CSE 560 PROJECT REPORT

Gridiron Galaxy: A Quantum Leap into Football Data Management

Name of the team: **NBK**

Harshavardhan Reddy Nadedi
50559985
hnadedi@buffalo.edu

Sai Ganesh Kishore Babu
5056555
saiganes@buffalo.edu

I. Introduction

A. BACKGROUND

Football has millions of players, fans, and stakeholders engaged in many facets of the game, making it a large and vibrant universe. Football-related data includes information about leagues, teams, players, and games. Transfermarkt is one of the most well-known sites for football statistics, with a huge dataset that includes detailed data on various topics. By extracting data from Transfermarkt and organizing it into user-friendly CSV files, this dataset enables users to research, examine, and display football-related statistics.

B. PROBLEM STATEMENT

Football data is complicated and distributed over several sources. Because of the size and lack of organization, it might be difficult to access this data and derive useful insights. In order to facilitate the analysis of trends, player performance, team dynamics, and other football-related topics by academics, analysts, and aficionados, it is necessary to provide a clear, organized, and current dataset.

C. OBJECTIVES

To combine Transfermarkt data into a single, comprehensive dataset that includes information on players, clubs, games, and associated characteristics. to produce a dataset that is simple to use, access, and comprehend so that a variety of users may interact with the information. to guarantee that the dataset is updated frequently using the most recent Transfermarkt statistics. to provide a wide range of analytics and insights by letting users combine data across multiple entities (e.g., linking players to their clubs and games).

D. TARGET USER

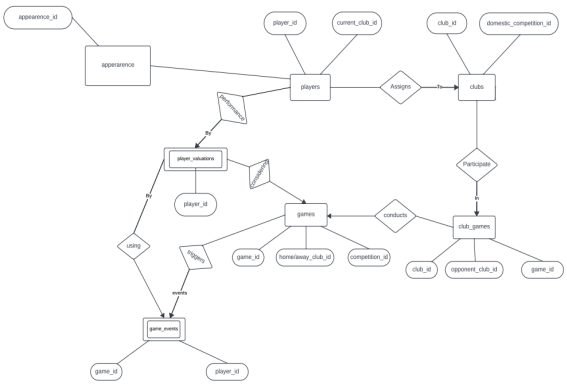
The Football Database Management System (DBMS) is a complete tool that makes professional football player acquisition, team performance analysis and strategic decision making easier. Football clubs, managers, agents, analysts, and enthusiasts are the target audience for this system. It includes features like player statistics tracking, talent scouting, contract negotiation assistance, match analysis, and data-driven decision support. The Database enables stakeholders to maximize players, enhance team competitiveness and push success both on and off the field with its user-friendly interface and powerful analytics tools.

E. REAL-LIFE SCENARIO:

People who are interested in using football data analysis to learn more about team tactics, player performance, or patterns in the game. Media professionals in need of football-related information for blogs, news pieces, and other media. supporters who take pleasure in studying football trends and statistics in order to broaden their knowledge and comprehension of the game. Those who want to use football data for more in-depth analysis or machine learning research in order to spot trends or forecast outcomes.

F. CHOOSING DATABASE OVER EXCEL FILE: When it comes to data management, consistency, security, scalability, and sharing, databases—with their strong architecture—offer a significant benefit over Excel files. Large volumes of data and complicated queries might be difficult for Excel to handle, even though it could be ideal for simpler analytics and smaller datasets. Databases, on the other hand, are made to easily manage large datasets and complex data interactions. In order to enable more in-depth research and smart decision-making, they support complicated searches and let users generate reports and visualizations. Additionally, databases concentrate data, which lowers the possibility of loss, a concern that arises when Excel files are kept in different places. In addition to ensuring data integrity, this centralized method facilitates user cooperation and access. Databases can also impose restrictions like main keys. Also, databases have the ability to set limitations such as main keys and foreign keys, which ensure correctness and consistency even with increasing datasets. Databases are perfect for sophisticated analytics and machine learning because they can handle complicated data conversions and calculations. For football clubs, managers, agents, analysts, and enthusiasts looking for a dependable, safe, and expandable platform for organizing and analyzing substantial amounts of football-related data, this makes them the go-to option.

ER DIAGRAM:



DESCRIPTIONS OF RELATIONS AND THEIR ATTRIBUTES:

The following is a description of each relation and its attributes of the dataset we used for inventory management and online order tracking:

APPEARANCES TABLE: This table records individual player appearances in various football games. Its attributes are as follows:

- appearance_id: The unique identifier for a player's appearance in a specific game.
- game_id: The identifier of the game in which the player appeared (foreign key from the "games" table).
- player_id: The identifier for the player who made the appearance (foreign key from the "players" table).
- player_club_id: The identifier of the club to which the player belonged at the time of the game.
- player_current_club_id: The identifier of the player's current club at the time of data retrieval.
- date: The date of the game.
- player_name: The name of the player.
- competition_id: The identifier for the competition in which the game took place (foreign key from the "competitions" table).
- yellow_cards: The number of yellow cards the player received during the game.
- red_cards: The number of red cards the player received during the game.
- goals: The number of goals scored by the player during the game.
- assists: The number of assists made by the player during the game.
- minutes_played: The number of minutes the player played in the game.

CLUB GAMES TABLE: This table records information about club performances in football games. Its attributes are as follows:

- game_id: The unique identifier for the game (foreign key from the "games" table).

- club_id: The identifier for the club that played in the game.
- own_goals: The number of goals scored by the club.
- own_position: The position of the club in the league or competition.
- own_manager_name: The name of the club's manager.
- opponent_id: The identifier for the opposing club (foreign key from the "clubs" table).
- opponent_goals: The number of goals scored by the opposing club.
- opponent_position: The position of the opposing club in the league or competition.
- opponent_manager_name: The name of the opposing club's manager.
- hosting: Specifies if the club was hosting the game ("home" or "away").
- is_win: Indicates if the club won the game.

CLUBS TABLE: This table holds information about football clubs. Its attributes are as follows:

- club_id: The unique identifier for the club.
- club_code: The unique code for the club.
- name: The name of the club.
- domestic_competition_id: The identifier for the domestic competition the club is a part of.
- squad_size: The number of players in the club's squad.
- average_age: The average age of the club's players.
- foreigners_number: The number of foreign players in the club.
- foreigners_percentage: The percentage of foreign players in the club.
- national_team_players: The number of players from the club who play for their national teams.
- stadium_name: The name of the club's stadium.
- stadium_seats: The number of seats in the stadium.
- net_transfer_record: The club's net transfer record.
- last_season: The club's most recent season in the competition.
- url: The club's website or related URL.

COMPETITIONS TABLE: This table contains information about football competitions. Its attributes are as follows:

- competition_id: The unique identifier for the competition.
- competition_code: A code representing the competition.
- name: The name of the competition.
- sub_type: The sub-type or category of the competition.
- type: The main type of the competition (e.g., league, cup, international).
- country_id: The identifier for the country where the competition is held.
- country_name: The name of the country.
- domestic_league_code: The domestic league code for the competition.
- confederation: The confederation to which the competition belongs.
- url: A URL with information about the competition.

GAME EVENTS TABLE: This table records events that occur during football games. Its attributes are as follows:

- game_id: The identifier for the game where the event occurred (foreign key from the "games" table).
- minute: The minute in which the event occurred.
- type: The type of event (e.g., goal, yellow card, red card, substitution).
- club_id: The identifier for the club involved in the event.
- player_id: The identifier for the player involved in the event.
- description: A description of the event.
- player_in_id: The identifier for the player who was substituted in, if applicable.

GAMES TABLE: This table represents individual football games. Its attributes are as follows:

- game_id: The unique identifier for the game.
- competition_id: The identifier for the competition in which the game was held.
- season: The season during which the game took place.

- round: The round or stage of the competition.
- date: The date of the game.
- home_club_id: The identifier for the home club.
- away_club_id: The identifier for the away club.
- home_club_goals: The number of goals scored by the home club.
- away_club_goals: The number of goals scored by the away club.
- home_club_position: The position of the home club in the league or competition.
- away_club_position: The position of the away club in the league or competition.
- home_club_manager_name: The name of the home club's manager.
- away_club_manager_name: The name of the away club's manager.
- stadium: The name of the stadium where the game was held.
- attendance: The number of spectators at the game.
- referee: The name of the referee for the game.
- url: A URL with information about the game.

PLAYER VALUATIONS TABLE: This table contains information on player market valuations. Its attributes are as follows:

- player_id: The unique identifier for the player.
- last_season: The most recent season for which the valuation is relevant.
- datetime: The date and time when the valuation was recorded.
- date: The specific date when the valuation was recorded.
- dateweek: The week of the year when the valuation was recorded.
- market_value_in_eur: The player's market value in euros.
- current_club_id: The identifier for the player's current club.
- player_club_domestic_competition_id: The identifier for the club's domestic competition.

PLAYERS TABLE: This table contains information about individual football players. Its attributes are as follows:

- player_id: The unique identifier for the player.
- first_name: The player's first name.
- last_name: The player's last name.
- name: The player's full name.
- last_season: The most recent season for which the player's information is relevant.
- current_club_id: The identifier for the player's current club.
- player_code: A unique code for the player.
- country_of_birth: The country where the player was born.
- city_of_birth: The city where the player was born.
- country_of_citizenship: The player's citizenship.
- date_of_birth: The player's date of birth.
- sub_position: The player's sub-position within the team.
- position: The player's main position within the team.
- foot: The player's preferred foot (left or right).
- height_in_cm: The player's height in centimeters.
- market_value_in_eur: The player's market value in euros.
- highest_market_value_in_eur: The highest market value in euros the player has achieved.
- contract_expiration_date: The date when the player's contract expires.
- agent_name: The name of the player's agent.
- image_url: A URL with the player's image.
- url: A URL with information about the player.

IV. Tables And Keys

Tables	Keys
Players	Primary key: player_id Foreign key: current_club_id reference to Clubs relation
Clubs	Primary key: club_id Foreign key: domestic_competition_id references Competitions
Competitions	Primary key: competition_id
Games	Primary key: game_id Foreign keys: competition_id references Competitions home_club_id, away_club_id reference Clubs

Appearances	Primary key: appearance_id Foreign keys: game_id references Games player_id references Players
Club Games	Foreign keys: game_id references Games club_id, opponent_id reference Clubs
Game Events	Foreign keys: game_id references Games club_id references Clubs player_id references Players
Player Valuations	Foreign keys: player_id references Players current_club_id references Clubs

DATABASE SCHEMAS:

- APPEARANCES TABLE (appearance_id VARCHAR primary key, game_id INTEGER foreign key, player_id INTEGER foreign key, player_club_id INTEGER, player_current_club_id INTEGER, date DATE, player_name VARCHAR, competition_id VARCHAR, yellow_cards INT, red_cards INT, goals INT, assists INT, minutes_played INT)
- CLUB_GAMES TABLE (game_id INTEGER foreign key, club_id INTEGER foreign key, own_goals INT, own_position INT, own_manager_name VARCHAR, opponent_id INTEGER foreign key, opponent_goals INT, opponent_position INT, opponent_manager_name VARCHAR, hosting VARCHAR, is_win INT)
- CLUBS TABLE (club_id INTEGER primary key, club_code VARCHAR, name VARCHAR, domestic_competition_id VARCHAR foreign key, squad_size INT, average_age NUMERIC, foreigners_number INT, foreigners_percentage NUMERIC, national_team_players INT, stadium_name VARCHAR, stadium_seats INT, net_transfer_record VARCHAR, last_season INT, url VARCHAR)
- COMPETITIONS TABLE (competition_id VARCHAR primary key, competition_code VARCHAR, name VARCHAR, sub_type VARCHAR, type VARCHAR, country_id INTEGER, country_name VARCHAR, domestic_league_code VARCHAR, confederation VARCHAR, url VARCHAR)
- GAME_EVENTS TABLE (game_id INTEGER foreign key, minute INT, type VARCHAR, club_id INTEGER foreign key, player_id INTEGER foreign key, description VARCHAR, player_in_id INTEGER)
- GAMES TABLE (game_id INTEGER primary key, competition_id VARCHAR foreign key, season INT, round VARCHAR, date DATE, home_club_id INTEGER foreign key, away_club_id INTEGER foreign key, home_club_goals INT, away_club_goals INT, home_club_manager_name VARCHAR, away_club_manager_name VARCHAR, stadium VARCHAR, attendance NUMERIC, referee VARCHAR, url VARCHAR, home_club_name VARCHAR, away_club_name VARCHAR, aggregate VARCHAR, competition_type VARCHAR)
- PLAYER_VALUATIONS TABLE (player_id INTEGER foreign key, last_season INT, datetime TIMESTAMP, date DATE, dateweek DATE, market_value_in_eur VARCHAR, n INT, current_club_id INTEGER foreign key, player_club_domestic_competition_id VARCHAR)
- PLAYERS TABLE (player_id INTEGER primary key, first_name VARCHAR, last_name VARCHAR, name VARCHAR, last_season INT, current_club_id INTEGER foreign key, player_code VARCHAR, country_of_birth VARCHAR, city_of_birth VARCHAR, country_of_citizenship VARCHAR, date_of_birth DATE, sub_position VARCHAR, position VARCHAR, foot VARCHAR, height_in_cm INT, market_value_in_eur NUMERIC, highest_market_value_in_eur NUMERIC, contract_expiration_date DATE, agent_name VARCHAR, image_url VARCHAR, url VARCHAR,

current_club_domestic_competition_id VARCHAR)

RELATION BETWEEN TABLES:

- Players and Clubs:
  - The PLAYERS table has a foreign key current_club_id referencing the CLUBS table's club_id. This indicates that a player is associated with a specific club.
- Players and Appearances:
  - The APPEARANCES table has a foreign key player_id referencing the PLAYERS table's player_id. This shows that each appearance record is tied to a specific player.
- Clubs and Competitions:
  - The CLUBS table has a foreign key domestic_competition_id referencing the COMPETITIONS table's competition_id. This indicates which competition each club is a part of.
- Games and Competitions:
  - The GAMES table has a foreign key competition_id referencing the COMPETITIONS table's competition_id. This shows which competition a game is part of.
- Games and Clubs:
  - The GAMES table has two foreign keys, home_club_id and away_club_id, both referencing the CLUBS table's club_id. This establishes the relationship between games and the clubs participating in them.
- Appearances and Games:
  - The APPEARANCES table has a foreign key game_id referencing the GAMES table's game_id. This links each appearance to a specific game.
- Club Games and Games:
  - The CLUB_GAMES table has a foreign key game_id referencing the GAMES table's game_id. This establishes the connection between games and club-specific statistics.
- Club Games and Clubs:
  - The CLUB_GAMES table has a foreign key club_id referencing the CLUBS table's club_id. This indicates which club the game data belongs to.
- Game Events and Games:
  - The GAME_EVENTS table has a foreign key game_id referencing the GAMES table's game_id. This shows the events associated with a specific game.
- Game Events and Players:
  - The GAME_EVENTS table has a foreign key player_id referencing the PLAYERS table's player_id. This connects game events to the players involved.
- Player Valuations and Players:
  - The PLAYER_VALUATIONS table has a foreign key player_id referencing the PLAYERS table's player_id. This links market valuations to specific players.

II. NORMALIZATION

We are currently evaluating the degree of normalization for every relation in our database to ensure maximum performance and accurate data. This includes looking at our database tables' structure to find any places where there may be transitive dependencies, partial dependencies, or redundant data. We may reduce update anomalies, cut down on redundancy, and increase query efficiency by reaching a higher degree of standardization. By making sure that our database follows best practices in design

A. Raw Data

The dataset used in this study was sourced from Kaggle, a collaborative platform designed for data exchange and analysis. We chose this dataset because it is relevant to our field of study, containing various information about football games, player statistics, competitions, clubs, and club games from 2012 to the present. Kaggle's intuitive interface and diverse range of tools for data analysis and collaboration make it an excellent resource for researchers and analysts.

The data extracted from Kaggle serves as our primary dataset. We would like to thank Kaggle and its contributors for providing this valuable resource. While most of the data we needed was available through Kaggle, some inconsistencies and redundancy in the dataset required us to modify and adjust certain entries, especially those related to player game statistics

player_id	first_name	last_name	name	last_season	current_club_id	player_code	country_of_birth
city_of_birth	country_of_citizenship	date_of_birth	sub_position	position	foot	height_in_cm	market_value_in_eur
highest_market_value_in_eur	contract_expiration_date	agent_name	image_url	url	current_club_domestic_competition_id	current_club_name	

Table 1 Attributes in Players

player_id	last_season	datetime	date	dateweek	market_value_in_eur	n
current_club_id	player_club_domestic_competition_id					

Table 2 Attributes in player_valuations

game_id	competition_id	season	round	date	home_club_id
away_club_id	home_club_goals	away_club_goals	home_club_position	away_club_position	home_club_manager_name
away_club_manager_name	stadium	attendance	referee	url	home_club_name
away_club_name	aggregate	competition_type			

Table 3 Attributes in games

game_id	minute	type	club_id	player_id	description	player_in_id

Table 4 Attributes in game_events

competition_id	competition_code	name	sub_type	type
country_id	country_name	domestic_league_code	confederation	url

Table 5 Attributes in competitions

club_id	club_code	name	domestic_competition_id	squad_size	average_age	foreigners_number
foreigners_percentage	national_team_players	stadium_name	stadium_seats	net_transfer_record	last_season	url

Table 6 Attributes in clubs

game_id	club_id	own_goals	own_position	own_manager_name	opponent_id
opponent_goals	opponent_position	opponent_manager_name	hosting	is_win	

Table 7 Attributes in club_games

appearance_id	game_id	player_id	player_club_id	player_current_club_id	date	minutes_played
player_name	competition_id	yellow_cards	red_cards	goals	date	

Table 8 Attributes in appearances relation

B. 1NF (First Normal Form)

In my database, each cell in a table holds a single, indivisible value. Each column has a unique name, and every column contains a uniform data type. Because all these conditions are met, my database is in First Normal Form (1NF).

C. 2NF (Second Normal Form)

To apply the Second Normal Form (2NF), we need to ensure that the relations are in 1NF and that all non-key attributes are fully functionally dependent on the primary key. This means that any attribute that is dependent on only part of the primary key should be decomposed into a separate relation.

Functional Dependencies (FDs)*:* A functional dependency occurs when the value of one attribute (or a set of attributes) uniquely determines the value of another attribute in the same relation.

Formally, given a relation *R* with attributes A1,A2,... An, a functional dependency X→Y holds if, every pair of tuples *t*1 and t2 in R agree on the values of attributes in X, they must also agree on the values of attributes in Y.

We identify the below Functional Dependencies (FDs) for table appearances, club_games, clubs, competitions, game_events, games, players.

Appearances Table

Primary Key: *appearance_id*

Functional Dependencies:

appearance_id→ game_id, player_id, player_club_id, player_current_club_id, date, player_name, competition_id, yellow_cards, red_cards, goals, assists, minutes_played

Since appearance_id is the primary key, it determines all other attributes in this table.

Observations:

The *game_id* and *player_id* act as foreign keys, linking to the games and players tables, respectively.

competition_id also acts as a foreign key to the competitions table.

The player_club_id and player_current_club_id are also dependent on appearance_id.

club_games Table

Primary Key: This table does not have an explicitly defined primary key, but a composite key of game_id and club_id could act as a primary key.

Functional Dependencies:

game_id, club_id → own_goals, own_position, own_manager_name, opponent_id, opponent_goals, opponent_position, opponent_manager_name, hosting, is_win

If the primary key is a composite of game_id and club_id, it determines all other attributes in this table.

Observations:

The game_id is a foreign key to the games table, and club_id and opponent_id are foreign keys to the clubs table.

clubs Table

Primary Key: club_id

Functional Dependencies:

club_id→club_code,name, domestic_competition_id, squad_size, average_age, foreigners_number, foreigners_percentage, national_team_players, stadium_name, stadium_seats, net_transfer_record, last_season, url

club_id as the primary key determines all other attributes in this table.

Observations:

The domestic_competition_id is a foreign key to the competitions table.

All other attributes are determined by club_id.

competitions Table

Primary Key: competition_id

Functional Dependencies:

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

competition_id as the primary key determines all other attributes in this table.

Observations:

The country_id acts as a foreign key to another table related to countries (which isn't shown here).

game_events Table

Primary Key: This table does not have an explicitly defined primary key, but a composite key of game_id, minute, and type could act as a primary key.

Functional Dependencies:

game_id → minute, type, club_id, player_id, description, player_in_id

player_id→ minute, type, club_id, game_id, description, player_in_id

club_id→ minute, type, player_id, game_id, description, player_in_id

Given the implied composite key, it determines other attributes.

Observations:

game_id, club_id, and player_id are foreign keys linking to other tables.

games Table

Primary Key: game_id

Functional Dependencies:

game_id → competition_id, season, round, date, home_club_id, away_club_id, home_club_goals, away_club_goals, home_club_position, away_club_position, home_club_manager_name, away_club_manager_name, stadium, attendance, referee, url, home_club_name, away_club_name, aggregate, competition_type

game_id as the primary key determines all other attributes in this table.

Observations:

The competition_id, home_club_id, and away_club_id are foreign keys to other tables.

player_valuations Table

Primary Key: This table does not have an explicitly defined primary key, but a composite key of player_id and datetime could act as a primary key.

Functional Dependencies:

D. 2NF DECOMPOSITION

Analysis for 2NF for Each Relation

appearances:

Primary Key: appearance_id

Given that appearance_id is a single-column primary key, there are no partial dependencies.

2NF Status: This relation is in 2NF.

player_id, datetime → last_season, date, dateweek, market_value_in_eur, n, current_club_id, player_club_domestic_competition_id

A composite key of player_id and datetime could determine all other attributes in this table.

Observations:

player_id and current_club_id are foreign keys linking to other tables.

players Table

Primary Key: player_id

Functional Dependencies:

player_id → first_name, last_name, name, last_season, current_club_id, player_code, country_of_birth, city_of_birth, country_of_citizenship, date_of_birth, sub_position, position, foot, height_in_cm, market_value_in_eur, highest_market_value_in_eur, contract_expiration_date, agent_name, image_url, url, current_club_domestic_competition_id, current_club_name

player_id as the primary key determines all other attributes in this table.

Observations:

The current_club_id and current_club_domestic_competition_id are foreign keys to other tables.

club_games:

Primary Key: game_id, club_id (composite key)

There are no attributes that depend on just one part of the composite key. All attributes are dependent on both game_id and club_id.

2NF Status: This relation is in 2NF.

clubs:

Primary Key: club_id

With a single-column primary key, there can't be partial dependencies.

2NF Status: This relation is in 2NF.

competitions:

Primary Key: competition_id

Since there's only one column for the primary key, there's no chance of partial dependencies.

2NF Status: This relation is in 2NF.

game_events:

Primary Key: game_id, player_id, club_id.

Partial dependencies are not possible with a primary key that has only a single column.

Given the composite primary key, the dependencies are on all elements of the key.

2NF Status: This relation is in 2NF.

games:

Primary Key: game_id

With a single-column primary key, all other attributes are fully dependent on this key.

2NF Status: This relation is in 2NF.

player_valuations:

Primary Key: player_id, datetime (composite key)

Given the composite primary key, all attributes are dependent on the entire primary key.

2NF Status: This relation is in 2NF.

players:

Primary Key: player_id

The relation has a single-column primary key, ensuring no partial dependencies.

2NF Status: This relation is in 2NF.

Conclusion

Based on the primary keys and the functional dependencies, all the provided relations appear to be in Second Normal Form (2NF). They meet the conditions of 1NF and have no partial dependencies. Therefore, each non-key attribute is fully functionally dependent on the entire primary key, confirming 2NF compliance.

Analysis for 3NF for Each Relation

appearances:

Primary Key: appearance_id

All non-key attributes (like game_id, player_id, player_club_id, player_name, etc.) depend on the primary key appearance_id. There is no evidence of transitive dependencies among the attributes.

3NF Status: This relation is in 3NF.

club_games:

Primary Key: game_id, club_id (composite key)

All attributes depend on the entire composite key. There are no transitive dependencies among attributes in this relation.

3NF Status: This relation is in 3NF.

clubs:

Primary Key: club_id

All attributes (such as club_code, name, squad_size, stadium_name, etc.) depend on club_id. There are no indications of transitive dependencies.

3NF Status: This relation is in 3NF.

competitions:

Primary Key: competition_id

All attributes depend on competition_id. There are no observed transitive dependencies.

3NF Status: This relation is in 3NF.

game_events:

Primary Key: game_id, minute, type (composite key)

There are no transitive dependencies among non-key attributes.

3NF Status: This relation is in 3NF.

games:

Primary Key: game_id

Attributes such as competition_id, season, home_club_id, etc., are all dependent on game_id. There are no transitive dependencies among non-key attributes.

3NF Status: This relation is in 3NF.

player_valuations:

Primary Key: player_id, datetime (composite key)

Attributes are dependent on the entire composite key, with no transitive dependencies among non-key attributes.

3NF Status: This relation is in 3NF.

players:

Primary Key: player_id

All attributes such as first_name, last_name, position, date_of_birth, etc., are dependent on player_id. No transitive dependencies are observed.

3NF Status: This relation is in 3NF.

Conclusion

Based on the above analysis, the entire database appears to be in Third Normal Form (3NF). There are no observed transitive dependencies among the non-key attributes. All non-key attributes are dependent solely on their respective primary keys, either simple or composite, without any indirect dependencies through other non-key attributes.

Boyce-Codd Normal Form (BCNF)

BCNF is an extension of 3NF with a stricter condition:

A relation is in BCNF if, for every functional dependency

$X \rightarrow Y$, either:X X is a superkey (contains a candidate key), orThe functional dependency is trivial ($Y \subseteq X$).

In essence, BCNF eliminates any dependency where a non-superkey attribute determines other attributes.

Analysis for BCNF

Given the relations and primary keys, we look for cases where non-superkeys determine attributes or sets of attributes.

appearances:

The primary key is appearance_id.

There are no dependencies where a non-superkey determines attributes.

BCNF Status: This relation is in BCNF.

club_games:

The primary key is a composite of game_id and club_id.

No non-superkey determines attributes.

BCNF Status: This relation is in BCNF.

clubs:

The primary key is club_id.

No other attribute or set of attributes determines other attributes.

BCNF Status: This relation is in BCNF.

competitions:

The primary key is competition_id.

All functional dependencies involve the primary key.

BCNF Status: This relation is in BCNF.

game_events:

The primary key is a composite of game_id, minute, and type.

There are no dependencies where a non-superkey determines other attributes.

BCNF Status: This relation is in BCNF.

games:

The primary key is game_id.

All functional dependencies involve the primary key.

BCNF Status: This relation is in BCNF.

player_valuations:

The primary key is a composite of player_id and datetime.

All dependencies are based on the composite primary key.

BCNF Status: This relation is in BCNF.

players:

The primary key is player_id.

No other attribute or set of attributes determines other attributes.

BCNF Status: This relation is in BCNF.

SQL QUERIES

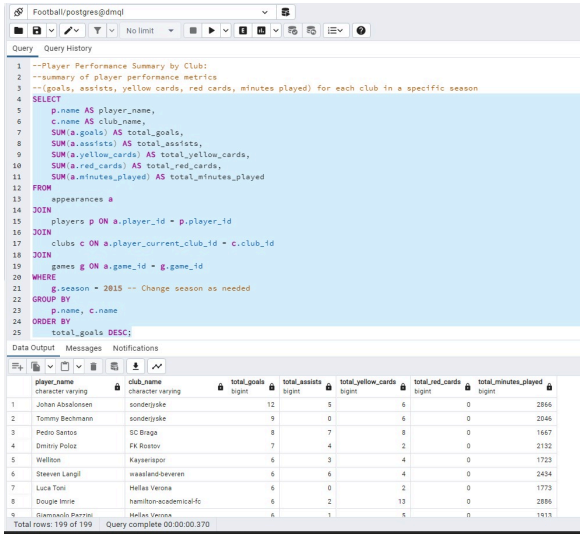
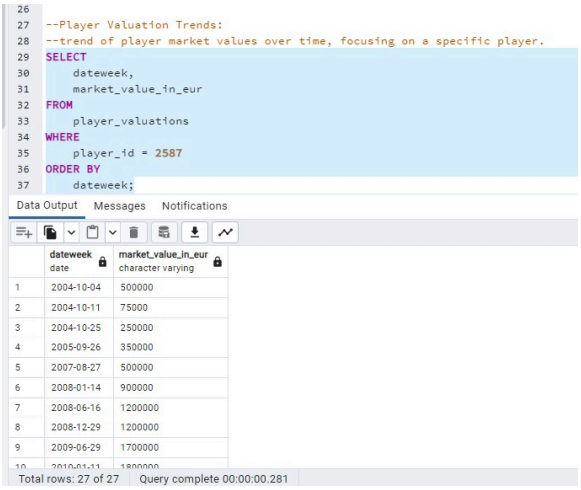


Fig 33. Query 1 - Player Performance summary by Club



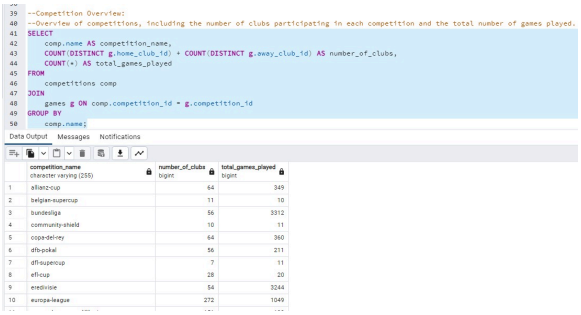
```sql
--Player Valuation Trends:
--trend of player market values over time, focusing on a specific player.
SELECT
    dateweek,
    market_value_in_eur
FROM
    player_valuations
WHERE
    player_id = 2587
ORDER BY
    dateweek;
```

Data Output | Messages | Notifications

	dateweek date	market_value_in_eur character varying
1	2004-10-04	500000
2	2004-10-11	75000
3	2004-10-25	250000
4	2005-09-26	350000
5	2007-08-27	500000
6	2008-01-14	900000
7	2008-06-16	1200000
8	2008-12-29	1200000
9	2009-06-29	1700000
10	2010-01-11	1800000

Total rows: 27 of 27 Query complete 00:00:00.281

Fig 34. Query 2 - Trent of player market values over time.

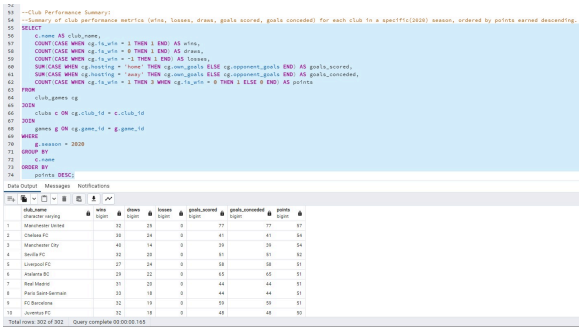


```sql
--Competition Overview:
--Overview of competitions, including the number of clubs participating in each competition and the total number of games played.
SELECT
    comp.name AS competition_name,
    COUNT(DISTINCT g.home_club_id) + COUNT(DISTINCT g.away_club_id) AS number_of_clubs,
    COUNT(*) AS total_games_played
FROM
    competitions comp
JOIN
    games g ON comp.competition_id = g.competition_id
GROUP BY
    comp.name;
```

Data Output | Messages | Notifications

	competition_name character varying (255)	number_of_clubs bigint	total_games_played bigint
1	allianz-cup	64	349
2	belgian-supercup	11	10
3	bundesliga	56	5312
4	community-shield	10	11
5	copa-del-rey	64	360
6	dfb-pokal	56	211
7	dfl-supercup	7	11
8	efl-cup	28	20
9	eredivisie	54	3244
10	europa-league	272	1049

Fig 35. Query 3 - Overview of competitions.



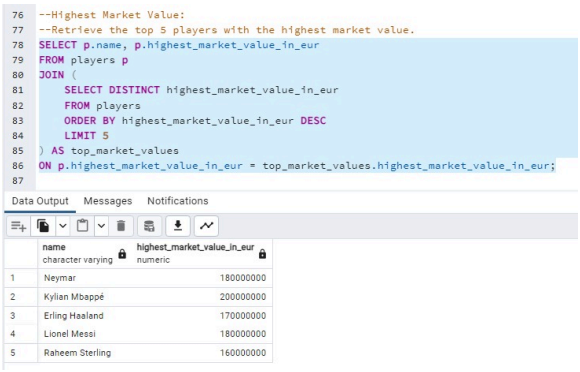
```sql
--Club Performance Summary:
--Summary of club performance metrics (wins, losses, draws, goals scored, goals conceded) for each club in a specific(2020) season, ordered by points earned descending.
SELECT
    c.name AS club_name,
    COUNT(CASE WHEN cg.is_win = 1 THEN 1 END) AS wins,
    COUNT(CASE WHEN cg.is_win = 0 THEN 1 END) AS draws,
    COUNT(CASE WHEN cg.is_win = -1 THEN 1 END) AS losses,
    SUM(CASE WHEN cg.hosting = 'home' THEN cg.own_goals ELSE cg.opponent_goals END) AS goals_scored,
    SUM(CASE WHEN cg.hosting = 'away' THEN cg.own_goals ELSE cg.opponent_goals END) AS goals_conceded,
    COUNT(CASE WHEN cg.is_win = 1 THEN 3 WHEN cg.is_win = 0 THEN 1 ELSE 0 END) AS points
FROM
    club_games cg
JOIN
    clubs c ON cg.club_id = c.club_id
JOIN
    games g ON cg.game_id = g.game_id
WHERE
    g.season = 2020
GROUP BY
    c.name
ORDER BY
    points DESC;
```

Data Output | Messages | Notifications

club_name character varying	wins bigint	draws bigint	losses bigint	goals_scored bigint	goals_conceded bigint	points bigint
1 Manchester United	32	25	0	77	77	97
2 Chelsea FC	28	24	0	41	41	84
3 Manchester City	48	14	0	39	39	54
4 Sevilla FC	32	20	0	61	61	92
5 Liverpool FC	27	24	0	58	58	81
6 Atalanta BC	29	22	0	65	65	91
7 Real Madrid	31	20	0	44	44	51
8 Paris Saint-Germain	33	18	0	44	44	51
9 FC Barcelona	32	19	0	59	59	51
10 Juventus FC	32	18	0	48	48	50

Total rows: 302 of 302 Query complete 00:00:00.765

Fig 36. Query 4 - Summary of club performance metrics

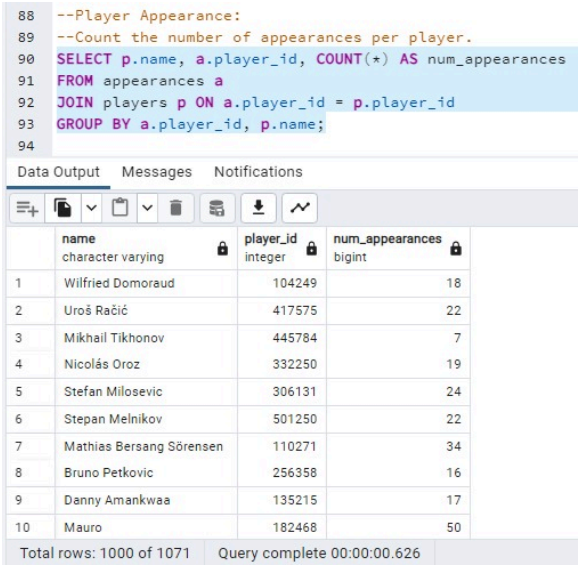


```sql
--Highest Market Value:
--Retrieve the top 5 players with the highest market value.
SELECT p.name, p.highest_market_value_in_eur
FROM players p
JOIN (
    SELECT DISTINCT highest_market_value_in_eur
    FROM players
    ORDER BY highest_market_value_in_eur DESC
    LIMIT 5
) AS top_market_values
ON p.highest_market_value_in_eur = top_market_values.highest_market_value_in_eur;
```

Data Output | Messages | Notifications

	name character varying	highest_market_value_in_eur numeric
1	Neymar	180000000
2	Kylian Mbappé	200000000
3	Erling Haaland	170000000
4	Lionel Messi	180000000
5	Raheem Sterling	160000000

Fig 37. Query 5 - Top 5 Players with the highest market value



```sql
--Player Appearance:
--Count the number of appearances per player.
SELECT p.name, a.player_id, COUNT(*) AS num_appearances
FROM appearances a
JOIN players p ON a.player_id = p.player_id
GROUP BY a.player_id, p.name;
```

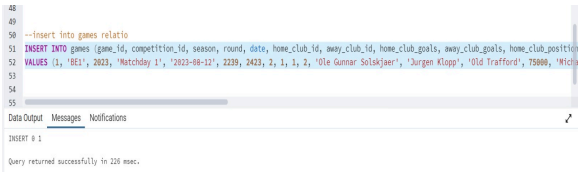
Data Output | Messages | Notifications

	name character varying	player_id integer	num_appearances bigint
1	Wilfried Domoraud	104249	18
2	Uroš Račić	417575	22
3	Mikhail Tikhonov	445784	7
4	Nicolás Oroz	332250	19
5	Stefan Milosevic	306131	24
6	Stepan Melnikov	501250	22
7	Mathias Bersang Sörensen	110271	34
8	Bruno Petkovic	256358	16
9	Danny Amankwaa	135215	17
10	Mauro	182468	50

Total rows: 1000 of 1071 Query complete 00:00:00.626

Fig 38. Query 6 - Number of appearance per player

INSERTION OF DATA:



```sql
--insert into games relatio
INSERT INTO games (game_id, competition_id, season, round, date, home_club_id, away_club_id, home_club_goals, away_club_goals, home_club_positio
VALUES (1, 'BE1', 2023, 'Matchday 1', '2023-08-12', 2239, 2423, 2, 1, 1, 2, 'Ole Gunnar Solskjaer', 'Jurgen Klopp', 'Old Trafford', 75000, 'Mich
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 226 msec.
```

Fig 1. Insert into GAMES table

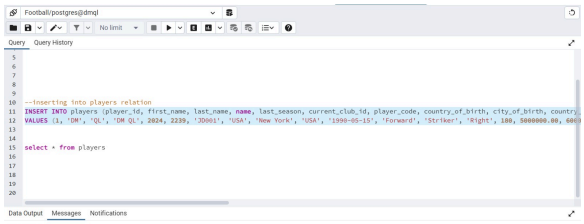


Fig 2. Insert into PLAYERS Table

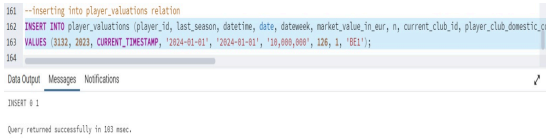


Fig 3. Insert into PLAYER_VALUATIONS Table

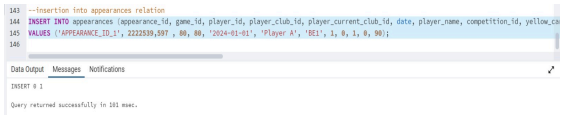


Fig 4. Insert into APPEARANCES

Fig 5. Insert Into CLUB_GAMES

Fig 6. Insert into COMPETITIONS

Fig 7. Insert into GAME_EVENTS

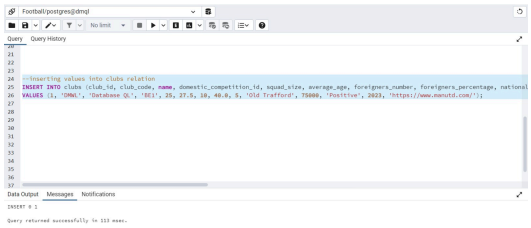


Fig 8. Insert into CLUBS

RETRIEVING OF DATA:

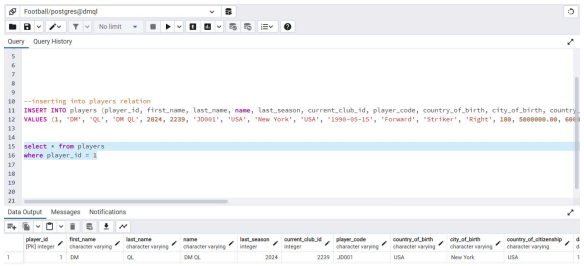


Fig 9. Retrieving inserted rows from PLAYERS table

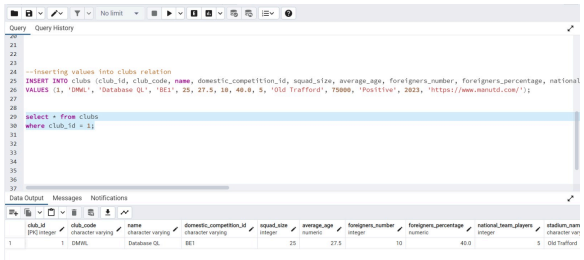


Fig 10. Retrieving rows from CLUBS table.

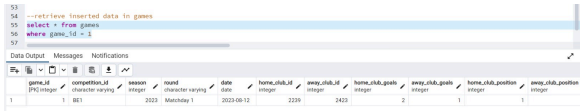


Fig 11. Retrieving rows from GAMES

Fig 12. Retrieving from GAME_EVENTS

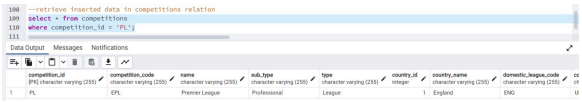


Fig 13. Retrieving from COMPETITIONS

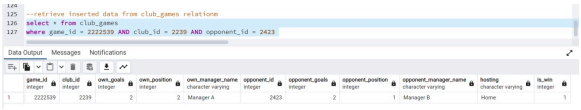


Fig 14. Retrieving from CLUB_GAMES

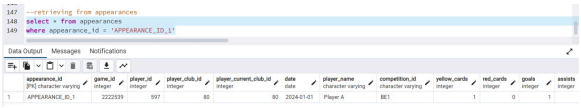


Fig 15. Retrieving from APPEARANCES

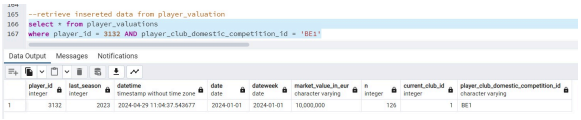


Fig 16. Retrieving from PLAYER_VALUATIONS

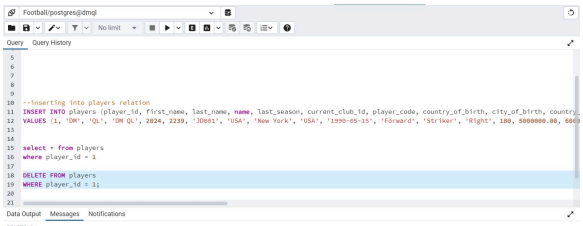


Fig 17. Deleting from PLAYERS

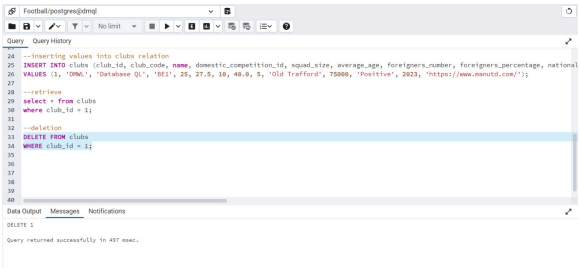


Fig 18. Deleting from CLUBS

```
--deletion from games relation
DELETE FROM games
WHERE game_id = 1;
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 641 msec.
```

Fig 19. Deleting from GAMES

```
--deletion from games relation
BEGIN;
-- Execute the DELETE statement
DELETE FROM game_events
WHERE game_id = 2222539;

-- Rollback the transaction to undo the DELETE operation
ROLLBACK;
```

Data Output | Messages | Notifications

```
DELETE 9

Query returned successfully in 130 msec.
```

Fig 20. Deleting from GAME_EVENTS

```
--delete competitions relation record
DELETE FROM competitions
WHERE competition_id = 'DMQL';
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 358 msec.
```

Fig 21. Deleting from COMPETITIONS

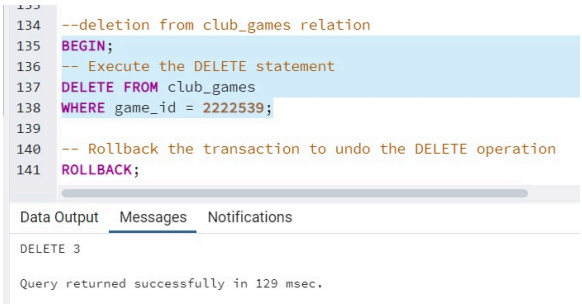


Fig 22. Deleting from CLUB_GAMES

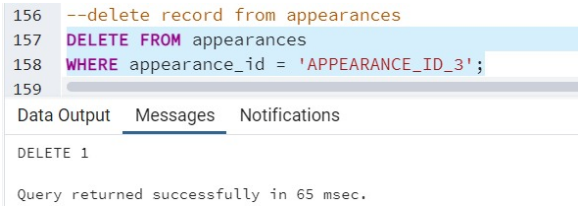


Fig 23. Deleting from APPEARANCES

Fig 24. Deleting from PLAYER_VALUATIONS

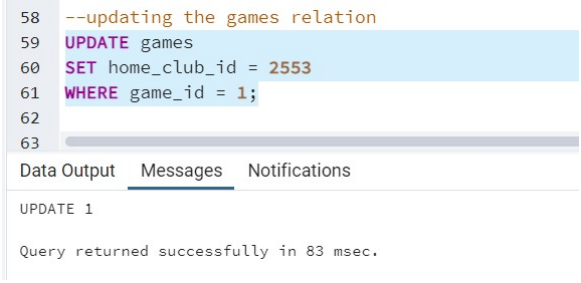


Fig 25. Updating GAMES Table

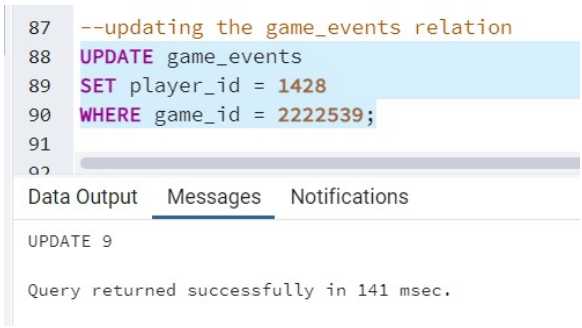


Fig 26. Updating GAME_EVENTS Table

Fig 27. Updating COMPETITIONS

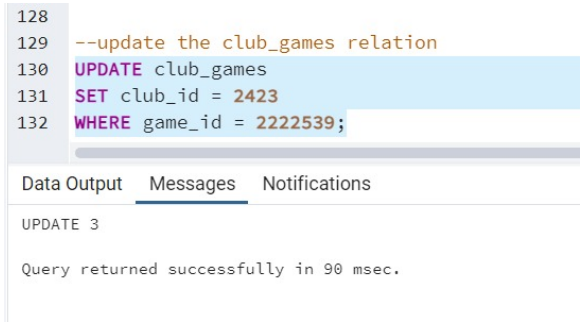


Fig 28. Updating CLUB_GAMES

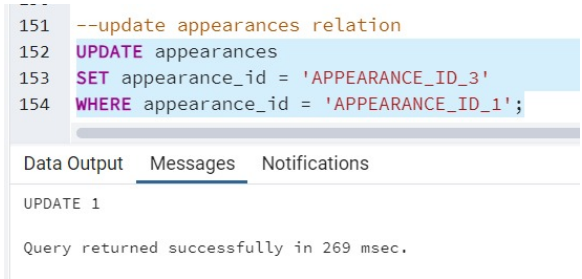


Fig 29. Updating APPEARANCES

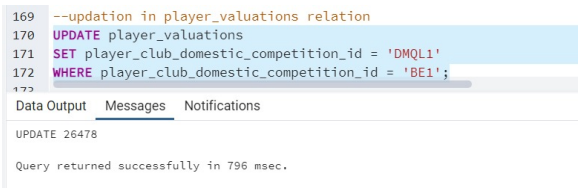


Fig 30. Updating PLAYER_VALUATIONS

Fig 31. Updating PLAYERS

Fig 32. Updating CLUBS

INDEXING

SQL indexing is a technique that helps speed up data retrieval, thereby improving database query performance. Query execution time can be greatly reduced by building indexes on frequently used columns, particularly those used in join conditions and filtering.

SELECT p.name, p.highest_market_value_in_eur

FROM players p

JOIN

SELECT DISTINCT highest_market_value_in_eur

FROM players

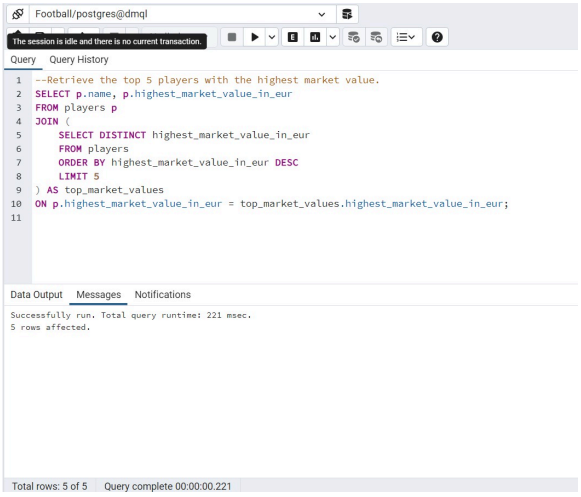
ORDER BY highest_market_value_in_eur DESC

LIMIT S

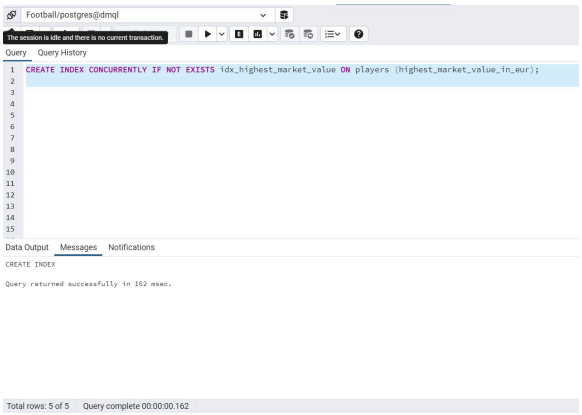
As top_market values

ON p-highest_market_value_in_eur = top_market_values.highest_market_value_in_eur;

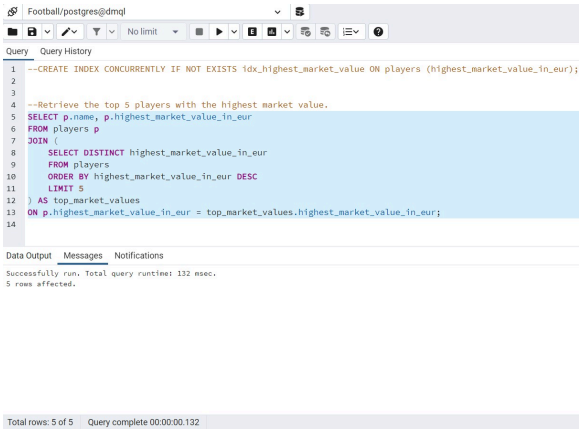
By joining the `players` table with a derived table that chooses and arranges the top {S} market values, the provided query finds players whose highest market values are among the top {S} distinct values. Indexing on the {highest_market_value_in_eur} column in the `players} table is essential to optimizing this query. The database can swiftly locate and sort market values thanks to this index, which makes it easier to choose the best {S} and speeds up the join process. By reducing the necessity for full-table scans, an index on {highest_market_value_in_eur} speeds up the database's ability to find relevant rows. Performance is enhanced overall as a result of quicker query execution and less processing overhead. If necessary, a second index on {name} can expedite the player names retrieval in the final output.



Before execution of Indexing(221 ms)



Creating Index

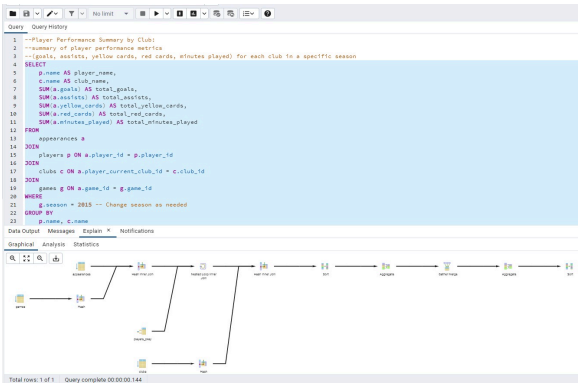


After adapting of Indexing concept exec (132 msec)

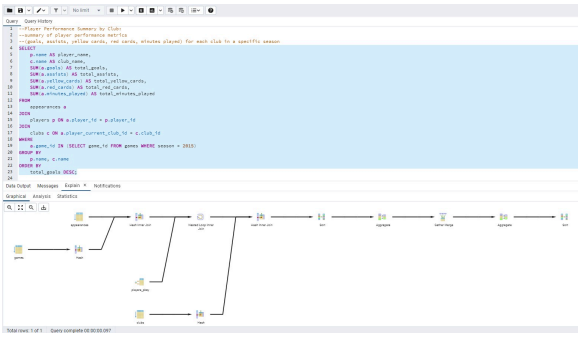
QUERY EXECUTION ANALYSIS

Query - 1

The way both SQL queries filter data from the games table by season is the primary distinction between them. After joining the games table, the first query uses a WHERE condition to retrieve games from a certain season, like 2015. Because it joins the full dataset before filtering, this method may be less efficient because it may require more processing, particularly if the games table is large or has complex relationships. On the other hand, the second query filters the main dataset by using the game_ids for the 2015 season, which are obtained via a subquery and then used in an IN clause. Since the dataset is narrowed before merging, this strategy typically reduces data amount and complexity early in the process, making it more efficient overall.



Player performance Before Optimization



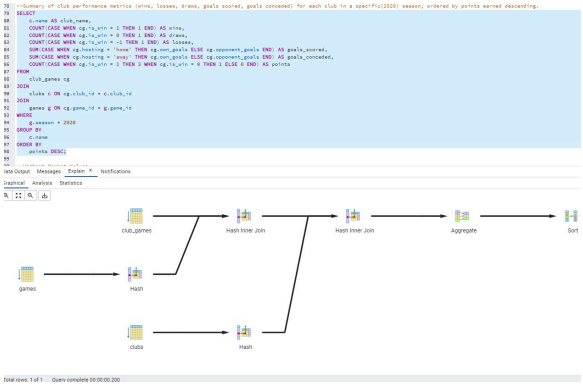
Player performance After Optimization

Consequently, the second method is more suited for

large or complicated datasets since it can result in quicker query execution and improved readability.

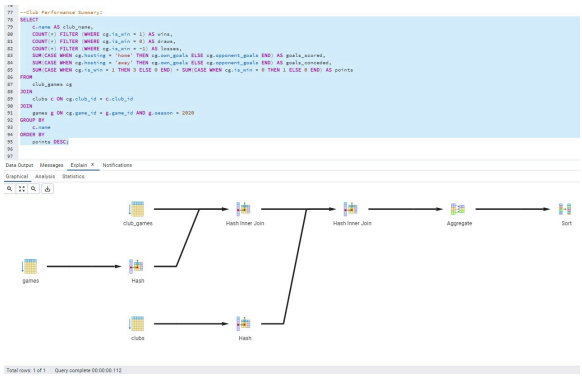
Query 2

The usage of `CASE` statements and `FILTER` clauses affects usability and efficiency in these two SQL queries that compute statistics for football clubs. Based on the value of `cg.is_win}, the first query calculates wins, draws, and losses using several `CASE` lines inside `COUNT`. To compute goals scored and allowed as well as to determine points from win/draw outcomes, it also combines {CASE} and `SUM}. However, this method needs to evaluate several conditional statements, which increases the computing expense.



In contrast, the second query makes advantage of {FILTER` clauses in the `COUNT` function to optimize this procedure. By filtering rows according to predetermined criteria, this technique streamlines the logic and eliminates the need for intricate {CASE` declarations. Because it simplifies calculation by allowing for the direct counting of rows that satisfy specific criteria, the filtering technique is typically more efficient. Furthermore, the second query's points computation improves clarity and maybe reduces ambiguity by separating win-based and draw-based points into two `SUM` statements. The season condition is also included in the `JOIN` clause of the second query, which may limit the dataset early in the query execution process and improve performance.

Club performance metrics Before Optimization

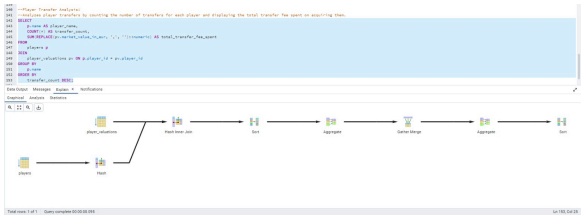


Overall, the second query is more optimized, easier to comprehend, and may execute faster than the first due to the usage of `FILTER` clauses and its simpler form.

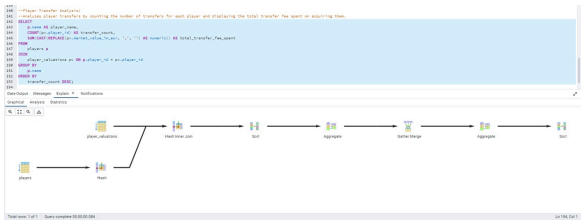
Query 3

The method applied for casting and counting determines how the two SQL queries are optimized, and this is the main distinction between them. The first query, `COUNT(pv.player_id)}, counts distinct occurrences of player IDs, necessitating a uniqueness check. Moreover, `market_value_in_eur} introduces extra processing overhead due to extra function calls during the string-to-number conversion. In comparison, the second query employs `COUNT(*)`, which is typically more efficient as it counts all rows without verifying the uniqueness of individual fields, thus offering a simple count of joined rows. The second query's casting makes use of PostgreSQL's native syntax, `::numeric}, which probably reduces overhead because of its straightforward methodology. Due to its easier counting, the second query is better streamlined overall.

Club performance metrics After Optimization



Player transfer Analysis Before Optimization

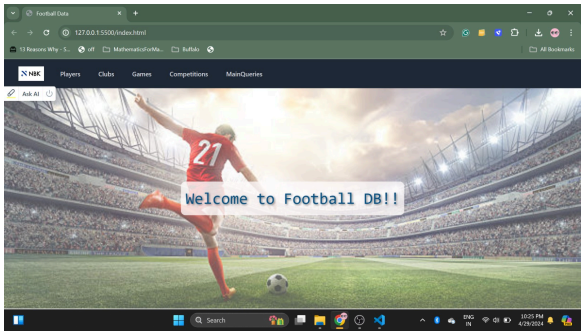


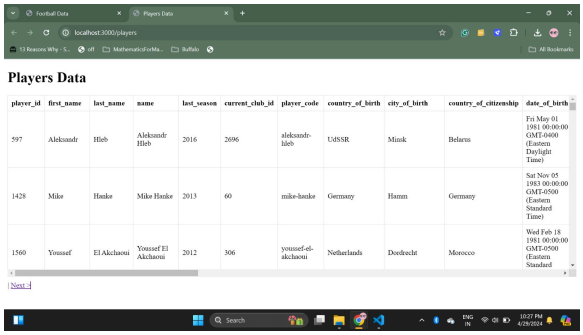
Player transfer Analysis After Optimization

Because of its more straightforward counting strategy and effective casting, the second query is generally more optimized than the first, resulting in faster execution and greater performance.

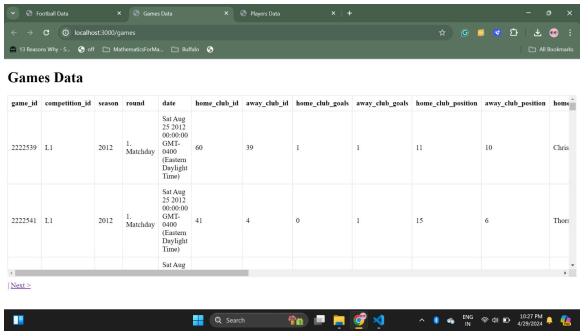
Web Application:

Welcome Page:

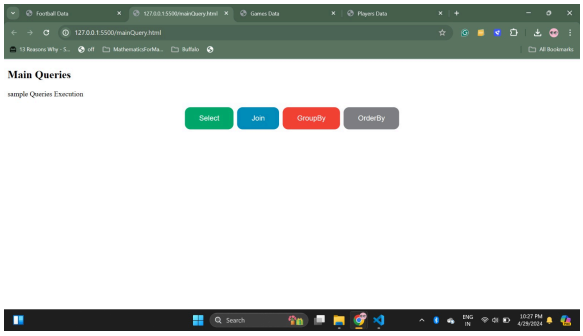




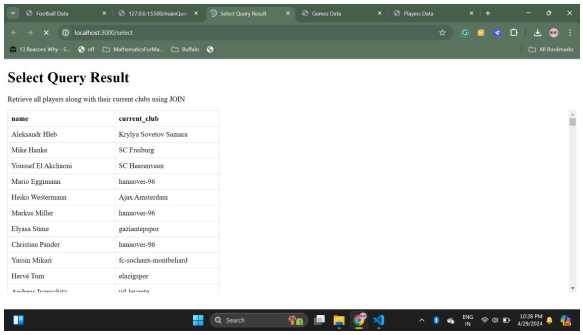
All Players Relation Data



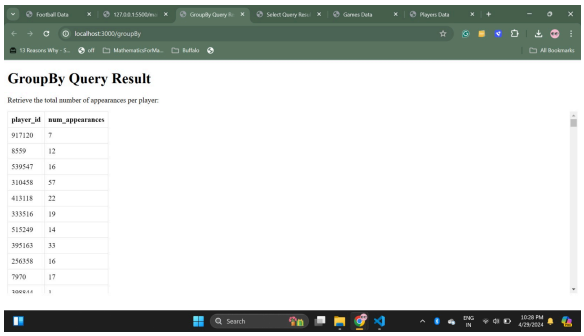
All Games relation data



Multiple Queries Page



Select Query from multiple query page



GroupBy Query

References:

[1]: Football Data from Transfermarkt (kaggle.com)

[2]: HTML Tutorial (w3schools.com)

[3]: https://www.javatpoint.com/dbms-normalization