

NBA Analytics



Big **B**rainDB

By: Aayush Patel, Yash Tyagi, Mitesh Patel, Jeevan Maddila

1. Briefly describe what the project accomplished.

Our project was an analysis of data of players from the National Basketball Association (NBA) which was pulled from [Basketball-Reference](#). We visualized and displayed data that consisted of various statistics related to players, teams, and injuries. The goal of the project was to create a one-stop destination for a user to learn about any player currently in the NBA and to see how those players compared to others in a meaningful way. We also implemented a favorites and recommendation system that would offer the user a unique experience that doesn't really exist on any other NBA analytics websites.

2. Discuss the usefulness of your project, i.e. what real problem you solved.

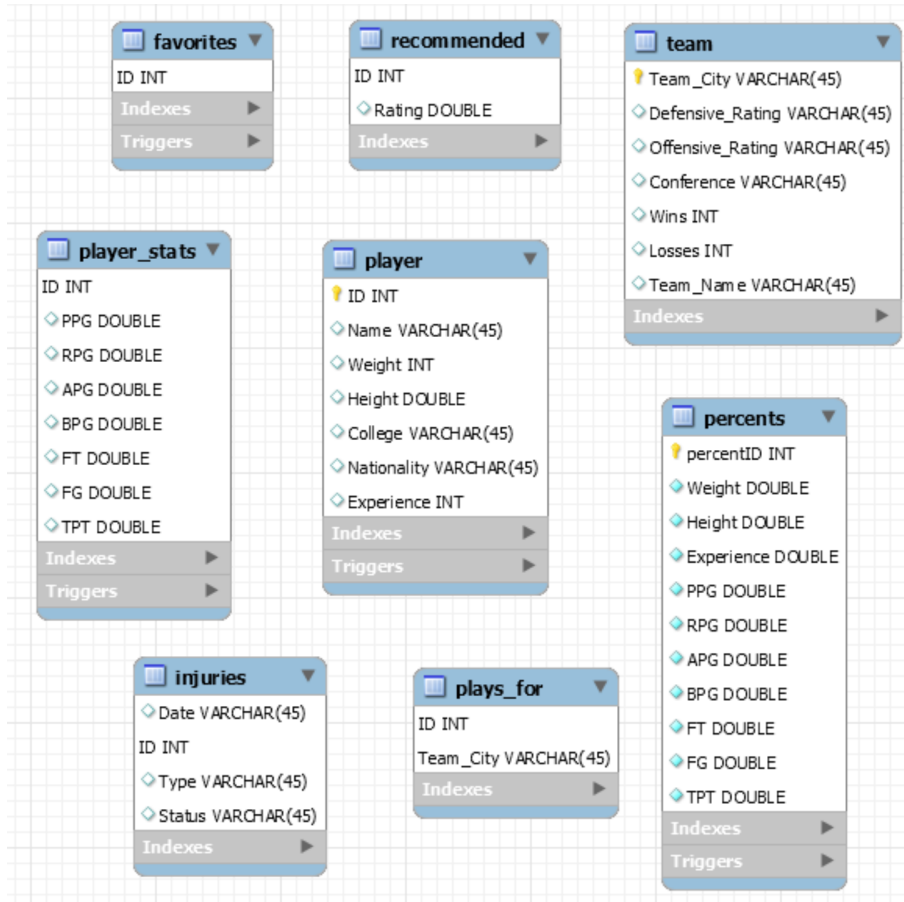
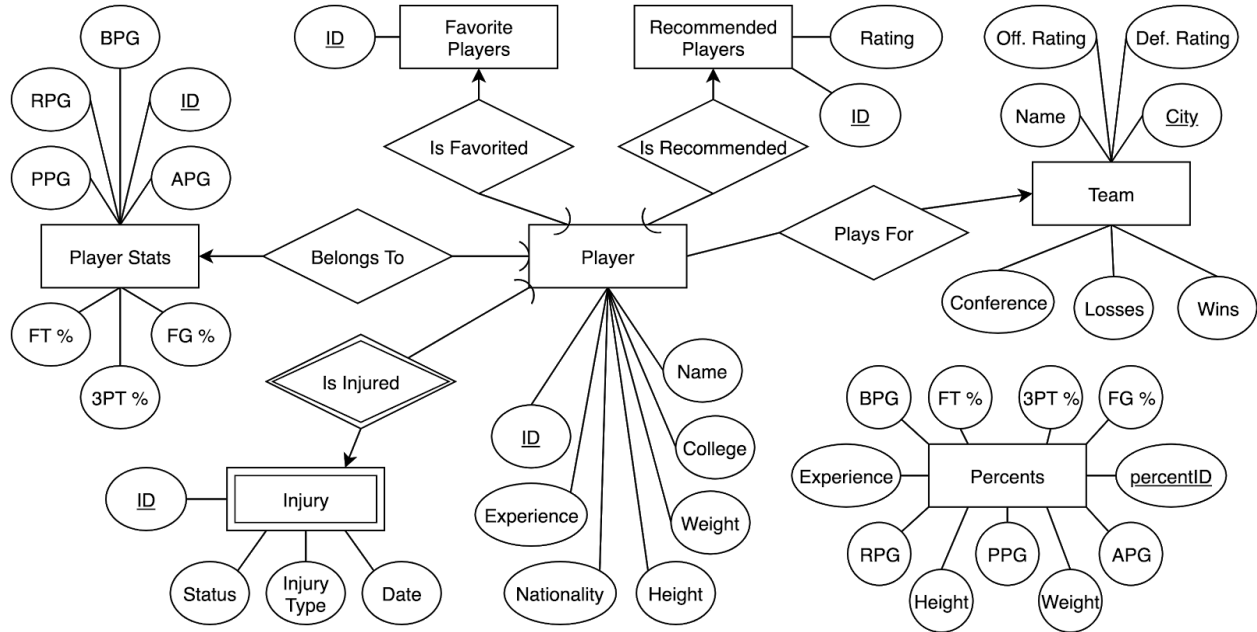
Our project is a way to analyze data that of a sport that millions of people watch. Because the NBA season lasts about three quarters of the year and has hundreds of players, we wanted to create a way that users could keep track of their favorite players and see other players they may like. There are not any websites that currently build a recommended players list based on a user's interests, which is why we decided to focus our app on showing this data.

3. Discuss the data in your database.

We pulled our data from the link provided above, and got all the players that currently play in the NBA. We also pulled statistics for all teams in the NBA, injuries players are going through, and the status of these injuries. We utilized MySQL to form our relational database and had the primary key be an auto-incrementing ID assigned when players are added to the database. The players table consisted of stats like weight, height, experience, nationality, college attended, and many other fields. The complete schema and ER Diagram can be seen below.

4. Include your ER Diagram and Schema

The DDL script to generate our database schema and data in it can be found on our github [repository](#) in [CS411-NBA/SQL/nbastats.sql](#)



5. Briefly discuss from where you collected data and how you did it (if crawling is automated, explain how and what tools were used)

All of your data was extracted from the website basketball-reference.com. In order to get the data in a clean format we used a [python API](#) that had the ability to process all the data and with a little work we were able to format the data in an advantageous way. In order to format the data we used the pandas and numpy packages in pandas since that allowed us to create different data frames and export those dataframes as csv files. From there we simply imported the data into MySQL and got to work.

6. Discuss how you used a NoSQL database in your project

We did not utilize a NoSQL database in this project, but we envision storing the shots players have made during select games, and displaying an analysis of the percentages of what location on the court they make the shots from. This could be a future implementation we may work on after this project.

7. Discuss your design decisions related to storing your app data in relational vs. non-relational databases.

Like mentioned above, the primary way of storing data was through our MySQL relational database. We had multiple tables (player, plays_for, teams, injuries, etc.) that we joined together in our queries (as needed) in order to access our data. The reason we did this is because the player, as expected, is the most important feature of the NBA. However, the player is related to the teams they play for, the status of their own injury (if they have one), their statistics, as well as their team. Because many of these features depend on each other and data would have to be pulled from different places, we decided to use a relational database. A non-relational database was not used for this project, but like mentioned before, we would use this to store all the shots of a player for each player. The reason we would do this is because there would not be a heavy relation, and there is not really a necessity of referencing the shots other than the shot map we would make. Additionally, this would be a large amount of data to store, which is where a non-relational database like MongoDB would be advantageous.

8. Clearly list the functionality of your application (feature specs)

We were able to get various functionalities incorporated into our website. They are as follows:

- 1) Basic Functions:

a) Create Player:

This function is for creating a player in the database. The player is added with the template-driven form, and then on the “create player” button’s click, the post service request is called and an INSERT INTO query is called using the user inputs from the form.

b) Read Player:

This function lets us view a certain group of players, teams, or injuries (depending on the page). When creating the datasource for our table, we call get service request which then calls a SELECT query. The get requests for certain pages may have an array of queries, which allows us to run different queries on different pages. We can view different permutations of the data by modifying the select statement.

c) Update Player:

The update function player is completed by prepopulating the form based on what data is already in the database for the specific player. The way that this information is saved is by reading the values from the table (which again, were rendered with the SELECT query) and storing it into a history state. This allows us to not have to query the database repeatedly and to see what values are needed. We also added form validation to the update form to make sure that the user inputs the data correctly. Once they click update, a put service request is called which calls an UPDATE query in the MySQL part of the code.

d) Delete Player:

We are able to delete a player by going to the same edit form above. In this form, we made it so that the user can click the delete player which removes the player given the ID. The ID is able to be pulled from the form because we directed the page routing to include the ID for the specific player that is being edited/deleted.

2) Tables

In the table, we were able to display all the useful information for our database tables in a clean way, as well as include action icons that users can click on to edit the page. This was done by using an Angular Material table, and allowed for us to make use of certain functions like the ones listed below:

a) Filtering





















We were able to do a keyword search on the table. We implemented this as it allows for a quick search of a player rather than trying to search the whole table for a certain player. We implemented this by including a function which updates on the key presses and that searches the table for the values inputted into the search box.

b) Sorting

We also implemented a quick sorting feature on our players table. This is helpful because the user can automatically order their results based on the information they want to see. It can also be done by any column which is equivalent to an ORDER BY ____ ASC/DESC query, rather than having to run a query for every permutation of data.

c) Pagination

Pagination allows for the data table to have a certain number of results per page of the table. We also created options for the user to choose from and look at the data by which means that the views were customizable. It also prevents the user from having to scroll to view the rest of the data, which provides the user with a nice user experience and interface.

Favorites	ID	Name	Weight	Height	College	Nationality	Experience	View	Edit
<input type="checkbox"/>	1	De'Andre Hunter	225	79	Virginia	US	0		
<input checked="" type="checkbox"/>	2	Trae Young	180	73	Oklahoma	US	1		
<input type="checkbox"/>	3	Vince Carter	220	78	UNC	US	21		
<input type="checkbox"/>	4	Cam Reddish	208	80	Duke	US	0		
<input type="checkbox"/>	5	Kevin Huerter	190	79	Maryland	US	1		
<input type="checkbox"/>	6	Bruno Fernando	233	81	Maryland	AO	0		
<input type="checkbox"/>	7	Damian Jones	245	83	Vanderbilt	US	3		
<input type="checkbox"/>	8	DeAndre' Bembry	210	77	Saint Joseph's	US	3		
<input type="checkbox"/>	9	John Collins	235	81	Wake Forest	US	2		
<input type="checkbox"/>	10	Brandon Goodwin	180	72	Central Florida, Florida Gulf Coast	US	1		

Items per page: 10 1 - 10 of 495 |< < > >|

Here you can see the table sorted by IDs and the pagination in the bottom right-hand corner. It also contains the checkboxes to add players to your favorites as well as the action icons for viewing/editing the player.

3) Teams

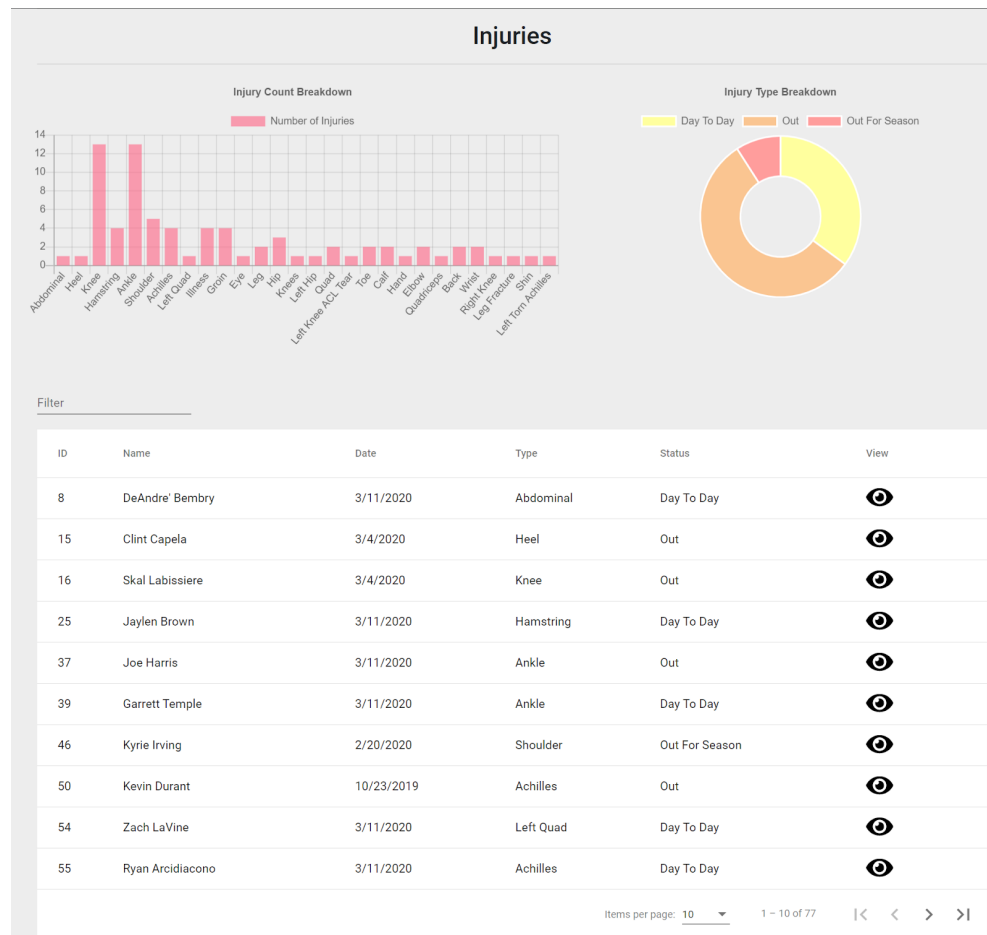
a) Team Roster

Team Roster: Atlanta Hawks		
ID	Name	View
1	DeAndre Hunter	
2	Trae Young	
3	Vince Carter	
4	Cam Reddish	
5	Kevin Huerter	
6	Bruno Fernando	
7	Damian Jones	
8	DeAndre' Bembry	
9	John Collins	
10	Brandon Goodwin	
Items per page: 10 1 - 10 of 16 < < > >		

Teams								
Filter								
Team Logo	Team City	Team Name	Defensive Rating	Offensive Rating	Conference	Wins	Losses	View Roster
	ATL	Atlanta Hawks	114.8	107.2	East	20	47	
	BOS	Boston Celtics	106.8	112.9	East	43	21	
	BRK	Brooklyn Nets	108.7	108.1	East	30	34	
	CHA	Charlotte Hornets	113.3	106.3	East	23	42	
	CHI	Chicago Bulls	109.8	106.7	East	22	43	
	CLE	Cleveland Cavaliers	115.4	107.5	East	19	46	
	DAL	Dallas Mavericks	110.6	116.7	West	40	27	
	DEN	Denver Nuggets	109.5	112.5	West	43	22	
	DET	Detroit Pistons	112.7	109	East	20	46	
	GSW	Golden State Warriors	113.8	105.2	West	15	50	
Items per page: 10 1 - 10 of 30 < < > >								

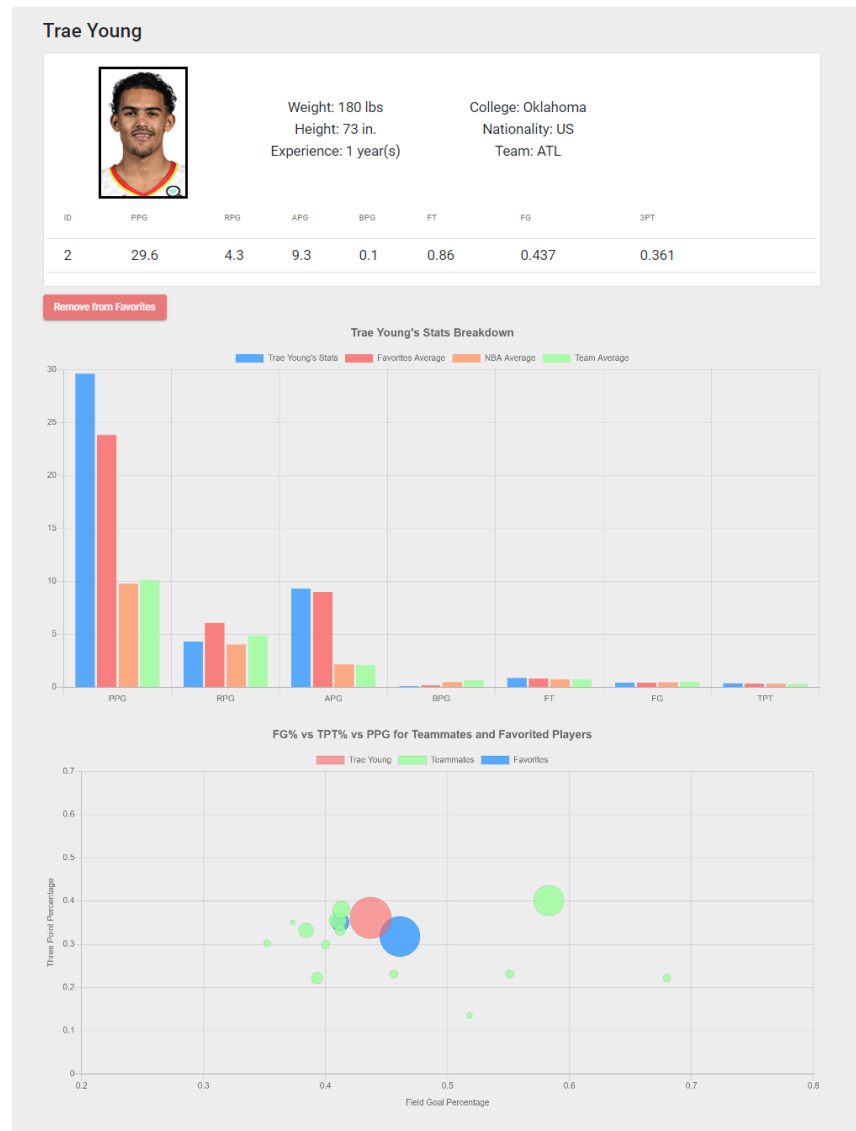
- i) The “Teams” page displays all the teams currently in the NBA, along with their Win-Loss record, the conference they belong to, and their offensive and defensive ratings. Users are also able to view team rosters by clicking on the corresponding button in the main teams table.

4) Injuries



- a) For the Injuries page, we have listed all players that are currently suffering from injuries. In each row we include their name, the date of their last injury status update, the type of injury, and their current injury status. Injury status can be one of three values: Day to Day, Out, Out for Season. At the top of the page we have included two graphs: one that lists counts of injuries per injury type, and the counts of each injury status for all injured players. This allows users to assess the condition of the NBA's injured players easily by looking at the two graphs instead of parsing the rows in the table.

5) View Player



- a) For View Player we allow the user to view all information about the player, this includes their personal information, stats, and any injury that they must have. To do this we wrote many select statements that take data from all tables they may be in and then get the data from there. We also wanted to add more functionality, so we created two graphs that might be useful for the user. The first is a simple bar chart which plots the player stats against, the NBA average, the users favorites average, and the player's team average. Again we used a couple of select statements to pull the information we needed and then use the charts.js package in order to create a good looking graph. We did something similar with our second

graph where we created a bubble chart that plotted a player's FG% vs their TPT% while using PPG to determine the size of the circle. We did this because we wanted to show how efficient a player while also highlighting how much they score. We did this very similarly to the bar chart with using multiple select statements to get the data and then use charts.js to plot it. For this we just plotted the player you are viewing, the players on the chosen players team, and all of the user favorited players.

6) Favorites

Note: Picture for Favorite page can be found in question 12

a) Weights for Recommendations

This function allows the user to put in their weights to generate their own custom recommendations. We provide 10 different sliders for the user to use, and they can select any combination of 10 while also placing any amount of emphasis they want on those sliders. Once the user submits this we activate a put request that contains an UPDATE query that puts all of the weights that the user entered into the precents table, this table is then used in a stored procedure to generate the recommendation.

b) Recommended

We generate the recommended players in a stored procedure (more explanation on this in question 12) that calculates the most similar players to the user favorited players using the percentages calculated in the last paragraph. After this is calculated we store all the results in a recommended table, and we use a simple select statement that outputs the 10 players who are the most similar to the favorited players and call them our recommended players.

9. Explain one basic function

One of the most basic functions that we can write is Create Player, and unsurprisingly this function is all about allowing the user to create a player and add them to the database. For this function we ran into the issue that we needed to input data into two different tables, but one of the tables needed an id in order to be successfully added to the table. This was a problem because we did not want the user to manually choose their id, but luckily we were able to fix this issue by using some HTML magic and filling in that record in the form for the user and not allowing them to change that field (a picture is in question 11 that shows this). Once this was done, we did not have a problem with this and we were able to successfully create a player and create their stats while also adding them into the player and player_stats table without any issue.

10. Show the actual SQL and NoSQL code snippet

```
router.post('/player/create', (req, res, next) => {
  db.query(
    'INSERT INTO player (ID, Name, Weight, Height, College, Nationality, Experience) VALUES (?, ?, ?, ?, ?, ?, ?); \n\
    INSERT INTO player_stats (ID, PPG, RPG, APG, BPG, FT, FG, TPT) VALUES (?, ?, ?, ?, ?, ?, ?, ?)',
    [req.body.ID, req.body.Name, req.body.Weight, req.body.Height, req.body.College, req.body.Nationality, req.body.Experience,
    req.body.ID, req.body.PPG, req.body.RPG, req.body.APG, req.body.BPG, req.body.FT, req.body.FG, req.body.TPT],
    (error) => {
      if (error) {
        console.error(error);
        res.status(500).json({status: 'error'});
      } else {
        res.status(200).json({status: 'ok'});
      }
    }
  );
});
```

11. List and briefly explain the dataflow, i.e. the steps that occur between a user entering the data on the screen and the output that occurs (you can insert a set of screenshots)

The screenshot shows a web form titled "Create Player". It is organized into two columns of input fields. The left column includes fields for ID (pre-filled with 496), Player Name (Abdu), Weight (160), Height, College (UIUC), Nationality, and Experience (5). The right column includes fields for PPG, RPG, APG, BPG, FT, FG, and TPT, all of which are currently empty. A blue "Add Player" button is located at the bottom left of the form.

Our data flow starts at the “Create Player” screen where the user inputs their desired attributes for a new player. Upon clicking “Add Player”, a POST request is made using the inputted attributes to the “Create Player” form.

```
createPlayer(player) {
  return this.request('POST', `${environment.serverUrl}/player/create`, player);
}
```

The POST request hits our request router, where the new player is inserted into the DB as follows:

```
router.post('/player/create', (req, res, next) => {
  db.query(
    'INSERT INTO player (ID, Name, Weight, Height, College, Nationality, Experience) VALUES (?, ?, ?, ?, ?, ?, ?); \n\
    INSERT INTO player_stats (ID, PPG, RPG, APG, BPG, FT, FG, TPT) VALUES (?, ?, ?, ?, ?, ?, ?, ?)',
    [req.body.ID, req.body.Name, req.body.Weight, req.body.Height, req.body.College, req.body.Nationality, req.body.Experience,
    req.body.ID, req.body.PPG, req.body.RPG, req.body.APG, req.body.BPG, req.body.FT, req.body.FG, req.body.TPT],
    (error) => {
      if (error) {
        console.error(error);
        res.status(500).json({status: 'error'});
      } else {
        res.status(200).json({status: 'ok'});
      }
    }
  );
});
```

The “Add Player” button, when pressed, redirects the user to the “View Player” corresponding to the new player’s ID. The “View Player” page makes a GET request using the new player’s ID, as follows:

```
getPlayerStats(playerID) {
  return this.request('GET', `${environment.serverUrl}/player/view/${playerID.ID}`, playerID)
}
```

This GET request hits the router, and the following query is executed:

```
router.get('/player/view/:ID', function (req, res, next) {
  db.query(
    'SELECT * FROM player_stats WHERE ID=?; \n\
    SELECT * FROM player WHERE ID=?; \n\
    SELECT * FROM injuries WHERE ID=?; \n\
    SELECT * FROM plays_for WHERE ID=?; \n\
    SELECT AVG(PPG), AVG(RPG), AVG(APG), AVG(BPG), AVG(FT), AVG(FG), AVG(TPT) FROM favorites NATURAL JOIN player P LEFT OUTER JOIN player_stats S ON P.ID = S.ID; \n\
    SELECT AVG(PPG), AVG(RPG), AVG(APG), AVG(BPG), AVG(FT), AVG(FG), AVG(TPT) FROM player P LEFT OUTER JOIN player_stats S ON P.ID = S.ID; \n\
    SELECT AVG(PPG), AVG(RPG), AVG(APG), AVG(BPG), AVG(FT), AVG(FG), AVG(TPT) FROM player P NATURAL JOIN player_stats S NATURAL JOIN plays_for F WHERE (SELECT Team_City FROM plays_for WHERE ID=?)=F.Team_City; \n\
    SELECT ID FROM favorites WHERE ID=?; \n\
    SELECT P.ID, FG, TPT, PPG, P.Name FROM player P NATURAL JOIN plays_for F LEFT OUTER JOIN player_stats S ON P.ID = S.ID WHERE (SELECT Team_City FROM plays_for WHERE ID=?) = F.Team_City; \n\
    SELECT P.ID, FG, TPT, PPG, P.Name FROM player P NATURAL JOIN favorites F LEFT OUTER JOIN player_stats S ON P.ID = S.ID',
    [req.params.ID, req.params.ID, req.params.ID, req.params.ID, req.params.ID, req.params.ID],
    (error, results) => {
      if (error) {
        console.log(error);
        res.status(500).json({status: 'error'});
      } else {
        res.status(200).json(results);
      }
    }
  );
});
```

This data is sent back to the “View Player” page which parses the returned tables as arrays, performs the necessary manipulation on the arrays in order to display them correctly in the UI components, and finally displays the new player with the values that were input in the beginning of our data flow, as seen in the image below.

Abdu

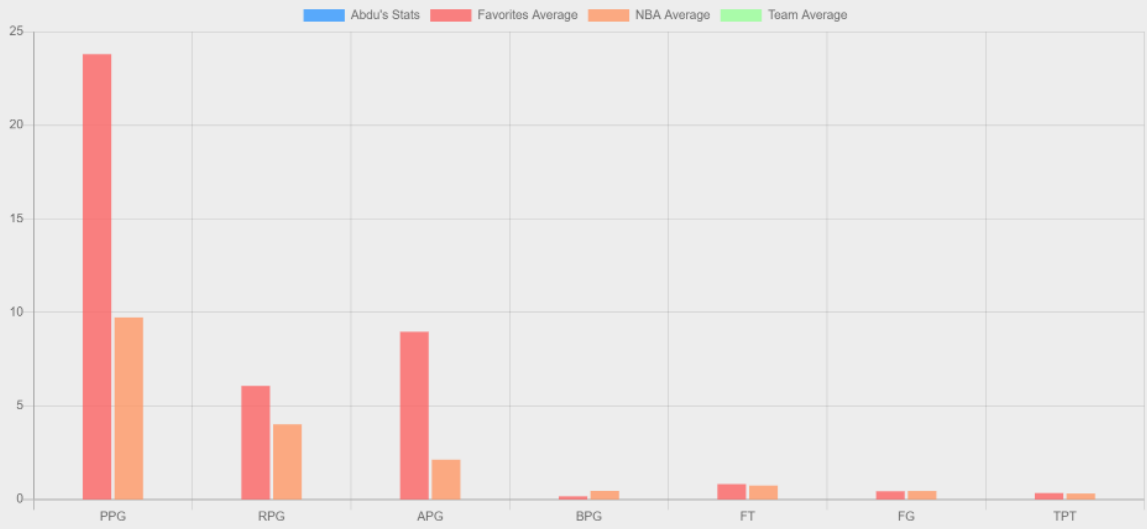


Weight: 160 lbs
Height: 0 in.
Experience: 5 year(s)
College: UIUC

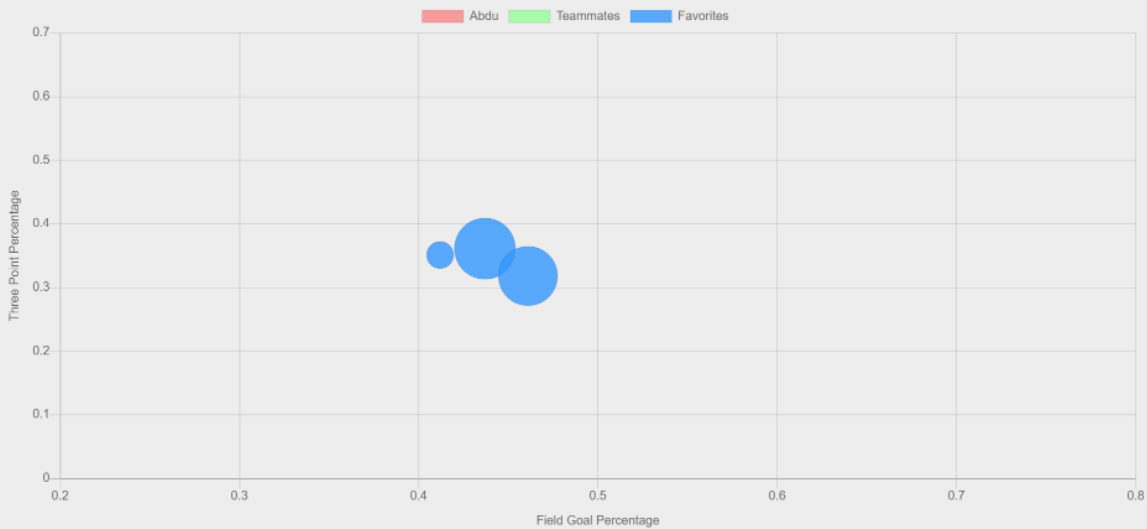
ID	PPG	RPG	APG	BPG	FT	FG	3PT
496	0	0	0	0	0	0	0

Add to Favorites

Abdu's Stats Breakdown



FG% vs TPT% vs PPG for Teammates and Favorited Players



12. Explain your advanced function 1 (AF1) and why it's considered as advanced. Being able to do it is very important both in the report and final presentation.

Our advanced function was a favoriting and recommendation service that allowed users to add and remove users to a favorite table, and then our function would be able to generate a list of 10 similar players that we could recommend to the user in order to find other players that the user may want to favorite. While this may not seem advanced, we provided many features that elevated this function. For one, in order to generate the recommended players we let them create a litany of options, and allow the user to generate what features they find important, and then determine how important these features are. For example, the user could choose to find recommended players with a filter of 80% Weight and 20% PPG, or a filter of 30% RPG, 20% FG, 50% Experience, or any combination of the 10 variables we included. Once we get the filter from the user we run a stored procedure which runs a formula that weights all of these variables and generates a rating for every single player that would allow us to find the recommended players. On top of this, we wanted to ensure that this recommendation list would get updated constantly, so we created triggers that would re-run the procedure any time the user would update, add or delete a player or its stats, and everytime a player was added or deleted from the user favorites list. We feel like this is very important because this would allow the recommendation list to change without a retry in the sliders from the user, which is helpful because if the user wants to change their favorites but not change their filters they could do that very easily. We believe that this is an advanced function because we wrote a stored procedure and multiple triggers that really tested our knowledge of the course, and we had to dynamically create a formula within the stored procedure that worked for any input from the user. Also, we believe the backend work needed to create favorites and easily add and remove players from favorites is also advanced. In whole, we believe that our favoriting feature along with using stored procedures and triggers in our recommendation system makes our function advanced.

Below I have included a picture for how our advanced function looks. Note how we also included a graph that allowed the user to visualize the weights they were inputting, and also the ability for the user to remove or delete favorites from this page.

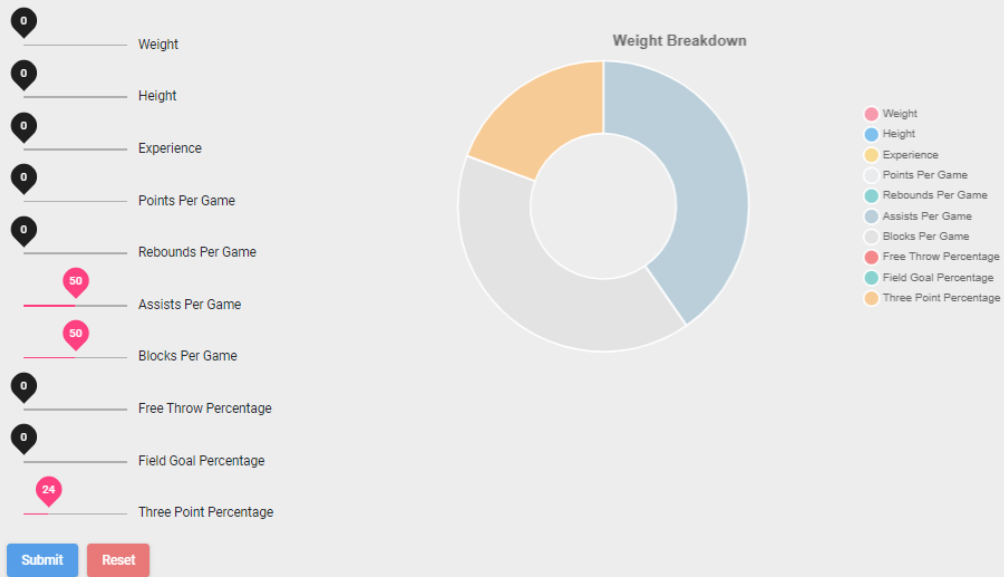
Favorite Players

ID	Name	View	Unfavorite
2	Trae Young		
108	Luka Dončić		
385	Ricky Rubio		

Items per page: 10 1 - 3 of 3 |< < > >|

Weights

Adjust the Weights Below According to the Priorities You Would Like to Set for Your Recommended Players



Recommended Players

Favorite	ID	Name	View
<input type="checkbox"/>	69	Devonte' Graham	
<input type="checkbox"/>	190	Malcolm Brogdon	
<input type="checkbox"/>	449	Kyle Lowry	
<input type="checkbox"/>	400	Damian Lillard	
<input type="checkbox"/>	335	Chris Paul	
<input type="checkbox"/>	236	Ja Morant	
<input type="checkbox"/>	170	Russell Westbrook	
<input type="checkbox"/>	34	Spencer Dinwiddie	
<input type="checkbox"/>	371	Ben Simmons	
<input type="checkbox"/>	383	Devin Booker	

Add to Favorites

13. Describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or were to maintain your project. Say you created a very robust crawler - share your knowledge. You learnt how to visualize a graph and make an interactive interface for it - teach all! Know how to minimize time building a mobile app - describe!

There were many technical challenges that we faced throughout the development of our app. Among all of them however, the one that gave us the most problems was our recommendation system. The role of this system was to use the favorited players to recommend similar players to the user. To allow the user more control, they were allowed to set weights on the importance of each statistic. Through the use of several triggers, a stored procedure would be called to calculate similarity ratings for all players in the database. The ten players with the best similarity rating would be then displayed to the user. Since mostly all of this was completely done on MySQL workbench, everything from learning how to create a complex stored procedure to learning the many restrictions and errors of MySQL was a challenge. In lecture and homework, we were only taught fairly simple procedures and they were incomparable to the one that we had to write. We started by writing out most of our procedure before testing it. This was a lesson learned because there were a wave of errors. There was some syntax highlighting that helped us a bit, but we mostly debugged by using the MySQL command line client locally before connecting the app. The first problem we had was with case sensitivity. We learned that the language did not differentiate between our lower and upper cased variables. We also had issues with the data types that our algorithm was using. Since we were dealing with decimals and integers together while doing mathematical operations on them, sometimes an operation would truncate the decimal. To solve this, we had to change all of our variables to doubles. Once all the minor errors were solved, we started to face bigger issues. Another problem we faced was with the client thinking that the called procedure had to return something. In reality, the procedure just had to run in the background internally and update the recommended table. We specifically did not want a return value. After some research, we learned that although the database shows an error, the procedure still runs just fine. The error was actually more of a warning. We also discovered some issues rooted in the fact that we had both NULLs and 0's in our data. This differentiation was important in other parts of the app, but for the recommendation they had to be treated the same. We figured out something was wrong because our averages were skewed. SQL skips NULL values and doesn't include them in averages whereas 0's still count. After solving a couple more issues we had finally gotten the procedure to work when manually called from the command line client. The next step was integrating it with the triggers and our app. Adding the triggers was fairly simple so then we went ahead to deploying the triggers and procedure to the app. We had done this without testing the triggers

locally. We immediately faced MySQL error 1422. There was an implicit commit happening in our procedure and we were not sure where. We had based our procedure loosely off the examples in lecture, specifically the part where an existing table is dropped before being recreated and repopulated. After doing some research, we learned that both DROP TABLE and CREATE TABLE both cause implicit commits which we were using. Instead our idea was to just use TRUNCATE TABLE but this also caused an implicit commit. With the way we wanted the recommendation system to work, we had to clear the table. Our only way of doing so was to add a cursor loop and use DELETE on each tuple instead. After doing that, everything started to fall into place and work. In the process of all this we learned a lot about MySQL. When getting unexpected values in a query or procedure, the best way to check what is going wrong is by using the workbench or the command line client locally. When there is an error thrown, one of the best things to do is google it. The official MySQL developer docs are always reliable. For any other needs, Stack Overflow always has you covered.

14. State if everything went according to the initial development plan and proposed specifications, if not - why?

We were very fortunate that we mostly were able to stay on track from our initial development plan, but as we expected we did run into a couple hurdles. The first hurdle, and the biggest hurdle was the coronavirus (obviously) because that did not allow us to work in person and forced us to work remotely for the majority of the project. Not giving us this face to face interaction made it a lot harder to be efficient and also we had moments where some of us had shady internet which created some connectivity issues that would halt work on our project. This was one of the main reasons we had to drop our second advanced function. Originally, we had planned to create a shot map in Mongo where we could show where a player was efficient in shooting in various places on the court. We thought this advanced function would be really cool, and it was very hard for us to decide to drop this from the webpage but ultimately we believe this let us create a better webpage as a whole.

15. Describe the final division of labor and how did you manage team work.

We used the LiveShare feature(essentially Google docs but for code) in VSCode in order to collaborate on the same project without having to deal with messy merge conflicts and other similar issues. This allowed us to implement different features in parallel and were immediately able to view the results in our app which greatly improved productivity. It was fairly simple to split up tasks for each feature, since there was a clear distinction between the work needed to be done on the backend on what needed to be done on the frontend in order to implement the feature. For example, when implementing the “View Player” page, we were able to divide the feature into two sections: setting up the UI and frontend to receive and display the data queried from the backend, and setting up the backend to query the DB correctly and send the data to the frontend. Since these tasks were in disjoint files, LiveShare enabled us to work on each part in parallel and view the results instantly without dealing with git. Mitesh and Yash primarily worked on frontend UI, Angular component work, and data processing while Aayush and Jeevan worked on DB query routing, HTTP request routing, and the DB schema setup.