NBA Database Application (NBADBA)

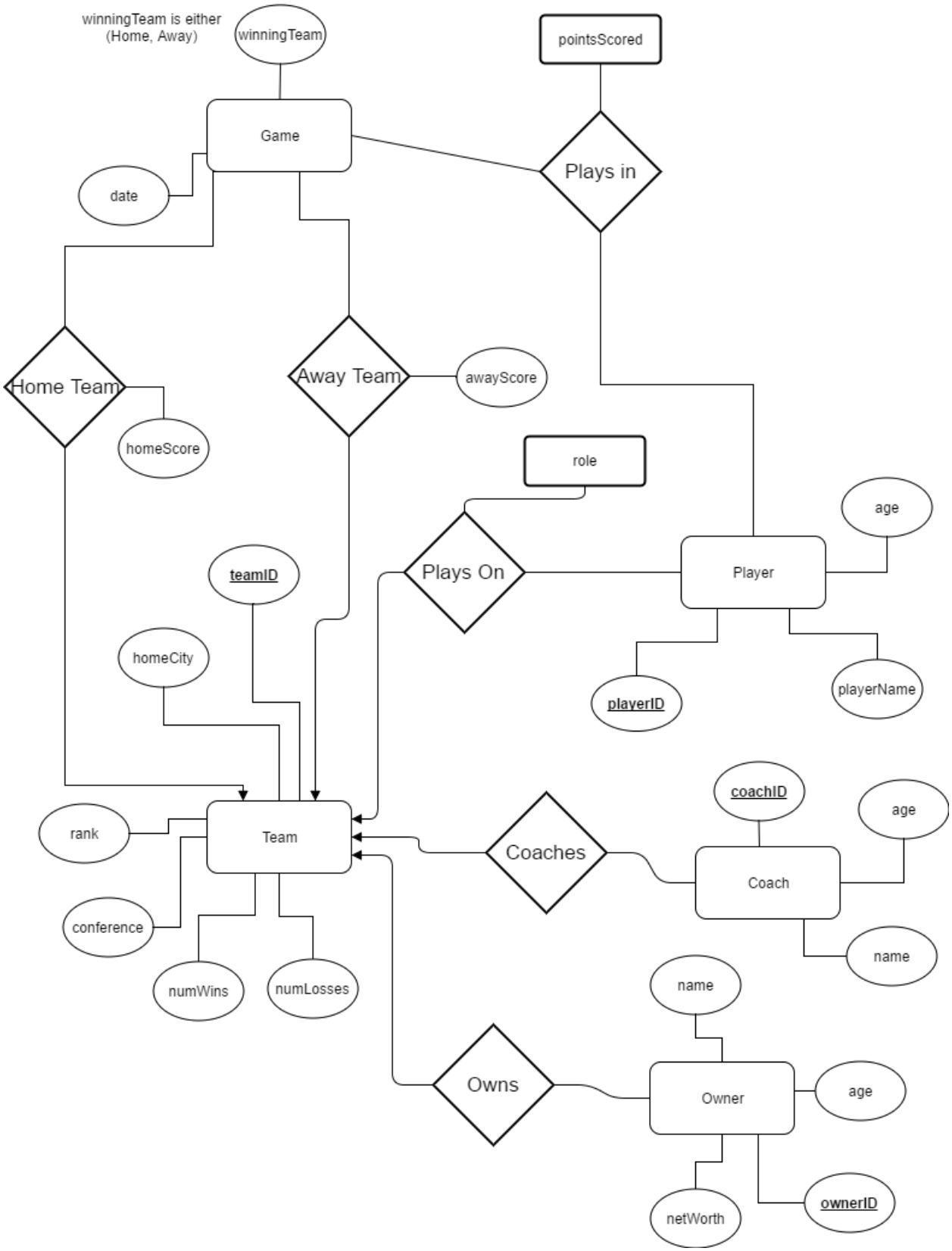
Rahul Pokharna, Ayush Karnawat, Sibi Sengottuvel
EECS 341

Application Background:

The NBA Database Application, or NBADBA for short, is an application that we created to represent the 2015-2016 season of the NBA. A possibility for the future is to use copies of this database for each season, with specifying the year to choose the specific portion of the database to query from. What our application currently does is give basic stats on players, teams, and games. We have tabs present to display all the players, games, and teams, and some simple SQL statements to fill them with information as well as list them all. We have separate tab for all other queries, which are forms where a user can input information and other variables to complete the forms for search queries, finding average heights of roles, and other such queries discussed below. We have information regarding all teams, players, coaches, and owners of each team. The main idea of this project is to create a place to show all of the data for a single season of the NBA in one place. A link to our hosted application.

Data Description:

The data is of every game of every team, player, coach, owner, game, and each player that plays in each game. There are over 200 coaches, 1230 games, 23000 instances of players playing in games, 30 teams, 380 players, and so on. The data itself is very large, and we restricted it to a single season for this reason. All players are on a team, we have their heights, how many points they scored in every game they played it, all the games, who won and what the scores were, and other data as seen in our tables below. Further in the document, we see a better description of the specifications of our data.



Our E/R Diagram

Functional Dependancies and Schema:

Foreign Keys are italicized, primary keys are underlined
Teams (TeamID, TeamName, Rank, Conference, numWins, numLosses)
F: (TeamID → {TeamName, HomeCity, Rank, Conference, numWins, numLosses})

Players (PlayerId, PlayerName, age, *TeamName*, Role)
F: (PlayerID → {PlayerName, age, TeamName, Role})

GameScore (GameID, Date, *HomeTeamID, AwayTeamID,* homeScore, awayScore)
F: ({Date, HomeTeamID, AwayTeamID} → { homeScore, awayScore,GameID}, {GameID} → {Date, HomeTeamID, AwayTeamID})

Games (GameId, Date, *HomeTeamID, AwayTeamID,* winningTeam)
F: ({Date, HomeTeamID, AwayTeamID} → {WinningTeam,GameID}, {GameID} → {Date, HomeTeamID, AwayTeamID})

Coach(coachID, CoachName, age, *TeamName*)
F: (coachID → { age, CoachName, TeamName})

Owner (ownerID, name, age, netWorth, *TeamName*)
F: (ownerID → {name, age, netWorth, TeamName})

PlaysIn(*PlayerID,* Date, *HomeTeamID, AwayTeamID*, PointsScored)
F: ({PlayerID, Date, HomeTeamID, AwayTeamID} → {PointsScored})

All of these tables are in BCNF because the primary keys are the only attributes that determine any other attributes. In Games and GameScore GameId also determines the other attributes, but this is still in BCNF because GameId is still a superkey as it is a candidate key. The only reason we included GameId instead of leaving it out is to make it easier to work with the Django implementation

Example Queries Supported by our Database:

For our queries, many of them are not seen easily or are obvious, but we do have many queries in our application. We have some to display every team, game and player, as well as the information inside of them, and some joins to get greater information regarding teams, such as the owner information in the teams tab. Many of our queries are built into the application pages themselves, showing all of the pertinent information there. Our first query below is built into the players tab as well, similar to the query mentioned above. Our Query 2 can be used to

determine many different queries, depending on how the user inputs the data. If the user is able to utilize it fully, that query can be used to determine many things, such as the points scored on average by every single Guard, or the average points on every player taller than 84 inches (7 feet), and so on. The % is a wildcard character, meaning that any length of characters can be in that area, from no characters to the max length, as long as something in the database is able to fill that area to complete the query, that condition will be satisfied.

For dynamic queries, we indicate the dynamic attribute with [attribute_name].

Query 1: Getting the average points scored for each player over all the games from the season. This query is done every time a user clicks on any player, and the resulting value is shown in a table on the specific player's page.

SQL:

```
SELECT avg(PlaysIn.points_scored)
FROM Players, PlaysIn
WHERE Players.player_id = PlaysIn.player_id
GROUP BY Players.player_id
```

RA:

$$\prod_{avg(points_{scored})} (player\_id\ \mathcal{G}_{avg(points_{scored})}\ \sigma_{playerId=playerId}(Players \bowtie PlaysIn))$$

TRC:

The book says "tuple relational calculus does not have any equivalent of the aggregate operation, and we arent sure how to implement it since each t is its own tuple, so we have added the aggregate function at the end of the query, even though it logically does not make sense.

$$\{t|\exists(player \in Players \wedge player\_id = [PLAYER\_ID] \wedge \exists(playsin \in Playsin \wedge playsin.player\_id = player.player\_id)) \wedge t[avgPoints] = average(player.pointsScored)\}$$

Query 2: The second query is the most dynamic query we have, where the user can specify many things about players, and get all the results with players the fit that criteria. For example, they can ask for players with name containing 'Jim' who are taller than 72 inches, and have avg points for game greater than 15.

SQL:

```
SELECT Player_Name, Team_name, Role, Height, avg(Points_Scored)
FROM Players p, PlaysIn, Teams t
WHERE Player_ID = Player_ID_ID
        AND player_Name like '%'+inputName+'%'
```

```
        AND height > minHeight
        AND height < maxHeight
        AND Team_Name like '%'+inputTeam+'%'
        AND p.team_id = t.team_id
        AND p.role like '%'+inputRole+'%'
GROUP BY Player_ID
Having avg(Points_Scored) > minAvg and avg(points_scored) < maxAvg
```

RA:

$$\prod_{playerName,teamName,avg(points_{scored}),role,height} \sigma_{avg(pointsScored)>[MINAVG] and\ avg(pointsScored)<[MAXAVG]}(player\_id\ \mathcal{G}_{avg(points_{scored})}$$

$$\sigma_{playerName=[PLAYERNAME],height>[minHeight],height<[maxHeight],TeamName=[TeamName],}\ (Players \bowtie PlaysIn \bowtie Teams))$$

TRC:

$$\{t|\exists(player \in Players \wedge player\_name = [PLAYER\_NAME] \wedge t[playerName] = player\_name \wedge t[height] = player.height \wedge player.height < [maxHeight] \wedge player.height > [minHeight] \wedge player.role = t[role] \wedge \exists(playsin \in Playsin \wedge playsin.player_{id} = player.player_{id} \wedge \exists(team \in Teams \wedge team.teamID = player.teamID \wedge teamName = [TEAMNAME])) \wedge t[avgPoints] = average(player.pointsScored) \wedge average(player.poinstScored) > [MINAVG] \wedge average(player.poinstScored) < [MAXAVG]\}$$

Query 3: This query gets the win percentage of the home games played of a specified team, and is displayed behind the scenes on the Team Page

SQL:

```
SELECT Team_Name, (count(*) / 41.)
FROM Games g, Teams t
WHERE t.team_ID = g.home_Team_ID_id
        AND g.winning_Team like 'home'
        AND t.team_ID = [team_id]
```

RA: $\prod_{\text{teamName},\frac{\text{size}}{41}.}(\sigma_{\text{teamID}=[\text{teamID}] \wedge winningTeam='HOME'}(Teams \bowtie_{teamID=homeTeamID} Games))$

TRC: $\{t|(\exists te \in Teams \wedge te[teamID] = [inputTeamID] \wedge (\exists g \in Games \wedge g[homeTeamID] = te[teamID] \wedge t[teamName] = te[teamName]) \wedge t[avgWins] = (size/41))\}$

Aggregate functions do not work in TRC, but I included to represent that it is there after the calculations had been completed

Query 4: This query returns every player that has played in every home game for a specified team

SQL:

For all players that played in every home game of a dynamically specified team.

```
SELECT p.Player_name
FROM Players p
WHERE NOT EXISTS (SELECT *
                  FROM games g
                  WHERE g.home_team_id_id = [team_id]
                  AND NOT EXISTS (SELECT *
                        FROM playsIn pi
                        WHERE g.away_team_id_id = pi.away_team_id_id
                              AND g.home_team_id_id = pi.home_team_id_id
                              AND g.date = pi.date
                              AND p.player_id = pi.player_id_id))
```

RA:

$$\prod_{date,homeTeamId,awayTeamID,playerID} \left(\sigma_{[teamID]=homeTeamId}(Game \bowtie PlaysIn)\right) \div \prod_{playerID} Players$$

TRC:

$$\{t|\exists \Big(player \epsilon Players \wedge t[playerName] = player.playerName0 \wedge \exists \Big(team \epsilon Teams \wedge team.teamID = [team_{id}] \wedge \neg\exists\big(game \epsilon Games \wedge team.teamID = game.homeTeamID \wedge \neg\exists(playsin \epsilon PlaysIn \wedge playsin.homeTeamID = game.homeTeamID \wedge playsin.awayTeamID = game.awayTeamID \wedge playsin.date = game.date \wedge playsin.playerID = player.playerID)\big)\Big)\Big)\}$$

Implementation:

We are using Django based server, and with Django, we used Python and SQLite3. We programmed the website on Windows and Mac machines, and are running the server on linux instance from Amazon Web Services. Django creates the frontend and connecting that to the backend with Python, we have SQLite3 holding our data. In order to ensure accuracy and

completeness, we get all of our data from stats.nba.com. We used endpoints such as this link to get data in json format for all of our tables. Even though these endpoints are not documented we were able to find all of the endpoints we needed, and make requests to them.The JSON objects we get from these requests are then parsed and put into the database using the sqlite3 package for python. All of the scripts used to scrape stats.nba.com can be found in the Database folder. We used a software to view the database and all of the data before we created the website itself called DB Browser for SQLite, which showed us everything in the database, constraints and any other pertinent information.

Role Contributions:

Each member contributed heavily to the completion of the project with lots of collaboration throughout the project even though we separated the roles.

Rahul came up with the database design and schemas. He also was able to find the endpoints from stats.nba.com needed to populate the database. He also wrote the code to create the sqlite3 database. He also wrote the SQL queries that were later put into Django.

Sibi wrote the python scripts that make all the requests to stats.nba.com and then parses the JSON data, and then put it into the sqlite database using SQL insert queries.

Ayush made the front-end using Django, injected all of the HTML, and CSS, and added in every query to the website.

We worked together to create a variety of queries to add to our application for users to parse through and view.

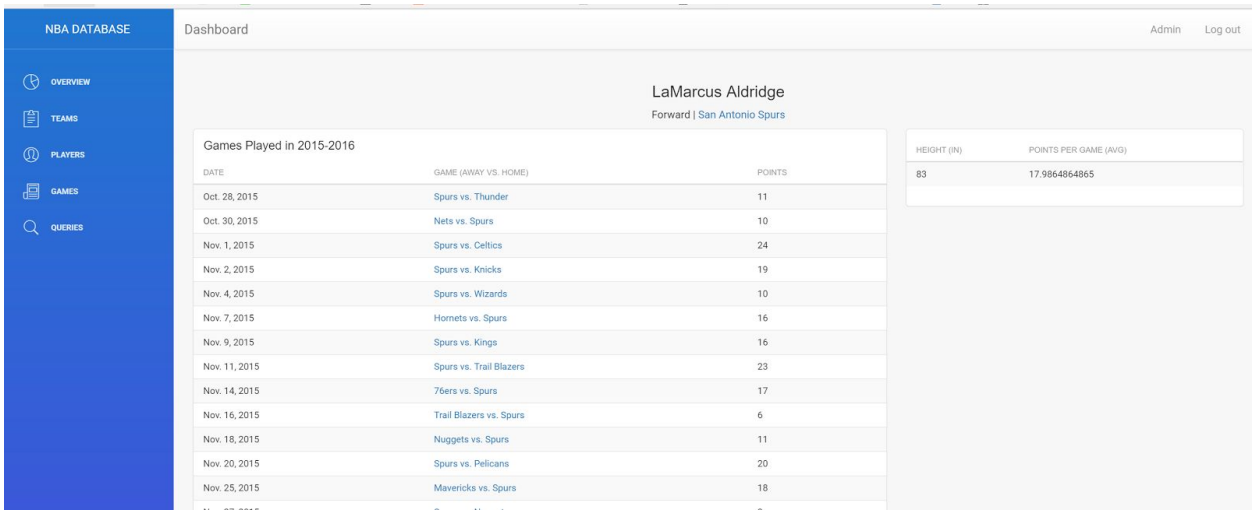
What We Learned:

In this project, we learned many more things than we did in class. This project allowed us to actually connect a database to a website and parse through data. It allowed us to learn how to use more versions of SQL to choose which fit our demands the best, as well as how other languages can interact with these SQL variants. We learned Python, reinforced our SQL background from the class, learned how to parse through JSON objects. We also learned a lot of HTML and CSS, as well as how to properly do an E/R Diagram. We learned how to develop

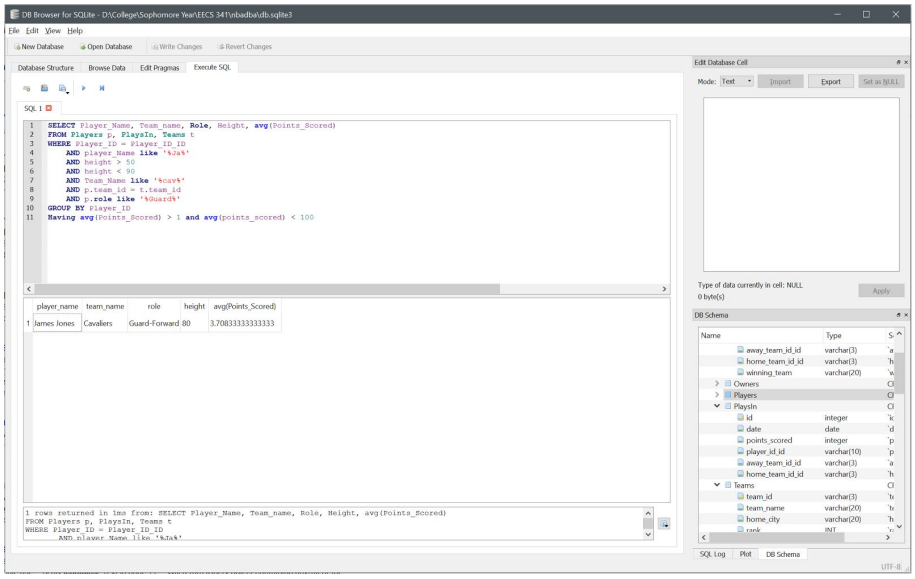
an application as a group as well as how to use git to collaborate on a project together. The project was a good experience in allowing us to see the practical applications of the database, and how we can use it to complete tasks that may be difficult to complete before learning it here.

Screenshots of Application Running and Queries Running:

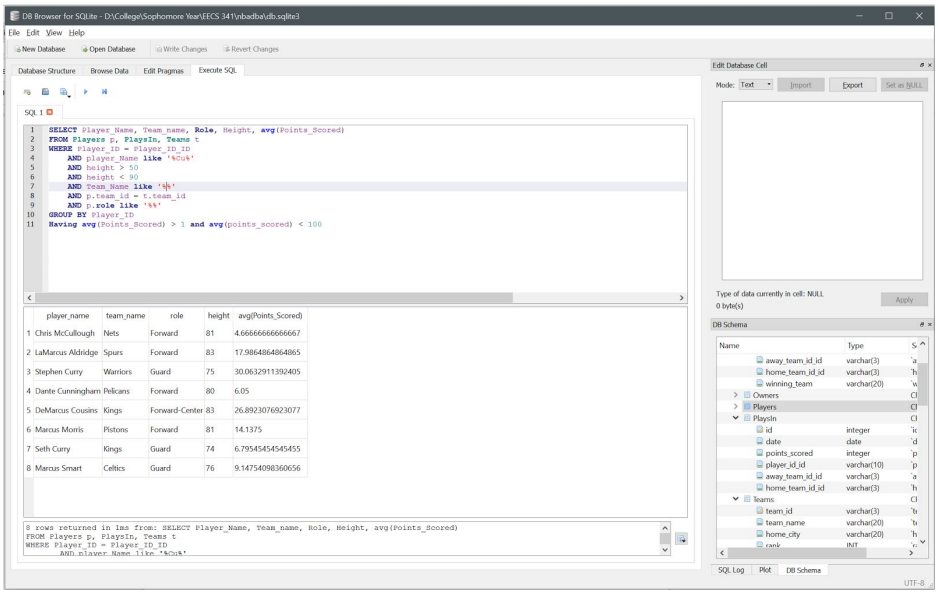
Query 1 (built in on the player's page) and also other queries shown



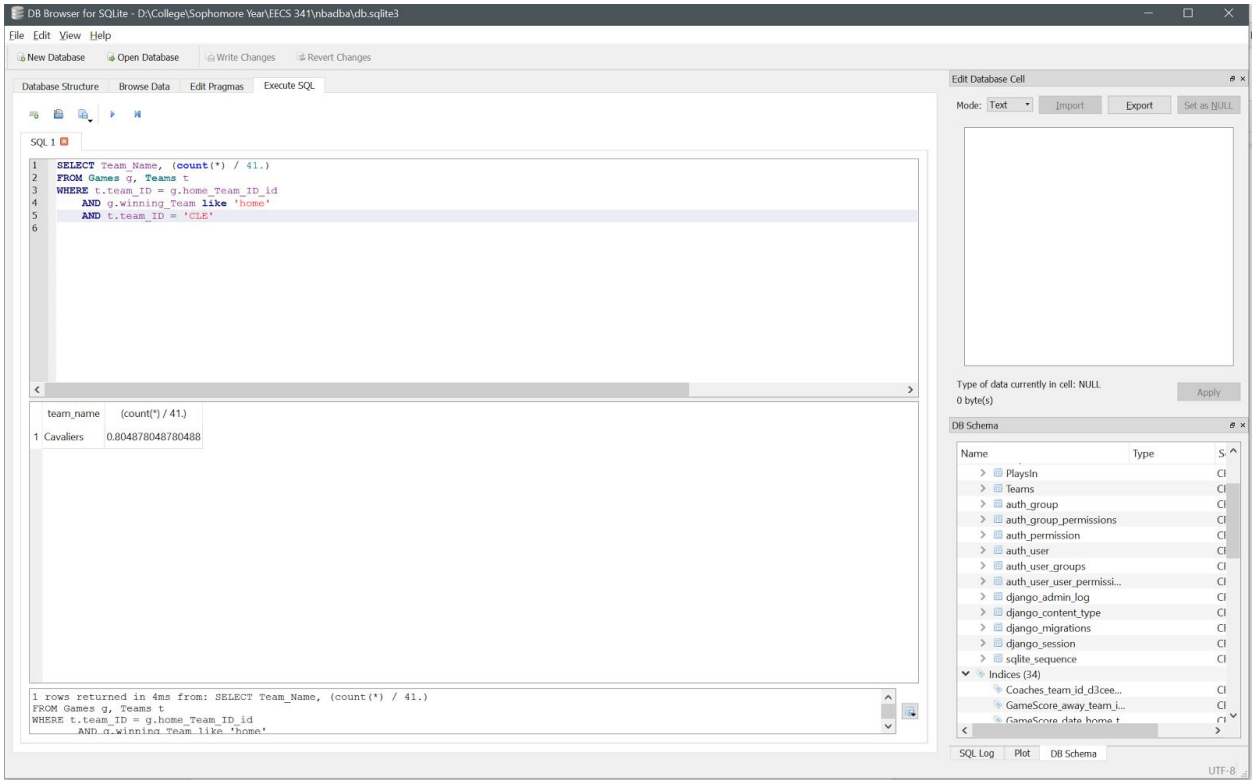
Query 2 running with sample inputs (testing software)



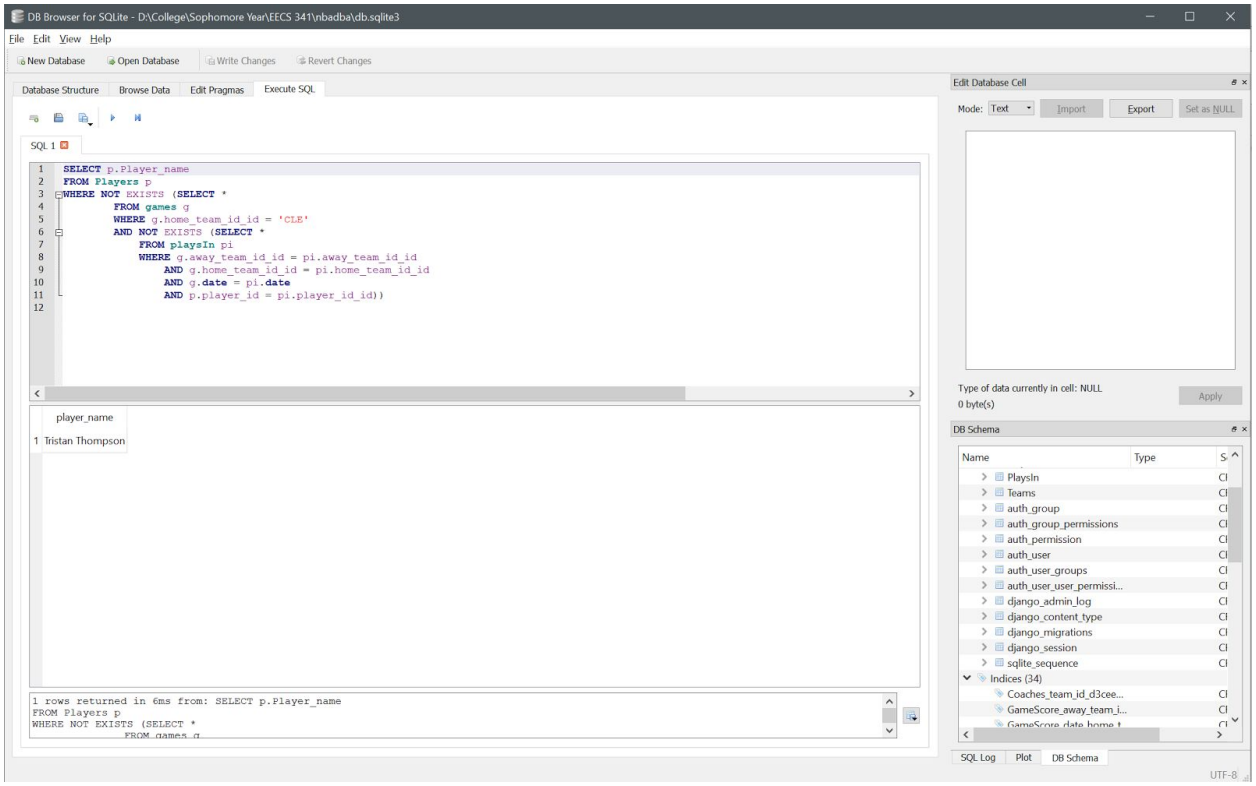
Query 2 again, with new inputs



Query 3 with Cavs as an input

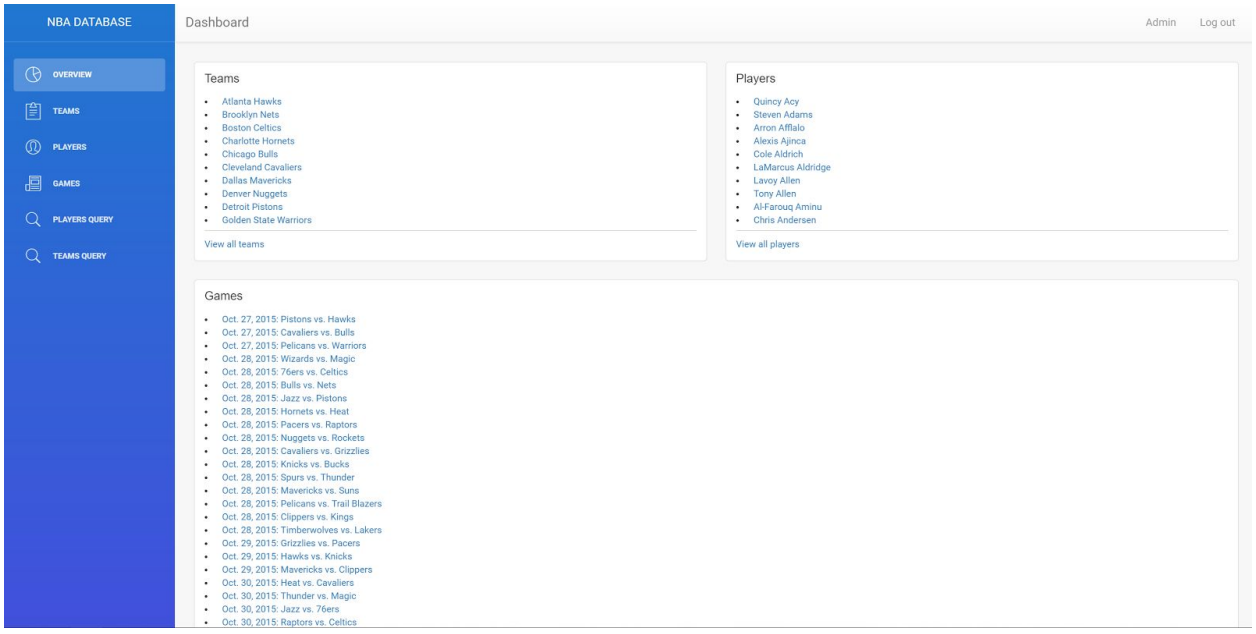


Query 4 with Cleveland as the input (showing all players who played in every home game)

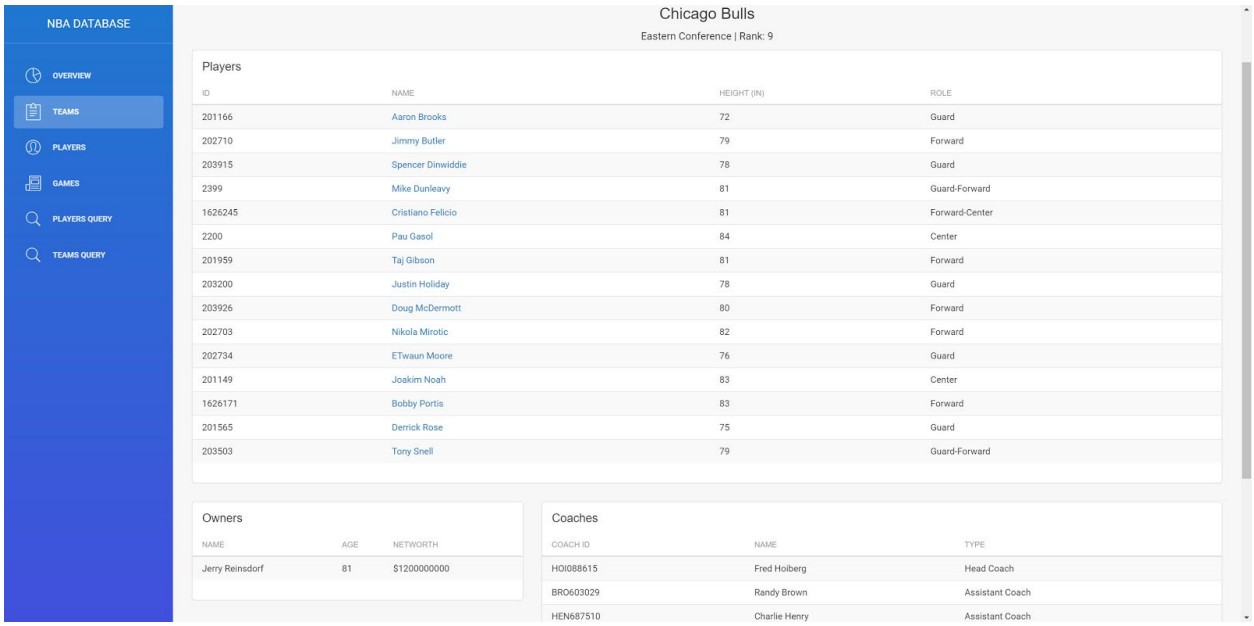


Application UI:

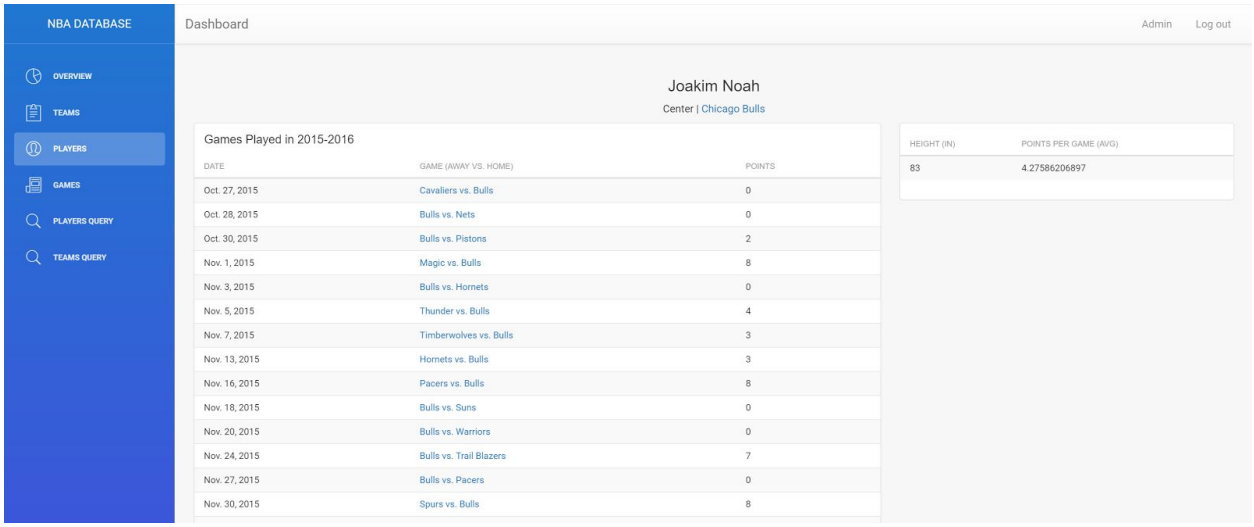
Overview



Team View

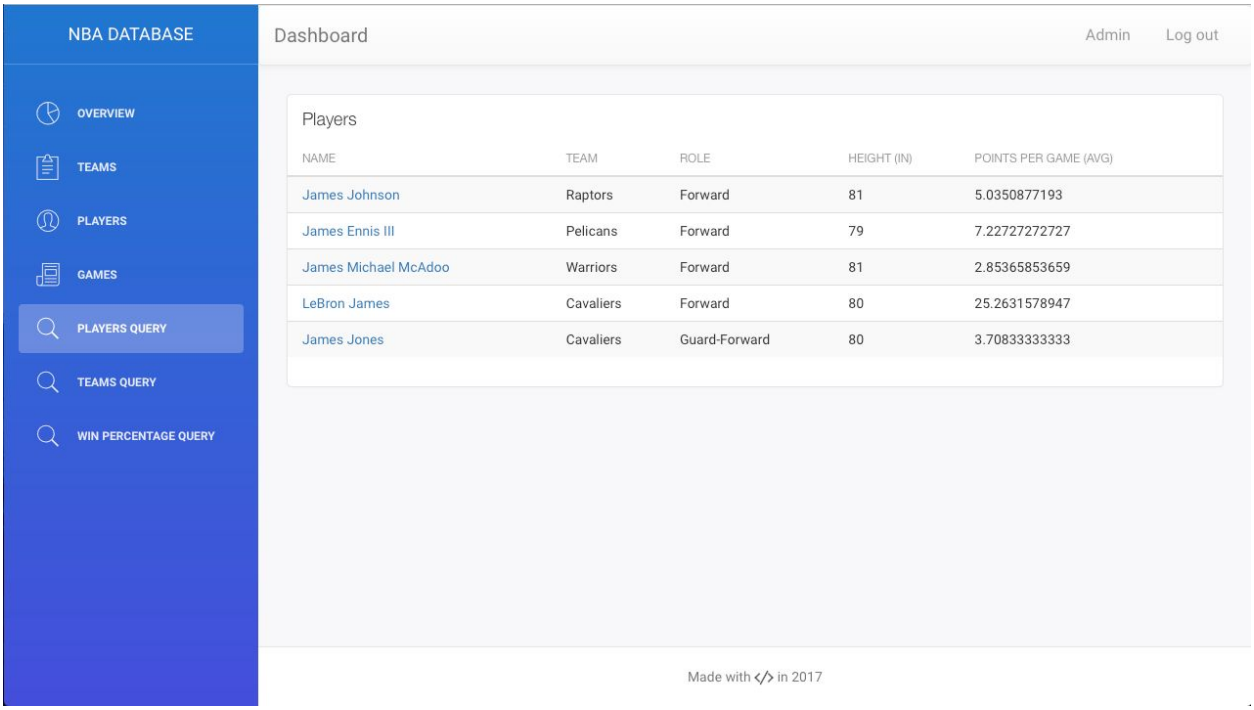


Player View

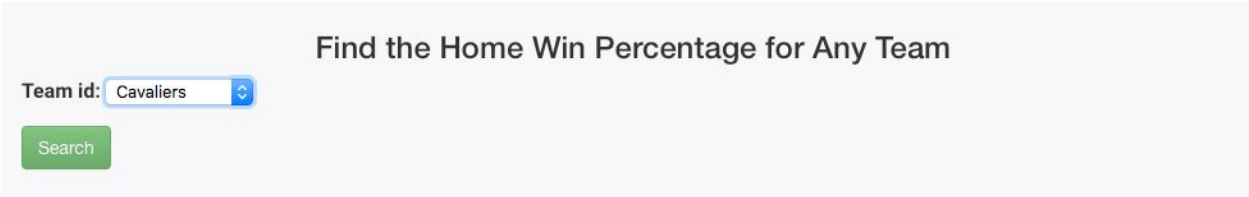


Query #1: With Sample Input

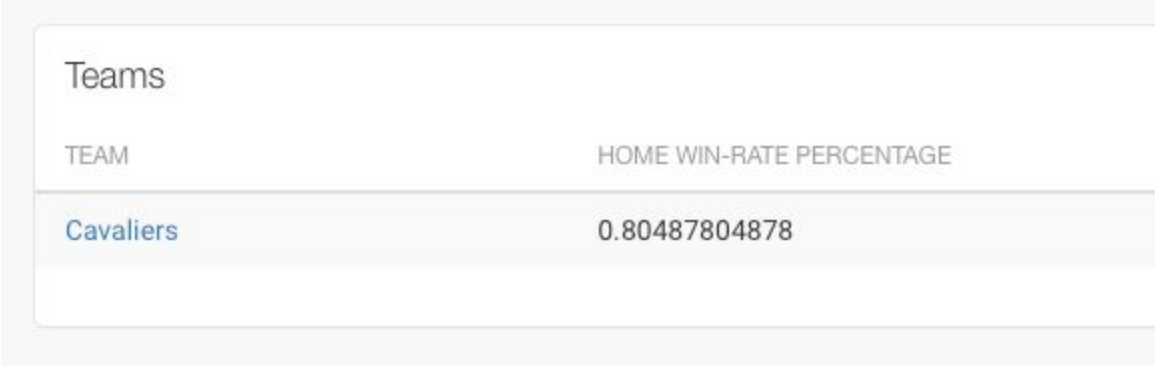
Results Returned From Query #1



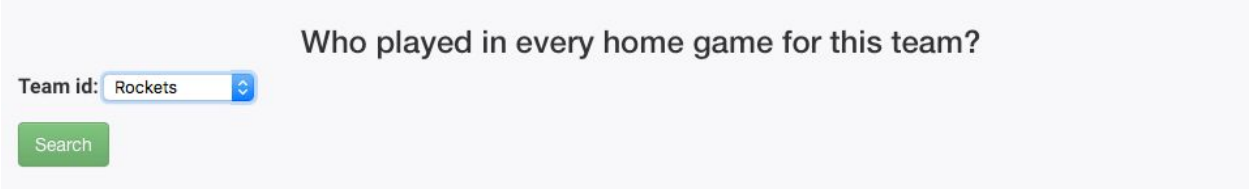
Query #2: With Sample Input



Results Returned From Query #2



Query #3: With Sample Input



Results Returned From Query #3