

The background of the slide is a light gray with a repeating pattern of soccer balls. The balls are rendered in a 3D style with white and dark gray panels. They are scattered across the slide, with some appearing larger and more prominent than others.

SPANISH LALIGA 2018/19

IST-Database and Data Mining

NOVEMBER 7, 2023

Gonzalo Jaraba Ruiz de Alegría – Daniel Raposo Sánchez

1. Introduction

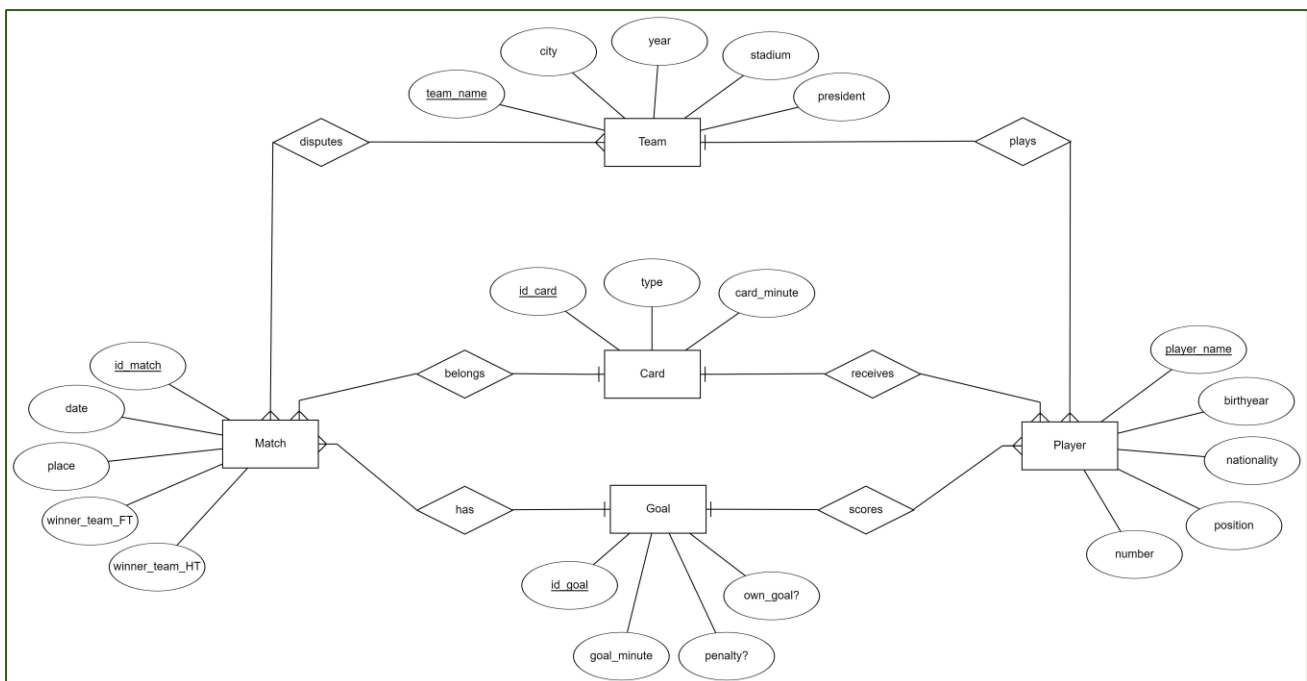
In this report it is explained the process carried out corresponding to the final project of the IST-Database and Data mining course. It covers the concepts of the subject and the most common issues when creating databases.

It includes a presentation of the chosen data set with its corresponding Entity-Relationship model and the design process behind it. Following this, the process of data extraction and preprocessing is explained in detail, putting emphasis on the main problems we have encountered and how we tackled them. To finish, we explain the DB implementation, some questions we have formulated converted to SQL Queries and a Full-stack application to showcase the results. All the code can be accessed through a GitHub repository¹.

2. Domain selected and ER Diagram

From the very beginning we chose to work with real data instead of randomly generating it since we thought that it is one of the main issues that database professionals face in their daily work, so we wanted to go through all the full process.

The domain chosen is the main Spanish soccer competition 'LaLiga'. We easily found CSV files with all the matches² and players³ of the 2018/19 season. With this data, we designed an ER diagram for the database containing information that we did not have at first: cards, goals, and teams. But we wanted a useful database with enough information to make some interesting queries. Extraction of this extra data is explained in the following sections.



The objective when designing our ER model is to make it as simple as possible, with no redundant information but still being able to access any type of data with ease. At first, we had the Team names into the Match table: but we wanted to divide these two to be able to provide some extra information about the teams and treat it as a separate entity.

¹ <https://github.com/danielraposo/dbm1laliga>

² <https://datahub.io/sports-data/spanish-la-liga>

³ https://www.kaggle.com/datasets/alvarob96/laliga_2018-19_season_player_stats

In Match we have its ID as primary key (PK) since the date or result of the match is not a distinct attribute, the date, the place where it happened, the winner team at full time and the winner team at half time, which will be indicated not by the name of the team but by “Away Team” or “Home Team”.

To connect each Team with its Match, we have a many-to-many relationship *disputes* which contains both the ID for each match as well as the team names as foreign keys (FK).

On the other side, we have Player which is connected to Team by a one-to-many relationship *plays*, which means that each player will have as FK the name of the team for which he plays. Also, its full name is used as PK (there are no two players with the same name and surnames).

In the middle, there are two entities: Goal and Card. They both have its particular ID as PK, and the ID from the Match as well as the name of the Player as FKs. To connect these, we have the next relationships: A Card belongs to a Match and a Player receives it, a Match has goals and a Player scores a Goal.

3. Data extraction and preprocessing

In this section we will detail where from and how all the data was extracted.

a. Match table

As mentioned above, we found a CSV containing information about every match played in the season. The CSV contained some attributes that we decided to discard such as betting probabilities or percentages of shots from each team. Also, we deleted the names of the teams and only kept the final and half-time results as “Away Team”, “Home Team” or “Draw” since these names will be present in the Team table.

b. Player table

To create this table, we followed some steps since we had partial information from a CSV file but wanted to eat data such as birthyear and nationality of each player. There was also the issue that some names in the CSV were the “artistic” name of the footballer rather than the full name, which meant that for the goals and cards, this information would not match.

So, first we used ChatGPT to obtain the birthyear and nationality as well as the full name (for most) of the players. After this, we double checked with the goals and cards to obtain discrepancies or similarities, for example: names with one surname instead of two, names without accent marks, shortened names (Daniel = Dani), etc. We even had to add some players that were not present in the CSV or that had no shirt number (due to exchange between teams or no match records).

c. Team table

The information from this table has been written manually since we only have 20 teams, each with its name which is unique, city, year of foundation, its stadium and president. They have all been extracted from Wikipedia and formatted into a CSV file.

d. Goal table

For the goals scored we did not find anything online. As in one season there usually are around a thousand goals, it was not convenient to search for each goal and manually write it down. We decided to implement a python algorithm that searched for this information and saved it as a CSV file. This algorithm works as follows:

1. Read a CSV containing all the matches information and iterate over each row.
2. For each row, create a google consult, parse the HTML response and search for the specific HTML identifier used by Google to display the goals information.
3. Extract all information from the goal (player’s name, minute, own goal?, penalty?) and create a new row in a CSV adding the match identifier that will be a foreign key.

We had some issues getting the information correctly (a lot of formatting) and because Google blocked our IP from making these automatic consults, we had to avoid it using virtual private networks.

Finally, we managed to get all the goals, but we had to go over each player making sure his name corresponded exactly to the name in the Player table, as they are foreign keys.

e. Card table

We had the exact same problem as with the goals. But this time was even tougher because this information is not directly shown when searching for a match on Google. It is needed to click on a button to display this information. After a lot of effort trying to make it completely autonomous and failing, we decided to implement a manual step in which it was necessary to copy the HTML code of the page containing the information we were looking for. So, we did not have to write each card one by one, but we did have to copy the HTML code for each match (there are 380) for the python script being able to extract all the cards with its type, minute, and player name.

As with the goals, we had to double check the players' names with the ones at the Player table.

4. Database implementation and population

As in the practical session in class, we used a docker container with a Postgres and a Jupyter Notebook image. The Postgres image to store and manage our database and the notebook to use python to easily connect and communicate with the database and to have all our work well stored. Note that it is important to mount a volume in the host machine to have persistence of the databases.

istdbdm 2 containers	-	Running (2/2)
notebook-1 d083edad765a	istdbdm-notebook:latest	Running
postgres-1 31f1e4d88b9a	postgres:latest	Running

It is important to stand out the order we followed when creating the tables. We had to be careful so when creating a table with a foreign key, this key already exists in a table in the database. Therefore, the order followed has been as follows: Team, Match, Player, Goal, Card, Disputes.

In order to populate the tables, since we had a CSV file for each one, we used the CSV library to read them and iterate over each row to insert it to the table as showed in the example, where we used a try-except clause for detecting errors when inserting and rolling back to the initial state to maintain consistency.

```
try:
    with open('team.csv', 'r') as f:
        reader = csv.reader(f)
        next(reader) # Skip the header row.
        for row in reader:
            cursor.execute("INSERT INTO team (team_name,city,year,stadium,president)
VALUES(%s,%s,%s,%s,%s)", row)
        con.commit()
        print("Successfully inserted data from CSV into team table")
except (Exception, psycopg2.Error) as error:
    con.rollback()
    print("Error While inserting data into the team table: ",error)
```

5. SQL Queries

Before starting this project, we formulated some questions that we thought would be interesting taking into consideration our data set:

- **In which matches the home team starts losing (result at half-time) but ends up winning?**

```
SELECT match.id_match,date,place,ARRAY_AGG(team_name) AS homeTeam_VS_awayTeam FROM match
  INNER JOIN disputes ON match.id_match = disputes.id_match
 WHERE match.winner_team_ht = 'AwayTeam' AND match.winner_team_ft = 'HomeTeam'
 GROUP BY match.id_match
 ORDER BY match.date ASC
```

Relational Algebra: $\Pi_{match.id-match,date,place,team-name}$
 $(\sigma_{match.winner-team-ht='AwayTeam' \wedge match.winner-team-ft='HomeTeam'}$
 $(disputes \bowtie Match))$

We look for the matches in which the winner team at half-time is the away team AND the winner team at full time is the home team. The JOIN is made on `disputes.id_match = match.id_match`.

- **Which players born after 1985 scored more than 15 goals in this season?**

```
SELECT player.player_name, birthyear, nationality, position, COUNT(*) AS goals FROM player
  INNER JOIN goal ON goal.player_name = player.player_name
 WHERE player.birthyear > 1985 AND (SELECT COUNT(*) FROM goal
 WHERE goal.player_name=player.player_name )>15
 GROUP BY player.player_name
 ORDER BY COUNT(*) DESC
```

Relational Algebra: $\Pi_{player.player-name,birthyear,nationality,position,goals}$
 $(\sigma_{player.birthyear \wedge goals>15} (Goal \bowtie Player))$

We look for the data about the players that scored more than 15 goals and have been born after 1985. To do so, we need to JOIN the tables Goal and Player when the name of the player that scored the goal matches the player's name. Then we need to COUNT how many times each player's name appears in Goal to select only those that exceed 15 while specifying that its birthyear > 1985.





- **Which are the top 5 players that received the highest number of yellow cards and from which teams are they?**

```
SELECT player.player_name, player.team_name, COUNT(*) AS yellow_cards FROM player
  INNER JOIN card ON card.player_name = player.player_name
 WHERE card.type = 'Yellow'
 GROUP BY player.player_name
 ORDER BY COUNT(*) DESC
 LIMIT 5
```

Relational Algebra: $\Pi_{player.player-name, player.team-name, yellow-cards}$
 $(\sigma_{card-type='Yellow'}(Card \bowtie Player))$

In this case we JOIN the tables Card and Player when the player's names match. After this we only choose the yellow cards that have been drawn to players, counting how many times each player name appears and only displaying the top 5 players with the highest number of yellow cards.

Finally, we wanted to recreate a score board with the result of the season. Like this:

Club	PJ	V	E	D	GF	GC	DG	Pts	Últimos 5				
1  Barcelona	38	26	9	3	90	36	54	87	–	✓	✗	✓	✓
2  Atlético Madrid	38	22	10	6	55	29	26	76	–	–	✗	✓	✓
3  Real Madrid	38	21	5	12	63	46	17	68	✗	✗	✓	✗	–
4  Valencia C. F.	38	15	16	7	51	35	16	61	✓	✓	✓	✗	✗
5  Getafe	38	15	14	9	48	35	13	59	–	✗	✓	✗	–

Each team with its played matches, wins, draws and losses. To do so, we had to make multiple SELECT lines and unify them into a single table. Due to this, there were some issues with the UNION operator of SQL, which identifies some columns as TEXT instead of INT when unifying. To solve this, we have used CAST (value AS bigint) inside our query.

- **What was the final score board for this season?**

```
SELECT team_name,
       MIN(played_matches) played_matches,
       MIN(wins) wins,
       MIN(draws) draws,
       MIN(losses) losses FROM (
(SELECT disputes.team_name, COUNT(*) AS played_matches, NULL AS wins, CAST(NULL AS bigint) AS
draws, CAST(NULL AS bigint) AS losses FROM match
      INNER JOIN disputes ON match.id_match = disputes.id_match
      GROUP BY disputes.team_name)
UNION
(SELECT disputes.team_name, NULL AS played_matches, COUNT(*) AS wins, CAST(NULL AS bigint) AS
draws, CAST(NULL AS bigint) AS losses FROM match
      INNER JOIN disputes ON match.id_match = disputes.id_match
      INNER JOIN team ON disputes.team_name = team.team_name
      WHERE (match.winner_team_ft = 'HomeTeam' AND team.stadium = match.place)
      OR (match.winner_team_ft = 'AwayTeam' AND team.stadium != match.place)
      GROUP BY disputes.team_name)
UNION
```

```

(SELECT disputes.team_name, NULL AS played_matches, NULL AS wins, COUNT(*) AS draws, CAST(NULL
AS bigint) AS losses FROM match
    INNER JOIN disputes ON match.id_match = disputes.id_match
    INNER JOIN team ON disputes.team_name = team.team_name
    WHERE (match.winner_team_ft = 'Draw' AND team.stadium = match.place)
    OR (match.winner_team_ft = 'Draw' AND team.stadium != match.place)
    GROUP BY disputes.team_name)
UNION
(SELECT disputes.team_name, NULL AS played_matches, NULL AS wins, CAST(NULL AS bigint) AS draws,
COUNT(*) AS losses FROM match
    INNER JOIN disputes ON match.id_match = disputes.id_match
    INNER JOIN team ON disputes.team_name = team.team_name
    WHERE (match.winner_team_ft = 'AwayTeam' AND team.stadium = match.place)
    OR (match.winner_team_ft = 'HomeTeam' AND team.stadium != match.place)
    GROUP BY disputes.team_name))
GROUP BY team_name
ORDER BY wins DESC

```

6. Transactions

In this section we explain the most important transactions that our database has:

- **Insert a new match with its associated goals, cards, and teams:** it is vital that each match has two teams associated with it, contained in the *disputes* table. For this to work, it makes sense to first add the teams to the relationship table and then add the result of the match and the goals/cards (if there's any). Therefore, we could specify that each match must have two teams associated with it and that their names should be different from each other. We could set here the atomicity and consistency properties since the transaction is cancelled if there aren't two teams or they have the same name.
- **Update the current status:** this operation is critical since the players' names appear on three different tables and can alter the stats of goals and cards for a player. This means that no two users should be altering the database at the same time (isolation) to avoid different inserts and also that there must be consistency between tables when it comes to these names.

In addition, these two transactions must be durable in the sense of time and even if the system experiences a failure.

7. Advanced features and Full Stack Development

Once having everything set up, we decided to create a visual interface to easily see our project's result. We deployed a container with the previous two images (Postgres and Notebook) and a new one for the front-end, for which we created an express server in JavaScript that connects to the Postgres server to retrieve information. Therefore, you can not only see the previous queries' results, but you can also make your own queries as the system is not static and evaluates everything in the server.

Spanish LaLiga 2028/19 Home Fixed Queries New Query

Start learning about spanish LaLiga 2018/2019 !



team_name	city	year	stadium	president
Real Madrid	Madrid	1902	Santiago Bernabéu	Florentino Pérez

Spanish LaLiga 2028/19 Home Fixed Queries New Query

Select one of the fixed queries to be evaluated by the server:

Query 3: Which are the top 5 players that received the highest number of yellow cards and from which teams are they?

player_name	team_name	yellow_cards
Éver Banega	Sevilla FC	16
Mario Gaspar	Villarreal CF	16
Ramiro Funes Mori	Villarreal CF	15
Álvaro González Soberón	Villarreal CF	15
Dani García	Athletic Club	13

Spanish LaLiga 2028/19 Home Fixed Queries New Query

Write a SQL query to be evaluated by the server:

SELECT * From card

Submit

id_card	type	card_minute	id_match	player_name
card1	Yellow	93	match1	José Luis Morales Nogales
card2	Yellow	82	match1	Antonio Luna Rodríguez
card3	Yellow	87	match2	Àlex Granell
card4	Yellow	86	match2	Rubén Alcaraz
card5	Yellow	41	match3	Guillermo Maripán