NBA Interactive Information Platform

Yizheng Wang
Zijing Wu
Yulin Yu
Zihao Zhang

Introduction

The National Basketball Association (NBA) is a professional basketball league in North America. The league is composed of 30 teams (29 in the United States and 1 in Canada) and hosts 82 games every season that lasts for a year. It is the premier men's professional basketball league in the world. In recent years, the data collection and analysis on NBA games, teams and players performance were more and more popular with the advanced technologies that could record every detail in each game. Therefore, we decided to bring our enthusiasm in basketball and NBA to the database and information system field to present this application.

The application was motivated by two perspectives. On one hand, for those who only watched one or two NBA games or only recognized Kobe Bryant, this is a great tool to get familiar with the NBA. For example, on our Home page, we presented a map that shows all NBA teams with their team names and logos on a map, splitting by eastern and western conference. In addition, we offered our personal recommendation on the best game of last season so that people could first have a quick glance at the statistics of this match and then jump into a drastic and exciting game with the game highlight. On the other hand, we also considered the common yet endless discussion among basketball fans. Not only showing the players who have the top 10 average points per game in NBA history, we also took average assist per game and average rebound per game into consideration. For some fans who cared about team contributions more than personal achievements, they could also find their favorite players on the list of NBA All Time.

Architecture

Technologies used

In the pre-development phase, **Web crawler (Xpath)** and **Colab** were used for data cleansing and integration. With cleaned data, we designed the frontend using **Figma** and implemented UI using **React.js**. For the backend, we used **MySQL** to store data, populated the data on **AWS**

RDS, and manipulated SQL queries with **Datagrip**. **Node.js** was applied to build the server, and **Axios**, the promise based HTTP client, was used to get requests from users.

Web architecture

The features of the **Home** page are mostly covered in the introduction section. The map is interactive and the view will get more information about the team name, arena name, located city and state after clicking on the logo of the interested team. The logo on the game of the season statistics board is also animated to show the abbreviation of the team. In addition, the value of the specific bar will be displayed when the view hovers the mouse on the horizontal bars in the All Time To 10 Leaders charts to allow the data to be presented in a clear and efficient manner.

On all **Game**, **Player**, **Team** pages, we provided the search functionality with a predefined form. Each attribute in the form works as an "AND" to further filter down the result. Considering users who had no previous knowledge of the NBA team or player name, we used the "LIKE" operator in our SQL queries to enable partial match. In this way, users only need to enter "Lakers" or "Kobe" to get the corresponding results for Los Angeles Lakers and Kobe Bryant. Beyond the basic search functionality and table format output, we also created more advanced visualizations with complex query to provide deeper insights.

On the **Game** page, we showed a fun fact about the home and away team, such as the year of founded, historical matchup statistics, and most valuable player in team history, on top of the page to offer some extra information to view who is new to the NBA game. Below the detailed game statistics of home and away teams, there is a horizontal bar chart showing the performance history of both teams. On the **Player** page, in addition to basic information, such as position, height, weight, jersey number and etc, we showed the player's details in terms of point, assist, and rebound with a radar chart. Next to the chart, we calculated and presented the player's network based on his teammate and teammate's previous teammates. We also plotted the player's salary throughout his career together with the team average salary. The insight behind this line chart is that the player usually played a more important role when his salary is above the team average salary. On the **Team** page, we created a flow chart that shows incoming and outgoing players in the last transfer period for the selected team. The visualization will help viewers understand the player flow and determine whether the team would become more or less competitive this season. Lastly, a vertical barchart with an overlaid line chart shows the salary per win of the specific team over seasons. Such visualization displays the ratio of team salary over seasonal win, which could represent a successful season if salary per win is low because either inexpensive players outperformed to win more games than expected or expensive players justified their values by winning a lot of games as expected.

Data Source

The dataset we used is from Kaggle, which automatically updates newest statistics from NBA official website stats.nba.com. The links are shown in the Appendix B. However, to avoid errors, we decided to use an offline version of the data for our website instead of streaming; that is, we downloaded the data, did data cleaning and wrangling, and uploaded the data we needed to the Amazon RDS server.

There are 16 tables within the data source and mainly 5 topics of the tables: news, draft, games, teams and players. Because we tend to focus on developing an interface for users to examine some facts and relationships among players, team and games, we decided to drop the draft and news tables completely.

For the games section, the record starts from the NBA season 1946 - 47 and includes every season game since. Note that it only has regular season games and neither pre-season nor playoff games are included. Some of the most informative statistics include box scores, Officials, season series, game summaries, timestamp, etc. There are 3 tables in this section: Game, Game-inactive_Players, and Game_officials. The biggest table within this section is Game, which contains 62,379 unique values for game_id as primary key, and 149 columns/attributes.

For the team section, tables contain detailed information about each NBA team. This is a relatively smaller section because there are only 30 teams in the NBA, and general information includes team name, location, stadium name, head coach, general manager, etc. Note that unlike games with timestamps, the information inside the team table only reflects the current situation. For example, there is only the name of the current owner inside the table. There are 4 tables in this section: Team, Team_attributes, Team_history, Team_salary. These tables are identified by team_id and have 30 instances each.

For the player section, the available information is more granular. The section contains their biological and personal information, such as height, weight, and their alma mater. Historically, there are more than 4500 players who had participated in the league and there were many business actions involved, and there are four main activities that have been recorded: initial drafting, contract signing, contract waiving and trading. Note that there is only aggregated performance information available in the data source. For example, we can only see the average points, assists or rebounds throughout their entire career instead of year by year performance. Also, we can see their total count of all star appearances but not years they appeared in nor which position they were awarded for. There are 5 related tables in this section: Player, Player_Attributes, Player_Bios, Player_Photos, Player_Salary. They are all of similar sizes around 4,500 instances. The two outstanding tables are Player_attributes and Player_Bios, one has 37 columns and the other has 40 columns.

Data Wrangling

When we first got access to the original NBA dataset, we examined the data completeness and comprehensiveness among different tables. For all the other tables under the categories of games, players and teams, data wrangling process is shown below.

- **Game**
  This main table keeps data of each independent game in the NBA league which need to be kept as the main reference for game results. However, as there are 149 columns in this table, we dropped features that are irrelevant or mostly filled with null value.
- **Game_Inactive_Players**
  This table keeps information about players who are inactive in a game during a specific game due any kind of reason (absence, injuries, etc.) Because our application will only focus on players who have attended games during their careers, we have decided to drop this table as well.
- **Game_Officials**
  This table keeps track of all three referees working for every individual game across the records. Since we do not need information about the referees, we decided to drop this table.
- **Player**
  This is the main table that keeps information about the name and id of every individual player in the league. However, besides name and id, this table only has records of the active status of this player. We decided to join this table with *Player_Attributes* with Player ID and renamed it as *Player.*
- **Player_Attributes**
  This table contains attributes of each player, including height, weight, school, country, jersey number, etc. We decided to join this table with *Player* with Player ID and renamed the joined table as *Player.* We also dropped a few columns that are irrelevant to our application.
- **Player_Bios**
  This table contains large career events for each player, including the drafting, signing new contract, waived by the team and trade with another team. It also included information about each player slug season in different teams and corresponding salary to that specific season. We kept most of the columns and dropped some of the irrelevant or filled with null values. However, although this table has information about players, it does not contain player id, which disables it from joining with the new *Player* table accurately. In order to connect/join this table with the previous *Player* table, we used the basketball-reference website in the table, together with *slugPlayerBREF* (player slug abbreviation) as the search key. We also scrapped the birthday from the website. Then we combined the date of birth and *namePlayerBREF* (name abbreviation of player), as the

reference key, joined with the new *Player* table, fetching the player id as the identifier. This enables us to use id and name to seize corresponding records of one player.

- **Player_Photos**
  This table contains some basic information of the starting season and photos of each player. Since player_bio has the ‘urlPlayerImageBREF’ attribute which has the image url , we decided to drop the table.
- **Player_Salary**
  This table contains details of the player's salary of that specific season. As we do not need players’ contract types nor amount, we decided to drop this table.
- **Team**
  This table contains basic information about a team, including official name, name abbreviation, founding year, base city, etc. Due to its small number of columns and consistency with *Team_Attributes*, we decided to join those two tables together using team ID.
- **Team_Attributes**
  This table contains information about the team's arena, capacity, owner, coach, etc. Due to its small number of columns and consistency with *Team*, we decided to join those two tables together using team ID.
- **Team_History**
  This table contains the founding year, nickname of each team. Because it has duplicate information from the table *Team*, we decided to drop it.
- **Team_Salary**
  This table contains the current salary cap and future forecast of each team from season 2020 to 2025. Because the salary information only spans across five seasons, we decided to drop it.

After cleaning, joining and formatting those tables mentioned above, we also did some examination of the player’s consistency as we want to ensure users could get the same information of each player. We then pair players, teams information across their corresponding tables and check if there exists corresponding records across tables. There are less than one percent of players who do appear in all the tables. Considering the ease of implementation and keeping the records consistent across tables, we dropped those few records.

Database

There are 16 tables in the original data source, but for the sake of better performance and lower complexity, we dropped and combined tables. There are four aggregated tables remaining: Game, Player, Player_Bios, and Team. Each of the tables are in their 3rd normal form because all non-key values depend on the whole key. In regard to size, Game table has 60,688 instances;

Team table has 30 instances; Player table has 3,570 instances; Player_Bios table has 32,398 instances.

Relational Schema

- **Team** (id, full_name, abbreviation, nickname, city, state, year_founded, arena, owner, general_manager, head_coach, d_league_affiliation, salary_20_21, logo_url)
- **Player** (id, full_name, player_slug, birth_date, school, country, last_affiliation, height, weight, season_exp, jersey, position, roster_status, games_played_current_season_flag, team_id, player_code, from_year, to_year,
  draft_year, draft_round, draft_number, pts, ast, reb, all_star_appearances)
  i. FOREIGN KEY Team_id REFERENCES Team(id)
- **Player_Bios** (id, namePlayerBREF, urlPlayerBioBREF, nameTable, urlPlayerImageBREF, slugPlayerBREF, numberTransactionPlayer, descriptionTransaction, isGLeagueMovement, isDraft, isSigned, isWaived, isTraded, slugSeason, nameTeam, slugLeague, amountSalary)
  i. FOREIGN KEY id REFERENCES Player(id)
- **Game** (game_id, team_id_home, team_abbreviation_home, team_name_home, game_date, wl_home, min_home, fgm_home, ftm_home, fta_home, ft_pct_home, reb_home, ast_home, pf_home, pts_home, plus_minus_home, team_id_away, team_abbreviation_away, team_name_away, wl_away, min_away, fgm_away, ftm_away, fta_away, ft_pct_away, reb_away, ast_away, pf_away, pts_away, plus_minus_away, season, team_city_name_home, team_nickname_home, pts_qtr1_home, pts_qtr2_home, pts_qtr3_home, pts_qtr4_home, pts_ot1_home, pts_ot2_home, pts_ot3_home, pts_ot4_home, pts_ot5_home, pts_ot6_home, pts_ot7_home, pts_ot8_home, pts_ot9_home, pts_ot10_home, team_city_name_away, team_nickname_away, pts_qtr1_away, pts_qtr2_away, pts_qtr3_away, pts_qtr4_away, pts_ot1_away, pts_ot2_away, pts_ot3_away, pts_ot4_away, pts_ot5_away, pts_ot6_away, pts_ot7_away, pts_ot8_away, pts_ot9_away, pts_ot10_away)
  i. FOREIGN KEY TEAM_ID_HOME REFERENCES Team(id)
  ii. FOREIGN KEY TEAM_ID_AWAY REFERENCES Team(id)ER Diagram

ER Diagram

Please see Appendix C.

Queries

Game of the Season

Game of the Season queries the Game, Player, Player Bios table to present the most attractive and exciting game in our mind. Both the basic and advanced statistics are returned and displayed

on the **Home** page. In addition, we would like to move a step forward to show the amount of salary of all current players in the team. In this way, we present a close game in terms of the total amount of salary paid, which is $113,969,639 vs. $116,544,248 for Los Angeles Lakers and Los Angeles Clippers respectively.

Fun Fact

Fun Fact queries the Game, Team, Player Bios table to fill the key information of a paragraph of fun fact, including the year of founded, historical matchup statistics, most valuable player and his salary of both home and away team in the Fun Fact section on the **Game** page.

Player Network

Player Network queries Team, Player table to show how many first, second and third teammates of this player in the Player Details on the **Player** page. This is a similar concept as in our homework: the first teammates are players who played directly with the selected player in the same team and the second/third teammates are players who played with players in the above list.

Salary per Win

Salary per Win queries Game, Team, Player Bios table and returns season, win count, total salary and salary per win to plot a bar chart with an overlaid line chart on the **Team** page. The win count in each season for the selected team is plotted as a vertical bar and the salary per win is shown as a trend line. Also, a sliding bar is at the bottom to help zoom into a specific time range.

Player Flow

Player Flow queries Team, Player, Player Bios table and returns the player name, previous team, next team, in/out status, and logo url to demonstrate the players transferred in from other teams and out to other teams of a selected team in the last transfer period on the **Team** page. The result is visualized in a flow chart with the selected team name and logo in the middle and players in/out on both sides with their previous or next team logo next to their names.

Performance Evaluation

In our project, we observed in our ER diagram that all data was in Third Normal Form. Based on the definition, there is no transitive dependency for non-prime attributes and no non-primary key attribute has transitively dependent relationships to the primary key. Therefore, we do not need to manipulate the schema since the original data is already in 3NF.

We also performed optimization on allocating data for the game of the season visualization. Most of the columns could be retrieved from the Game table. However, we would like to move a step forward to show the amount of salary of all current players in the team. In order to calculate the amount of salary of all current players in both home and away teams, we need to consecutively join Game, Player, and Player Bios to get the result. The first attempt joins the tables, groups by columns and then uses having to filter the result. After learning and understanding the query optimization chapter, we realized that 1) the join order and 2) the filtering before joining is very important in terms of query runtime. First, the original join order was from Game (60688 rows) to Player (3570 rows) to Player Bios (32398 rows). This created a large table which contains 69,570,708 rows, which is not optimal in terms of either memory or runtime. Therefore, we decided to perform selection on each table before joining the table. Such query-restructuring technique results in a Game table of 1 row, Player table of 409 rows and Player Bios table of 367 rows. The joined table has only 12 rows, which is very sufficient. The runtime improved from 5 seconds and 437 milliseconds to 311 milliseconds after the query optimization.

We also applied the same technique to all of the rest complex queries and observed significant improvements in runtime. The Fun Fact query improved from 1 seconds and 831 milliseconds to 165 milliseconds. The Player Network query improved from 2 seconds and 394 milliseconds to 339 milliseconds. The Salary per Win query improved from 1 seconds 452 milliseconds to 174 milliseconds. The Player Flow query improved from 2 seconds 114 milliseconds to 293 milliseconds.

Technical Challenges and Solutions

Across the design and implementation of this application, we met several challenges due to data inconsistency, front-back end communication, and other problems. We worked as a team and found solutions to each problem mentioned above and listed their details below.

1. When we tried to design the relationship between each player-related table, we tend to use player id as the only key for identification of each player. However, table *Player_Bios* does not have a player id for reference. The name itself could be used as a search key because it's abbreviation of the name and problem of duplicate name is possible as well. After some analysis we found the uniqueness of the combination between slug name and DOB (date of birth) across every players-related table. Since the newly joined *Player* table has those two attributes and *Player_Bios* has *slugPlayerBREF*, we only need to add in DOB for table *Player_Bios*. We then used the basketball-reference website attributes in the table *Player_Bios* with the *slugPlayerBREF*, web-scraping the DOB for each player as the search key. Then the join of those two tables will provide player id for the *Player_Bios* as the primary search key.

2. There are often times when queries from the client side need to be cleaned before sending to the backend in order to get the desired output and vice versa. For example, when we extracted a player name from the url, because the url does not allow space, the extracted information became “Kobe%20Bryant” instead of “Kobe Bryant” that we needed. Similarly, sometimes data from the backend has a type of string, while the data itself is actually a numeric value. In such cases, we manipulated such data to convert to the correct data type.
3. Adapting bootstrapping charts to our app is also challenging because it requires thorough understanding of the configuration of these charts, such as input data format, parameters, and aesthetics. Many data cleanings were performed to generate insightful and visually appealing graphs.
4. Because our application uses several complex queries, fetching data from backend to frontend was very slow. However, with query optimizations, we were able to improve the speed impressively from 5 seconds to hundreds of milliseconds.

Extra Credit Features

There are a total of three extra credit features implemented in this application:

- We incorporated Google Map and used player headshots api to get dynamic information for teams and players. We also fetch a streaming video from the Youtube API.
- Unit tests for backend were performed and reached 96% of coverage.
- The application is successfully deployed via Heroku: https://league-pass.herokuapp.com/

Appendix

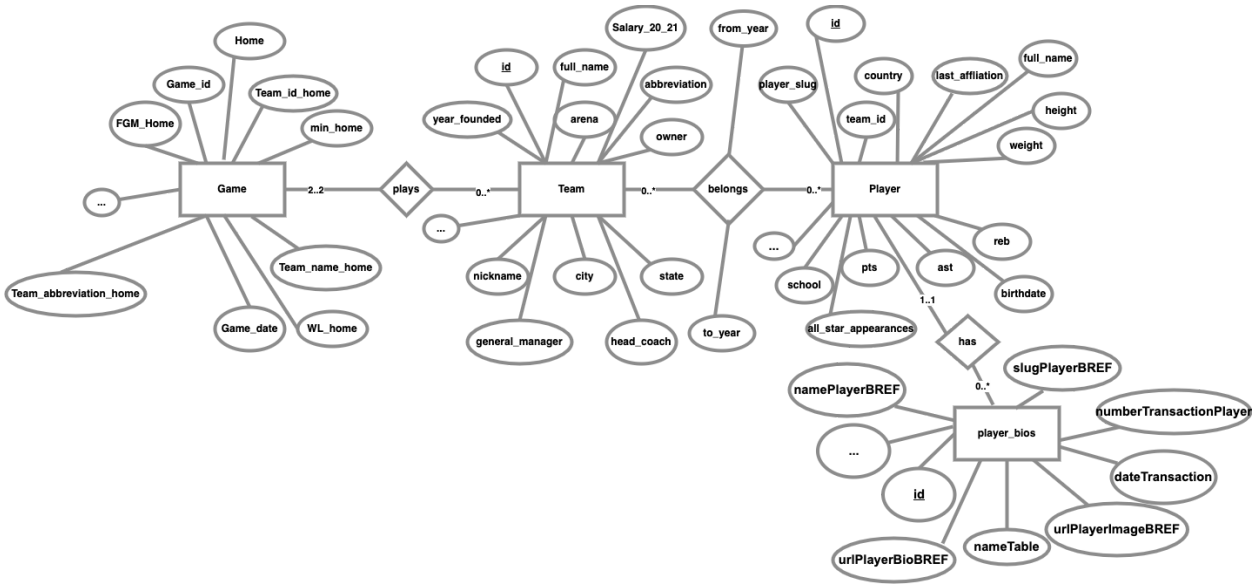
Appendix A: Github Repository

- https://github.com/yulinyu1013/CIS-550-NBA-Dream-Team-Project

Appendix B: Data Repositories and Glossary

- NBA datasets: https://www.kaggle.com/wyattowalsh/basketball
- Basketball Statistics Glossary: https://en.wikipedia.org/wiki/Basketball_statistics

Appendix C: Entity Relationship Diagram



Appendix D: SQL Queries

1. Game of the season (complex query)

```
WITH home_salary(team_name_home, team_salary_home)
     AS (SELECT G.team_name_home,
                Sum(PB.amountsalary)
         FROM   (SELECT team_id_home,
                        team_name_home
                 FROM   game
                 WHERE  team_abbreviation_home = 'LAL'
                        AND team_abbreviation_away = 'LAC'
                        AND game_date = '2020-12-22') AS G
                JOIN (SELECT id,
                             team_id
                      FROM   player
                      WHERE  to_year = 2020) AS P
                  ON G.team_id_home = P.team_id
                JOIN (SELECT id,
                             amountsalary
                      FROM   player_bios
                      WHERE  LEFT(slugseason, 4) = '2020') AS PB
                  ON P.id = PB.id
         GROUP  BY G.team_name_home),
     away_salary(team_name_away, team_salary_away)
     AS (SELECT G.team_name_away,
                Sum(PB.amountsalary)
         FROM   (SELECT team_id_away,
```

```
                       team_name_away
                FROM   game
                WHERE  team_abbreviation_home = 'LAL'
                       AND team_abbreviation_away = 'LAC'
                       AND game_date = '2020-12-22') AS G
               JOIN (SELECT id,
                            team_id
                     FROM   player
                     WHERE  to_year = 2020) AS P
                 ON G.team_id_away = P.team_id
               JOIN (SELECT id,
                            amountsalary
                     FROM   player_bios
                     WHERE  LEFT(slugseason, 4) = '2020') AS PB
                 ON P.id = PB.id
         GROUP  BY G.team_name_away)
SELECT team_abbreviation_home,
       G.team_name_home,
       game_date,
       pts_home,
       reb_home,
       ast_home,
       fgm_home,
       ftm_home,
       fta_home,
       ft_pct_home,
       pf_home,
       team_abbreviation_away,
       G.team_name_away,
       pts_away,
       reb_away,
       ast_away,
       fgm_away,
       ftm_away,
       fta_away,
       ft_pct_away,
       pf_away,
       team_salary_home,
       team_salary_away
FROM   game G
       JOIN home_salary HS
         ON G.team_name_home = HS.team_name_home
       JOIN away_salary AWS
         ON G.team_name_away = AWS.team_name_away
WHERE  team_abbreviation_home = 'LAL'
```

```
        AND team_abbreviation_away = 'LAC'
        AND game_date = '2020-12-22';
```

2. Top 10 all time leaders

```
(SELECT full_name     AS player,
        'pts'         AS category,
        Round(pts, 3) AS value
 FROM   player
 ORDER  BY pts DESC
 LIMIT  10)
UNION
(SELECT full_name     AS player,
        'ast'         AS category,
        Round(ast, 3) AS value
 FROM   player
 ORDER  BY ast DESC
 LIMIT  10)
UNION
(SELECT full_name     AS player,
        'reb'         AS category,
        Round(reb, 3) AS value
 FROM   player
 ORDER  BY reb DESC
 LIMIT  10);
```

3. Fun fact (complex query)

```
WITH home_team
     (
          team_name,
          year_founded
     )
     AS
     (
            SELECT full_name,
                   year_founded
            FROM   team
            WHERE  full_name = '${params.home_team}'
     )
     ,
     away_team
     (
          team_name,
          year_founded
     )
     AS
```

```
(
        SELECT full_name,
               year_founded
        FROM   team
        WHERE  full_name = '${params.away_team}'
)
,
home_team_mvp
(
     team_name,
     player_name,
     salary
)
AS
(
          SELECT    nameteam,
                    nameplayerbref,
                    Max(amountsalary)
          FROM      player_bios
          WHERE     nametable = 'Salaries'
          AND       nameteam = '${params.home_team}'
          AND       amountsalary IS NOT NULL
          GROUP BY nameteam,
                    nameplayerbref
          ORDER BY Max(amountsalary) DESC limit 1
)
,
away_team_mvp
(
     team_name,
     player_name,
     salary
)
AS
(
          SELECT    nameteam,
                    nameplayerbref,
                    Max(amountsalary)
          FROM      player_bios
          WHERE     nametable = 'Salaries'
          AND       nameteam = '${params.away_team}'
          AND       amountsalary IS NOT NULL
          GROUP BY nameteam,
                    nameplayerbref
          ORDER BY Max(amountsalary) DESC limit 1
```

```
    )
SELECT     w2.*,
           ht.year_founded AS home_team_founded,
           at.year_founded AS away_team_founded,
           htm.player_name AS home_mvp,
           htm.salary      AS home_mvp_salary,
           atm.player_name AS away_mvp,
           atm.salary      AS away_mvp_salary
FROM       (
                  SELECT '${params.home_team}'          AS home_team,
                         '${params.away_team}'          AS away_team,
                         Sum(lal_win)                   AS home_win,
                         Count(lal_win) - Sum(lal_win) AS away_win,
                         Count(lal_win)                 AS total_match
                  FROM   (
                                SELECT team_name_home,

team_name_away,IF(team_name_home = '${params.home_team}'
                                AND    wl_home = 'W'
                                OR     team_name_away =
'${params.home_team}'
                                AND    wl_away = 'W', 1, 0) as lal_win
                                FROM   game
                                WHERE  (
                                              team_name_home =
'${params.home_team}'
                                       AND    team_name_away =
'${params.away_team}')
                                OR     (
                                              team_name_home =
'${params.away_team}'
                                       AND    team_name_away =
'${params.home_team}')) AS w) AS w2
LEFT JOIN home_team ht
ON         w2.home_team = ht.team_name
LEFT JOIN away_team at
ON         w2.away_team = at.team_name
LEFT JOIN home_team_mvp htm
ON         w2.home_team = htm.team_name
LEFT JOIN away_team_mvp atm
ON         w2.away_team = atm.team_name;
```

4. Game search

```
SELECT game_id,
       team_abbreviation_home,
```

```
        team_name_home,
        game_date,
        season,
        pts_home,
        reb_home,
        ast_home,
        fgm_home,
        ftm_home,
        fta_home,
        ft_pct_home,
        pf_home,
        team_abbreviation_away,
        team_name_away,
        pts_away,
        reb_away,
        ast_away,
        fgm_away,
        ftm_away,
        fta_away,
        ft_pct_away,
        pf_away
FROM    game
WHERE   (
                team_abbreviation_home LIKE '%${params.home}%'
        OR      team_name_home LIKE '%${params.home}%')
AND     (
                team_abbreviation_away = '%${params.away}%'
        OR      team_name_away LIKE '%${params.away}%')
AND     game_date >= '${params.min_date}'
AND     game_date <= '${params.max_date}'
AND     pts_home >= ${params.pts_home_low}
AND     pts_home <= ${params.pts_home_high}
AND     pts_away >= ${params.pts_away_low}
AND     pts_away <= ${params.pts_away_high}
AND     reb_home >= ${params.reb_home_low}
AND     reb_home <= ${params.reb_home_high}
AND     reb_away >= ${params.reb_away_low}
AND     reb_away <= ${params.reb_away_high}
AND     ast_home >= ${params.ast_home_low}
AND     ast_home <= ${params.ast_home_high}
AND     ast_away >= ${params.ast_away_low}
AND     ast_away <= ${params.ast_away_high};
```

5. Performance history of teams on Game page

```
SELECT 'Home'                                   AS game,
```

```
        Count(*)                                        count,
        Count(CASE
                WHEN wl_home = 'W' THEN 1
              END)                                      win,
        Count(CASE
                WHEN wl_home = 'L' THEN 1
              END)                                      lost,
        Avg(IF(wl_home = 'W', plus_minus_home, 0)) avg_win_pts,
        Avg(IF(wl_home = 'L', plus_minus_home, 0)) avg_lost_pts
FROM    game
WHERE   team_abbreviation_home = '${name}'
UNION
SELECT 'Away'                                           AS game,
        Count(*)                                        count,
        Count(CASE
                WHEN wl_away = 'W' THEN 1
              END)                                      win,
        Count(CASE
                WHEN wl_away = 'L' THEN 1
              END)                                      lost,
        Avg(IF(wl_away = 'W', plus_minus_away, 0)) avg_win_pts,
        Avg(IF(wl_away = 'L', plus_minus_away, 0)) avg_lost_pts
FROM    game
WHERE   team_abbreviation_away = '${name}';
```

6. Player search

```
SELECT p.id         AS player_id,
       t.full_name AS team,
       p.full_name AS full_name,
       player_slug,
       birth_date,
       school,
       country,
       last_affiliation,
       height,
       weight,
       season_exp,
       jersey,
       position,
       roster_status,
       games_played_current_season_flag,
       team_id,
       player_code,
       from_year,
       to_year,
```

```
        draft_year,
        draft_round,
        draft_number,
        Round(pts,3) AS pts,
        Round(ast,3) AS ast,
        Round(reb,3) AS reb,
        all_star_appearances,
        salary
FROM    player P
JOIN
        (
                 SELECT   id,
                          Max(amountsalary) AS salary
                 FROM     player_bios
                 WHERE    nametable = 'Salaries'
                 GROUP BY id) S
ON      p.id = s.id
JOIN
        (
               SELECT id,
                      full_name
               FROM   team) T
ON      p.team_id = t.id
WHERE   p.full_name LIKE '${params.first_name}%'
AND     p.full_name LIKE '%${params.last_name}'
AND     player_slug LIKE '%${params.player_slug}%'
AND     salary >= ${params.min_salary}
AND     pts >= ${params.pts_low}
AND     pts <= ${params.pts_high}
AND     reb >= ${params.reb_low}
AND     reb <= ${params.reb_high}
AND     ast >= ${params.ast_low}
AND     ast <= ${params.ast_high}
AND     position LIKE '%${params.position}%';
```

7. Player network (complex query)

```
WITH first_team(team_id, team_name)
     AS (SELECT P.team_id,
                T.full_name AS team_name
         FROM   player P
                JOIN team T
                  ON P.team_id = T.id
         WHERE  P.full_name = '${name}'),
     first_teammate(full_name, team_id, connection)
     AS (SELECT P.full_name,
```

```
            P.team_id,
            1
    FROM    player P
            JOIN first_team FT
              ON P.team_id = FT.team_id
    WHERE   P.full_name != '${name}'),
second_team(team_id, team_name)
AS (SELECT P.team_id,
            T.full_name AS team_name
    FROM    player P
            JOIN team T
              ON P.team_id = T.id
    WHERE   P.full_name IN (SELECT full_name
                            FROM   first_teammate)),
second_teammate(full_name, team_id, connection)
AS (SELECT P.full_name,
            P.team_id,
            2
    FROM    player P
            JOIN second_team ST
              ON P.team_id = ST.team_id
    WHERE   P.full_name != '${name}'
            AND P.full_name NOT IN (SELECT full_name
                                    FROM   first_teammate)),
third_team(team_id, team_name)
AS (SELECT P.team_id,
            T.full_name AS team_name
    FROM    player P
            JOIN team T
              ON P.team_id = T.id
    WHERE   P.full_name IN (SELECT full_name
                            FROM   second_teammate)),
third_teammate(full_name, team_id, connection)
AS (SELECT P.full_name,
            P.team_id,
            3
    FROM    player P
            JOIN third_team TT
              ON P.team_id = TT.team_id
    WHERE   P.full_name != '${name}'
            AND P.full_name NOT IN (SELECT full_name
                                    FROM   first_teammate)
            AND P.full_name NOT IN (SELECT full_name
                                    FROM   second_teammate))
SELECT
```

```
connection,
       Count(full_name) AS numPlayer
FROM   first_teammate
GROUP  BY connection
UNION
SELECT connection,
       Count(full_name) AS numPlayer
FROM   second_teammate
GROUP  BY connection
UNION
SELECT connection,
       Count(full_name) AS numPlayer
FROM   third_teammate TT
GROUP  BY connection;
```

8. Player salary vs. team average salary

```
SELECT *
FROM   (
              SELECT id,
                     nameplayerbref    AS 'namePlayer',
                     player.nameteam   AS 'nameTeam',
                     player.slugseason AS slugseason,
                     salaryfromplayer,
                     salaryacrossteam,IF(salaryfromplayer >
salaryacrossteam, 1, 0) as valuedplayer
              FROM   (
                              SELECT   id,
                                       nameplayerbref,
                                       nameteam,
                                       slugseason,
                                       sum(amountsalary) AS
'salaryFromPlayer'
                              FROM     player_bios
                              WHERE    nametable = 'Salaries'
                              GROUP BY id,
                                       nameplayerbref,
                                       slugseason,
                                       nameteam
                              ORDER BY id,
                                       slugseason) AS player
              JOIN
                     (
                              SELECT   slugseason,
                                       nameteam,
```

```
                                        avg(amountsalary) AS
'salaryAcrossTeam'
                                FROM    player_bios
                                GROUP BY slugseason,
                                        nameteam) AS team
                ON      player.slugseason = team.slugseason
                AND     player.nameteam = team.nameteam) t
WHERE  id = ${id};
```

9. Team search

```
SELECT id AS team_id,
       full_name,
       abbreviation,
       nickname,
       city,
       state,
       year_founded,
       arena,
       owner,
       d_league_affiliation
FROM   team
WHERE  (
              abbreviation = '%${params.name}%'
       OR     full_name LIKE '%${params.name}%')
AND    year_founded >= ${params.year_founded_min}
AND    year_founded <= ${params.year_founded_max}
AND    city LIKE '%${params.city}%'
AND    state LIKE '%${params.state}%'
AND    arena LIKE '%${params.arena}%'
AND    owner LIKE '%${params.owner}%';
```

10. Salary per win (complex query)

```
SELECT TW.season,
       TW.win_count,
       TS.salary,
       TS.salary / TW.win_count AS salary_per_win
FROM   (SELECT season,
               Count(game_id) AS win_count
        FROM   game
        WHERE  ( team_name_home = '${name}'
                 AND wl_home = 'W' )
                OR ( team_name_away = '${name}'
                     AND wl_away = 'W' )
        GROUP  BY season) AS TW
       JOIN (SELECT *
```

```
            FROM   (SELECT Cast(LEFT(slugseason, 4) AS SIGNED) AS
season,
                           nameteam                            AS
team_name,
                           Sum(amountsalary)                   AS
salary
                    FROM   player_bios
                    WHERE  nametable = 'Salaries'
                           AND nameteam IS NOT NULL
                    GROUP  BY season,
                              nameteam
                    UNION
                    SELECT 2020         AS season,
                           full_name    AS team_name,
                           salary_20_21 AS salary
                    FROM   team) AS Salary
            WHERE  team_name = '${name}') AS TS
        ON TW.season = TS.season
ORDER  BY season;
```

11. Player flow on Team page (complex query)

```
WITH team_logo
     (
          full_name,
          logo_url
     )
     AS
     (
            SELECT full_name,
                   logo_url
            FROM   team
     )
SELECT id,
       player_flow.full_name AS player_full_name,
       nextteam,
       prevteam,
       transferyear,
       status,
       t1.logo_url AS nextteamlogourl,
       t2.logo_url AS prevteamlogourl
FROM   (
              SELECT player_id.id AS id,
                     full_name,
                     nameteam AS nextteam,
                     prevteam,
```

```
                transferyear,
                CASE
                        WHEN prevteam LIKE '${name}' THEN 'Out'
                        ELSE 'In'
                END AS status
        FROM    (
                        SELECT id,
                                full_name
                        FROM   player) AS player_id
        JOIN
                (
                        SELECT id,
                                nameteam,
                                prevteam,
                                Substring(slugseason, 1, 4) AS
transferyear
                        FROM   (
                                        SELECT id,
                                                nameteam,
                                                slugseason,
                                                Ifnull(Lag(nameteam)
OVER (partition BY id), 'N/A') AS prevteam
                                        FROM   player_bios
                                        WHERE  nametable =
'Salaries') AS t
                        WHERE  nameteam != prevteam) AS
wholerecords
        ON     player_id.id = wholerecords.id
        WHERE  transferyear LIKE '2019'
        AND    (
                        prevteam LIKE '${name}'
                OR     nameteam LIKE '${name}') ) AS player_flow
JOIN   team_logo                                        AS t1
ON     nextteam = t1.full_name
JOIN   team_logo AS t2
ON     prevteam = t2.full_name limit 10;
```

Appendix E: Backend Unit Test Coverage

```
-----------------|---------|----------|---------|---------|
File             | % Stmts | % Branch | % Funcs | % Lines |
-----------------|---------|----------|---------|---------|
All files        |   89.61 |    64.28 |   96.15 |   89.56 |
 dbOperations.js |   86.07 |      100 |     100 |   86.07 |
 routes.js       |   90.47 |    64.63 |     100 |    90.4 |
 server.js       |   96.15 |       50 |      50 |   96.15 |
-----------------|---------|----------|---------|---------|
```