Faculdade de Engenharia da Universidade do Porto

English Premier League Season 2018/2019

Database 2023/2024 – LEIC:

Professor Lázaro Gabriel Barros da Costa

Group 301

Daniel Costa Basílio up201806838@edu.fc.up.pt

Gonçalo Nuno Leitão Pinho da Costa
up202103336@fe.up.pt

Juan Ignacio Medone Sabatini up202302380@fe.up.pt

Index

Index of figures

1.Introduction

This project was developed for the database course of LEIC from FEUP. We aimed to implement a database regarding the 2018/2019 football Premier League using three CSVs that were freely available online.

This way, this database will have a round by round game results and all information that can be obtained from each game, like the red cards, yellow cards, minutes of the goal. We will also join information about the players, referees, coaches and teams so that we can have an holistic perspective about the 2018/2019 season of one of the most famous football leagues around the world.

In order to achieve this, we designed an UML model. In the first place we started with our own model, that was then reviewed with the help of AI and ended up with a final refined model. From this UML we designed the relational schema and functional dependencies of this database, using AI to help us develop this database.

Finally, we designed a create and a populate SQL files, that were then reviewed by the AI so that we could get to the results.

All this information can be read in this report, the first models we built from the ground and all the steps and conversation we got with AI until we get to the final result.

2.Domain Definition

The premier league is the highest level of the English football league system. This competition is one of the most famous football leagues around the world, which is played between August to May, this being saved in the season as season start and season end as timestamps.

In this tournament there are 20 teams, where each team plays twice against all others (once away and other at home), making it 38 games in total. The winner of this tournament is the team that in the end of the 38 games has more points. A victory is valued at a substantial three points, while a draw merits one point, and a loss garners none. Most games are played on the weekend (Saturday and Sunday) with occasional weekday evening fixtures. Every time a game is played, accessing the result, all of the information about a team itself, the number of victories, losses, draws and points are updated automatically.

Each of these contests is played on a stadium, belonging to the home team, and is directed by one referee, from whom we will have access to his experience. A game is played in two parts, with 45 minutes each, separated by a 15-minute break. In the end of the game the number of goals of each one of the teams is saved in the form home team goals and away team goals. This game starts in a specific date where we can see as a timestamp.

Over the course of the game some events occur, such as, goals, yellow cards and red cards, and for each event we have access to the team of the event and the game. Also, for goals we save the minute of the goal, and for the cards we save the total number of that color cards.

Furthermore, every team in each game has it own statistics, like shots made, ball possession and expected goals. Ball possession represents the percentage of time of the game that the team had the ball, and expected goals is a metric evaluation of the probability of a scoring opportunity that may result in a goal.

Every team has its own country and players, where we can see his position, minutes played, goals and yellow and red cards, and its own stadium with a name, where the home games of that team are played. Finally, each team has one coach, where we will have access to his name.

Even though, the tournament belongs to the English football league system, Welsh teams can take part in this tournament. As for the players in a team, there is no limitation about foreign players, with players of over 100 countries playing in this tournament. For this matter we have a Country table to save the country of the players and the teams, where we only have the name of the country.

Every person (player, referee and coach) has its name saved, besides what is saved in each of the persons itself.

3. Conceptual Modeling

3.1. Conceptual Modeling before AI

Before using generative AI, we got to a design model where we can see in figure 1. In this diagram we can see the classes that we got to with the data we had and its relations.

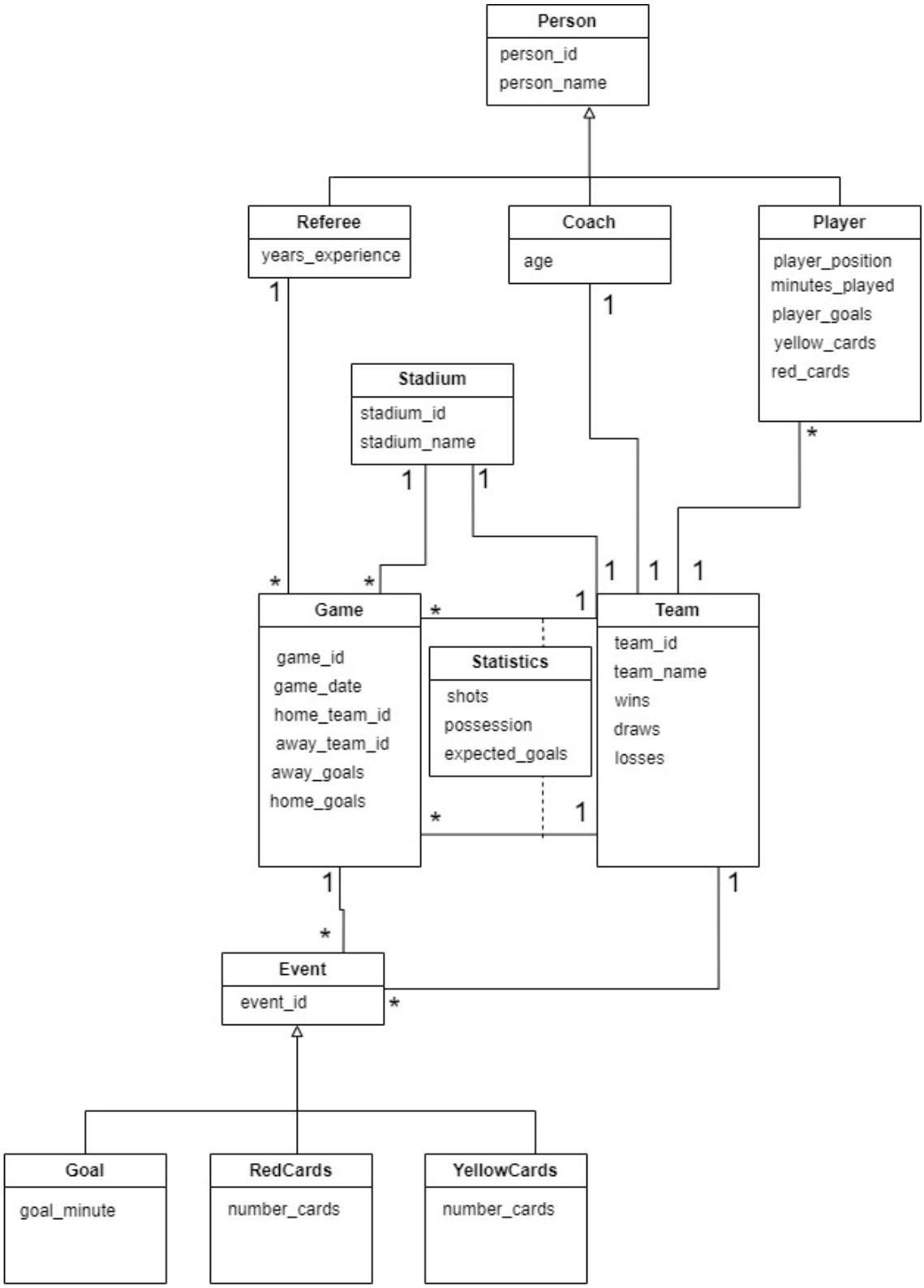


Figure 1- UML before AI

3.2. AI Integration for Conceptual Modeling

In order to improve our first model, we asked some generative Ais freely available on the market, such as Chat GPT, Bard AI and Bing AI. However, Chat GPT proved to be most helpful one.

In the first interaction we asked for help building a database for the premier league season 2018/2019, giving him information about our available data from the CSV. The first answer, shown in figure 2, is not what we wanted. It saved a lot of data in the class game, not making the separation of what should be a separate class connected to the game.

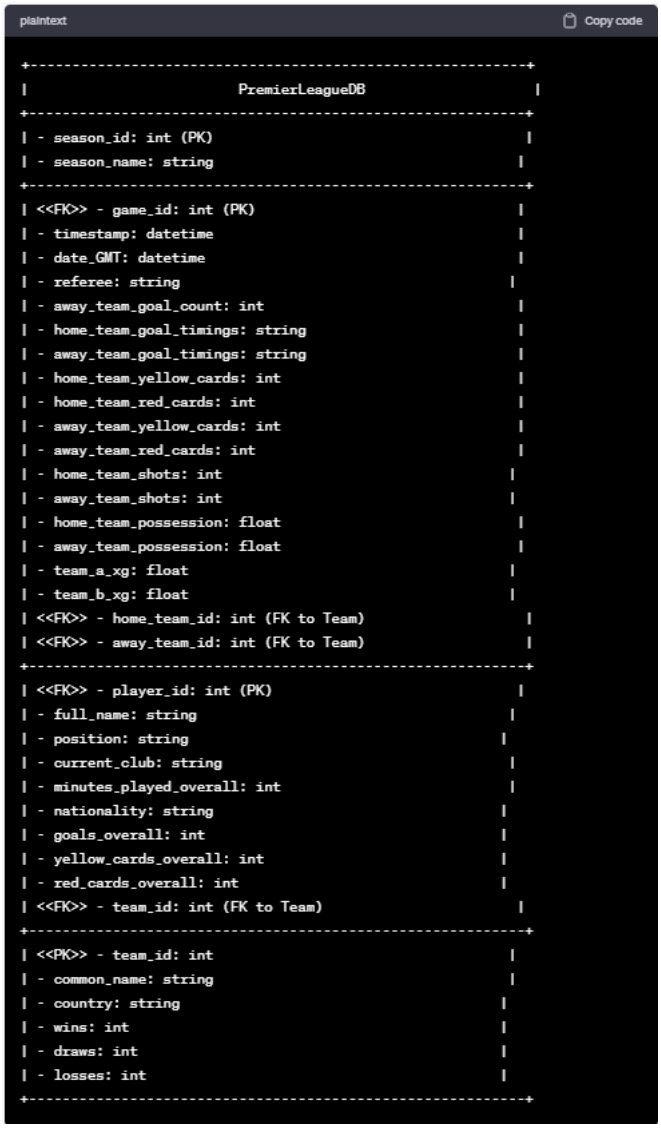


Figure 2- First AI answer about CM

Given the fact that we were not happy with this first interaction, we let him know that and the reasons behind.

We told him we wanted to separate that class into different classes, so it would be easier to watch data and also to access it. We told we didn't want that much information on the game class, we told what classes we wanted to create extra, and we got a more positive answer, as we can see in figure 3.

Figure 3- Second AI answer about CM

Chat GPT added three more classes Goal, Yellow Card, Red Cards and Statistics. However we were not completely satisfied, because we didn't save in the same class both teams information about the game, we wanted to store teams' statistics about the game separately, so we let GPT know that.

And by this point, as we can see in figure 4, GPT created a class for away statistics and home statistics.

Figure 4- Third AI answer about CM

Given the fact that this was not what we wanted we got it changed to what we can see in figure 5, where we have one statistics class, with two foreign keys, one for the team and one for the game, giving the possibility to distinguish both teams' statistics.

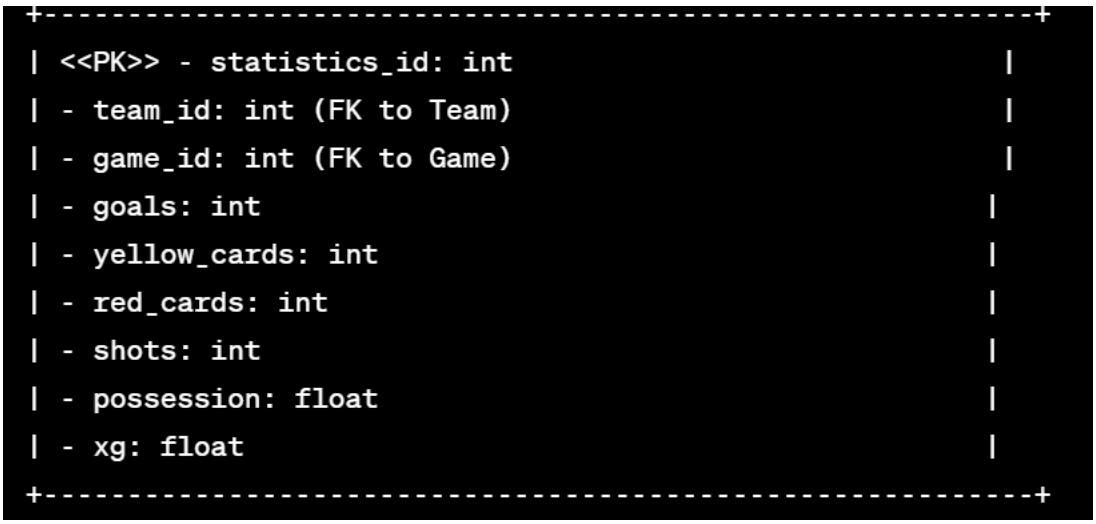


Figure 5- Fourth AI answer about CM

Now we had a new problem, which is the duplicated data that we were saving. We were saving the red cards, yellow cards and goals in the statistics class and also in separate tables, one for each of these events. As we wanted to keep these events separated from the statistics class, we asked GPT to correct this issue, leading to our final answer, shown in figure 6.

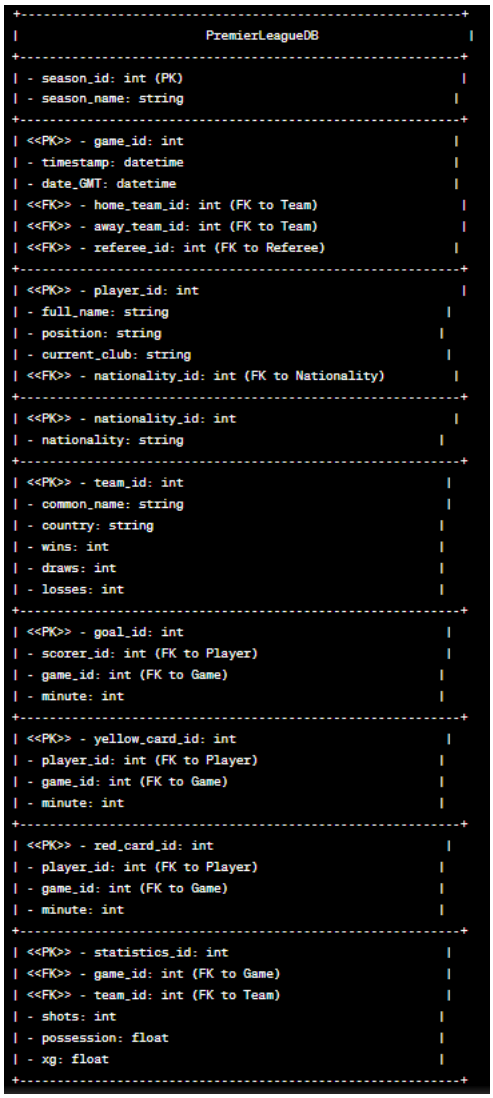
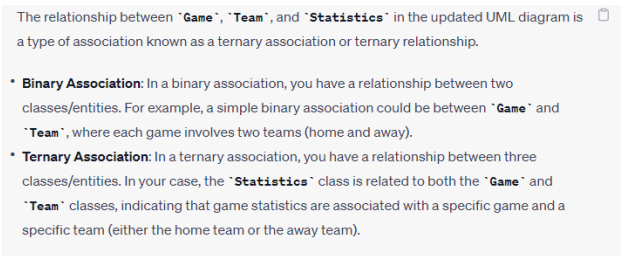


Figure 6- Final AI answer about CM

Finally, we had a doubt about what type of relation existed in the relation between game, team and statistics, given the fact that we initially thought that it was an association class, even though were not fully confident. As we can see in figure 7, Chat GPT confirmed that we were wrong, saying that it is a ternary association and not an association class.



The relationship between `Game`, `Team`, and `Statistics` in the updated UML diagram is a type of association known as a ternary association or ternary relationship.

- **Binary Association**: In a binary association, you have a relationship between two classes/entities. For example, a simple binary association could be between `Game` and `Team`, where each game involves two teams (home and away).
- **Ternary Association**: In a ternary association, you have a relationship between three classes/entities. In your case, the `Statistics` class is related to both the `Game` and `Team` classes, indicating that game statistics are associated with a specific game and a specific team (either the home team or the away team).

Figure 7- AI answer about relation between team, game and statistics

3.3. UML after AI

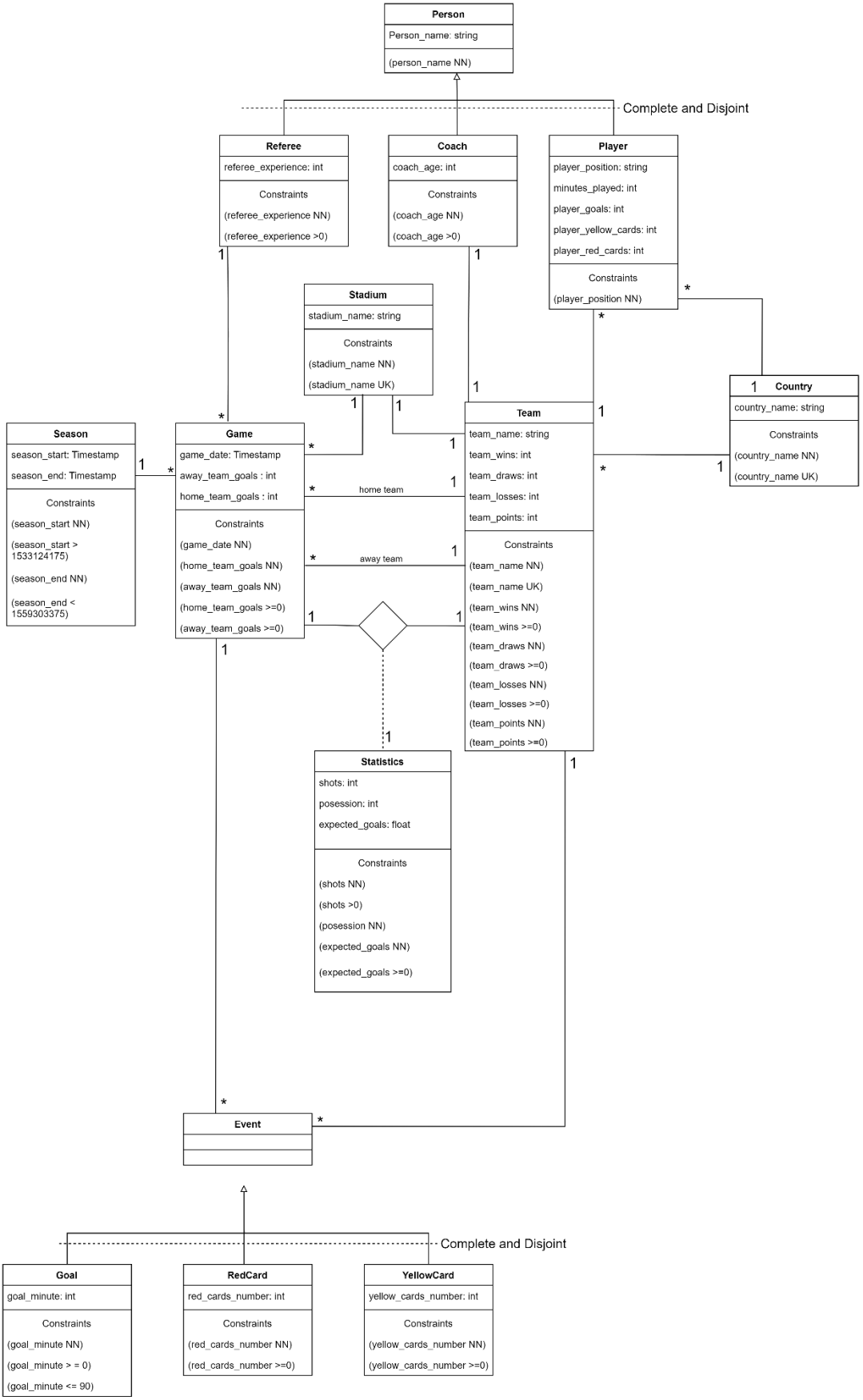


Figure 8- UML after AI about CM

3.4 Final UML

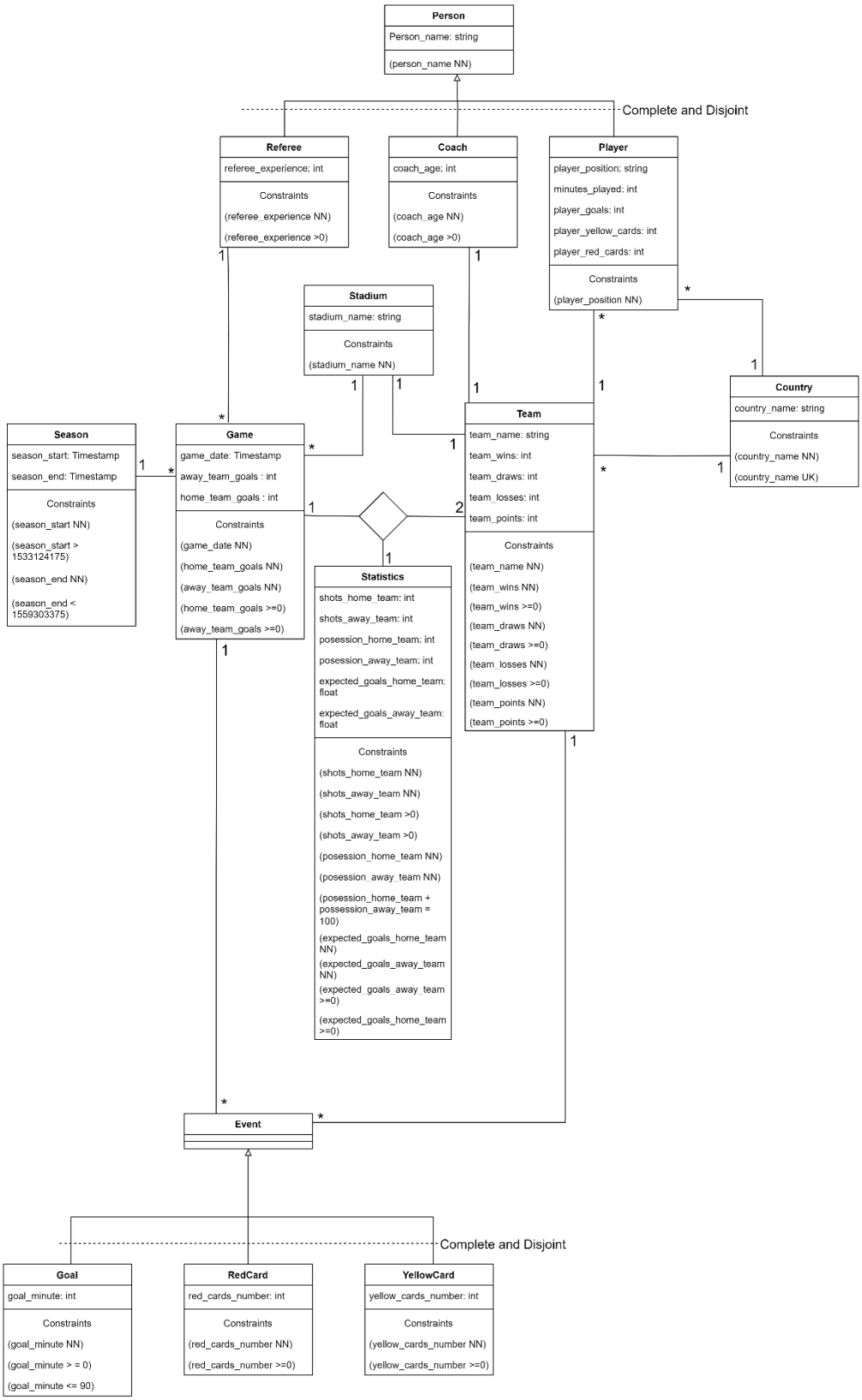


Figure 9- Final UML

After receiving feedback about our UML after AI we decided to make a few changes. Instead of having two statistics for game (one for each team) we implemented a statistics class for the game where we are going to join both team's statistics in just one class, where we are going to have home statistics and away statistics for a specific game. This way, we didn't need to have the two connections between game and team because of the ternary relation between game, team and statistics. Given this fact we needed to change the multiplicity in this relation. We also needed to add the constraints in the statistics class for the new elements we added.

4. Relational Schema

4.1 Relational Schema before AI

Person (person_id, person_name)

Coach (person_id -> Person, coach_age, team_id->Team)

Referee (person_id -> Person, referee_experience)

Player (person_id -> Person, player_position, minutes_played, player_goals, player_yellow_cards, player_red_cards, player_team->team, player_country->Country)

Stadium (stadium_id, stadium_name, team_id->Team)

Game (game_id, game_date, away_team_goals, home_team_goals, game_stadium->Stadium, game_season -> Season, game_referee ->Referee)

Team (team_id, team_name, team_wins, team_draws, team_losses, team_points, team_country -> Country)

Statistics (statistics_id, shots_home_team, shots_away_team, possession_home_team, possession_away_team, expected_goals_home_team, expected_goals_away_team)

GameTeamStatistics (game_id->Game, home_team_id->Team, away_team_id->Team, statistics_id->Statistics)

Country (country_id, country_name)

Event (event_id, event_team->Team, game_id->Game)

Goal (event_id ->Event, goal_minute)

Red_card(event_id -> Event, red_cards_number)

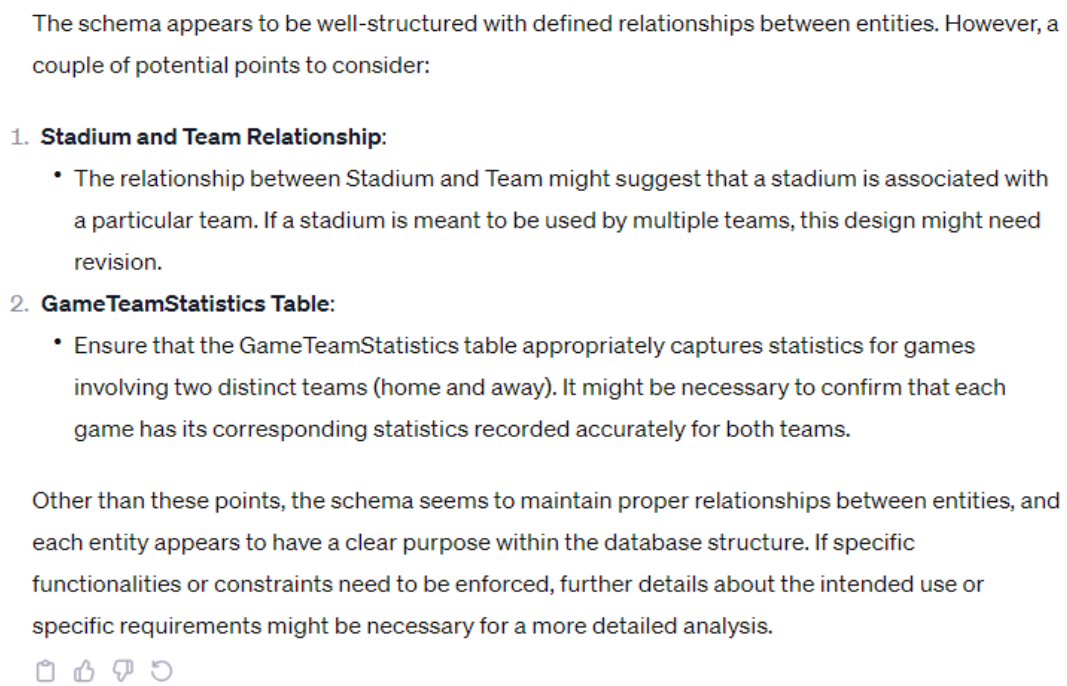
Yellow_card(event_id -> Event, yellow_cards_number)

Season (season_id, season_start, season_end)

4.2 AI Integration for Relational Schema

In order to do our AI integration for our Relational Schema we adopted two strategies. The first one was to give GPT our database, describing for him our UML, giving him every table and each row of that table and compare the outcome with what we had designed. After this we compared the two and they were the same, which means GPT did the same Relational Schema as we did.

The second attempt was to give him our built Relational Schema and ask him if he could see anything that was not being well designed or values that didn't have any relation, and, once again, we see our Relational Schema being approved by GPT, given the fact that as we can see in figure 10, GPT only point that the stadium could belong to more than one team, and that is not true for our database, and also in the GameTeamStatistics table we already took in consideration that this table has two teams, one home team and a away team.


The schema appears to be well-structured with defined relationships between entities. However, a couple of potential points to consider:

1. **Stadium and Team Relationship**:
   - The relationship between Stadium and Team might suggest that a stadium is associated with a particular team. If a stadium is meant to be used by multiple teams, this design might need revision.
2. **GameTeamStatistics Table**:
   - Ensure that the GameTeamStatistics table appropriately captures statistics for games involving two distinct teams (home and away). It might be necessary to confirm that each game has its corresponding statistics recorded accurately for both teams.

Other than these points, the schema seems to maintain proper relationships between entities, and each entity appears to have a clear purpose within the database structure. If specific functionalities or constraints need to be enforced, further details about the intended use or specific requirements might be necessary for a more detailed analysis.

Figure 10- AI integration for RS

Besides, given the impossibility of giving GPT an image so that he could give us the Relational Schema, we considered this AI integration as done, with no updates to do to our Relational Schema.

4.3 Final Relational Schema

Person (person_id, person_name)

Coach (person_id -> Person, coach_age, team_id->Team)

Referee (person_id -> Person, referee_experience)

Player (person_id -> Person, player_position, minutes_played, player_goals, player_yellow_cards, player_red_cards, player_team->team, player_country->Country)

Stadium (stadium_id, stadium_name, team_id->Team)

Game (game_id, game_date, away_team_goals, home_team_goals, game_stadium->Stadium, game_season -> Season, game_referee ->Referee)

Team (team_id, team_name, team_wins, team_draws, team_losses, team_points, team_country -> Country)

Statistics (statistics_id, shots_home_team, shots_away_team, possession_home_team, possession_away_team, expected_goals_home_team, expected_goals_away_team)

GameTeamStatistics (game_id->Game, home_team_id->Team, away_team_id->Team, statistics_id->Statistics)

Country (country_id, country_name)

Event (event_id, event_team->Team, game_id->Game)

Goal (event_id ->Event, goal_minute)

Red_card(event_id -> Event, red_cards_number)

Yellow_card(event_id -> Event, yellow_cards_number)

Season (season_id, season_start, season_end)

5. Functional Dependencies

5.1 Functional Dependencies before AI

Table Person:

Person_id -> person_name

Table Coach:

person_id -> person_name; coach_age; team_id

Table Referee:

person_id -> person_name; referee_experience

Table Player:

person_id -> person_name; player_position; minutes_played; player_goals; player_yellow_cards; player_red_cards

Table Stadium:

stadium_id -> stadium_name; team_id

Table Season:

season_id -> season_start; season_end

Table Country:

country_id -> country_name

Table Game:

game_id -> game_date; home_team_goals; away_team_goals; game_season; game_referee

Table Team:

team_id -> team_name; team_wins; team_draws; team_losses; team_points

Table Statistics:

statistics_id -> shots_home_team; shots_away_team; possession_home_team; possession_away_team; expected_goals_home_team; expected_goals_away_team

Table GameTeamStatistics:

Statistics_id, game_id, home_team_id -> away_team_id

statistics_id, game_id, away_team_id -> home_team_id

statistics_id, home_team_id, away_team_id -> game_id

game_id, home_team_id, away_team_id -> statistics_id

Table Event:

Event_id -> event_team; game_id

Table Goal:

event_id -> event_team; game_id; goal_minute

Table RedCard:

Event_id -> event_team; game_id; red_cards_number

Table YellowCard:

Event_id -> event_team; game_id; yellow_cards_number

5.2 AI Integration for Functional Dependencies

In order to ask the Chat GPT for the functional dependencies for our database, we gave him our relational schema and asked to give me the functional dependencies. At this point we compared our functional dependencies with the one that GPT gave and we identified that both tables event and person were not presents. So we asked if they shouldn't be represented in the functional dependencies, giving that our generalization was complete and disjoint. Besides this we gave an example for our event table, how we would insert the data and GPT confirmed that both event and person tables should be represented in the functional dependencies.

However, he said that the table goal (as an example) should be represented by event_id -> goal_minute. One more time we gave an example of what our database would be like and asked if the event_id in the goal didn't allow to identify the event_team and event_game and GPT said that in our database that is true.

After this first change, we had a doubt about the functional dependencies when is unique key, so I asked if an attribute is unique, if that didn't mean that he could identify the other attributes, and the answer was positive, as we can see in figure 11.

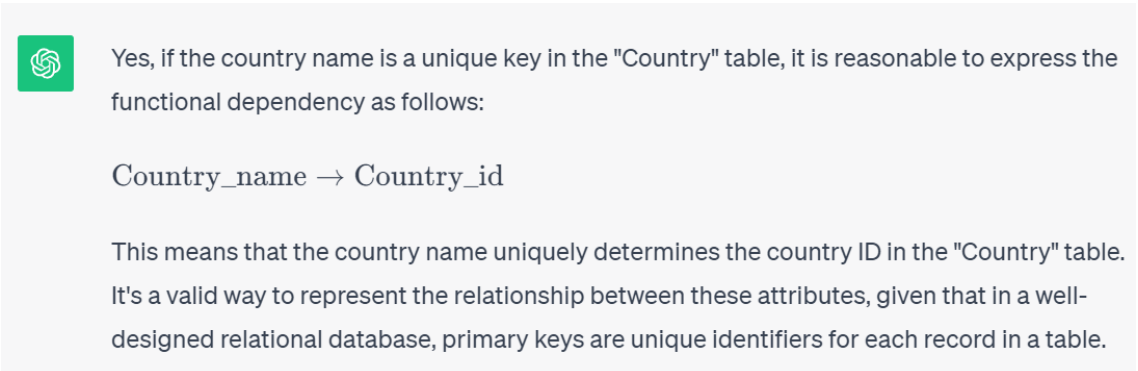


Figure 11- First AI answer on FD

5.3 Final Functional Dependencies

Table Person:

Person_id -> person_name

Table Coach:

person_id -> person_name; coach_age; team_id

Table Referee:

person_id -> person_name; referee_experience

Table Player:

person_id -> person_name; player_position; minutes_played; player_goals; player_yellow_cards; player_red_cards

Table Stadium:

stadium_id -> stadium_name; team_id

Table Season:

season_id -> season_start; season_end

Table Country:

country_id -> country_name

country_name -> country_id

Table Game:

game_id -> game_date; home_team_goals; away_team_goals; game_season; game_referee

Table Team:

team_id -> team_name; team_wins; team_draws; team_losses; team_points

Table Statistics:

statistics_id -> shots_home_team; shots_away_team; possession_home_team; possession_away_team; expected_goals_home_team; expected_goals_away_team

Table GameTeamStatistics:

Statistics_id, game_id, home_team_id -> away_team_id

statistics_id, game_id, away_team_id -> home_team_id

statistics_id, home_team_id, away_team_id -> game_id

game_id, home_team_id, away_team_id -> statistics_id

Table Event:

Event_id -> event_team; game_id

Table Goal:

event_id -> event_team; game_id; goal_minute

Table RedCard:

Event_id -> event_team; game_id; red_cards_number

Table YellowCard:

Event_id -> event_team; game_id; yellow_cards_number

After the first interaction with GPT we confirmed that both person and event tables, being the super classes of a complete and disjoint association, should be present in our functional dependencies. Also, in each subclass we could identify the elements present in them and also the elements in the super class.

Furthermore, every time we had a unique key that was not the primary key, we didn't have the implication that a unique key identifies all the other elements on that table. This way, we added new dependencies for the country, which was the only table that had an attribute that was a unique key.

Regarding possible violations to the Normal Boyce-Codd Form, we can see that id doesn't happen, given the fact that on the right side of every dependency is a superkey. Finally, we can guarantee that there is no violation to the Third Normal Form, because for every non-trivial functional dependency either the left side of the dependency is a superkey or the right side is only made by prime attributes.

6. SQLite Database Creation

After designing the first create file to create the tables in the database we asked Chat GPT for the same file. We couldn't identify many differences despite the order of the first lines, where we dropped tables if they existed. With our create file we did the drop in the order that we created the tables, however, given the fact that the tables were connected and we couldn't delete a table and having that values referenced in a different table. As we were having some problems with the drop of our document, we checked GPT's document where the drops were organized in a way that we could delete all tables with no problem as shown in the coming figure.

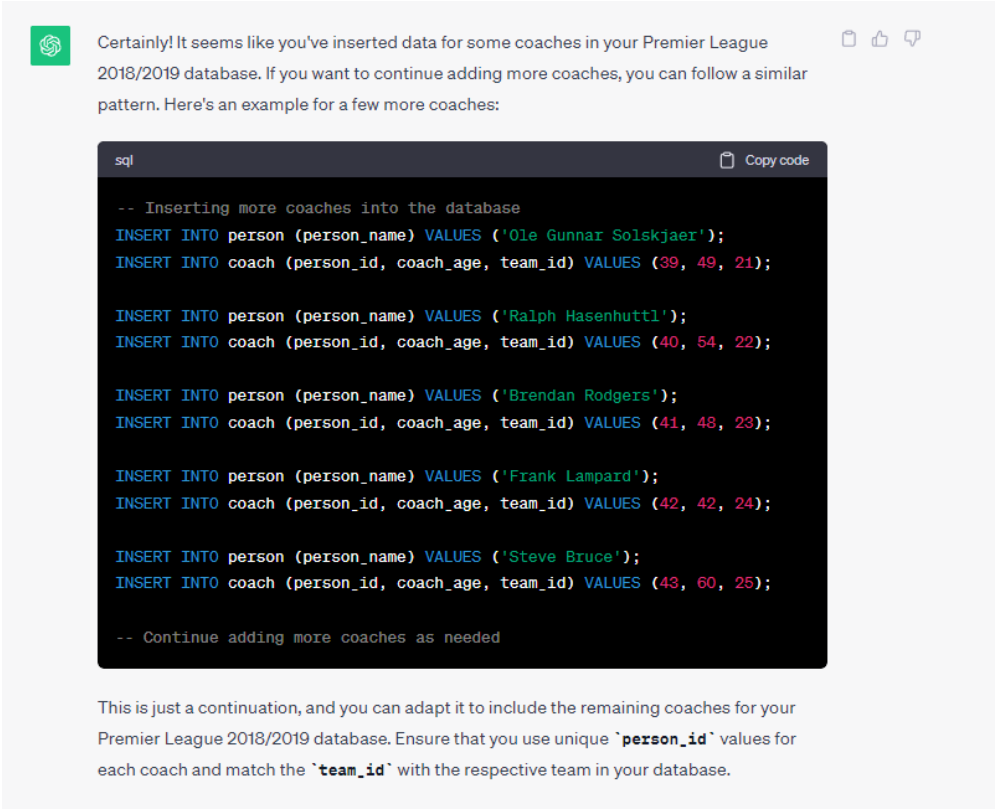
```
-- Drop existing tables (if they exist)
DROP TABLE IF EXISTS goal;
DROP TABLE IF EXISTS red_card;
DROP TABLE IF EXISTS yellow_card;
DROP TABLE IF EXISTS event;
DROP TABLE IF EXISTS gameTeamStatistics;
DROP TABLE IF EXISTS statistics;
DROP TABLE IF EXISTS team;
DROP TABLE IF EXISTS game;
DROP TABLE IF EXISTS stadium;
DROP TABLE IF EXISTS player;
DROP TABLE IF EXISTS referee;
DROP TABLE IF EXISTS coach;
DROP TABLE IF EXISTS person;
DROP TABLE IF EXISTS season;
DROP TABLE IF EXISTS country;
```

Figure 12- First AI answer on create

7. SQLite Database Population

Regarding the population of our database, we initially started by populating our database with the data that we had access using the CSV's. After this, we asked Chat GPT help to get more data.

One thing that was missing from our database was some coaches, we only had one coach per team, and that is not what happened during the 2018/2019 season of premier league, so we asked GPT help to bridge this gap.



Certainly! It seems like you've inserted data for some coaches in your Premier League 2018/2019 database. If you want to continue adding more coaches, you can follow a similar pattern. Here's an example for a few more coaches:

sql — Copy code

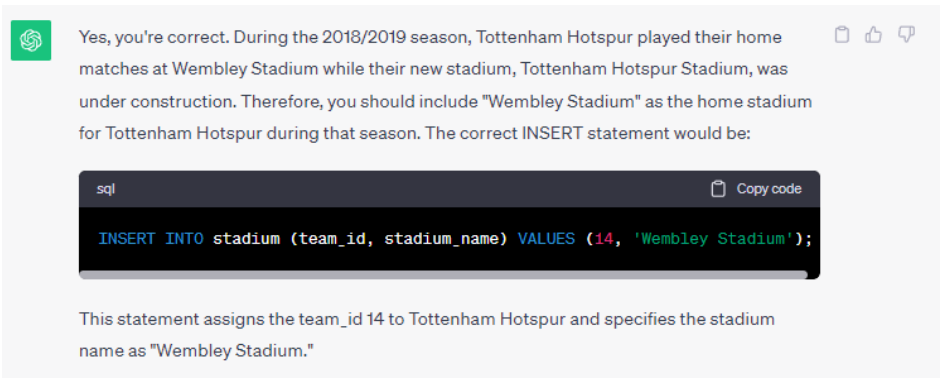
```sql
-- Inserting more coaches into the database
INSERT INTO person (person_name) VALUES ('Ole Gunnar Solskjaer');
INSERT INTO coach (person_id, coach_age, team_id) VALUES (39, 49, 21);

INSERT INTO person (person_name) VALUES ('Ralph Hasenhuttl');
INSERT INTO coach (person_id, coach_age, team_id) VALUES (40, 54, 22);

INSERT INTO person (person_name) VALUES ('Brendan Rodgers');
INSERT INTO coach (person_id, coach_age, team_id) VALUES (41, 48, 23);

INSERT INTO person (person_name) VALUES ('Frank Lampard');
INSERT INTO coach (person_id, coach_age, team_id) VALUES (42, 42, 24);

INSERT INTO person (person_name) VALUES ('Steve Bruce');
INSERT INTO coach (person_id, coach_age, team_id) VALUES (43, 60, 25);

-- Continue adding more coaches as needed
```

This is just a continuation, and you can adapt it to include the remaining coaches for your Premier League 2018/2019 database. Ensure that you use unique `person_id` values for each coach and match the `team_id` with the respective team in your database.

Figure 13- First AI answer on populate

In figure 13 we can see that GPT returned a SQL file including the coaches that were missing from our database.

Besides that, I asked if we were missing some stadium and GPT replied that Tottenham Hotspur had its stadium being constructed so they used two stadiums during that season, 'Wembley stadium' and 'Tottenham Hotspur Stadium', so GPT gave us the line that was missing in our database as we can see in the next figure.



Yes, you're correct. During the 2018/2019 season, Tottenham Hotspur played their home matches at Wembley Stadium while their new stadium, Tottenham Hotspur Stadium, was under construction. Therefore, you should include "Wembley Stadium" as the home stadium for Tottenham Hotspur during that season. The correct INSERT statement would be:

sql — Copy code

```sql
INSERT INTO stadium (team_id, stadium_name) VALUES (14, 'Wembley Stadium');
```

This statement assigns the team_id 14 to Tottenham Hotspur and specifies the stadium name as "Wembley Stadium."

Figure 14- Second AI answer on populate

Finally, we asked GPT if there were any players missing and he said no, none the less we wanted more players, so we asked to add 10 more players to our database and the result in the following figure.

```
-- Players from other leagues
-- Player 1: Goalkeeper
INSERT INTO person (person_name) VALUES ('Jan Oblak');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 2: Left Back
INSERT INTO person (person_name) VALUES ('David Alaba');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 3: Central Midfielder
INSERT INTO person (person_name) VALUES ('Marco Verratti');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 4: Striker
INSERT INTO person (person_name) VALUES ('Kylian Mbappe');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 5: Defensive Midfielder
INSERT INTO person (person_name) VALUES ('Casemiro');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 6: Right Back
INSERT INTO person (person_name) VALUES ('Joshua Kimmich');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 7: Left Winger
INSERT INTO person (person_name) VALUES ('Leroy Sane');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 8: Central Attacker
INSERT INTO person (person_name) VALUES ('Erling Haaland');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 9: Right Winger
INSERT INTO person (person_name) VALUES ('Sadio Mane');
INSERT INTO player (person_id, player_position, minutes_played, player_goals

-- Player 10: Attacking Midfielder
INSERT INTO person (person_name) VALUES ('Hakan Calhanoglu');
INSERT INTO player (person_id, player_position, minutes_played, player_goals
```

Figure 15- Third AI answer on populate

Given the amount of data that represented all the games (4437 lines) we were not able to verify our database in what concerns to the games information.

Furthermore, we asked GPT to verify our database regarding the points, wins, draws and losses for each team and the answer was the values were well inserted, both grammatically and the data itself.

This way, we considered the interaction with GPT done and finished successfully.

8. Notes

For the populate file, the best way to update each element of each team after a game, would be to create a trigger, that will evaluate the result, and after that would update the wins, losses and draws for each team. At the same point a trigger would be activated in order to update the number of points of each team after updating the team's result.

However, this feature was not implemented because we shouldn't implement triggers. This way, we did a manual update of every team's points, wins, draws and losses at the end of the file, after all games.

9. Running the project

In order to run our project, create the database and see all data, we must first enter SQLite3 environment, and then create or open a database file. Firstly to create the tables we should run the command ":read create2.sql", this will create the tables. After this we can run the command ":read populate2.sql" that will insert our data in the tables we previously created.

10. Final Thoughts

Since the work is done, we can say that we are now able to design, create and populate a database successfully.

During the project we learnt to design a database, conceptually, then transfer that to a relational model in order to finally create it.

We can say that the project was successfully completed, we achieved the goals we aimed for. In every step we used AI generative tools to help us get to the final result, and to check that everything was implemented correctly.

Before the delivery of the project, we tested our database, trying to run it locally on our machines, what was successfully done. After creating it we compared the results that were presented in our database to the real data, and we can see that nothing seems to be different.

To conclude, given the fact that our database is created without errors and the data present in the database is correct, we can say we successfully completed the project.