversed order considering the creation of the tables. The reason is that if Table1 would have a foreign key dependent on Table2, we would not delete Table2 first, leaving Table1 with a foreign key depending on “nothing”, but we would delete first Table1 and after that, Table2.

```
17 › CREATE TABLE Season( ⋯
23 › CREATE TABLE Player( ⋯
29 › CREATE TABLE Team( ⋯
34 › CREATE TABLE Contract( ⋯
42 › CREATE TABLE Stage( ⋯
53 › CREATE TABLE [Match]( ⋯
65 › CREATE TABLE Goal( ⋯
73 › CREATE TABLE Elimination( ⋯
81 › CREATE TABLE [Group]( ⋯
87 › CREATE TABLE GroupTeam (⋯
```

- **Creating tables and adding constraints**

  We firstly created the tables Season, Player and Team because they do not have any foreign keys.

  Note: All ids which are Primary Keys have unique values, therefore there cannot exist two instances of the same relation with the same id.

Season

```
CREATE TABLE Season(
    idSeason INTEGER PRIMARY KEY,
    startDate DATE UNIQUE NOT NULL,
    endDate DATE UNIQUE NOT NULL,
    CONSTRAINT Season_CHK1_Date CHECK (endDate >= startDate));
```

Constraints:

- **Primary Key (KEY Constraint)**: idSeason. We chose this key to be the primary key because it is easier working with IDs than with Dates.
- **Unique/Candidate Keys (KEY and NOT NULL Constraints):** {start_date} , {end_date}. When creating a Season, the start_date and the end_date cannot be null.
- **CHECK Constraint**: the “endDate” attribute should be bigger than the “startDate” attribute. In other words, the Season cannot end before it begins.

Player

```
CREATE TABLE Player(
    idPlayer INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    birthdate DATE,
    nationality TEXT NOT NULL);
```

Constraints:

- **Primary Key (KEY Constraint):** idPlayer. We cannot uniquely identify a Player only by its name, birthdate and nationality, as there can be multiple players with the exact same values for these attributes.
- **NOT NULL Constraint:** a Player must have a name and a nationality, they cannot be null.

Team

```
CREATE TABLE Team(
    idTeam INTEGER PRIMARY KEY,
    name TEXT UNIQUE NOT NULL,
    country TEXT);
```

Constraints:

- **Primary Key (KEY Constraint)**: idTeam. We chose this key to be the primary key because it is easier working with IDs than with Text.
- **Unique/ Candidate Key (KEY and NOT NULL Constraints):** {name}. A team is uniquely identified by its name.

Next, we created the tables Contract, Stage, Match, Goal, Elimination and Group because they contain foreign keys that depend on the first 3 tables created. Also, Elimination has a foreign key which depends on the Stage table, so this order of creation is correct.

Note: We added extra parentheses to the tables "[Match]" and "[Group]" because "Match" and "Group" are reserved words in SQLite.

Contract

```
CREATE TABLE Contract(
    idPlayer INTEGER REFERENCES Player(idPlayer),
    idTeam INTEGER REFERENCES Team(idTeam),
    startDate DATE,
    endDate DATE,
    CONSTRAINT Contract_PK PRIMARY KEY (idPlayer, idTeam, startDate),
    CONSTRAINT Contract_CHK1_sDeD CHECK (endDate >= startDate));
```

Constraints:

- **Composite Primary Key (KEY Constraint):** {idPlayer, idTeam, startDate}
- **Foreign Keys (Extern Key Constraint):** idPlayer, idTeam
- **CHECK Constraint:** the "endDate" attribute should be bigger than the "startDate" attribute. In other words, the Contract cannot end before it begins.

Stage

```
CREATE TABLE Stage(
    idStage INTEGER PRIMARY KEY,
    typee TEXT NOT NULL,
    week INTEGER,
    idSeason INTEGER,
    qualifyPrize INTEGER,
    winPrize INTEGER,
    drawPrize INTEGER,
    isRetour BOOLEAN,
    CONSTRAINT Stage_FK1 FOREIGN KEY (idSeason) REFERENCES Season(idSeason));
```

Constraints:

- **Primary Key (KEY Constraint):** idStage
- **NOT NULL Constraint: "**typee" attribute cannot be null. In other words, the stage type must be one of the following: "GroupStage", "EighthfinalsStage", "QuarterfinalsStage", "SemiFinals", "Final", "Winner", "NotQualified"
- **Foreign Key (EXTERNAL KEY Constraint):** idSeason

Match

```
CREATE TABLE [Match](
    idMatch INTEGER PRIMARY KEY,
    date DATE NOT NULL,
    numberGoalsHome INTEGER NOT NULL,
    numberGoalsAway INTEGER NOT NULL,
    idHome INTEGER NOT NULL,
    idAway INTEGER NOT NULL,
    idWinner INTEGER,
    idStage INTEGER,
    CONSTRAINT Match_Team_FK1 FOREIGN KEY (idHome) REFERENCES Team(idTeam),
    CONSTRAINT Match_Team_FK2 FOREIGN KEY (idAway) REFERENCES Team(idTeam),
    CONSTRAINT Match_Team_FK3 FOREIGN KEY (idWinner) REFERENCES Team(idTeam),
    CONSTRAINT Match_Team_FK4 FOREIGN KEY (idStage) REFERENCES Stage(idStage));
```

Constraints:

- **Primary Key (KEY Constraint):** idMatch
- **NOT NULL Constraint:** "date"**, "**numberGoalsHome", **"**numberGoalsAway", "idHome", "idAway" attributes cannot be null. Even if the number of goals scored by the Home/Away team is 0, the value of **"**numberGoalsHome" / **"**numberGoalsAway" must be 0. "idHome" and "idAway" cannot be null as there must be 2 Teams that play a match. "idWinner"

can be null because there is a chance that there will not be any winner (draw).

- **Foreign Key (EXTERNAL KEY Constraint):** idHome, idAway, idWinner

Goal

```
CREATE TABLE Goal(
    minuteTime INTEGER,
    idPlayer INTEGER REFERENCES Player(idPlayer),
    idMatch INTEGER REFERENCES Match(idMatch),
    isAg BOOLEAN,
    CONSTRAINT Goal_PK PRIMARY KEY (minuteTime, idPlayer, idMatch),
    CONSTRAINT Goal_CHK1_m CHECK (minuteTime >= 0 AND minuteTime <= 120));
```

Constraints:

- **Composite Primary Key (KEY Constraint):** {minuteTime, idPlayer, idMatch}. Only this group of attributes can uniquely identify a goal instance.
- **Foreign Key (EXTERNAL KEY Constraint):** idPlayer, idMatch, idWinner
- **CHECK Constraint:** the value of “minuteTime” must always range between 0 and 120 because a match must be minimum 90 minutes and maximum 120 minutes long. Therefore, a goal can be scored since the beginning of the game, until the end of it.

Elimination

```
CREATE TABLE Elimination(
    idTeam INTEGER REFERENCES Team(idTeam),
    idSeason INTEGER REFERENCES Season(idSeason),
    idStage INTEGER NOT NULL,
    CONSTRAINT Elimination_PK PRIMARY KEY (idTeam, idSeason),
    CONSTRAINT Elimination_FK FOREIGN KEY (idStage) REFERENCES Stage(idStage)
);
```

Constraints:

- **Composite Primary Key (KEY Constraint):** {idTeam, idSeason}. A team is eliminated from the competition exactly once per season.
- **Foreign Key (EXTERNAL KEY Constraint):** idTeam, idSeason, idStage
- **NOT NULL Constraint:** idStage must not be null because each team will be eliminated once and only once every season.

Group

```
CREATE TABLE [Group](
    idGroup INTEGER,
    letter CHAR,
    idSeason INTEGER REFERENCES Season(idSeason),
    CONSTRAINT Group_PK PRIMARY KEY (idGroup));
```

Constraints:

- **Primary Key (KEY Constraint):** idGroup. The id uniquely identifies a Group instance. This is the easiest way to identify a Group instance, regardless of the season or letter.
- **Composite Candidate Key:** {letter, idSeason → Season} can also uniquely identify a Group instance. Our PK makes things a lot easier, on the other hand.

GroupTeam

```
CREATE TABLE GroupTeam (
    idGroup INTEGER REFERENCES [Group] (idGroup) ,
    idTeam INTEGER REFERENCES Team (idTeam),
    CONSTRAINT GroupTeam_PK PRIMARY KEY (idGroup , idTeam));
```

Constraints:

- **Composite Primary Key (KEY Constraint):** idGroup, idTeam. This composite key uniquely identifies a GroupTeam instance.

Testing our .sql files

After creating the files “criar.sql” and “povoar.sql”, we tested them on a Linux Operating System with the “sqlite3” tool installed.

On a command line, being located in the same directory as the 2 .sql files, we executed the next commands:

sqlite3 Test.db “.read criar.sql”
sqlite3 Test.db “.read povoar.sql”
sqlite3 Test.db

Now, we are interacting with our .db file. Here, we can test some simple queries.
Based on our method of testing our work, we did not find any errors or anomalies.

Participation of the group elements

All 3 of us worked together to develop the database that we have created. The tasks were divided equally regarding the workload and difficulty. We had the chance to learn from each other and to ask questions when we had any doubts.

The division of the tasks between the students of our group was as follows:
Task A - Emilia-Bianca, Gabriela, Alexandru
Task B - Emilia-Bianca, Gabriela, Alexandru
Task C - Emilia-Bianca, Alexandru
Task D - Gabriela
Task E - Gabriela, Emilia-Bianca, Alexandru
Task F - Emilia-Bianca, Alexandru

G. Database Queries

For this task, we thought about 10 natural questions someone would have about a Champion's League season. Then, we converted those natural questions in queries and created the 10 int.sql files, as requested.

These are the 10 queries we created for our database. We mention that the order of the queries described here in "natural language" is the same as the "int.sql" files. In each "sql" file, there is a description of the query and the SQL code that makes the query work.

1. What is the amount of money received by the winner of the Final of Champion's League season 2022-2023?
   List a single column named "prize (€)" with a single row. The single cell should contain the requested amount of money.

2. List all the Portuguese teams that are involved in the 2022-2023 Champion's League tournament.
   Order them by name.

3. List all the countries that have teams involved in the group stage of Champion's League 2022-2023 edition, and their number of teams.
   Order them by the number of teams they have, in descending order and, then, alphabetically by name.

4. Who is the top scorer so far of Champions League edition 2022/2023 ?
   List his name, his nationality, the name of the team that he plays in as "team", and the number of goals scored as "scored goals".

5. List all the teams that have the average age of their squad smaller than 26 years.
   List team name as "team" and average age of the squad as "avg age".
   Order by average age.

6. Compute the standings of the group "A"
   List the columns in this order:
   - the group letter ("A") as "group_letter";
   - the team name as "team_name";
   - number of games played as "games played"
   - number of won matches as "W";
   - number of draws as "D";
   - number of lost matches as "L";
   - number of goals scored as "goals scored";

- number of goals conceded as "goals conceded";
- goal difference as "goal diff";
- number of points as "points";

Order the teams just as in real life (by points, goal difference, goals scored and goals conceded).

7. We all know it is an amazing feeling when your team scores a goal in the last 5 minutes of the game.
   Therefore, what matches would change their scores if we ignore the goals scored after minute 85.
   List the following columns:
   - date of the match as "date";
   - the letter of the group in which the game happened as "group_letter";
   - the names of the 2 teams
   - the old score
   - the new score, if the goals scored after minute 85 would be ignored

   Order by the letter of the group.

8. Show the results of all the spectacular matches that happened in the 2022/2023 Champions League.
   A spectacular match is a match with 5 or more goals scored in total, where both teams scored at least once.
   List the date of the match, the group in which the match happened, the name of the teams and the score.

9. We want to know how many goals Giovanni Di Lorenzo scored in each game that his team played so far this season.
   List the date of the match, the names of the two teams, their number of goals, and, finally, the number of goals scored by Giovanni in that match.

10. We want to know the number of goals that happened in each group so far in the 2022/2023 Champions League season.
    List the group letter and the number of goals as “total_goals”.
    Order by “total_goals” in descending order.

H. Database Triggers

For our database, we created 3 triggers that enforces or simulates some constraints. Those triggers will be described in the same order as they can be found in the project folder.

1. **Trigger “Remove_Match”**

   This first trigger enforces the constraint that, if a match is removed from our database, all the goals related to this match have to be removed.
   This is an “after delete” trigger.

   Note: Because the verification process for this trigger involves deleting a match and some goals, it is recommended to create the database again from scratch. (using the “criar.sql” and “povoar.sql” files).

2. **Trigger “Insert_Goal”**

This trigger ensures that, when we insert a goal, the goal is scored by a player that plays for one of the teams involved in the match (at the time of the goal).
This is a “before insert” trigger. Thus, if the goal does not have the correct values of its field, we will not introduce it in our database.

3. **Trigger “Insert_Contract”**

This trigger ensures that, when we insert a new contract, the player has finished all of his previous contracts.
There can not be a player that is under contract with two different teams and the same time.
It is a “before insert” trigger.

Testing our .sql files

On a command line, being located in the same directory as the 2 .sql files, we executed the next commands:

sqlite3 Test.db ".read criar.sql"
sqlite3 Test.db ".read povoar.sql"
sqlite3 Test.db

To check if our queries and triggers worked, we used the command-line "sqlite3" application in a Linux environment.

After creating and populating the database, we tested each query using the command: **sqlite3 Test.db ".read intN.sql"**

To test the triggers, we did the same command as before, for all of the 3 files linked to the trigger.

Based on this method, we did not find any anomalies or errors.

Participation of the group elements

The second part of our project was a great way to improve our team-working skills. Each of us participated in some way to both of the tasks.

The division of the tasks between the students of our group was as follows:
Task G - Emilia-Bianca, Gabriela, Alexandru
Task H - Emilia-Bianca, Gabriela, Alexandru