# Advanced Data Management Project

Delaram Doroudgarian

Seyede Aida Atarzadeh Hosseini

2025/2026

## Workload

**Q1:** Retrieve the total minutes played for a given *player* in a selected *season*.

**Q2:** The stadiums with more than 50,000 seats, and all players who have played in along with the game information.

**Q3:** Retrieve all players who currently play for a given club, including their position and birth country.

**Q4:** For a selected *competition* and *season*, retrieve all games, showing the date, round, and the stadium name and attendance.

**Q5:** Retrieve the player position and valuation values for players with a given id, who have played at least one game in a given season (Minutes_played > 0) in a stadium with a given name, considering only valuation entries within a given date range.

**Q6:** For a given season, retrieve the top K players with the highest total minutes played, including player name and current club.

**Q7:** For a given competition and season, compute for each stadium the average attendance and number of games hosted.

**Q8:** Retrieve the total minutes played by each player across all games, including player name and current club.

**Q9:** Retrieve all valuation entries within a given date range for players currently playing for a given club.

**Q10:** Retrieve the total number of games played in each stadium, sorted in descending order of the number of games hosted.

**Step 1:**

```
Q1(
 E = Player,
   LS = [Player(player_id)_!, Game(season)_Ps],
   LP = [Game(minutes_played)_Ps]
```

)

Q2(

  E = Stadium,

  LS = [Stadium(seats)_!],

 LP = [ Stadium(name)_!, Player(player_id, name)_PsPd, Game_Pd]

)

Q3(

Entity = Club,

    LS = [Club(club_id)_!],

    LP = [Player(player_id, position)_CP, Country(Name)BirthCp ]

)

Q4(

E = Game,

LS = [Competition(competition_id)_Has, Game(season)_!],

LP = [Game(date, round)_!, Stadium(name, attendance)_Pd]

)

Q5 (

    E = Player,

    LS = [Player(player_id)_!, Stadium(name)_PdPs, Player_Valuation(date)_of,

    Game(season)_Ps]

    LP = [Player(position, name)_! Player_Valuation(market value)_of]

)

Q6 (

E = Game

LS = [Game(season)_!]

LP =  [Player(name, minutes_played)_Ps, Club(club_id)_CpPs]

)

Q7 (

E = Game

LS = [Game(season)_!, Competition(competition_id)_Has]

LP = [Game(game_id)_!, Stadium(attendance)_Pd]

)

Q8 (

E = Player

LS = []

LP = [Player(name)_!, Game(minutes_played)_Ps, Club(name)_Cp]

)

Q9(

E = Club

LS = [Club(club_id)_!, Player_valuation(date)_OfCp]

LP = [Player_valuation(market_value)__OfCp, Player(Player_id)_Cp]

)

Q10 (

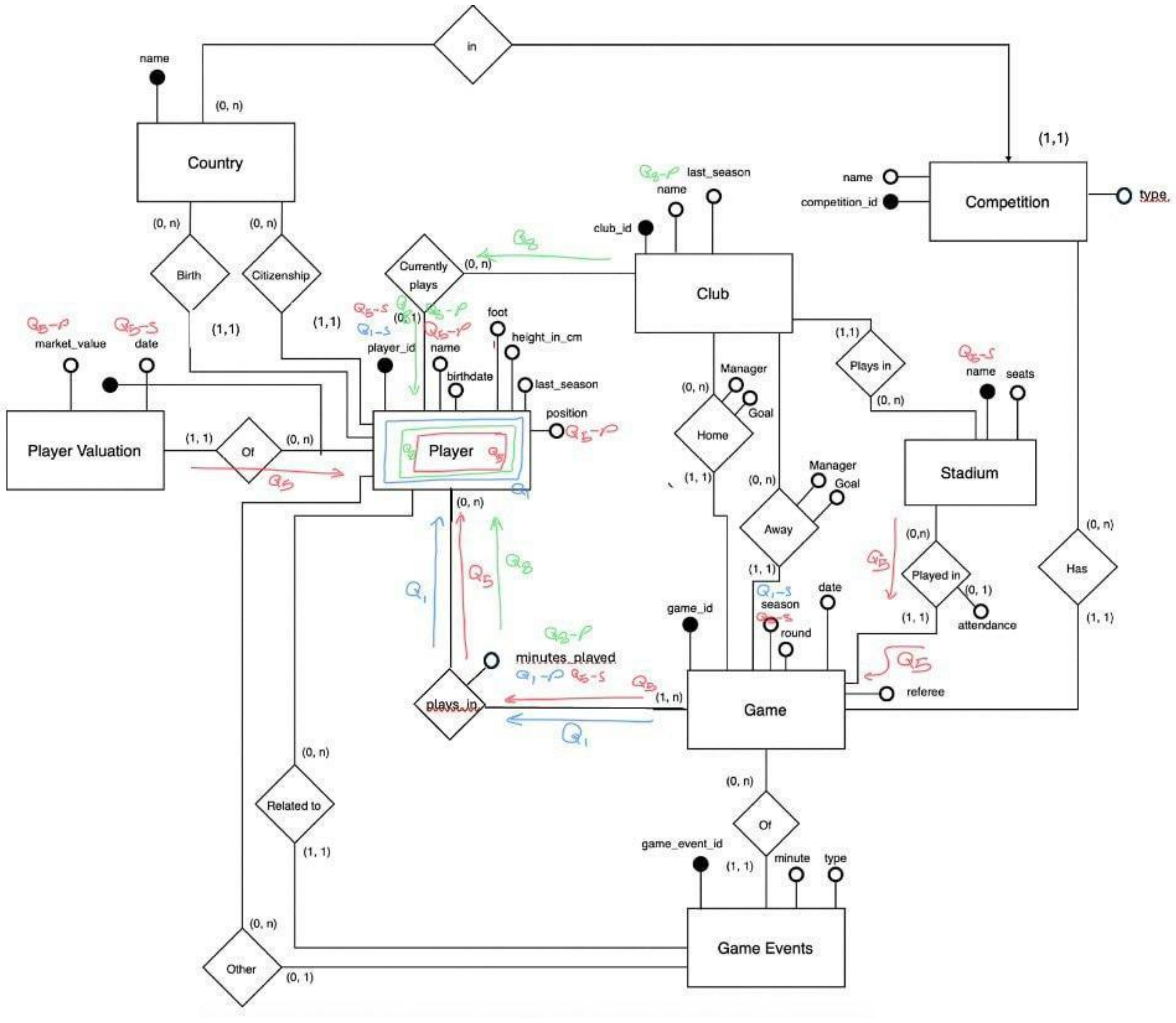
E = Game

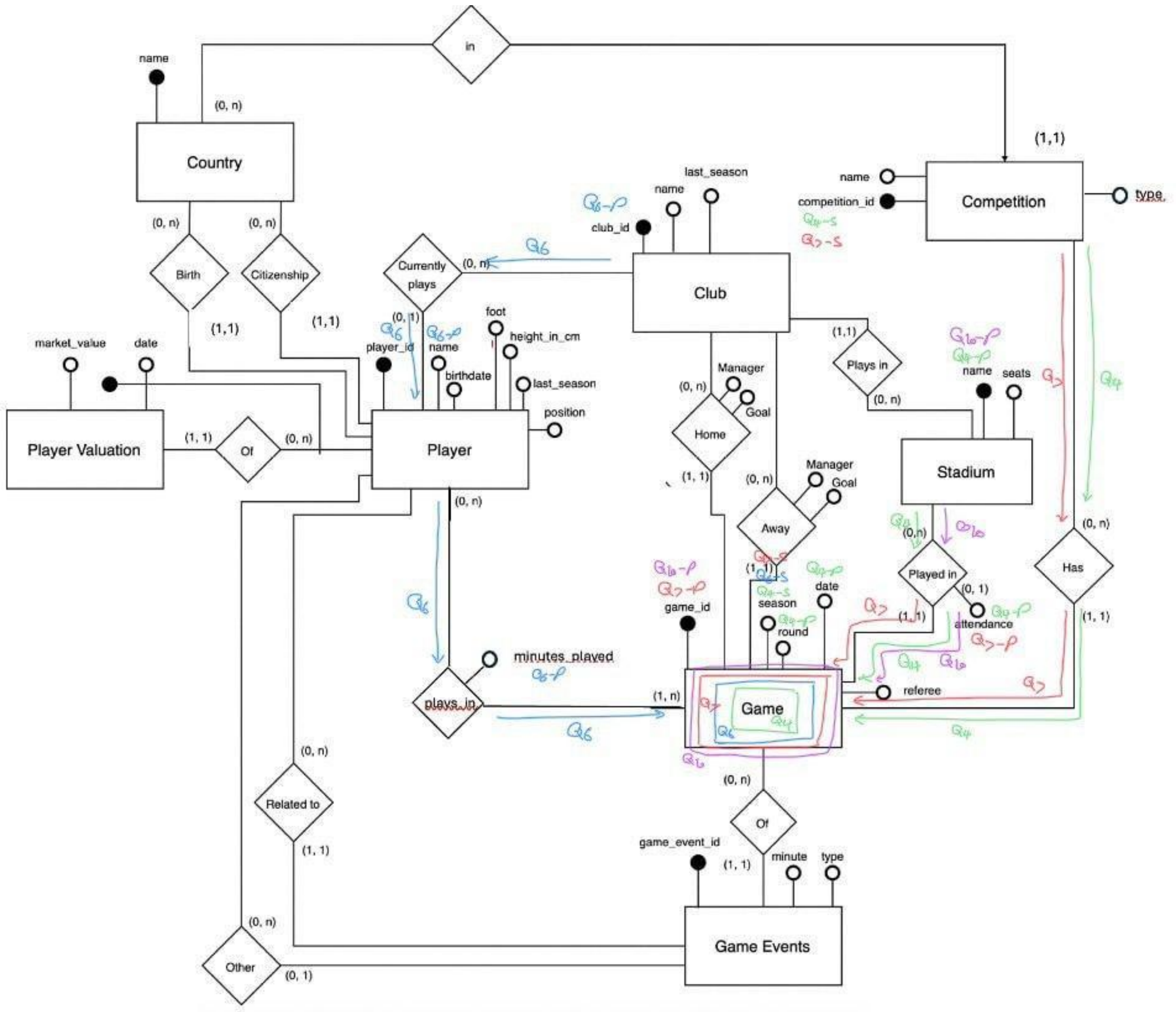
LS = []

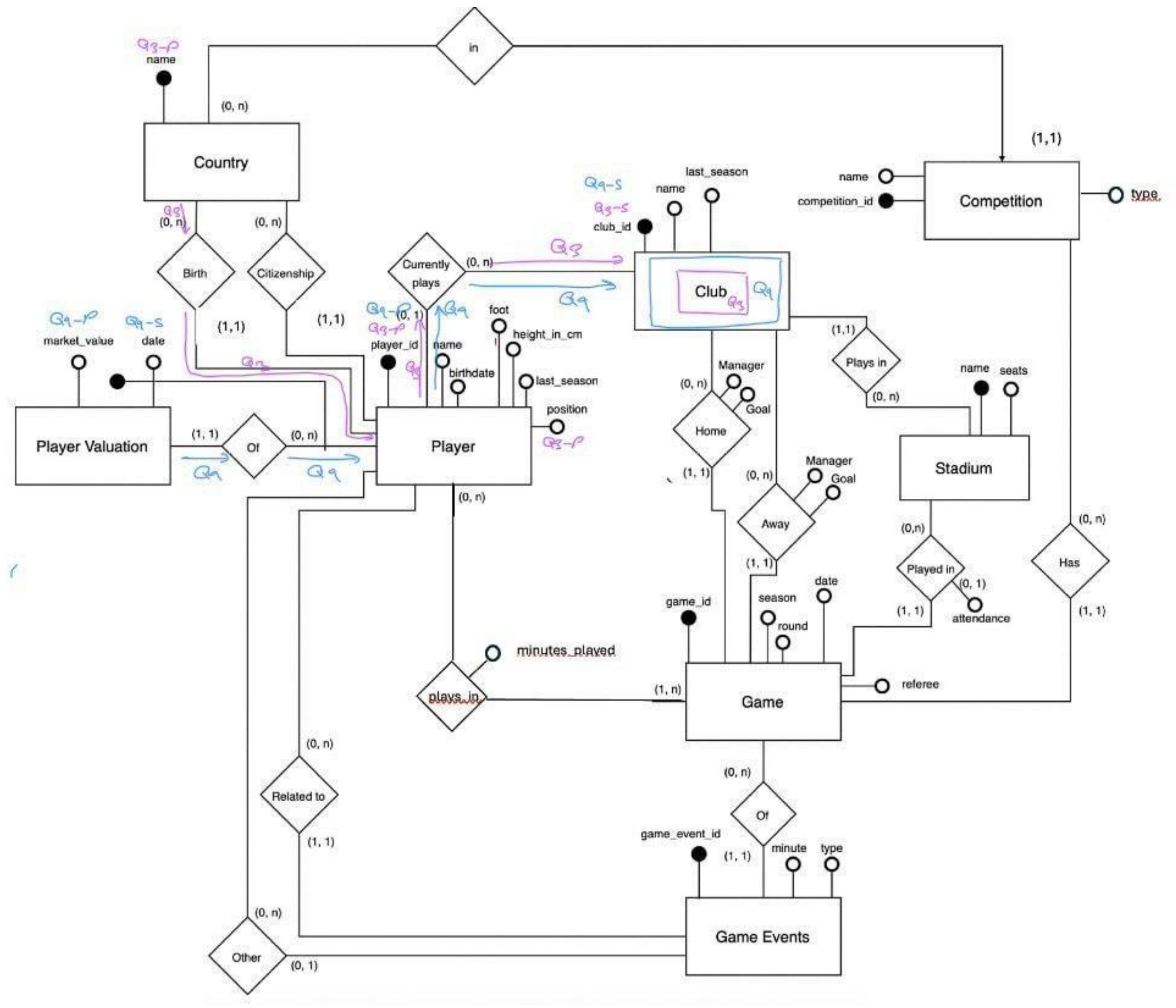
LP = [Game(game_id)_!, Stadium(name)_Pd]

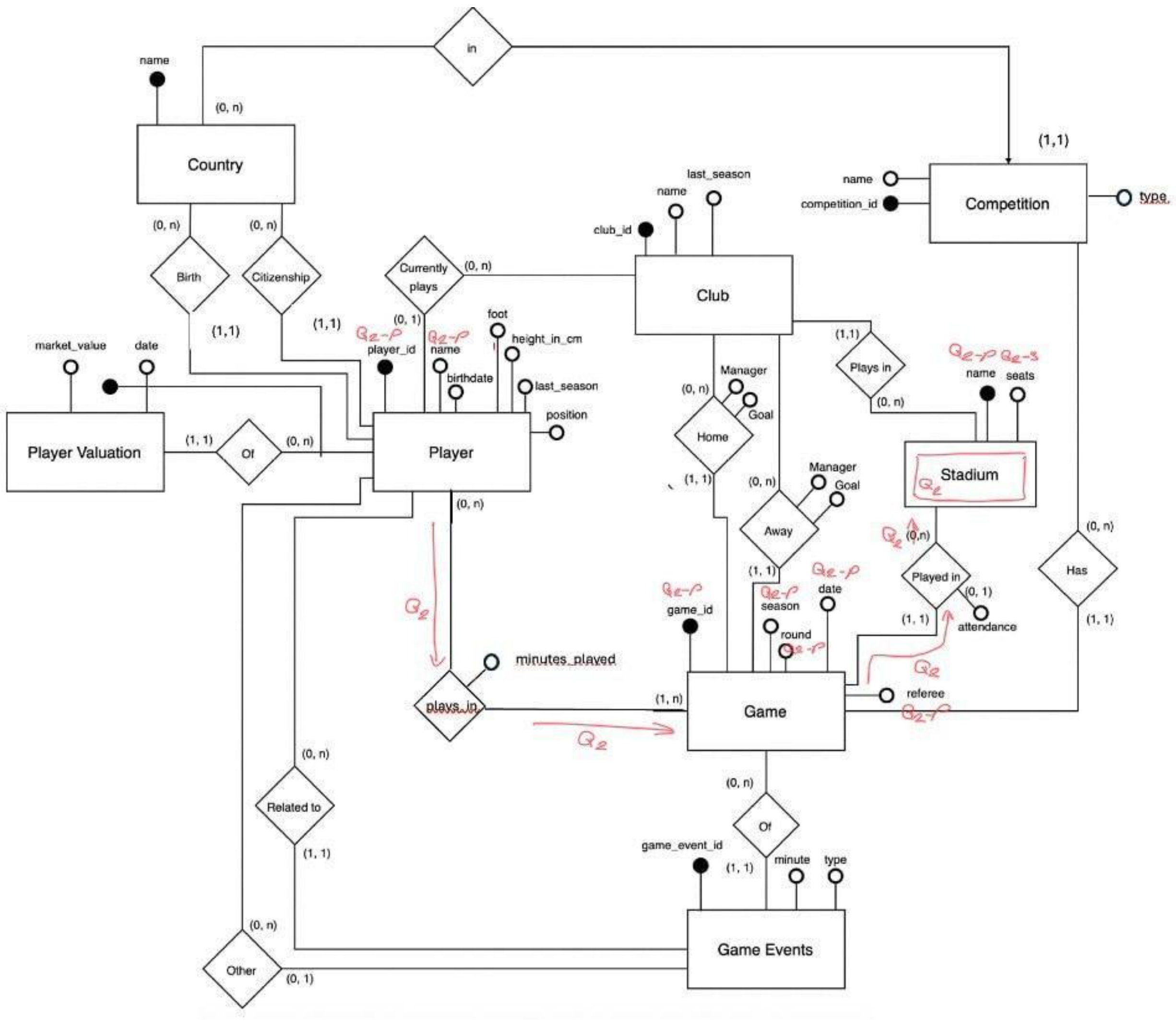
)

## Step 2




name
Country
(0, n)
in
(1,1)
Competition
name
competition_id
type
Birth
Citizenship
(1,1)
Player Valuation
market_value
date
Of
Player
player_id
name
birthdate
foot
height_in_cm
last_season
position
Currently plays
Club
club_id
last_season
Plays in
Stadium
seats
Home
Away
Manager
Goal
Played in
attendance
Has
Game
game_id
season
round
date
referee
plays_in
minutes_played
Related to
Other
Game Events
game_event_id
minute
type
Q6
Q7
Q4
Q10


in
Country
name
Competition
competition_id
type
Club
club_id
last_season
Currently plays
Birth
Citizenship
Player Valuation
market_value
date
Of
Player
player_id
birthdate
foot
height_in_cm
position
Plays in
Stadium
seats
Home
Away
Manager
Goal
Played in
attendance
Has
Game
game_id
season
round
referee
plays_in
minutes_played
Related to
Other
Game Events
game_event_id
minute
Q3-P
Q9-S
Q3-S
Q9-P
Q3
Q9



## Step 3:

Player (Q1, Q5, Q8) : {

Player_id, Name, Position,

Plays_in: [{Game:{Stadium_name, Season, Minutes_played}}],

Valuation: [{Date, Market_value}],

club_name

}

Game (Q4, Q6, Q7, Q10) : {

Game_id, Season, Date, Round,

Competition_id,

Played_in: {stadium_name, Attendance}}

Played_by: [{name, Minues_played, Club_id}]

}

Stadium (Q2): {

Seats, Name

Played_games: [{game_id,season,round,data,refree, players:[{player_id, name}]
}],

}

Club (Q3, Q9): {

club_id,

players: [{player_id,position, Country_of_birth, valuations:[{market_value, date}]}]

}

## Query Types

Q1 → **Random access**

Q2 → **Random access**

Q3 → **Random access**

Q4 → **Random access**

Q5 → **Random access**

Q6 → **Random access**

Q7 → **Random access**

Q8 → **Batch**

Q9 → **Random access**

Q10 → **Batch**

## Partition keys and super-identifiers:

Player Collection: Partition key = (Player_id), Super identifier = (Player_id)

- Q1: Not efficient (it contains one equality condition for the partition key attribute and further condition over another attribute → **So, one node but requested aggregates not sequentially stored on disk**)

- Q5: Not efficient ( it contains one equality condition for the partition key attribute and further conditions over other attributes → **So, one node but requested aggregates not sequentially stored on disk**)

Game Collection: Partition key = (Season, Competition_id), Super identifier = (season, Competition_id, Game_id)

- Q4: Efficient (it contains exactly one equality condition for each partition key attribute → **So, one node, sequentially stored aggregates**)

- Q6: Not efficient (it does not contain equality conditions over all partition key attributes → **So, more than one node**)

- Q7: Efficient (it contains exactly one equality condition for each partition key attribute → **So, one node, sequentially stored aggregates**)

- Q10: Not efficient (All nodes accessed)

Stadium Collection: Partition key = (Seats), Super identifier = (Seats, Name)

- Q2: Efficient (it contains exactly one equality condition for each partition key attribute → **So, one node, sequentially stored aggregates**)

Club Collection: Partition key = (Club_id), Super identifier = (Club_id)

- Q3: Efficient (it contains exactly one equality condition for each partition key attribute → **So, one node, sequentially stored aggregates**)
- Q9: Not efficient ( it contains one equality condition for the partition key attribute and further condition over another attribute → **So, one node but requested aggregates not sequentially stored on disk**)

## Indexes:

**Player Collection:**

- Local ordered index: (Player_id)
- Other useful indexes on:
    - (Season), to improve Q1 efficiency
    - (Stadium_name, Date), to improve Q5 efficiency

**Game Collection:**

- Local ordered index: (season, Competition_id)
- No other indexes are needed

**Stadium Collection:**

- Local ordered index: (Seats, Name)
- No other indexes are needed

**Club Collection:**

- Local ordered index: (Club_id)
- Another useful index on:
    - (Date), to improve Q9 efficiency

## Design in MongoDB:

### <u>Player Collection:</u>

```
{

 "_id": "P20458",

 "Name": "Lautaro Martinez",

 "Position": "Forward",

 "Games": [

   {

      "Season": "2022/2023",

      "Stadium_name": "San Siro",

      "Minutes_played": 90

    }

    {

      "Season": "2022/2023",

      "Stadium_name": "Olimpico",

      "Minutes_played": 78

     }

 ],

 "market_values": [
    {
      "Date": "2021-06-01",
      "Market_value": 60000000
    },
    {
      "Date": "2022-06-01",
      "Market_value": 75000000
    }
  ],
  "Currently_club": "Inter Milan"
}
```

**1- Create the collection:**

db.creaeCollection("Player")

**2- Create an index on the partition key attribute:**

db.player.createIndex({ "player_id": 1},{unique: true})

**3- Create the partition key:**

db.adminCommand(shardCollection: "db.player", key: {"player_id": 1})

**4. Create useful indices:**

db.player.createIndex({ "season":1}, {unique: false});
db.player.createIndex({"stadium_name":1,"date":1}, {unique: false});

**5. Q1: Retrieve the total minutes played for a given *player* in a selected *season*:**

```
db.player.aggregate([
  {
    $match: {
      _id: "P20458"
    }
  },
  {
    $unwind: "$games"
  },
  {
    $match: {
      "games.season": "2022/2023"
    }
  },
  {
    $group: {
      _id: "$_id",
       total_minutes: {
      $sum: "$games.minutes_played"
      }
    }
  }
])
```

**6. Q5: Retrieve the player position and valuation values for players with a given id, who have played at least one game in a given season (Minutes_played > 0) in a stadium with a given name, considering only valuation entries within a given date range:**

```
db.player.aggregate([
 {
		$match: {
			_id: "P20458"
		}
 },
{
    $unwind: "$games"
  },


  {
    $match: {
      "games.season": "2022/2023",
      "games.stadium_name": "San Siro",
      "games.minutes_played": { $gt: 0 }
    }
  },
  {
    $project: {
      _id: 0,
      position: 1,
      market_values: {
        $filter: {
          input: "$market_values",
          as: "v",
          cond: {
            $and: [
              { $gte: ["$$v.date", "2022-01-01"] },
              { $lte: ["$$v.date", "2023-12-31"] }
            ]
          }
        }
      }
    }
  }
])
```

**7. Q8: Retrieve the total minutes played by each player across all games, including player name and current club:**

```
db.player.aggregate([
  {
    $unwind: "$games"
  },
  {
    $group: {
      _id: "$_id",
      name: { $first: "$name" },
      club: { $first: "$currently_club" },
      total_minutes: {
        $sum: "$games.minutes_played"
      }
    }
  },
{$project: {
      _id: 0,
      name: 1,
      total_minutes: 1,
      club: 1
}
}
```

])

## Game Collection

```
{
_id: {
  "Game_id": "G10234",
  "Season": "2022/2023",
   "Competition_id": "C1",
}
"Date": "2023-03-18",
"Round": 27,
 "stadium_name": "San Siro",
 "attendance": 74231
  "Played_by": [
    {
      "name": "Lautaro Martinez",
      "minutes_played": 90,
      "club_id": "INT"
    },
    {
      "name": "Rafael Leao",
      "minutes_played": 85,
      "club_id": "MIL"
    },
    {
      "name": "Olivier Giroud",
      "minutes_played": 72,
      "club_id": "MIL"
    }
  ]
}
```

**1- Create the collection:**

```
db.creaeCollection(“game”)
```

**2- Create an index on the partition key attribute:**

```
db.game.createIndex({ "season": 1,"competition_id": 1},{unique: false})
```

**3- Create the partition key:**

```
db.adminCommand(shardCollection: “db.game”, key: {"season": 1,"competition_id": 1})
```

**4- Create a unique index on the super-identifier:**

```
db.game.createIndex({ "season": 1,"competition_id": 1, “game_id”: 1},{unique: true})
```

**5- Q4: For a selected *competition* and *season*, retrieve all games, showing the date, round, and the stadium name and attendance:**

```
db.game.find({competition_id: “C1”, season: "2022/2023"}, {_id: 0, date: 1, round: 1,
stadium_name: 1, attendance: 1})
```

**6- Q6: For a given season, retrieve the top K players with the highest total minutes played, including player name and current club:**

```
db.game.aggregate([
   {$match: {season: 2012}},
   {$unwind: "$played_by"},
   {$group: {_id: "$played_by.name", club_id: { $first: "$played_by.club_id" }, total_minutes:{$sum:
"$played_by.minutes_played"}}},
   {$sort: {total_minutes: -1}},
   {$limit: 5},
   {$project: {
    _id: 0,

    name: "$_id",

    club_id: 1,

    total_minutes: 1

  }}

])
```

**7- Q7: For a given competition and season, compute for each stadium the average attendance and number of games hosted.**

```
db.game.aggregate([

{ $match: {

    "competition_id": "C1",

    "season": "2022/2023"

}},

{ $group: {

    _id: "$stadium_name",

    avg_attendance: {

      $avg: "$attendance"

    },

    number_of_games: {

      $sum: 1

    }

  }

 },

{ $project: {

    _id: 0,

    stadium_name: "$_id",
    avg_attendance: 1,

    number_of_games: 1

  }

 }

])
```

**8- Q10: Retrieve the total number of games played in each stadium, sorted in descending order of the number of games hosted:**

```
db.game.aggregate([

{ $group: {

      _id: "$stadium_name",

      number_of_games: { $sum: 1 }

    }},

  {$sort: {

      number_of_games: -1

    }},

  { $project: {

      _id: 0,

      stadium_name: "$_id",

      number_of_games: 1

    }}

])
```

## Stadium Collection

```
{

_id: {
 “Seats”: 99354,
“Name”: "Camp Nou"
},
"Played_games": [
   {
     "game_id": "G4501",
     "season": "2022/2023",
     "round": 12,
     "date": "2022-11-06",
     "referee": "Antonio Mateu Lahoz",
     "players": [
       {
         "player_id": "P101",
         "name": "Robert Lewandowski"
       },
       {
         "player_id": "P102",
         "name": "Pedri"
       }
     ]
   },
   {
     "id": "G4789",
     "season": "2022/2023",
     "round": 18,
     "date": "2023-01-15",
     "referee": "Daniele Orsato",
      "players": [
       {
         "player_id": "P103",
         "name": "Gavi"
       },
       {
         "player_id": "P104",
         "name": "Ansu Fati"
       }
      ]
    }
  ]
}
```

**1- Create the collection:**

db.creaeCollection(“Stadium”)

**2- Create an index on the partition key attribute:**

db.Stadium.createIndex({ "seats": 1},{unique: false})

**3- Create the partition key:**

db.adminCommand(shardCollection: “db.Stadium”, key: {"seats": 1})

**4- Create a unique index on the super-identifier:**

db.Stadium.createIndex({ "seats": 1,"name": 1},{unique: true})

**5- Q2: The stadiums with more than 50,000 seats, and all players who have played in along with the game information:**

db.stadium.find({seats: {$gte: 50000}}, {_id: 0, name:1, seats: 1, played_games: 1})

## Club Collection

```
{
        “_id”:{
                “club_id”:”C5678”,
        }
        "players": [
        {
                "player_id": "P20458",
                "position": "Forward",
                "Country_of_birth": "Argentina"
                “market_values”: [
                {
                        market_value: 75000000,
                        date: 2023-06-01
                }]
        },
        {
                "player_id": "P17821",
                "position": "Midfielder",
                "country_of_birth": "Italy"
                “market_values”: [
                {
                        market_value: 75000000,
                        date: 2023-06-01
                }]
        },
        ]
```

}

**1- Create the collection:**

```
db.creaeCollection(“Club”)
```

**2- Create an index on the partition key attribute:**

```
db.club.createIndex({ "club_id": 1},{unique: true})
```

**3- Create the partition key:**

```
db.adminCommand(shardCollection: “db.club”, key: {"club_id": 1})
```

**4- Q3: Retrieve all players who currently play for a given club, including their position and birth country:**

```
db.club.find({ _id: "C5678" }, { _id: 0, players: {player_id: 1, position: 1, country_of_birth: 1}})
```

**5- Q9: Retrieve all valuation entries within a given date range for players currently playing for a given club:**

```
db.club.aggregate([

{$match:{ _id: 2578}},

{$unwind: "$players"},

{$unwind: "$players.valuations"},

{$match:{"players.valuations.date": {$gte: "2012-06-22", $lte: "2015-11-23"}}},

{$project:

{

_id: 0,

Player_id: "$players.player_id",

Market_value: "$players.valuations.market_value_in_eur",

Date: "$players.valuations.date"

}

}

])
```

# Design in Cassandra:

## Player Collection:

### 1- Choosing UDT for representing complex types of Plays_in and Valuation appearing in the schema:

```
CREATE TYPE game_t(

            stadium_name text,
            season text,
            minutes_played int

);

CREATE TYPE valuation_t(

            val_date date,
            market_value int

);
```

### 2- Creating table and specifying primary key:

```
CREATE TABLE player(

            player_id text PRIMARY KEY,
            name text,
            position text,
            plays_in list<frozen<game_t>>,
            valuation list<froxen<valuation_t>>,
            currently_club text,

);
```

### 3-insert:

```
INSERT INTO player (

  player_id,

  name,

  position,

  plays_in,
  valuation,

  currently_club

)

VALUES(

        “P20458”,
        “Lautaro Martinez”,
        “Forward”,
        [

                { "season": "2022/2023", "stadium_name": "San Siro","minutes_played": 90},
                {"season": "2022/2023", "stadium_name": "Olimpico","minutes_played": 78},

        ],

        [

                { "val_date": "2021-06-01", "market_value": 6000000},
                { "val_date": “2022-06-01", "market_value": 75000000},

        ],
        "Inter Milan"

)
```

**4- Q1: Retrieve the total minutes played for a given *player* in a selected *season*:**

```
SELECT SUM(plays_in.minutes_played)
FROM player
WHERE player_id =”P20458”
```

NOT ADMITTED: it is not admitted in Cassandra because the query requires aggregation (SUM of minutes_played) over a collection of nested values. Cassandra does not support aggregation or filtering on elements inside collections or UDTs.

**5- Q5: Retrieve the player position and valuation values for players with a given id, who have played at least one game in a given season (Minutes_played > 0) in a stadium with a given name, considering only valuation entries within a given date range:**

```
SELECT position, valuation.market_value
FROM player
WHERE player_id=”P20458” AND plays_in.season=”2022/2023” AND valuation.val_date >= '2021-01-01'
```

NOT ADMITTED: Q5 is not admitted in Cassandra because it requires filtering on nested attributes inside collections (plays_in and valuation), checking the existence of at least one matching element, and applying range conditions on valuation dates.

### 6- Q8: Retrieve the total minutes played by each player across all games, including player name and current club.

SELECT name, currently_club, SUM(plays_in.minutes_played)

FROM player

NOT ADMITTED: it is not admitted in Cassandra because the query requires aggregation (SUM of minutes_played) over a collection of nested values. Cassandra does not support aggregation or filtering on elements inside collections or UDTs.

## Game Collection:

### 1- Choosing UDT for representing complex type of Played_by appearing in the schema:

- ➔ Create type playerInfo_t(Name text, Minutes_played integer, Club_id text);
- ➔ Create type stadium_t(name text, attendance integer);

### 2- Creating table and specifying primary key:

```
➔ Create table Game(

        Game_id text,

        Season text,

        Date text,

        Round integer,

        Completion_id text,

        Played_in frozen<stadium_t>

        Played_by set<frozen<playerInfo_t>>

        Primary key ((season, competition_id), game_id)

  )
```

### 3- Insert:

➔ **Insert into Game**

**Values**(“G102340”,"2022/2023", "2023-03-18", 27, "C1", {"Name": "San Siro", "Attendance": 74231},

{

{"name": "Lautaro Martinez", "Minutes_played": 90, "Club_id": "INT"},

{"name": "Rafael Leao","Minutes_played": 85,"Club_id": "MIL"},

{"name": "Olivier Giroud", "Minutes_played": 72, "Club_id": "MIL"}

}

)

**4- CQL for Q4: For a selected *competition* and *season*, retrieve all games, showing the date, round, and the stadium name and attendance:**

➔ Select game_id, date, round, played_in

From Game

Where season = ”2022/2023”

AND competition_id = “C1”

(ADMITTED)

**5- CQL for Q6: For a given season, retrieve the top K players with the highest total minutes played, including player name and current club:**

➔ Select played_by.name, played_by.club_id, sum(played_by.minutes_played) AS total_minutes_played

from Game

where season = “2022/2023”

group by played_by.name, played_by.club_id,

order by total_minutes_played desc

limit 5

NOT ADMITTED: It requires aggregation, grouping, and ordering over non-primary-key attributes stored inside a frozen collection. According to the query-driven design approach, a dedicated table is required.

**6- CQL for Q7: For a given competition and season, compute for each stadium the average attendance and number of games hosted:**

➔ Select played_in.name, avg(played_in.attendance) AS avg_attend, sum(1) AS num_hosted_games

From Game

Where season = “2022/2023” and competition_id = “C1”

Group by played_in.name

➔ NOT ADMITTED: It requires aggregation and grouping over non-primary-key attributes stored inside a frozen collection. According to the query-driven design approach, a dedicated table is required.

### 7- CQL for Q10: Retrieve the total number of games played in each stadium, sorted in descending order of the number of games hosted.

➔ Select played_in, sum(game_id) as num_played_game

From Game

Group by played_in

Sorted by num_played_game desc

➔ NOT ADMITTED: It requires aggregation and grouping over non-primary-key attributes. According to the query-driven design approach, a dedicated table is required.

## Club Collection

### 1- Choosing UDT for representing complex types of players and market_value appearing in the schema:

➔ Create type market_value_t(market_value int,valuation_date date)

➔ Create type playerInfo_t (

```
        Player_id text,
        Position text,
        Country_name text,
        market_values list<frozen<market_value_t>>
)
```

### 2- Creating table and specifying primary key:

➔ Create table club (

Club_id text partition key,

Players list<frozen<playerInfo_t >>

)

### 3- Insert:

➔ **Insert into club**

**Values** (

“INT”,

[

{

```
        "player_id": "P20458",
        "position": "Forward",
        "Country_name": "Argentina",
           market_values: [

            { market_value: 60000000, valuation_date: '2021-06-01' },

            { market_value: 75000000, valuation_date: '2022-06-01' }

           ]
```

},

{

```
        player_id: 'P30112',
        position: 'Midfielder',
        country_name: 'Italy',
        market_values: [

            { market_value: 25000000, valuation_date: '2021-07-01' }

           ]
```

}

]

)

## 4- CQL for Q3: Retrieve all players who currently play for a given club, including their position and birth country:

➔ Select Players

From club

Where Club_id = “INT”

(ADMITTED)

**5- CQL for Q9: Retrieve all valuation entries within a given date range for players currently playing for a given club:**

```
SELECT players.market_values

FROM club

WHERE club_id = 'INT'

  AND players.market_values.valuation_date >= '2021-01-01'

  AND players.market_values.valuation_date <= '2022-01-01';
```

- ➔ NOT ADMITTED: It is not admitted because Cassandra does not allow selection conditions on nested attributes inside collections; therefore, filtering by valuation date must be performed at the application level.

## Studium Collection

**1- Choosing UDT for representing a complex type of game, player_info appearing in the schema:**

```
CREATE TYPE player_info_t(

        player_id text,
        name text

);

CREATE TYPE game_t(

        game_id text,
        season text,
        round int,
        game_date date,
        referee text,
        players list<frozen<player_info_t>>

);
```

**2- Creating table and specifying primary key:**

```
CREATE TABLE stadium(

        Name text,
        Seats int,
        Played_games list<frozen<game_t>>
        PRIMARY KEY (seats, name)

)
```

## 3- Insert:

```
INSERT INTO stadium(

        name,
        seats,
        played_games

)

VALUES(

        'Camp Nou',
        99354,
         [

  {

           game_id: 'G4501',

           season: '2022/2023',

           round: 12,

           game_date: '2022-11-06',

           referee: 'Antonio Mateu Lahoz',

           players: [

             { player_id: 'P101', player: 'Robert Lewandowski' },

             { player_id: 'P102', player: 'Pedri' }

           ]

  },

  {

           game_id: 'G4789',

           season: '2022/2023',

           round: 18,

           game_date: '2023-01-15',

           referee: 'Daniele Orsato',
           players: [
             { player_id: 'P103', name: 'Gavi' },
             { player_id: 'P104', name: 'Ansu Fati' }
           ]
  }
]
)
```

**4- Q2: The stadiums with more than 50,000 seats, and all players who have played in along with the game information.**

```
SELECT *
FROM stadium
WHERE seats > 50000
```

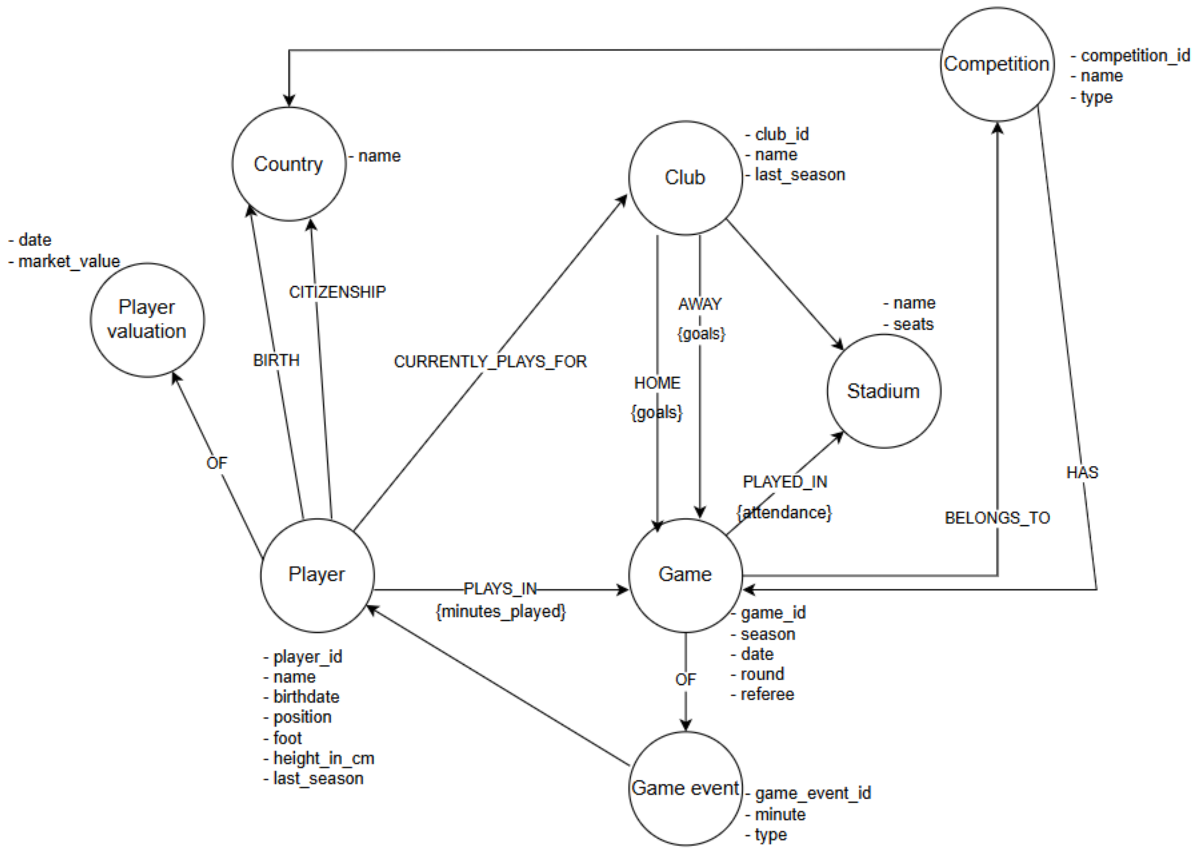
(ADMITTED)

## Design in Neo4j:

The original ER diagram has been translated into a Neo4j graph schema by mapping entities to node labels and associations to relationship types.

Entity attributes have been modeled as node properties, while attributes depending on associations (e.g., minutes played, attendance) have been modeled as relationship properties.

This graph-based representation preserves the semantics of the ER model and naturally supports path-based queries of the workload.



**Q1:** Retrieve the total minutes played for a given *player* in a selected *season*

➔ MATCH (p:Player {player_id: "P101"}) -[r:PLAYS_IN]-> (g:Game {season: "2022/2023"})

RETURN SUM(r.minutes_played) AS total_minutes_played;

**Q2:** The stadiums with more than 50,000 seats, and all players who have played in along with the game information.

➔ MATCH (s:Stadium) <-[pi:PLAYED_IN]- (g:Game) <-[r:PLAYS_IN]- (p:Player)

WHERE s.seats > 50000

RETURN s, p, g;

**Q3:** Retrieve all players who currently play for a given club, including their position and birth country.

➔ MATCH (p:Player)-[:CURRENTLY_PLAYS_FOR]->(cl:Club {club_id: "INT"}), (p)-[:BIRTH]->(co:Country)

RETURN  p.name AS player_name, p.position AS position, co.name AS birth_country;

**Q4:** For a selected *competition* and *season*, retrieve all games, showing the date, round, and the stadium name and attendance.

➔ MATCH (c:Competition {competition_id: "C1"}) <-[:BELONGS_TO]- (g:Game {season: "2022/2023"}) -[pi:PLAYED_IN]-> (s:Stadium)

RETURN g.date AS game_date, g.round, s.name AS stadium_name, pi.attendance AS attendance;

**Q5:** Retrieve the player position and valuation values for players with a given id, who have played at least one game in a given season (Minutes_played > 0) in a stadium with a given name, considering only valuation entries within a given date range.

➔ MATCH (p:Player {player_id: "P101"}) -[r:PLAYS_IN]-> (g:Game {season: "2022/2023"}) -[:PLAYED_IN]-> (s:Stadium {name: "San Siro"}), (p) -[:OF]-> (v:Player_Valuation)

WHERE r.minutes_played > 0 AND v.date >= "2023-01-01" AND v.date <= "2023-12-31"

RETURN p.position AS player_position, v.market_value AS market_value, v.date AS valuation_date;

**Q6:** For a given season, retrieve the top K players with the highest total minutes played, including player name and current club.

➔ MATCH (p:Player)-[r:PLAYS_IN]->(g:Game {season: "2022/2023"}), (p)-[:CURRENTLY_PLAYS_FOR]->(cl:Club)

WITH p.player_id AS pid, p.name AS player_name, cl.name AS current_club, sum(pi.minutes_played) AS total_minutes

ORDER BY total_minutes DESC

LIMIT 5

RETURN player_name,  current_club, total_minutes

**Q7:** For a given competition and season, compute for each stadium the average attendance and number of games hosted.

➔ MATCH (co:Competition {competition_id: "C1"}) <-[:BELONG_TO]- (g:game {season: “2022/2023”}), (s:Stadium) <-[pi:PLAYED_IN]- (g)

```
RETURN s.name AS stadium, AVG(pi.attendance) as average_attendance, COUNT(g) AS
number_of_hosted_games
```

**Q8:** Retrieve the total minutes played by each player across all games, including player name and current club.

```
MATCH (p:Player)-[r:PLAYS_IN]->(g:Game), (p)-[:CURRENTLY_PLAYS_FOR]->(cl:Club)

RETURN

  p.name AS player_name,

  c.name AS current_club,

  sum(r.minutes_played) AS total_minutes_played
```

**Q9: Retrieve all valuation entries within a given date range for players currently playing for a given club.**

```
MATCH (p:Player)-[:CURRENTLY_PLAYS_FOR]->(c:Club), (p)-[:OF]->(v:Player_valuation)

WHERE c.club_id = “c1”

 AND v.date >= "2023-01-01"

 AND v.date <= "2023-06-01"

RETURN

  p.player_id AS player_id,

  v.market_value AS market_value
```

**Q10: Retrieve the total number of games played in each stadium, sorted in descending order of the number of games hosted.**

```
MATCH (g:Game)-[:PLAYED_IN]->(s:Stadium)

RETURN

  s.name AS stadium_name,

  count(g) AS total_games

ORDER BY total_games DESC
```

## The most suitable system for our application

In the designed system, data are organized into well-defined aggregates such as **Player**, **Game**, **Club**, and **Stadium**, and the system workload mainly consists of analytical queries based on aggregation, grouping, and sorting operations. Queries such as computing the total minutes played by players (Q1, Q8), retrieving the top players with the highest total minutes played in a season (Q6), and computing the average attendance and number of games hosted by each stadium (Q7) are executed directly over nested data within these aggregates. Therefore, the system requires a database that can efficiently support complex aggregation operations. Cassandra is not suitable for this system because it does not support aggregation and filtering over data stored inside collections and UDTs, and supporting the workload would require designing multiple query-specific tables and applying excessive denormalization.

Although Neo4j is well suited for modeling relationships among entities, the query pattern of this system is not graph-analytic or traversal-heavy. The primary focus of the queries is on statistical and analytical computations rather than on complex graph traversals. In this system, relationships among entities can be materialized at the data design level, eliminating the need for multi-hop traversals at runtime. In contrast, MongoDB, with its aggregate-oriented data model and mature aggregation framework, enables direct execution of the system queries over aggregates. By consciously adopting denormalization, this approach reduces query complexity and provides simple, predictable, and efficient access patterns for the defined workload.

## logical schema in system Mongo DB

**Player Collection**

```
player: {
        Player_id, Name, Position,
        Games: [{Stadium_name, Season, Minutes_played}],
        market_values: [{Date, Market_value}],
        Currently_club,
}
```

**Game Collection**

```
game: {
            Game_id, Season, Date, Round,
            Competition_id,
            stadium_name, Attendance,
            Played_by: [{name, Minues_played, Club_id}]
}
```

**Stadium Collection**

```
stadium: {
            Seats, Name
            Played_games: [{game_id,season,round,data,referee, players:[{player_id, name}]
}],
}
```

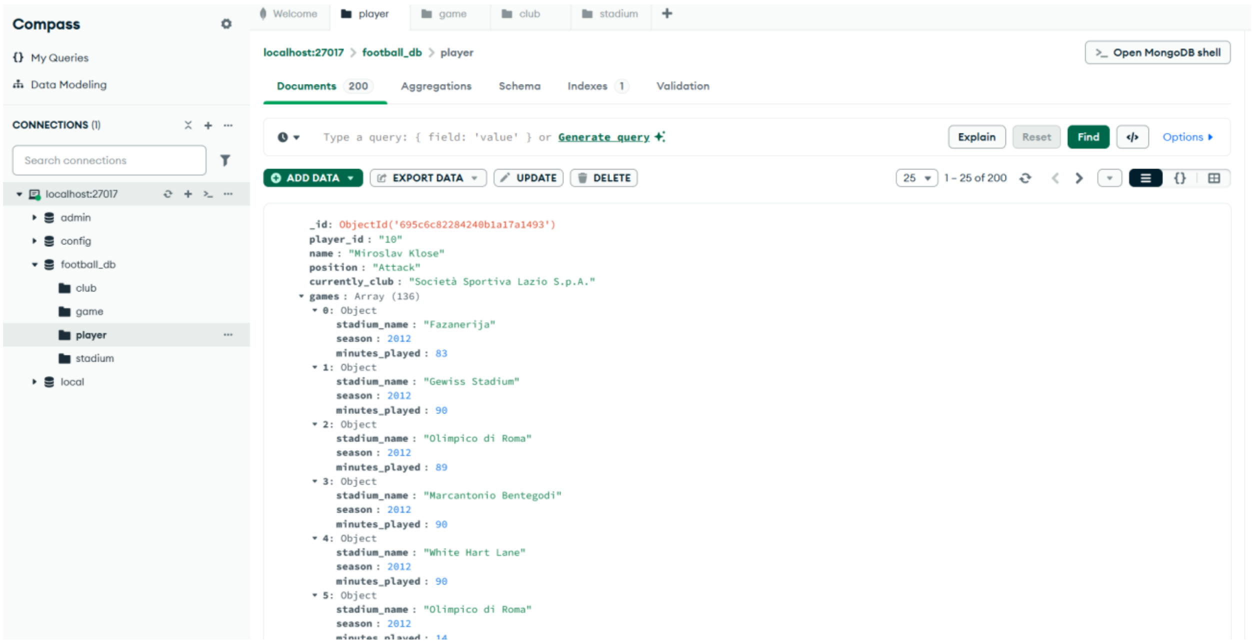
**Club Collection**

```
club: {
            club_id,
            players: [{player_id,position, Country_of_birth, market_values[{market_value,
           date}]}]
}
```

## Creating an instance of our schema in MongoDB:

To instantiate the logical schema in MongoDB, a real-world football dataset with a reasonable size (few MB) was used. The original data was transformed into JSON format to match the aggregate-oriented schemas, introducing nested documents and arrays according to the design.

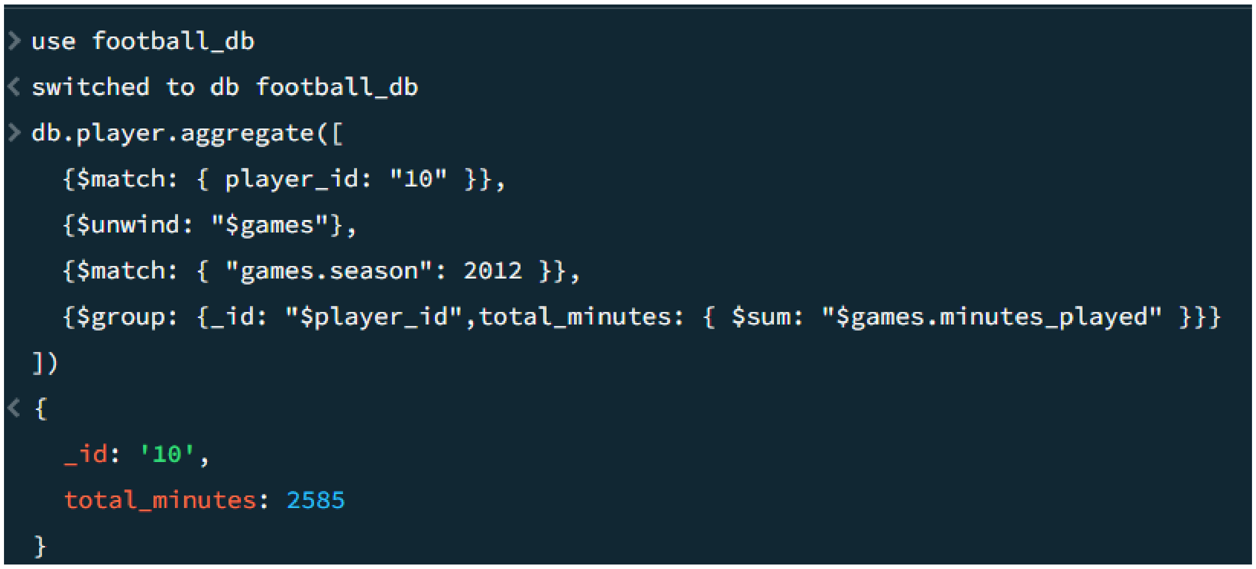
A database named **football_db**, along with the required collections, was created using **MongoDB Compass**. The transformed data (JSON files) was imported into the **player**, **game**, **club**, and **stadium** collections, each corresponding to one logical aggregate defined in the previous step.



```
_id: ObjectId('695c6c82284240b1a17a1493')
player_id : "10"
name : "Miroslav Klose"
position : "Attack"
currently_club : "Società Sportiva Lazio S.p.A."
games : Array (136)
  0: Object
    stadium_name : "Fazanerija"
    season : 2012
    minutes_played : 83
  1: Object
    stadium_name : "Gewiss Stadium"
    season : 2012
    minutes_played : 90
  2: Object
    stadium_name : "Olimpico di Roma"
    season : 2012
    minutes_played : 89
  3: Object
    stadium_name : "Marcantonio Bentegodi"
    season : 2012
    minutes_played : 90
  4: Object
    stadium_name : "White Hart Lane"
    season : 2012
    minutes_played : 90
  5: Object
    stadium_name : "Olimpico di Roma"
    season : 2012
    minutes_played : 14
```

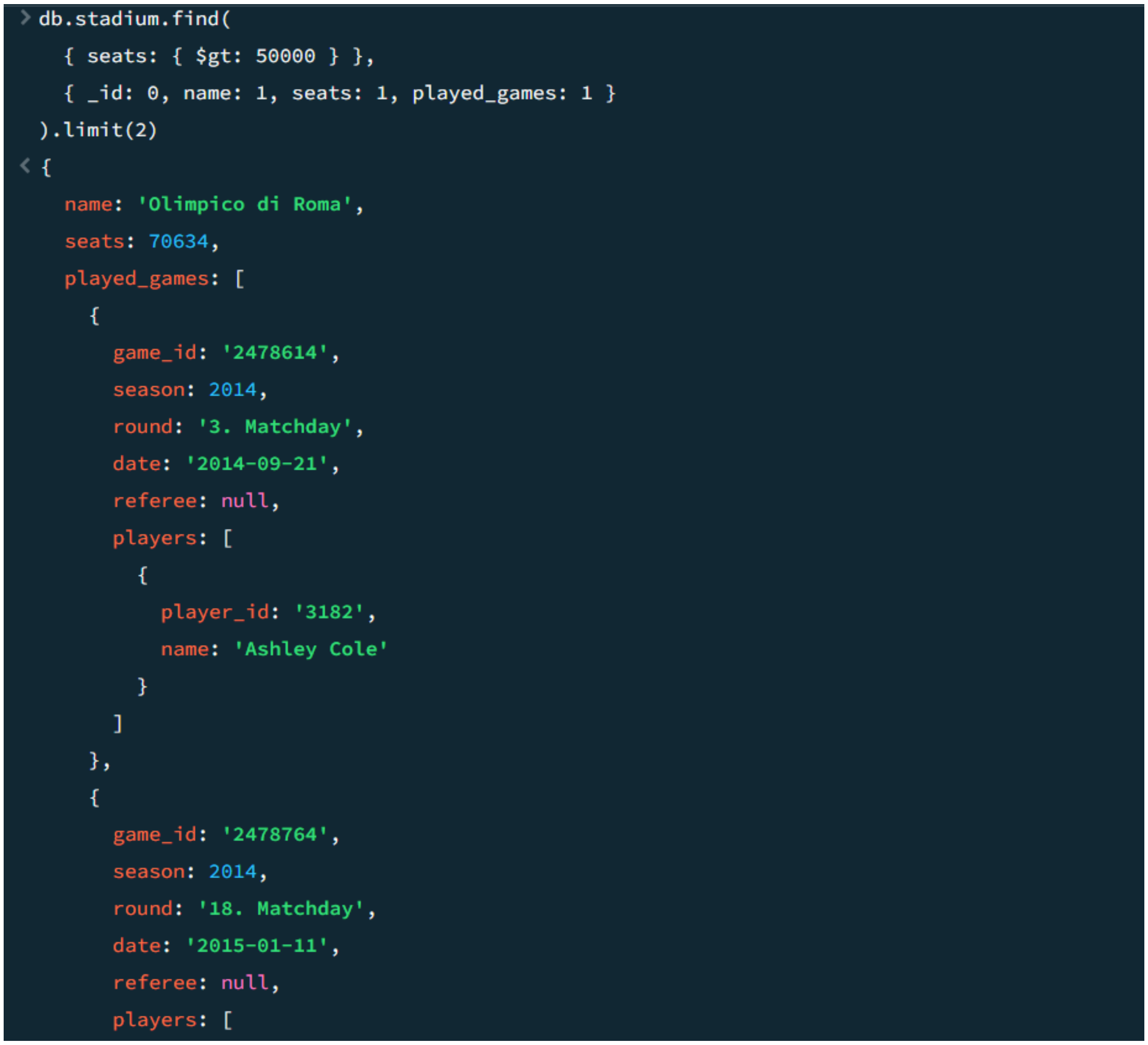
## Implementation of the workload in system

**Q1:** Retrieve the total minutes played for a given *player* in a selected *season*.

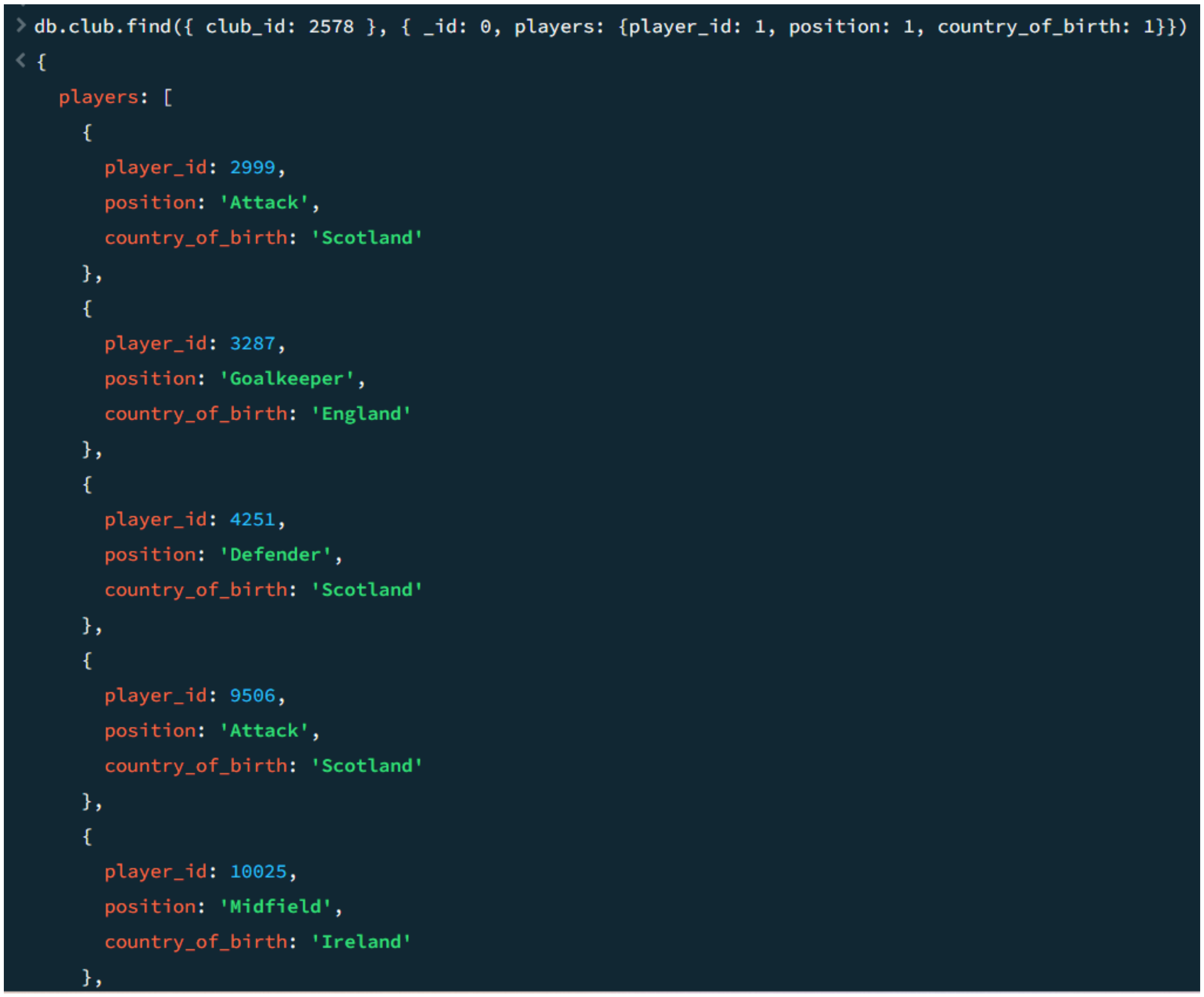


```
> use football_db
< switched to db football_db
> db.player.aggregate([
    {$match: { player_id: "10" }},
    {$unwind: "$games"},
    {$match: { "games.season": 2012 }},
    {$group: {_id: "$player_id",total_minutes: { $sum: "$games.minutes_played" }}}
  ])
< {
    _id: '10',
    total_minutes: 2585
  }
```

**Q2:** The stadiums with more than 50,000 seats, and all players who have played in along with the game information.

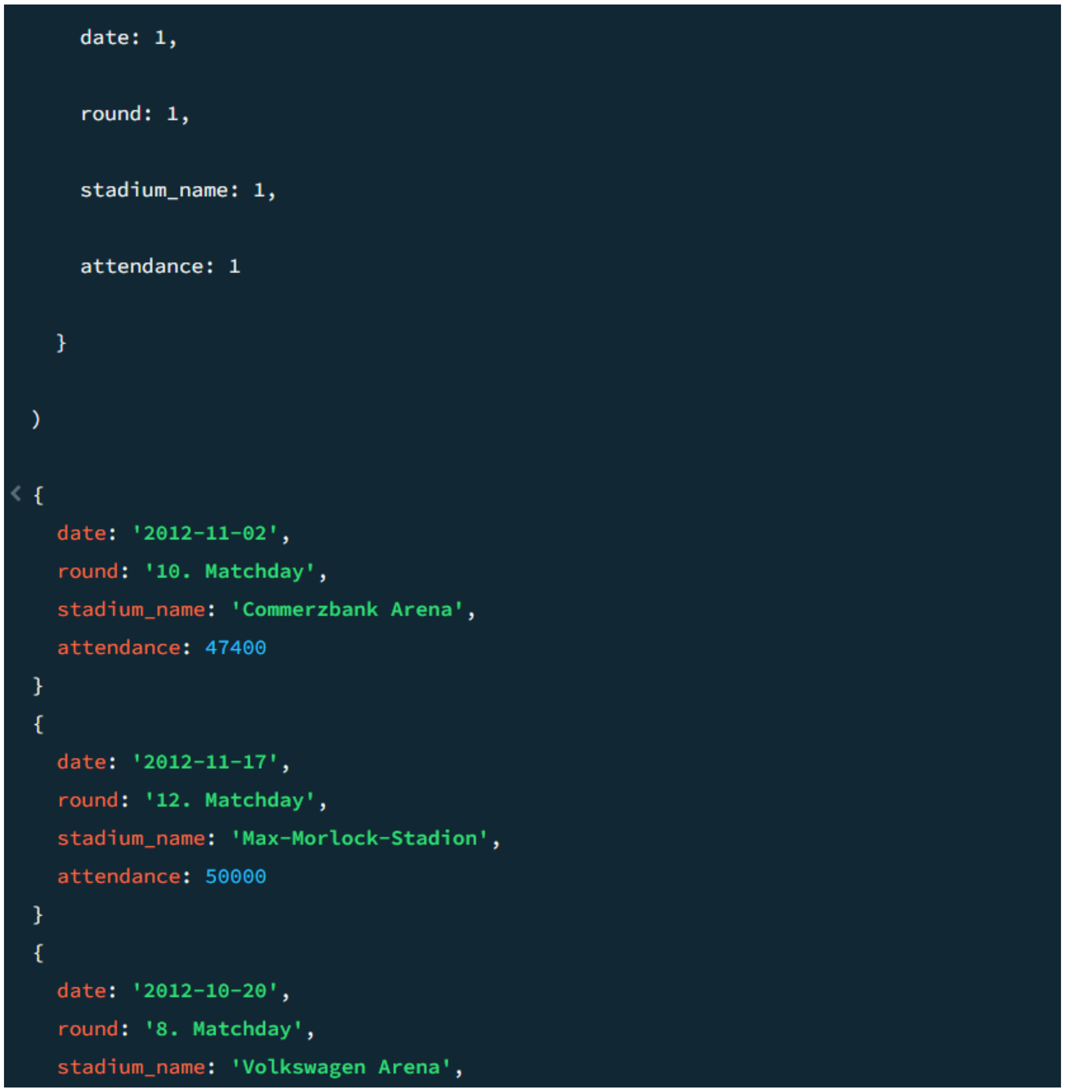

```
> db.stadium.find(
    { seats: { $gt: 50000 } },
    { _id: 0, name: 1, seats: 1, played_games: 1 }
  ).limit(2)
< {
    name: 'Olimpico di Roma',
    seats: 70634,
    played_games: [
      {
        game_id: '2478614',
        season: 2014,
        round: '3. Matchday',
        date: '2014-09-21',
        referee: null,
        players: [
          {
            player_id: '3182',
            name: 'Ashley Cole'
          }
        ]
      },
      {
        game_id: '2478764',
        season: 2014,
        round: '18. Matchday',
        date: '2015-01-11',
        referee: null,
        players: [
```

**Q3:** Retrieve all players who currently play for a given club, including their position and birth country.



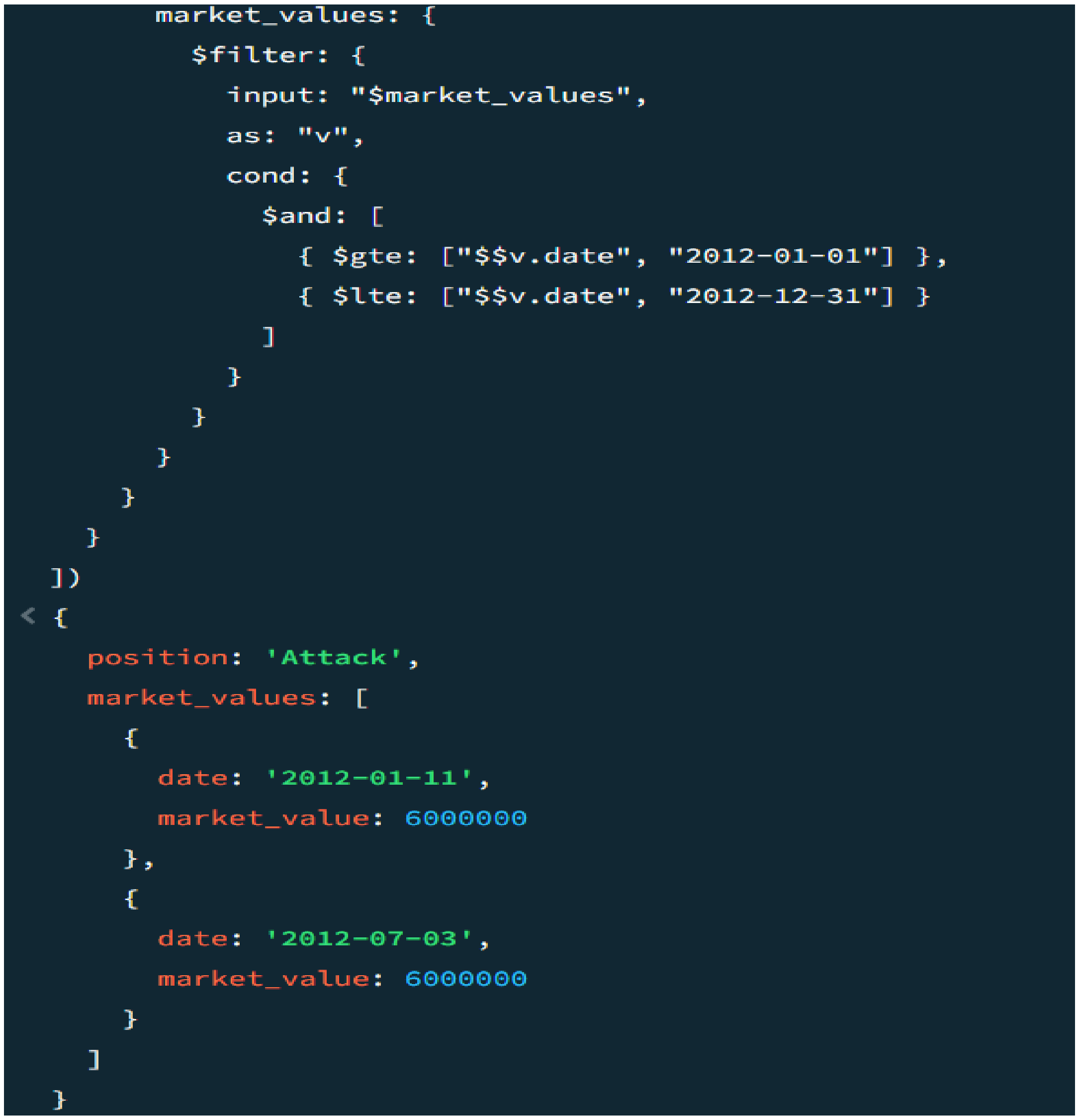
```
> db.club.find({ club_id: 2578 }, { _id: 0, players: {player_id: 1, position: 1, country_of_birth: 1}})
< {
    players: [
      {
        player_id: 2999,
        position: 'Attack',
        country_of_birth: 'Scotland'
      },
      {
        player_id: 3287,
        position: 'Goalkeeper',
        country_of_birth: 'England'
      },
      {
        player_id: 4251,
        position: 'Defender',
        country_of_birth: 'Scotland'
      },
      {
        player_id: 9506,
        position: 'Attack',
        country_of_birth: 'Scotland'
      },
      {
        player_id: 10025,
        position: 'Midfield',
        country_of_birth: 'Ireland'
      },
```

**Q4:** For a selected *competition* and *season*, retrieve all games, showing the date, round, and the stadium name and attendance.



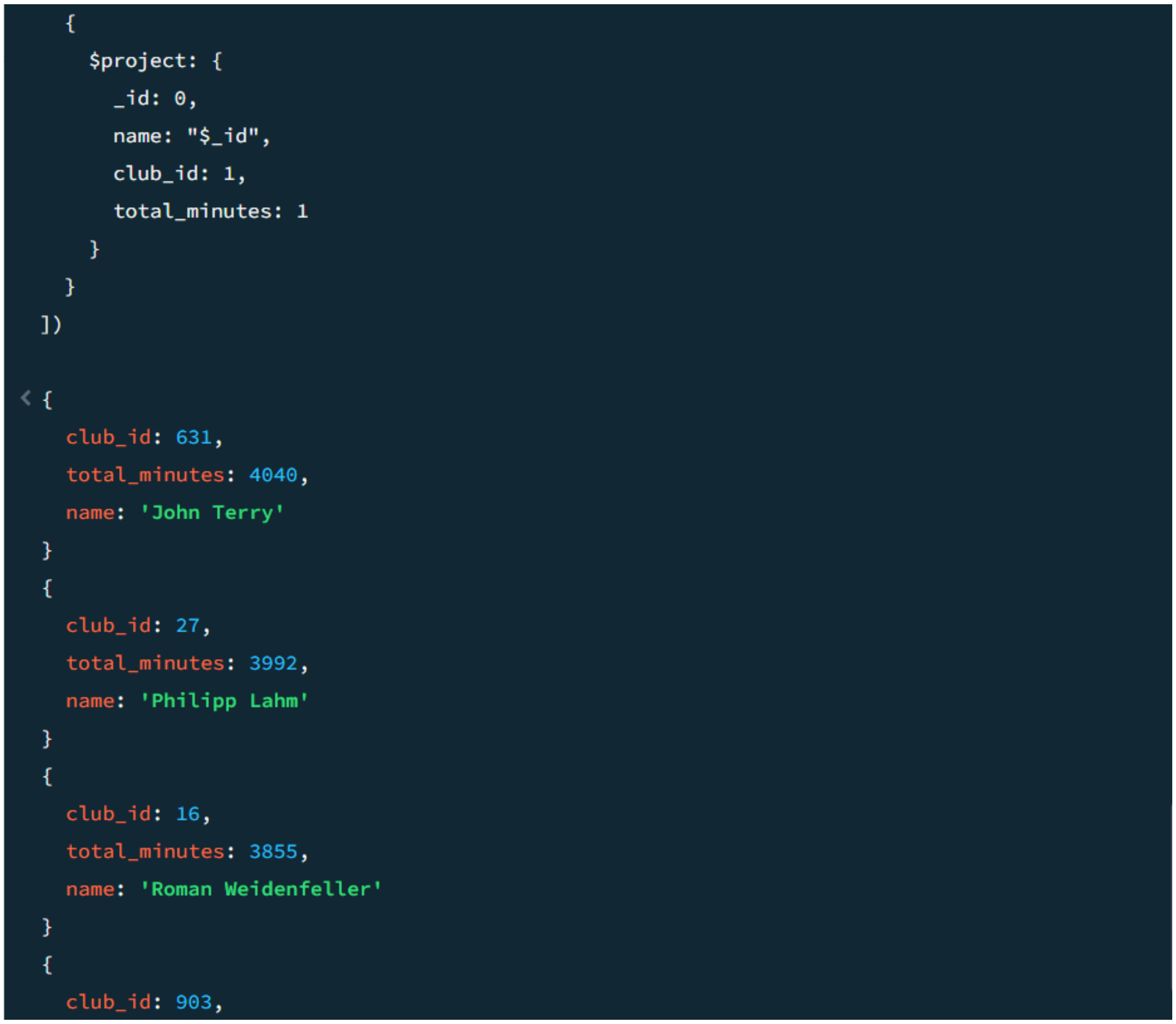
```
    date: 1,

    round: 1,

    stadium_name: 1,

    attendance: 1

  }

)

< {
    date: '2012-11-02',
    round: '10. Matchday',
    stadium_name: 'Commerzbank Arena',
    attendance: 47400
  }
  {
    date: '2012-11-17',
    round: '12. Matchday',
    stadium_name: 'Max-Morlock-Stadion',
    attendance: 50000
  }
  {
    date: '2012-10-20',
    round: '8. Matchday',
    stadium_name: 'Volkswagen Arena',
```

**Q5:** Retrieve the player position and valuation values for players with a given id, who have played at least one game in a given season (Minutes_played > 0) in a stadium with a given name, considering only valuation entries within a given date range.



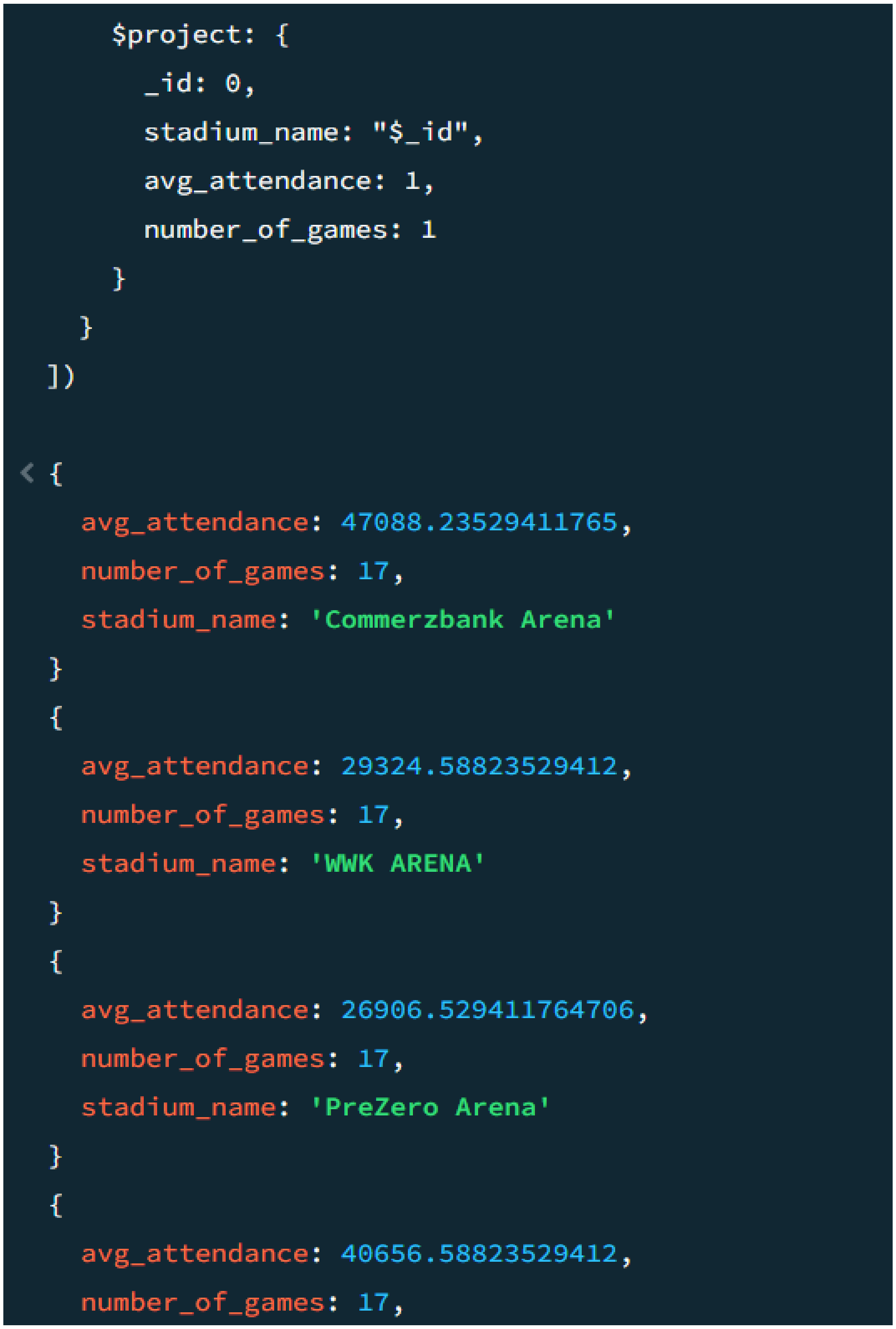
```
        market_values: {
          $filter: {
            input: "$market_values",
            as: "v",
            cond: {
              $and: [
                { $gte: ["$$v.date", "2012-01-01"] },
                { $lte: ["$$v.date", "2012-12-31"] }
              ]
            }
          }
        }
      }
    }
  ])
< {
    position: 'Attack',
    market_values: [
      {
        date: '2012-01-11',
        market_value: 6000000
      },
      {
        date: '2012-07-03',
        market_value: 6000000
      }
    ]
  }
```

**Q6:** For a given season, retrieve the top K players with the highest total minutes played, including player name and current club.



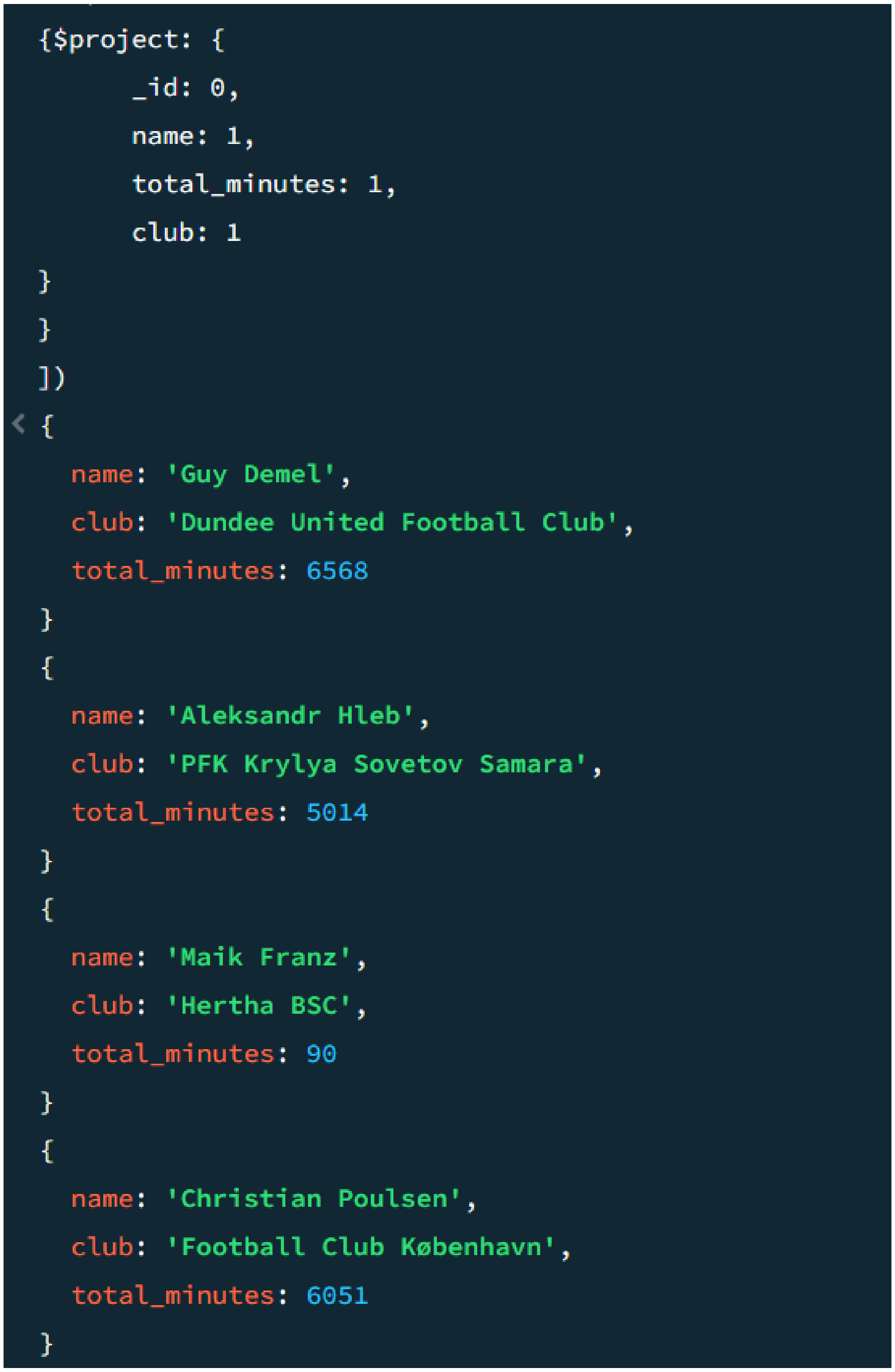
```
  {
    $project: {
      _id: 0,
      name: "$_id",
      club_id: 1,
      total_minutes: 1
    }
  }
])

< {
    club_id: 631,
    total_minutes: 4040,
    name: 'John Terry'
  }
  {
    club_id: 27,
    total_minutes: 3992,
    name: 'Philipp Lahm'
  }
  {
    club_id: 16,
    total_minutes: 3855,
    name: 'Roman Weidenfeller'
  }
  {
    club_id: 903,
```

**Q7:** For a given competition and season, compute for each stadium the average attendance and number of games hosted.



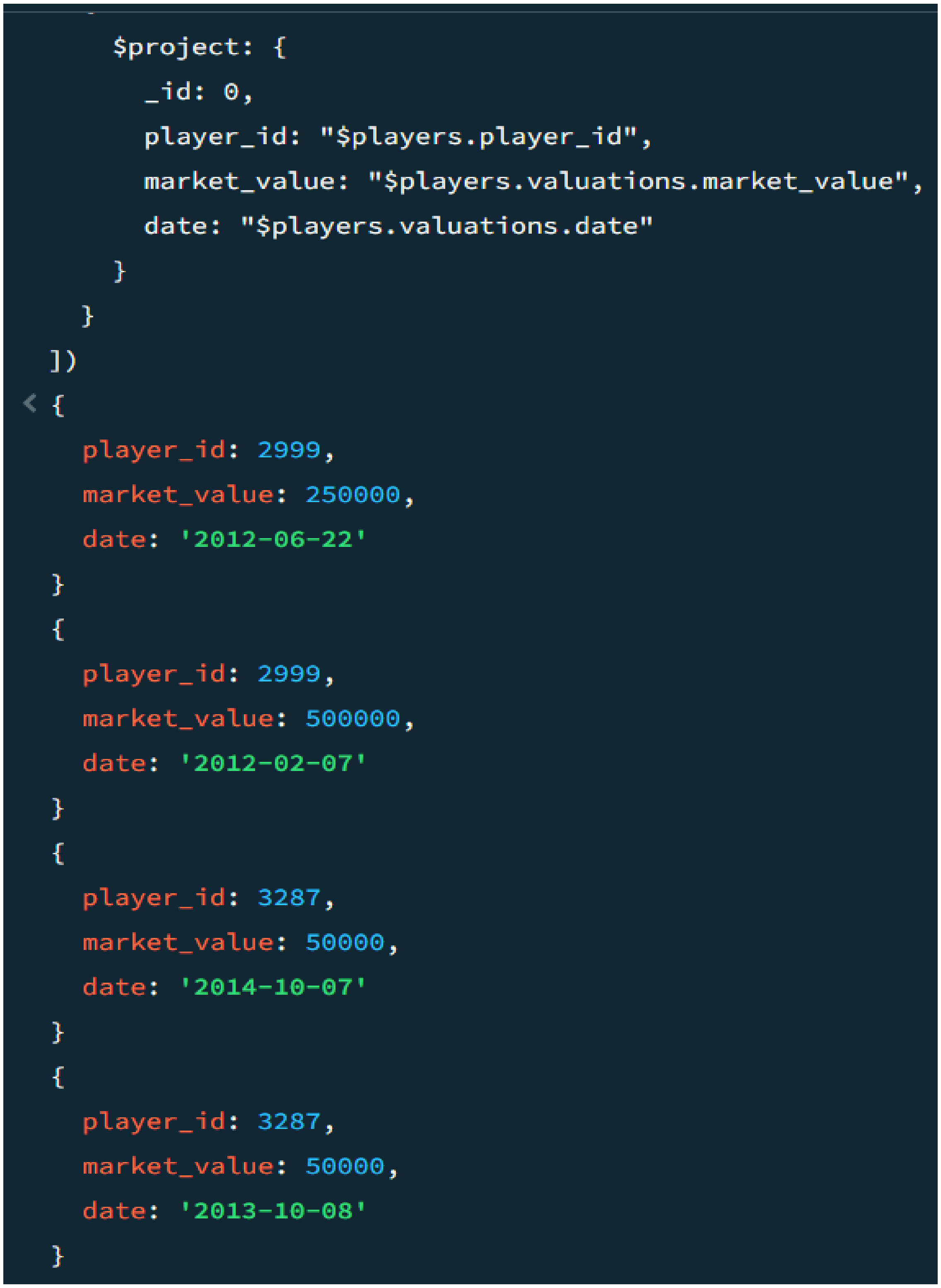
```
      $project: {
        _id: 0,
        stadium_name: "$_id",
        avg_attendance: 1,
        number_of_games: 1
      }
    }
  ])

< {
    avg_attendance: 47088.23529411765,
    number_of_games: 17,
    stadium_name: 'Commerzbank Arena'
  }
  {
    avg_attendance: 29324.58823529412,
    number_of_games: 17,
    stadium_name: 'WWK ARENA'
  }
  {
    avg_attendance: 26906.529411764706,
    number_of_games: 17,
    stadium_name: 'PreZero Arena'
  }
  {
    avg_attendance: 40656.58823529412,
    number_of_games: 17,
```

**Q8:** Retrieve the total minutes played by each player across all games, including player name and current club.



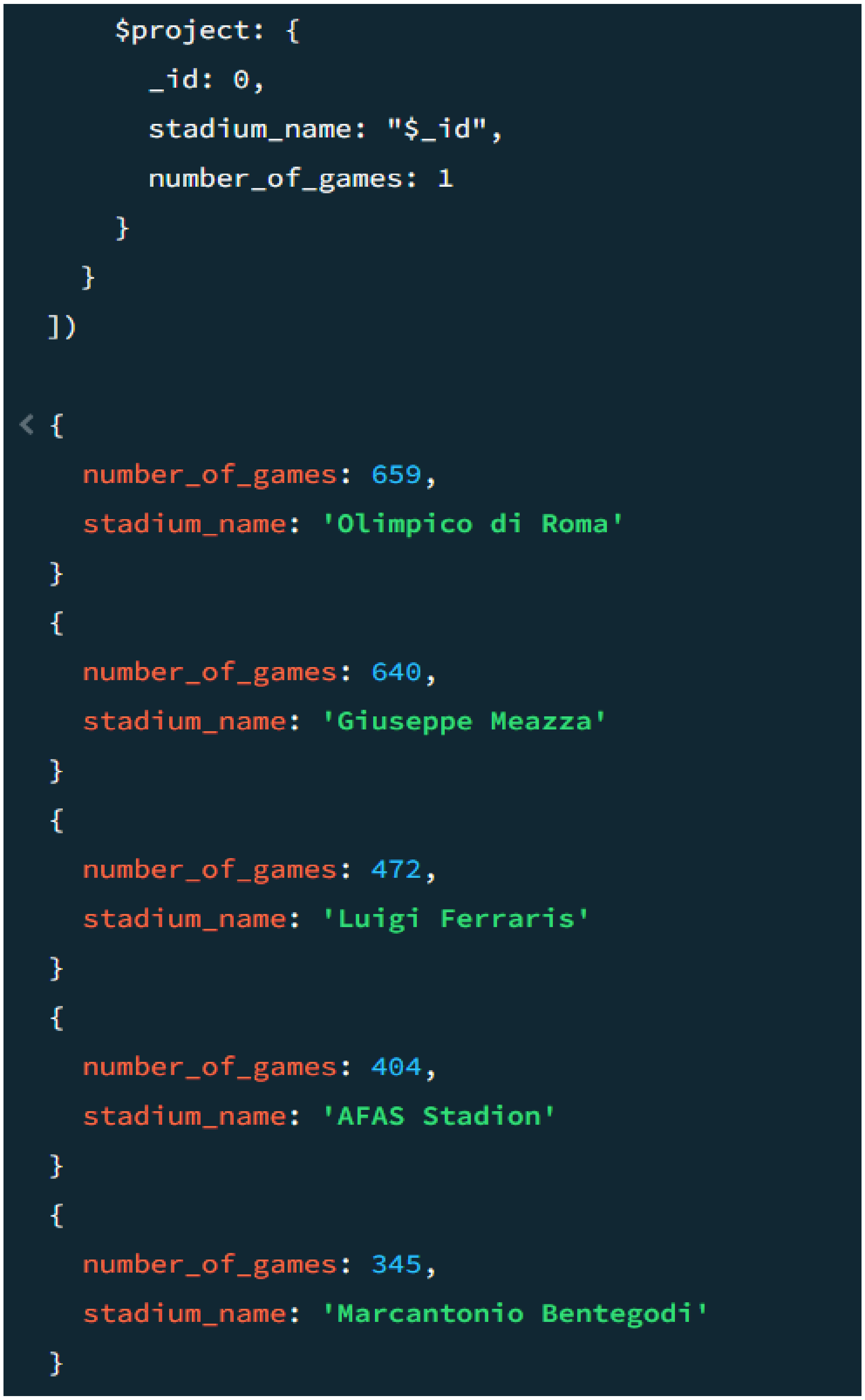
```
{$project: {
      _id: 0,
      name: 1,
      total_minutes: 1,
      club: 1
}
}
])
{
  name: 'Guy Demel',
  club: 'Dundee United Football Club',
  total_minutes: 6568
}
{
  name: 'Aleksandr Hleb',
  club: 'PFK Krylya Sovetov Samara',
  total_minutes: 5014
}
{
  name: 'Maik Franz',
  club: 'Hertha BSC',
  total_minutes: 90
}
{
  name: 'Christian Poulsen',
  club: 'Football Club København',
  total_minutes: 6051
}
```

**Q9:** Retrieve all valuation entries within a given date range for players currently playing for a given club.



```
      $project: {
        _id: 0,
        player_id: "$players.player_id",
        market_value: "$players.valuations.market_value",
        date: "$players.valuations.date"
      }
    }
  ])
< {
    player_id: 2999,
    market_value: 250000,
    date: '2012-06-22'
  }
  {
    player_id: 2999,
    market_value: 500000,
    date: '2012-02-07'
  }
  {
    player_id: 3287,
    market_value: 50000,
    date: '2014-10-07'
  }
  {
    player_id: 3287,
    market_value: 50000,
    date: '2013-10-08'
  }
```

**Q10:** Retrieve the total number of games played in each stadium, sorted in descending order of the number of games hosted.



```
      $project: {
        _id: 0,
        stadium_name: "$_id",
        number_of_games: 1
      }
    }
  ])

< {
    number_of_games: 659,
    stadium_name: 'Olimpico di Roma'
  }
  {
    number_of_games: 640,
    stadium_name: 'Giuseppe Meazza'
  }
  {
    number_of_games: 472,
    stadium_name: 'Luigi Ferraris'
  }
  {
    number_of_games: 404,
    stadium_name: 'AFAS Stadion'
  }
  {
    number_of_games: 345,
    stadium_name: 'Marcantonio Bentegodi'
  }
```

## Use of Generative AI

We used an LLM to review the written Workload, in order to check for possible textual or conceptual mistakes. Also, LLMs were used to support the installation of MongoDB Compass, the transformation of the real dataset into JSON according to the logical schema, and to validate some MongoDB and Cypher queries. All final decisions and implementations, technical and design aspects were fully developed by us and AI did not contribute to any design, modeling, or implementation aspects of the project.