Data Warehouses - Project report

Group A - Burgués Llavall Gerard, Filatova Violetta, Regulski Michał
Wrocław University of Science and Technology

February 5, 2021

Contents

1 Introduction **2**
1.1 Goal 2
1.2 Scope 2
1.3 Data set 2

2 Implementation **2**
2.1 Tools 2
2.2 Architecture 3

3 ETL process **4**

4 Data analysis **7**
4.1 Three pointers rank-up 7
4.2 Showing specific players 8
4.3 Showing specific players in division 9
4.4 Number of wins at home 10
4.5 Best free throwers 11
4.6 The best players in each position 12
4.7 Top 10 cities with the highest amount of points 13
4.8 Correlation between height of players and how many 3-pointers they made 14

5 Results **15**
5.1 Three pointers rank-up 15
5.2 Showing specific players 16
5.3 Showing specific players in division 17
5.4 Number of wins at home 18
5.5 Best free throwers 19
5.6 The best players in each position 20
5.7 Top 10 cities with the highest amount of points 21
5.8 Correlation between height of players and how many 3-pointers they made 22

6 Summary **23**

7 Appendix **23**

1 Introduction

1.1 Goal

Nowadays, Team Managers create their team based on statistics and data of the player from previous years. Our goal is to help team managers and coaches to make decisions for their teams. We will create some charts from specific data that may be interesting for the user.

1.2 Scope

The scope of this project involves parts as:

- Receiving and processing the data incoming from API
- ETL process
- Storing the data in OLAP database
- Data analysis
- Data visualization

1.3 Data set

The data set comes from the **balldontlie API**. It is a free API containing multiple statistics about NBA players and games from different seasons. The API is quite rich in data, it contains the data from seasons which are not available from official NBA API, but the main reason why we haven't used the official NBA API is that it is not officially documented, neither it behaves in an expected way. The data coming from endpoints of balldontlie API is formatted as JSON files. Unfortunately the biggest drawback of this API is a rate limit of requests - 60 requests/minute. To solve this problem, we decided to use appropriate set of programming libraries.

2 Implementation

2.1 Tools

The tools we have used for the implementation of the project are:

- RabbitMQ - Message broker as an asynchronous queue for orchestration of the ETL process.
- Greenplum - An OLAP database based on PostgreSQL allowing to process data in a parallel way. Compatibility with a PostgreSQL is a crucial feature allowing to connect to the database using reactive driver for JVM.
- Kotlin - A multi-platform language focused on the inseparability with Java and JVM. It's more concise and powerful comparing to Java.
- Spring Framework - A framework allowing to create complex applications using Dependency Injection. It doesn't force programmers to program in a single way, but it offers different approaches. There are multiple sub-projects extending the capabilities of the framework. The most significant libraries used alongside the Spring Core are:
  - Project Reactor - a default reactive library cooperating with Spring Framework
  - Reactor RabbitMQ - a reactive client for RabbitMQ which **should** respect backpressure.
  - WebClient (Spring Web Reactive) - a reactive HTTP client which is a part of reactive stack of the Spring Web.

  - PostgreSQL R2DBC - reactive driver for the PostgreSQL database.
  - Spring Data R2DBC - wrapper on the driver allowing to create concise and safe SQL code using fluent API
- PowerBI and Tableau - Applications for the data analysis and data visualization

2.2 Architecture

The figure 1 represents the architecture of the project. The two integral parts are:

- The ETL processor application - The coordinator of the ETL processes. It manage the problem of API rate limits by pushing and pulling messages to the RabbitMQ. It deals with the backpressure for the messages on the RabbitMQ, backpressure of the HTTP client for communication with the balldontlie API and the process of storing the data inside the Greenplum database.
- Greenplum database - stores the data fetched from the balldontlie API and allows to process the data in an efficient way. We also use the database with external tools: PowerBI and Tableau.

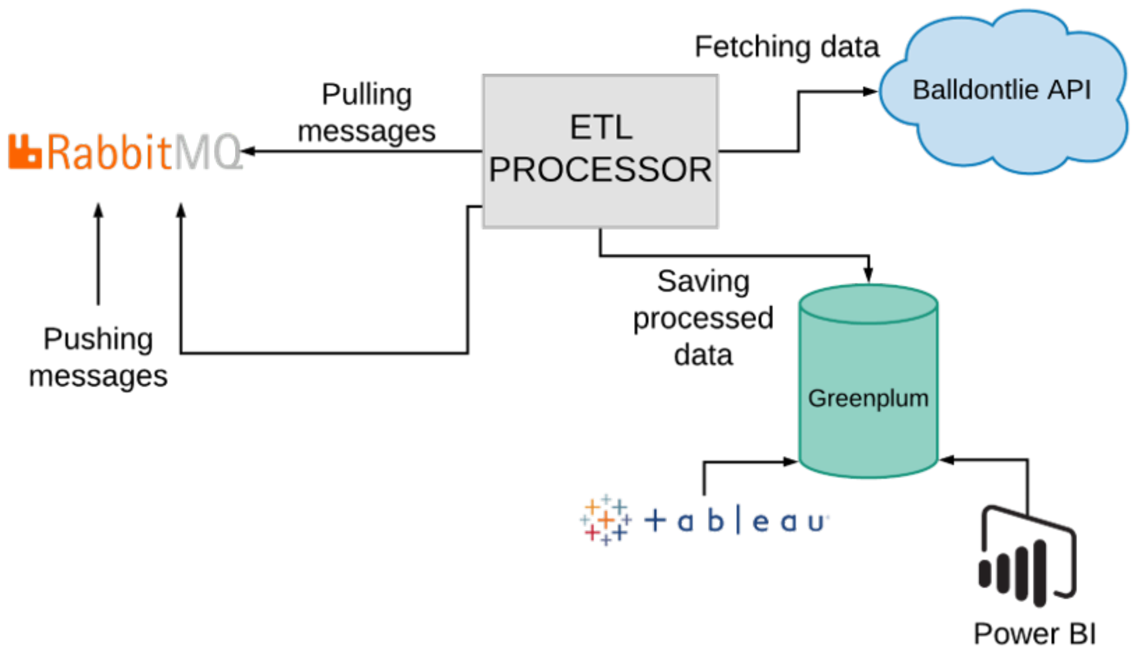


Figure 1: The diagram of the architecture of the application made during the project

The ETL approach we used in a project is a Kappa architecture, which supports near real-time analytic when the data is fetched from the API and transformed immediately.

3 ETL process

The whole ETL process for every possible entity is described in an abstract class AbstractETLProcessor. This allows to prepare comparably simpler entity-specific code. The method responsible for that is presented in the Listing 1.

```kotlin
private fun process(
    queue: Queue,
    toMessage: (AcknowledgableDelivery) -> Mono<T1>,
    extract: (T1) -> Mono<T2>,
    transform: (T2) -> Mono<Pair<List<T3>, Boolean>>,
    load: (Pair<List<T3>, Boolean>) -> Mono<Boolean>,
): Flux<*> {
    val flux1 = getEtlStatus()
        .filter { !it.done }
        .flatMap { sendMessages(prepareInitialMessages()) }

    val flux2 = rabbitReceiver.consumeManualAck(
        queue.queueName, ConsumeOptions()
            .qos(1)
            .exceptionHandler(
                ExceptionHandlers.RetryAcknowledgmentExceptionHandler(
                    20.seconds.toJavaDuration(),
                    500.milliseconds.toJavaDuration(),
                    ExceptionHandlers.CONNECTION_RECOVERY_PREDICATE
                )
            )
    )
        .delayElements(1.minutes.toJavaDuration())
        .flatMap { delivery ->
            toMessage(delivery)
                .flatMap { message ->
                    extract(message)
                        .retryExponentialBackoff(
                            times = 3,
                            first = 10.seconds.toJavaDuration(),
                            max = 1.minutes.toJavaDuration(),
                            jitter = true
                        )
                }
                .flatMap { data -> transform(data) }
                .flatMap { data -> load(data) }
                .flatMap { done ->
                    if (done) {
                        saveEtlStatus()
                    } else {
                        Mono.just(1)
                    }
                }
                .doFinally {
                    if (it == SignalType.ON_COMPLETE) {
                        delivery.ack()
                    } else {
                        delivery.nack(true)
                    }
                }
        }

    return flux1.thenMany(flux2)
}
```

Listing 1: The method responsible for an abstraction of ETL process

The method can be considered as higher-order function, because it requires passing 4 functions transforming the data from one step to another fulfilling contract. flatmap() function allows to perform mapping from one type to another in an asynchronous manner. AbstractETLProcessor additionally forces type-safety and helps following what the contracts should be fulfilled. The implementation of this class is pretty simple and short. In the Listing 2 one of the implementations has been shown.

```kotlin
@Service
class PlayersETLProcessor(
    private val playersClient: PlayersClient, //other beans skipped
) : AbstractETLProcessor<PageMessage, PlayersWrapper, Player>(/*some beans*/) {

    override val queue = PLAYERS // defines from which queue messages are pulled of

    override val tableName: String = PLAYERS_TABLE //defines the output table in the database

    /**
     * Method for data fetching from API.
     */
    override fun extract(message: PageMessage): Mono<PlayersWrapper> = with(message) {
        playersClient.getPlayers(page, perPage)
    }

    /**
     * Method for data transformation from the extraction step to the form acceptable for the
     * data warehouse. The second element in the tuple keeps tracking if this is the last
     * batch of the data for given entity.
     */
    override fun transform(data: PlayersWrapper): Mono<Pair<List<Player>, Boolean>> {
        val sendMessages = if (data.meta.currentPage == 1) {
            Flux.fromIterable(1 until data.meta.totalPages)
                .flatMap { page -> sendMessages(PageMessage(page = page + 1).toMono()) }
                .then(data.toMono())
        } else {
            data.toMono()
        }

        return sendMessages.map { playersWrapper ->
            val isLastPage = playersWrapper.meta.currentPage == playersWrapper.meta.totalPages
            val players = playersWrapper.data.map { player ->
                with(player) {
                    Player(
                        id = id,
                        firstName = firstName,
                        lastName = lastName,
                        position = position,
                        heightFeet = heightFeet,
                        heightInches = heightInches,
                        weightPounds = weightPounds,
                        teamId = team.id
                    )
                }
            }

            players to isLastPage
        }
    }

    /**
     * Method for inserting the data into the data warehouse.
     */
    override fun load(data: Pair<List<Player>, Boolean>): Mono<Boolean> = Flux.fromIterable(data.first)
```

```
        .filterWhen { player ->
            r2dbcEntityTemplate.select<Game>()
                .matching(Query.query(Criteria.where("id").isEqual(player.id)))
                .exists()
                .not()
        }
        .flatMap { player ->
            r2dbcEntityTemplate.insert<Player>().using(player).onErrorContinue { e, _ -> }
        }
        .then(data.second.toMono())

    /**
     * Method for feeding RabbitMQ with an initial message.
     */
    override fun prepareInitialMessages(): Publisher<PageMessage> = PageMessage().toMono()

}
```

Listing 2: The implementation of the AbstractETLProcessor for Player entity.

4 Data analysis

4.1 Three pointers rank-up

Based on the player percentage as an input, the query will show all those players with a higher percentage from the 3-point line. This information will be useful for the user (team manager) in case he/she needs a new 3-point shooter for the team.

```
CREATE FUNCTION show_best_3_pt(a_percentage DOUBLE PRECISION)
    RETURNS TABLE
    (
        season                   INTEGER,
        minutes                  TEXT,
        three_pointer_percentage DOUBLE PRECISION,
        first_name               TEXT,
        last_name                TEXT,
        team_name                TEXT,
        three_pointer_attempted  DOUBLE PRECISION,
        three_pointer_made       DOUBLE PRECISION
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT averages.season,
               averages.minutes,
               averages.three_pointer_percentage,
               p.first_name,
               p.last_name,
               t."name",
               averages.three_pointers_attempted,
               averages.three_pointers_made
        FROM averages
            JOIN players p ON averages.player_id = p.id
            JOIN teams t ON p.team_id = t.id
        WHERE averages.three_pointer_percentage >= a_percentage
        GROUP BY averages.season, p.first_name, averages.minutes,
            averages.three_pointer_percentage, p.last_name, t.name,
            averages.three_pointers_attempted, averages.three_pointers_made
        ORDER BY averages.three_pointer_percentage;
END;
$$ LANGUAGE PLPGSQL;
```

Listing 3: Three-pointers rank-up query

4.2 Showing specific players

In this case we will show to the user all those players with more than X points and Y assists (X and Y as parameters).

```
CREATE FUNCTION show_specific_players(s_points INTEGER, s_assists INTEGER)
    RETURNS TABLE
    (
        v_game_id            BIGINT,
        first_name           TEXT,
        last_name            TEXT,
        v_points             INTEGER,
        v_assists            INTEGER,
        v_season             INTEGER,
        v_home_team_score    INTEGER,
        v_visitor_team_score INTEGER,
        home_team_name       TEXT,
        visitor_team_name    TEXT,
        v_winner_team_id     BIGINT,
        v_home_team_id       BIGINT,
        v_visitor_team_id    BIGINT
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT game_id,
               stats.first_name,
               stats.last_name,
               points,
               assists,
               season,
               home_team_score,
               visitor_team_score,
               t1.name,
               t2.name,
               winner_team_id,
               home_team_id,
               visitor_team_id
        FROM stats
            JOIN teams t1 ON t1.id = stats.home_team_id
            JOIN teams t2 ON t2.id = stats.visitor_team_id
        WHERE points >= s_points
          AND assists >= s_assists
          AND winner_team_id = team_id
        ORDER BY points;
END;
$$ LANGUAGE PLPGSQL;
```

Listing 4: Showing specific players query

4.3 Showing specific players in division

We will want to show what players have the most affect in each division. Setting points and assists as parameters we will show what players have more than those values (average) and show in which division they play and what season it was.

```
CREATE FUNCTION sort_by_division(s_points INTEGER, s_assists INTEGER)
    RETURNS TABLE
    (
        points     DOUBLE PRECISION,
        assists    DOUBLE PRECISION,
        first_name TEXT,
        last_name  TEXT,
        division   TEXT
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT averages.points, averages.assists, p.first_name, p.last_name, t.division
        FROM averages
                 JOIN players p ON averages.player_id = p.id
                 JOIN teams t ON p.team_id = t.id
        WHERE averages.points >= (s_points)
          AND averages.assists >= (s_assists)
        ORDER BY t.division;
END;
$$ LANGUAGE PLPGSQL;
```

Listing 5: A query finding specific players in division

4.4 Number of wins at home

We will show to the team manager how many times each team wins at home during specific months (parameter), for example, the last 4 months of the season. This will be useful to know if the team improve or relax at a specific moment of the season.

```
CREATE FUNCTION how_a_team_changes(date1 DATE, date2 DATE)
    RETURNS TABLE
    (
        team_full_name TEXT,
        season         INTEGER,
        date11         DATE,
        date12         DATE,
        number         BIGINT
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT t.name,
               date1,
               date2,
               COUNT(*)
        FROM games
                 JOIN stats s ON games.id = s.game_id
                 JOIN teams t ON t.id = s.home_team_id
        WHERE games.date >= date1
          AND games.date <= date2
          AND games.home_team_id = games.winner_team_id
        GROUP BY games.season, t.name, games.home_team_id, games.winner_team_id;
END;
$$ LANGUAGE PLPGSQL;
```

Listing 6: A query showing how many times a team won at home

4.5 Best free throwers

Given minutes and position as parameters we want to show the percentage for those players who play in a specific position and for more than 'X' minutes. The user will use this information for multiple things. For example, to have at least one player who can make easily free throws at the end of the game.

```
CREATE FUNCTION center_player(s_minutes TEXT, s_position TEXT)
    RETURNS TABLE
    (
        minutes               TEXT,
        first_name            TEXT,
        last_name             TEXT,
        v_position            TEXT,
        free_throw_percentage DOUBLE PRECISION
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT averages.minutes,
               p.first_name,
               p.last_name,
               p.position,
               averages.free_throw_percentage
        FROM averages
                 JOIN players p ON p.id = averages.player_id
        WHERE averages.minutes < (s_minutes)
          AND p.position = s_position
        ORDER BY averages.free_throw_percentage;

END
$$ LANGUAGE PLPGSQL;
```

Listing 7: A query showing best free throwers

4.6 The best players in each position

This query illustrates information about the best player for each position, based on the data on the points scored by him. This gives a picture of the general state of the players.

```
CREATE OR REPLACE FUNCTION best_player()
    RETURNS TABLE
    (
        player_id  BIGINT,
        first_name TEXT,
        last_name  TEXT,
        position   TEXT,
        points     DOUBLE PRECISION
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT DISTINCT ON (p.position) (p.first_name || ' ' || p.last_name) AS "full_name",
                                        p.position,
                                        s.points
        FROM stats AS s
                 JOIN players AS p ON s.player_id = p.id
        GROUP BY (p.first_name || ' ' || p.last_name), p.position, s.points
        ORDER BY p.position, s.points DESC;
END;
$$ LANGUAGE PLPGSQL;
```

Listing 8: A query finding the best player in each position

4.7 Top 10 cities with the highest amount of points

The purpose of this query is to show which cities the teams with the most points for all time belong to, and based on this data, select the top 10 cities.

```
CREATE OR REPLACE FUNCTION top_cities()
    RETURNS TABLE
    (
        team_id BIGINT,
        city    TEXT,
        points  INTEGER
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT s.team_id, t.city, SUM(s.points)
        FROM stats AS s
                 JOIN teams t ON s.team_id = t.id
        GROUP BY s.team_id, s.points, t.city
        ORDER BY s.points DESC
        LIMIT 10;
END;
$$ LANGUAGE PLPGSQL;
```

Listing 9: A query finding the top 10 cities with the highest amount of points

4.8 Correlation between height of players and how many 3-pointers they made

Based on this information, you can learn about the importance of the player's physical parameters, establish a relationship with his performance and playing position.

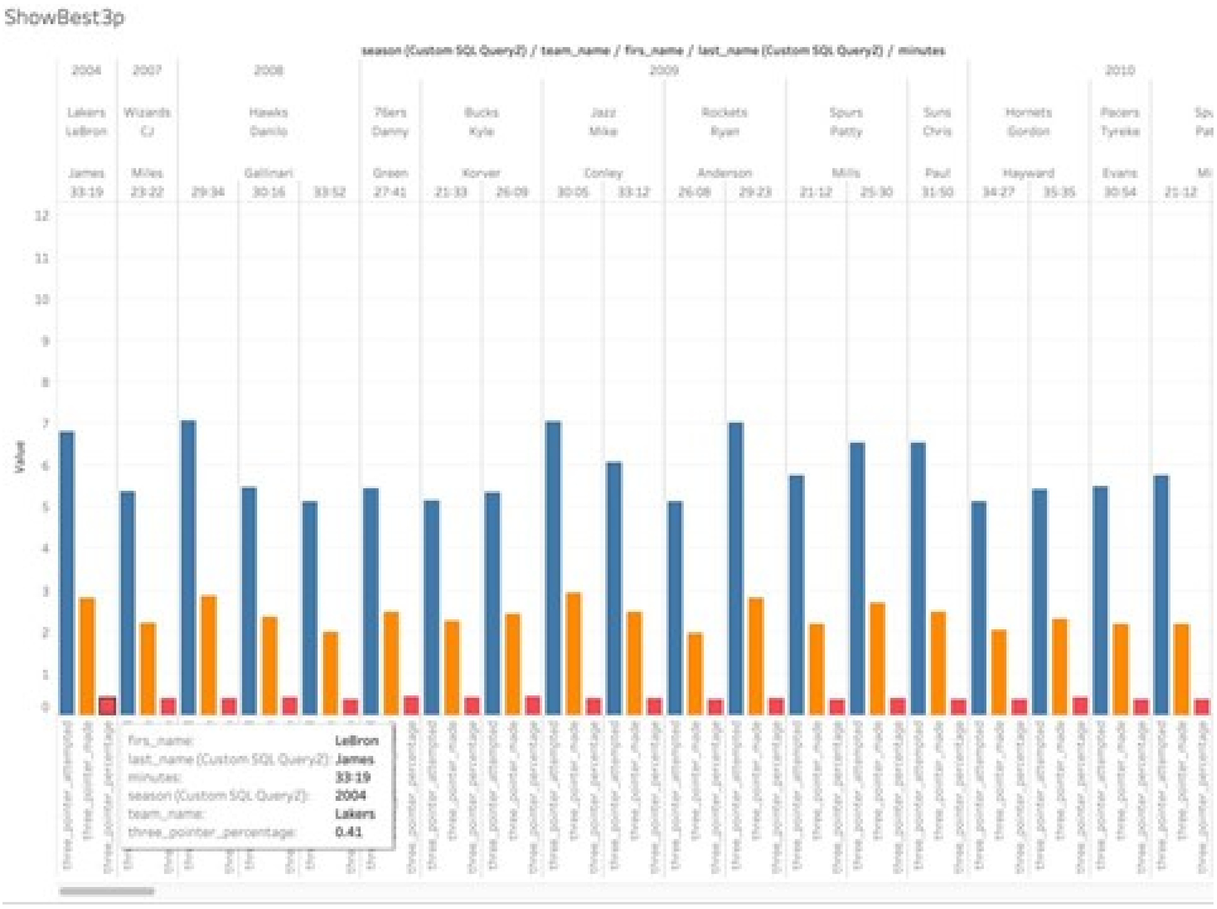
```
CREATE OR REPLACE FUNCTION
    corr_height_player()
    RETURNS TABLE
    (
        first_name          TEXT,
        last_name           TEXT,
        height_inches       INTEGER,
        three_pointers_made DOUBLE PRECISION,
        position            TEXT
    )
AS
$$
BEGIN
    RETURN QUERY
        SELECT p.first_name,
               p.last_name,
               p.height_inches,
               s.three_pointers_made,
               p.position
        FROM stats AS s
                 JOIN players AS p ON s.player_id = p.id
        WHERE p.height_inches > 0
        GROUP BY p.first_name,
                 p.last_name, p.height_inches, p.position, p.height_inches, s.three_pointers_made
        ORDER BY p.height_inches, s.three_pointers_made;
END;
$$ LANGUAGE PLPGSQL;
```

Listing 10: A query finding the correlation between height of players and how many 3-pointers they make

5 Results

5.1 Three pointers rank-up

As it can be seen in the next bar chart, we have chosen those players with more than 38% of effectiveness (parameter). But it is important to know how many average attempts they have made. The user does not want a player who attempted 1 three pointer and made it. The Team Manager wants a player who attempted several times and have a good average of three points made.



firs_name: LeBron
last_name (Custom SQL Query2): James
minutes: 33:19
season (Custom SQL Query2): 2004
team_name: Lakers
three_pointer_percentage: 0.41

Figure 2: Visualization of results of “Three-pointers rank-up” query

5.2 Showing specific players

As we observe in the image below, we are showing all those players who have more than 30 points and 12 assists in a specific game. This may help the team manager once his team wants to beat these players team.

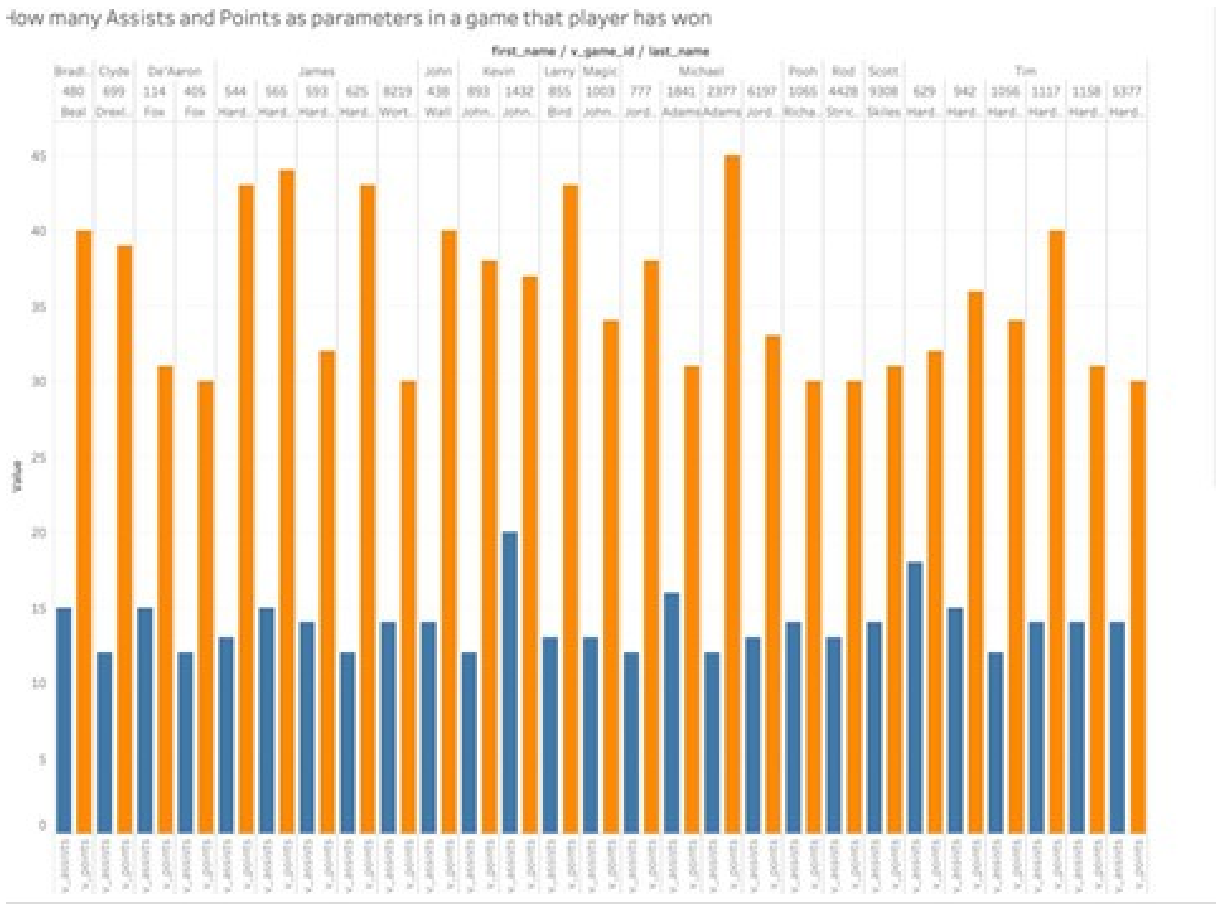


Figure 3: Visualization of results of "Show specific players" query

5.3 Showing specific players in division

I the next bar chart we can see that there are not many players who average more than 25 points and 9 assists in the league. This is not just to show the best scorers in the league but also to find Average players e.g., 10 points and 5 assists. The team manager will have a full perspective of each division.

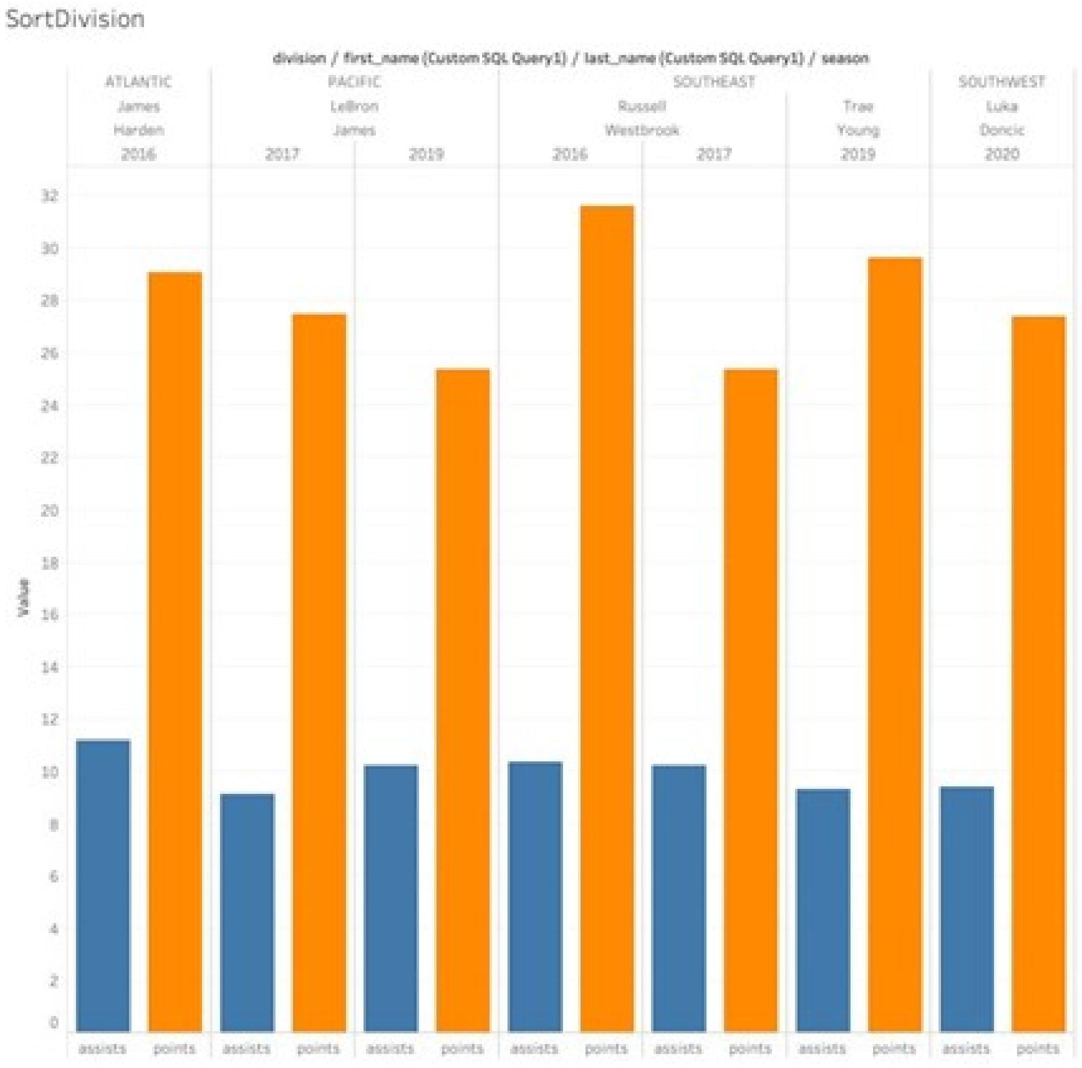
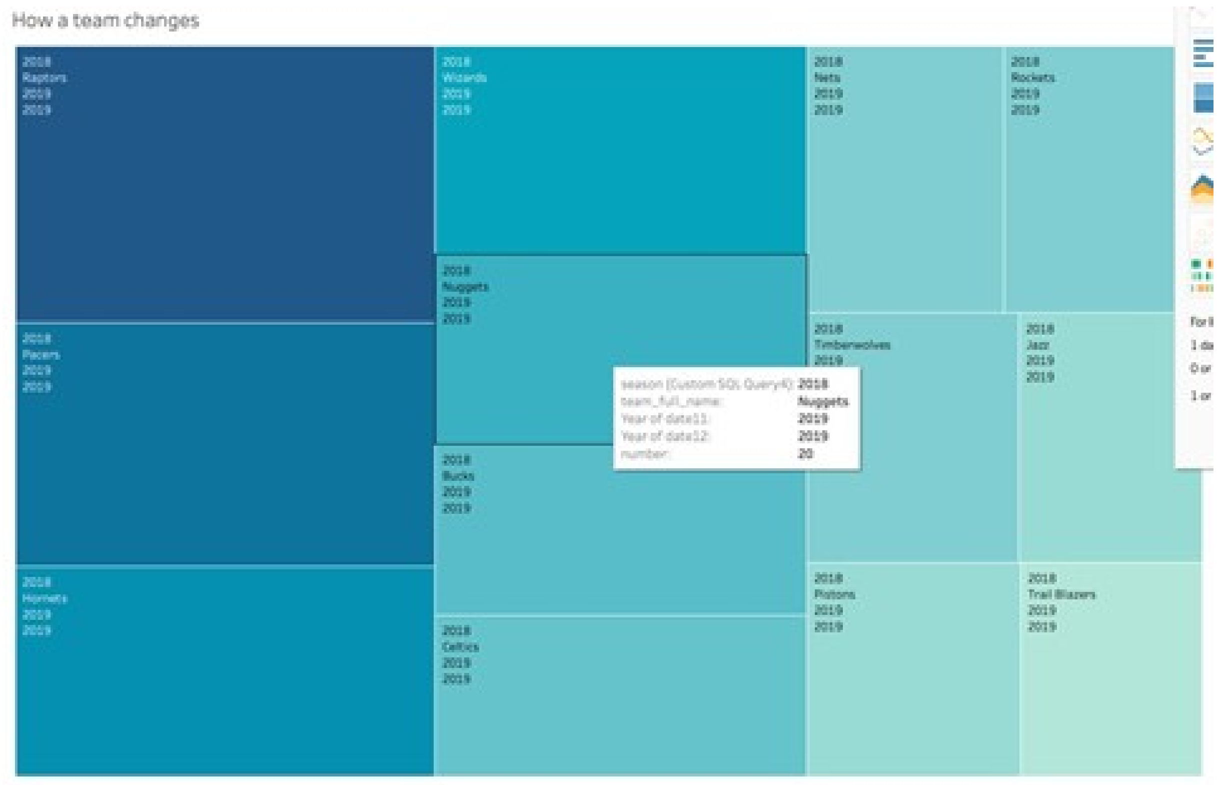


Figure 4: Visualization of results of “Showing specific players in division” query

5.4 Number of wins at home

Below we observe how many games a team has won at home during 15-01-2019 and 15-06-2019. Also is important to show what season it was and the user can check for different seasons and see the improvement of the team for the same dates.



season (Custom SQL Query4): 2018
team_full_name: Nuggets
Year of date11: 2019
Year of date12: 2019
number: 20

Figure 5: Visualization of results of a query showing how many times a team won at home

5.5 Best free throwers

In the following case we choose those players who play more than 30' in the Center position. As we can see, there are some players who have high percentage during some seasons.

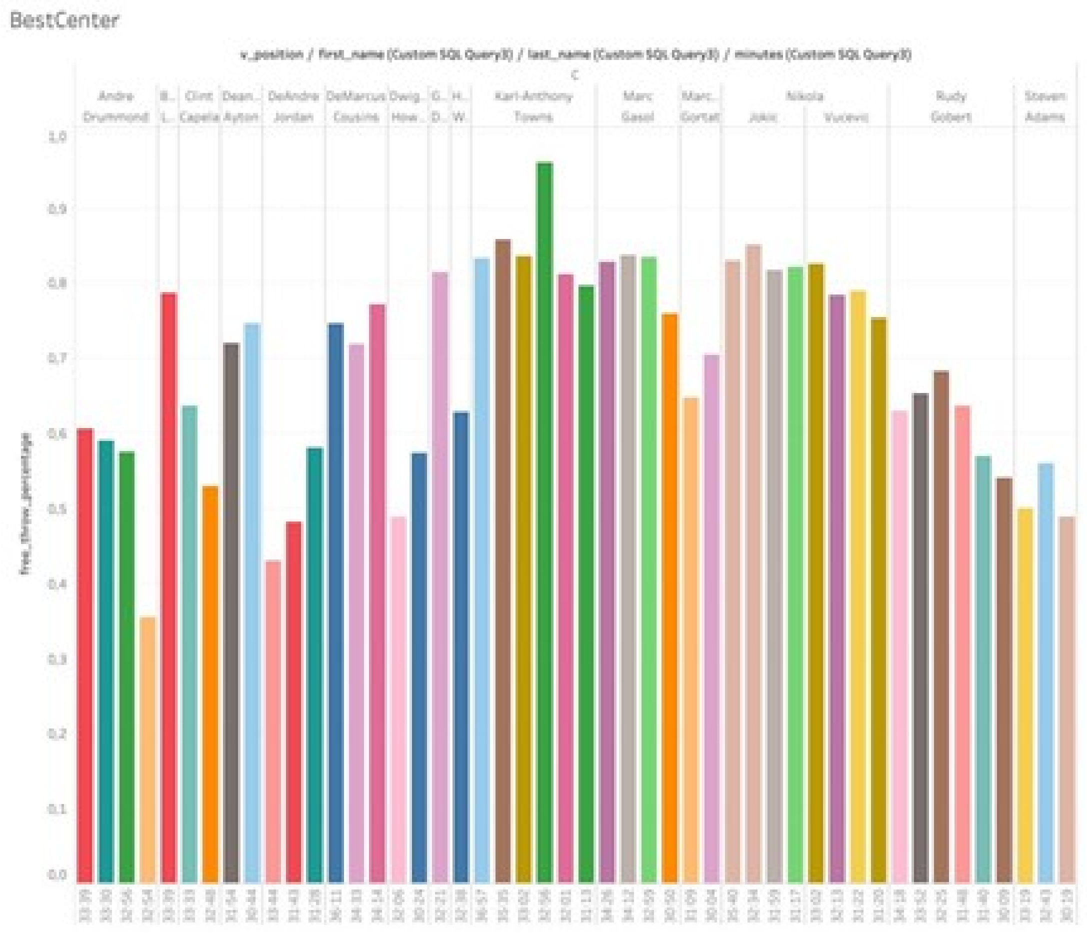


Figure 6: Visualization of results of a query showing best free throwers

5.6 The best players in each position

This information can be useful in researching players' careers, relationships with the team they play, and other related parameters. The graph displays the player's full name. Also, we can conclude on what position the most successful player is playing.

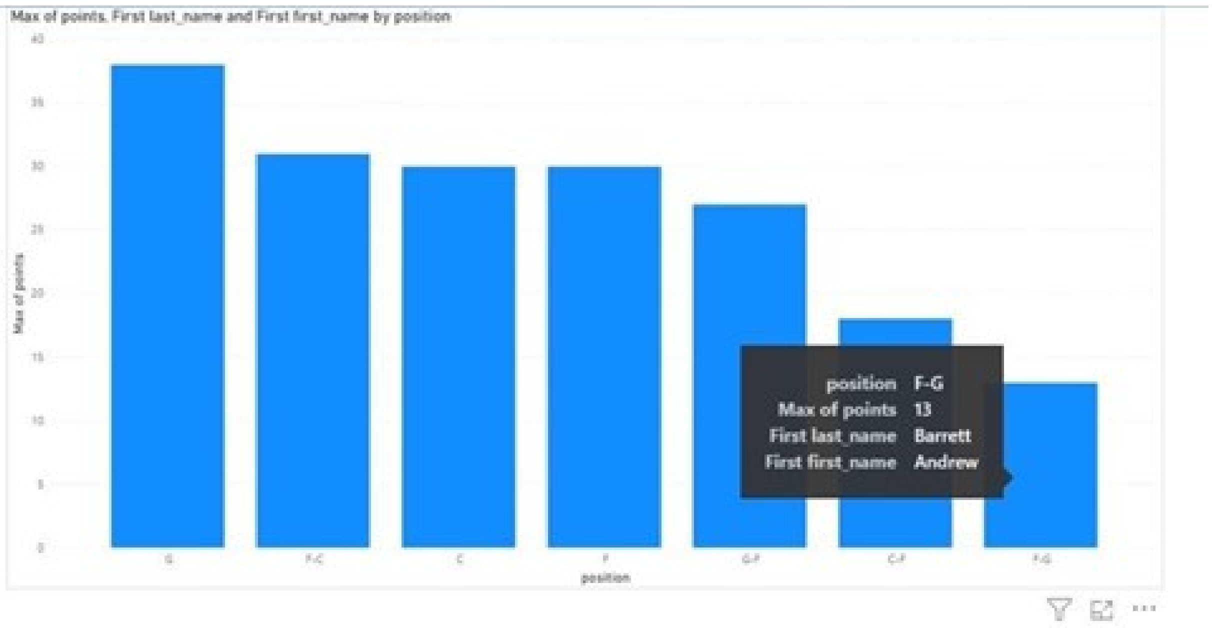


Figure 7: Visualization of results of a query showing the best players in each position

5.7 Top 10 cities with the highest amount of points

The top 10 cities are marked on the map. The user can use this information, for example, when he needs to establish the geography of teams with the best training. Also, it can be useful for comparison when calculating statistics of home and away games, and whether the scale of the city affects the success of the team.

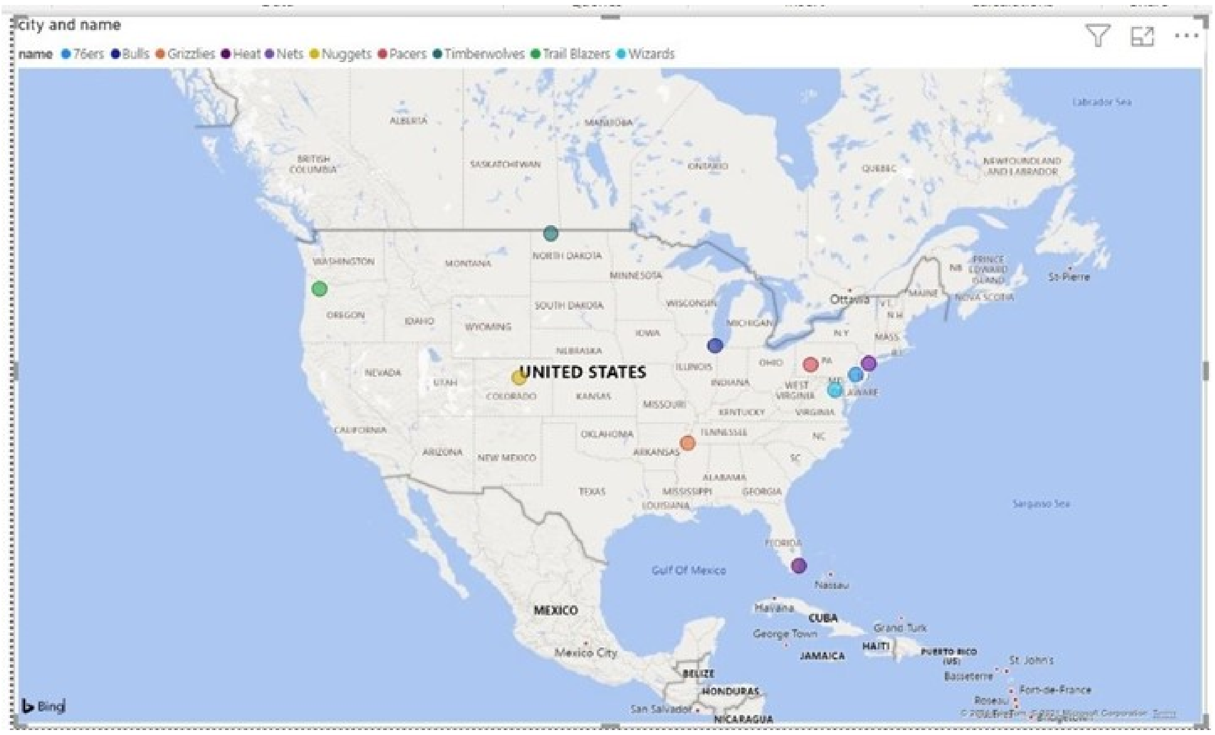

city and name
name ● 76ers ● Bulls ● Grizzlies ● Heat ● Nets ● Nuggets ● Pacers ● Timberwolves ● Trail Blazers ● Wizards
UNITED STATES
MEXICO
Gulf Of Mexico
Bing

Figure 8: Visualization of results of a query finding top 10 cities by the amount of points

5.8 Correlation between height of players and how many 3-pointers they made

The graph shows the number of three-pointers taken by various players and their height; This shows that the higher the player is, the more often he makes a three-pointer, although the position remains the main factor.

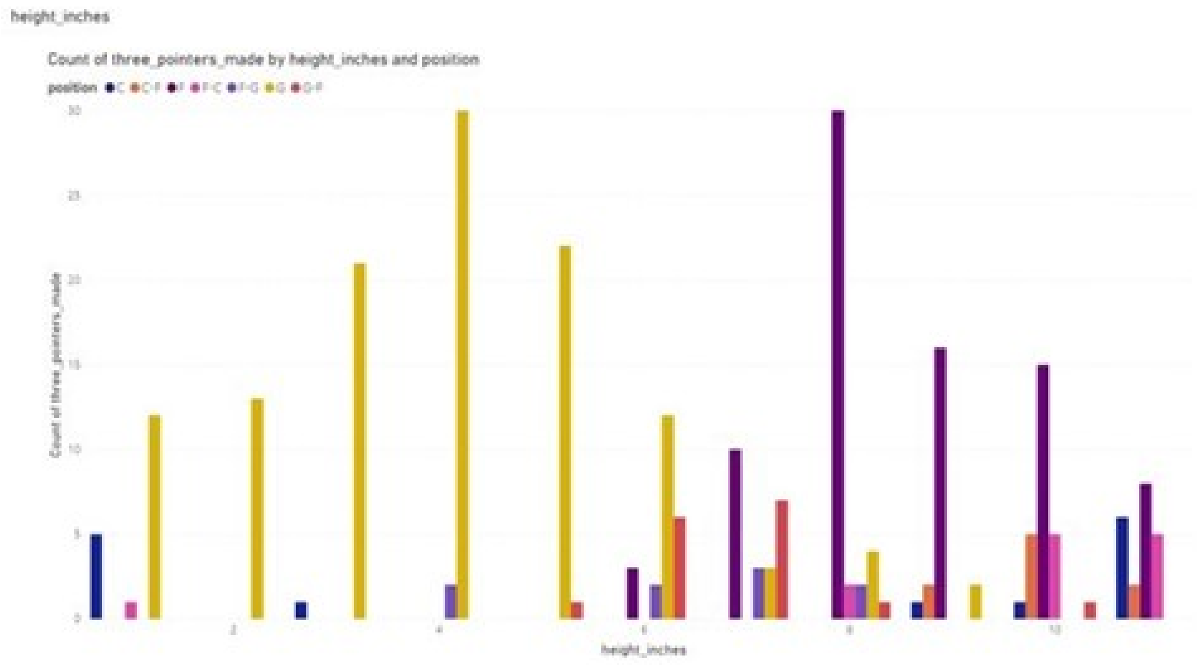


Figure 9: Visualization of the correlation between height of players and how many 3-pointers they made

6 Summary

The project is dedicated to the statistics of basketball games and makes it possible to analyze its various factors. In our opinion the project goals and scope has been accomplished. Unfortunately we have faced some issues with the reactive programming libraries, which have been a problem to use, because since they should react on the backpressure, but they seemed to not work properly, we could not find an easy way of fixing the problem of the API rate limits. We have tested out two approaches: Project Reactor + Kotlin Coroutines and pure Project Reactor code. In both cases the ETL processor application was overwhelmed by the messages from the RabbitMQ, which should be throttled by the speed of API. This is a very important part of the application which can be improved.
Basically, all the applied tools coped with their purpose. The process of writing queries to the database was carried out successfully and the user has the opportunity to receive the information he needs without redundancy. Based on the received data from the queries, the user can create graphs or visualize information in another way, and this data is enough to draw a conclusion or find a pattern.

7 Appendix

List of Listings

1 The method responsible for an abstraction of ETL process 4
2 The implementation of the AbstractETLProcessor for Player entity. 6
3 Three-pointers rank-up query 7
4 Showing specific players query 8
5 A query finding specific players in division 9
6 A query showing how many times a team won at home 10
7 A query showing best free throwers 11
8 A query finding the best player in each position 12
9 A query finding the top 10 cities with the highest amount of points 13
10 A query finding the correlation between height of players and how many 3-pointers they make 14

List of Figures

1 The diagram of the architecture of the application made during the project . . . 3
2 Visualization of results of “Three-pointers rank-up” query 15
3 Visualization of results of “Show specific players” query 16
4 Visualization of results of “Showing specific players in division” query 17
5 Visualization of results of a query showing how many times a team won at home 18
6 Visualization of results of a query showing best free throwers 19
7 Visualization of results of a query showing the best players in each position . . . 20
8 Visualization of results of a query finding top 10 cities by the amount of points . 21
9 Visualization of the correlation between height of players and how many 3-pointers they made 22