BLG 317E - Database Systems
CRN - 13508
Fall 2025

Transfermarkt

Team: *RocketTeam*

Team Members:
Furkan Kural - 150210056
İlke Başak Baydar - 150140709
Mustafa Çağşak - 150220060
Onat Barış Ercan - 150210075

Instructors: Kadir Özlem
Teaching Assistants: Burcu Kartal, Elif Yıldırım, Selahaddin Şentop, Hacer Akıncı

Contents

1 Introduction

Nowadays, the football industry is more than just a game played on the field. It's a worldwide market where billions of dollars' worth of financial transactions happen and enormous data are maintained.

Our Transfermarkt project aims to provide a platform that displays various statistics relating to matches, players, clubs, and transfers, using dataset from a website of the same name, and offering open access to and the ability to manage all data.

2 Technology Stack

Transfermarkt is built using following core technology stack.

Database Management Systems (DBMS): MySQL was selected as the relational database backend due of its dependability. Additionally, InnoDB storage engine was used specifically to maintain name consistency across different languages.

Backend: Python and Flask was used for backend framework.

Frontend: We used HTML5 for the structural layout and Bootstrap 5 as a CSS framework to ensure the application is responsive and visually consistent. In Addition, we used embedded data with Jinja2, handled dynamic operations with jQuery and jQuery UI, user-friendly dropdown menus with Choices.js and added those alert boxes with SweetAlert2.

3 Database Design

Entity Relationship (ER) Diagram can be seen in Figure 1. Due to the large number of attributes, the generated ER diagram does not include all attributes. Detailed ER diagrams will be shown separately for each table. Relational mapping can be also examined with ER Diagram with using Crow's Foot Notation in Figure 2. Referential integrity and cascading rules is shown in Figure 3.

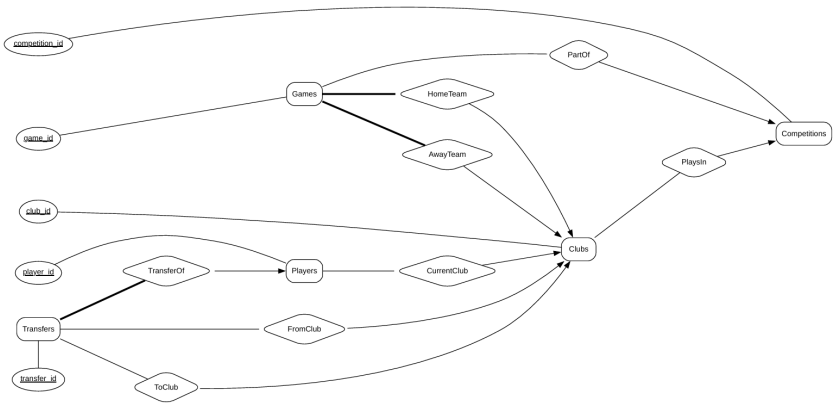

Figure 1: ER Diagram without Attributes

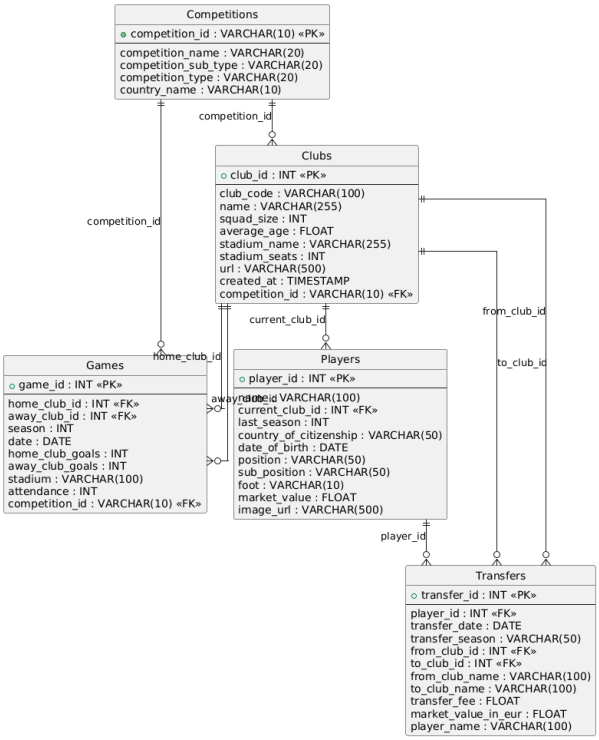

Figure 2: ER Diagram with Using Crow's Feet Notation

Table / Relation	**Constraint Strategy**	**Reasoning**
Games Table	`ON DELETE CASCADE` *(For home/away clubs & competition)*	**Strict Dependency:** A game fixture cannot logically exist without the participating teams or the league. Prevents orphan records.
Players Table	`ON DELETE SET NULL` *(For current_club_id)*	**Independence:** Players exist independently of clubs. If a club is deleted, the player remains in the system as a "Free Agent" (NULL).
Transfers Table *(Player Relation)*	`ON DELETE CASCADE` *(For player_id)*	**Intrinsic Data:** A transfer history record belongs to the player. If the player is removed, their personal history becomes irrelevant.
Transfers Table *(Club Relation)*	`ON DELETE SET NULL` *(For from/to clubs)*	**History Preservation:** Financial records (transfer fees) shouldbe preserved for accounting evenifa club entity is dissolved.
All Tables	`ON UPDATE CASCADE` *(For all Foreign Keys)*	**Data Consistency:** Ensures that ifa Primary Key (e.g., `club_id`) is modified, the change is automatically reflected across all references.

Figure 3: Referential Integrity and Cascading Rules

Detailed analyses for the tables used are given below.

3.1 Analysis of Transfers Table

When we focus on the transfer table in the ER diagram, the detailed view that emerges is shown in Figure 4.

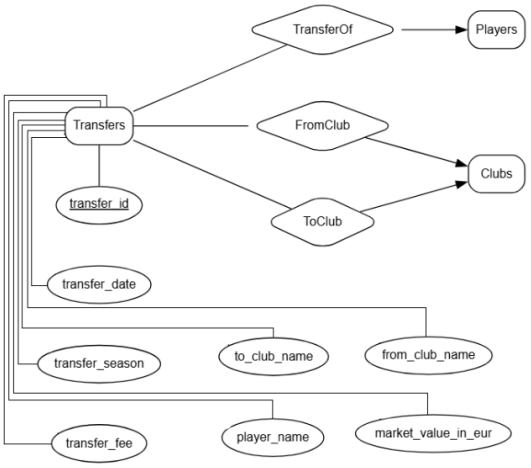

Figure 4: ER Diagram for Transfers Table

The Transfers table has an auto-increment **primary key** named "transfer_id". Since such a key does not exist in the source from which the dataset was obtained, the table is created during creation, making it easier to track existing data without gaps.

The Transfers table has **3 foreign keys**. One is the "player_id" taken from the Players table, and the other two are "from_club_id" and "to_club_id", which are formed from the "club_id" in the Clubs table. These foreign keys enable access to the tables from which they are obtained, thus implementing the **Third Normal Form (3NF)** approach which aims to reduce data duplication.

In addition, the Transfers table also contains **7 non-key attributes** which are "player_name", "to_club_name","from_club_name", "transfer_date", "transfer_season", "transfer_fee" and "market_value_in_eur". As can be seen, "player_name", "to_club_name" and "from_club_name" attributes, although also found in the Players and Clubs tables, were reused in the Transfers table. While this constitutes a **3NF** violation, the purpose of this **controlled denormalization** is to provide easier access for log operations. For example, being able to display these within the transfers module without needing a separate join operation to access a transferred player provides ease and speed of operation and is also user-friendly because the user does not need to know the transfer ID, for example, for CRUD operations.

The relationship between the Transfers and Players tables is **One-to-Many** because a player can have multiple transfer records. Similarly, there are two separate **One-to-Many** relationships between the Clubs and Transfers tables; because a club can transfer players many times.

As shown in Figure 3, the referential integrity and cascading rules of the Transfers table are as follows: Like all tables, the Transfers table also relies on the **ON UPDATE CASCADE** rule for all three foreign keys; the purpose of this is to maintain data consistency. There are two different methods for deleting a player. The "player_id" method works with the **ON DELETE CASCADE** rule because it is illogical to keep track of a player's transfer records when they are deleted from the system. However, for two foreign keys based on the "club_id" from the Clubs table, **ON DELETE SET NULL** is used because a team being deleted is not required automatically delete the transfer record; the relevant club (from club or to club information) is set to null.

3.2 Analysis of Clubs Table

The Clubs table is the central part of our database structure, connecting players, matches, and transfers. You can see the visual representation of these connections in Figure 5.

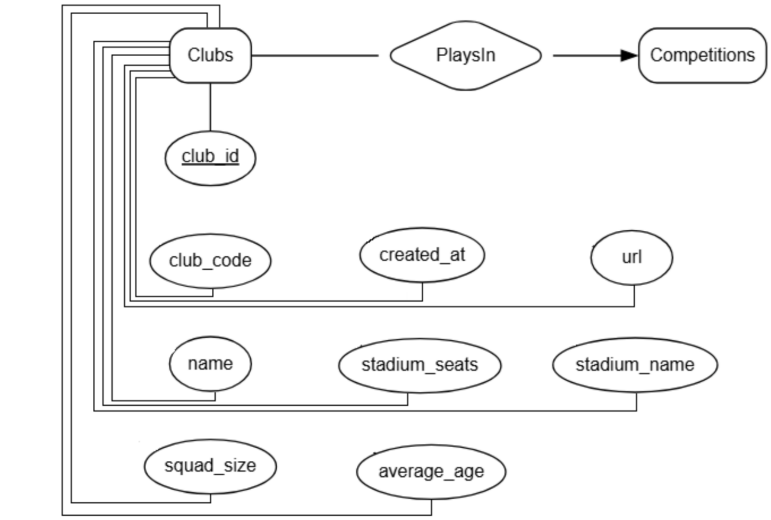

Figure 5: ER Diagram for Clubs Table

For the Primary Key, we chose **club_id**. We specifically decided not to use an auto-increment integer here. Since we are using real data from Transfermarkt, we wanted to keep the original IDs from the website. This makes it easier for us to check our data against the real website later.

An important detail in our implementation is how we handled the relationship with the Competitions table. In our SQL script, we did not define the foreign key immediately when creating the table. Instead, we used an **ALTER TABLE** statement at the end of the script to add the `competition_id` column and the constraint. We did this to avoid potential errors during the database creation process; it ensures that the Competitions table is fully created before we try to link the clubs to it.

For this relationship, we used the **ON DELETE SET NULL** rule. This means if a league is deleted, the clubs in that league are not deleted; they just remain in the system without a league assignment.

The Clubs table also has important relationships with other tables:

- **Players:** A one-to-many relationship, as a club has many players.

- **Transfers:** The table is referenced twice in the Transfers table (once as *from_ club*, once as *to_ club*), allowing us to track the full history of player movements.

Finally, we stored other details like `name`, `squad_size`, `average_age`, `stadium_name`, `stadium_seats`, and `url` to show detailed information on the club profile pages.

3.3 Analysis of Players Table

As shown in the ER diagram (Figure 6), the Players table uses `player_id` as the **Primary Key** and references `club_id` from the Clubs table as a **Foreign Key**. Other attributes stored in the table are: `name`, `position`, `sub_position`, `market_value`, `foot`, `last_season`, `date_of_birth`, `country_of_citizenship`, and `image_url`.

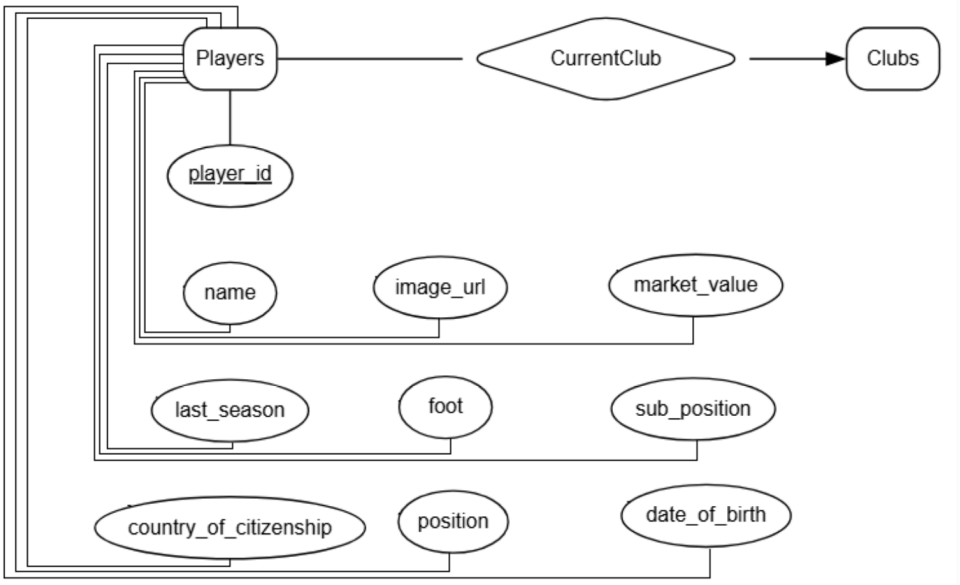

Figure 6: ER Diagram for Players Table

The first dataset had **Transitive Dependencies**; for example, columns like `current_club_name` and `domestic_competition_id`, which are dependent on `club_id`, were also present in the Players table. To apply the **Third Normal Form (3NF)** and remove this redundancy, we removed club-related details from this table and only kept `club_id` as a Foreign Key from the Clubs table.

This structure implements a **One-to-Many** strategy: a player is assigned to only one club, but a club can contain multiple players. The relationship between the Players table and other tables is designed as follows:

- When a club is deleted from the **Clubs table**, the `club_id` in the Players table becomes NULL (**ON DELETE SET NULL**). This provides that if a club is removed, the player data remains in the system, treating them as a free agent.

- For the relationship with the **Transfers table**, the `player_id` serves as a reference for the relevant player's transfer records. This allows the system to access the specific transfer history of a player.

3.4 Analysis of Games Table

The Games table is a central component of our database, as it stores the core data for every match event. As illustrated in the ER diagram (Figure 7), this table acts as a bridge between clubs and competitions. We used `game_id` as the **primary key** with an auto-increment feature to ensure each match has a unique and permanent identifier.

The most important aspect of this table is its relational structure. It maintains a dual relationship with the Clubs table: one for the **HomeTeam** and one for the **AwayTeam**. This design is necessary because every match requires two distinct teams, and both foreign keys (`home_club_id` and `away_club_id`) point back to the same Clubs table. Additionally, the `competition_id` foreign key links each game to a specific league or tournament, ensuring that matches are correctly categorized.

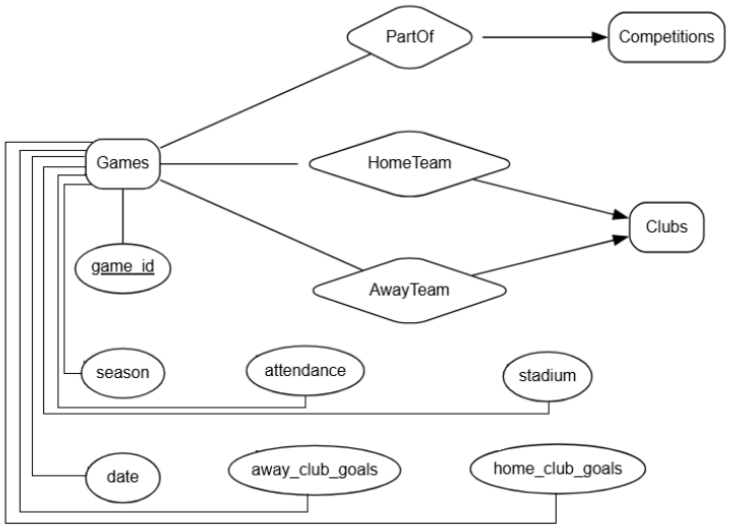

Figure 7: ER Diagram for the Games Table

In terms of attributes, the table stores essential match-day information such as `season`, `date`, and `stadium`. It also tracks performance metrics through `home_club_goals` and `away_club_goals`, which are crucial for calculating win/loss statistics in the frontend. To handle large crowds and stadium data, the `attendance` and `stadium` fields are included.

4 Website and Functionalities

Main page of Transfermarkt website is shown in Figure 8. This main page navigates to the Transfers, Players, Games and Clubs pages that each of them containing CRUD operations and statistics related to each table.

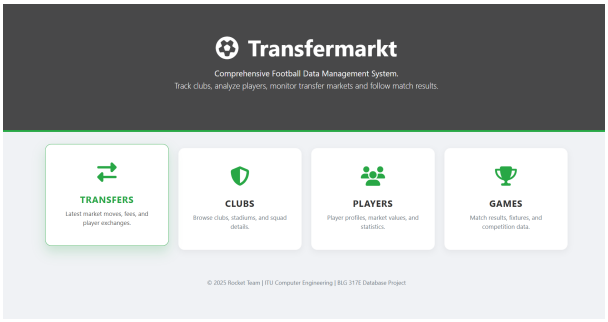

Figure 8: Main Page of Transfermarkt

4.1 Transfers Page

Clicking the Transfers card on the main page it is redirected to the Transfers homepage. The screenshot of this page is given below. As shown in the Figure 9, this page lists

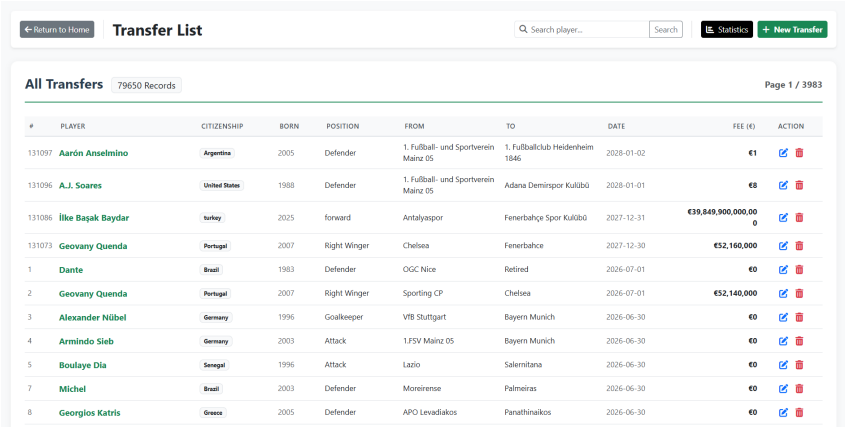

Figure 9: Transfers Page

all transfer records paginated from most recently.Pagination logic utilizes SQL LIMIT and OFFSET commands to retrieve records in batches of 20, avoiding full-table loads into memory.

The total number of transfers is also dynamically displayed next to the "All Transfers" heading. The Search box above allows you to search by player name. The list here is retrieved from the Players table via "player_id" and the backend constructs SQL queries conditionally, appending "WHERE ... LIKE" clauses only when a search parameter is provided.This query is also listed assuming the user started the search from the beginning. For example, if a player starts searching for a player named "John," the options will be listed as "J," "JO," and so on.

When the user hover over each player, information such as their photo and position is dynamically displayed from the Players table. Additionally, each row in the table contains buttons for performing Update and Delete operations.

Clicking the New Transfer button in the upper right corner opens an accordion-style insertion window within the page. Updating and adding new transfer have same restrictions for maintaining data consistency.

4.1.1 Checking Data Consistency

Several warning and error messages have been added to the add and update sections. These measures aim to ensure data consistency, both during player and club information selection via the dropdown menu and during manual entry. Examples of these messages can be seen in the Figure 10.

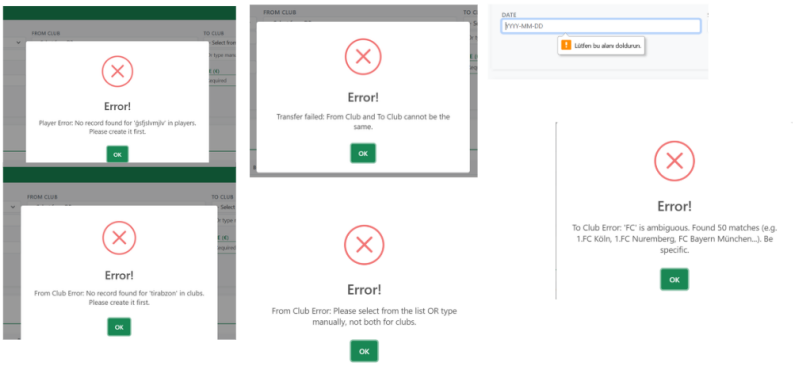

Figure 10: Error Messages About Adding and Updating Transfers

The situations in which these messages appear can be listed as follows:

- **Entity Existence Verification:**Transfer can only be created for existing Players and Clubs.Through the find_entity helper function, which queries the database to verify the entity's existence. If query returns no results, it gives error.
- **Logical Consistency:** If from_club_id and to_club_id are identical, the transaction is rejected to prevent redundant records.
- **Mandatory fields:** Enforces non-null constraints on Date and Fee fields, requiring a strict YYYY-MM-DD date format.
- **XOR Logic for Inputs:** Users can either select an entity from a dropdown ID list OR type a name manually, but not both simultaneously.
- **Ambiguity Resolution:** If a user inputs a generic term (e.g., "FC" or "John"), the system detects multiple matches (COUNT $>$ 1) and halts the process, forcing the user to be specific to prevent incorrect Foreign Key linkage.

When the user clicks the update button, the page that opens displays that the Player Name/Player ID field cannot be changed during updates, which is important to avoid ambiguity.

Additionally, upon a successful insert/update, the system checks the transfer_date. If the transfer date is today or in the future, it executes a secondary UPDATE on the Players table (current_club_id), ensuring the player's active profile reflects the move immediately.

When user clicks the delete button, The system warns the user that this action is irreversible, because this is hard delete operation.Delete operation is wrapped in a transaction block with auto-rollback to prevent data corruption in its API.

4.1.2 Statistics Page

When clicking the Statistics button on the Transfer page will redirect the user to the statistics page, which contains some statistics about transfers. This module leverages advanced SQL aggregation functions to generate financial insights without heavy application-level processing. Screenshot about this page as follows.

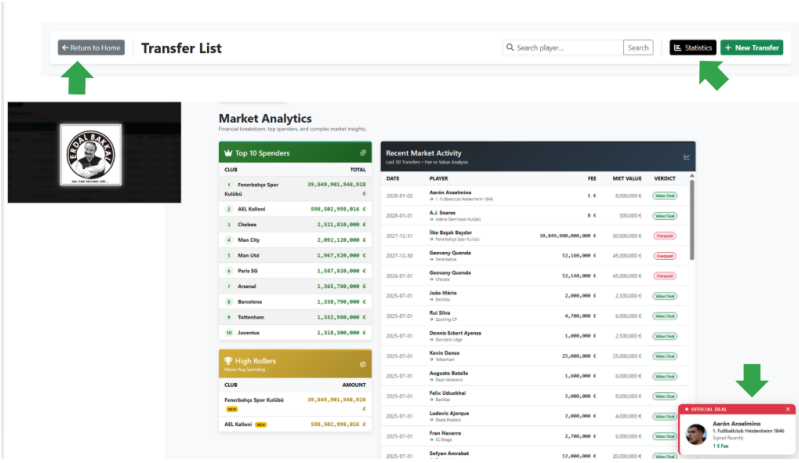

Figure 11: Statistics Page of Transfers

Three basic statistics have been prepared in this section and can be summarized as follows:

- **Latest Market Activity:** This module lists the 50 most recent transfers and performs a "Fee vs. Value Analysis". By calculating the difference between a player's market_value and their transfer_fee, the system dynamically generates a "Verdict" label—categorizing deals as either a "Value Deal" or "Overpaid" to evaluate transaction efficiency.

- **Top 10 Spenders:** This summary table ranks the most active clubs in the market based on their cumulative financial output. It utilizes SQL SUM() aggregation and GROUP BY clauses to consolidate total transfer fees, providing a clear view of which entities are leading the market in spending

- **High Roller Clubs:** This advanced metric identifies market outliers using a complex nested subquery logic. Rather than using a static filter, it establishes a dynamic threshold by calculating the overall market average and filtering for clubs

whose spending exceeds 20% of that average.This highlights clubs that are spending significantly more than the norm, relative to current market conditions. This query as follows:

```sql
SELECT
    t.to_club_name,
    SUM(t.transfer_fee) AS total_spent,
    COUNT(*) AS transfer_count
FROM transfers t
WHERE t.transfer_fee > 0
GROUP BY t.to_club_name
HAVING total_spent > (
    SELECT AVG(club_total) * 0.2
    FROM (
        SELECT SUM(transfer_fee) AS club_total
        FROM transfers
        WHERE transfer_fee > 0
        GROUP BY to_club_name
    ) AS subquery_table
)
ORDER BY total_spent DESC
LIMIT 5;
```

Listing 1: Example: Complex Query for Identifying High Roller Clubs

4.2 Clubs Page

The Clubs module is the section where we manage the team data. While it looks like a standard user interface, the backend logic relies on specific database strategies to ensure data integrity and query performance.

4.2.1 Club Management Panel

This page lists all the clubs registered in our system. You can see the interface in Figure 12.

When fetching data for this list, our main concern was handling the relationship between `Clubs` and `Competitions`. We implemented a **LEFT JOIN** in our SQL query instead of a standard INNER JOIN.

We made this choice for data safety. Some clubs in our database might not be assigned to a league yet (meaning their `competition_id` is NULL). If we had used an INNER JOIN, these clubs would effectively disappear from the list because they wouldn't match the join condition. By using LEFT JOIN, we ensure that every club is visible in the management panel, regardless of its league status.

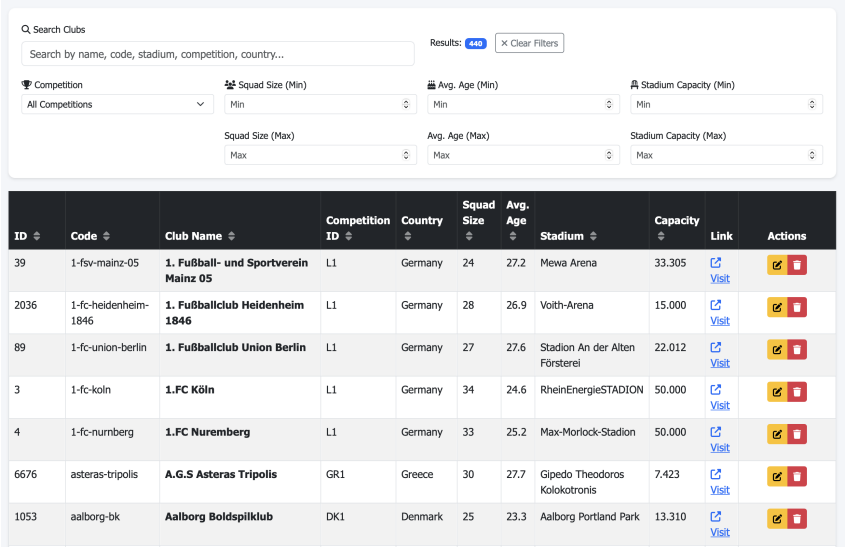

Figure 12: Club Management Panel

4.2.2 Club Details and Transfer History

The Club Details page, shown in Figure 13, presents the most complex database challenge in this module: displaying the "Transfers by Season" history.

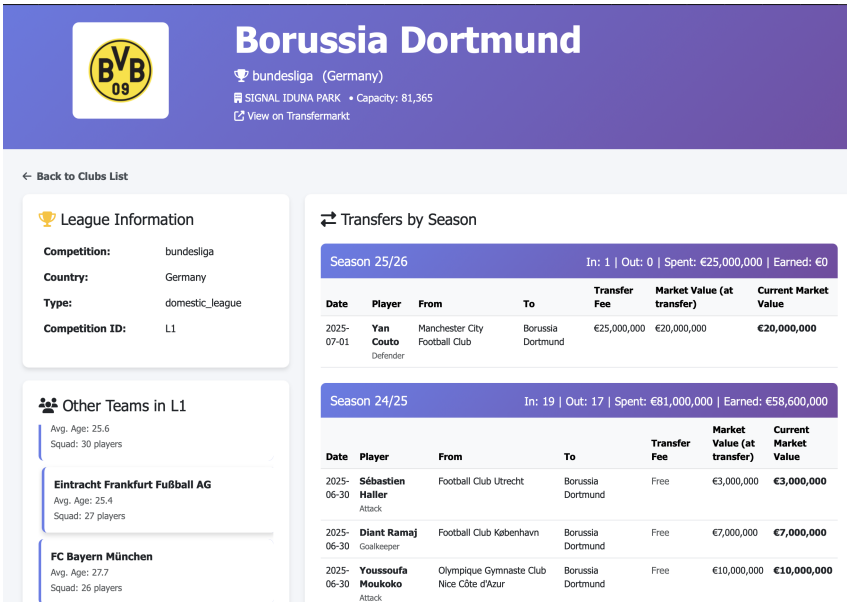

Figure 13: Club Details Page with Transfer History

The difficulty here is that a transfer record in the database only contains the IDs of the teams (`from_club_id` and `to_club_id`). To display meaningful names to the user, we needed to join the `Clubs` table. However, since a transfer involves two different clubs (a sender and a receiver), we had to join the `Clubs` table **twice** in the same query.

To achieve this, we used SQL aliases (`from_club` and `to_club`). This allows us to fetch both club names in a single database request, preventing the performance issue known as the "N+1 query problem," where the server sends a separate query for every single row.

Here is the complex join query we used for this feature:

```sql
SELECT
    t.transfer_id,
    t.transfer_season,
    t.transfer_date,
    t.player_name,
    t.transfer_fee,
    t.market_value_in_eur AS transfer_market_value,
    p.market_value AS current_market_value, -- Joined from Players
    from_club.name AS from_club_name,       -- 1st Join (Alias: from_club)
    to_club.name AS to_club_name            -- 2nd Join (Alias: to_club)
FROM Transfers t
LEFT JOIN Players p ON t.player_id = p.player_id
LEFT JOIN Clubs from_club ON t.from_club_id = from_club.club_id
LEFT JOIN Clubs to_club ON t.to_club_id = to_club.club_id
WHERE (t.from_club_id = %s OR t.to_club_id = %s)
ORDER BY t.transfer_season DESC, t.transfer_date DESC
LIMIT 50;
```

Listing 2: Multi-Join Query with Aliases for Transfer History

Overall, the Clubs module demonstrates how we used advanced SQL features to solve real-world data presentation problems. By carefully designing our queries with multiple joins and aliases, we ensured that the application handles complex relationships between clubs, players, and transfers efficiently. This approach allowed us to provide a seamless experience for the user while maintaining data consistency in the background.

4.3 Players Page

The Players module serves as the main directory for the 30,000+ athletes in the database. It is split into two page: management panel for searching and CRUD operations, and a detailed profile page for analytics.

4.3.1 Listing and Management

The main interface features a robust **Filtering and Sorting** system. Users can filter results by position, sub-position (which updates dynamically based on the selected position), country, preferred foot, and age range. Also, the list can be sorted by name, market value, or age in ascending or descending order.

The interface and its features can be seen in Figure 14.

The table displays essential data including the player's name (onclick takes you to the player details page), current club, position, and market value. To handle the vast dataset,we created real-time search and a pagination mechanism that displays **50 players each page**.

CRUD Operations are handled via a collapsible form system:

- **Add Player:** A collapsible form allows users to create new players with validation for required fields.
- **Edit Player:** An inline edit mode allows updates to specific fields while keeping non-editable fields (like ID) read-only to preserve integrity.

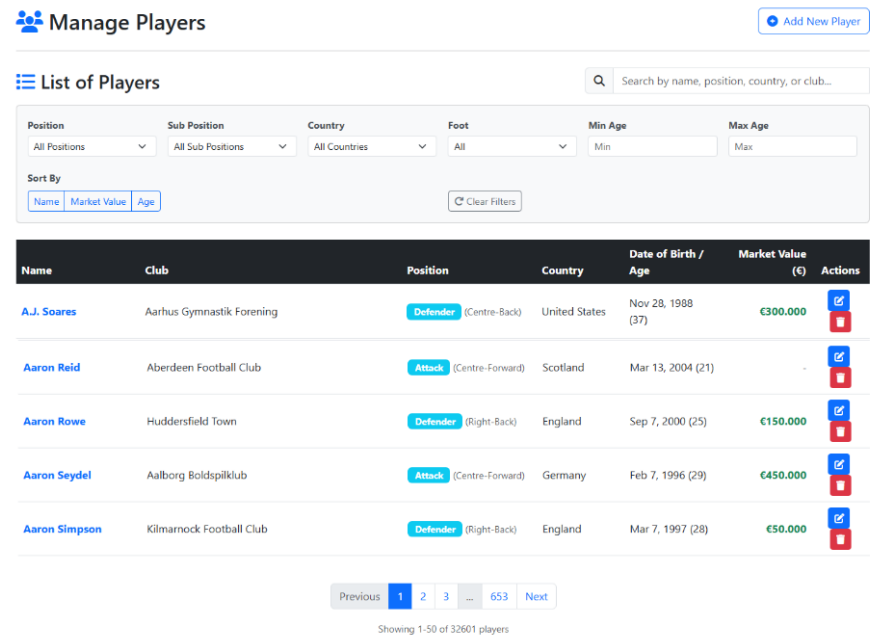

Figure 14: Players Management Interface and Filtering System

- **Delete Player:** Removal requires confirmation. This is a cascading hard delete that also removes the player's associated transfer history.

4.3.2 Player Details and Analytics

Clicking on a player's name takes us to the **Player Details Page**, which looks like the Figure 15 shown. This page features a "Player Information Panel" that displays biographical data, including a profile image, position, foot preference, and current status.

A key feature of this page is the **Comparative Analytics** section. Using complex SQL queries, we compare the player's statics against their current club and league. To provide correctness, we implemented a logic filter (`last_season >= 2023`) to exclude inactive players or those from non-European leagues from the average calculations.

Age Comparison: We calculate the weighted average age of the league the player plays in.

```sql
SELECT
    SUM(c2.average_age * c2.squad_size) / SUM(c2.squad_size) AS
   league_avg_age
FROM clubs c2
WHERE c2.competition_id = %s
  AND c2.average_age IS NOT NULL
  AND c2.squad_size > 0
```

Listing 3: Weighted League Average Age Calculation

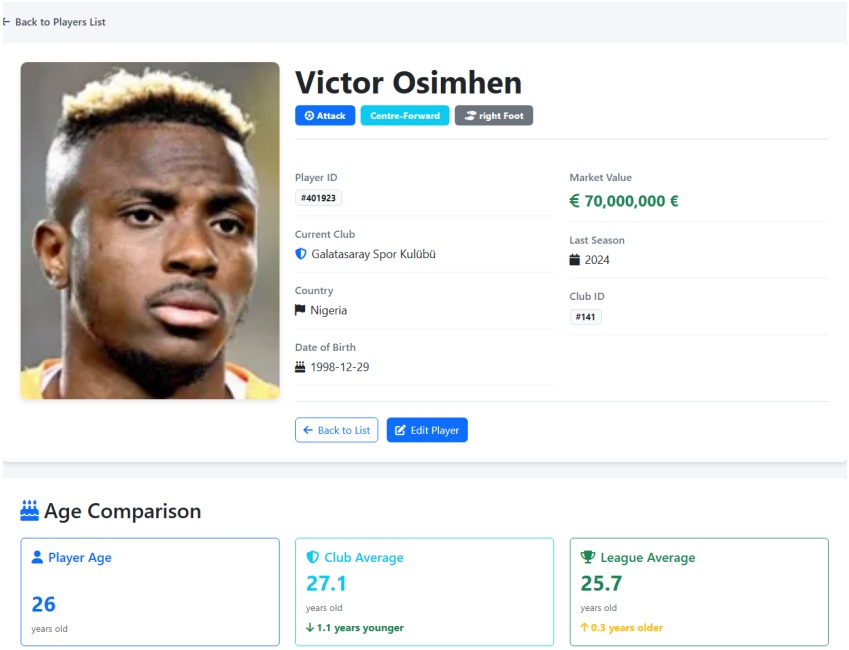

Figure 15: Player Details Page with Analytics and Transfer History (Part I)

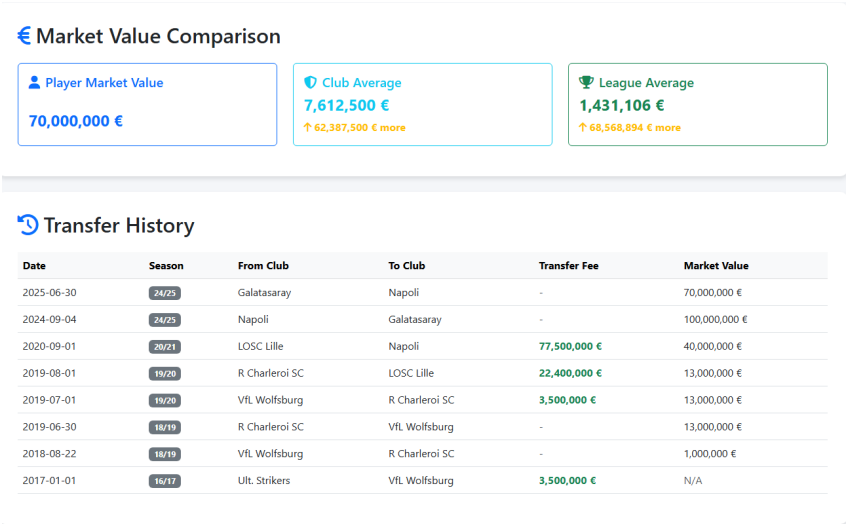

Figure 15: Player Details Page with Analytics and Transfer History (Part II)

Market Value Comparison: We also compare the player's market value against the league average, helping visualize if a player is above or below the standard for their competition.

```sql
SELECT AVG(p2.market_value) AS league_avg_mv
FROM players p2
INNER JOIN clubs c2 ON p2.current_club_id = c2.club_id
WHERE c2.competition_id = %s
  AND p2.market_value IS NOT NULL
  AND p2.last_season >= 2023
```

Listing 4: League Average Market Value Calculation

Finally, the page renders the full **Transfer History** of the player in chronological order, fetching data from the Transfers table to show their career history.

4.4 Games Page

The Games module is one of the most dynamic parts of our application. It allows users to track historical match data, manage game records, and perform detailed performance comparisons between teams.

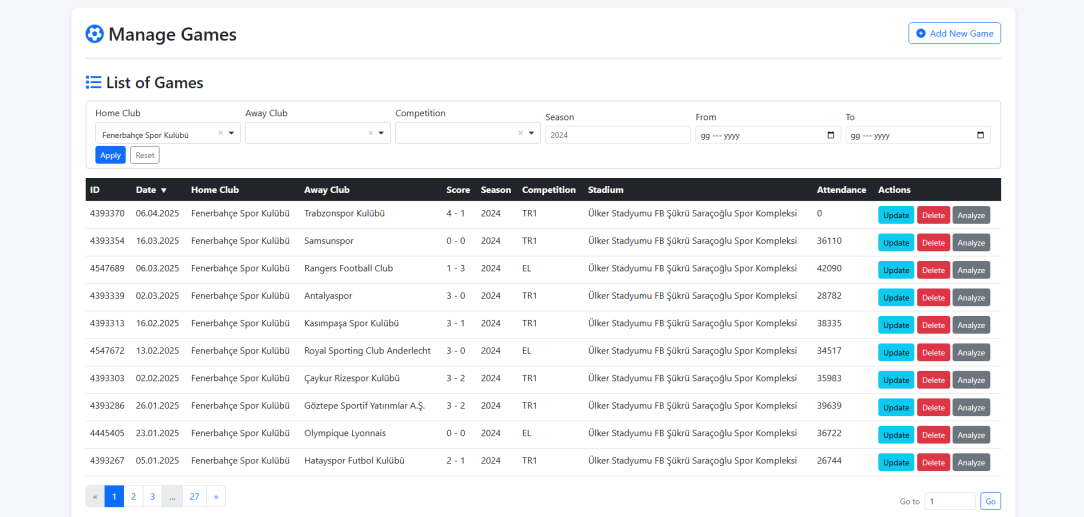

Figure 16: Games Management Interface and Filtering System

4.4.1 Listing and Advanced Filtering

To provide a smooth user experience, the Games page lists all matches in a clear table format. Since the dataset is quite large, we implemented an **Advanced Filtering** section. Users can filter the games by:

- **Home and Away Clubs:** Using searchable dropdowns to find specific matchups.
- **Season and Competition:** To view matches from a particular year or league (e.g., TR1, EL).
- **Date Range:** Filtering matches between two specific dates using a calendar picker.

Furthermore, the table features a comprehensive **sorting system**. Except for the ID column, users can click on any column header to sort the data in ascending or descending order. This allows the match list to be organized by **Date**, **Home/Away Club Names**, **Score**, **Stadium**, or **Attendance**. Technically, the backend handles this by dynamically adding an `ORDER BY` clause to the SQL query based on the user's selection.

The backend handles these filters by dynamically building SQL queries. We also used **Pagination** (as seen at the bottom of Figure 16) to ensure the page loads quickly by only fetching 10-20 records at a time.

4.4.2 Data Management and Integrity

Each row in the match list includes "Update" and "Delete" buttons for quick management. When adding or updating a game, the system relies on the following database logic:

- **Auto-Increment Keys:** The `game_id` is automatically generated, so the user doesn't need to track ID numbers.
- **Relational Constraints:** Our SQL structure uses `ON DELETE CASCADE` for foreign keys (`home_club_id`, `away_club_id`, etc.). This means if a competition is removed, all related games are automatically cleaned up to prevent "ghost" data.
- **Score Tracking:** The system separately stores `home_club_goals` and `away_club_goals`, allowing us to calculate win/loss statistics instantly.

4.4.3 Head-to-Head (H2H) Analytical Sidebar

The most distinctive feature of the Games module is the **"Analyze"** button. When clicked, it opens a sidebar (Figure 17) that performs a deep-dive analysis of the rivalry between the two selected clubs.

This sidebar aggregates data from multiple tables to show:

- **Overall Rivalry:** A summary of total wins for each team and the number of draws.
- **Club Snapshots:** A side-by-side comparison of squad size, total market value, and average age of both clubs.
- **Last 5 Meetings:** The most recent results between these two teams, providing a "form guide."
- **Historical Highlights:** Identification of the "Biggest Win" and "Most Expensive Transfer" involving these two clubs.

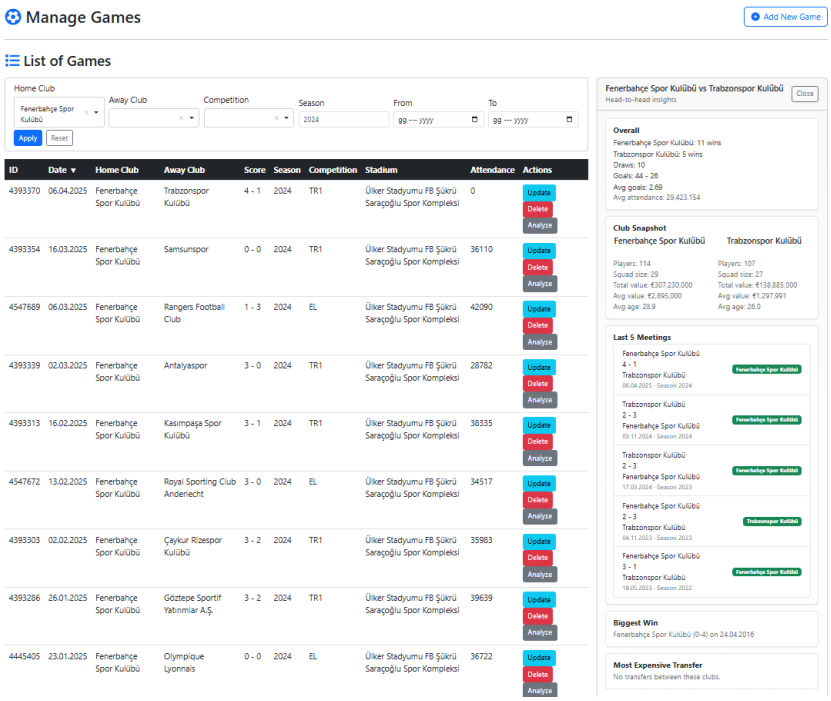

Figure 17: Head-to-Head (H2H) Insights Sidebar

This feature showcases our ability to use complex SQL `JOIN` operations and `GROUP BY` clauses to turn raw data into meaningful insights for the user.

5 Difficulties and Solutions

The initial problem we encountered was that, due to the size of our dataset, we could not utilize the paid cloud services. Furthermore, since each of us has different work hours and class schedules, turning one of our computers into a server would have made things even more difficult. Therefore, each of us had to work on our own localhost MySQL database. To ensure infrastructure compatibility, a common SQL schema was created and shared via GitHub and configuration was standardized using environment variables files (.env) for managing database information.

It was challenging to accurately convert the source data to the relational database schema because it was in raw CSV format. Manual data entry was not feasible due to the huge amount of data, and using a single approach also increased the possibility of inconsistent data. Two distinct data transfer systems were developed, supporting one another and catering to distinct situations. These are:

- The SQL script -insert_data_from_csv_to_db.sql
- The Python script - load_tables_from_csv.py

Due to the working at different times and in different local environments, there was a possibility of code and database structure conflicts. The Flask Blueprint design was used to divide the project into modular components as Transfers, Players, Clubs, Games and this allowed each member to work independently on their own defined table and backend logic, reducing disagreements that arose during the integration of the developed modules into the main project.

6 Individual Contributions

Details of the specific efforts and tasks completed by each team member is given below.

Member	**Contribution**
İlke Başak Baydar	Architected the database schema and implemented SQL queries for financial analytics. Developed end-to-end features for Transfer Management, Statistics, and Edit Transfer modules. Designed the application's visual identity, responsive Main Page layout and interactive widgets.
Furkan Kural	Built a comprehensive club management system with advanced filtering, intelligent search with relevance-based ranking. Designed a responsive web interface featuring real-time statistics dashboards, interactive club profiles, and seamless list-to-detail navigation. Developed complex SQL queries utilizing multi-table joins across Clubs, Competitions, Players, and Transfers tables to aggregate transfer statistics and market value analytics
Onat Barış Ercan	Built player management system with filtering, sorting, search, and pagination. Designed responsive UI with dynamic dropdowns, loading states, and seamless list-to-detail navigation. Developed complex SQL queries utilizing multi-table joins, subqueries and aggregate functions to extract comprehensive player analytics
Mustafa Çağşak	Developed game management features, including filtering, sorting, pagination, and expanded match data. Built a head-to-head analytics panel with recent meetings, win/draw summaries, biggest win, and transfer highlights that refresh after edits. Standardized and refined the UI/UX with reusable dropdowns, responsive layout and aligned add/edit/filter interactions.