NoSQL Project

Cellitti Francesco, Negroni Edoardo

6 November 2023

1 Introduction

It was decided to do the project on the topic **"Data analysis with a NoSQL system"**; in order to do that it was advised to use **MongoDB**, exploiting as **IDE Studio 3T**.

MongoDB is a NoSQL (non-relational) database management system widely used to store structured, semi-structured or unstructured data. It is known for its scalability, flexibility and ability to handle huge volumes of data efficiently. Here are some of the main features and functionalities of MongoDB:

1. **Scalable data storage**: MongoDB is designed to handle large amounts of data and can scale horizontally across clusters of servers to increase storage capacity and query speed.

2. **Data model flexibility**: it uses a document data model, which means that data is stored in JSON documents (BSON actually, a binary representation of JSON). This allows data to be stored with flexible and complex structures without the need for a rigid schema. Documents are grouped into collections. Collections are similar to tables in a relational database but more flexible. A MongoDB database contains one or more collections. It is the logical unit for organizing data.

3. **Query speed**: MongoDB offers high performance for queries due to its indexing capability and in-memory storage.

4. **High availability and fault tolerance**: MongoDB supports replication and geographic distribution of data to ensure high availability and fault tolerance. → Replication and auto-sharding

5. **Advanced aggregations and analysis**: it provides powerful data aggregation and analysis operations, enabling complex calculations to be performed on the data.

In general, MongoDB is suitable for scenarios where the data structure changes frequently, or where there is a need to store large volumes of data, perform complex queries, and easily scale horizontally to handle growing workloads. It is used in a wide range of applications, including web applications, data analytics, Internet of Things (IoT) and more.

1.1 Indexes in MongoDB

Indexes are a key feature of MongoDB since they are used to improve significantly the speed and efficiency of database queries. Without indexes, MongoDB must scan every document in a collection to return query results, which can be slow and resource-intensive. If an appropriate index exists for a query, MongoDB uses the index to limit the number of documents it must scan. While indexes undeniably enhance the efficiency of query execution, introducing an index can have adverse consequences on the performance of write operations. In collections where write operations significantly outnumber read operations, the use of indexes can be costly, as each insert operation necessitates the concurrent update of associated indexes.

In our analysis, we noticed that in some quite complex queries, which required the simultaneous use of very large collections, the use of indexes significantly sped up the time taken to execute it (often the query even exceeded the time limit without returning a result). Hence, this feature of MongoDB was critical in our operations.

2 Data

Thanks to the structure of Studio 3T, with ”import” from its menu, it’s possible to import the data easily in all the most common format.

This data are collected by David Cereijo [?] in different CSV files using Transfermarket information and it includes:

- 60,000+ games from many seasons on all major competitions
- 400+ clubs from those competitions
- 30,000+ players from those clubs
- 400,000+ player market valuations historical records
- 1,200,000+ player appearance records from all games

The data are collected in different database, linked in this way:

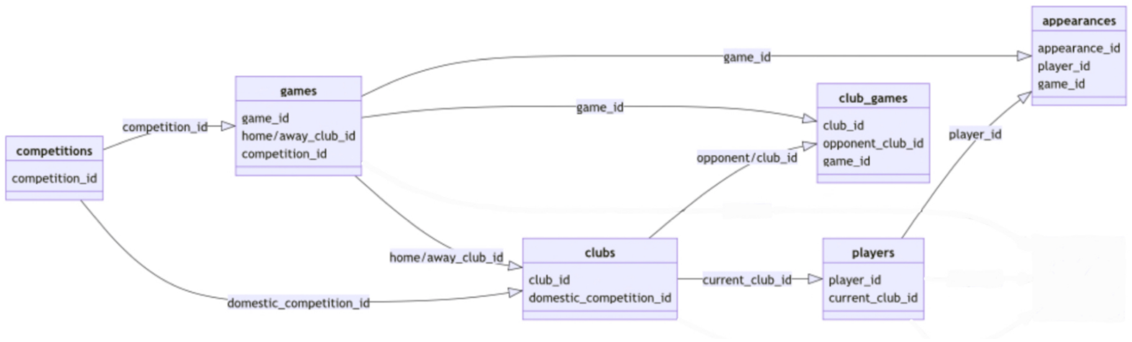


Figure 1: Structure of database

Describing very quickly the databases:

- **competitions**: containing information about all the European main competition (all the major tournament for each country);
- **games**: all the games played in that competition with characteristics of the game and result;
- **clubs**: all the clubs participating in these competitions since 2012;
- **club_games**: other characteristics for clubs;
- **players**: all the players that played for the team linked in the other database;
- **appearances**: how many minutes and what a player did in a single match

2.1 Cleaning dataset

In order to answer better to our project question we decide to drop all the players retired before 2022 (not of our interest) and all the games from 2012 to 2018 for studying the behavior of players only in the last four season (the person who collected the data stopped in April 2023). Eventually, applying the following query we are able to drop a lot of values (this is only an example of the queries we did since in this step they were all the same).

```
db.Appearences.deleteMany({
        'date': {$regex:'^2018'}
});
```

3 Research question and analysis

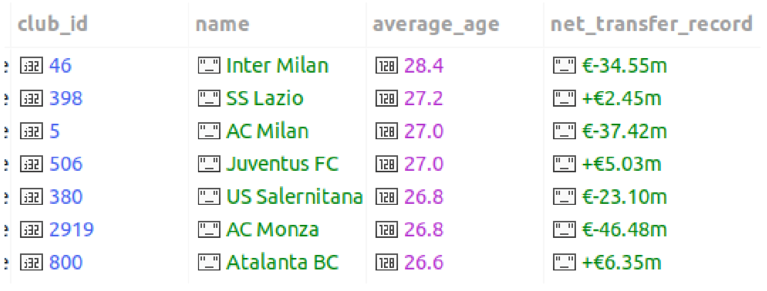
The potential of using MongoDB in the world of football allows a list of players to be analysed and grouped according to the occurrences of teams and certain characteristics analysed on a match history.

In this project, it was decided to put oneself in the shoes of a sports director of a team. The latter was instructed by the president of the club to rejuvenate the team while keeping the team level high by replacing older players with youngsters with similar characteristics. In order to do that we will have to perform several queries and retrieve informations from different collections to obtain the result we need.

To decide which team to choose in our analysis, we took the two highest average ages in the 2022 Serie A championship using the query:

```
db.Clubs.find(
    {'domestic_competition_id': 'IT1','last_season':2022},
    {'club_id':1,'name':1,
    'average_age':1,
    'net_transfer_record':1}).sort({'average_age':-1}).limit(10)
```

Select from the list of the ten clubs all the teams from Serie A (IT1) that played during the season 2022/2023, the information like id, name, average age, sorted by average age in descending order. The results show that the two teams selected for this analysis are Inter Milan and AC Milan (the first and third "oldest teams").



club_id	name	average_age	net_transfer_record
46	Inter Milan	28.4	€-34.55m
398	SS Lazio	27.2	+€2.45m
5	AC Milan	27.0	€-37.42m
506	Juventus FC	27.0	+€5.03m
380	US Salernitana	26.8	€-23.10m
2919	AC Monza	26.8	€-46.48m
800	Atalanta BC	26.6	+€6.35m

Figure 2: First Serie A teams ordered by average age

After that, we select the 8 oldest players for each team (excluding the goalkeepers) in order to find someone with same features to replace all of them.

```
db.Players.find({
  'current_club_id': 5,
  'last_season': { $eq: 2022 },
  'position': { $ne: 'Goalkeeper' }},
  {
  'name': 1,
  'date_of_birth': 1,
  'sub_position': 1,
  'foot': 1,
  'market_value_in_eur': 1,
  'contract_expiration_date':1,
  'height_in_cm':1,
   'player_id':1})
   .sort({'date_of_birth': 1})
   .limit(8);


var documents = old_Milan.toArray();

// Create a new collection and insert the documents
db.createCollection('Milan');
db.Milan.insertMany(documents);
```

AC Milan	**Inter Milan**
Ibrahimovic (42)	Dzeko (37)
Giroud (37)	Acerbi (35)
Kjaer (34)	D'Ambrosio (35)
Florenzi (32)	Darmian (34)
Messias (32)	Mkhitaryan (34)
Rebic (30)	De Vrij (31)
Krunic(30)	Brozovic (31)
Bakayoko (29)	Lukaku (30)

Table 1: The 8 oldest player of AC Milan and Inter Milan

3.1 Adding new statistics

For all the players (and also only for Milan and Inter) we complete the dataset adding the total sum of some feature creating new ones, during the years 2019 to 2022. In order:

- create the new collection called 'AggregatedPlayers';

  ```
  db.createCollection('AggregatedPlayers');
  ```

- $group groups documents based on the "player id" field and calculates the "total_yellow_cards", "total_red_cards" , "total_goals", "total_assists", "total_minutes_played" and calculates the sum of each field for all documents in each group.

  ```
  {
  $group: {
    _id: "$player_id",
    total_yellow_cards: { $sum: "$yellow_cards" },
    total_red_cards: { $sum: "$red_cards" },
    total_goals: { $sum: "$goals" },
    total_assists: { $sum: "$assists" },
    total_minutes_played: { $sum: "$minutes_played" },
    presence: { $sum: 1 }
      }
  }
  ```

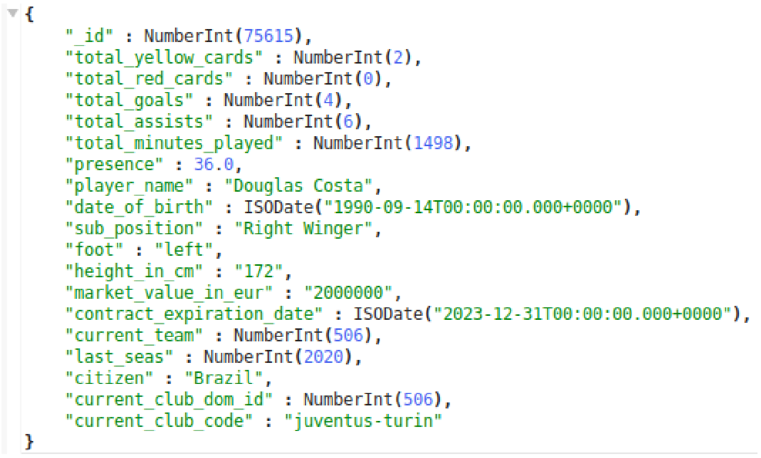
- $lookup is used to combine data from the "Appearences" and "Players" collections based on the "player_id" field. This operation essentially performs a join operation:

```
    {
    $lookup: {
      from: "Players",
      localField: "_id",
      foreignField: "player_id",
      as: "playerInfo"
    }
    },
    {$unwind: "$playerInfo"}
```

- $project reshapes the output data and renames some fields.

```
  {
  $project: {
    player_id: "$player_id",
    player_name: "$playerInfo.name",
    date_of_birth: "$playerInfo.date_of_birth",
    sub_position: "$playerInfo.sub_position",
    foot: "$playerInfo.foot",
    height_in_cm: "$playerInfo.height_in_cm",
    market_value_in_eur: "$playerInfo.market_value_in_eur",
    contract_expiration_date: "$playerInfo.contract_expiration_date",
    current_team: "$playerInfo.current_club_id",
    last_seas:"$playerInfo.last_season",
    citizen:"$playerInfo.country_of_citizenship",
    current_club_dom_id:"$current_club_domestic_id",
    total_yellow_cards: 1,
    total_red_cards: 1,
    total_goals: 1,
    total_assists: 1,
    total_minutes_played: 1,
    presence: 1
  }},
{$out:'AggregatedPlayers'}])
```

With the existing structure, we augment this dataset by incorporating additional information for each player, specifically identifying the team they are currently playing for from the collection 'Clubs'. This is an example for one player (one row) in the JSON format:



```
{
    "_id" : NumberInt(75615),
    "total_yellow_cards" : NumberInt(2),
    "total_red_cards" : NumberInt(0),
    "total_goals" : NumberInt(4),
    "total_assists" : NumberInt(6),
    "total_minutes_played" : NumberInt(1498),
    "presence" : 36.0,
    "player_name" : "Douglas Costa",
    "date_of_birth" : ISODate("1990-09-14T00:00:00.000+0000"),
    "sub_position" : "Right Winger",
    "foot" : "left",
    "height_in_cm" : "172",
    "market_value_in_eur" : "2000000",
    "contract_expiration_date" : ISODate("2023-12-31T00:00:00.000+0000"),
    "current_team" : NumberInt(506),
    "last_seas" : NumberInt(2020),
    "citizen" : "Brazil",
    "current_club_dom_id" : NumberInt(506),
    "current_club_code" : "juventus-turin"
}
```

Figure 3: JSON example of AggregatedPlayerClubs dataset

3.2 Merge datasets Appearences and Games conditionally on team_home_id/team_away_id with team_id

Index Creation: The initial part of the code is responsible for creating indexes on the "Appearences" and "Games" collections. Indexes are used to optimize query performance by allowing MongoDB to quickly locate relevant data without scanning the entire collection.

- db.Appearences.createIndex({ game_id: 1 }): this line creates an ascending index on the "game_id" field in the "Appearences" collection.
- db.Games.createIndex({ game_id: 1 }): Similarly, it creates an ascending index on the "game_id" field in the "Games" collection.

```
db.Appearences.createIndex({ game_id: 1 });
db.Games.createIndex({ game_id: 1 });
```

Data Aggregation and Transformation: The following section of the code is where data from the "Appearences" and "Games" collections is aggregated and transformed based on specific conditions (in order to obtain other useful features).

```
db.Appearences.aggregate([
```

- $lookup is used to combine data from the "Appearences" and "Games" collections based on the "game_id" field. This operation essentially performs a join operation;

```
{
$lookup: {
  from: "Games",
  localField: "game_id",
  foreignField: "game_id",
  as: "gameInfo"
    }
}
```

- $unwind is used to deconstruct arrays created by the $lookup operation.

```
    {$unwind: "$gameInfo"}
```

- $group groups documents based on the ”player_id” field and calculates the ”goals_scored” and ”goals_conceded” values based on specific conditions.

```
    $group: {
      _id: "$player_id",
      goals_scored: {
        $sum: {
          $cond: [
            {
              $eq: ["$player_club_id", "$gameInfo.home_club_id"]
            },
            "$gameInfo.home_club_goals",
            "$gameInfo.away_club_goals"
          ]
        }
      },
      goals_conceded: {
        $sum: {
          $cond: [
            {
              $eq: ["$player_club_id", "$gameInfo.home_club_id"]
            },
            "$gameInfo.away_club_goals",
            "$gameInfo.home_club_goals"
    ]}}}}
```

- Another $lookup operation combines the aggregated data with the ”AggregatedPlayersClub” collection.

```
{
    $lookup: {
      from: "AggregatedPlayersClub",
      localField: "_id",
      foreignField: "_id",
      as: "playerClubInfo"}}
```

- $project reshapes the output data and renames some fields.

```
    {$unwind: "$playerClubInfo"},
    {
     $project: {
      player_id: "$_id",
      total_yellow_cards: "$playerClubInfo.total_yellow_cards",
      total_red_cards: "$playerClubInfo.total_red_cards",
      total_goals: "$playerClubInfo.total_goals",
      total_assists: "$playerClubInfo.total_assists",
      total_minutes_played: "$playerClubInfo.total_minutes_played",
      presence: "$playerClubInfo.presence",
      goals_scored: 1,
      goals_conceded: 1,
      player_name: "$playerClubInfo.player_name",
      date_of_birth: "$playerClubInfo.date_of_birth",
      sub_position: "$playerClubInfo.sub_position",
      foot: "$playerClubInfo.foot",
      height_in_cm: "$playerClubInfo.height_in_cm",
      market_value_in_eur: "$playerClubInfo.market_value_in_eur",
      contract_expiration_date: "$playerClubInfo.contract_expiration_date",
      last_seas: "$playerClubInfo.last_seas",
      citizen: "$playerClubInfo.citizen",
      current_club_dom_id: "$playerClubInfo.current_club_dom_id",
      current_club_code: "$playerClubInfo.current_club_code"
    }},
    {$out: "Final_Players"}])
```

Data Export: The final part of the code exports the aggregated and transformed data into a new collection called "Final_Players."

Further Calculations: After the data is in the "Final_Players" collection, two more operations are performed in separate aggregation queries:

- The first aggregation query calculates the "min_90" field, which is the result of dividing "total_minutes_played" by 90, essentially normalizing the minutes played to a 90-minute game.

```
  db.Final_Players.aggregate([ {
  $addFields: {
      min_90: {$divide: ["$total_minutes_played",90]}}}}, {
  $merge: {
    into: "Final_Players",
    whenMatched: "merge",
    whenNotMatched: "insert"
  }}])
```

- The second aggregation query calculates various statistics per 90 minutes (e.g., "goals_90," "assists_90," "conceded_90," and "scored_90") by dividing specific numeric fields by "min_90." This provides a per-90-minute view of player statistics.

```
db.Final_Players.aggregate([
  {
    $addFields: {
      goals_90: {
        $divide: [
          { $toDouble: "$total_goals" },
          "$min_90"]},
      assists_90: {
          $divide: [
          { $toDouble: "$total_assists" },
          "$min_90"]},
      conceded_90: {
          $divide: [
          { $toDouble: "$goals_conceded" },
          "$min_90"]},
      scored_90: {
          $divide: [
          { $toDouble: "$goals_scored" },
          "$min_90"
        ]}}},
```

- Both aggregation queries then merge the results back into the ”Final_Players” collection, updating existing documents with new values and inserting new documents if no match is found.

```
{
$merge: {
  into: "Final_Players",
  whenMatched: "merge",
  whenNotMatched: "insert"
}}])
```

4 Findings and results

Now, after preparing the database, we’re able to do the scout job, searching the players based on certain parameters of interest such that they are the best solution in relation with the player to substitute.

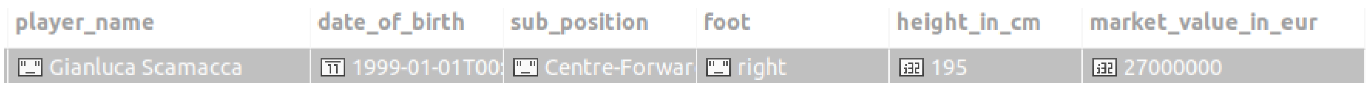
Starting from the oldest player, Zlatan Ibrahimovic, could be good to find as a solution for the striker position a player with a good scoring propensity and some physical characteristics as type of foot or height. The AC Milan needs a good striker in order to substitute a player like Ibrahimovic, so maybe spend at least ten millions:

```
var playerReference = db.Final_Players.findOne({
  ’player_id’: 3455,
  ’last_seas’: 2022
});

var characteristics = {
   ’market_value_in_eur’: { $gt: 10000000 },
   ’date_of_birth’: { $gt: new Date(’1997-01-01’) },
  ’sub_position’: playerReference.sub_position,
  ’height_in_cm’: { $gte: playerReference.height_in_cm},
  ’foot’: playerReference.foot,
  ’goals_90’: {$gt: 0.6}

};
db.Final_Players.find(characteristics);
```

This research leads us to the following result:



player_name	date_of_birth	sub_position	foot	height_in_cm	market_value_in_eur
Gianluca Scamacca	1999-01-01T00:	Centre-Forwar	right	195	27000000

Figure 4: Substitute of Zlatan Ibrahimovic

Hence, in the entire database, the best way for replacing Ibrahimovic based on his statistics is Gianluca Scamacca.

Let’s apply the same criteria for Edin Dzeko from Inter Milan changing a bit the parameters of interest:

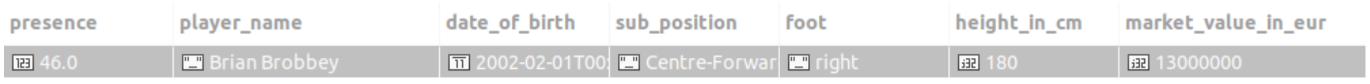
```
var playerReference = db.Final_Players.findOne({
  ’player_id’: 28396,
  ’last_seas’: 2022
});

var characteristics = {
      $and: [
        { ’market_value_in_eur’: { $gt: 10000000 } },
        { ’market_value_in_eur’: { $lt: 50000000 } }
      ],
   ’date_of_birth’: { $gt: new Date(’1998-01-01’) },
  ’sub_position’: playerReference.sub_position,
  'height_in_cm': { $lte: playerReference.height_in_cm},
  'foot': playerReference.foot,
  'goals_90': {$gt: 0.8},
  'presence': {$gt:30},

};

db.Final_Players.find(characteristics);
```

Searching a shorter player, with more than 0.8 goals scored per 90 minutes, at least 30 appearances and a value between ten and fifty millions, the result shows that:



presence	player_name	date_of_birth	sub_position	foot	height_in_cm	market_value_in_eur
46.0	Brian Brobbey	2002-02-01T00:	Centre-Forwar	right	180	13000000

Figure 5: Substitute of Edin Dzeko

Moving on to other positions in the field, we have to find a great alternative for Francesco Acerbi choosing from these parameters:

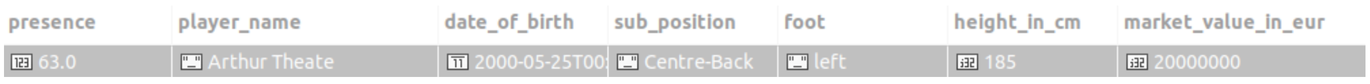
```
var playerReference = db.Final_Players.findOne({
  'player_id': 131075,
  'last_seas': 2022
});

var characteristics = {

   $and: [
        { 'market_value_in_eur': { $gt: 10000000 } },
        { 'market_value_in_eur': { $lt: 22000000 } }
      ],
   'date_of_birth': { $gt: new Date('2000-01-01') },
  'sub_position': playerReference.sub_position,
  'height_in_cm': { $lte: playerReference.height_in_cm},
  'foot': playerReference.foot,
  'conceded_90': {$lt: 2},
  'presence': {$gt:30}

};

db.Final_Players.find(characteristics);
```



presence	player_name	date_of_birth	sub_position	foot	height_in_cm	market_value_in_eur
63.0	Arthur Theate	2000-05-25T00:	Centre-Back	left	185	20000000

Figure 6: Substitute of Francesco Acerbi

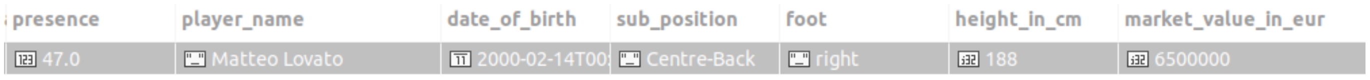
The correspondent player in AC Milan is Simon Kjaer, as a backup player that should grow in the bench with a market value lower than five millions and italian for championship rules leads us to:

```
var playerReference = db.Final_Players.findOne({
  'player_id': 48859,
  'last_seas': 2022
});

var characteristics = {
   'market_value_in_eur': { $lt: 10000000 },
   'date_of_birth': { $gt: new Date('2000-01-01') },
  'sub_position': playerReference.sub_position,
  'foot': playerReference.foot,
  'conceded_90': {$lt: 2},
  'citizen': 'Italy',
  'presence':{$gt: 10}

};

db.Final_Players.find(characteristics);
```



presence	player_name	date_of_birth	sub_position	foot	height_in_cm	market_value_in_eur
47.0	Matteo Lovato	2000-02-14T00	Centre-Back	right	188	6500000

Figure 7: Substitute of Simon Kjaer

For Mkhitaryan, offensive midfielder of Inter Milan, we obtain:

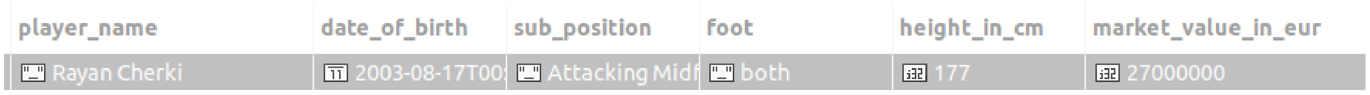
```
var playerReference = db.Final_Players.findOne({
  'player_id': 55735,
  'last_seas': 2022
});

var characteristics = {

   'market_value_in_eur': { $gt: 20000000 },
   'date_of_birth': { $gt: new Date('1999-01-01') },
  'sub_position': playerReference.sub_position,
  'foot': playerReference.foot,
  'scored_90': {$gt: 2},
  'presence':{$gt: 10},
  'assists_90':{$gt: 0.3}

};

db.Final_Players.find(characteristics);
```



player_name	date_of_birth	sub_position	foot	height_in_cm	market_value_in_eur
Rayan Cherki	2003-08-17T00	Attacking Midf	both	177	27000000

Figure 8: Substitute of Henrikh Mkhitaryan

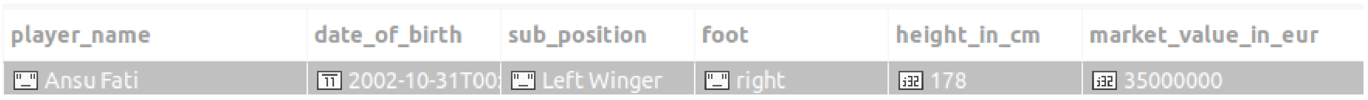
Finally, the last player for whom we want to find a replacement is Rebic, left wing of Milan:

```
var playerReference = db.Final_Players.findOne({
  'player_id': 187587,
  'last_seas': 2022
});

var characteristics = {
    $and: [
        { 'market_value_in_eur': { $gt: 20000000 } },
        { 'market_value_in_eur': { $lt: 40000000 } }
      ],
    'date_of_birth': { $gt: new Date('2000-01-01') },
  'sub_position': playerReference.sub_position,
  'foot': playerReference.foot,
  'scored_90': {$gt: 3},
  'presence':{$gt: 20},
  'goals_90':{$gt:0.5}

};

db.Final_Players.find(characteristics);
```



player_name	date_of_birth	sub_position	foot	height_in_cm	market_value_in_eur
Ansu Fati	2002-10-31T00:	Left Winger	right	178	35000000

Figure 9: Substitute of Ante Rebic

To conclude, through the use of MongoDB and Studio 3T, we performed an analysis using different sports datasets with a large amount of data. We obtained the hoped result that is to find possible replacements for the older players of the Milan and Inter Milan teams to rejuvenate them without, however, significantly worsening the team, rather trying to improve it; to do this in fact we made our choices based on a filter on certain relevant characteristics (some created by us and others already present).

5 References

MongoDB : https://www.mongodb.com/
Dataset repositories : https://data.world/dcereijo/player-scores
Studio 3T : https://studio3t.com/knowledge-base/article/mongodb-aggregation-framework/