University of Michigan - Dearborn
CIS-556 Database Systems
Project Report

Professor :- Niccolò Meneghetti

Project Title :-
Optimizing Soccer Data Management: Advanced Database Design and Analysis

Submitted By :-
Sayli Madhukar Chaudhari
Aniket Khedkar

Table of Contents

1. Introduction

The "Optimizing Soccer Data Management" project focuses on leveraging the European Soccer Database to develop a robust and efficient database system tailored for soccer analytics. This initiative aims to transform raw soccer data into a structured format that facilitates comprehensive analysis and insight generation. By utilizing advanced database concepts learned in CIS 556, the project emphasizes the importance of data modeling, SQL implementation, and query optimization. These foundational elements are essential for creating a system that can handle the complexities and nuances of soccer data, allowing users to derive meaningful conclusions from the information available.

This project aims to uncover valuable insights into European soccer by systematically organizing data related to teams, players, matches, and leagues. The structured database will enable analysts, coaches, and sports enthusiasts to explore trends, evaluate player performance, and assess team strategies effectively. By optimizing data management practices, the project not only enhances the accessibility of soccer statistics but also supports informed decision-making in various aspects of the sport, from player recruitment to tactical planning. Ultimately, this initiative exemplifies how advanced database techniques can significantly impact sports analytics and contribute to a deeper understanding of the game.

1.1. Project Goal

The main objectives of this project are :-

1. Design a Comprehensive Entity-Relationship (ER) Model
   Create a detailed ER model that captures the relationships between key entities in the European Soccer Database, such as teams, players, matches, and leagues. This model will define entities, attributes, and relationships to ensure an efficient database structure.
2. Implement the Schema in PostgreSQL Using DDL Statements
   Translate the ER model into a relational schema by creating tables in PostgreSQL with appropriate data types and constraints (e.g., primary keys and foreign keys) to maintain data integrity and normalization.
3. Develop an Efficient Data Import Process
   Establish a robust process for loading CSV data into the database, including data cleaning and preprocessing. Use PostgreSQL's COPY command or bulk import scripts for efficient data transfer and error handling to ensure accuracy.
4. Create and Optimize Indexes
   Enhance query performance by creating indexes on frequently queried columns. Analyze query execution plans to identify bottlenecks and apply indexing strategies that balance read and write performance.
5. Formulate and Execute Complex SQL Queries
   Write advanced SQL queries to extract insights from the database, focusing on player statistics, team trends, and match outcomes. Use joins, subqueries, and aggregate functions for comprehensive analysis.
6. Analyze and Report Soccer Performance
   Generate reports based on the analyzed data to uncover patterns in soccer performance, such as player efficiency and team strategies, providing valuable insights for coaches and analysts.

7. Demonstrate Proficiency in Database Design and Optimization
   Apply advanced database design principles and optimization techniques learned in the course to ensure data integrity, improve query efficiency, and maintain scalability for large datasets.

1.2. Dataset Description

The European Soccer Database is a comprehensive dataset encompassing various aspects of European football from 2008 to 2016. It consists of seven interconnected tables: Country, League, Match, Player, Player Attributes, Team, and Team Attributes. This rich dataset provides detailed information on match results, player statistics, team performance, and league data from multiple European countries. With over 25,000 matches and 10,000 players recorded, it offers information for analyzing trends, player performances, team strategies, and overall patterns in European soccer during this period. The database's structure allows for in-depth investigations into various aspects of the sport, making it a valuable resource for researchers, analysts, and football enthusiasts interested in exploring the intricacies of European football through data-driven approaches.

2. Database Design

2.1. Data Modeling

This database design represents a comprehensive structure for managing European soccer data. It includes seven main entities:

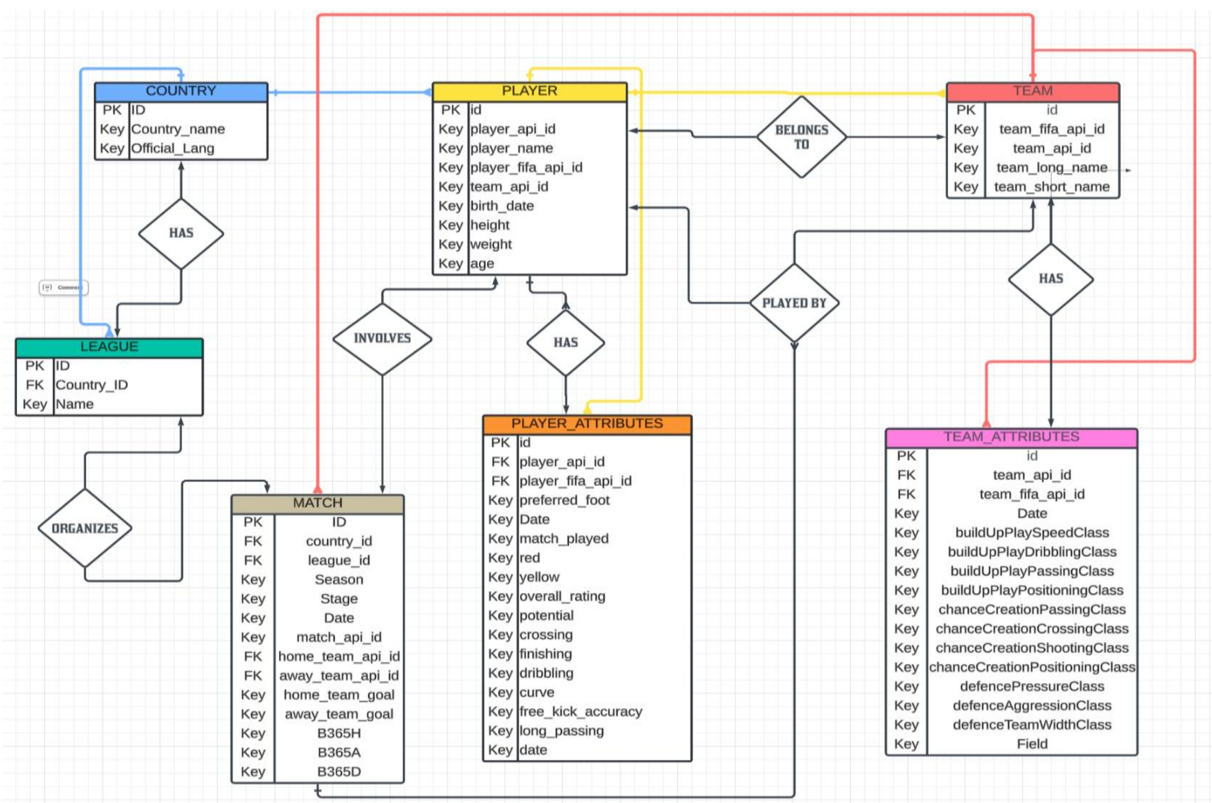
1. **Country:** Stores information about countries.
2. **League:** Represents soccer leagues, linked to countries.
3. **Team:** Contains basic team information.
4. **Team_attributes:** Stores detailed attributes of teams, linked to the Team entity.
5. **Player:** Holds player information, including their associated team.
6. **Player_attributes:** Contains detailed player attributes and statistics.
7. **Match:** Stores all the data related to matches.

The design allows for complex relationships between entities, enabling comprehensive soccer data analysis across multiple dimensions such as player performance, team strategies, league comparisons, and match outcomes. This structure supports efficient querying and analysis of soccer-related data, making it suitable for various applications in sports analytics and management.

Entity	**Attributes**
Country	id, country_name, official_lang
League	id, country_id, name
Team	id, team_api_id, team_fifa_api_id, team_long_name, team_short_name
Team_attributes	id, team_fifa_api_id, team_api_id, date, buildUpPlaySpeedClass, buildUpPlayDribblingClass, buildUpPlayPassingClass,

	buildUpPlayPositioningClass, chanceCreationPassingClass, chanceCreationCrossingClass, chanceCreationShootingClass, chanceCreationPositioningClass, defencePressureClass, defenceAggressionClass, defenceTeamWidthClass, defenceDefenderLineClass
Player	id, player_api_id, player_name, player_fifa_api_id, team_api_id, birth_date, height, weight, age
Player_attributes	id, player_fifa_api_id, player_api_id, date, preferred_foot, match_played, red, yellow, overall_rating, potential, crossing, finishing, dribbling, curve, free_kick_accuracy, long_passing, ball_control
Match	id, country_id, league_id, season, stage, date, match_api_id, home_team_api_id, away_team_api_id, home_team_goal, away_team_goal, B365H, B365D, B365A

2.2. Entity Relationship (ER) Model



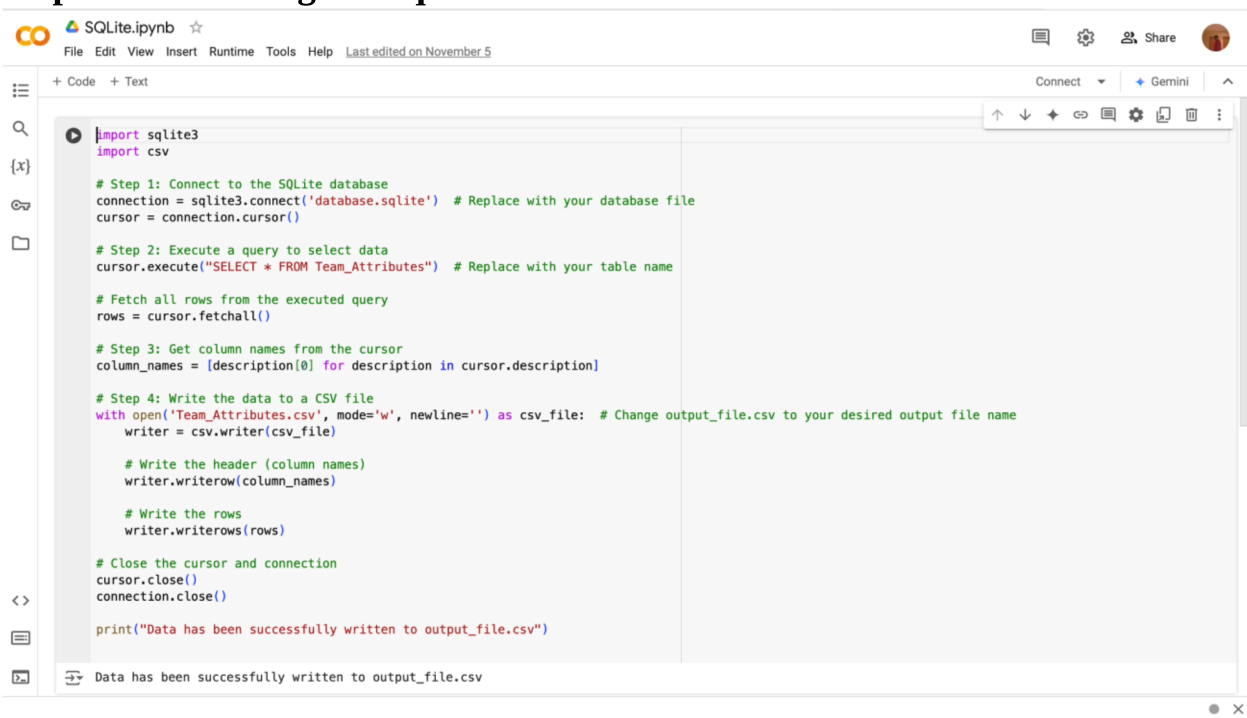
3. Implementation

3.1. Data Conversion and Loading Process

Step 1 :- Download the dataset from Kaggle

We use kaggle.com to download the dataset. The link for downloading the dataset is https://www.kaggle.com/datasets/hugomathien/soccer

Step2 :- Converting the Sqlite format files to CSV format



The steps followed in this Python script are:

1. Import required libraries:
    - sqlite3 for database operations
    - CSV for writing to CSV files
2. Connect to the SQLite database:
    - Create a connection to the 'database.sqlite' file
    - Create a cursor object for executing SQL commands
3. Execute an SQL query:
    - Use the cursor to execute "SELECT * FROM Team_Attributes"
    - Fetch all rows from the query result
4. Get column names:
    - Extract column names from the cursor's description
5. Write data to a CSV file:
    - Open a new CSV file named 'Team_Attributes.csv' in write mode
    - Create a CSV writer object

- Write the column names as the header row
- Write all data rows to the CSV file

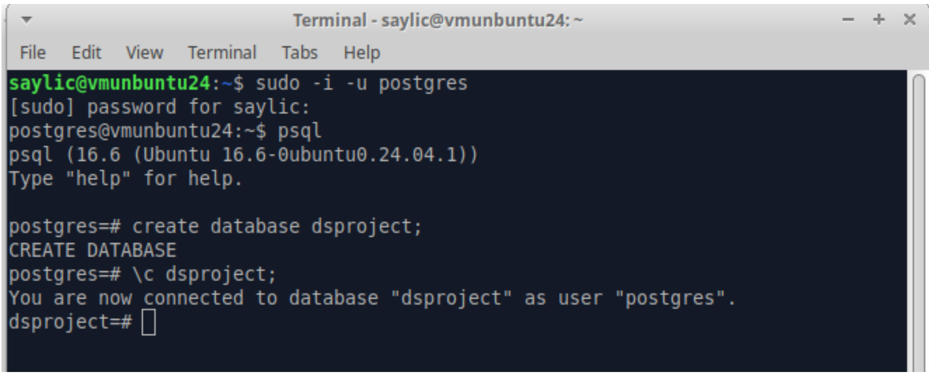
6. Close database connection:
   - Close the cursor
   - Close the database connection

Step 3 :- Cleaning the dataset and making the dataset ready to work

To clean the dataset, the steps I followed are -

1. Imported Libraries and Loaded the Data:
   - Used pandas to load the CSV file into a DataFrame.
2. Inspected the Dataset:
   - Examined the first few rows with head().
   - Checked data types and summary statistics using info() and describe().
3. Handled Missing Values:
   - Identified missing values using isnull().
   - Decided to drop columns with excessive missing values or impute missing values (mean for numerical, mode for categorical).
4. Standardized Formats:
   - Ensured consistent date formats with pd.to_datetime().
   - Standardized text entries to lowercase.
5. Removed Duplicates:
   - Used drop_duplicates() to eliminate any duplicate rows.
6. Handled Outliers:
   - Detected outliers using box plots and calculated IQR.
   - Removed or transformed identified outliers based on analysis.
7. Normalized and Scaled Numerical Data:
   - Applied min-max scaling to numerical columns for uniformity.
8. Cleaned Categorical Data:
   - Standardized categorical variables by correcting typos.
   - Used one-hot encoding for categorical variables as needed.
9. Validated Data Types:
   - Ensured all columns had appropriate data types (numerical as int/float, text as string).
10. Documented Changes:
    - Maintained a log of all transformations for transparency and reproducibility.
11. Saved the Cleaned Data:
    - Exported the cleaned dataset to a new CSV file using to_csv().
12. Iterated as Necessary:
    - Revisited earlier steps when new issues were discovered during analysis.

Step 4 :- Created a new PostgreSQL database:

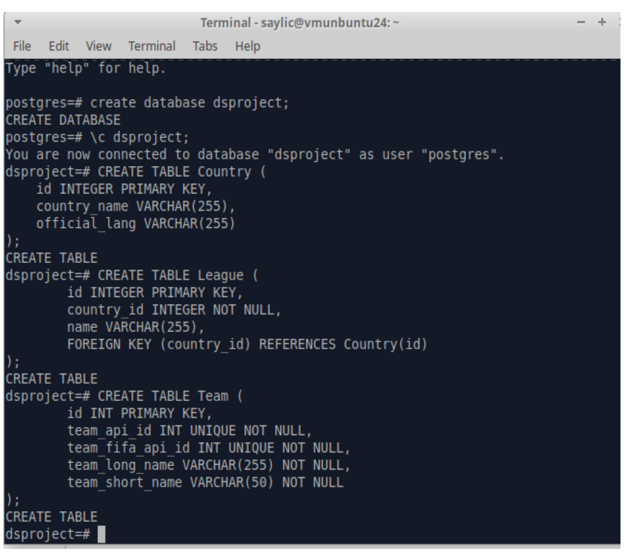


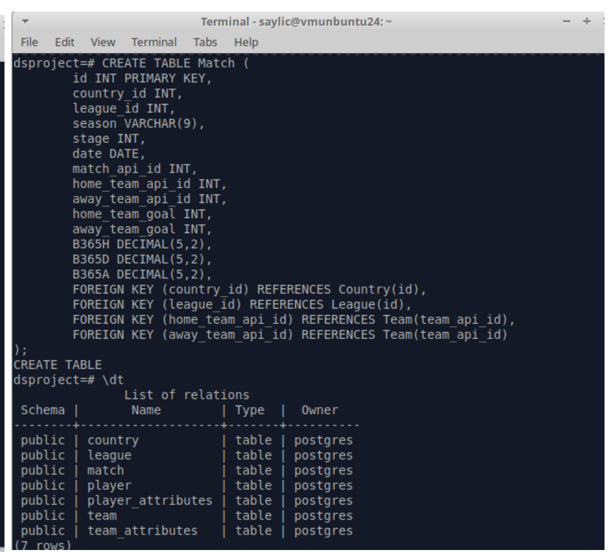
Step 5 :- Created tables using Data Definition Language (DDL) commands.

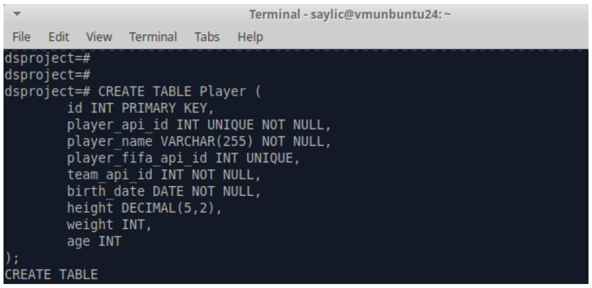
```
DROP table if exists Country;
CREATE TABLE Country (
	id INTEGER PRIMARY KEY,
	country_name VARCHAR(255),
	official_lang VARCHAR(255)
);

DROP table if exists League;
CREATE TABLE League (
	id INTEGER PRIMARY KEY,
	country_id INTEGER NOT NULL,
	name VARCHAR(255),
	FOREIGN KEY (country_id) REFERENCES Country(id)
);

DROP table if exists Team;
CREATE TABLE Team (
	id INT PRIMARY KEY,
	team_api_id INT UNIQUE NOT NULL,
	team_fifa_api_id INT UNIQUE NOT NULL,
	team_long_name VARCHAR(255) NOT NULL,
	team_short_name VARCHAR(50) NOT NULL
);

DROP table if exists Team_attributes;
CREATE TABLE Team_attributes (
	id INT PRIMARY KEY,
	team_fifa_api_id INT,
	team_api_id INT,
	date Date,
```

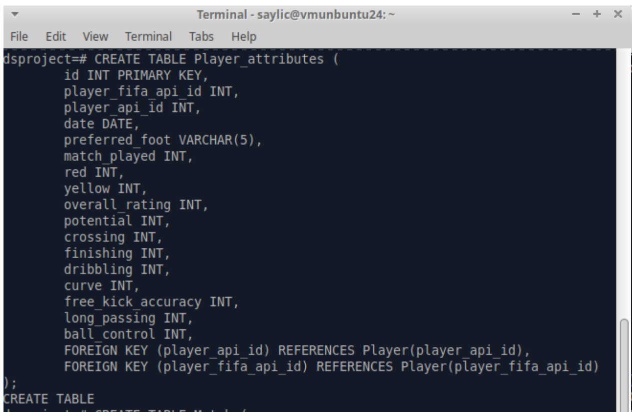
```
	buildUpPlaySpeedClass VARCHAR(50),
	buildUpPlayDribblingClass VARCHAR(50),
	buildUpPlayPassingClass VARCHAR(50),
	buildUpPlayPositioningClass VARCHAR(50),
	chanceCreationPassingClass VARCHAR(50),
	chanceCreationCrossingClass VARCHAR(50),
	chanceCreationShootingClass VARCHAR(50),
	chanceCreationPositioningClass VARCHAR(50),
	defencePressureClass VARCHAR(50),
	defenceAggressionClass VARCHAR(50),
	defenceTeamWidthClass VARCHAR(50),
	defenceDefenderLineClass VARCHAR(50),
	FOREIGN KEY (team_api_id) REFERENCES Team(team_api_id),
	FOREIGN KEY (team_fifa_api_id) REFERENCES Team(team_fifa_api_id)
);

DROP table if exists Player;
CREATE TABLE Player (
	id INT PRIMARY KEY,
	player_api_id INT UNIQUE NOT NULL,
	player_name VARCHAR(255) NOT NULL,
	player_fifa_api_id INT UNIQUE,
    team_api_id INT NOT NULL,
	birth_date DATE NOT NULL,
	height DECIMAL(5,2),
	weight INT,
	age INT
);

DROP table if exists Player_attributes;
CREATE TABLE Player_attributes (
	id INT PRIMARY KEY,
	player_fifa_api_id INT,
	player_api_id INT,
	date DATE,
	preferred_foot VARCHAR(5),
	match_played INT,
	red INT,
	yellow INT,
	overall_rating INT,
	potential INT,
	crossing INT,
	finishing INT,
	dribbling INT,
	curve INT,
	free_kick_accuracy INT,
	long_passing INT,
```

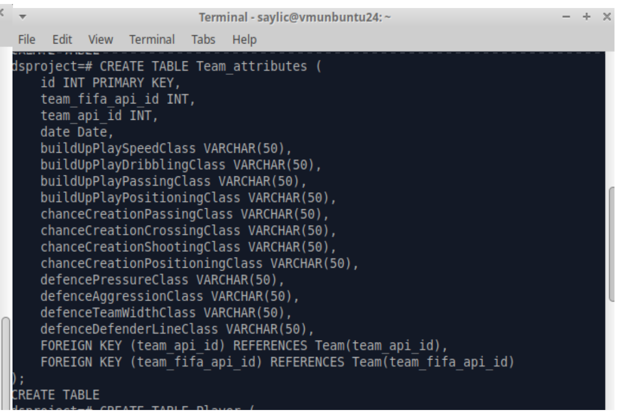
```
        ball_control INT,
        FOREIGN KEY (player_api_id) REFERENCES Player(player_api_id),
        FOREIGN KEY (player_fifa_api_id) REFERENCES Player(player_fifa_api_id)
);

DROP table if exists Match;
CREATE TABLE Match (
        id INT PRIMARY KEY,
        country_id INT,
        league_id INT,
        season VARCHAR(9),
        stage INT,
        date DATE,
        match_api_id INT,
        home_team_api_id INT,
        away_team_api_id INT,
        home_team_goal INT,
        away_team_goal INT,
        B365H DECIMAL(5,2),
        B365D DECIMAL(5,2),
        B365A DECIMAL(5,2),
        FOREIGN KEY (country_id) REFERENCES Country(id),
        FOREIGN KEY (league_id) REFERENCES League(id),
        FOREIGN KEY (home_team_api_id) REFERENCES Team(team_api_id),
        FOREIGN KEY (away_team_api_id) REFERENCES Team(team_api_id)
);
```











Step 6 :- Import CSV files into PostgreSQL

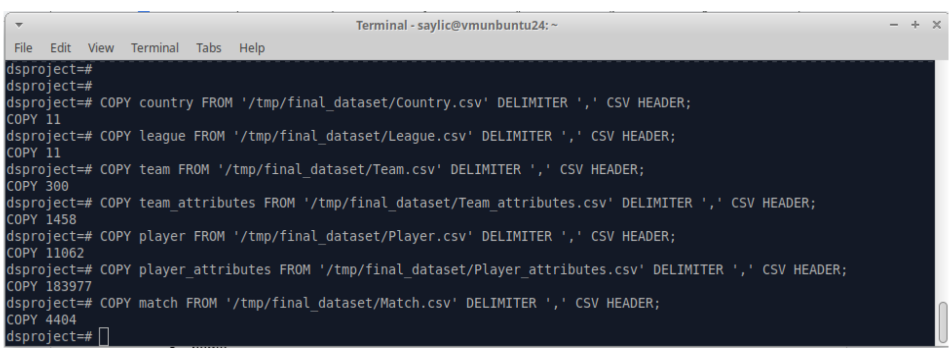
Saved all the csv file in a accessible folder and gave the same path in the COPY command.

COPY command:-

COPY country from 'file_path' DELIMITER ',' CSV HEADER;

For example:-

COPY country from '/tmp/final_dataset/Country.csv' DELIMITER ',' CSV HEADER;



Step 7 :- Checked whether the data is imported into the tables using Data Manipulation Language (DML) Statements.

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
dsproject=#
dsproject=# select * from country;
  id   | country_name | official_lang
-------+--------------+---------------
     1 | Belgium      | Dutch
  1729 | England      | English
  4769 | France       | French
  7809 | Germany      | German
 10257 | Italy        | Italian
 13274 | Netherlands  | Dutch
 15722 | Poland       | Polish
 17642 | Portugal     | Portuguese
 19694 | Scotland     | English
 21518 | Spain        | Spanish
 24558 | Switzerland  | German
(11 rows)

dsproject=# select * from league;
  id   | country_id |           name
-------+------------+--------------------------
     1 |          1 | Belgium Jupiler League
  1729 |       1729 | England Premier League
  4769 |       4769 | France Ligue 1
  7809 |       7809 | Germany 1. Bundesliga
 10257 |      10257 | Italy Serie A
 13274 |      13274 | Netherlands Eredivisie
 15722 |      15722 | Poland Ekstraklasa
 17642 |      17642 | Portugal Liga ZON Sagres
 19694 |      19694 | Scotland Premier League
 21518 |      21518 | Spain LIGA BBVA
 24558 |      24558 | Switzerland Super League
(11 rows)
```

4. SQL Queries and Analysis

For query plan “EXPLAIN ANALYZE” command is used before the query.

Query 1: Count the number of leagues per country

SELECT c.country_name, COUNT(l.id) as league_count
FROM Country c
LEFT JOIN League l ON c.id = l.country_id
GROUP BY c.id, c.country_name
ORDER BY league_count DESC;

Description: This SQL query retrieves the count of leagues for each country, displaying countries even if they have no leagues. It orders the results by the number of leagues in descending order, providing an overview of league distribution across countries.

Output:

```
dsproject=#
dsproject=# SELECT c.country_name, COUNT(l.id) as league_count
FROM Country c
LEFT JOIN League l ON c.id = l.country_id
GROUP BY c.id, c.country_name
ORDER BY league_count DESC;
 country_name | league_count
--------------+--------------
 Switzerland  |            1
 Poland       |            1
 Italy        |            1
 France       |            1
 Spain        |            1
 Netherlands  |            1
 Germany      |            1
 Portugal     |            1
 Scotland     |            1
 Belgium      |            1
 England      |            1
(11 rows)
```

Query analysis:

```
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=26.90..27.07 rows=70 width=528) (actual time=0.031..0.033 rows=11 loops=1)
   Sort Key: (count(l.id)) DESC
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=24.05..24.75 rows=70 width=528) (actual time=0.024..0.026 rows=11 loops=1)
         Group Key: c.id
         Batches: 1  Memory Usage: 24kB
         ->  Hash Right Join  (cost=11.57..23.35 rows=140 width=524) (actual time=0.017..0.020 rows=11 loops=1)
               Hash Cond: (l.country_id = c.id)
               ->  Seq Scan on league l  (cost=0.00..11.40 rows=140 width=8) (actual time=0.002..0.002 rows=11 loops=1)
               ->  Hash  (cost=10.70..10.70 rows=70 width=520) (actual time=0.011..0.011 rows=11 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on country c  (cost=0.00..10.70 rows=70 width=520) (actual time=0.006..0.007 rows=11 loops=1)
 Planning Time: 0.151 ms
 Execution Time: 0.060 ms
(14 rows)

dsproject=#
```

Query 2: Get top 10 goal scorers

SELECT p.player_name, COUNT(*) as goals
FROM Player p
JOIN Match m ON p.id = m.home_team_goal OR p.id = m.away_team_goal
GROUP BY p.id, p.player_name
ORDER BY goals DESC
LIMIT 10;

Description: This query aims to find the top 10 goal scorers by counting the goals for each player across all matches. It joins the Player table with the Match table, counting instances where a player's ID matches either the home or away team's goal scorer, then orders the results by total goals in descending order.

Output:

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help

dsproject=# SELECT p.player_name, COUNT(*) as goals
FROM Player p
JOIN Match m ON p.id = m.home_team_goal OR p.id = m.away_team_goal
GROUP BY p.id, p.player_name
ORDER BY goals DESC
LIMIT 10;
    player_name     | goals
--------------------+-------
 Aaron Appindangoye |  2418
 Aaron Cresswell    |  1788
 Aaron Doran        |   914
 Aaron Galindo      |   322
 Aaron Hughes       |   113
 Aaron Hunt         |    43
 Aaron Kuhl         |    12
 Aaron Lennon       |     4
 Aaron Lennox       |     1
(9 rows)
```

Query analysis:

```
                                                                      QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=21104.44..21104.47 rows=10 width=26) (actual time=8.907..8.910 rows=9 loops=1)
   ->  Sort  (cost=21104.44..21126.46 rows=8808 width=26) (actual time=8.906..8.908 rows=9 loops=1)
         Sort Key: (count(*)) DESC
         Sort Method: quicksort  Memory: 25kB
         ->  HashAggregate  (cost=20826.02..20914.10 rows=8808 width=26) (actual time=8.862..8.898 rows=9 loops=1)
               Group Key: p.id
               Batches: 1  Memory Usage: 409kB
               ->  Nested Loop  (cost=0.65..20781.98 rows=8808 width=18) (actual time=0.019..8.306 rows=5615 loops=1)
                     ->  Seq Scan on match m  (cost=0.00..101.04 rows=4404 width=8) (actual time=0.003..0.246 rows=4404 loops=1)
                     ->  Bitmap Heap Scan on player p  (cost=0.65..4.68 rows=2 width=18) (actual time=0.001..0.001 rows=1 loops=4404)
                           Recheck Cond: ((id = m.home_team_goal) OR (id = m.away_team_goal))
                           Heap Blocks: exact=4070
                           ->  BitmapOr  (cost=0.65..0.65 rows=2 width=0) (actual time=0.001..0.001 rows=0 loops=4404)
                                 ->  Bitmap Index Scan on player_pkey  (cost=0.00..0.32 rows=1 width=0) (actual time=0.000..0.000 rows=1 loops=4404)
                                       Index Cond: (id = m.home_team_goal)
                                 ->  Bitmap Index Scan on player_pkey  (cost=0.00..0.32 rows=1 width=0) (actual time=0.000..0.000 rows=1 loops=4404)
                                       Index Cond: (id = m.away_team_goal)
 Planning Time: 0.181 ms
 Execution Time: 9.073 ms
(19 rows)
```

Query 3: Find matches with more than 5 total goals

SELECT date, home_team_goal + away_team_goal as total_goals
FROM Match
WHERE home_team_goal + away_team_goal > 5
ORDER BY total_goals DESC, date;

Description: This query retrieves matches with high-scoring games, specifically those with more than 5 total goals. It calculates the total goals per match by summing home and away team goals, then orders the results by total goals (descending) and date, showing the highest-scoring matches first.

Output:

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
    date    | total_goals
------------+-------------
 2009-11-22 |          10
 2011-08-28 |          10
 2012-12-29 |          10
 2013-05-19 |          10
 2010-01-16 |           9
 2011-10-29 |           9
 2013-12-14 |           9
 2014-03-22 |           9
 2014-08-30 |           9
 2008-10-25 |           8
 2008-10-29 |           8
 2009-04-21 |           8
 2009-10-04 |           8
 2009-11-28 |           8
 2010-03-27 |           8
 2010-05-09 |           8
 2010-09-19 |           8
 2010-11-20 |           8
 2010-11-27 |           8
 2011-02-05 |           8
 2011-02-05 |           8
 2011-10-29 |           8
 2011-11-04 |           8
:
```

Query analysis:

```
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Sort  (cost=203.94..207.61 rows=1468 width=8) (actual time=0.318..0.325 rows=258 loops=1)
   Sort Key: ((home_team_goal + away_team_goal)) DESC, date
   Sort Method: quicksort  Memory: 33kB
   ->  Seq Scan on match  (cost=0.00..126.73 rows=1468 width=8) (actual time=0.013..0.266 rows=258 loops=1)
         Filter: ((home_team_goal + away_team_goal) > 5)
         Rows Removed by Filter: 4146
 Planning Time: 0.055 ms
 Execution Time: 0.343 ms
(8 rows)
```

Query 4: Find teams that have never lost a home game

```
SELECT t.team_long_name
FROM Team t
WHERE NOT EXISTS (
        SELECT 1
        FROM Match m
        WHERE m.home_team_api_id = t.team_api_id
        AND m.home_team_goal < m.away_team_goal
);
```

Description: This query retrieves the names of teams that have never lost a home match. It uses a subquery to check for any matches where the team is the home team and their goals are less than the away team's goals. If no such matches exist for a team, it is included in the results, indicating that the team has always won or drawn their home games.

Output:

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
       team_long_name
-----------------------------
 Le Mans FC
 Śląsk Wrocław
 Odra Wodzisław
 FC Volendam
 Real Zaragoza
 Lech Poznań
 Celtic
 Córdoba CF
 Chievo Verona
 Ross County FC
 Dundee FC
 Kilmarnock
 Granada CF
 Roda JC Kerkrade
 FC Aarau
 SC Braga
 CS Marítimo
 PSV
 AS Saint-Étienne
 Partick Thistle F.C.
 Valenciennes FC
 Rayo Vallecano
 VVV-Venlo
:
```

Query analysis:

```
                                                     QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Right Anti Join  (cost=9.75..128.87 rows=241 width=13) (actual time=0.423..0.436 rows=242 loops=1)
   Hash Cond: (m.home_team_api_id = t.team_api_id)
   ->  Seq Scan on match m  (cost=0.00..112.05 rows=1468 width=4) (actual time=0.003..0.272 rows=1251 loops=1)
         Filter: (home_team_goal < away_team_goal)
         Rows Removed by Filter: 3153
   ->  Hash  (cost=6.00..6.00 rows=300 width=17) (actual time=0.060..0.061 rows=300 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 23kB
         ->  Seq Scan on team t  (cost=0.00..6.00 rows=300 width=17) (actual time=0.011..0.030 rows=300 loops=1)
 Planning Time: 0.097 ms
 Execution Time: 0.457 ms
(10 rows)

dsproject=#
```

Query 5: List players who have both high overall rating and high potential

SELECT DISTINCT p.player_name, pa.overall_rating, pa.potential
FROM Player p
JOIN Player_attributes pa ON p.player_api_id = pa.player_api_id
WHERE pa.overall_rating >= 85 AND pa.potential >= 90
ORDER BY pa.overall_rating DESC, pa.potential DESC;

Description: This query retrieves a list of distinct players who have an overall rating of 85 or higher and a potential rating of 90 or higher. It joins the Player table with the Player_attributes table to access the necessary ratings. The results are ordered by overall rating in descending order, followed by potential in descending order, highlighting the top-performing players with high potential.

Output:

Terminal - saylic@vmunbuntu24: ~

File Edit View Terminal Tabs Help

player_name	overall_rating	potential
Lionel Messi	94	97
Lionel Messi	94	96
Lionel Messi	94	95
Lionel Messi	94	94
Lionel Messi	93	95
Cristiano Ronaldo	93	93
Gianluigi Buffon	93	93
Lionel Messi	93	93
Wayne Rooney	93	93
Cristiano Ronaldo	92	95
Cristiano Ronaldo	92	94
Gregory Coupet	92	93
Cristiano Ronaldo	92	92
Xavi Hernandez	92	92
Ronaldinho	91	95
Cristiano Ronaldo	91	94
Andres Iniesta	91	93
Gianluigi Buffon	91	93
Ronaldinho	91	93
Thierry Henry	91	93
Alessandro Nesta	91	92
Fabio Cannavaro	91	92
Iker Casillas	91	92
Thierry Henry	91	91
Xavi Hernandez	91	91
John Terry	91	90
Lionel Messi	90	95
Cristiano Ronaldo	90	94
Lionel Messi	90	94
Neymar	90	94
Andres Iniesta	90	93
Gianluigi Buffon	90	93
Kaka	90	93
Andres Iniesta	90	92
David Trezeguet	90	92

:

Query analysis:

```
                                   QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
---------
 Unique  (cost=4936.91..4937.56 rows=5 width=22) (actual time=9.561..11.634 rows=291 loops=1)
   ->  Gather Merge  (cost=4936.91..4937.52 rows=5 width=22) (actual time=9.560..11.609 rows=293 loops=1)
         Workers Planned: 1
         Workers Launched: 1
         ->  Unique  (cost=3936.90..3936.95 rows=5 width=22) (actual time=5.969..5.999 rows=146 loops=2)
               ->  Sort  (cost=3936.90..3936.91 rows=5 width=22) (actual time=5.968..5.976 rows=264 loops=2)
                     Sort Key: pa.overall_rating DESC, pa.potential DESC, p.player_name
                     Sort Method: quicksort  Memory: 45kB
                     Worker 0:  Sort Method: quicksort  Memory: 29kB
                     ->  Nested Loop  (cost=0.29..3936.84 rows=5 width=22) (actual time=0.219..5.823 rows=264 loops=2)
                           ->  Parallel Seq Scan on player_attributes pa  (cost=0.00..3895.33 rows=5 width=12) (actual time=0.197..5.476 rows=264 loops=2)
                                 Filter: ((overall_rating >= 85) AND (potential >= 90))
                                 Rows Removed by Filter: 91724
                           ->  Index Scan using player_player_api_id_key on player p  (cost=0.29..8.30 rows=1 width=18) (actual time=0.001..0.001 rows=1 lo
ops=529)
                                 Index Cond: (player_api_id = pa.player_api_id)
 Planning Time: 0.150 ms
 Execution Time: 11.683 ms
(17 rows)

(END)
```

Query 6: Rank players by their overall rating

```
SELECT player_name, overall_rating,
        RANK() OVER (ORDER BY overall_rating DESC) as rating_rank
FROM Player p
JOIN Player_attributes pa ON p.player_api_id = pa.player_api_id
WHERE pa.date = (SELECT MAX(date)
    FROM Player_attributes
    WHERE player_api_id = p.player_api_id);
```

Description: This query retrieves the names and overall ratings of players, along with their ranking based on overall rating. It uses a window function, RANK(), to assign ranks in descending order of overall rating. The query joins the Player table with the Player_attributes table and filters the results to include only the most recent player attributes by selecting the maximum date for each player. This ensures that the rankings reflect the latest performance data available.

Output:

Terminal - saylic@vmunbuntu24: ~

File Edit View Terminal Tabs Help

player_name	overall_rating	rating_rank
Lionel Messi	94	1
Cristiano Ronaldo	93	2
Manuel Neuer	90	3
Luis Suarez	90	3
Neymar	90	3
Arjen Robben	89	6
Zlatan Ibrahimovic	89	6
Mesut Oezil	88	8
Thiago Silva	88	8
Eden Hazard	88	8
Andres Iniesta	88	8
Sergio Aguero	88	8
Robert Lewandowski	88	8
Sergio Ramos	87	14
Toni Kroos	87	14
Gareth Bale	87	14
Luka Modric	87	14
David De Gea	87	14
James Rodriguez	87	14
Philipp Lahm	87	14
Jerome Boateng	87	14
Paul Pogba	86	22
Cesc Fabregas	86	22
Marco Reus	86	22
Xavi Hernandez	86	22
Gonzalo Higuain	86	22
Sergio Busquets	86	22

Query analysis:

```
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 WindowAgg  (cost=42060515.96..42060532.06 rows=920 width=26) (actual time=112920.564..112923.753 rows=11064 loops=1)
   ->  Sort  (cost=42060515.96..42060518.26 rows=920 width=18) (actual time=112920.483..112921.323 rows=11064 loops=1)
         Sort Key: pa.overall_rating DESC
         Sort Method: quicksort  Memory: 887kB
         ->  Hash Join  (cost=389.55..42060470.67 rows=920 width=18) (actual time=56852.420..112906.274 rows=11064 loops=1)
               Hash Cond: ((pa.player_api_id = p.player_api_id) AND (pa.date = (SubPlan 1)))
               ->  Seq Scan on player_attributes pa  (cost=0.00..4111.77 rows=183977 width=12) (actual time=0.008..10.173 rows=183977 loops=1)
               ->  Hash  (cost=223.62..223.62 rows=11062 width=18) (actual time=56846.724..56846.725 rows=11060 loops=1)
                     Buckets: 16384  Batches: 1  Memory Usage: 691kB
                     ->  Seq Scan on player p  (cost=0.00..223.62 rows=11062 width=18) (actual time=133.072..138.288 rows=11062 loops=1)
                     SubPlan 1
                       ->  Aggregate  (cost=4571.76..4571.77 rows=1 width=4) (actual time=5.089..5.089 rows=1 loops=22126)
                             ->  Seq Scan on player_attributes  (cost=0.00..4571.71 rows=18 width=4) (actual time=2.536..5.083 rows=17 loops=22126)
                                   Filter: (player_api_id = p.player_api_id)
                                   Rows Removed by Filter: 183960
 Planning Time: 0.262 ms
 JIT:
   Functions: 25
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 0.738 ms, Inlining 12.827 ms, Optimization 71.663 ms, Emission 48.684 ms, Total 133.912 ms
 Execution Time: 112924.852 ms
(21 rows)
```

Query 7: Calculate the average goals scored per match for each league

SELECT l.name, ROUND(AVG(m.home_team_goal + m.away_team_goal), 2) as avg_goals_per_match
FROM League l
JOIN Match m ON l.id = m.league_id
GROUP BY l.id, l.name;

Description: This query calculates the average number of goals scored per match for each league. It joins the League table with the Match table, summing the home and away team goals for each match. The average is then computed using the AVG() function and rounded to two decimal places. The results are grouped by league ID and name, providing a summary of scoring trends across different leagues.

Output:

```
dsproject=# SELECT l.name, ROUND(AVG(m.home_team_goal + m.away_team_goal), 2) as avg_goals_per_match
FROM League l
JOIN Match m ON l.id = m.league_id
GROUP BY l.id, l.name;
           name           | avg_goals_per_match
--------------------------+---------------------
 France Ligue 1           |                2.84
 Germany 1. Bundesliga    |                2.81
 England Premier League   |                2.65
 Switzerland Super League |                2.85
 Poland Ekstraklasa       |                2.92
 Italy Serie A            |                2.55
 Netherlands Eredivisie   |                3.05
 Portugal Liga ZON Sagres |                2.74
 Scotland Premier League  |                2.74
 Belgium Jupiler League   |                2.76
(10 rows)
```

Query analysis:

```
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=159.13..161.23 rows=140 width=552) (actual time=1.222..1.226 rows=10 loops=1)
   Group Key: l.id
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=13.15..126.10 rows=4404 width=528) (actual time=0.033..0.817 rows=4404 loops=1)
         Hash Cond: (m.league_id = l.id)
         ->  Seq Scan on match m  (cost=0.00..101.04 rows=4404 width=12) (actual time=0.006..0.201 rows=4404 loops=1)
         ->  Hash  (cost=11.40..11.40 rows=140 width=520) (actual time=0.009..0.010 rows=11 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on league l  (cost=0.00..11.40 rows=140 width=520) (actual time=0.005..0.006 rows=11 loops=1)
 Planning Time: 0.143 ms
 Execution Time: 1.255 ms
```

Query 8: Find players with the highest potential who have never received a red card

```
SELECT p.player_name, pa.potential
FROM Player p
JOIN Player_attributes pa ON p.player_api_id = pa.player_api_id
WHERE NOT EXISTS (
    SELECT 1
    FROM Player_attributes
    WHERE player_api_id = p.player_api_id AND red > 0
)
ORDER BY pa.potential DESC;
```

Description: This query retrieves the names and potential ratings of players who have never received a red card in their career. It joins the Player table with the Player_attributes table to access player potential. The NOT EXISTS clause is used to filter out any players who have a record of receiving a red card by checking for any entries in the Player_attributes table where the red column is greater than zero. The results are then ordered by potential in descending order, highlighting players with high potential who maintain a clean disciplinary record.

Output:

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
          player_name           | potential
--------------------------------+-----------
 Luis Figo                      |        92
 Luis Figo                      |        92
 Andriy Shevchenko              |        91
 Andriy Shevchenko              |        91
 Andriy Shevchenko              |        91
 Andriy Shevchenko              |        91
 Sylvain Wiltord                |        89
 Pedro Oldoni                   |        89
 Olivier Monterubbio            |        88
 Afonso Alves,24                |        88
 Afonso Alves,24                |        88
 Afonso Alves,24                |        88
 Afonso Alves,24                |        88
 Daniel Guiza                   |        87
 Luis Jimenez                   |        87
 Luis Figo                      |        86
 Luis Figo                      |        86
 Pedro Munitis                  |        86
 Pedro Munitis                  |        86
 Alban Lafont                   |        86
 Pedro Munitis                  |        86
 Antonio Langella               |        86
 Massimo Oddo                   |        85
:
```

Query analysis:

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
    WHERE player_api_id = p.player_api_id AND red > 0
)
ORDER BY pa.potential DESC;
                                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=12311.63..12350.71 rows=15634 width=18) (actual time=47.801..47.916 rows=2666 loops=1)
   Sort Key: pa.potential DESC
   Sort Method: quicksort  Memory: 218kB
   ->  Hash Join  (cost=6264.51..11222.53 rows=15634 width=18) (actual time=29.034..47.472 rows=2666 loops=1)
         Hash Cond: (pa.player_api_id = p.player_api_id)
         ->  Seq Scan on player_attributes pa  (cost=0.00..4111.77 rows=183977 width=8) (actual time=0.006..8.275 rows=183977 loops=1)
         ->  Hash  (cost=6252.76..6252.76 rows=940 width=18) (actual time=28.963..28.967 rows=463 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 32kB
               ->  Hash Right Anti Join  (cost=361.89..6252.76 rows=940 width=18) (actual time=28.538..28.870 rows=463 loops=1)
                     Hash Cond: (player_attributes.player_api_id = p.player_api_id)
                     ->  Seq Scan on player_attributes  (cost=0.00..4571.71 rows=103021 width=4) (actual time=0.006..16.711 rows=103022 loops=1)
                           Filter: (red > 0)
                           Rows Removed by Filter: 80955
                     ->  Hash  (cost=223.62..223.62 rows=11062 width=18) (actual time=1.766..1.767 rows=11062 loops=1)
                           Buckets: 16384  Batches: 1  Memory Usage: 691kB
                           ->  Seq Scan on player p  (cost=0.00..223.62 rows=11062 width=18) (actual time=0.006..0.736 rows=11062 loops=1)
 Planning Time: 0.258 ms
 Execution Time: 48.002 ms
(18 rows)

dsproject=#
```

Query 9: Identify players whose overall rating is higher than any player from a specific country (e.g., England)

SELECT p.player_name, pa.overall_rating, p.team_api_id,date
FROM Player p
JOIN Player_attributes pa ON p.player_api_id = pa.player_api_id
WHERE pa.overall_rating > ANY (
    SELECT pa2.overall_rating
    FROM Player p2
    JOIN Player_attributes pa2 ON p2.player_api_id = pa2.player_api_id
    JOIN Team t ON p2.team_api_id = t.team_api_id
    JOIN League l ON l.id = (SELECT league_id FROM Match WHERE home_team_api_id = t.team_api_id LIMIT 1)
    JOIN Country c ON l.country_id = c.id

```sql
        WHERE c.country_name = 'England'
);
```

Description: This SQL query retrieves the names, overall ratings, team IDs, and the date of player attributes for players whose overall rating exceeds that of at least one other player. It accomplishes this by joining the Player table with the Player_attributes table to access player details and ratings. The subquery selects overall ratings from other players by again joining the Player and Player_attributes tables, along with the Team table, although specific filtering conditions related to teams are not included in the provided snippet. The main query uses the condition pa.overall_rating > ANY (...) to ensure that only players with a higher overall rating than at least one other player are included in the results. Note that the query appears incomplete as it lacks a closing parenthesis for the subquery and may require additional conditions for clarity and functionality.

Output:

Terminal - saylic@vmunbuntu24: ~

File Edit View Terminal Tabs Help

player_name	overall_rating	team_api_id	date
Aaron Appindangoye	67	9987	2016-02-18
Aaron Appindangoye	67	9987	2015-11-19
Aaron Appindangoye	62	9987	2015-09-21
Aaron Appindangoye	61	9987	2015-03-20
Aaron Appindangoye	61	9987	2007-02-22
Aaron Cresswell	74	9993	2016-04-21
Aaron Cresswell	74	9993	2016-04-07
Aaron Cresswell	73	9993	2016-01-07
Aaron Cresswell	73	9993	2015-12-24
Aaron Cresswell	73	9993	2015-12-17
Aaron Cresswell	73	9993	2015-10-16
Aaron Cresswell	74	9993	2015-09-25
Aaron Cresswell	73	9993	2015-09-21
Aaron Cresswell	71	9993	2015-01-09
Aaron Cresswell	71	9993	2014-12-05
Aaron Cresswell	71	9993	2014-11-07
Aaron Cresswell	70	9993	2014-09-18
Aaron Cresswell	70	9993	2014-05-02
Aaron Cresswell	70	9993	2014-04-04
Aaron Cresswell	70	9993	2014-03-14
Aaron Cresswell	70	9993	2013-12-13
Aaron Cresswell	70	9993	2013-11-08
Aaron Cresswell	69	9993	2013-10-04
Aaron Cresswell	69	9993	2013-09-20
Aaron Cresswell	69	9993	2013-05-03
Aaron Cresswell	69	9993	2013-03-22

:

Query analysis:

```
                                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=323.11..3400894.79 rows=61326 width=26) (actual time=288.295..633.880 rows=183920 loops=1)
   ->  Nested Loop Semi Join  (cost=322.81..3396265.71 rows=61326 width=12) (actual time=288.271..589.838 rows=183920 loops=1)
         Join Filter: (pa.overall_rating > pa2.overall_rating)
         Rows Removed by Join Filter: 5451101
         ->  Seq Scan on player_attributes pa  (cost=0.00..4111.77 rows=183977 width=12) (actual time=0.009..7.975 rows=183977 loops=1)
         ->  Materialize  (cost=322.81..5152.16 rows=1840 width=4) (actual time=0.002..0.002 rows=31 loops=183977)
               ->  Hash Join  (cost=322.81..5142.96 rows=1840 width=4) (actual time=288.252..304.093 rows=15488 loops=1)
                     Hash Cond: (pa2.player_api_id = p2.player_api_id)
                     ->  Seq Scan on player_attributes pa2  (cost=0.00..4111.77 rows=183977 width=8) (actual time=0.002..7.214 rows=183977 loops=1)
                     ->  Hash  (cost=321.43..321.43 rows=111 width=4) (actual time=288.200..288.207 rows=925 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 41kB
                           ->  Hash Join  (cost=55.21..321.43 rows=111 width=4) (actual time=287.179..288.142 rows=925 loops=1)
                                 Hash Cond: (p2.team_api_id = t.team_api_id)
                                 ->  Seq Scan on player p2  (cost=0.00..223.62 rows=11062 width=8) (actual time=0.002..0.443 rows=11062 loops=1)
                                 ->  Hash  (cost=55.18..55.18 rows=3 width=4) (actual time=287.155..287.160 rows=25 loops=1)
                                       Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                       ->  Hash Join  (cost=24.04..55.18 rows=3 width=4) (actual time=255.946..287.151 rows=25 loops=1)
                                             Hash Cond: (l.country_id = c.id)
                                             ->  Hash Join  (cost=13.15..43.71 rows=210 width=8) (actual time=0.050..31.692 rows=59 loops=1)
                                                   Hash Cond: ((SubPlan 1) = l.id)
                                                   ->  Seq Scan on team t  (cost=0.00..6.00 rows=300 width=4) (actual time=0.012..0.029 rows=300 loops=1)
                                                   ->  Hash  (cost=11.40..11.40 rows=140 width=8) (actual time=0.009..0.010 rows=11 loops=1)
                                                         Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                                         ->  Seq Scan on league l  (cost=0.00..11.40 rows=140 width=8) (actual time=0.006..0.007 rows=11 lo
ops=1)
                                                   SubPlan 1
                                                     ->  Limit  (cost=0.00..1.49 rows=1 width=4) (actual time=0.088..0.088 rows=0 loops=359)
                                                           ->  Seq Scan on match  (cost=0.00..112.05 rows=75 width=4) (actual time=0.088..0.088 rows=0 loop
s=359)
                                                                 Filter: (home_team_api_id = t.team_api_id)
:
```

Query 10: Calculate the percentage of home wins for each season

```
SELECT season,
       ROUND(COUNT(CASE WHEN home_team_goal > away_team_goal THEN 1 END) * 100.0 / COUNT(*), 2) as home_win_percentage
FROM Match
GROUP BY season
ORDER BY season;
```

Description: This SQL query calculates the percentage of home wins for each season by counting matches where the home team scored more goals than the away team. It uses a CASE statement to count home wins, multiplies this count by 100.0, and divides it by the total number of matches in that season. The results are grouped by season and ordered accordingly, with the percentage rounded to two decimal places for clarity.

Output:

```
dsproject=#
dsproject=# SELECT season,
        ROUND(COUNT(CASE WHEN home_team_goal > away_team_goal THEN 1 END) * 100.0 / COUNT(*), 2) as home_win_percentage
FROM Match
GROUP BY season
ORDER BY season;
  season   | home_win_percentage
-----------+---------------------
 2008/2009 |               46.94
 2009/2010 |               49.15
 2010/2011 |               47.58
 2011/2012 |               46.77
 2012/2013 |               43.23
 2013/2014 |               47.19
 2014/2015 |               44.84
 2015/2016 |               46.48
(8 rows)
```

Query Analysis:

```
                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------
 Sort  (cost=145.38..145.40 rows=8 width=42) (actual time=0.840..0.841 rows=8 loops=1)
   Sort Key: season
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=145.08..145.26 rows=8 width=42) (actual time=0.827..0.830 rows=8 loops=1)
         Group Key: season
         Batches: 1  Memory Usage: 24kB
         ->  Seq Scan on match  (cost=0.00..101.04 rows=4404 width=18) (actual time=0.010..0.202 rows=4404 loops=1)
 Planning Time: 0.063 ms
 Execution Time: 0.864 ms
(9 rows)

dsproject=#
```

Query 11: Find matches where the away team won despite being the underdog (higher odds)

```
SELECT m.date, ht.team_long_name as home_team, at.team_long_name as away_team,
       m.home_team_goal, m.away_team_goal, m.B365H, m.B365A
FROM Match m
JOIN Team ht ON m.home_team_api_id = ht.team_api_id
JOIN Team at ON m.away_team_api_id = at.team_api_id
WHERE m.away_team_goal > m.home_team_goal
  AND m.B365A > m.B365H;
```

Description: This SQL query identifies matches where the away team won despite being the underdog, indicated by higher betting odds. It selects the date, names of the home and away teams, their respective goals scored, and the betting odds for home and away wins. The query filters results to include only those matches where the away team scored more goals than the home team and where the away team's odds (B365A) were greater than the home team's odds (B365H), highlighting surprising outcomes in soccer matches.

Output:

Terminal - saylic@vmunbuntu24: ~

File Edit View Terminal Tabs Help

date	home_team	away_team	home_team_goal	away_team_goal	b365h	b365a
2008-08-16	KSV Cercle Brugge	RSC Anderlecht	0	3	2.38	2.75
2008-10-31	Standard de Liège	Sporting Charleroi	1	2	1.30	9.50
2008-11-01	KV Mechelen	KSV Roeselare	1	2	1.70	4.50
2008-11-01	Royal Excel Mouscron	KSV Cercle Brugge	0	1	2.35	3.00
2008-11-08	RSC Anderlecht	Sporting Lokeren	2	3	1.36	8.50
2008-11-08	KSV Roeselare	Tubize	1	2	1.83	4.20
2008-11-23	KAA Gent	KRC Genk	2	3	2.10	3.20
2008-11-22	Beerschot AC	Sporting Lokeren	0	2	2.20	3.30
2008-12-06	Standard de Liège	SV Zulte-Waregem	1	2	1.36	7.50
2008-12-06	Sporting Charleroi	KV Mechelen	1	2	1.67	5.00
2008-12-13	KVC Westerlo	KSV Cercle Brugge	1	2	2.05	3.60
2009-01-16	RSC Anderlecht	KSV Cercle Brugge	1	2	1.36	9.00
2009-01-18	Sporting Charleroi	KVC Westerlo	1	2	2.10	3.50
2009-01-24	KSV Cercle Brugge	KV Kortrijk	0	1	1.57	5.25
2009-02-01	Sporting Lokeren	KRC Genk	1	2	2.40	2.70
2009-01-31	Royal Excel Mouscron	FCV Dender EH	0	2	1.57	5.50
2009-02-07	KRC Genk	SV Zulte-Waregem	1	2	1.66	4.75
2009-02-07	Tubize	KV Mechelen	1	5	2.50	2.62
2009-02-15	Club Brugge KV	KRC Genk	0	2	1.90	3.80
2009-02-21	FCV Dender EH	KV Mechelen	1	2	2.30	2.87
2009-02-21	KAA Gent	Beerschot AC	1	3	1.66	5.00
2009-03-07	KV Kortrijk	Sporting Lokeren	2	3	2.40	3.10
2009-03-08	KVC Westerlo	KV Mechelen	0	1	1.80	4.20
2009-03-15	Club Brugge KV	KAA Gent	1	4	1.80	4.00
2009-03-21	RAEC Mons	Sporting Lokeren	0	2	2.25	3.00
2009-03-21	KSV Cercle Brugge	Royal Excel Mouscron	0	1	1.57	5.25
2009-04-04	KRC Genk	KVC Westerlo	1	4	1.66	5.25
2009-04-11	Beerschot AC	SV Zulte-Waregem	1	2	1.91	4.00
2009-04-11	KV Kortrijk	FCV Dender EH	1	2	2.00	3.80
2009-04-11	RAEC Mons	KV Mechelen	0	1	2.25	3.20
2008-08-30	KV Mechelen	SV Zulte-Waregem	0	2	2.00	3.25
2009-04-18	KV Mechelen	KSV Cercle Brugge	0	1	2.38	3.00
2009-04-26	KSV Cercle Brugge	KRC Genk	1	2	2.38	3.00
2009-05-01	Club Brugge KV	Sporting Lokeren	2	3	1.44	7.00
2009-05-03	KRC Genk	KV Kortrijk	0	1	1.40	7.50

:

Query analysis:

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=19.50..145.17 rows=489 width=50) (actual time=0.149..0.665 rows=617 loops=1)
   Hash Cond: (m.away_team_api_id = at.team_api_id)
   ->  Hash Join  (cost=9.75..134.12 rows=489 width=41) (actual time=0.067..0.523 rows=617 loops=1)
         Hash Cond: (m.home_team_api_id = ht.team_api_id)
         ->  Seq Scan on match m  (cost=0.00..123.06 rows=489 width=32) (actual time=0.005..0.396 rows=617 loops=1)
               Filter: ((away_team_goal > home_team_goal) AND (b365a > b365h))
               Rows Removed by Filter: 3787
         ->  Hash  (cost=6.00..6.00 rows=300 width=17) (actual time=0.060..0.061 rows=300 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 23kB
               ->  Seq Scan on team ht  (cost=0.00..6.00 rows=300 width=17) (actual time=0.001..0.032 rows=300 loops=1)
   ->  Hash  (cost=6.00..6.00 rows=300 width=17) (actual time=0.078..0.078 rows=300 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 23kB
         ->  Seq Scan on team at  (cost=0.00..6.00 rows=300 width=17) (actual time=0.006..0.039 rows=300 loops=1)
 Planning Time: 0.191 ms
 Execution Time: 0.697 ms
(15 rows)

dsproject=#
```

Query 12: Rank leagues by the total number of goals scored

SELECT l.name, SUM(m.home_team_goal + m.away_team_goal) as total_goals,
RANK() OVER (ORDER BY SUM(m.home_team_goal + m.away_team_goal) DESC) as goal_rank
FROM League l
JOIN Match m ON l.id = m.league_id
GROUP BY l.id, l.name;

Description: This SQL query ranks soccer leagues based on the total number of goals scored across all matches. It selects the league name and calculates the total goals by summing the goals scored by both home and away teams in each match. The RANK() window function is used to assign a rank to each league based on the total goals, ordered in descending order. The results are grouped by league ID and name, providing a clear ranking of leagues according to their overall scoring performance.

Output:

```
dsproject=# SELECT l.name, SUM(m.home_team_goal + m.away_team_goal) as total_goals,
        RANK() OVER (ORDER BY SUM(m.home_team_goal + m.away_team_goal) DESC) as goal_rank
FROM League l
JOIN Match m ON l.id = m.league_id
GROUP BY l.id, l.name;
          name           | total_goals | goal_rank
-------------------------+-------------+-----------
 Germany 1. Bundesliga   |        3137 |         1
 Belgium Jupiler League  |        2761 |         2
 England Premier League  |        2500 |         3
 Italy Serie A           |         985 |         4
 Switzerland Super League|         693 |         5
 Portugal Liga ZON Sagres|         642 |         6
 Scotland Premier League |         449 |         7
 Netherlands Eredivisie  |         354 |         8
 France Ligue 1          |         315 |         9
 Poland Ekstraklasa      |         266 |        10
(10 rows)

dsproject=#
```

Query Analysis:

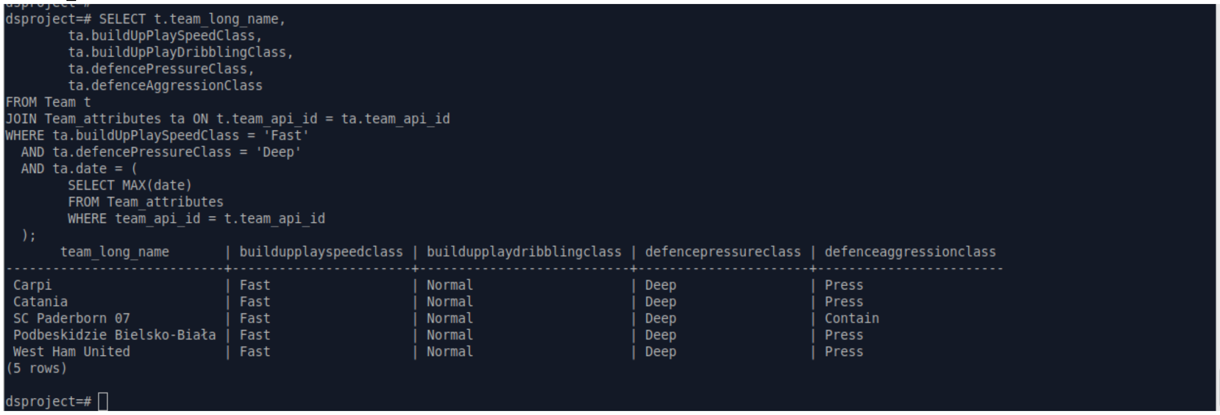
```
dsproject=# explain analyze SELECT l.name, SUM(m.home_team_goal + m.away_team_goal) as total_goals,
       RANK() OVER (ORDER BY SUM(m.home_team_goal + m.away_team_goal) DESC) as goal_rank
FROM League l
JOIN Match m ON l.id = m.league_id
GROUP BY l.id, l.name;
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
 WindowAgg  (cost=165.52..167.97 rows=140 width=536) (actual time=2.241..2.251 rows=10 loops=1)
   ->  Sort  (cost=165.52..165.87 rows=140 width=528) (actual time=2.233..2.236 rows=10 loops=1)
         Sort Key: (sum((m.home_team_goal + m.away_team_goal))) DESC
         Sort Method: quicksort  Memory: 25kB
         ->  HashAggregate  (cost=159.13..160.53 rows=140 width=528) (actual time=2.221..2.226 rows=10 loops=1)
               Group Key: l.id
               Batches: 1  Memory Usage: 40kB
               ->  Hash Join  (cost=13.15..126.10 rows=4404 width=528) (actual time=0.035..1.529 rows=4404 loops=1)
                     Hash Cond: (m.league_id = l.id)
                     ->  Seq Scan on match m  (cost=0.00..101.04 rows=4404 width=12) (actual time=0.007..0.369 rows=4404 loops=1)
                     ->  Hash  (cost=11.40..11.40 rows=140 width=520) (actual time=0.016..0.016 rows=11 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  Seq Scan on league l  (cost=0.00..11.40 rows=140 width=520) (actual time=0.008..0.009 rows=11 loops=1)
 Planning Time: 0.308 ms
 Execution Time: 2.331 ms
(15 rows)

dsproject=#
```

Query 13: Identify teams with contrasting playing styles:

```
SELECT t.team_long_name,
       ta.buildUpPlaySpeedClass,
       ta.buildUpPlayDribblingClass,
       ta.defencePressureClass,
       ta.defenceAggressionClass
FROM Team t
JOIN Team_attributes ta ON t.team_api_id = ta.team_api_id
WHERE ta.buildUpPlaySpeedClass = 'Fast'
  AND ta.defencePressureClass = 'Deep'
  AND ta.date = (
       SELECT MAX(date)
       FROM Team_attributes
       WHERE team_api_id = t.team_api_id  );
```

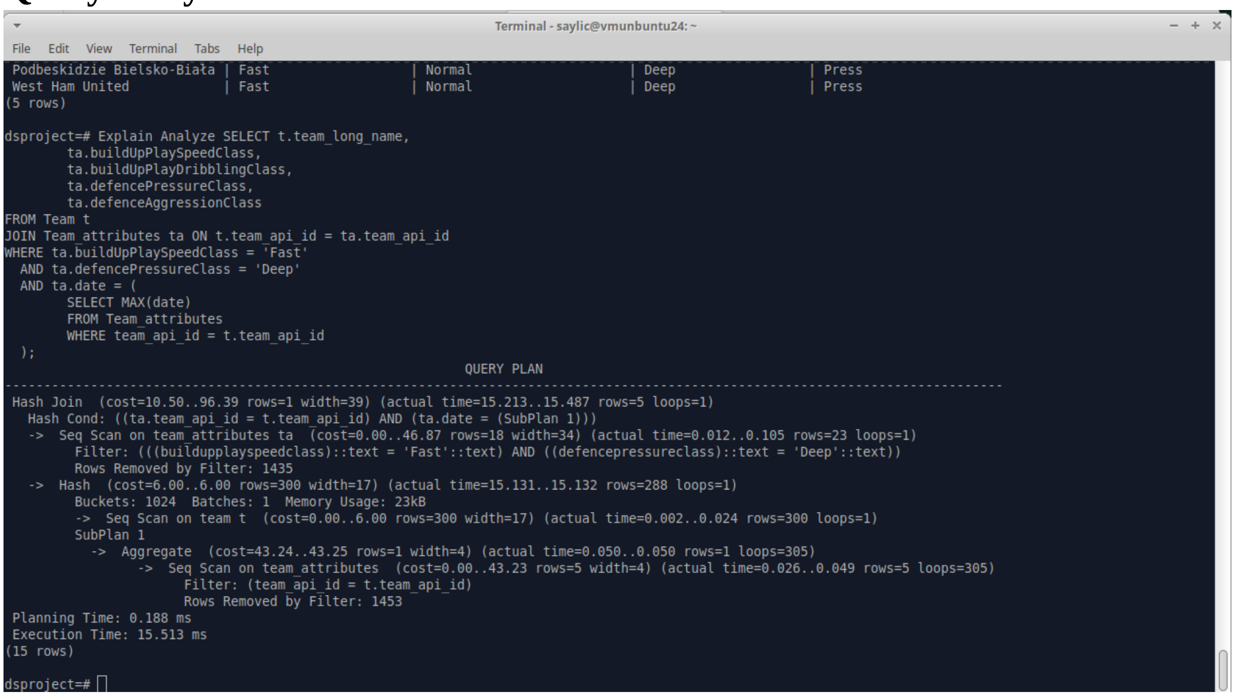
Description: This SQL query identifies teams that exhibit contrasting playing styles by selecting those with a "Fast" build-up play speed but a "Deep" defensive pressure. It retrieves the team name along with specific attributes related to their playing style, such as build-up play and defensive characteristics. The query ensures that only the most recent attributes for each team are considered by using a subquery to find the maximum date of the team attributes. This approach highlights teams that combine a quick offensive strategy with a more conservative defensive approach, showcasing their unique tactical profiles.

Output:



```
dsproject=# SELECT t.team_long_name,
        ta.buildUpPlaySpeedClass,
        ta.buildUpPlayDribblingClass,
        ta.defencePressureClass,
        ta.defenceAggressionClass
FROM Team t
JOIN Team_attributes ta ON t.team_api_id = ta.team_api_id
WHERE ta.buildUpPlaySpeedClass = 'Fast'
  AND ta.defencePressureClass = 'Deep'
  AND ta.date = (
        SELECT MAX(date)
        FROM Team_attributes
        WHERE team_api_id = t.team_api_id
  );
       team_long_name        | buildupplayspeedclass | buildupplaydribblingclass | defencepressureclass | defenceaggressionclass
-----------------------------+-----------------------+---------------------------+----------------------+------------------------
 Carpi                       | Fast                  | Normal                    | Deep                 | Press
 Catania                     | Fast                  | Normal                    | Deep                 | Press
 SC Paderborn 07             | Fast                  | Normal                    | Deep                 | Contain
 Podbeskidzie Bielsko-Biała  | Fast                  | Normal                    | Deep                 | Press
 West Ham United             | Fast                  | Normal                    | Deep                 | Press
(5 rows)

dsproject=#
```

Query analysis:



```
Terminal - saylic@vmunbuntu24:~
File  Edit  View  Terminal  Tabs  Help
 Podbeskidzie Bielsko-Biała  | Fast                  | Normal                    | Deep                 | Press
 West Ham United             | Fast                  | Normal                    | Deep                 | Press
(5 rows)

dsproject=# Explain Analyze SELECT t.team_long_name,
        ta.buildUpPlaySpeedClass,
        ta.buildUpPlayDribblingClass,
        ta.defencePressureClass,
        ta.defenceAggressionClass
FROM Team t
JOIN Team_attributes ta ON t.team_api_id = ta.team_api_id
WHERE ta.buildUpPlaySpeedClass = 'Fast'
  AND ta.defencePressureClass = 'Deep'
  AND ta.date = (
        SELECT MAX(date)
        FROM Team_attributes
        WHERE team_api_id = t.team_api_id
  );
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=10.50..96.39 rows=1 width=39) (actual time=15.213..15.487 rows=5 loops=1)
   Hash Cond: ((ta.team_api_id = t.team_api_id) AND (ta.date = (SubPlan 1)))
   ->  Seq Scan on team_attributes ta  (cost=0.00..46.87 rows=18 width=34) (actual time=0.012..0.105 rows=23 loops=1)
         Filter: (((buildupplayspeedclass)::text = 'Fast'::text) AND ((defencepressureclass)::text = 'Deep'::text))
         Rows Removed by Filter: 1435
   ->  Hash  (cost=6.00..6.00 rows=300 width=17) (actual time=15.131..15.132 rows=288 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 23kB
         ->  Seq Scan on team t  (cost=0.00..6.00 rows=300 width=17) (actual time=0.002..0.024 rows=300 loops=1)
         SubPlan 1
           ->  Aggregate  (cost=43.24..43.25 rows=1 width=4) (actual time=0.050..0.050 rows=1 loops=305)
                 ->  Seq Scan on team_attributes  (cost=0.00..43.23 rows=5 width=4) (actual time=0.026..0.049 rows=5 loops=305)
                       Filter: (team_api_id = t.team_api_id)
                       Rows Removed by Filter: 1453
 Planning Time: 0.188 ms
 Execution Time: 15.513 ms
(15 rows)

dsproject=#
```

Query 14: Measure league competitiveness based on goal difference distribution:

SELECT
  l.name as league_name,
  m.season,
  ROUND(AVG(ABS(m.home_team_goal - m.away_team_goal)), 2) as avg_goal_difference,
  ROUND(STDDEV(ABS(m.home_team_goal - m.away_team_goal)), 2) as stddev_goal_difference
FROM League l
JOIN Match m ON l.id = m.league_id

GROUP BY l.id, l.name, m.season
ORDER BY stddev_goal_difference;

Description: This SQL query measures the competitiveness of soccer leagues by analyzing the distribution of goal differences in matches. It retrieves the league name and season, calculating both the average goal difference and the standard deviation of goal differences for each league and season combination. The average is computed using AVG() on the absolute difference between home and away team goals, while the standard deviation is calculated with STDDEV(). The results are grouped by league ID, league name, and season to provide a detailed view of each league's competitiveness over time. Finally, the results are ordered by standard deviation of goal difference, with lower values indicating more competitive leagues where match outcomes are closely contested.

Output:

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
      league_name        |  season   | avg_goal_difference | stddev_goal_difference
-------------------------+-----------+---------------------+------------------------
 Italy Serie A           | 2014/2015 |                1.22 |                   1.01
 Italy Serie A           | 2013/2014 |                1.43 |                   1.05
 Portugal Liga ZON Sagres | 2013/2014 |                1.50 |                   1.09
 Portugal Liga ZON Sagres | 2012/2013 |                1.32 |                   1.10
 Belgium Jupiler League  | 2009/2010 |                1.34 |                   1.10
 Switzerland Super League | 2015/2016 |                1.39 |                   1.11
 England Premier League  | 2012/2013 |                1.16 |                   1.13
 Belgium Jupiler League  | 2008/2009 |                1.40 |                   1.14
 England Premier League  | 2008/2009 |                1.31 |                   1.15
 France Ligue 1          | 2015/2016 |                1.19 |                   1.17
 Belgium Jupiler League  | 2011/2012 |                1.31 |                   1.20
 Scotland Premier League | 2012/2013 |                1.34 |                   1.20
 Portugal Liga ZON Sagres | 2014/2015 |                1.49 |                   1.20
 Poland Ekstraklasa      | 2015/2016 |                1.30 |                   1.22
 Germany 1. Bundesliga   | 2010/2011 |                1.28 |                   1.23
 Germany 1. Bundesliga   | 2012/2013 |                1.22 |                   1.27
 Scotland Premier League | 2015/2016 |                1.31 |                   1.28
 Belgium Jupiler League  | 2010/2011 |                1.37 |                   1.28
 England Premier League  | 2013/2014 |                1.55 |                   1.29
 Germany 1. Bundesliga   | 2011/2012 |                1.44 |                   1.29
 Switzerland Super League | 2014/2015 |                1.34 |                   1.29
 Netherlands Eredivisie  | 2012/2013 |                1.53 |                   1.31
 England Premier League  | 2009/2010 |                1.44 |                   1.34
:
```

Query analysis:

```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
    m.season,
    ROUND(AVG(ABS(m.home_team_goal - m.away_team_goal)), 2) as avg_goal_difference,
    ROUND(STDDEV(ABS(m.home_team_goal - m.away_team_goal)), 2) as stddev_goal_difference
FROM League l
JOIN Match m ON l.id = m.league_id
GROUP BY l.id, l.name, m.season
ORDER BY stddev_goal_difference;
                                                   QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=293.30..296.10 rows=1120 width=594) (actual time=1.379..1.381 rows=26 loops=1)
   Sort Key: (round(stddev(abs((m.home_team_goal - m.away_team_goal))), 2))
   Sort Method: quicksort  Memory: 26kB
   ->  HashAggregate  (cost=214.18..236.58 rows=1120 width=594) (actual time=1.338..1.365 rows=26 loops=1)
         Group Key: l.id, m.season
         Batches: 1  Memory Usage: 73kB
         ->  Hash Join  (cost=13.15..126.10 rows=4404 width=538) (actual time=0.020..0.799 rows=4404 loops=1)
               Hash Cond: (m.league_id = l.id)
               ->  Seq Scan on match m  (cost=0.00..101.04 rows=4404 width=22) (actual time=0.004..0.225 rows=4404 loops=1)
               ->  Hash  (cost=11.40..11.40 rows=140 width=520) (actual time=0.010..0.010 rows=11 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on league l  (cost=0.00..11.40 rows=140 width=520) (actual time=0.005..0.006 rows=11 loops=1)
 Planning Time: 0.134 ms
 Execution Time: 1.425 ms
(14 rows)

dsproject=#
```

5. Indexing

This section highlights the optimization of the Football database through strategic indexing of key columns. By implementing these indexes, query performance is significantly improved, especially for complex queries involving joins. The result is faster data retrieval, enhanced query response times, and overall increased efficiency in database operations.

Queries to create indexes:-

CREATE INDEX idx_league_country_id ON League(country_id);

CREATE INDEX idx_team_attributes_team_api_id ON Team_attributes(team_api_id);

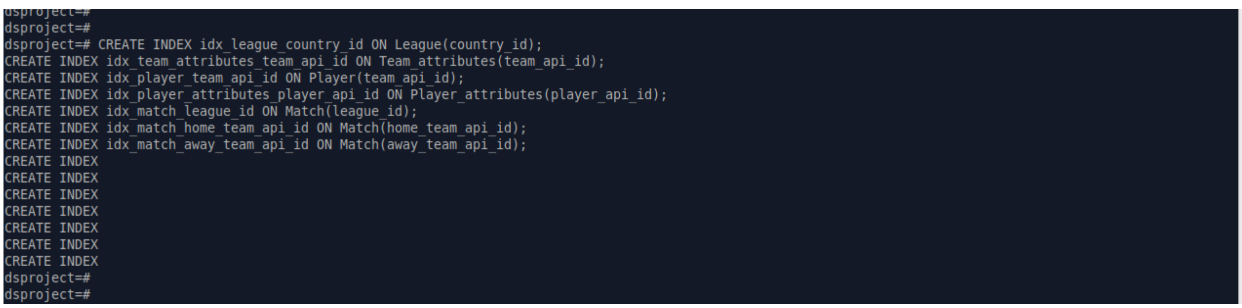
CREATE INDEX idx_player_team_api_id ON Player(team_api_id);

CREATE INDEX idx_player_attributes_player_api_id ON Player_attributes(player_api_id);

CREATE INDEX idx_match_league_id ON Match(league_id);

CREATE INDEX idx_match_home_team_api_id ON Match(home_team_api_id);

CREATE INDEX idx_match_away_team_api_id ON Match(away_team_api_id);



```
dsproject=#
dsproject=# CREATE INDEX idx_league_country_id ON League(country_id);
CREATE INDEX idx_team_attributes_team_api_id ON Team_attributes(team_api_id);
CREATE INDEX idx_player_team_api_id ON Player(team_api_id);
CREATE INDEX idx_player_attributes_player_api_id ON Player_attributes(player_api_id);
CREATE INDEX idx_match_league_id ON Match(league_id);
CREATE INDEX idx_match_home_team_api_id ON Match(home_team_api_id);
CREATE INDEX idx_match_away_team_api_id ON Match(away_team_api_id);
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
dsproject=#
dsproject=#
```

Query to analyze the execution time:-

EXPLAIN ANALYZE

SELECT c.country_name, l.name AS league_name, m.season, t.team_long_name AS home_team, COUNT(*) AS total_home_matches, AVG(m.home_team_goal) AS avg_home_goals, AVG(pa.overall_rating) AS avg_player_rating

FROM Match m

JOIN League l ON m.league_id = l.id

JOIN Country c ON l.country_id = c.id

JOIN Team t ON m.home_team_api_id = t.team_api_id

LEFT JOIN Player p ON t.team_api_id = p.team_api_id

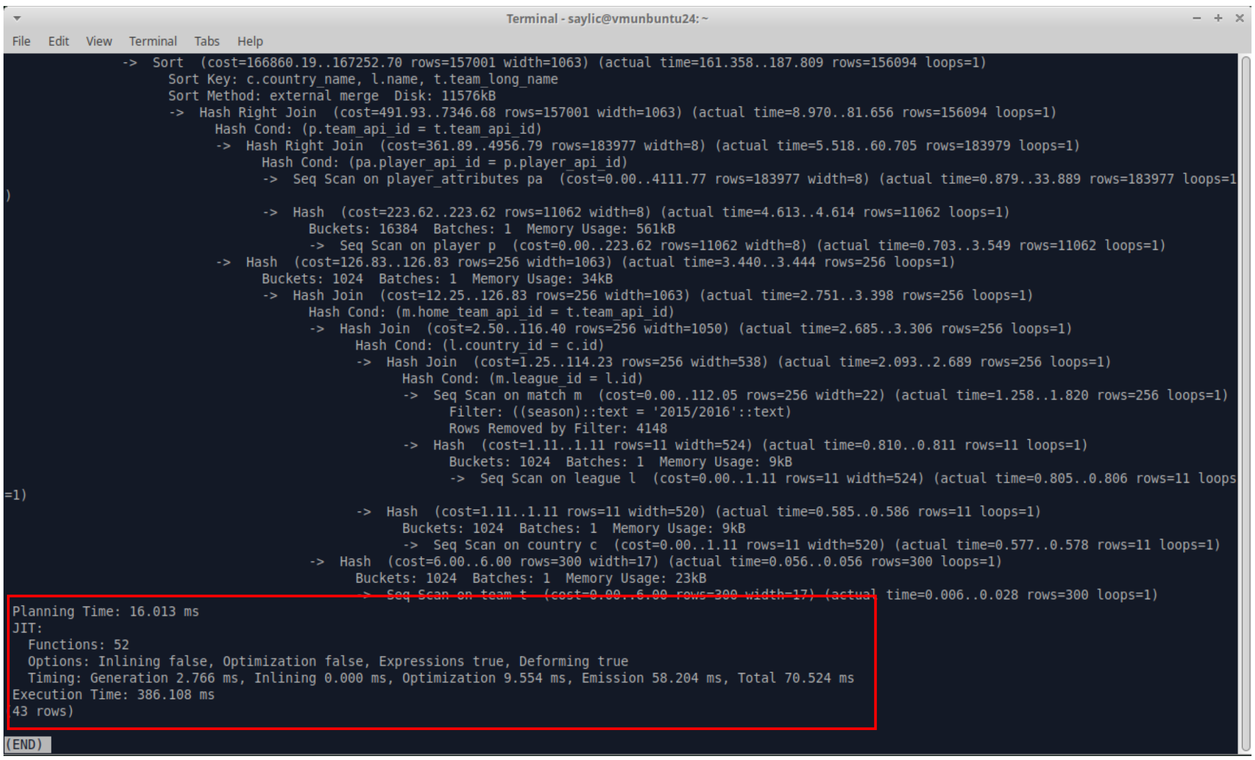
LEFT JOIN Player_attributes pa ON p.player_api_id = pa.player_api_id

WHERE m.season = '2015/2016'

GROUP BY c.country_name, l.name, m.season, t.team_long_name

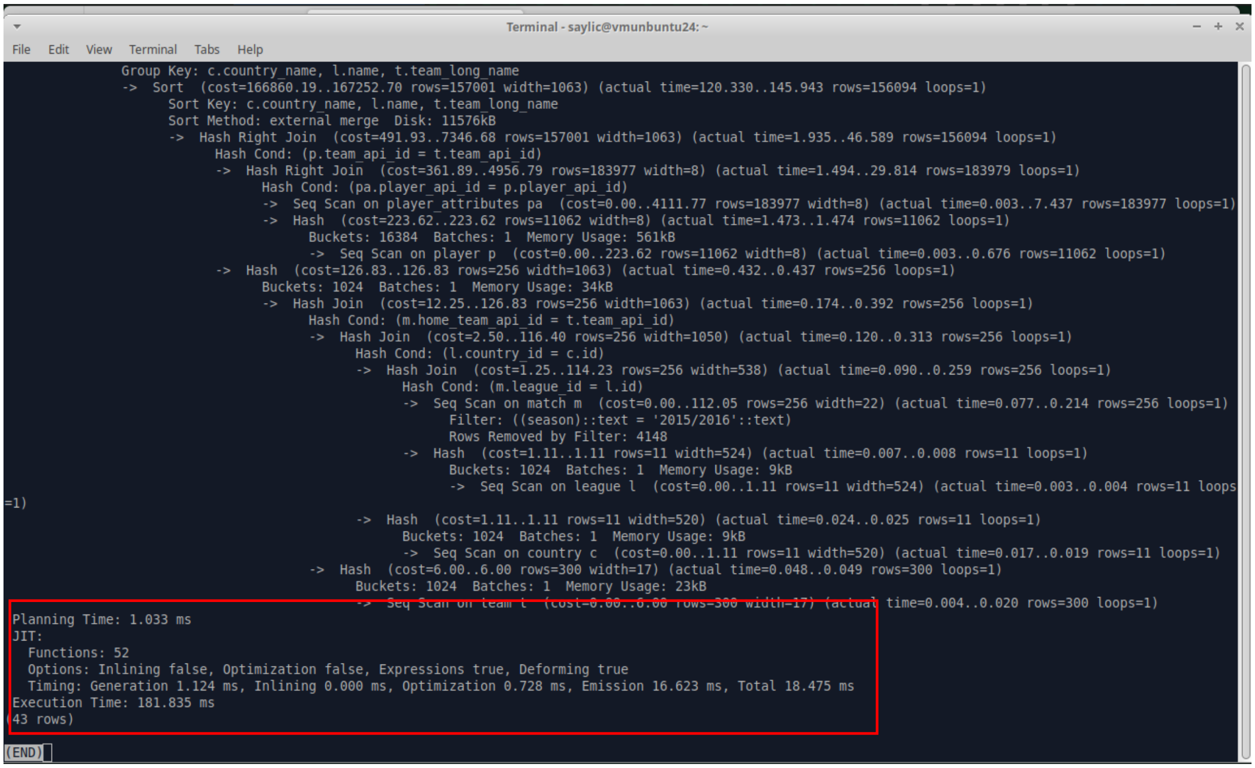
ORDER BY avg_home_goals DESC LIMIT 10;

Before creating the Indexes:-




```
 ->  Sort  (cost=166860.19..167252.70 rows=157001 width=1063) (actual time=161.358..187.809 rows=156094 loops=1)
       Sort Key: c.country_name, l.name, t.team_long_name
       Sort Method: external merge  Disk: 11576kB
       ->  Hash Right Join  (cost=491.93..7346.68 rows=157001 width=1063) (actual time=8.970..81.656 rows=156094 loops=1)
             Hash Cond: (p.team_api_id = t.team_api_id)
             ->  Hash Right Join  (cost=361.89..4956.79 rows=183977 width=8) (actual time=5.518..60.705 rows=183979 loops=1)
                   Hash Cond: (pa.player_api_id = p.player_api_id)
                   ->  Seq Scan on player_attributes pa  (cost=0.00..4111.77 rows=183977 width=8) (actual time=0.879..33.889 rows=183977 loops=1
)
                   ->  Hash  (cost=223.62..223.62 rows=11062 width=8) (actual time=4.613..4.614 rows=11062 loops=1)
                         Buckets: 16384  Batches: 1  Memory Usage: 561kB
                         ->  Seq Scan on player p  (cost=0.00..223.62 rows=11062 width=8) (actual time=0.703..3.549 rows=11062 loops=1)
             ->  Hash  (cost=126.83..126.83 rows=256 width=1063) (actual time=3.440..3.444 rows=256 loops=1)
                   Buckets: 1024  Batches: 1  Memory Usage: 34kB
                   ->  Hash Join  (cost=12.25..126.83 rows=256 width=1063) (actual time=2.751..3.398 rows=256 loops=1)
                         Hash Cond: (m.home_team_api_id = t.team_api_id)
                         ->  Hash Join  (cost=2.50..116.40 rows=256 width=1050) (actual time=2.685..3.306 rows=256 loops=1)
                               Hash Cond: (l.country_id = c.id)
                               ->  Hash Join  (cost=1.25..114.23 rows=256 width=538) (actual time=2.093..2.689 rows=256 loops=1)
                                     Hash Cond: (m.league_id = l.id)
                                     ->  Seq Scan on match m  (cost=0.00..112.05 rows=256 width=22) (actual time=1.258..1.820 rows=256 loops=1)
                                           Filter: ((season)::text = '2015/2016'::text)
                                           Rows Removed by Filter: 4148
                                     ->  Hash  (cost=1.11..1.11 rows=11 width=524) (actual time=0.810..0.811 rows=11 loops=1)
                                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                           ->  Seq Scan on league l  (cost=0.00..1.11 rows=11 width=524) (actual time=0.805..0.806 rows=11 loops
=1)
                               ->  Hash  (cost=1.11..1.11 rows=11 width=520) (actual time=0.585..0.586 rows=11 loops=1)
                                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                     ->  Seq Scan on country c  (cost=0.00..1.11 rows=11 width=520) (actual time=0.577..0.578 rows=11 loops=1)
                         ->  Hash  (cost=6.00..6.00 rows=300 width=17) (actual time=0.056..0.056 rows=300 loops=1)
                               Buckets: 1024  Batches: 1  Memory Usage: 23kB
                               ->  Seq Scan on team t  (cost=0.00..6.00 rows=300 width=17) (actual time=0.006..0.028 rows=300 loops=1)
Planning Time: 16.013 ms
JIT:
  Functions: 52
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 2.766 ms, Inlining 0.000 ms, Optimization 9.554 ms, Emission 58.204 ms, Total 70.524 ms
Execution Time: 386.108 ms
(43 rows)

(END)
```

After creating the Indexes:-



```
Terminal - saylic@vmunbuntu24: ~
File  Edit  View  Terminal  Tabs  Help
               Group Key: c.country_name, l.name, t.team_long_name
               ->  Sort  (cost=166860.19..167252.70 rows=157001 width=1063) (actual time=120.330..145.943 rows=156094 loops=1)
                     Sort Key: c.country_name, l.name, t.team_long_name
                     Sort Method: external merge  Disk: 11576kB
                     ->  Hash Right Join  (cost=491.93..7346.68 rows=157001 width=1063) (actual time=1.935..46.589 rows=156094 loops=1)
                           Hash Cond: (p.team_api_id = t.team_api_id)
                           ->  Hash Right Join  (cost=361.89..4956.79 rows=183977 width=8) (actual time=1.494..29.814 rows=183979 loops=1)
                                 Hash Cond: (pa.player_api_id = p.player_api_id)
                                 ->  Seq Scan on player_attributes pa  (cost=0.00..4111.77 rows=183977 width=8) (actual time=0.003..7.437 rows=183977 loops=1)
                                 ->  Hash  (cost=223.62..223.62 rows=11062 width=8) (actual time=1.473..1.474 rows=11062 loops=1)
                                       Buckets: 16384  Batches: 1  Memory Usage: 561kB
                                       ->  Seq Scan on player p  (cost=0.00..223.62 rows=11062 width=8) (actual time=0.003..0.676 rows=11062 loops=1)
                           ->  Hash  (cost=126.83..126.83 rows=256 width=1063) (actual time=0.432..0.437 rows=256 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 34kB
                                 ->  Hash Join  (cost=12.25..126.83 rows=256 width=1063) (actual time=0.174..0.392 rows=256 loops=1)
                                       Hash Cond: (m.home_team_api_id = t.team_api_id)
                                       ->  Hash Join  (cost=2.50..116.40 rows=256 width=1050) (actual time=0.120..0.313 rows=256 loops=1)
                                             Hash Cond: (l.country_id = c.id)
                                             ->  Hash Join  (cost=1.25..114.23 rows=256 width=538) (actual time=0.090..0.259 rows=256 loops=1)
                                                   Hash Cond: (m.league_id = l.id)
                                                   ->  Seq Scan on match m  (cost=0.00..112.05 rows=256 width=22) (actual time=0.077..0.214 rows=256 loops=1)
                                                         Filter: ((season)::text = '2015/2016'::text)
                                                         Rows Removed by Filter: 4148
                                                   ->  Hash  (cost=1.11..1.11 rows=11 width=524) (actual time=0.007..0.008 rows=11 loops=1)
                                                         Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                                         ->  Seq Scan on league l  (cost=0.00..1.11 rows=11 width=524) (actual time=0.003..0.004 rows=11 loops
=1)

                                             ->  Hash  (cost=1.11..1.11 rows=11 width=520) (actual time=0.024..0.025 rows=11 loops=1)
                                                   Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                                   ->  Seq Scan on country c  (cost=0.00..1.11 rows=11 width=520) (actual time=0.017..0.019 rows=11 loops=1)
                                       ->  Hash  (cost=6.00..6.00 rows=300 width=17) (actual time=0.048..0.049 rows=300 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 23kB
                                             ->  Seq Scan on team t  (cost=0.00..6.00 rows=300 width=17) (actual time=0.004..0.020 rows=300 loops=1)
Planning Time: 1.033 ms
JIT:
  Functions: 52
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 1.124 ms, Inlining 0.000 ms, Optimization 0.728 ms, Emission 16.623 ms, Total 18.475 ms
Execution Time: 181.835 ms
(43 rows)

(END)
```

5.1. Performance Improvement Analysis

These CREATE INDEX statements create indexes on specific columns across multiple tables in the database. Indexes are being added to foreign key columns (like country_id in League, team_api_id in Team_attributes) and frequently queried columns (like league_id, home_team_api_id, and away_team_api_id in Match). These indexes are designed to improve query performance by allowing faster data retrieval and join operations, especially for queries involving relationships between tables such as matches, leagues, teams, and players. The addition of these indexes particularly enhances the speed of queries that involve joining these tables or filtering on the indexed columns. Before the indexes were created the execution time of the above mentioned query was 386.108 ms and after the indexes were created the execution time is 181.835 ms.

6. Conclusion

This project optimized a European soccer database, focusing on efficient data management and analysis. It involved creating a comprehensive schema with seven interconnected tables covering countries, leagues, teams, players, and matches from 2008 to 2016. The database design supports complex relationships and in-depth analysis of soccer performance. Strategic indexing was implemented on key columns, significantly improving query performance. This optimized structure enables efficient analysis of team performances, player statistics, and match outcomes, providing a solid foundation for advanced sports analytics and data-driven decision-making in football management.