DATABASE PROJECT

WWI-18-DSB
Tim Kauer, Sven Metzger, Georg Schieck

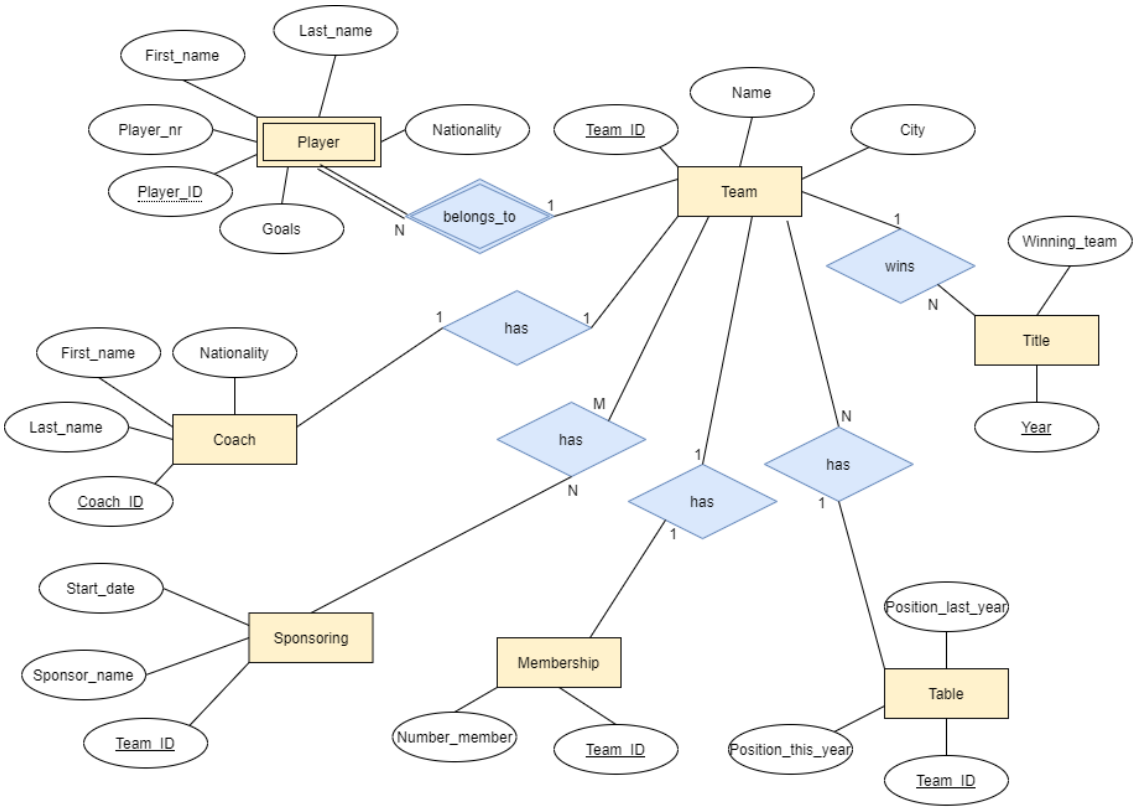
1. REQUIREMENT SPECIFICATION

The goal of this project is to implement a soccer team database filled with data about Teams and correlated data such as Players or Sponsors. The idea is to regard the data as statistics on teams at a certain point in time. User may access the data using the UI powered by FLASK. Beyond that, User may edit and update the data via CRUD operations and search for specific statistics by executing SQL-Queries.

The following bulletpoints represent a set of requirements for SOCCER database which is used to keep track of all the teams.

- The database is keeping track of every team's team_id, name and the city it is located in.
- For each team the database will contain data about the players playing for said team. It will keep track of each player's Player_ID, Player_nr, First_name & Last_Name, his Nationality and finally his number of Goals.
- The coach of each team is described by a Coach_ID, a First_name & Last_Name and his/her Nationality
- Each team is getting sponsored by at least one company. The sponsors are described by their Sponsor_name, the Start_date of the sponsoring contract and the corresponding Team_ID.
- The database holds an extra table which will contain the titles which can be won by a team. It holds information about the Year in which the title was won and the Winning_team that got the title in said year.
- The membership of a team is described by the Number_member and the corresponding Team_ID.
- Last but not least the database holds information about the ranking of the teams in a table called „Table". The ranking is described by the Team_ID, the Position_last_year and the Position_this_year so it is possible if the team made improvements or fell short of last years record.

2. ER-DIAGRAMM



3. RELATIONS & NORMALIZATION

After creating the ER-Diagramm we converted it to relations. The following image shows an example for every table that will be part of the database.

Player

Player_ID	Team_ID	Player_Nr	First_Name	Last_Name	Goals	Nationality
1	1	24	Lionel	Messi	45	Argentinian

Team

Team_ID	Name	City
1	1. FCB	München

Coach

Team_ID	First_Name	Last_Name	Nationality
1	Joachim	Löw	German

Title

Year	Winning_team
1999	1

Sponsoring

Team_ID	Sponsor_Name	Start_date
1	BMW	2004

Membership

Team_ID	Number_member
1	5221

Table

Team_ID	Position_this_year	Position_last_year
1	1	1

3.1 First normal form (1NF)

- The relations as seen above are described by attributes which only contain atomic/indivisible values and the values of each attribute contain only single values.

3.2 Second normal form (2NF)

- The second normal form also applies to the relations seen above. As seen in the image before every non-key attribute ist fully functionally dependent on the primary key. It is also important to mention, that a relation needs to be in the first normal form, which was already stated above.

3.3 Third normal form (3NF)

- A table or relation is in the third normal form if it is in the second normal form and if it contains only attributes that are non-transitively dependent on the primary key. Since this is already fulfilled in the above shown relations there was no need for normalization for this database.

3.4 Declaration of Relations using SQL DDL

```
CREATE TABLE Sponsoring (
   Sponsoring_id Int PRIMARY_KEY,
   start_date datetime,
   sponsor_name String(30),
   team_id Int,
   contract Int
)

foreign key (t_id) references Team(id)

foreign key (s_id) references Sponsoring(team_id)

primary key {s_id, t_id}
```

```
CREATE TABLE Player (
   Player_id Int PRIMARY KEY,
   Team_id Int FOREIGN KEY,
   fname String(128),
   lname String(100),
   nationality String(100),
   pl_goals Int,
   pl_no Int
)

constraint foreign key (team_id) references Team (id)
```

```
CREATE TABLE Team (
   id int PRIMARY KEY,
   team_name String(128),
   city String(100)
)
```

```
Weak entity:
foreign key (Team_id) references Team(id)
primary key {Team_id, Player_id}
```

```
CREATE TABLE Title (
   year int PRIMARY KEY,
   winning_team String(128)
)

constraint foreign key (winning_team) references Team (team_name)
```

```
CREATE TABLE Table (
   Table_ID int PRIMARY KEY,
   matches int,
   wins int,
   remis int,
   defeats int,
   points int
)

constraint foreign key (Table_ID) references Team (id)
```

```
CREATE TABLE Coach (
   Coach_ID int PRIMARY KEY,
   first_name String(30),
   last_name String(30),
   nationality String(30)
)

constraint foreign key (Coach_id) references Team (id)
```

```
CREATE TABLE Membership (
   Membership_ID int PRIMARY KEY,
   number_members int

)

constraint foreign key (Membership_id) references Team (id)
```

Index

A create index statement becomes redundant due to the use of id's as primary keys. They will be created automatically. [1]

The DDL would look like this:

```
CREATE INDEX team_id ON Team(team_name)
```

[1] See documentation:
https://www.postgresql.org/docs/9.4/indexes-unique.html#:~:text=PostgreSQL%20automatically%20creates%20a%20unique,mechanism%20that%20enforces%20the%20constraint.

View

We decided to implement a view of all Teams on the database start page:

```
CREATE VIEW All Teams in Database
AS SELECT id, team_name, city
FROM Team;
```

3.5 SQL QUERIES

Query that searches for the last name of a player given as input variable “name”

```
SELECT lname
FROM Player
WHERE lname = name; # seems to be easy, but implementation was pretty
hard
```

Query Player by number of scored goals (most goals at the top)

```
SELECT *
FROM Player
ORDER BY pl_goals DESC.
```

Query Teams by number of goals (most goals at the top) using Renaming, a Join and a Subquery

```
SELECT Team.id, Team.team_name, last_orders.goals
FROM Team
JOIN (SELECT Player.team_id, SUM(Player.pl_goals) AS goals
      FROM Player
      GROUP BY Player.team_id)AS last_orders
ON last_orders.team_id = Team.id
ORDER BY last_orders.goals DESC
```

#Query table and join with team names

```
SELECT Team.team_name,
       Table.matches,
       Table.wins,
       Table.remis,
       Table.defeats,
       Table.points
FROM Table
JOIN Team ON Table.id = Team.id
ORDER BY Table.points DESC
```

#Query the team obtaining most titles using subquery, Count-funtion and renaming

```
SELECT Team.id,
      Team.team_name,
      titl.titles
FROM Team
JOIN (SELECT Title.winning_team, COUNT(Title.pl_goals) AS titles
      FROM Title
      GROUP BY Player.team_id)AS title
ON titl.winning_team = Team.team_name
ORDER BY titl.titles DESC
```

#Query whether team “Eintracht Frankfurt” exists in database (returns a boolean value)

```
SELECT
CASE
WHEN EXISTS
(SELECT 1 FROM Team WHERE team_name=”Eintracht Frankfurt”)
THEN 1
ELSE 0
END
```

#Query the aggregate membership using the SUM-Function

```
SELECT SUM(Membership.number_members)
FROM Membership
```

4. APPLICATION

- The application was created using python as a programming language.
- To realise the use of SQL we made use of the SQLAlchemy toolkit for python. SQLAlchemy is the Object Relational Mapper which provides developers with the full power and flexibility of SQL.
- Furthermore we used the Flask framework for this project. Flask provides the needed tools and libraries to create fully functional web applications.
- The website itself is based on simple html which is easy to use and more than enough to fulfill this task.
- In addition to the technology mentioned above Docker was used to create a connection between the web application and the local PgAdmin.