Mini project report on

Football Tournament & League Management System

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science Engineering

UE23CS351A – DBMS Project

Submitted by:

Sneh Patel	**PES2UG23CS584**
Sarthak Wadhwa	**PES2UG23CS535**

Under the guidance of

Prof. Shilpa S

Assistant Professor

PES University

AUG - DEC 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Football Tournament & League Manager

is a bonafide work carried out by

Sneh Patel	**PES2UG23CS584**
Sarthak Wadhwa	**PES2UG23CS535**

In partial fulfilment for the completion of fifth semester DBMS Project (UE23CS351A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2025 – DEC. 2025. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

Signature

Prof. Shilpa S
Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Football Tournament & League Manager** has been carried out by us under the guidance of **Prof. Shilpa a**nd submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2025.

Sneh Patel	**PES2UG23CS584**
Sarthak Wadhwa	**PES2UG23CS535**

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Shilpa S for her continuous guidance, assistance, and encouragement throughout the development of this **UE23CS351A**- DBMS Project.

I take this opportunity to thank Dr. Sandesh B J, C, Professor, ChairPerson,

Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department.

I am deeply grateful to Prof. Jawahar Doreswamy, Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this DBMS Project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

This project presents the development of a **Football League Management System** using a Database Management System (DBMS) to efficiently organize and manage the essential data involved in a football tournament environment. The system includes five core entities: tournament, team, player, matches, and score. Each entity is modeled as a separate table in the database, enabling structured storage, retrieval, and manipulation of information related to teams, players, fixtures, results, and performance statistics.

The system provides complete CRUD (Create, Read, Update, Delete) functionalities through an interactive frontend interface developed using **Tkinter** in Python, making database interactions seamless and user-friendly. Advanced features such as automatic points calculation, dynamic leaderboard generation, match result processing, win-percentage computation, and match scheduling enhance the system's usability and accuracy. These features are enabled through SQL mechanisms like **triggers, stored procedures, and functions**, ensuring automated and consistent data handling. MySQL serves as the backend database, enabling robust relational operations and supporting complex queries used throughout the application.

Overall, the Football League Management System offers a scalable, interactive, and efficient solution for managing tournament operations. It enables organizers to streamline match management, track team progress, analyze performance, and maintain accurate records, thereby providing a structured and automated environment for football league administration.

TABLE OF CONTENTS

INTRODUCTION

The Football League Management System is an integrated digital platform designed to streamline the administration and monitoring of football tournaments. This system enables efficient management of core activities such as team registration, player information handling, match scheduling, result entry, and leaderboard generation. By organizing all tournament-related operations within a structured database environment, the system ensures smooth coordination between organizers, teams, and administrators while maintaining complete accuracy in match and performance records.

Through a well-designed and scalable database management system, the platform securely stores and manages essential data related to tournaments, teams, players, matches, and scores. Automated calculations—such as points assignment, match result processing, and win-percentage analysis—enhance efficiency and reduce manual errors. The inclusion of SQL features like stored procedures, triggers, and functions further enriches system intelligence by automating key operations and enforcing data integrity.

The system's organized architecture simplifies handling large volumes of sporting data, supporting tournament organizers in maintaining records, updating results, and analyzing team performance. With a user-friendly GUI built using Python's Tkinter, the platform allows seamless interaction with the database, making complex operations accessible through intuitive interfaces. Developed using modern database technologies and interactive frontend tools, the Football League Management System delivers a dynamic and reliable solution tailored to meet the demanding needs of sports management environments.

PROBLEM DEFINITION WITH USER REQUIREMENT SPECIFICATIONS

Managing football tournaments manually poses significant challenges, especially as the number of teams, players, and matches increases. Traditional management methods—such as paper-based records or scattered spreadsheets—often lead to data duplication, inconsistencies, delays in updating match results, and difficulties in analyzing team performance. These issues make it harder for organizers to maintain accurate tournament standings, manage player information, or process match outcomes efficiently.

The **Football League Management System** addresses these limitations by introducing a centralized and automated database solution that ensures fast, secure, and reliable handling of all football-related data. The system streamlines operations such as team registration, match scheduling, score entry, leaderboard generation, and performance analytics. With automation features powered by SQL procedures, triggers, and functions, the platform reduces manual workload and eliminates inconsistencies in record-keeping.

The primary user requirements for this platform include the following:

- **Tournament administrators:** They manage the overall functioning of the league and require a system that allows them to add, update, or delete teams and players, schedule matches, assign venues, and maintain tournament details. They must be able to record match results, view leaderboards, track team statistics, and ensure data accuracy through automated checks. Administrators also need to generate reports on tournament progress. The system should provide a reliable and responsive interface to handle large amounts of league data efficiently.

- **Team Managers / Coaches**: They use the system to maintain and update team information, including coaching details and player data such as age, position, height, weight, and jersey number. They need access to match schedules, results, and performance insights like leaderboard rankings and win percentages. A simple and intuitive interface is essential to help them track their team's progress and receive timely updates.

- **Data Operators / Record Keepers**: They handle entering and validating match results, requiring tools that ensure accurate data entry through built-in checks and automated point calculations. They must be able to retrieve match histories and maintain consistent records with the help of procedures and triggers that reduce manual effort. The system should provide simple, reliable tools that support accurate and efficient data management.

This project aims to fulfill these user requirements by developing a structured, scalable, and interactive **Football League Management System**. The system integrates MySQL for robust data handling and Python Tkinter for a user-friendly interface, ensuring that tournament organizers and team managers can efficiently perform their tasks within a unified, reliable platform.

LIST OF SOFTWARES/TOOLS/PROGRAMMING LANGUAGES USED

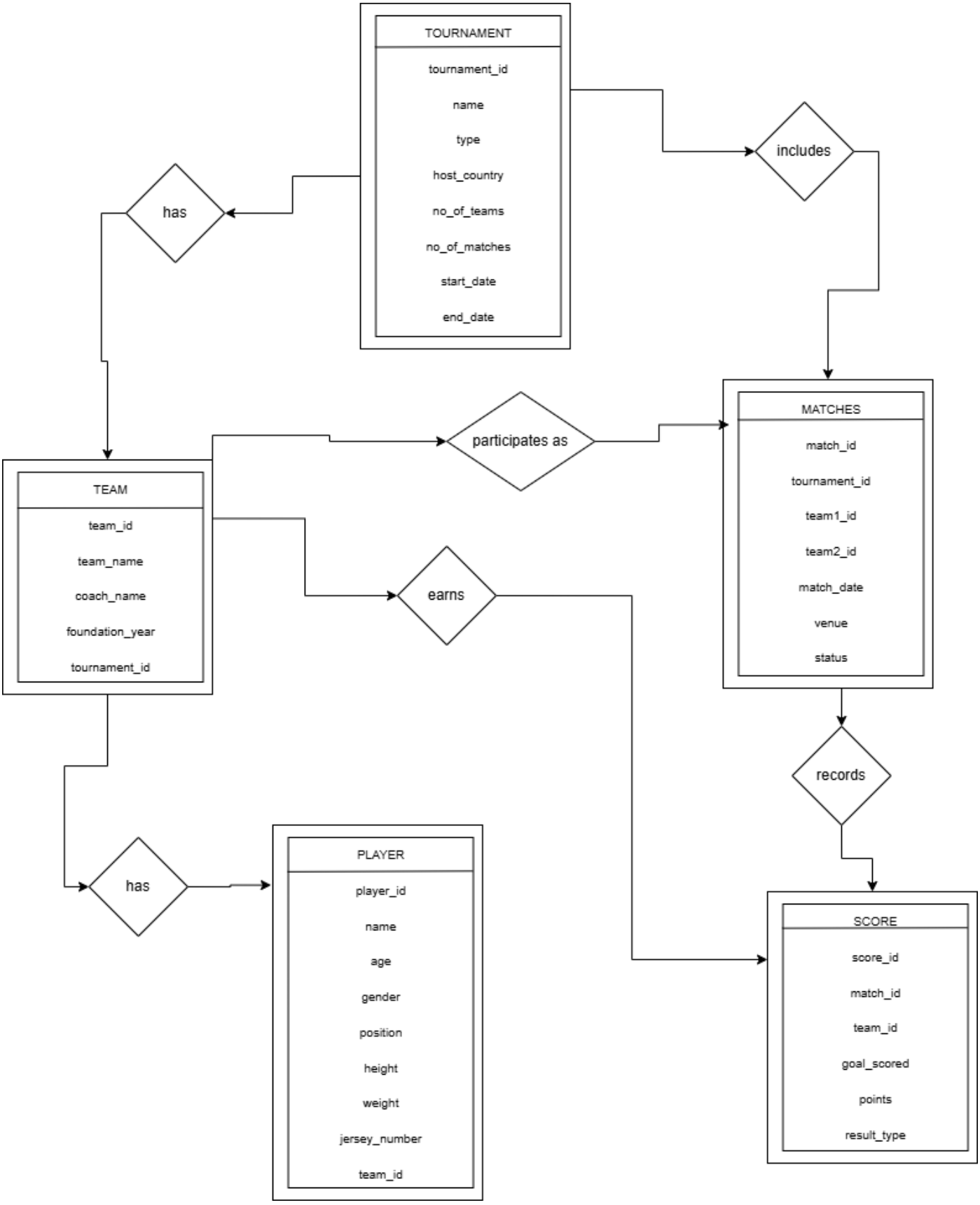
For the Football League Management System, the following tools and technologies were used:

- **Database:** MySQL, chosen for its reliability, strong support for relational data management, and efficient handling of complex queries, triggers, procedures, and functions.

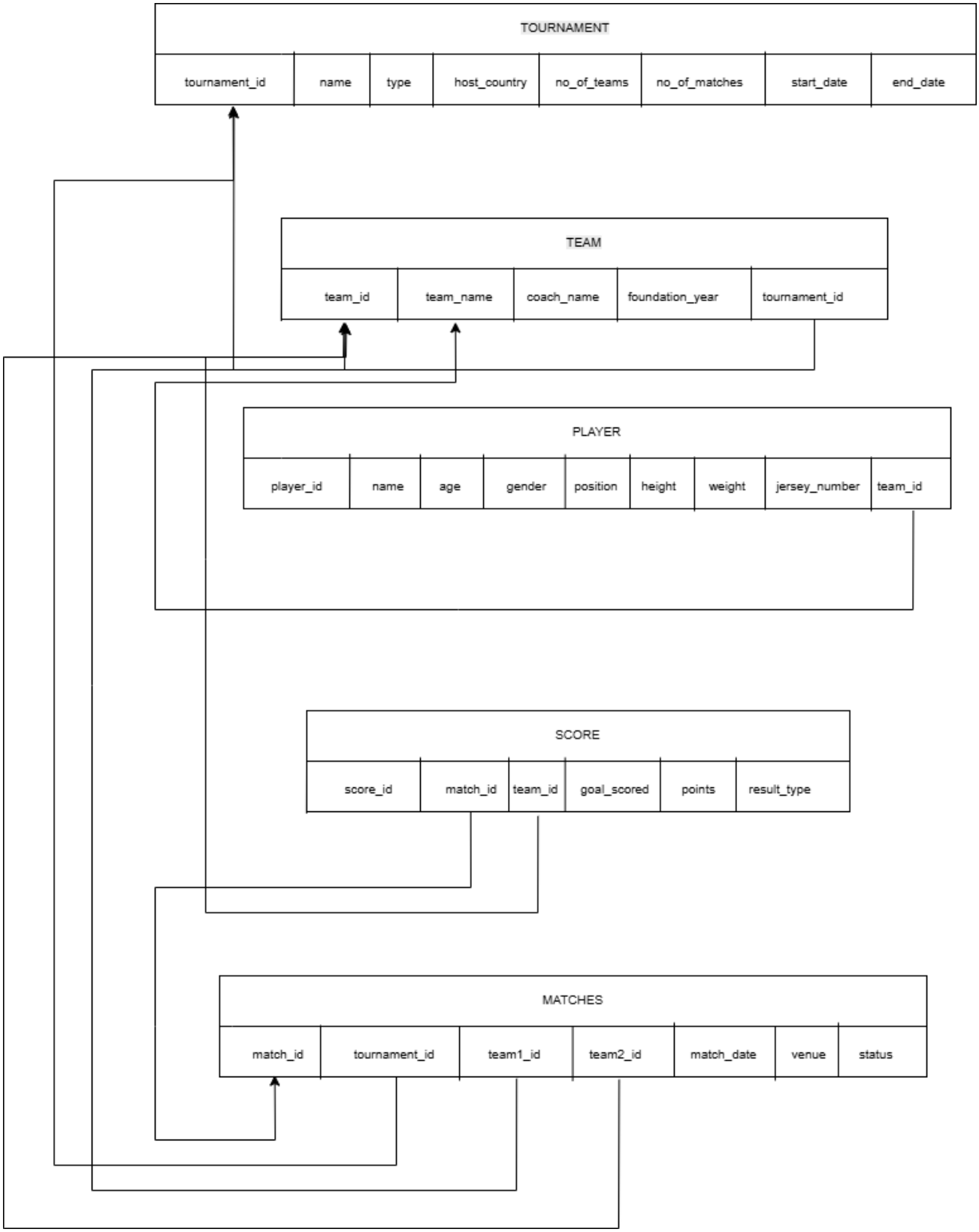
- **Programming Language:** Python, selected for its simplicity and extensive library support, making it ideal for database connectivity and GUI development.

- **GUI Framework:** Tkinter, used to create a user-friendly desktop interface for performing CRUD operations, viewing leaderboards, managing match results, and interacting with the database seamlessly.

ER MODEL


TOURNAMENT
tournament_id
name
type
host_country
no_of_teams
no_of_matches
start_date
end_date
includes
has
MATCHES
match_id
tournament_id
team1_id
team2_id
match_date
venue
status
participates as
TEAM
team_id
team_name
coach_name
foundation_year
tournament_id
earns
records
has
PLAYER
player_id
name
age
gender
position
height
weight
jersey_number
team_id
SCORE
score_id
match_id
team_id
goal_scored
points
result_type

ER TO RELATIONAL MAPPING



DDL STATEMENTS

```sql
-- 1. CREATE DATABASE
DROP DATABASE IF EXISTS FootballLeagueDB;
CREATE DATABASE FootballLeagueDB;
USE FootballLeagueDB;
```

```sql
-- ------------------------------------------------------------
-- 2. CREATE TABLE: TOURNAMENT
-- ------------------------------------------------------------
CREATE TABLE Tournament (
    tournament_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    type ENUM('League', 'Knockout') NOT NULL,
    host_country VARCHAR(100) NOT NULL,
    no_of_teams INT CHECK (no_of_teams > 0),
    no_of_matches INT CHECK (no_of_matches >= 0),
    start_date DATE,
    end_date DATE
);
```

```sql
-- ------------------------------------------------------------
-- 3. CREATE TABLE: TEAM
-- ------------------------------------------------------------
CREATE TABLE Team (
    team_id INT AUTO_INCREMENT PRIMARY KEY,
    team_name VARCHAR(100) NOT NULL,
    coach_name VARCHAR(100),
    foundation_year YEAR,
    tournament_id INT,
    FOREIGN KEY (tournament_id) REFERENCES Tournament(tournament_id)
        ON DELETE CASCADE
);
```

```sql
-- ------------------------------------------------------------
-- 4. CREATE TABLE: PLAYER
-- ------------------------------------------------------------
CREATE TABLE Player (
    player_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    age INT CHECK (age > 0),
    gender ENUM('M', 'F'),
    position VARCHAR(50),
    height_cm DECIMAL(5,2),
    weight_kg DECIMAL(5,2),
    jersey_number INT,
    team_id INT,
    FOREIGN KEY (team_id) REFERENCES Team(team_id)
        ON DELETE CASCADE
);
```

```sql
-- ------------------------------------------------------------
-- 5. CREATE TABLE: MATCHES
-- ------------------------------------------------------------
CREATE TABLE Matches (
    match_id INT AUTO_INCREMENT PRIMARY KEY,
    tournament_id INT,
    team1_id INT,
    team2_id INT,
    match_date DATE,
    venue VARCHAR(100),
    status ENUM('Scheduled', 'Completed'),
    FOREIGN KEY (tournament_id) REFERENCES Tournament(tournament_id)
        ON DELETE CASCADE,
    FOREIGN KEY (team1_id) REFERENCES Team(team_id)
        ON DELETE CASCADE,
    FOREIGN KEY (team2_id) REFERENCES Team(team_id)
        ON DELETE CASCADE
);

-- ------------------------------------------------------------
-- 6. CREATE TABLE: SCORE
-- ------------------------------------------------------------
CREATE TABLE Score (
    score_id INT AUTO_INCREMENT PRIMARY KEY,
    match_id INT,
    team_id INT,
    goals_scored INT CHECK (goals_scored >= 0),
    points INT CHECK (points >= 0),
    result_type ENUM('Win', 'Loss', 'Draw'),
    FOREIGN KEY (match_id) REFERENCES Matches(match_id)
        ON DELETE CASCADE,
    FOREIGN KEY (team_id) REFERENCES Team(team_id)
        ON DELETE CASCADE
);
```

```sql
-- ------------------------------------------------------------
-- 7. CREATE VIEW: LEADERBOARD
-- ------------------------------------------------------------
CREATE VIEW Leaderboard AS
SELECT
    t.team_id,
    t.team_name,
    COUNT(DISTINCT s.match_id) AS matches_played,
    SUM(CASE WHEN s.points = 3 THEN 1 ELSE 0 END) AS wins,
    SUM(CASE WHEN s.points = 1 THEN 1 ELSE 0 END) AS draws,
    SUM(CASE WHEN s.points = 0 THEN 1 ELSE 0 END) AS losses,
    SUM(s.goals_scored) AS goals_for,
    SUM(s.points) AS total_points
FROM Team t
JOIN Score s ON t.team_id = s.team_id
GROUP BY t.team_id, t.team_name
ORDER BY total_points DESC, goals_for DESC;
```

DML STATEMENTS (CRUD OPERATION SCREENSHOTS)

```
-- ------------------------------------------------------------
-- 1. INSERT DATA INTO TOURNAMENT
-- ------------------------------------------------------------
INSERT INTO Tournament (name, type, host_country, no_of_teams, no_of_matches, start_date, end_date)
VALUES ('Numinova Football League', 'League', 'India', 4, 6, '2025-10-01', '2025-10-15');
```

```
-- ------------------------------------------------------------------
-- 2. INSERT DATA INTO TEAM
-- ------------------------------------------------------------------
INSERT INTO Team (team_name, coach_name, foundation_year, tournament_id)
VALUES
('Titans FC', 'Arjun Mehta', 1998, 1),
('Galaxy United', 'Ravi Kumar', 2002, 1),
('Lions SC', 'Rohan Patel', 2005, 1),
('Panthers FC', 'Sameer Khan', 2010, 1);
```

```
-- ------------------------------------------------------------------
-- 5. INSERT DATA INTO SCORE
-- ------------------------------------------------------------------
INSERT INTO Score (match_id, team_id, goals_scored, points, result_type)
VALUES
(1, 1, 2, 3, 'Win'),
(1, 2, 0, 0, 'Loss'),
(2, 3, 1, 1, 'Draw'),
(2, 4, 1, 1, 'Draw'),
(3, 1, 2, 3, 'Win'),
(3, 3, 1, 0, 'Loss'),
(4, 2, 3, 3, 'Win'),
(4, 4, 2, 0, 'Loss');
```

```sql
-- ------------------------------------------------------------
-- 4 INSERT DATA INTO MATCHES
-- ------------------------------------------------------------
INSERT INTO Matches (tournament_id, team1_id, team2_id, match_date, venue, status)
VALUES
(1, 1, 2, '2025-10-02', 'Mumbai Arena', 'Completed'),
(1, 3, 4, '2025-10-03', 'Delhi Stadium', 'Completed'),
(1, 1, 3, '2025-10-05', 'Kolkata Ground', 'Completed'),
(1, 2, 4, '2025-10-06', 'Pune Turf', 'Completed');
```

```sql
-- ------------------------------------------------------------
-- 3 INSERT DATA INTO PLAYER
-- ------------------------------------------------------------
INSERT INTO Player (name, age, gender, position, height_cm, weight_kg, jersey_number, team_id) VALUES
('Rahul Singh', 24, 'M', 'Forward', 178.5, 72.3, 9, 1),
('Vikram Desai', 27, 'M', 'Midfielder', 175.2, 70.8, 8, 1),
('Arjun Nair', 25, 'M', 'Goalkeeper', 185.0, 78.0, 1, 2),
('Ravi Menon', 23, 'M', 'Defender', 180.0, 76.5, 4, 2),
('Kunal Joshi', 28, 'M', 'Forward', 177.0, 71.0, 10, 3),
('Amit Rao', 26, 'M', 'Midfielder', 172.5, 68.4, 6, 4);
```

```sql
-- ------------------------------------------------------------
-- 6. SELECT QUERIES (Data Retrieval)
-- ------------------------------------------------------------

-- Show all teams with their coaches
SELECT team_name, coach_name, foundation_year FROM Team;

-- Show all players of a specific team
SELECT p.name, p.position, t.team_name
FROM Player p
JOIN Team t ON p.team_id = t.team_id
WHERE t.team_name = 'Titans FC';

-- Show match results with winning team
SELECT m.match_id, t.team_name AS Team, s.goals_scored, s.result_type
FROM Matches m
JOIN Score s ON m.match_id = s.match_id
JOIN Team t ON s.team_id = t.team_id
WHERE m.status = 'Completed';

-- Display the leaderboard
SELECT * FROM Leaderboard;

-- Find the top scoring team
SELECT team_name, total_points
FROM Leaderboard
ORDER BY total_points DESC
LIMIT 1;
```

```sql
-- ------------------------------------------------------------
-- 7. UPDATE EXAMPLE
-- ------------------------------------------------------------
-- Update a player's weight
SET SQL_SAFE_UPDATES = 0;
UPDATE Player
SET weight_kg = 73.5
WHERE name = 'Rahul Singh';
SET SQL_SAFE_UPDATES = 1;

-- ------------------------------------------------------------
-- 8. DELETE EXAMPLE
-- ------------------------------------------------------------
-- Delete a player (e.g., removing injured player)
SET SQL_SAFE_UPDATES = 0;

DELETE FROM Player
WHERE name = 'Amit Rao';

SET SQL_SAFE_UPDATES = 1;


-- ------------------------------------------------------------
-- 9. SHOW UPDATED RESULTS
-- ------------------------------------------------------------
SELECT * FROM Player;
SELECT * FROM Leaderboard;
```

```python
# ---------------- Helpers to load DB data ----------------
def load_team_options():
    """Populate team_name_to_id and update comboboxes for creating matches."""
    global team_name_to_id
    team_name_to_id = {}
    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT team_id, team_name FROM Team ORDER BY team_name;")
        rows = cursor.fetchall()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", f"Failed to load teams:\n{e}")
        rows = []
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()

    names = []
    for team_id, team_name in rows:
        team_name_to_id[team_name] = team_id
        names.append(team_name)

    # update the create-match comboboxes
    team1_combobox['values'] = names
    team2_combobox['values'] = names

    # if teams exist, select first option by default
    if names:
        team1_combobox.current(0)
        if len(names) > 1:
            team2_combobox.current(1)
        else:
            team2_combobox.current(0)
    else:
        team1_combobox.set('')
        team2_combobox.set('')
```

```
def load_match_list():
    """Fetch matches and populate match_combobox with readable labels."""
    global match_label_to_id
    match_label_to_id = {}

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("""
            SELECT m.match_id,
                   CONCAT(m.match_id, ': ',
                          COALESCE(t1.team_name, CONCAT('Team#', m.team1_id)),
                          ' vs ',
                          COALESCE(t2.team_name, CONCAT('Team#', m.team2_id)),
                          ' (', DATE_FORMAT(m.match_date, '%Y-%m-%d'), ')') AS label
            FROM Matches m
            LEFT JOIN Team t1 ON m.team1_id = t1.team_id
            LEFT JOIN Team t2 ON m.team2_id = t2.team_id
            ORDER BY m.match_date, m.match_id;
        """)
        rows = cursor.fetchall()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", f"Failed to load matches:\n{e}")
        rows = []
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()

    labels = []
    for match_id, label in rows:
        match_label_to_id[label] = match_id
        labels.append(label)

    match_combobox['values'] = labels
    if labels:
        match_combobox.current(0)
    else:
        match_combobox.set('')
```

```python
# ---------------- CRUD / Actions ----------------
def fetch_teams():
    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT team_id, team_name, coach_name, foundation_year, tournament_id FROM Team")
        rows = cursor.fetchall()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
        rows = []
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()

    for i in team_table.get_children():
        team_table.delete(i)
    for row in rows:
        team_table.insert("", tk.END, values=row)

    # Also refresh team dropdown options (so create-match comboboxes show latest)
    load_team_options()
```

```python
def add_team():
    name = team_name_entry.get().strip()
    coach = team_coach_entry.get().strip()
    year_text = team_year_entry.get().strip()

    if not name or not coach:
        messagebox.showerror("Error", "Team Name and Coach Name are required!")
        return

    foundation_year = None
    if year_text:
        try:
            foundation_year = int(year_text)
        except ValueError:
            messagebox.showwarning("Input Error", "Foundation Year must be a number.")
            return

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute(
            "INSERT INTO Team (team_name, coach_name, foundation_year, tournament_id) VALUES (%s, %s, %s, %s)",
            (name, coach, foundation_year, 1)
        )
        db.commit()
        messagebox.showinfo("Success", f"Team '{name}' added successfully!")
        fetch_teams()
        load_match_list()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()
```

```
def delete_team():
    selected = team_table.focus()
    if not selected:
        messagebox.showwarning("Selection Error", "Select a team to delete.")
        return
    values = team_table.item(selected, "values")
    team_id = values[0]

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("DELETE FROM Team WHERE team_id = %s", (team_id,))
        db.commit()
        messagebox.showinfo("Deleted", "Team deleted successfully!")
        fetch_teams()
        load_match_list()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()
```

```python
def update_player_weight():
    player_name = update_name_entry.get().strip()
    new_weight = update_weight_entry.get().strip()

    if not player_name or not new_weight:
        messagebox.showwarning("Input Error", "Both fields are required!")
        return

    try:
        weight_val = float(new_weight)
    except ValueError:
        messagebox.showwarning("Input Error", "Weight must be a number (kg).")
        return

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SET SQL_SAFE_UPDATES = 0;")
        cursor.execute("UPDATE Player SET weight_kg = %s WHERE name = %s", (weight_val, player_name))
        cursor.execute("SET SQL_SAFE_UPDATES = 1;")
        db.commit()
        messagebox.showinfo("Success", f"Weight updated for {player_name}.")
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()
```

```python
def show_leaderboard():
    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT team_id, team_name, matches_played, wins, draws, losses, goals_for, total_points FROM Leaderboard")
        rows = cursor.fetchall()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
        rows = []
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()

    for i in leaderboard_table.get_children():
        leaderboard_table.delete(i)
    for row in rows:
        leaderboard_table.insert("", tk.END, values=row)
```

QUERIES (JOIN QUERY, AGGREGATE FUNCTION QUERIES AND NESTED QUERY)

```sql
-- Show all players of a specific team
SELECT p.name, p.position, t.team_name
FROM Player p
JOIN Team t ON p.team_id = t.team_id
WHERE t.team_name = 'Titans FC';

-- Show match results with winning team
SELECT m.match_id, t.team_name AS Team, s.goals_scored, s.result_type
FROM Matches m
JOIN Score s ON m.match_id = s.match_id
JOIN Team t ON s.team_id = t.team_id
WHERE m.status = 'Completed';


CREATE VIEW Leaderboard AS
SELECT
    t.team_id,
    t.team_name,
    COUNT(DISTINCT s.match_id) AS matches_played,
    SUM(CASE WHEN s.points = 3 THEN 1 ELSE 0 END) AS wins,
    SUM(CASE WHEN s.points = 1 THEN 1 ELSE 0 END) AS draws,
    SUM(CASE WHEN s.points = 0 THEN 1 ELSE 0 END) AS losses,
    SUM(s.goals_scored) AS goals_for,
    SUM(s.points) AS total_points
FROM Team t
JOIN Score s ON t.team_id = s.team_id
GROUP BY t.team_id, t.team_name
ORDER BY total_points DESC, goals_for DESC;
```

```
DELIMITER $$
CREATE PROCEDURE TeamPerformance(IN p_team_name VARCHAR(100))
BEGIN
    SELECT
        t.team_name,
        COUNT(s.match_id) AS matches_played,
        SUM(s.goals_scored) AS total_goals,
        SUM(s.points) AS total_points
    FROM Team t
    JOIN Score s ON t.team_id = s.team_id
    WHERE t.team_name = p_team_name
    GROUP BY t.team_name;
END $$
DELIMITER ;
```

```
CREATE VIEW Leaderboard AS
SELECT
    t.team_id,
    t.team_name,
    COUNT(DISTINCT s.match_id) AS matches_played,
    SUM(CASE WHEN s.points = 3 THEN 1 ELSE 0 END) AS wins,
    SUM(CASE WHEN s.points = 1 THEN 1 ELSE 0 END) AS draws,
    SUM(CASE WHEN s.points = 0 THEN 1 ELSE 0 END) AS losses,
    SUM(s.goals_scored) AS goals_for,
    SUM(s.points) AS total_points
FROM Team t
JOIN Score s ON t.team_id = s.team_id
GROUP BY t.team_id, t.team_name
ORDER BY total_points DESC, goals_for DESC;
```

```
DELIMITER $$
CREATE PROCEDURE TeamPerformance(IN p_team_name VARCHAR(100))
BEGIN
    SELECT
        t.team_name,
        COUNT(s.match_id) AS matches_played,
        SUM(s.goals_scored) AS total_goals,
        SUM(s.points) AS total_points
    FROM Team t
    JOIN Score s ON t.team_id = s.team_id
    WHERE t.team_name = p_team_name
    GROUP BY t.team_name;
END $$
DELIMITER ;
```

```
def load_match_list():
    """Fetch matches and populate match_combobox with readable labels."""
    global match_label_to_id
    match_label_to_id = {}

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("""
            SELECT m.match_id,
                   CONCAT(m.match_id, ': ',
                          COALESCE(t1.team_name, CONCAT('Team#', m.team1_id)),
                          ' vs ',
                          COALESCE(t2.team_name, CONCAT('Team#', m.team2_id)),
                          ' (', DATE_FORMAT(m.match_date, '%Y-%m-%d'), ')') AS label
            FROM Matches m
            LEFT JOIN Team t1 ON m.team1_id = t1.team_id
            LEFT JOIN Team t2 ON m.team2_id = t2.team_id
            ORDER BY m.match_date, m.match_id;
        """)
        rows = cursor.fetchall()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", f"Failed to load matches:\n{e}")
        rows = []
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()

    labels = []
    for match_id, label in rows:
        match_label_to_id[label] = match_id
        labels.append(label)

    match_combobox['values'] = labels
    if labels:
        match_combobox.current(0)
    else:
        match_combobox.set('')
```

```
def add_match_result():
    sel = match_combobox.get().strip()
    if not sel:
        messagebox.showwarning("Input Error", "Select a match first.")
        return

    try:
        match_id = match_label_to_id[sel]
    except KeyError:
        messagebox.showerror("Input Error", "Selected match not recognized. Try reloading match list.")
        load_match_list()
        return

    try:
        goals1 = int(goals1_entry.get().strip())
        goals2 = int(goals2_entry.get().strip())
    except ValueError:
        messagebox.showwarning("Input Error", "Goals must be integers.")
        return

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT team1_id, team2_id FROM Matches WHERE match_id = %s", (match_id,))
        row = cursor.fetchone()
        if not row:
            messagebox.showerror("Input Error", f"Match ID {match_id} does not exist (it may have been deleted).")
            load_match_list()
            return
        team1_id, team2_id = row
```

```
        cursor.callproc('AddMatchResult', (int(match_id), int(team1_id), int(goals1), int(team2_id), int(goals2)))
        db.commit()
        messagebox.showinfo("Success", "Match result added successfully!")
        show_leaderboard()
        load_match_list()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()
```

```
def calculate_win_percentage():
    team_id_text = win_team_entry.get().strip()
    if not team_id_text:
        messagebox.showwarning("Input Error", "Enter team ID!")
        return
    try:
        team_id = int(team_id_text)
    except ValueError:
        messagebox.showwarning("Input Error", "Team ID must be an integer.")
        return

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT GetWinPercentage(%s)", (team_id,))
        result = cursor.fetchone()
        win_pct = result[0] if result else 0
        messagebox.showinfo("Win Percentage", f"Team {team_id} Win % = {win_pct}")
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()
```

```
def show_leaderboard():
    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT team_id, team_name, matches_played, wins, draws, losses, goals_for, total_points FROM Leaderboard")
        rows = cursor.fetchall()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
        rows = []
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()

    for i in leaderboard_table.get_children():
        leaderboard_table.delete(i)
    for row in rows:
        leaderboard_table.insert("", tk.END, values=row)
```

```python
# ---------------- CRUD / Actions ----------------
def fetch_teams():
    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT team_id, team_name, coach_name, foundation_year, tournament_id FROM Team")
        rows = cursor.fetchall()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
        rows = []
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()

    for i in team_table.get_children():
        team_table.delete(i)
    for row in rows:
        team_table.insert("", tk.END, values=row)

    # Also refresh team dropdown options (so create-match comboboxes show latest)
    load_team_options()
```

```python
# ---------------- Create Match (GUI) ----------------
def create_match_from_gui():
    # read inputs from create-match widgets
    t1_name = team1_combobox.get().strip()
    t2_name = team2_combobox.get().strip()
    mdate = match_date_entry.get().strip()
    venue = match_venue_entry.get().strip()

    if not (t1_name and t2_name and mdate and venue):
        messagebox.showwarning("Input Error", "All create-match fields are required.")
        return

    if t1_name == t2_name:
        messagebox.showwarning("Input Error", "Team1 and Team2 must be different.")
        return

    # lightweight date validation (YYYY-MM-DD)
    if not re.match(r"^\d{4}-\d{2}-\d{2}$", mdate):
        messagebox.showwarning("Input Error", "Match Date must be in YYYY-MM-DD format.")
        return

    # map names to IDs
    if t1_name not in team_name_to_id or t2_name not in team_name_to_id:
        messagebox.showerror("Input Error", "Selected team(s) not found. Try reloading teams.")
        return

    t1 = team_name_to_id[t1_name]
    t2 = team_name_to_id[t2_name]

    db = None
    cursor = None
    try:
        db = connect_db()
        cursor = db.cursor()
```

```python
        # double-check both teams exist (safer)
        cursor.execute("SELECT COUNT(*) FROM Team WHERE team_id IN (%s, %s)", (t1, t2))
        cnt = cursor.fetchone()[0]
        if cnt < 2:
            messagebox.showerror("Input Error", "One or both selected teams do not exist.")
            return

        cursor.execute(
            "INSERT INTO Matches (tournament_id, team1_id, team2_id, match_date, venue, status) VALUES (%s,%s,%s,%s,%s,%s)",
            (1, t1, t2, mdate, venue, 'Scheduled')
        )
        db.commit()
        messagebox.showinfo("Created", "Match created successfully.")
        # refresh UI lists
        load_match_list()
    except mysql.connector.Error as e:
        messagebox.showerror("Database Error", str(e))
    finally:
        if cursor:
            cursor.close()
        if db:
            db.close()
```

STORED PROCEDURE, FUNCTIONS AND TRIGGERS

```
-- ------------------------------------------------------------
-- ⚡ TRIGGER 1: Auto-update Tournament Match Count
-- ------------------------------------------------------------
DELIMITER $$
CREATE TRIGGER trg_update_match_count
AFTER INSERT ON Matches
FOR EACH ROW
BEGIN
    UPDATE Tournament
    SET no_of_matches = no_of_matches + 1
    WHERE tournament_id = NEW.tournament_id;
END $$
DELIMITER ;

-- ------------------------------------------------------------
-- ⚡ TRIGGER 2: Prevent Negative Goals or Points
-- ------------------------------------------------------------
DELIMITER $$
CREATE TRIGGER trg_validate_score
BEFORE INSERT ON Score
FOR EACH ROW
BEGIN
    IF NEW.goals_scored < 0 OR NEW.points < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Goals and Points must be non-negative!';
    END IF;
END $$
DELIMITER ;
```

```sql
-- ------------------------------------------------------------
-- ⚙ PROCEDURE 1: Add Match Result
-- ------------------------------------------------------------
DELIMITER $$
CREATE PROCEDURE AddMatchResult(
    IN p_match_id INT,
    IN p_team1_id INT,
    IN p_team1_goals INT,
    IN p_team2_id INT,
    IN p_team2_goals INT
)
BEGIN
    DECLARE cnt INT DEFAULT 0;
    DECLARE team1_points INT DEFAULT 0;
    DECLARE team2_points INT DEFAULT 0;
    DECLARE team1_result VARCHAR(10);
    DECLARE team2_result VARCHAR(10);

    -- check match exists
    SELECT COUNT(*) INTO cnt FROM Matches WHERE match_id = p_match_id;
    IF cnt = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Match ID does not exist in Matches table.';
    END IF;
```

```
    -- determine results
    IF p_team1_goals > p_team2_goals THEN
        SET team1_points = 3; SET team2_points = 0;
        SET team1_result = 'Win'; SET team2_result = 'Loss';
    ELSEIF p_team1_goals < p_team2_goals THEN
        SET team1_points = 0; SET team2_points = 3;
        SET team1_result = 'Loss'; SET team2_result = 'Win';
    ELSE
        SET team1_points = 1; SET team2_points = 1;
        SET team1_result = 'Draw'; SET team2_result = 'Draw';
    END IF;

    START TRANSACTION;
    -- optional: remove existing scores for this match to avoid duplicates
    DELETE FROM Score WHERE match_id = p_match_id;

    INSERT INTO Score (match_id, team_id, goals_scored, points, result_type)
      VALUES (p_match_id, p_team1_id, p_team1_goals, team1_points, team1_result);

    INSERT INTO Score (match_id, team_id, goals_scored, points, result_type)
      VALUES (p_match_id, p_team2_id, p_team2_goals, team2_points, team2_result);

    UPDATE Matches SET status = 'Completed' WHERE match_id = p_match_id;

    COMMIT;
END $$
DELIMITER ;
```

```sql
-- ------------------------------------------------------------
-- ⚙ PROCEDURE 2: Team Performance Summary
-- ------------------------------------------------------------
DELIMITER $$
CREATE PROCEDURE TeamPerformance(IN p_team_name VARCHAR(100))
BEGIN
    SELECT
        t.team_name,
        COUNT(s.match_id) AS matches_played,
        SUM(s.goals_scored) AS total_goals,
        SUM(s.points) AS total_points
    FROM Team t
    JOIN Score s ON t.team_id = s.team_id
    WHERE t.team_name = p_team_name
    GROUP BY t.team_name;
END $$
DELIMITER ;
```

```sql
-- ------------------------------------------------------------
-- ▦ FUNCTION: Calculate Team Win Percentage
-- ------------------------------------------------------------
DELIMITER $$
CREATE FUNCTION GetWinPercentage(p_team_id INT)
RETURNS DECIMAL(5,2)
DETERMINISTIC
BEGIN
    DECLARE total_matches INT;
    DECLARE total_wins INT;
    DECLARE win_percentage DECIMAL(5,2);

    SELECT COUNT(DISTINCT match_id)
    INTO total_matches
    FROM Score
    WHERE team_id = p_team_id;

    SELECT COUNT(*)
    INTO total_wins
    FROM Score
    WHERE team_id = p_team_id AND result_type = 'Win';

    IF total_matches = 0 THEN
        SET win_percentage = 0;
    ELSE
        SET win_percentage = (total_wins / total_matches) * 100;
    END IF;


     RETURN win_percentage;
END $$
DELIMITER ;
```

FRONT END DEVELOPMENT (FUNCTIONALITIES/FEATURES OF THE APPLICATION)

For the **Football League Management System**, the frontend has been developed using Python Tkinter, a lightweight and powerful GUI framework that provides an interactive desktop interface. The frontend acts as the main interaction layer between the user and the MySQL database, allowing administrators, team managers, and data operators to perform various operations easily. The interface is organized using a tab-based layout, making navigation simple and intuitive. Below is an overview of the primary features and functionalities implemented in the application:

1. **Team Management (CRUD for Team Table)**

- Add Teams: Users can enter team details such as team name, coach name, and foundation year, and add the team to the database with a single click.
- View Teams: The application displays all teams in a structured table format, showing team ID, name, coach, foundation year, and tournament ID.
- Delete Teams: Allows removal of selected teams from the database. Deleted teams are automatically removed from match options as well.
- Auto-Updating Team List: The interface refreshes automatically to show newly added or removed teams.

2. **Match Management (Create & Read Operations for Matches Table)**

- Schedule New Matches: Users can select Team 1, Team 2, match date, and venue to create and save new match schedules.
- Match List Display: All scheduled matches are displayed dynamically in a dropdown, with readable labels (e.g., “1: Titans FC vs Lions SC (2025-10-05)”).
- Input Validations: The system ensures teams are different and dates follow the correct format (YYYY-MM-DD).

3. **Match Result Entry (Uses Stored Procedure & Score Table Update)**

- Add Match Results: Users can input goals scored by both teams for any selected match.
- Automatic Result Calculation: The stored procedure calculates:
    - winner / loser
    - match points (3/1/0)

    - win/loss/draw result type
- Automatic Score Insertion: Score entries for both teams are inserted automatically.
- Match Status Update: The match is marked as "Completed" automatically once results are added.

4. **Leaderboard Display (Aggregate + Join + View)**
    - View Tournament Standings: Displays a table showing:
        - matches played
        - wins, draws, losses
        - goals scored
        - total points
    - Auto-Generated Rankings: Data is retrieved from the SQL VIEW “Leaderboard,” which processes scoring automatically.
    - Real-time Refresh: Leaderboard updates instantly after adding any new match result.

5. **Player Management (Update Operation for Player Table)**
    - Update Player Weight: Users can update a player’s weight using a simple form.
    - Input Validation: Ensures weight is numeric and player name is not empty.
    - Safe Update Handling: The GUI temporarily disables SQL safe update mode to allow update by player name.

6. **Team Statistics (Function Execution - GetWinPercentage)**
    - Win Percentage Calculation: Users can input a team ID to calculate its win rate.
    - SQL Function Integration: The GUI calls the GetWinPercentage(team_id) function.
    - Instant Display: Results are shown via popup dialog.

7. **Automatic Data Refresh & Error Handling**
    - Auto-Refresh: Team lists, match lists, and leaderboard refresh automatically after relevant operations.
    - Exception Handling: All database operations are wrapped in try–except blocks to show user-

friendly error messages.

- Input Validation: Ensures correctness of data before sending it to the database.

8. **User-Friendly Interface Features**

- Tabbed Layout: Organized into five tabs:
    1. Manage Teams
    2. Add Match Results
    3. Leaderboard
    4. Player Weight Update
    5. Team Statistics
- Color-Coded Buttons: (Green for Add, Red for Delete, Blue for actions) for intuitive UX.
- Responsive Table Views: Treeview widgets display data in tabular form with scrollable layouts.
- Form-Based Input: User-friendly fields for entering match dates, venues, scores, etc.

These frontend features make the Football League Management System intuitive, interactive, and efficient. Through Tkinter, users can easily manage teams, schedule matches, record results, view rankings, and analyze team statistics, significantly simplifying tournament administration.

REFERENCES/BIBLIOGRAPHY

1. MySQL Documentation
   MySQL. (2024). *MySQL 8.0 Reference Manual*. Retrieved from [https://dev.mysql.com/doc/] (https://dev.mysql.com/doc/)

2. Draw.io (Diagrams.net)
   Diagrams.net. (2024). *Draw.io for ER Diagrams*. Retrieved from [https://app.diagrams.net/] (https://app.diagrams.net/)

3. Python Programming Language
   Python Software Foundation. (2024). *Python 3.10 Documentation*. Retrieved from [https://docs.python.org/3/] (https://docs.python.org/3/)

4. SQL Tutorial for Database Management
   W3Schools. (2024). *SQL Tutorial*. Retrieved from [https://www.w3schools.com/sql/] (https://www.w3schools.com/sql/)

GITHUB PROJECT SUBMISSION LINK

https://github.com/SNEH-17PATEL/Football-League-Management-System

APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Definitions

- Database Management System (DBMS):
  A software system used to store, manage, and retrieve structured data efficiently.
- Entity-Relationship (ER) Model:
  A conceptual design technique used to represent database entities such as Team, Player, Matches, and the relationships between them.
- CRUD Operations:
  The four basic functions of persistent storage—Create, Read, Update, and Delete—performed on database tables.
- Primary Key:
  A unique identifier for each record in a database table (e.g., team_id, player_id, match_id).
- Foreign Key:
  A reference field used to link two tables together, ensuring relational integrity (e.g., team_id in the Player table).
- View:
  A virtual table created using a SQL query, such as the Leaderboard view that summarizes team performance.
- Trigger:
  A stored database operation that executes automatically in response to specific events like INSERT, UPDATE, or DELETE.
- Stored Procedure:
  A reusable SQL program that performs complex tasks (e.g., AddMatchResult automatically inserts scores for both teams).
- Function:
  A SQL routine that returns a single value, such as win percentage (GetWinPercentage).
- Relational Database:
  A structured collection of data stored in tables with relationships between them.

Acronyms

- DDL: Data Definition Language

- DML: Data Manipulation Language
- DBMS: Database Management System
- ER: Entity-Relationship
- SQL: Structured Query Language
- CRUD: Create, Read, Update, Delete
- PK: Primary Key
- FK: Foreign Key
- GUI: Graphical User Interface
- IDE: Integrated Development Environment
- OS: Operating System

Abbreviations

- app: Application
- tbl: Table
- mod: Module
- fn: Function
- cmd: Command
- ref: Reference
- db: Database
- cfg: Configuration