


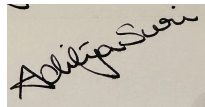


| | |
|---------------------------------|------------------|
| Group ID | 3 |
| Group leader | Asmita Chhabra |
| Group members other than leader | Nandika Aggarwal |
| | Aditya Suri |
| | Anushka Doshi |
| Project type | Demo B |

Declaration

We, undersigned, the group members of group 3, declare the following.

1. The project is done by us.
2. We have cited references and resources that we used while implementing the project.
3. We know that if plagiarism is detected, then it will be reported to the disciplinary committee.

| | |
|------------------|--|
| Asmita Chhabra |  |
| Nandika Aggarwal |  |
| Anushka Doshi |  |
| Aditya Suri |  |

A.Problem statement

We plan to analyze and support extensive querying and exploration for a football database sourced from Transfermarkt. The dataset is composed of multiple CSV files with information on competitions, games, clubs, players, and appearances.

The objective of this project is to develop a **football analytics window** using Python's Tkinter for the user interface and a MySQL database for backend data storage. This tool enables users to interactively analyze, search, and visualize various aspects of football players' and clubs' performance, market behavior, and career trajectories.

B.Database setup

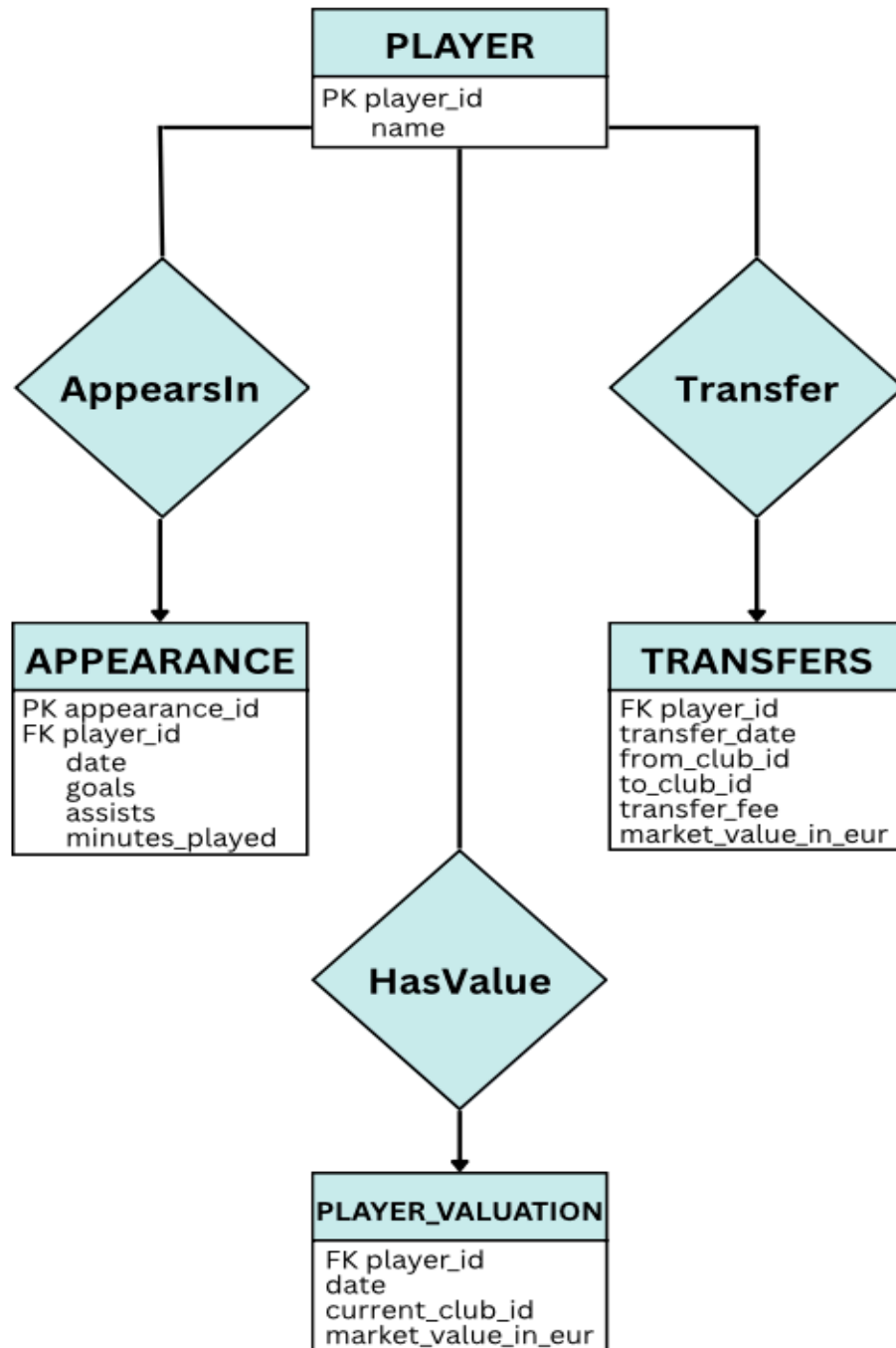
This database is designed to manage and analyze information related to football (soccer) player statistics, transfers, clubs, and related entities.

Real-World Facts and Requirements Considered in the Database Design

1. **Each player plays for exactly one club at a time.**
 - A player cannot be simultaneously registered with multiple clubs.
 - Foreign key from **players** to **clubs** ensures each player has a valid current club.
2. **Clubs can have zero or many players.**
 - A club may have a full roster or may be newly created with no players yet.
3. **Each transfer record must be tied to a player and (optionally) a previous club.**
 - Transfers may involve free agents (no previous club), so previous club can be **NULL**.
4. **Transfer fees can be zero (free transfer), fixed amount, or undisclosed.**
 - A **fee** field allows zero or specific values; a flag for **undisclosed** can be added if needed.
5. **Each player has a unique ID, and multiple attributes (name, position, nationality, age, etc.)**
 - To uniquely identify players and enable analytics based on multiple features.
6. **Players can receive multiple red cards and yellow cards over time.**
 - Card data must be normalized and linked via a separate table or within a **match_performance** table.
7. **Each club must belong to a country and a league.**
 - To track regional competitions and make location-based analyses possible.
8. **Age and birthdate must be consistent.**
 - Age is often derived from the birthdate for dynamic calculations.
9. **Player IDs must remain unique across seasons.**
 - Even if a player changes clubs or retires, the player ID should not be reused.
10. **Player statistics like goals, assists, matches played, etc., can vary seasonally.**
 - Seasonal data is often stored separately for detailed performance tracking.
11. **Transfers can occur only during transfer windows.**

- Real-world timing constraints can be reflected via a `transfer_date` and `season` field.
12. **Players can change clubs multiple times in their career.**
 - The database must support a **many-to-one** relationship between transfers and players.
 13. **Player positions are standardized (Goalkeeper, Defender, Midfielder, Forward).**
 - Ensures consistent filtering and analysis in UI or analytics tools.
 14. **Salary, market value, and contract duration may vary by club and over time.**
 - These are dynamic attributes that may require separate historical tracking tables.
 15. **Clubs may undergo name changes or relocations.**
 - Club identity should be tracked using unique IDs, not names alone.

E-R Model



Functional dependency

Normalization

1. Normalization of the table: Club

Table structure:

club_id

club_code

name

Domestic_competition_id

First Normal Form (1NF)

1NF requires that each cell contains only atomic (indivisible) values and each record is unique.

- All fields contain atomic values (no lists, commas, or sets).
- Each row has a unique `club_id`.

Therefore, the table is in 1NF.

Second Normal Form (2NF)

2NF requires that the table is in 1NF and that all non-prime attributes (non-primary key columns) are fully functionally dependent on the entire primary key.

- Primary Key: `club_id`
- Functional Dependencies:
 - `club_id` → club_code, name, domestic_competition_id`

Therefore,

- `club_id` is a single-column primary key.
- All other attributes depend entirely on `club_id` (not partially).

Therefore, the table is in 2NF

Third Normal Form (3NF)

3NF requires that the table is in 2NF and there are no transitive dependencies i.e., non-prime attributes should not depend on other non-prime attributes.

Checking for transitive dependencies:

- `club_code`, `name`, and `domestic_competition_id` all depend directly on `club_id`.
- None of them depend on each other.

Therefore it is in the third NF.

2. Normalization of the table: competition

Table structure:

competition_id
competition_code
name
sub_type
type
country_id
country_name
domestic_league_code
confederation

First Normal Form (1NF)

1NF requires that all attributes must contain only atomic values and each row must be unique.

Analysis:

- All columns contain single, indivisible values.
- There are no repeating values.
- Each row represents one competition.

Conclusion: The table is in 1NF.

Second Normal Form (2NF)

2NF requires that the table is in 1NF and that all non-key attributes are fully functionally dependent on the entire primary key.

Assumption: competition_id is the primary key.

Functional dependencies:

competition_id → competition_code, name, sub_type, type, country_id, country_name, domestic_league_code, confederation

Analysis:

- All attributes depend on competition_id alone (which is a single-column primary key).
- No partial dependencies exist.

Conclusion: The table is in 2NF.

Third Normal Form (3NF)

3NF requires that the table is in 2NF and that there are no transitive dependencies.

Observation:

- country_id → country_name

This implies a transitive dependency:

competition_id → country_id → country_name

To remove the transitive dependency, we should separate the country information into a different table.

Revised structure:

1. competition table: competition_id, competition_code, name, sub_type, type, country_id, domestic_league_code, confederation
2. country table: country_id, country_name

Conclusion: The original table is not in 3NF because country_name is transitively dependent on the primary key.

3. Normalization of the table: `player`

Table structure:

player_id
first_name
last_name
name
last_season
current_club_id
player_code
country_of_birth
city_of_birth
country_of_citizenship
date_of_birth
sub_position
position
foot
height_in_cm
contract_expiration_date
agent_name

First Normal Form (1NF)

All attributes must be atomic and each row should be unique.

Analysis:

- Every field contains atomic values (no repeating groups or multi-valued attributes).
- Each row is uniquely identified by `player_id`.

Conclusion: The table satisfies 1NF.

Second Normal Form (2NF)

Table must be in 1NF and all non-key attributes must depend on the full primary key.

Observation:

- Primary key is a single column: `player_id`.
- All other attributes are directly dependent on `player_id`.

Conclusion: The table satisfies 2NF since there are no partial dependencies.

Third Normal Form (3NF)

The table must be in 2NF and there should be no transitive dependencies.

Looking for transitive dependencies

Example:

- $\text{city_of_birth} \rightarrow \text{country_of_birth}$
- $\text{country_of_citizenship} \rightarrow$ likely related to country details.

To achieve 3NF:

- Create a separate `country` table: `country_id`, `country_name`
- Create a separate `city` table: `city_id`, `city_name`, `country_id`

A possible revised structure (suggested for 3NF):

1. player (`player_id`, `first_name`, `last_name`, `name`, `last_season`, `current_club_id`, `player_code`, `date_of_birth`, `sub_position`, `position`, `foot`, `height_in_cm`, `contract_expiration_date`, `agent_name`, `country_of_birth_id`, `city_of_birth_id`, `country_of_citizenship_id`)
2. country (`country_id`, `country_name`)
3. city (`city_id`, `city_name`, `country_id`)

Conclusion: The current structure is not in 3NF due to possible transitive dependencies.

4. Normalization of the table: `player_valuation`

Table structure:

`player_id`

Date

`market_value_in_eur`

`Current_club_id`

`Player_club_domestic_competition_id`

First Normal Form (1NF)

All values in the table are atomic. There are no repeating groups. Each row represents a unique entry

Conclusion: The table is in 1NF.

Second Normal Form (2NF)

The assumed composite primary key is (player_id, date). All non-key attributes (market_value_in_eur, current_club_id, player_club_domestic_competition_id) depend on the full key (not just a part of it). So, there are no partial dependencies.

Conclusion: The table is in 2NF.

Third Normal Form (3NF)

We check for transitive dependencies.

player_club_domestic_competition_id depends on current_club_id (not directly on the full key).

This creates a transitive dependency via current_club_id.

To remove this, we should keep only current_club_id in this table and get the domestic competition id from the club table when needed.

Conclusion: The table is not in 3NF due to a transitive dependency.

To normalize it to 3NF:

- Remove player_club_domestic_competition_id from this table.

Use the club table (linked via current_club_id) to access domestic_competition_id.

C.Database exploration

Queries

1. Top N Goal Scorers

Main Query:

Fetches players with highest total goals, with optional filters.

```
SELECT a.player_name,  
       SUM(a.goals) AS total_goals,  
       p.country_of_citizenship,  
       c.name AS club_name,  
       YEAR(a.date) AS season  
FROM appearance a  
LEFT JOIN player p ON a.player_id = p.player_id  
LEFT JOIN club c ON a.player_club_id = c.club_id  
WHERE a.goals IS NOT NULL  
GROUP BY a.player_name, p.country_of_citizenship, c.name, season  
ORDER BY total_goals DESC  
LIMIT {n}
```

Filter Sub-Queries:

Used for dropdowns for filtering:

Club filter : `SELECT DISTINCT name FROM club WHERE name IS NOT NULL`

Seasons filter: `SELECT DISTINCT YEAR(date) FROM appearance WHERE date IS NOT NULL ORDER BY YEAR(date)`

Nationality filter: `SELECT DISTINCT country_of_citizenship FROM player WHERE country_of_citizenship IS NOT NULL`

2. Top N Aggressive Players

Main Query:

Finds players with most yellow/red cards.

```
SELECT  
  a.player_name,  
  SUM(a.{card_type}) AS total_cards,  
  p.country_of_citizenship,  
  c.name AS club_name,
```

```
YEAR(a.date) AS season
FROM appearance a
JOIN player p ON a.player_id = p.player_id
JOIN club c ON a.player_club_id = c.club_id
WHERE a.{card_type} IS NOT NULL
GROUP BY a.player_name, p.country_of_citizenship, c.name, season
ORDER BY total_cards DESC
LIMIT {n}
```

3. Player Information

Main Query:

Retrieves detailed player profile (position, nationality, club, etc.).

```
SELECT
    p.name AS player_name,
    p.position,
    p.sub_position,
    p.country_of_citizenship,
    p.country_of_birth,
    p.city_of_birth,
    c.name AS current_club
FROM player p
LEFT JOIN club c ON p.current_club_id = c.club_id
WHERE p.name LIKE %s
```

4. Player Transfer Flowchart

Main Query:

Shows a player's transfer history between clubs.

```
SELECT from_club_name, to_club_name, transfer_season  
FROM transfers  
WHERE player_name LIKE %s  
ORDER BY transfer_season
```

5. Player Club History

Main Query:

Counts how many clubs a player has played for.

```
SELECT  
    p.name AS player_name,  
    COUNT(DISTINCT t.to_club_id) AS clubs_played_for  
FROM player p  
JOIN transfers t ON p.player_id = t.player_id  
WHERE p.name LIKE %s  
GROUP BY p.player_id, p.name
```

6. Player Valuation Trend

Subquery 1:

Finds the player's ID from their name.

```
SELECT player_id FROM player WHERE name LIKE %s
```

Subquery 2:

Fetches historical market value data.

```
SELECT date, market_value_in_eur
FROM player_valuation
WHERE player_id = %s
ORDER BY date
```

7. Club Spending Analysis

Main Query:

Calculates a club's transfer spending, revenue, and net balance.

```
SELECT
    c.name AS club_name,
    SUM(CASE WHEN t.to_club_id = c.club_id THEN t.market_value_in_eur ELSE 0 END) AS spending,
    SUM(CASE WHEN t.from_club_id = c.club_id THEN t.market_value_in_eur ELSE 0 END) AS revenue,
    SUM(spending - revenue) AS net_spending
FROM club c
LEFT JOIN transfers t ON c.club_id IN (t.to_club_id, t.from_club_id)
WHERE c.name LIKE %s
GROUP BY c.club_id, c.name
```

8. Oldest Players

Main Query:

Lists the oldest players by age.

```
SELECT
    name,
    country_of_citizenship,
    date_of_birth,
    TIMESTAMPDIFF(YEAR, date_of_birth, CURDATE()) AS age
FROM player
WHERE date_of_birth IS NOT NULL
ORDER BY age DESC
LIMIT {n}
```


User interface

| Query Purpose | UI Customization (User Filters) | Output |
|---------------------------|--|---|
| Top N Goal Scorers | - Enter N (number of players) - Filter by: Club, Season, Nationality | Top scorers with goals, nationality, club, and season. |
| Top Aggressive Players | - Enter N (number of players) - Select card type: yellow_cards or red_cards | Players with most cards, nationality, club, and season. |
| Player Information Search | - Enter player name | Detailed player profile (position, birth info, current club). |
| Player Transfer Flowchart | - Enter player name | List of transfers (from/to clubs + season). |
| Player Club History | - Enter player name | Total number of clubs the player has played for. |

DTSC221 Course Project Report
AY 2024-2025, UG SEM 4
Faculty: Dr. Girija Limaye
Group ID: 3

| | | |
|-------------------------------|---------------------|---|
| Player Valuation Trend | - Enter player name | Line graph of market value over time. |
| Club Spending Analysis | - Enter club name | Total spending, revenue, and net transfer balance for the club. |
| Oldest Players | - Enter N | List of oldest players (name, age, nationality, birth date). |

Demo video

 Demo Video Group 3.mov

Implementation

| Component | Technology |
|-----------|------------|
|-----------|------------|

| | |
|----|----------------------------------|
| OS | Cross-platform (Win/macOS/Linux) |
|----|----------------------------------|

| | |
|----------|-----------|
| Database | MySQL 8.0 |
|----------|-----------|

| | |
|--------------|------------------|
| UI Framework | Tkinter (Python) |
|--------------|------------------|

| | |
|---------------|---------------------------|
| Visualization | Matplotlib + mplotcursors |
|---------------|---------------------------|

| | |
|----------|-------------|
| Language | Python 3.9+ |
|----------|-------------|

| | |
|--------------|------------------------|
| DB Connector | mysql-connector-python |
|--------------|------------------------|

Code Structure

1. Core Functions

- Database: `connect_db()`, `fetch_clubs()`, `fetch_nationalities()`, `fetch_seasons()`
- Tab Logic: 8 functions (e.g., `show_top_performers_by_goals()`, `show_valuation_trend()`)
 - Handle inputs, dynamic SQL queries, and outputs.

2. UI Setup

- Tkinter notebook with 8 tabs.
- Each tab has:
 - Inputs: Entry fields, dropdowns (Combobox)
 - Outputs: Text boxes, Canvas (transfers), Matplotlib graphs.

3. Execution Flow

- DB connection → UI initialization → User triggers queries → Results displayed.

Key Features

- **Dynamic Filtering:** Clubs/seasons/nationalities populate dropdowns from DB.
- **Visualizations:** Interactive transfer flowchart + valuation trend graph.
- **Single-File Design:** Combines DB, UI, and logic.

UI Elements

- **Inputs:** Text entries, dropdowns (Combobox).
- **Outputs:** Scrollable text boxes, Canvas, Matplotlib plots.

D. Statement of contribution

| | |
|----------------------------|----------------------------------|
| Data Upload & DB Setup | Asmita |
| Query Ideation | Asmita, Anushka, Nandika, Aditya |
| UI Design and Tkinter Code | Asmita, Anushka, Nandika |
| ER Diagram | Anushka |
| Normalization | Nandika |
| Testing & Debugging | Aditya |

E.References

Link to dataset : <https://www.kaggle.com/datasets/davidcariboo/player-scores>