Final Database Systems CS355
Report

Soccer Database

Presented by
Siyabonga Mabuza
Fares Yahmadi

2024

PROFESSOR
Christine Reilly

Table of Contents

1. Introduction

Soccer is the world's most popular sport, with billions of players and viewers globally. It is a billion dollar industry where teams have paid prices above $200 million for individual players. As a result clubs invest a substantial amount of money into finding the most talented players all around the world. The database we have made is a scouting database that clubs can use to find the most talented young players across Europe's top five soccer leagues. Through queries, they can filter out the kind of players they need for their specific team needs and potentially find a future star before other teams have the chance to. A database like this could be expanded to include leagues from all over the world, and help teams discover talented players who play in underrepresented parts of the world. Given the large number of players around the world, this database helps make the process of talent discovery easier for clubs.

2. Entity Relationsips

2.1 Tables and Attributes

In this soccer Database we have created 4 tables: **Player**, **Team**, **League** and **Transfers**. Each table represents a key entity in the soccer database and its attributes are designed to store relevant information. The relationships between these 4 tables have been carefully established to ensure data constraints and reflect a real-world scouting database. These entities and relationships are presented in figure 1 (ER DIAGRAM).

Player: Represents an individual soccer player. It holds information about each footballer, and is connected to the teams they play for as well as their transfer history
Attributes:
Player_ID
Player_name
Age (age in current season)
Position (specific position e.g. attacker)
Height (height in ft)
Foot (left or right footed)
Transfer_price (price which current team paid for player)
Goals (number of goals scored)
Assists (number of assists provided)
Games_played (number of games played)
Mins_played (number of mins played)
Total_shots (number of shots taken)
Total_passes (number of passes attempted)
Total_tackles (number of tackles attempted)
Shots_on_target (number of shots on target)
Successful_passes (number of passes completed)
Successful_tackles (number of clean tackles)
clean_sheets (number of games where no goals are conceded)
saves penalties_saved (number of penalties saved)

Team: represents a soccer team. The team entity stores information about soccer clubs and links players to teams they currently play for.
Attributes:
Team_id (unique id for each team)
Team_name (official name of the team)
League_id(league which the team participates in)

League: represents the soccer league in which the team is competing in.
Attributes:
League_id (unique id for the league)
League_name (official name of the league)
Country (country which the league is located in)

Transfer history: represents the transfer history of players capturing the details of players movements between teams.
Attributes:
Transfer_id (unique id of the transfer)
Transfer_fee (amount of money in $ spent on the player)
Transfer_date (date of the completion of the transfer)

2.2 Relationships

As we mentioned before, these 4 tables are connected through relationships which we have designed as the following:

Player-Team Relationship:
A player belongs exactly to one team, and a team can have multiple players. This is a one to many relationship.

Team-League Relationship:
Each team participates in one league, and a league can have multiple teams. This is a one to many relationship.

Transfer Relationship:
Transfers track player movements between teams. Each transfer follows a player to both their previous team and their new team. This creates a relationship between the Player and Team tables.

2.3 ER Diagram

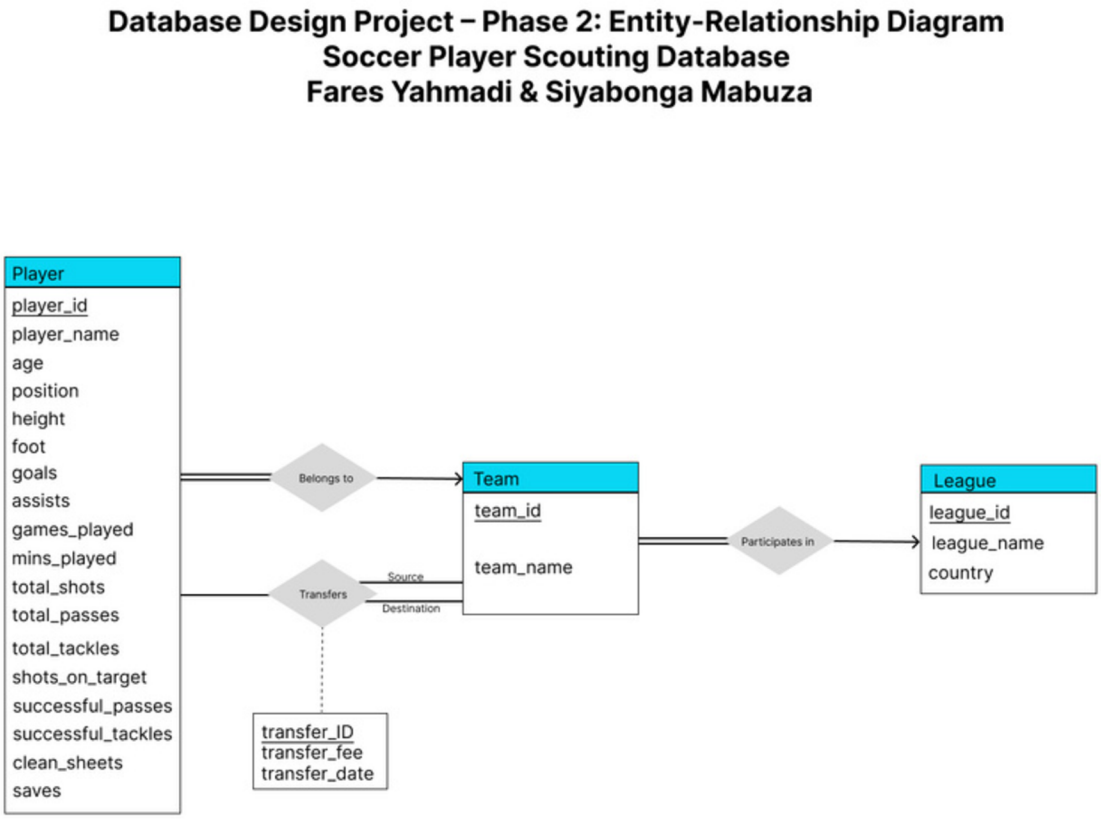

Figure 1: ER Diagram for Soccer Database

3. Database Schema

Once we have established the tables and their relationships, we can define the schema for each table. The schema specifies the structure of the tables, its attributes, their data types, constraints, and represents their relationships through primary and foreign keys.

3.1 League table

```
CREATE TABLE league (
    league_id SMALLINT,
    league_name varchar(40) NOT NULL,
    country varchar(10),
    PRIMARY KEY (league_id)
);
```

SQL Code Snippet 1 (Creating league table)

This creates the League table with its attributes league_id, league_name and country, where league_id is set as the primary key. For league_name we have enforced the NOT NULL constraint to ensure that every league has a name.

3.2 Team table

```
CREATE TABLE team (
    team_id SMALLINT,
    team_name varchar(40) NOT NULL,
    league_id SMALLINT,
    PRIMARY KEY (team_id),
    FOREIGN KEY (league_id) REFERENCES league (league_id)
        ON DELETE CASCADE
);
```

SQL Code Snippet 2 (Creating team table)

Creates the Team table with its attributes team_id, team_name and league_id, where team_id is set as the primary key and league_id as a foreign key. For team_name we have enforced the NOT NULL constraint to ensure that every team has a name. Additionally, the ON DELETE CASCADE constraint is applied to the foreign key to ensure that when a league is deleted, all associated teams are also removed.

3.3 Player table

```
CREATE TABLE player (
    player_id SMALLINT,
    player_Name varchar(60) NOT NULL,
    age SMALLINT NOT NULL,
    position varchar(20) NOT NULL,
    height SMALLINT NOT NULL,
    foot CHAR(1) NOT NULL,
    goals SMALLINT,
    assists SMALLINT,
    games_played SMALLINT,
    mins_played SMALLINT,
    total_shots SMALLINT,
    shots_on_target SMALLINT,
    total_passes SMALLINT,
    successful_passes SMALLINT,
    total_tackles SMALLINT,
    successful_tackles SMALLINT,
    clean_sheets SMALLINT,
    saves SMALLINT,
    team_id SMALLINT,
    PRIMARY KEY (player_id),
    FOREIGN KEY (team_id) REFERENCES team (team_id)
        ON DELETE CASCADE
);
```

SQL Code Snippet 3 (Creating player table)

The Player table is created with its attributes player_id, player_name, age, position, height, foot, team_id, and other performance indicators Player_id is set as the primary key and team_id as a foreign key referencing the team_id attribute in the Team table. For key attributes like player_name, age, position, height, and foot, we have enforced the NOT NULL constraint to ensure that essential information about each player is always provided. Additionally, the ON DELETE CASCADE constraint is applied to the foreign key to ensure that when a team is deleted, all associated players are also removed.

3.4 Transfers table

```
CREATE TABLE transfers (
    player_id SMALLINT,
    old_team_id SMALLINT,
    current_team_id SMALLINT,
    transfer_id SMALLINT,
    transfer_fee FLOAT,
    transfer_date DATE NOT NULL,
    PRIMARY KEY (player_id, old_team_id, current_team_id, transfer_id),
    FOREIGN KEY (player_id) REFERENCES player (player_id)
        ON DELETE CASCADE,
    FOREIGN KEY (old_team_id) REFERENCES team (team_id)
        ON DELETE CASCADE,
    FOREIGN KEY (current_team_id) REFERENCES team (team_id)
        ON DELETE CASCADE
);
```

SQL Code Snippet 4 (Creating transfers table)

Creates the Transfer table with its attributes player_id, old_team_id and current_team_id, transfer_id, transfer_fee, transfer_date where player_id, old_team_id, current_team_id, transfer_id are set as the primary key and player_id, old_team_id and current_team_id as foreign keys. For the transfer_date we have enforced the NOT NULL constraint to ensure that every transfer has a date. Additionally, the ON DELETE CASCADE constraint is applied to the foreign keys to ensure that when a player is deleted, the associated transfers are also removed.

4. Queries and APIs

4.1 Queries

Once we have created the 4 tables and established its relationship, we have designed 6 queries that utilize different concepts of SQL. These queries are:

4.1.1 Query 1

Query 1: A query on one table that uses a condition to restrict the rows that are returned from the table.

Input: Player position
Output: Player ID, Player Name, Player Age, Player Position
Description: Query returns key information on players who play in the given position.

```
SELECT
    player_id,
    player_Name,
    age,
    position
FROM
    player
WHERE
    position = "Goalkeeper";
```

SQL Query 1 Code Snippet 5

4.1.2 Query 2

Query 2: A query that joins two or more tables, plus contains a condition that restricts the rows that are returned from at least one of the tables.

Input: Transfer Fee
Output: Transfer ID, Player Name, Old Team Name, New Team Name, Transfer Fee, Transfer Date
Description: Gets transfer information on all transfers which were valued at less than the given transfer fee

```
SELECT
    TR.transfer_id,
    P.player_Name,
    OT.team_name AS "Old Team",
    CT.team_name AS "New Team",
    TR.transfer_fee,
    TR.transfer_date
FROM
    player P
    INNER JOIN
    transfers TR ON P.player_id = TR.player_id
    INNER JOIN
    team OT ON TR.old_team_id = OT.team_id
    INNER JOIN
    team CT ON TR.current_team_id = CT.team_id
WHERE
    TR.transfer_fee < 11.0;
```

SQL Query 2 Code Snippet 6

4.1.3 Query 3

A query that uses a complex condition to restrict the rows that are returned. A complex condition is more than one simple condition with the simple conditions conjoined with AND or OR.

Input: Player Age, League Country
Output: Player Name, Player Age, Team Name
Description: Gets players who are younger than given age and play for teams in the given country.

```
SELECT
    P.player_Name,
    P.age,
    T.team_name
FROM
    player P
    INNER JOIN
    team T ON P.team_id = T.team_id
    INNER JOIN
    league L ON T.league_id = L.league_id
WHERE
    (P.age < 16) AND (L.country = 'Italy');
```

SQL Query 3 Code Snippet 7

4.1.4 Query 4

A query that includes a result attribute that uses an SQL aggregate function.
Input: Player Position
Output: Average minutes played by players in given position
Description: Gets the average minutes played by players in given position in the database

```
SELECT
    AVG(CAST(P.mins_played AS FLOAT)) AS "Average Defender mins"
FROM
    player P
WHERE
    P.position = 'Defender';
```

SQL Query 4 Code Snippet 8

4.1.5 Query 5

A query that has restricted grouped results (using GROUP BY in conjunction with HAVING).

Input: Transfer Fee, (Goals+Assists)
Output: Player Name, Team ID, Goals, Assists, Transfer Fee
Description: Gets players with more than given number of (goals + assists) and have a transfer fee of less than the given amount

```
SELECT
    P.player_Name,
    TR.current_team_id,
    SUM(P.goals) AS total_goals,
    SUM(P.assists) AS total_assists,
    TR.transfer_fee
FROM
    player P
    INNER JOIN
    transfers TR ON P.player_id = TR.player_id
WHERE
    (TR.transfer_fee < 15.0)
GROUP BY
    P.player_Name, TR.current_team_id
HAVING
    SUM(P.goals + P.assists) >= 30;
```

SQL Query 5 Code Snippet 9

4.1.6 Query 6

A query that requires a sub-query or uses set operators (UNION, INTERSECT, EXCEPT).

Input: Saves, League ID
Output: Player Name, Player Position, Saves, Clean Sheets, Games Played
Description: Gets players who have made more saves than given amount, excluding player in given league.

```
(SELECT
    P.player_Name,
    P.position,
    P.saves,
    P.clean_sheets,
    P.games_played
FROM
    player P
WHERE
    P.saves > 60)
```

```
EXCEPT
(SELECT
 P.player_Name,
 P.position,
 P.saves,
 P.clean_sheets,
 P.games_played
FROM
 player P
 INNER JOIN
 team T ON P.team_id = T.team_id
 INNER JOIN
 league L ON T.league_id = L.league_id
WHERE
 T.league_id = 2);
```

SQL Query 6 Code Snippet 10

4.2 API

We then wrote an API which prompts users to input the necessary parameters for each of the six required queries. It would then take that data and input into our queries and return a table with the players who meet the parameters set by the query. The API is user friendly, allowing users who are not familiar with SQL to extract useful data about players including their game statistics and transfer history.

5. Large Data Generation

5.1 Libraries and Assumptions

In order to insert a large amount of data into our soccer database, we opted to write a python script using libraries such as Faker, MariaDb and Random. The Faker library was used to create a random name for each player. The python script allowed us to control the different values of our attributes, making the data more realistic and also making data insertion much more efficient.

For every table in our Soccer Database (Player, Team, League, Transfer) we built a generate function that creates data for our attributes and later inserts those values into their respective tables. This efficiently generated all the players for the given teams in each league. Before we discuss the functions that help generate the data for the Soccer database, it is important to note:

- There are 5 leagues.
- Every league has between 18 to 20 teams.
- In total we have 96 teams.
- Every team contains 30 players: 27 players and 3 goalkeepers.
- Therefore we need to generate 96*30 = 2,880 players for our database.
- 600 of those players have a transfer history.

5.2 Generate Functions

In the generator file we have 5 different functions that populate the database in order to be able to test the queries. These functions are the following:

1- def generate_players(x,db_cur,team_id):

—-> Generates 27 players with infield positions' (Attacker, Midfielder, Defender) attributes using the faker and random library and inserts them into the player table.
x : is a loop counter that increases by 30 to insert 30 players per team
Db_cur : cursor object
Team_id: corresponds to the id of the team we are inserting players into

2 - def generate_goalkeepers(x,db_cur, team_id):

—-> Generates 3 goalkeeper attributes and inserts them into the player table.
x : is a loop counter that increases by 30 to insert 30 players per team
Db_cur : cursor object
Team_id: corresponds to the id of the team we are inserting the goalkeepers into

3- def generate_leagues(leagues_array, db_cur):

—-> Generates leagues attributes and inserts the data into the league table .
League_array : list of tuples containing the league ID, league name and league country.
db_cur : cursor object

4- def generate_team(teams, team_id, league_id,db_cur):

—-> Generates team attributes and inserts them into the team table.
Teams: list of teams from a specific league
Team_id: integer that increments by 1 to assign each team a unique ID. 100 for German teams, 200 for French teams etc ..
League_id : every team is assigned a league_id which indicates which league it belongs too.

5- def generate_transfer(x,db_cur):

—-> Generates transfers history attributes and inserts them into the transfers table.
x : number of transfers being generated. In our case x = 600.
Db_cur: cursor object.

5.3 Generating the data

In the main, these 5 different functions will be called to generate the data in question.
1 - We call the generate_league function by passing leagues array as a parameter. This generates 5 leagues with unique IDs and inserts them in the leagues table.
2- Using the get_ids_from_db function we are able to access all the IDs of the different leagues we just inserted. We iterate through those IDs to generate their respective teams and assign them unique IDs that correspond to the league. The generate_team function also inserts this data into the team table.

3- Once we have created the league and team table we can begin to add 30 players: 27 infield players and 3 goalkeepers to each team. For that we iterate through all the team ids we have just inserted, and call the generate_goalkeeper and generate_players functions. To ensure every player has a unique ID we create a count1 that increments by 30.

6. Conclusion

6.1 Improvements

As mentioned earlier, the database could be improved further by adding more leagues and players to it. This can allow teams to discover talent from all over the world, and give opportunities to players who often go unrecognized through exposure from the database. We could also add more specific queries to enable scouts to get players who fit their desired profile more accurately.

6.2 Challenges

The biggest challenge we faced while making this database was generating realistic data. Initially we tried to use ChatGPT to create players for every team and each of the 5 leagues. But it was producing data that was missing some of the needed players statistics, as well as producing data that is extremely unrealistic. For example, it produced goalkeepers with multiple goals, which would make it difficult to see if our queries are useful. To fix this problem we wrote a python program with multiple restrictions in order to produce realistic players. Within that python file we also used the built in ‘faker’ library, which generated unique random names for each player in the database. This made our database look more realistic and it saved us the trouble of having to come up with thousands of random names ourselves.

6.3 What we learnt ?

The most interesting thing we learned from this project was the power of queries to extract meaningful insights from a well-designed database. By creating specific queries, such as filtering players based on their position, age, or performance metrics like goals and assists, I realized how databases can simplify complex decision-making processes. This reinforced the idea that the true value of a database lies in how effectively it allows users to retrieve and analyze data to meet their needs. It was fascinating to see how combining logical relationships and targeted queries could provide clubs with actionable information to identify potential star players.