Designing & Using Databases

Final Project Report

Soccer Database with UI

https://github.com/dsanmart/soccer_database

Pablo Ortega, Diego Sanmartin, Henning Gruhl and Tomas Vintimilla

November 30th, 2021

Table of Contents:

Systems Analysis and Requirements:

Our system is geared towards Managers, coaches, and club owners, and provides them with useful statistical data on player and team performance based on the data that resides in the system.

Depending on the type of data that the user wants to insert into the system, he will have to note the types of columns that exist within every table in the system. In other words, the user is constrained by the type of data that each column accepts. A column that only accepts integers (of specific length), shall not accept other data types.

Also, since some table objects are declared mandatory and share the same column name with optional tables, the displayal of new optional data that has been stored in the system may not be possible if the user has not added data to specific tables. For instance, a new player may not be displayed if no teams have been previously added to the database or if the new player does not share the same team id.

The relationship between the objects and their constraints are represented by our ER diagram that you can find on ***DFD & ER Diagrams***.

In terms of the function of each existing object, the name of each table is already self-explanatory but we nevertheless provide a description of each table's functions below.

Here is a list of the existing tables, their constraints and finally their function within the system.

- amateur/professional (int, tinyint, varchar) gives information on whether the player is amateur or professional as well as their salary, start date, end date, hobbies, occupation and institute
- event_type (int, tinyint) provides information on the events of a match used for producing statistics.
- match (int, date, varchar, tinyint) this object provides, season, date, place local & away team, winner team and the extra time.
- matches(int, date, varchar, tinyint) similar to match object but also provides information on if the match concluded in a draw.
- player_pers_info (int , varchar , tinyint) information on which team the player belongs to, their name, weight, height, date of birth, total number of national matches the

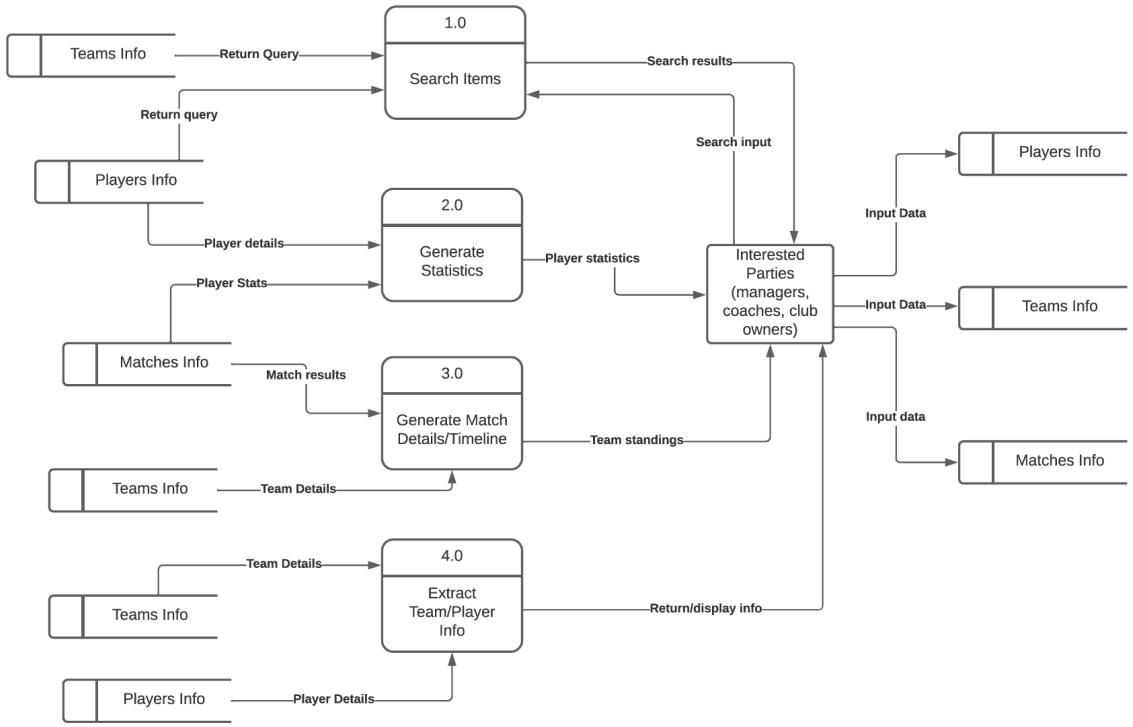
player has played, total number of international played matches, and finally a photo of him.

- player_pos (int , varchar , date) player position, start date and end date of said position
- team (int , varchar) name of the team, name of the team's stadium, an image of their logo, where it is located at, and an image of their stadium.

Since we did not have the chance to interview anyone in the football space nor their user base, we decided to leave out this part of the System Analysis and Requirements Stage. But, it is worth mentioning that some of the authors of this project are passionate about football.

DFD & ER Diagrams:

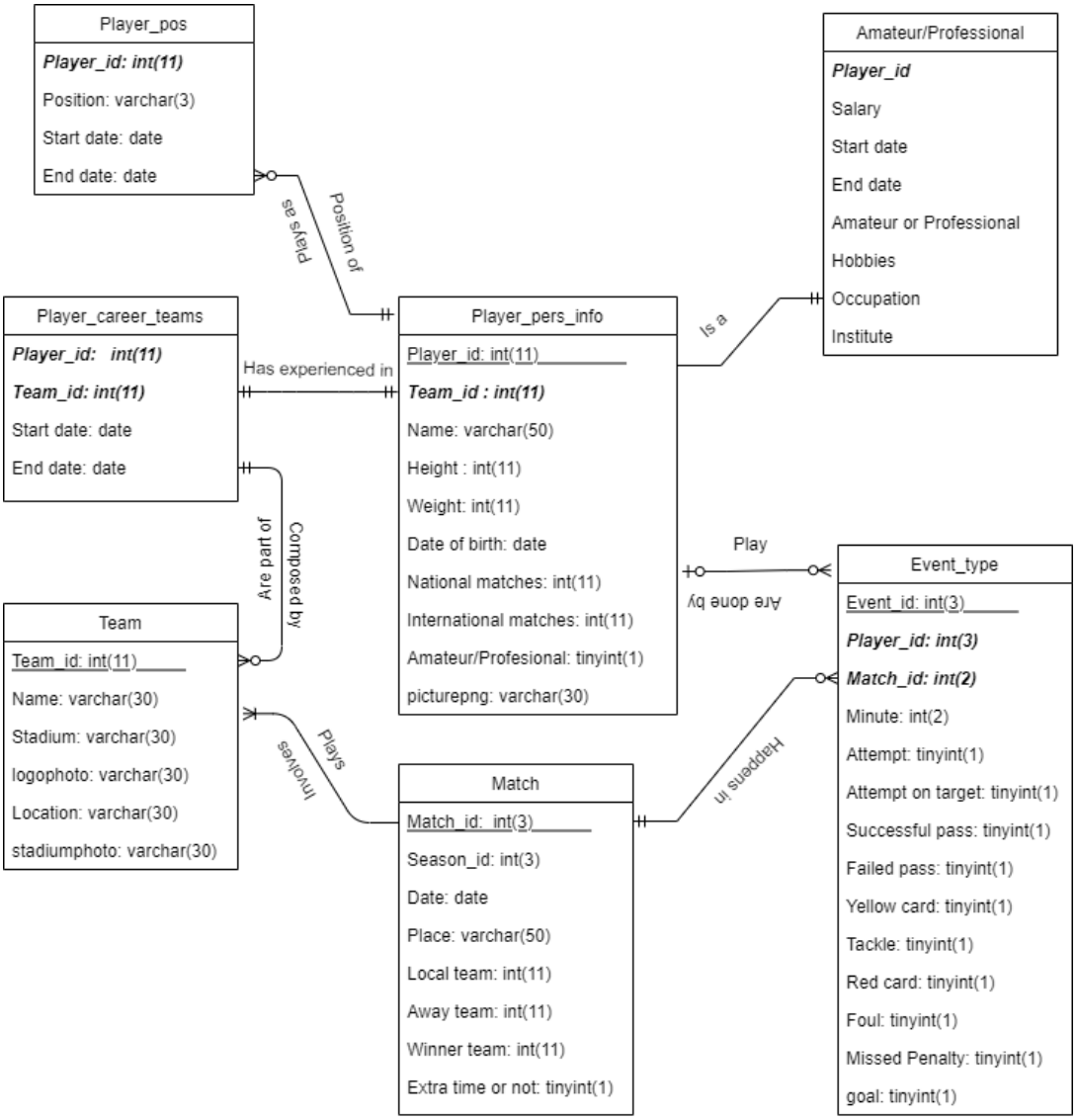
1. DFD DIAGRAM



The DFD Diagram proposed for this database, as seen above, is one composed of one main external entity, four processes, and three different data stores, which are repeated in some cases simply for visual purposes. The external entities in this case are the interested

parties for the database, ie Managers, Coaches, Club owners, etc. The external entity has the ability of inputting data into the data stores, as seen on the right-hand side of the diagram. In regards to the data stores, we have divided them in three categories: players info, teams info, and matches info, nevertheless, they are all part of the same database, but due to visualization purposes, they have been split. Moving on, as it can be seen on the left hand side, the data stores are followed by the processes which are then followed by the external entity. In regards to the processes, these are the four main functionalities of our database, these being searching for an item, generating statistics, generating match details & timeline, and extracting player/team information. These processes retrieve data from the different data stores, which are indicated with the arrows, along with the specific data that is being extracted. Eventually, these processes return back to the entity the desired output.

2. ER DIAGRAM



The ER diagram we have used for the soccer database is composed of 7 tables, where the primary keys(yellow) are team_id, match_id event_id, and player_id, with their corresponding foreign keys(blue) in the related tables.

We want to point out how we have represented the events of a game in the diagram since it was the most difficult part and the most different one between the class groups. We created a table, called event_type, with a primary key, event_id, that will differentiate each event by the other; and then there are two foreign keys, player_id, and match_id because its event is done by different players and in different matches. Then we create an attribute with the minute in which the event occurs, and finally we create variables with all the possible events that can happen in a match represented as booleans, where, for instance, if there is a

goal, the attribute goal will be a 1 and the rest a 0. The way we create the table will help us to count how many goals or yellow cards have been during a match or to create all the statistics of an entire season in an easier way.

Normalisation:

Our DBMS is considered to be in the normalisation level 3NF due to the fact that it satisfies the conditions for both NF1 and NF2, in addition to having no transitive functional dependencies. To begin with, the DBMS satisfies NF1 since the tables do not contain multiple values, but rather single values each corresponding to a specific attribute. Moving on, the database has also achieved NF2 since it complies with NF1, and also all non-key attributes are fully functional dependent on the primary key. Lastly, our database management system is said to achieve NF3 since it complies with both NF1, NF2 and in addition there is no transitive dependency for non-prime attributes.

Indexing:

To optimize our database and to improve the performance of the database we could use indexing.

All our primary keys will be indexed: team_id, player_id, event_id and match_id. But another variables that could be indexed to have a more quickly access to the data could be the the team.name(name of the teams) and the player_pers_info.name(name of each player), specially for the search engine, where the user types a player/team and using indexing will be more efficient to have it sorted.

Code:

In our 20 files (11 PHP and 9 CSS) we have used many languages like SQL, PHP, HTML, Javascript, and CSS. Since we have a lot of code, we uploaded it to github and we are going to explain very briefly how we did the code by giving some examples.

1. SQL FOR TABLES

All the SQL code that creates the tables of the database is in the github repository file called "soccer.sql". Here, we will explain some complex SQL queries that we used in order to represent the data on our database.

Firstly we had to connect each PHP file with our database, so we could manipulate the tables and extract the info we were interested in. Then, we created many variables using LEFT JOIN, GROUP BY, ORDER BY, COUNT(*)...

To see how we use them, we have an example where we use most of this functions:

In the team_info_stats.php, we collect the data to create the stats for each team. To see the most- goal scorers(pichichis) of our database we create a variable using this code:

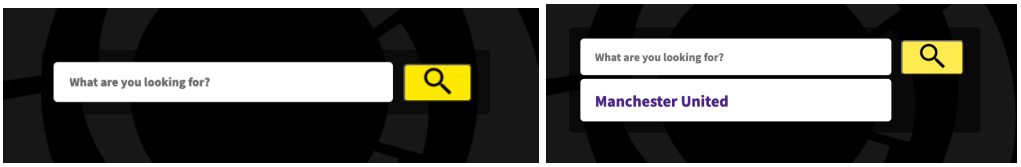
```
$pichichi = "SELECT count(goal), player_pers_info.name FROM (event_type LEFT JOIN player_pers_info ON event_type.player_id = player_pers_info.player_id) WHERE goal=1 AND team_id=$id GROUP BY event_type.player_id ORDER BY count(goal) DESC" ;

$pichichis = mysqli_query($db, $pichichi);
```

Here, we SELECT how many goals each player scored using Count(*) and the name of the player. Then we had to create a table merging event_type table and player_pers_info, since in the event_type table we only know the id of the player that did each event, and we want to know the name. For the Count(*) function to be effective, we had to filter and only take only the rows where the variable was 1 and group by player_id. Finally, we order the final data in descending order of goals scored.

2. USER INTERFACE

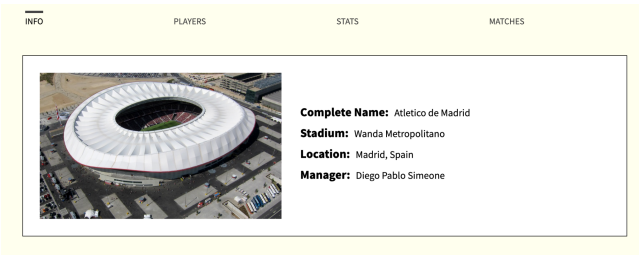
To design the user interface we used HTML, JavaScript and CSS. We first did a main navigation bar that was placed at the top of the page in order for the user to navigate around the different web pages. When the user mouse hovers over one of the page names an animation shows on top of the name that was done with css hover function.

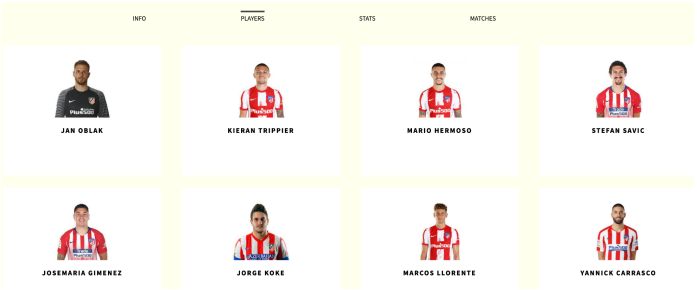
In the home screen we introduced a search bar that allows the user to search any team or any player in the database and navigate to its page by clicking on it.



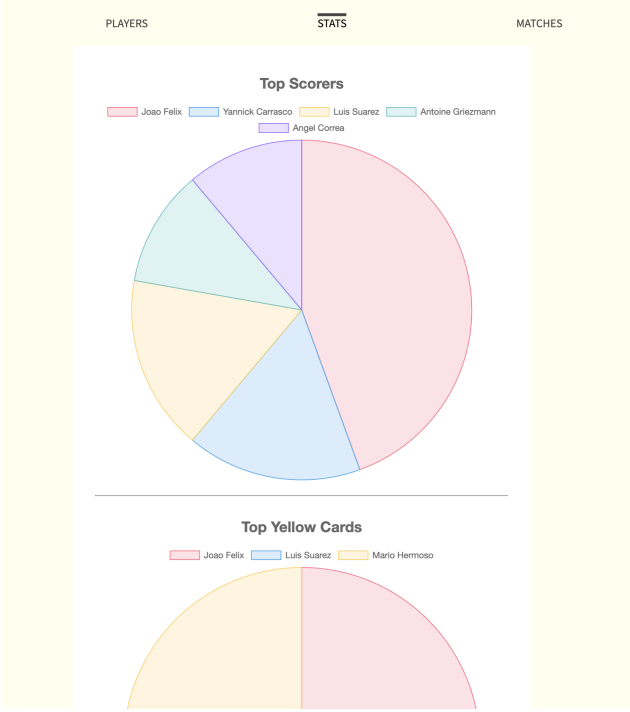
On the teams page, we made a class that styled a card for an individual team so that it could be used by every team on the database. This included the logo, name (with link to the team page), location, and stadium of the team.

When you click on a team name, it will guide you to the team_info.php page and get the team id to filter the chosen team's data. On the team info page, we introduce a second navigation bar with the same operation code as the main nav to allow the user to open different pages for the team's info, players, stats, or matches.

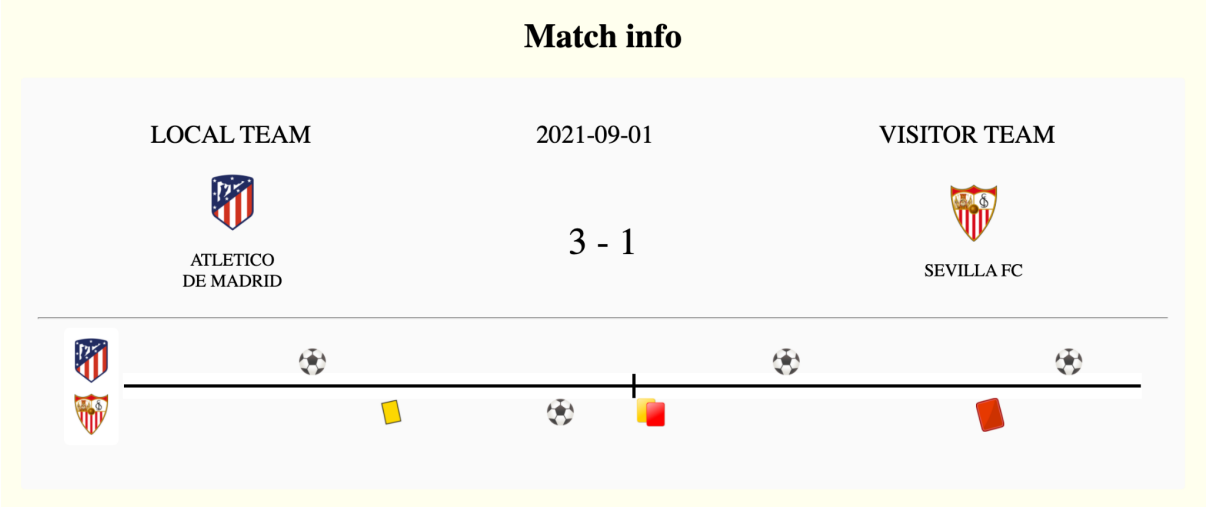




On the players page, we used the same card style as the team cards on the teams page and you could click on the players' names in order to go to the player's info page. For the stats page, we used an open-source JavaScript library called Chart.js.



On the matches tab we showed all the matches from the chosen team with their corresponding date and showing whether they played local or away. If the user clicks on the "vs" text, it will navigate them to the chosen match's info. On the match info page, it shows the teams with the result of the match and a line representing a timeline of the match with the events that have happened in their corresponding minute of the line.



To represent the events in their corresponding minute and side of the line we selected the minute of each event and split the events of each team. First the goals (WHERE goal=1), then direct red cards (WHERE yellow_card=0 and red_card=1), then second yellow cards (WHERE yellow_card=1 and red_card=1), then yellow cards (WHERE yellow_card=1 and red_card=0).

In order to represent the events in the corresponding part of the line we made a formula that is responsive for all screens. For this we did a div of 60% of the screen that included all the data (white background on the picture). Then we did a second div (100% of the 60%) under the <hr> that separates the score from the timeline. The team logos we assigned a width of 3% of that second container and for the line we made another div of width 90%. Hence the line is 54% of the entire screen (90% of 60%).

To every event, we moved it to left by 23+(minute*54)/90 %. Using percentages is what allows it to be responsive for all screen sizes. We added 20% from the left of the first container that is centered plus 3% from the team logos and then multiplied every minute by 54 and divided it by 90 minutes which is the length of each match.

```
$yellow22= mysqli_query($db, $yellow2);
while($row = $yellow22-> fetch_assoc()){
    echo "<img src='pics/amarilla.png' style= 'width: 1.5%; position: absolute; left:"  . 23+($row["minute"]*54)/90 .  "%' >"  ;
}
```

Also, on the players page from the main navigation bar the user can see all the players from the database with the same format as the team players and click on them to see their information.

Finally, on the main stats page, the user can see on a bar chart the players with most goals, yellow cards, red cards, fouls and missed penalties from the entire database.

Improvement on Database Structure and Implementation:

In order to improve the speed and ease of adding data from events that happen in the matches, we could substitute our event_type table with 2 new tables. One would collect the event_id, match_id, player_id and minute. The second one, connected to the first, would have the event_id as a foreign key and an event name that would be a string saying whether it was a goal, yellow card, missed penalty, etc.

This new structure would avoid the necessity of typing several tiny integers for every event and will reduce the density of the database. It will also change the operation of some features like the timeline of the matches as it won't allow us to check if it was a second

yellow red card (WHERE yellow_card=1 and red_card=1) or direct red card (WHERE yellow_card=0 and red_card=1). But we could easily fix this by naming the event name as "second_yellow" when a player gets 2 yellow cards.

The new ER Diagram of the database would then look like this: