# CS 411 Final Project Report

1. **Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**

   Overall, our final project doesn't differ that much from the original proposal as we did implement an NBA guesser game for users to guess a specific player given hints. These hints are based on aggregate player statistics, team wins, and boxscore season highs and averages, which we also mentioned in our original proposal. We did have a couple of changes and those are listed below:
   - We added a prediction feature that allows users to input two teams and a given year and the feature predicts who would win in a head-to-head matchup based on which team won more head-to-head matchups for the inputted year, which team won more games that year, and which team had the higher point differential
   - The game doesn't include a generate button to generate a new player to guess for, but user can simply reload the page to generate a new player
   - The game also doesn't include a give up button but they lose after 10 guesses
   - The game does incorporate retired NBA players but only those that have retired recently and have played at least one game in the last three years
   - The UI mockup for the game as shown on the project proposal differs slightly from that of the final project

2. **Discuss what you think your application achieved or failed to achieve regarding its usefulness.**

   We think our game is a useful and fun way for our users to test their NBA knowledge. Although the game can be difficult to win, we think it's a great indicator for fans to really see how much they know about the NBA today as the guessing game is only based on data from the last three seasons.

   We wanted our game to be different from other NBA guessing games that are popular online and we achieved this by including hints that are not found in other games (aggregate player statistics, team wins, and boxscore season highs and averages) and a sophisticated prediction feature.

   Furthermore, our prediction functionality can help users gain an understanding of which teams will win in a head-to-head matchup, which can be useful when trying

to gain NBA knowledge (ex: a new fan can figure out which teams are good, and which are bad).

3. **Discuss if you changed the schema or source of the data for your application.**

   No, we did not change the scheme or the source of the data for the guesser game. As mentioned in our project proposal, we sourced all of our data from the NBA API.

4. **Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**

   No, we did not change our ER diagram or our table implementations. There are no differences between our original design and the final design.

   From Stage 5, how we would like to improve our database design for the future:
   - One change in the database that would have helped the overall understanding of each entity would be including a quality of the team such as team name or abbreviation for each tuple. Currently, because the team name attribute doesn't exist, we are required to join with Organization since organization has a name attribute to find the particular team id for a given team name.
   - Additionally, in our initial design of the application and database, we expected to include a much larger range of years, meaning it was necessary to have an organization and team table to discern organizations that changed their name over the course of their existence. In reality since we were only using data from the last 3 years (inclusive), we could have limited ourselves to the **Team** table.
   - Boxscore should include the team_id of the team the player played for during the game. This is very useful, especially when considering a player can be traded and play for multiple teams during a season.
   - We considered many indexing options during earlier stages, so we are satisfied with our system optimization.

5. **Discuss what functionalities you added or removed. Why?**

   From our proposal, we introduced functionality to predict the winner of an NBA game. We also included many CRUD operations including examining player

attributes, adding/deleting organizations, and updating/inserting the most recent NBA games available from the NBA API. We even created a trigger to ensure the validity of these newly inserted games since the NBA API is external (not something we created).

From our last stage, we increased the complexity of our prediction function by increasing the parameters from 1 to 3 (it previously only looked at head-to-head matchups, now it also looks at total wins and point differential). We also added more hints as well as a text box and guessing functionality for our guessing game.

6. **Explain how you think your advanced database programs complement your application.**

   For our application, we had four advanced database queries, each of which serve a purpose either on the guesser game page or the predictions page.

   The first advanced query, which describes the game where a player had their season high points given a year, is used as one of the hints for the guesser game. We simply display a certain player's max points and the user must guess that player.

   The second advanced query, which describes the number of team wins in a season, is also used as a hint for the guesser game. For the player that the user must guess, we find what team they played on during a particular year and display that team's wins as a hint.

   The third advanced query, which describes a player's double-double rate, is also used as a hint for the game. Unlike the previous two queries, double-double rate is encompassed in a stored procedure that loops through all games a player played in a particular year and finds the amount of games the player had double digit points and assists and the amount of games they played. The stored procedure then calculates the double double rate by dividing these two values, which is displayed on the guesser page as a hint.

   The fourth advanced query, which describes who won more head-to-head matchups, is used as a parameter for our prediction feature. It also is encompassed in a stored procedure that finds how many matchups team 1 won against team 2 and if team 1 won over 50% of matchups, they are declared

winners. This, along with other parameters, are used to come up with a prediction that is displayed on the predictions page.

Along with the two stored procedures, we also had a trigger when updating/inserting new games that are available. We are pulling these games from the NBA API and while we trust this API, we use a trigger that includes safeguards when inserting new games to ensure the validity of the data used in our game. Our trigger handles this by checking a condition before new games are inserted and it doesn't insert them if their date is invalid.

7. **Each team member should 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 where to maintain your project.**

   Anish: Some technical challenges the team faced were rendering different pages for specific parts of the game: guesser, prediction, miscellaneous as well as passing sql results from server.js to our ejs files to be displayed. Getting more familiar with Node JS/express (app.get/post, connection.query, res.send/render, etc.) and understanding how the server.js file connects with our ejs files helped us dynamically use input data for our SQL queries and then post that information using ejs tagged variables on the user interface.

   Mehar: One technical challenge the team faced was creating the database schema and actually populating the tables. Firstly, we needed a way to relate Games to BoxScores to Players to Teams etc. Because we had quite a few entities, designing the UML diagrams was time consuming and forced us to carefully analyze every association between objects. Further, since we were using an external API for retrieving NBA statistics, we were limited by the information that API had. Lastly, we had to debug connection issues with the NBA API. This was particularly a problem when getting large amounts of data, for example when we first populated our tables. We frequently saw timeout errors that required StackOverflow solutions to fix.

   Ojas: A technical problem that I repeatedly encountered was connection between the backend python scripts and the front end javascript implementation. In order to update games or create the entire guesser program, I wrote the code in python since I was more familiar with the language. As a result, I needed to find a connection between nodeJS and python. This was more difficult than running all our database commands in nodeJS itself since the output from the python script

was returned as a nodeJS event, which we didn't have much experience using. Since it was an event, and the event we chose to look at was `stdout`, it would print out all output from standard-out. The solution was to only print out necessary information from the python script that would be used in the front end. Unfortunately, this made it difficult to log errors or info in python but solved our prevailing problem.

8. **Are there other things that changed comparing the final application with the original proposal?**

   No, nothing else has been changed.

9. **Describe future work that you think, other than the interface, that the application can improve on**

   Other than the interface, we'd love to improve on the prediction algorithm by using regression analysis on the parameters to fine-tune the weights, or by introducing a feedforward network to train a prediction model.

   For the guesser, we can find new hints to give users. We can also take in user feedback to understand if our game is too easy or too difficult, which could influence the new hints we provide.

   Finally, we believe our application can be improved by storing user information and having a login. With this feature, users can check their performance history. Furthermore, we can have a streak-based feature that tells users how many guesses in a row they've gotten correct to make the game more enjoyable!

10. **Describe the final division of labor and how well you managed teamwork.**

    Anish: Built basic front-end structure of the game (rendering different pages on different addresses, navigating between pages with buttons), designed the predictions page, CRUD operations, SQL queries for obtaining team and year information for stored procedure on predictions page
    Mehar: Designed the UML diagram, the SQL schema, created advanced queries, stored procedures, triggers, and the prediction algorithm
    Ojas: Created python parsers to populate SQL tables, performed indexing analysis on SQL tables, wrote back end code for both update games crud operation and main guesser application.

The teamwork was managed well since we communicated frequently, set times for group meetings, and divided future deliverables evenly.