FIFA 23 Players Archive: Comprehensive Data & Analysis

Sri Guna Kaushik
srigunak
50478870
Engineering Science Data Science
Email: srigunak@buffalo.edu

Shri Harsha Adapala
sadapala
50495139
Engineering Science Data Science
Email: sadapala@buffalo.edu

Ajay Reddy Timmareddy
ajayredd
50496496
Computer Science and Engineering
Email: ajayredd@buffalo.edu

***Abstract*—In the fast-evolving realm of professional football, the adoption of structured databases over conventional Excel files is poised to revolutionize the way player data is managed and analyzed. This paradigm shift offers a multitude of advantages, from heightened efficiency in handling extensive player data and ensuring data consistency, to robust security measures, streamlined data relationships, scalability, and advanced data analysis capabilities. Structured databases are set to enhance the accuracy, reliability, and overall quality of player information, reduce data loss risks, and foster collaboration among diverse stakeholders. This transition signifies a critical milestone in the optimization of data-driven decision-making in the football industry.**

1. Problem statement

In the fast-paced world of professional football, it's crucial for clubs, national teams, scouts, and analysts to handle and study player data efficiently. However, the way they currently do this using Excel files has problems. It makes it hard to use the valuable information effectively. So, our goal is to make a complete database for football players. We'll use a structured database, which is better than regular Excel files, to help with this.

Using a database instead of an Excel file offers several advantages in managing and analyzing data, especially when dealing with complex and extensive datasets, as is often the case in the world of professional football:

1.1. Efficiency

A database is designed to efficiently store and retrieve extensive player data, handling large volumes without slowing down. Excel files can become sluggish as the data grows.

1.2. Data Consistency

Databases enforce data consistency and reliability through rules and checks, ensuring that the player's information remains accurate.

1.3. Data Security

Databases offer stronger security features, including user authentication and authorization. We can control who accesses the data and what they can do with it, safeguarding player information.

1.4. Data Relationships

Databases enable us to define connections between different data tables. This simplifies representing complex football associations, like player transfers, team affiliations, and performance histories.

1.5. Scalability

As player data accumulates over time, a database can easily adapt to accommodate the growth. It remains effective even as the dataset expands.

1.6. Data Analysis

Databases provide robust tools for in-depth data analysis, surpassing what Excel offers. This empowers us to gain deeper insights into player performance, scouting, and team strategies.

1.7. Data Quality

Databases support data validation and cleaning, helping to maintain accurate and consistent player data.

1.8. Data Recovery

Databases often include backup and recovery mechanisms, reducing the risk of losing player data due to accidental deletions or file issues.

1.9. Data Collaboration

Databases enable multiple users, such as scouts, analysts, and team management, to work together on the data simultaneously, fostering collaboration and informed decision-making in the world of football.

2. Background of the Problem

In football, whether it's at the club or national level, we've seen a big increase in the amount of information about players. This info covers things like player profiles, how they perform, and more. But right now, people mostly use Excel files to keep track of all this, and that causes problems. It leads to issues like having the same data in multiple places, not all the data being the same, and it's not easy for everyone to access the data whenever they need it. All of this makes it harder for us to make good decisions, like finding new players, helping players get better, and managing resources. With the importance of using data to understand and improve football, these problems are even more obvious.

3. Objective and Contribution

Our project seeks to address these challenges by developing a centralized and structured football player database. By creating a database system, we aim to achieve the following objectives:

3.1. Efficient Data Management

We will provide a robust platform for storing, and organizing player-related data, reducing the risk of data duplication, and improving data consistency.

3.2. Real-time Updates

This database will allow enabling clubs, scouts, and analysts to access the most recent player information, such as recent performance statistics and injuries.

3.3. Data Analysis

By centralizing data, the database will facilitate more sophisticated and efficient data analysis, providing valuable insights into player performance, scouting opportunities, and strategy optimization.

3.4. Enhanced Accessibility

This system will provide secure and role-based access to authorized users, ensuring that relevant stakeholders have access to the data they need.

Our project is important in the world of football because it helps people/talent scouts make smart decisions using data. Nowadays, using data is a big deal in sports like football, and having the right information about players is like having a secret weapon. It can help clubs find great new players, make their current players better, prevent injuries, and come up with better game plans. By fixing the problems with using Excel files and making a structured database, our project could change the way football teams handle player data. This could make them more competitive and successful in the football world.

4. Target Users

In a football player database project, many different people would use and manage the database. Let's look at a real-life example to see who they might be:

4.1. Club Coaches and Managers

The club's coaching staff and management would use the database to access player profiles, performance statistics, injury records, and other data to make informed decisions about the team's lineup, strategy, and training.

4.2. Scouts

Scouts tasked with identifying potential new talents would utilize the database to evaluate player statistics, track player development, and assess transfer prospects.

4.3. Players

Players themselves might have access to certain aspects of their profiles, including performance analytics, to help them track their progress and set personal goals.

4.4. Team Analysts

Data analysts would use the database to generate reports, insights, and visualizations to assist the coaching staff and management in decision-making.

5. Transform the original data into Boyce-Codd Normal Form (BCNF) and Verification

The initial dataset comprises three tables: Coaches, Teams, and Players. Remarkably, the Coaches table already adheres to the Boyce-Codd Normal Form (BCNF), as all functional dependencies exhibit candidate keys on the left side and none are trivial. However, both the Players and Teams tables do not meet BCNF criteria.

In the Players table, we identified a partial dependency between the attributes team_id and league_id. To address this issue, we eliminated the league_id attribute, as team_id alone uniquely determines it. Additionally, we removed the league_name attribute, as team_id uniquely identifies league_name through the transitive property. Similarly, we excluded the team_name attribute since team_id uniquely determines it due to the partial dependency. We maintained 1NF and 2NF by splitting the original player's table into two: players and player_stats. This division addressed the multi-valued positions attribute, ensuring atomicity in the player's table and reducing redundancy, while also allowing independent modifications to player statistics without affecting core player data.

For the Teams table, to align with BCNF and 2NF standards, we decomposed the Teams table into 'teams' and 'leagues', and further split 'teams' into 'teams' and 'team_stats'. This restructuring ensures BCNF compliance by separating league-related data, while the latter division guarantees full dependency on primary keys, enhancing data integrity and reducing redundancy in our database.

Let's assess the BCNF (Boyce-Codd Normal Form) compliance of each table.:

5.1. countries

It has nationality_id, and nationality_name, as attributes of the relation. nationality_id is the candidate key. This relationship also has no foreign keys.

The functional dependencies are as follows:

- `nationality_id` → `nationality_name`

All the functional dependencies in the tables exhibit candidate keys on the left-hand side, and none of these dependencies are trivial.

If we look at attribute closure of nationality_id we can see that $\{\text{nationality\_id}\}^+$ = {nationality_id, nationality_name}. This confirms that the table meets the requirements of BCNF.

5.2. leagues

It has league_id, league_name, league_level, and nationality_id as attributes of the relation. league_id is the candidate key. This relationship also has one foreign key nationality_id which refers to countries.

The functional dependencies are as follows:

- `league_id` → `league_name`
- `league_id` → `league_level`
- `league_id` → `nationality_id`

All the functional dependencies in the tables exhibit candidate keys on the left-hand side, and none of these dependencies are trivial.

If we look at attribute closure of league_id we can see that $\{\text{league\_id}\}^+$ = {league_id, league_name, league_level, nationality_id}. This confirms that the table meets the requirements of BCNF.

5.3. coaches

It has coach_id, short_name, long_name, dob, age, and nationality_id as attributes of the relation. coach_id is the candidate key. This relationship also has one foreign nationality_id which refers to countries.

The functional dependencies are as follows:

- `coach_id` → `short_name`
- `coach_id` → `long_name`
- `coach_id` → `dob`
- `coach_id` → `age`
- `coach_id` → `nationality_id`

All the functional dependencies in the tables exhibit candidate keys on the left-hand side, and none of these dependencies are trivial.

If we look at attribute closure of coach_id we can see that $\{\text{coach\_id}\}^+$ = {coach_id, short_name, long_name, dob, age, nationality_id}. This confirms that the table meets the requirements of BCNF.

5.4. teams

It has team_id, team_name, league_id, overall, midfield, defence, coach_id, home_stadium, rival_team, international_prestige, domestic_prestige, club_worth_eur, starting_xi_average_age, whole_team_average_age, and captain as attributes of the relation. team_id is the candidate key. This relationship also has three foreign keys league_id, coach_id, and captain which refer to leagues, coaches, and players respectively.

The functional dependencies are as follows:

- `team_id` → `team_name`
- `team_id` → `league_id`
- `team_id` → `overall`
- `team_id` → `midfield`
- `team_id` → `defence`
- `team_id` → `coach_id`
- `team_id` → `home_stadium`
- `team_id` → `rival_team`
- `team_id` → `international_prestige`
- `team_id` → `domestic_prestige`
- `team_id` → `club_worth_eur`
- `team_id` → `starting_xi_average_age`
- `team_id` → `whole_team_average_age`
- `team_id` → `captain`

All the functional dependencies in the tables exhibit candidate keys on the left-hand side, and none of these dependencies are trivial.

If we look at attribute closure of team_id we can see that $\{\text{team\_id}\}^+$ = {team_id, team_name, league_id, overall, midfield, defence, coach_id, home_stadium, rival_team, international_prestige, domestic_prestige, club_worth_eur, starting_xi_average_age, whole_team_average_age, captain}. This confirms that the table meets the requirements of BCNF.

5.5. team_stats

It has team_impact_index, team_id, team_impact_factor, and team_impact_score as attributes of the relation. team_impact_index is the candidate key. This relationship also has one foreign key team_id which refers to teams.

The functional dependencies are as follows:

- `team_impact_index` → `team_id`
- `team_impact_index` → `impact_factor`
- `team_impact_index` → `impact_score`

All the functional dependencies in the tables exhibit candidate keys on the left-hand side, and none of these dependencies are trivial.

If we look at attribute closure of team_impact_index we can see that $\{$team_impact_index$\}^+$ = $\{$team_impact_index, team_id, team_impact_factor,team_impact_score$\}$. This confirms that the table meets the requirements of BCNF.

5.6. players

It has player_id, short_name, long_name, overall, potential, value_eur, wage_eur, age, dob, height_cm, weight_kg, club_id, national_team_id, nationality_id, and player_face_url as attributes of the the relation. player_id is the candidate key. This relationship also has three foreign keys club_team_id, and national_team_id both referring to teams, and nationality_id referring to countries.

The functional dependencies are as follows:

- `player_id` → `short_name`
- `player_id` → `long_name`
- `player_id` → `overall`
- `player_id` → `potential`
- `player_id` → `value_eur`
- `player_id` → `wage_eur`
- `player_id` → `age`
- `player_id` → `dob`
- `player_id` → `height_cm`
- `player_id` → `weight_kg`
- `player_id` → `club_id`
- `player_id` → `national_team_id`
- `player_id` → `nationality_id`
- `player_id` → `player_face_url`

All the functional dependencies in the tables exhibit candidate keys on the left-hand side, and none of these dependencies are trivial.

If we look at attribute closure of player_id we can see that $\{$player_id$\}^+$ = $\{$player_id, short_name, long_name, overall, potential, value_eur, wage_eur, age, dob, height_cm, weight_kg, club_id, national_team_id, nationality_id, player_face_url$\}$. This confirms that the table meets the requirements of BCNF.

5.7. player_stats

It has player_impact_index, player_id, player_impact_factor, and player_impact_score as attributes of the relation. player_impact_index is the candidate key. This relationship also has one foreign key player_id which refers to players.

The functional dependencies are as follows:

- `player_impact_index` → `player_id`
- `player_impact_index` → `impact_factor`
- `player_impact_index` → `impact_score`

All the functional dependencies in the tables exhibit candidate keys on the left-hand side, and none of these dependencies are trivial.

If we look at attribute closure of player_impact_index we can see that $\{$player_impact_index$\}^+$ = $\{$player_impact_index, player_id, player_impact_factor,player_impact_score$\}$. This confirms that the table meets the requirements of BCNF.

6. ER Diagram

The ER Diagram consists of seven distinct entities and encompasses seven binary associations that interlink them.

These Seven entities include:

Countires: This entity represents the countries with which players, teams, or coaches may be associated.

Leagues: This entity encompasses comprehensive data about prominent football leagues located in diverse countries across the globe.

Coaches: Within this entity, one can access valuable information regarding numerous coaches who oversee the teams featured in the ”Teams” entity.

Teams: Comprising information regarding both domestic clubs and international teams from around the world.

Team Stats: This entity stores statistical information regarding teams.

Players: This entity serves as a repository of comprehensive information about players worldwide. It encompasses details on their skills, traits, and personal backgrounds.

Player Stats: This entity stores statistical information regarding players.

The seven relationships include:

participates: The relationship between ***leagues*** and ***teams*** is one-to-many, meaning a league can have multiple teams participating in it.

coaches: The relationship between ***teams*** and ***coaches*** is one-to-many, meaning a coach can train multiple teams.

plays: The relationship between ***teams*** and ***players*** is one-to-many, meaning a player can play for multiple teams.

is captained by: The relationship between ***teams*** and ***players*** is one-to-one, meaning a team can have only one captain.

represents: Captures the relationship from ***teams***, ***players*** and ***coaches*** to ***countires*** is one-to-one, meaning a team, coach and player can only one represent one country.

has stats: Captures the relationship between ***teams*** and ***team_stats*** and also between ***players*** and ***player_stats*** and both are one-to-many relationships. This means a single player and team can have multiple stats corresponding to them.

The ER diagram outlines a soccer database with entities for players, teams, coaches, leagues, and countries, capturing details from personal statistics to league data. It shows key relationships like player affiliations and team league participation, and uses primary and foreign keys for data integrity, indicating a well-structured and efficient database design.

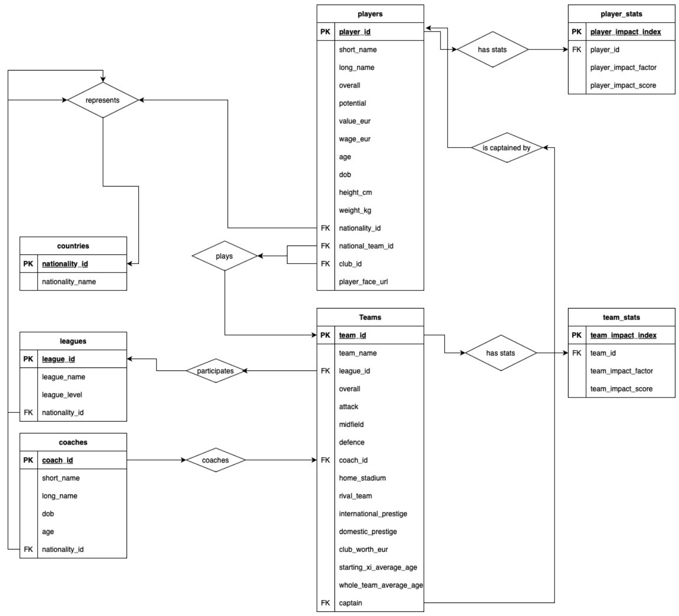

Figure 1. ER Diagram of FIFA 23 Database

7. Challenges with Large Datasets and Solutions through Indexing

Handling large datasets often presents unique challenges, such as slow query performance and difficulties in data management. To address these issues, adopting indexing strategies is a common and effective approach. Indexing significantly improves the speed of data retrieval operations by creating an efficient path to access the data.

During my experience with large datasets, the primary challenges faced included prolonged data retrieval times and inefficient execution of queries, which led to bottlenecks in data processing and analytics tasks. To solve these issues, specific types of indexes, such as B-tree indexes for range queries and hash indexes for equality searches, were implemented. Additionally, considering the balance between the speed of read operations and the overhead of maintaining indexes during write operations was crucial.

we're working with a database that includes a particularly large table named player_stats, which contains approximately 4.1 million records. This table, by far, overshadows the others in size. An initial query was performed to extract data, structured as follows:

```
SELECT players.short_name, player_stats.player_impact_score
FROM players
INNER JOIN player_stats
ON players.player_id = player_stats.player_id
where player_stats.player_impact_factor = 'skill_moves';
```

This SQL query aims to join the players table with the player_stats table, focusing specifically on rows where the player_impact_factor is equal to skill_moves. It retrieves the short names of the players along with their corresponding impact scores based on this criterion. Initially, executing this query took about 208 milliseconds and returned 57,646 rows.

To optimize this process, an index was created on the player_stats table. The index, named player_stats_index, was specifically designed to include both player_id and player_impact_factor as its key columns:

```
CREATE INDEX player_stats_index ON player_stats(player_id, player_impact_factor);
```

The creation of this index significantly improved the performance of the query. The execution time dramatically decreased to just 28 milliseconds, while still successfully fetching the same number of rows (57,646). This improvement illustrates the effectiveness of indexing in databases, especially when dealing with large datasets and complex queries that involve joins and specific condition filters. By structuring the index to align with the query's join and where-clause conditions, the database engine was able to retrieve the required data much more efficiently.

8. Database Operations and Query Performance Evaluation

8.1. Select Queries

- Show all the players from **Spain**

```
SELECT
    p.*
FROM
    players p
JOIN
    countries c
    ON p.nationality_id = c.nationality
WHERE
    c.nationality_name = 'Spain';
```

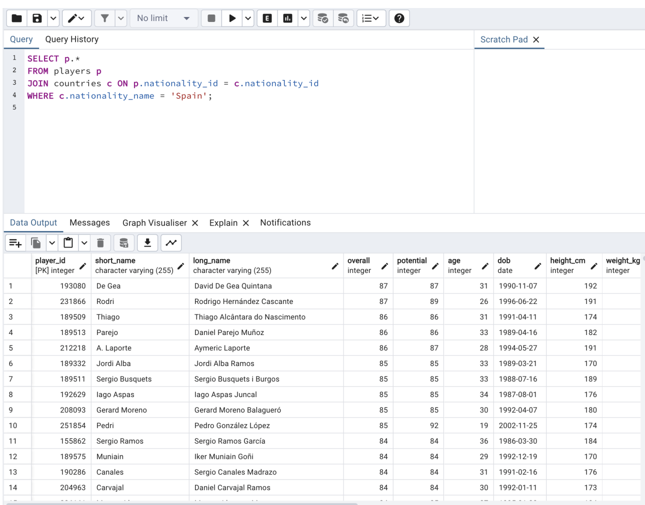

Figure 2. Select Query 1: Show all the players from Spain

- Select all teams in the league **Premier League** and order them according to overall performance

```
SELECT
    *
FROM
    teams t
JOIN
    leagues l
    ON t.league_id = l.league_id
WHERE
    league_name = 'Premier League'
ORDER BY
    overall DESC;
```

- Get the average age of players in each team.

```
SELECT
    t.team_name,
    AVG(p.age) AS average_age
FROM
    players p
```

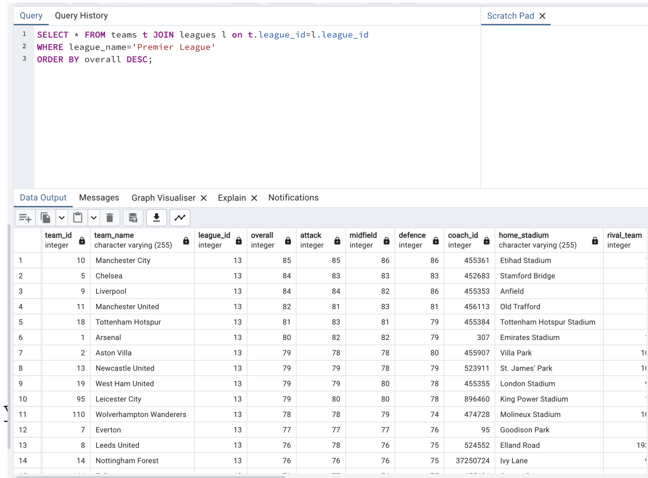

Figure 3. Select Query 2: Select all teams in the league Premier League and order them according to overall performance

```
JOIN
    teams t
    ON p.club_id = t.team_id
GROUP BY
    t.team_name;
```

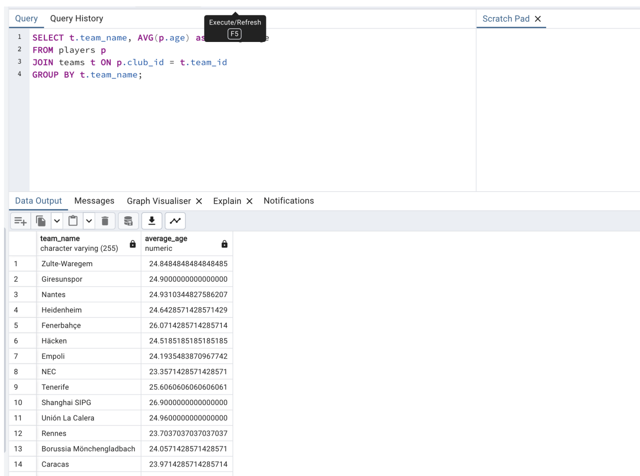

Figure 4. Select Query 3: Get the average age of players in each team.

- Find the highest-paid player in each team.

```
SELECT
    t.team_name,
    max_player.short_name,
    max_player.wage_eur
FROM
    teams t
JOIN (
    SELECT
        club_id,
        short_name,
        MAX(wage_eur) AS wage_eur
```

```
    FROM
        players
    GROUP BY
        club_id,
        short_name
) max_player
ON t.team_id = max_player.club_id;
```

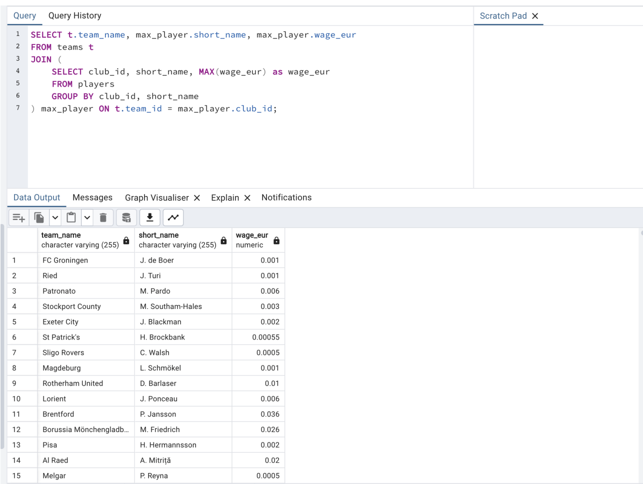

Figure 5. Select Query 4: Find the highest-paid player in each team.

- Find the most profitable team considering salaries as spending and ‘club_worth_eur‘ as the total value of income

```
SELECT
    t.team_name,
    t.club_worth_eur -
    COALESCE(p.total_salaries, 0)
    AS profitability
FROM
    teams t
LEFT JOIN (
    SELECT
        club_id,
        SUM(wage_eur)
        AS total_salaries
    FROM
        players
    GROUP BY
        club_id
) p
ON t.team_id = p.club_id
WHERE
    p.total_salaries IS NOT NULL
ORDER BY
    profitability DESC
LIMIT 1;
```

- List top 10 youngest players.

```
SELECT short_name, age
FROM players
```

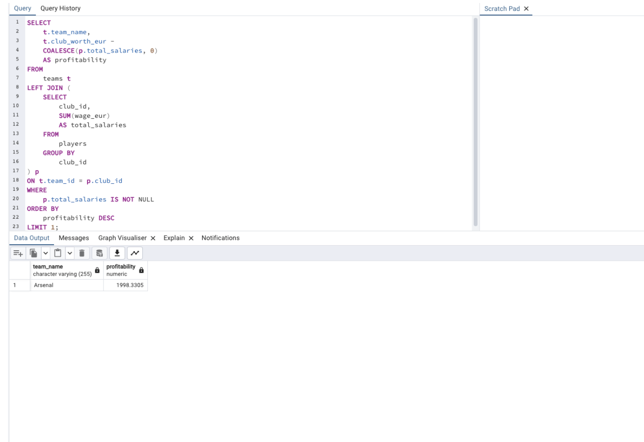

Figure 6. Select Query 5: Find the most profitable team.

```
ORDER BY age ASC
LIMIT 10;
```

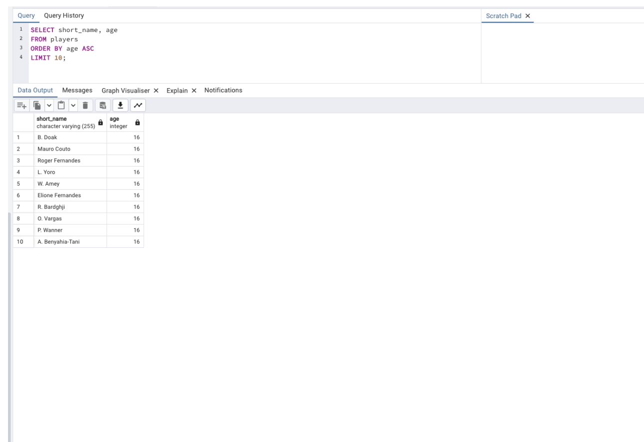

Figure 7. Select Query 6: List the top 10 youngest players.

- Find the team with the most experienced players.

```
SELECT
    t.team_name,
    AVG(p.age) AS average_age
FROM
    players p
JOIN
    teams t ON p.club_id = t.team_id
GROUP BY
    t.team_name
ORDER BY
    average_age DESC
LIMIT 1;
```

8.2. Insert Queries

- Add new player.

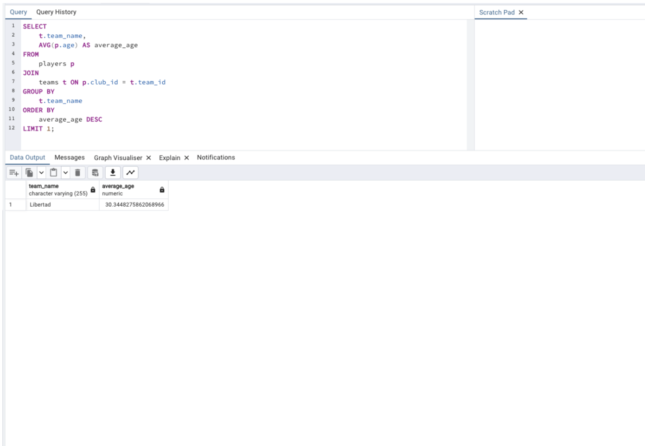

Figure 8. Select Query 7: Find the team with the most experienced players.

```
INSERT INTO players
     (player_id, short_name,
     long_name, overall,
     potential, age, dob,
     height_cm, weight_kg,
     value_eur, wage_eur,
     club_id, national_team_id,
     nationality_id, player_face_url)
VALUES
     (201, 'J. Doe',
     'John Doe', 85, 90,
     25, '1995-01-01',
     180, 75, 55000000,
     250000, 1, 10, 5,
     'http://example/jdoe_face.jpg');
```

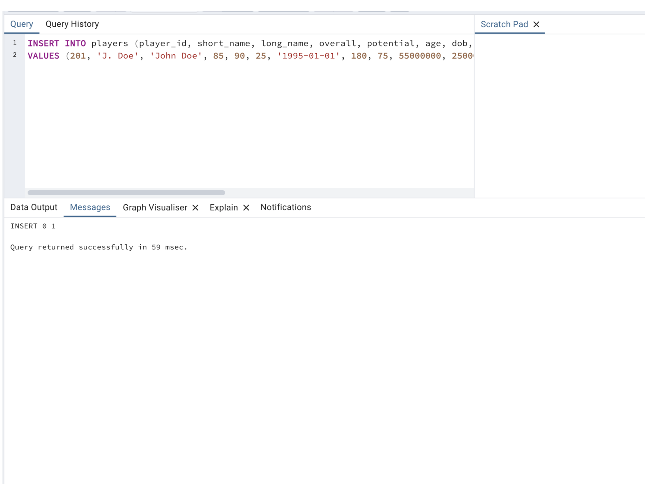

Figure 9. Insert Query 1: Add new player.

- Add a new team.

```
INSERT INTO teams
     (team_id, team_name,
     league_id, overall,
     attack, midfield,
     defence, coach_id,
     home_stadium, rival_team,
     international_prestige,
     domestic_prestige,
      club_worth_eur,
      starting_xi_average_age,
      whole_team_average_age,
      captain)
VALUES
     (201, 'FC Example', 1,
     80, 81, 79, 78, 1,
     'Example Stadium', 102,
     5, 4, 800000000,
     26.5, 27, 1001);
```

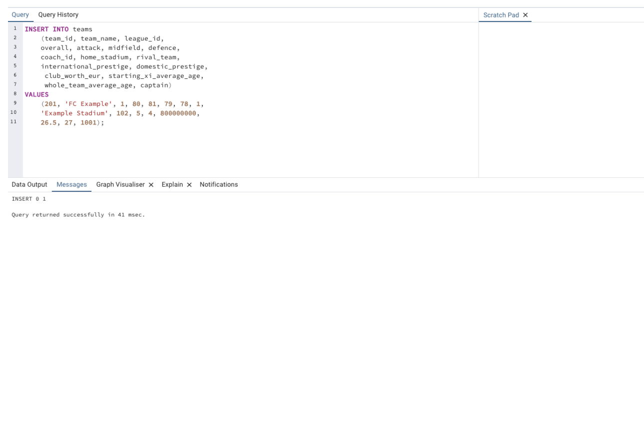

Figure 10. Insert Query 2: Add new team.

8.3. Delete Queries

- Delete added team.

```
DELETE FROM teams
WHERE team_id = 201;
```

- Delete added player.

```
DELETE FROM players
WHERE player_id = 201;
```

8.4. Update Queries

- Update player_id 1179 overall performance to 88.

```
UPDATE players
SET overall = 88
WHERE player_id = 1179;
```

- Update the coach of Real Madrid.

```
UPDATE teams
SET coach_id =
(SELECT coach_id
```

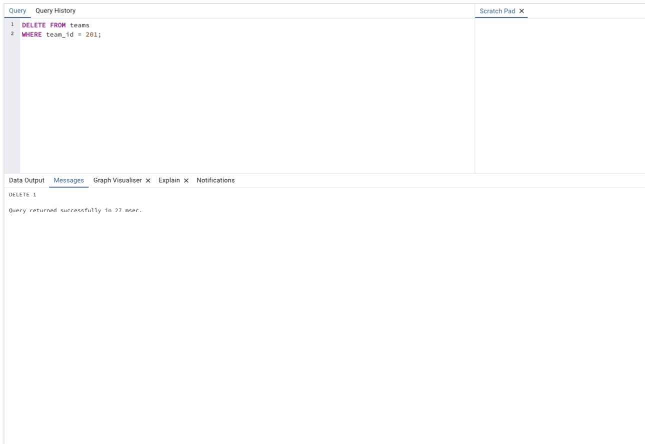

Figure 11. Delete Query 1: Delete added team

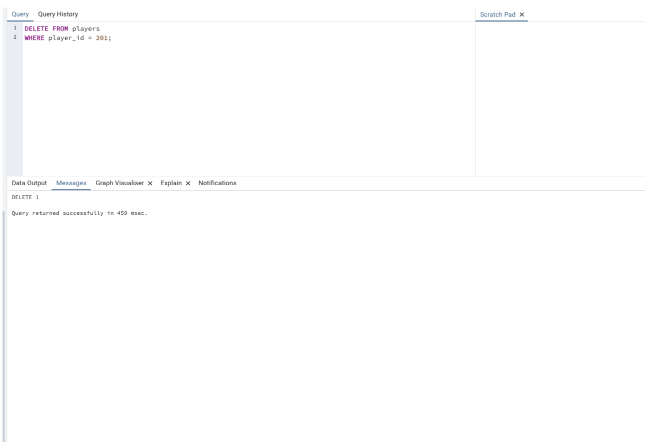

Figure 12. Delete Query 1: Delete added player

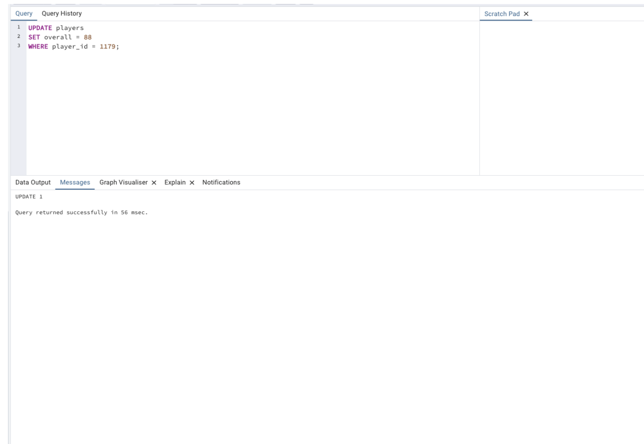

Figure 13. Update Query 1: Update player_id 1179 overall performance to 88.

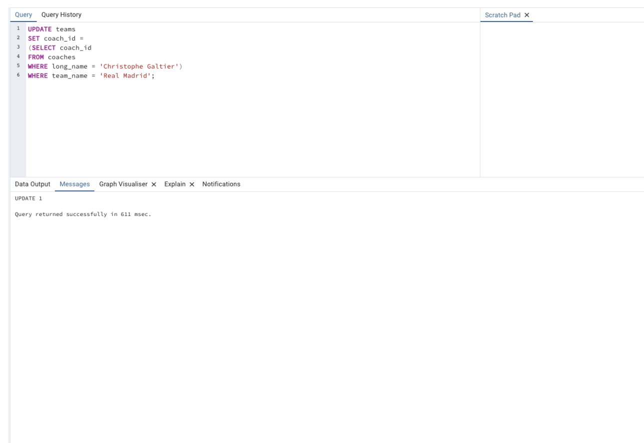

Figure 14. Update Query 2: Update the coach of Real Madrid.

```
    FROM coaches
    WHERE long_name = 'Christophe Galtier')
    WHERE team_name = 'Real Madrid';
```

9. Query Execution Analysis

9.1. Query Involving Multiple Joins

Query:

```
SELECT
    p.short_name,
    p.long_name,
    t.team_name,
    l.league_name
FROM
    players p
JOIN
    teams t
    ON p.club_id = t.team_id
JOIN
    leagues l
    ON t.league_id = l.league_id
WHERE
    l.league_name = 'Premier League';
```

Improvement Plan: Add indexes to ‘club_id‘ in ‘players‘ and ‘league_id‘ in ‘teams‘. Use ‘EXPLAIN‘ to ensure the query uses these indexes.

9.2. Aggregation Query on Player or Team Stats

Query:

```
SELECT
    t.team_name,
    AVG(ps.player_impact_score)
    AS avg_impact_score
FROM
    team_stats ts
JOIN
    teams t
    ON ts.team_id = t.team_id
```

```
GROUP BY
    t.team_name;
```

Improvement Plan: Add an index to ‘team_id‘ in ‘team_stats‘. Ensure the ‘player_impact_score‘ is stored in an efficient format.

9.3. Query Involving Date or Age Calculations

Query:

```
SELECT
    short_name,
    long_name,
    age,
    dob
FROM
    players
WHERE
    dob BETWEEN '1990-01-01'
    AND '2000-12-31';
```

Improvement Plan: Add an index on ‘dob‘. Consider storing pre-calculated ages.

10. Conclusion

The project aims to address the inefficiencies and data integrity issues inherent in managing football player data through Excel files by creating a structured database. This database offers numerous advantages, such as improved efficiency, data consistency, security, scalability, data analysis capabilities, data quality, and collaboration opportunities. By centralizing and structuring player data, the project not only facilitates smarter decision-making for talent scouts, coaches, managers, and analysts but also has the potential to transform the way football teams handle player information, making them more competitive and successful in the highly data-driven world of football. Furthermore, transforming the initial dataset into Boyce-Codd Normal Form (BCNF) ensures data integrity and relational efficiency, particularly in the Players and Teams tables, which were appropriately normalized to meet BCNF criteria.

11. Visualization

Integrating Tableau into your database project enhances data analysis and presentation by allowing for the creation of interactive, visually appealing dashboards. Key considerations include preparing clean, well-organized data, designing comprehensive dashboards with various charts and graphs, ensuring real-time data updates, and maintaining accessibility for users. It’s also important to adhere to security and compliance standards, provide necessary training for your team, and continually iterate the visualizations based on user feedback. This integration not only simplifies data interpretation but also transforms complex data sets into actionable insights for decision-making.

Tableau Dashboard Link

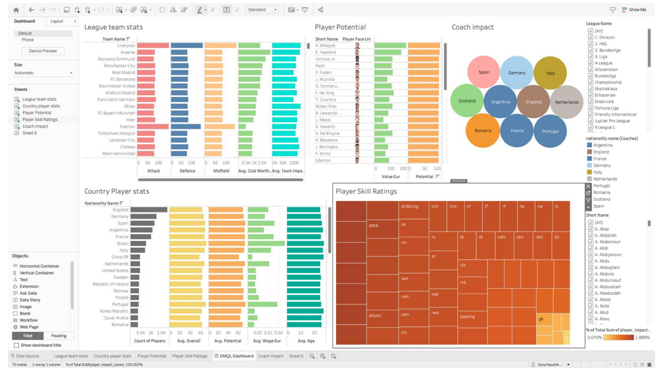

Figure 15. Tableau Visualization

12. Contributions

1) **Guna Kaushik Undru:**
   - *Documentation:* Prepared project documentation including reports.
   - *Database Design:* Designed the database schema and relationships.
   - *Visualization:* Created visual representations of data for better insights and understanding in Tableau.
2) **Shri Harsha Adapala:**
   - *Documentation:* Prepared project documentation including reports.
   - *Database Design:* Designed the database schema and relationships.
   - *Query Analysis:* Analyzed and optimized database queries for efficiency.
3) **Preethi Sree Allam:**
   - *Documentation:* Prepared project documentation including reports.
   - *Database Design:* Designed the database schema and relationships.
   - *Query Analysis:* Analyzed and optimized database queries for efficiency.

Acknowledgments

We would like to thank EA Sports and Kaggle for providing access to Players’ Data

References

[1] Kaggle Dataset: FIFA Players. Retrieved from https://www.kaggle.com/datasets/joebeachcapital/fifa-players