**Object Relational and No-SQL DB**

**Individual Assignment 2**

**Marks 100 (40% Weight)**

**NOSQL DB - MongoDB**

**Problem Statement**

Design, populate, and query a MongoDB database for an event, focusing on real-time data processing, complex aggregation pipelines, and custom indexing strategies.

**Requirements:**

Set up a MongoDB database to track any popular live event such as cricket match or soccer game, NBA game or any other event that is tracked on multiple subscribers on various devices.

**Part 1: Database Schema Design**

**1.Data Description:**

1. **2020-2021.csv - Match Data Attributes:**
   * Date: The date on which the match was played. *Type: string or datetime.*
   * Time: The kickoff time of the match. *Type: string.*
   * HomeTeam: The name of the home team. *Type: string.*
   * AwayTeam: The name of the away team. *Type: string.*
   * FTHG (Full Time Home Goals): The number of goals scored by the home team by the end of the match. *Type: integer.*
   * FTAG (Full Time Away Goals): The number of goals scored by the away team by the end of the match. *Type: integer.*
   * FTR (Full Time Result): The result of the match, typically 'H' for home win, 'D' for draw, and 'A' for away win. *Type: string.*
   * Referee: The name of the referee overseeing the match. *Type: string.*
2. **player\_stats\_2020-2021.csv - Player Data Attributes:**
   * Name: The name of the player. *Type: string.*
   * Team: The team to which the player belongs. *Type: string.*
   * Nationality: The nationality of the player. *Type: string.*
   * Position: The playing position of the player on the field. *Type: string.*
   * Age: The age of the player. *Type: integer.*
   * Matches: The number of matches played by the player. *Type: integer.*
   * Goals: The total number of goals scored by the player during the season. *Type: integer.*
   * Assists: The number of assists made by the player. *Type: integer.*
   * Yellow Cards, Red Cards: The number of yellow and red cards the player received. *Type: integer.*

**2.Import data premier league 2020-2021 matches and player stats**

Pandas was imported for data manipulation and pymongo for MongoDB interactions. It loads match and player data into Pandas DataFrames from CSV files, displaying the first few rows to verify the data. Then I connects to a MongoDB database, setting the stage for further data insertion and analysis within a non-relational database framework.

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

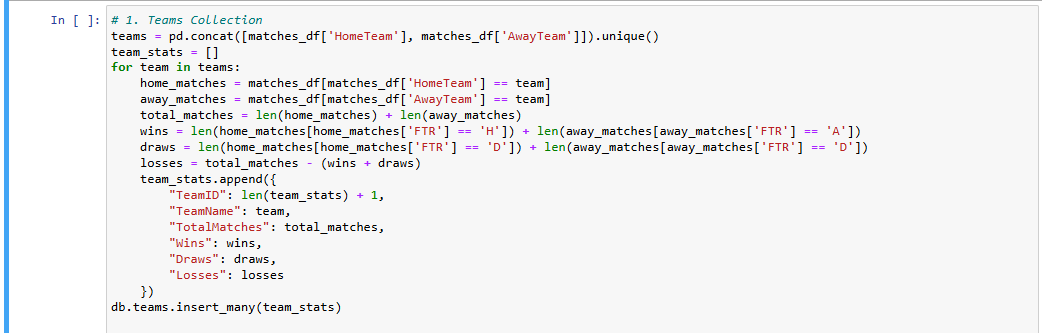
Description automatically generated**

**3.Schema Planning:**

**a.Design a schema that can accommodate the at least 5 collections and their attributes:**

**Identify and create at least 5 collections and their attributes.**

**1.Teams Collection:**



* In my Python script, I start by extracting a list of unique teams from both the 'HomeTeam' and 'AwayTeam' columns of the matches\_df DataFrame. This ensures I cover all teams involved in the matches without any duplicates.
* Next, I calculate the statistics for each team by filtering matches\_df to find all the matches where a team played as either home or away. For each team, I determine the number of total matches, wins, draws, and losses based on the 'FTR' (Full Time Result) column—counting home wins when the team won as 'HomeTeam', away wins when won as 'AwayTeam', and draws directly from matches that ended in a draw.
* I compile these statistics into a dictionary for each team, including identifiers and calculated fields such as total matches, wins, draws, and losses.
* Finally, I use the insert\_many method to add these team statistics to the 'teams' collection in my MongoDB database.
* **Description**: Contains information about each team in the Premier League.
* **Attributes**:
  + TeamID (unique identifier)
  + TeamName (e.g., Chelsea, Arsenal)
  + TotalMatches (derived: total matches played by the team)
  + Wins, Draws, Losses (derived: total outcomes per team)

**2.Players Collection**

**A screenshot of a computer code

Description automatically generated**

* Team Mapping: First, I query the MongoDB 'teams' collection to create a mapping from team names to their respective MongoDB IDs (TeamID). This is necessary to maintain referential integrity between the 'players' and 'teams' collections, ensuring each player is linked to a valid team.
* Player Data Preparation: Using the iterrows() method on the players\_df DataFrame, I iterate over each player entry. For each player, I construct a dictionary that includes:
  + PlayerID: An identifier I generate by adding 1 to the current index to ensure uniqueness.
  + Name, Nationality, Position, Age, MatchesPlayed, and Goals: These attributes are directly pulled from the DataFrame columns. Goals is particularly important as it quantifies each player's scoring contributions.
  + TeamID: I use the team name to TeamID mapping created earlier to reference the correct team ID. If the team name doesn't exist in the map, I handle it by assigning None.
* Data Insertion into MongoDB: Once the player data is structured in the required format, I insert it into the 'players' collection in the MongoDB database using insert\_many().
* **Description**: Stores detailed information about players.
* **Attributes**:
  + PlayerID (unique identifier)
  + Name (e.g., Mason Mount)
  + TeamID (references Teams collection)
  + Nationality
  + Position (e.g., GK, DF, MF, FW)
  + Age
  + MatchesPlayed

**3.Matches Collection**

A computer code with many colored text

Description automatically generated with medium confidence

* Initialize the Matches List: I start by creating an empty list named matches to hold the dictionary of match data for each game.
* Iterate Over Matches DataFrame: Using the iterrows() method, I loop through each row in the matches\_df DataFrame. For each row, I create a dictionary that contains the match details:
  + MatchID: Assigned by adding 1 to the current index, ensuring a unique identifier for each match.
  + Date and Time: Directly taken from the DataFrame, these fields specify when the match occurred.
  + HomeTeamID and AwayTeamID: I use a mapping (team\_id\_map) from team names to their MongoDB IDs to ensure the matches are linked to the correct teams in the database.
  + FullTimeScore: Constructed by concatenating the full-time home goals (FTHG) and away goals (FTAG) from the DataFrame.
  + Result: The outcome of the match (FTR) indicating whether the home team won, it was a draw, or the away team won.
  + Referee: The official who refereed the match.
  + HomeTeamName and AwayTeamName: These are added for easy reference without needing to join with the teams collection later.
* Insert Data into MongoDB: After preparing the list of match dictionaries, I insert them into the 'matches' collection in the MongoDB database using insert\_many(). This method efficiently inserts all the match entries at once.
* **Description**: Records details of each match.
* **Attributes**:
  + MatchID (unique identifier)
  + Date
  + Time
  + HomeTeamID (references Teams collection)
  + AwayTeamID (references Teams collection)
  + FullTimeScore (e.g., 3-1)
  + Result (H, A, D)
  + Referee

4.**Commentary Collection**

**A computer code with text

Description automatically generated with medium confidence**

* Initialize Commentary List: I start by creating an empty list named commentary which will store dictionaries. Each dictionary corresponds to an individual commentary record related to a specific match.
* Iterate Through DataFrame: Using the iterrows() method, I loop through each row of the matches\_df DataFrame. For each row:
  + CommentaryID: I assign a unique identifier for each commentary by adding 1 to the current index, ensuring each commentary entry is uniquely identified.
  + MatchID: This is also set as the row index + 1, assuming each row in the DataFrame corresponds to a match and that match identifiers are sequential.
  + Comment: Extracted from the "Commentary Section" column of the DataFrame, this field contains the actual text of the commentary.
  + Minute: Extracted from the "Comment Minute" column, indicating the minute of the match when the commentary event occurred.
* Insert Data into MongoDB: After constructing the list of commentary dictionaries, I insert them into the 'commentary' collection of the MongoDB database using insert\_many().
* **Description**: Contains live commentary or notable events during matches.
* **Attributes**:
  + CommentaryID (unique identifier)
  + MatchID (references Matches collection)
  + PlayerID (references Players collection)
  + Comment (e.g., "Mason Mount scores a goal")
  + Minute (e.g., 53)

**5.Match Statistics Collection**

**A computer code with text

Description automatically generated with medium confidence**

* Initialize the List: I begin by initializing an empty list called match\_stats, which will hold each player's statistics for both home and away games as individual dictionary entries.
* Loop Through Matches DataFrame: I iterate through each row in the matches\_df DataFrame using the iterrows() method, processing statistics for both home and away matches separately:
  + Home Matches Statistics:
    - I filter the DataFrame for rows where the team is listed under 'HomeTeam'.
    - For each row in the home matches, I append a dictionary to match\_stats that includes:
      * StatID: A unique identifier calculated by incrementing the length of match\_stats.
      * MatchID: Pulled from the DataFrame's index incremented by 1, assuming matches are indexed sequentially.
      * PlayerID: Set as the index incremented by 1, intended to uniquely identify players if their IDs are aligned with DataFrame indexing.
      * Goals: Extracted from the 'FTHG' (Full Time Home Goals) field, indicating goals scored by the home team.
      * Assists: Set to 0, as assist data isn't available or calculated. This is a placeholder that can be modified if assist data becomes available.
  + Away Matches Statistics:
    - Similarly, I process away matches by filtering for rows where the team is listed under 'AwayTeam'.
    - Each row's stats are appended similarly to the home match processing but using 'FTAG' (Full Time Away Goals) for goals scored by the away team.
* **Description**: Captures statistics for individual players in each match.
* **Attributes**:
  + StatID (unique identifier)
  + MatchID (references Matches collection)
  + PlayerID (references Players collection)
  + Goals
  + Assists
  + YellowCards
  + RedCards

**6.events collect tion(live commentary stored with player name, action, … which is created based on data from live commentary in part 4 )**

****

**A screen shot of a computer

Description automatically generated**

 create\_event\_documents Function: Iterates over each comment in the live\_commentary collection, checks for predefined actions using regex, and creates an event document if a match is found. It assumes a function get\_player\_name\_from\_comment to extract the player's name from the comment.

 get\_player\_name\_from\_comment Function: A placeholder function to extract the player's name from a comment.

**b.Identify fields for each collection, including relationships between documents.**

**1. Teams Collection**

* **Fields**:
  + TeamID (Unique identifier for each team)
  + TeamName (Name of the team, e.g., Arsenal, Chelsea)
  + TotalMatches (Derived field: Total matches played by the team)
  + Wins (Derived field: Total wins for the team)
  + Draws (Derived field: Total draws for the team)
  + Losses (Derived field: Total losses for the team)
* **Relationships**:
  + TeamID is referenced in the Players collection and the Matches collection.

**2. Players Collection**

* **Fields**:
  + PlayerID (Unique identifier for each player)
  + Name (Player's name, e.g., Mason Mount)
  + TeamID (Foreign key: References the TeamID in the Teams collection)
  + Nationality (Player's nationality, e.g., English, French)
  + Position (Player's position, e.g., GK, DF, MF, FW)
  + Age (Player's age during the season)
  + MatchesPlayed (Total number of matches the player participated in)
* **Relationships**:
  + TeamID links the player to the corresponding team in the Teams collection.
  + PlayerID is referenced in the Match Statistics and Commentary collections.

**3. Matches Collection**

* **Fields**:
  + MatchID (Unique identifier for each match)
  + Date (Date of the match)
  + Time (Time of the match)
  + HomeTeamID (Foreign key: References the TeamID in the Teams collection for the home team)
  + AwayTeamID (Foreign key: References the TeamID in the Teams collection for the away team)
  + FullTimeScore (Final score of the match, e.g., 3-1)
  + Result (Match result: H for home win, A for away win, D for draw)
  + Referee (Name of the referee)
  + YellowCards (Total yellow cards issued in the match)
  + RedCards (Total red cards issued in the match)
* **Relationships**:
  + HomeTeamID and AwayTeamID reference the Teams collection.
  + MatchID is referenced in the Commentary and Match Statistics collections.

**4. Commentary Collection**

* **Fields**:
  + CommentaryID (Unique identifier for each commentary entry)
  + MatchID (Foreign key: References the MatchID in the Matches collection)
  + Comment (Text of the commentary, e.g., "Mason Mount scores a brilliant goal")
  + Minute (Minute of the match when the event occurred)
* **Relationships**:
  + MatchID links the commentary to the corresponding match in the Matches collection.

**5. Match Statistics Collection**

* **Fields**:
  + StatID (Unique identifier for each statistic entry)
  + MatchID (Foreign key: References the MatchID in the Matches collection)
  + PlayerID (Foreign key: References the PlayerID in the Players collection)
  + Goals (Number of goals scored by the player in the match)
  + Assists (Number of assists by the player in the match)
* **Relationships**:
  + MatchID links the statistic to the corresponding match in the Matches collection.

PlayerID links the statistic to the corresponding player in the Players collection

**b.2.Relationships Overview:**

**1. Teams Collection**

* **Primary Entity**: Represents football teams.
* **Relationships**:
  + **Players Collection**: One-to-many relationship where one team can have multiple players. This is implemented via the TeamID in the Players Collection that references the TeamID in the Teams Collection.
  + **Matches Collection**: One-to-many relationship where each team can participate in multiple matches either as a home team or an away team. This is facilitated by HomeTeamID and AwayTeamID in the Matches Collection that reference the TeamID in the Teams Collection.

**2. Players Collection**

* **Primary Entity**: Represents individual players.
* **Relationships**:
  + **Teams Collection**: Many-to-one relationship where many players belong to one team. The TeamID in the Players Collection is a reference to the TeamID in the Teams Collection.
  + **Match Statistics Collection**: One-to-many relationship where each player can have multiple entries in the match statistics. This is linked via the PlayerID in the Match Statistics Collection.

**3. Matches Collection**

* **Primary Entity**: Represents individual football matches.
* **Relationships**:
  + **Teams Collection**: Many-to-one relationship where each match is associated with two teams (home and away). The HomeTeamID and AwayTeamID in the Matches Collection link to the TeamID in the Teams Collection.
  + **Commentary Collection**: One-to-many relationship where each match can have multiple commentary entries. Linked via MatchID.
  + **Match Statistics Collection**: One-to-many relationship where each match can have multiple statistical entries. Also linked via MatchID.

**4. Commentary Collection**

* **Primary Entity**: Contains real-time or post-match commentary entries for matches.
* **Relationships**:
  + **Matches Collection**: Many-to-one relationship where multiple commentaries can be associated with a single match. This is linked through the MatchID in the Commentary Collection to the MatchID in the Matches Collection.

**5. Match Statistics Collection**

* **Primary Entity**: Contains statistics such as goals and assists for players in specific matches.
* **Relationships**:
  + **Matches Collection**: Many-to-one relationship where multiple statistical records can be linked to a single match. This is linked through the MatchID.
  + **Players Collection**: Many-to-one relationship where multiple statistical records can be associated with a single player, linked through the PlayerID.

**c.Decide which fields to embed versus reference across collections.**

**1. Teams Collection**

* **Fields**: TeamID, TeamName, TotalMatches, Wins, Draws, Losses
* **Decision**:
  + **Reference**: No fields are embedded from other collections. TeamID is primarily used as a reference in the Players and Matches collections to establish a relationship.

**2. Players Collection**

* **Fields**: PlayerID, Name, TeamID, Nationality, Position, Age, MatchesPlayed, Goals
* **Decision**:
  + **Reference**: TeamID is a reference to the Teams Collection. This maintains a clear link to the player’s team without duplicating team data across multiple player documents, which simplifies updates to team information.

**3. Matches Collection**

* **Fields**: MatchID, Date, Time, HomeTeamID, AwayTeamID, FullTimeScore, Result, Referee
* **Decision**:
  + **Reference**: HomeTeamID and AwayTeamID are references to the Teams Collection.
  + **Embed**: It can be beneficial to embed HomeTeamName and AwayTeamName despite them also being indirectly available via HomeTeamID and AwayTeamID. This embedding is useful for quickly retrieving match details without needing to join with the Teams Collection, which can enhance read performance in scenarios where match details are frequently accessed.

**4. Commentary Collection**

* **Fields**: CommentaryID, MatchID, Comment, Minute
* **Decision**:
  + **Reference**: Maintains a reference to the Matches Collection through MatchID. Embedding specific match details is unnecessary since the commentary typically does not require direct access to all match fields.

**5. Match Statistics Collection**

* **Fields**: StatID, MatchID, PlayerID, Goals, Assists
* **Decision**:
  + **Reference**: Uses references for both MatchID and PlayerID. This avoids duplication and ensures that statistics can be updated or corrected without the need to modify embedded data in multiple locations.

**d.Justify your design choices in terms of MongoDB document model principles (e.g., for performance and scalability).**

1. Teams Collection

Design Choice: No embedding; only reference via TeamID.

* Justification: Teams are a central entity that relate to various other collections like players and matches. By using references instead of embedding, we avoid data duplication and ensure that updates to team information (e.g., team name changes, league status) are centralized, which simplifies maintenance and reduces the risk of data inconsistencies.

2. Players Collection

Design Choice: Referencing TeamID for team association.

* Justification: Players inherently belong to teams, and their data is often accessed in the context of the team they play for. By referencing TeamID instead of embedding team data directly into each player document, we ensure that the database can scale efficiently without redundant data, while still providing fast access to player information in a team context. This approach also makes it easier to query players by team and update team assignments without modifying numerous player documents.

3. Matches Collection

Design Choice: Embedding team names (HomeTeamName, AwayTeamName) along with using team ID references (HomeTeamID, AwayTeamID).

* Justification: Including team names directly in match documents caters to frequent queries that fetch match summaries where displaying team names is essential. This reduces the need for additional joins to the Teams collection, which can improve read performance significantly—critical during high-traffic scenarios like live match tracking. The use of team IDs ensures integrity and provides a way to access detailed team data when necessary.

4. Commentary Collection

Design Choice: Referencing MatchID.

* Justification: Commentaries are tied directly to matches, often accessed as part of match timelines or summaries. By referencing MatchID, we maintain a clear linkage to the associated match without embedding potentially large match data into every commentary document, which keeps commentary documents lightweight and queries fast.

5. Match Statistics Collection

Design Choice: Using references for MatchID and PlayerID.

* Justification: Match statistics are dynamic and can be numerous. By referencing both match and player identifiers, these statistics remain directly linked to their respective matches and players without introducing redundant data. This approach supports efficient updates (e.g., correcting statistical errors post-match) and simplifies aggregating statistics for players and matches.

Scalability and Performance Optimization:

* References over embedding for critical links ensures that the database can scale out by distributing data across multiple collections without excessive data replication.
* Selective embedding for performance improves query performance for common access patterns, like fetching match summaries, which are latency-sensitive operations, particularly important during live events.
* Maintaining data integrity through references allows the database structure to adhere to the principles of normalization where practical, reducing anomalies and simplifying updates.

**Part 2: Data Ingestion**

1. Populate the Database:
   1. Write a script in Python, to populate the database with mock data for your collection by sourcing the data from the official or any other relevant sources.
   2. Example: teams, players, matches, Live commentary, etc.
2. Data Consistency Check:
   1. Write at least 5 queries in python to check for data consistency across collections.
   2. For example:
      1. Verify that each player has only one associated team.
      2. Ensure event commentary is aligned chronologically and associated with a valid event.
      3. You can generate more queries based on your own definition of the ensuring consistency.

**1.Function: check\_player\_team\_consistency()**

A screenshot of a computer program

Description automatically generated

* **Purpose**: Verifies that each player is linked to an existing team in the Teams collection. This ensures there are no players with invalid or missing team associations.
* **Operation**:
  + **$lookup stage**: Joins the players collection with the teams collection using the TeamID field. The result of this lookup is stored in a new array field called team\_info.
  + **$match stage**: Filters the documents where the team\_info array is empty ($size: 0). An empty team\_info array indicates that no corresponding team was found for the TeamID listed in the player's document.
* **Return**: Outputs a list of player documents where no corresponding team could be found, indicating data inconsistency.

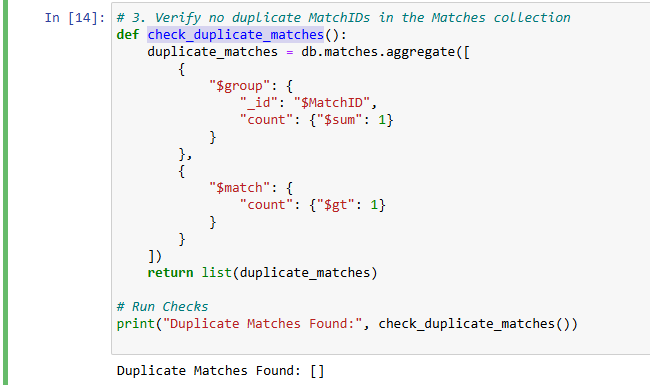
**2.Function: check\_commentary\_consistency()**

**A screenshot of a computer code

Description automatically generated**

* Purpose: Checks to make sure that all commentary entries in the commentary collection are aligned chronologically and linked to a valid match in the matches collection.
* Operation:
  + $lookup stage: This stage joins documents from the commentary collection with those from the matches collection, matching on the MatchID field. The resulting match information is stored in a new field called match\_info.
  + $match stage: This stage filters the resulting documents to identify commentary entries that do not have a corresponding match document (match\_info is empty, i.e., $size: 0).
* Return: Returns a list of commentary documents that are not associated with any valid match, indicating inconsistencies or orphaned entries.3. Function: check\_duplicate\_matches()

**3.Function: check\_duplicate\_matches ()**

****

* Purpose: Ensures that each match in the database has a unique identifier, which is critical for reliable data management and retrieval.
* Operation:
  + $group stage: Groups the documents in the matches collection by MatchID and counts the number of occurrences of each MatchID using $sum: 1.
  + $match stage: Filters these grouped results to identify any MatchIDs that appear more than once (count: {$gt: 1}), indicating duplicates.
* Return: Returns a list of documents, each representing a MatchID that has duplicates, along with the count of how many times each duplicate appears.

**4.Function: check\_player\_existence\_in\_statistics()**

**A screenshot of a computer program

Description automatically generated**

* Purpose: To verify that all players referenced in match statistics are registered in the players directory, preventing errors or inconsistencies in player-related queries and analyses.
* Operation:
  + $lookup stage: Joins documents from the players collection with those in the match\_statistics collection based on the PlayerID. This operation attempts to find and append player information to each match statistic document.
  + $match stage: Filters the results to find entries where the joined player\_info array is empty ($size: 0), indicating that no matching player document was found in the players collection.
* Return: Provides a list of match statistic entries that reference nonexistent players, aiding in identifying and rectifying data discrepancies.

**5.Function: check\_team\_matches\_count()**

****

* Purpose: Ensures that the TotalMatches value in the teams collection accurately reflects the number of matches played by each team as recorded in the matches collection.
* Operation:
  + $lookup stage: Performs a left outer join to the matches collection from the teams collection. It uses the $match stage inside a $pipeline to filter matches where either HomeTeamID or AwayTeamID matches the team's ID ($TeamID).
  + $count stage: Counts the total number of matches found for each team.
  + $match stage: Filters the results to identify any discrepancies between the count of matches derived from the matches collection and the TotalMatches field in the teams collection.
* Return: Outputs a list of teams where the recorded TotalMatches does not match the actual matches found, highlighting potential data inconsistencies**.**

**Part 3: Aggregation and Analytics**

1. Complex Aggregation Queries: Write MongoDB aggregation queries to provide match aggregation such as:

**a.Top Scorers: List the top 5 players with the highest scores in the event.**

**Function: find\_top\_scorers\_directly()**

A screenshot of a computer program

Description automatically generated

I double checked with real data the sult showed correctly

A screenshot of a black screen

Description automatically generated

* Purpose: To retrieve the top five goal scorers from the players collection, sorting them by the Goals field in descending order.
* Operation:
  + $sort stage: Orders the players by the number of goals scored, from highest to lowest.
  + $limit stage: Restricts the output to the top five documents, ensuring only the players with the most goals are returned.
  + $project stage: Specifies the fields to include in the output, omitting the MongoDB internal \_id field and including player name, total goals, team ID, and nationality. This helps in focusing the output on relevant information.
* Output: The script lists each top scorer's name and total goals, which provides clear and concise feedback on the query's results.
* Error Handling: If no documents match the criteria (possibly due to an empty collection or incorrect field names), it informs the user that no results were found.

b.Team Performance Over Time: Calculate each team's performance trend over the course of the tournament by aggregating match results chronologically.

**Function: analyze\_team\_performance()**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

I double checked again with real data the result showed correctA screenshot of a score

Description automatically generated

1. **Retrieve Team ID**: Searches for a specific team by name within the teams collection to obtain its unique Team ID. If the team isn't found, it issues a warning and stops further execution.
2. **Aggregation Pipeline**: Executes a series of MongoDB aggregation operations on the matches collection to process match data involving the specified team:
   * Filters matches where the team was either the home or away team.
   * Converts the match date from string to date format for accurate chronological sorting.
   * Calculates points for each match based on the result—3 points for a win, 1 for a draw, and 0 for a loss, adjusting points allocation based on whether the team was playing at home or away.
   * Sorts all relevant matches by date.
   * Groups the data to push each match’s date and points into an array and sums up the points to calculate cumulative points.

Output: Prints the cumulative points over the season and details for each match, such as the date, points from that match, and the running total of points

**c.Real-Time Match Summary: Create an aggregation pipeline to generate a live event/match summary with stats like current score, top scorer etc.**

**Function: generate\_match\_summary()**

A screenshot of a computer code

Description automatically generated

1. Aggregation Pipeline:
   * Match Filtering: It starts by filtering the documents in the matches collection for the given match\_id.
   * Data Enrichment: Uses $lookup to join data from the commentary and players collections to bring in detailed commentary and player information related to the match.
   * Data Transformation:
     + Unwinds the Commentary array to process individual comments.
     + Enhances the document with TopScorer details by filtering the PlayerDetails array to match player IDs from the commentary.
     + Groups the results by MatchID and aggregates the details such as the teams involved, the score, and counts of specific events like yellow and red cards.
   * Final Lookups: Further enriches the document by resolving team names from the teams collection for both home and away teams.
   * Projection: Defines the structure of the output document, specifying which fields to include in the final output.

Output: Executes the pipeline and prints out a detailed match summary, including teams, score, top scorer, count of yellow and red cards, and a list of all commentary entries for the match.

**Part 4: Real-Time Operations**

1. **Real-Time Commentary Updates:**
   1. **Set up a simulation to add real-time commentary updates (e.g., ball-by-ball commentary) to matches. Write a script to insert a new document into the Live Commentary collection every 10 seconds during a simulated match.**

** Function: generate\_commentary:**

A computer screen shot of a computer code

Description automatically generated A screenshot of a computer

Description automatically generated

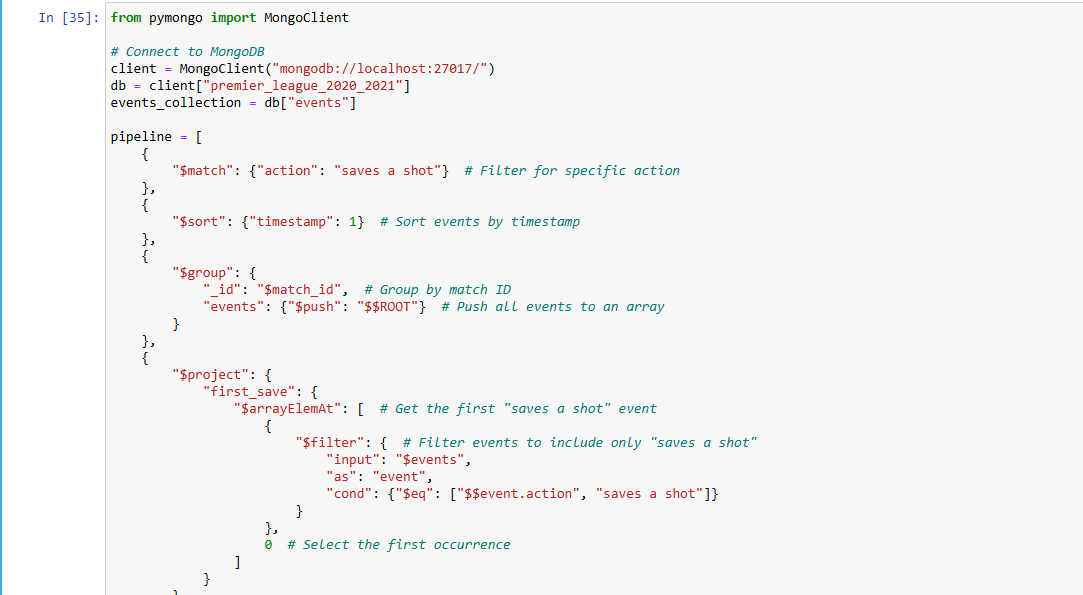
* Accepts a match document as input.
* Retrieves player data for both teams involved in the match using their TeamID.
* Parses the match's start time to generate accurate timestamps for each comment.
* Iteratively generates commentary for each minute of the match. Within each minute, commentary is generated every 10 seconds, simulating a real-time event.
* For each generated event, randomly selects whether the action is made by a player from the home or away team and randomly selects the action from the predefined list.
* Creates a commentary entry with a timestamp and the description of the action, which is then inserted into the live\_commentary MongoDB collection.
* Outputs the commentary to the console for monitoring purposes.

 Processing All Matches:

* Retrieves all matches from the matches collection and applies the generate\_commentary function to each, simulating a live commentary for every match in the database.

**Nested Array Query: $filter operator is used to isolate only those commentary entries where a player "saves a shot" from the array of live match comments.**

**Implement at least one nested array query to retrieve aggregate information (a tricky use of $filter and $arrayElemAt in MongoDB).**

****

**The output provides a concise summary for each match where a "saves a shot" action occurred. For each match, it prints:**

* **Match ID, teams involved, date, and time.**
* **Detailed information about the first "save a shot" action, including the exact time and the player who performed the save**

**Part 5: Deliverables**

1. Python Code: Python scripts that:

* Create collections.
* Populate data
* Run aggregation queries.
* Real time operations

1. Document your Work: Prepare a brief report (5 Pages) summarising:
   * Schema and design decisions and their rationale.
   * Key challenges and solutions encountered during the data ingestion and querying process.