Progress Report #2

We have made several modifications to our database design and overall organization since the first milestone:

- Rather than users competing head-to-head, each user scores points for the week, and those scores accumulate across the course of the season. The user with the highest total score at the end of the season is the league champion. This helps eliminate problems like randomizing matchups, dealing with an odd number of users, and handling small leagues where players might compete head-to-head more than once.

- Player scores will be computed from a gamma distribution. After all of their data is scraped from UltiAnalytics, gamma distributions for each stat (goals, assists, blocks, catches, completions, throwaways, drops, and Callahans) will be created in Python. Next, we will randomly pull values from each of these statistics based on the given gamma distributions 16 times, once for each week of the season. For each week, the randomly chosen values for each statistic will be combined based on Jeffrey's algorithm from Milestone 1 to find each player's total weekly score. Thus, we will generate a data structure in Python that looks something like this:

Player	**Week**	**Goals**	**...**	**Callahans**	**Score**
Jonathan Nethercutt	1	4	...	0	63.8
Jimmy Mickle	1	6	...	1	79.0
Dylan Freechild	1	5	...	0	52.9
...	...	...	...	...	...
Jonathan Nethercutt	16	2	...	1	51.3
Jimmy Mickle	16	6	...	1	67.7
Dylan Freechild	16	3	...	0	88.3

- This data structure in Python will be translated into a table in SQL. Thus, the results of the entire season will be known to the database, but not the users. Only when the users increment to the next week will that week's stats be revealed.

- Because of these changes, many new SQL statements were written. These include statements creating the new tables and queries to access data from the different tables.

- For our project, we chose to make a web based application with a Flask backend and PostgreSQL database. We chose to use Flask because of its relatively easier implementation compared to other backend frameworks such as Java Spring, and our team's familiarity with Python helped as well. Furthermore, it was helpful to view the sample beers web application to help guide us with our own project. To integrate our application with our database, we used a variety of libraries, including Flask Migrate and SQLAlchemy. We were able to successfully set up some sample tables this way. For our front end, we are using Bootstrap to stylize our pages.

Since Milestone 1, we have updated the tables as well to be more concise and be more compatible with the data that we decided to collect. We got rid of extraneous entity sets such as games and fantasygames. Using what we collected from Ultianalytics we were able to set up a backend database and populate our fantasy database which has 6 relations (*Users*, *Scores*, *Week*, *Players*, *Performs*). As of right now, we don't have any sort of password implemented on our front end side of things so we got rid of the password attribute from *Users*, but may add it back as we see fit in the future. *Users* get a weekly fantasy score which is stored in *Scores* which is identified by the UserID and the weekNum keys. *Week* is just a table that stores weekNum as its only attribute. *Rosters* is the relationship set between *Users* and *Players* as one user can have multiple players, but one player can only be assigned to one user. The *Rosters* relation holds the activeStatus attribute for whether a specific PlayerName is active for a given week. *Players* stores the average statistics regarding a specific player gathered from their past performance in the 2018 season. That way users can see whether a player will be worth drafting at a glance. Additionally, a player will have a RosteredStatus that indicates whether the player has already been drafted or not. Similar to *Scores*, *Players* have a performance each week and that is encapsulated in the *Performs* relationship set, which holds the weekly statistics.

New E/R Diagram

.