Group 27
Members: John Geyer, Juan Padilla, Omar Paladines, Max Shestov
GitHub: https://github.com/johngeyer/CIS550Project

In this phase, you will set up the sharing environment, explain your project idea in more detail, discuss the technologies to be used, and assign responsibilities to each group member. You should meet with your project TA, and then do the following:

1. Setup Subversion/Git to share source code and starter data files. See http://www.seas.upenn.edu/cets/answers/subversion.html for details, and be sure that whoever sets it up grants access to everyone in the group. You should also add your assigned TA and Professor to it so we can see what you are doing.
2. Write a document containing the following:
    1. Motivation for the idea

Our idea is to provide a baseball analysis and statistics reference tool. In particular, our application will support complex queries/splits information so that users can request to see player and team statistics over specific time periods and seasons. Additionally, users will be able to search for players based on statistics, for example “show me all players who have over 150 strikeouts and less than 20 home runs, sorted by batting average.” This functionality is meant to be similar to Baseball-Reference’s Play Index (but without the $30 annual fee). Also, we will provide a prediction tool (based either on linear regression or machine learning) that can be used to predict future player performance (for example, a user will be able to see predictions for each player’s statistics for the upcoming season). This functionality will rely almost exclusively on the relational database. Player biographical data will also be gathered from the Lahman database and incorporated on each “Player Page” (see below).

In addition to the tools described above, we are going to incorporate salary information into our application as well. Interestingly, due to the high number of trades that occur in the MLB, there are dozens of players who play for one team but are getting paid by several others (and there are also some players who aren’t playing at all yet still getting paid! See Prince Fielder for example…). Salary information is available online on websites like Cot’s Contracts, however, it isn’t presented in a very convenient way. Our web-app aims to fix this. We plan to use the Neo4J database to represent/store player salary information, with

nodes being teams and players, and an edge with weight X representing “Team A pays Player B Amount X.” We think that we may be able to provide a cool visual representation of the complex salary connections in the MLB using this database. At the very least, we want each “Player Page” to contain not only the player’s statistics and predictions but also his salary information (amount from each team, what percent of the team’s payroll he comprises, etc.).

2. Description of the complementary sources you intend to use for data, and how you intend to ingest the data into your database

To obtain player statistics (hits, home runs, RBI, strikeouts, etc.), we plan to use the play-by-play data available from Retrosheet.com. Our play-by-play data will come from the so-called Retrosheet Event Files spanning the years 1952-2016. These event files (.EVN or .EVA extensions) were easily downloaded from Retrosheet.com, and I used Retrosheet’s provided parsers (bevent.exe, bgame.exe) to parse the files and pipe the results into CSV files, one file for each year. Our plan is to then clean the CSV files using R (we are more comfortable with R than Python), then read these CSV files into our relational database (we plan on using the Oracle database provided to all engineering students, i.e. the database we used during the second homework). “Cleaning” will consist of extracting just the outcomes of each play (home run, double, strikeout, walk, etc.) and removing all the other extraneous details in the event files--there are 150 (!!) total fields for each event, and we are really only concerned with a handful of them. In addition, cleaning will involve modifying the files so that they match the relational schema provided in this milestone. Currently, the CSV event files have already been extracted and are stored locally on John’s computer (they are too large to push to the Git Repo).

We also have game-by-game data which also came from Retrosheet and was extracted and put into CSV files in the same way as above. These files contain game-level information (aggregate team statistics for the teams involved rather than player-by-player statistics like the event files). These will be useful for team-level data as opposed to player-by-player data and will be used on our “Team Pages” on our web application. Again, we plan on reading these into R, cleaning them, then importing them into our relational database according to the schema given in this milestone. Currently the unprocessed CSV files are locally stored on John’s computer (they are too large to push to the Git Repo).

Player biographical data (height, weight, etc.) will be obtained through the Lahman database (which can be downloaded as a CSV) and incorporated into our relational model. The CSV files are again currently stored on John's computer since they are too large to put in the git repo

The payroll/salary data is available at Cot's Contracts. Unfortunately the data is spread out over dozens of Google Sheets files. Our plan is to write a bash script which will pipe these files into CSVs similarly to how we handled the Retrosheet data, which we can then combine and clean using R. We are in the process of learning how to use Neo4J, so I can't give any specifics at the moment about how we plan to go from CSV into Neo4J. We have assigned this as a responsibility to some of our group members (see below) to figure out over the next week or two.

As potential extra credit, we may try to incorporate a feed from ESPN or one of the other major sports websites about any games currently in progress. Additionally, we may try to create usernames/account information and the ability to have customized settings (i.e. which stats you want presented, or what the default homepage is). These last things will only be completed if time permits.

3. Relational schema and description of NoSQL component.

The NoSQL component will use Neo4J to store information about payroll and player salaries. We intend to represent players and teams as nodes with a directed edge from Team A to player B with weight W if team A pays player B an amount W dollars as part of the player's salary. This way we can easily collect data about team payroll and spending, as well as display which teams pay which players. The schema for this relationship will be Pays(teamid, playerid, amount, year).

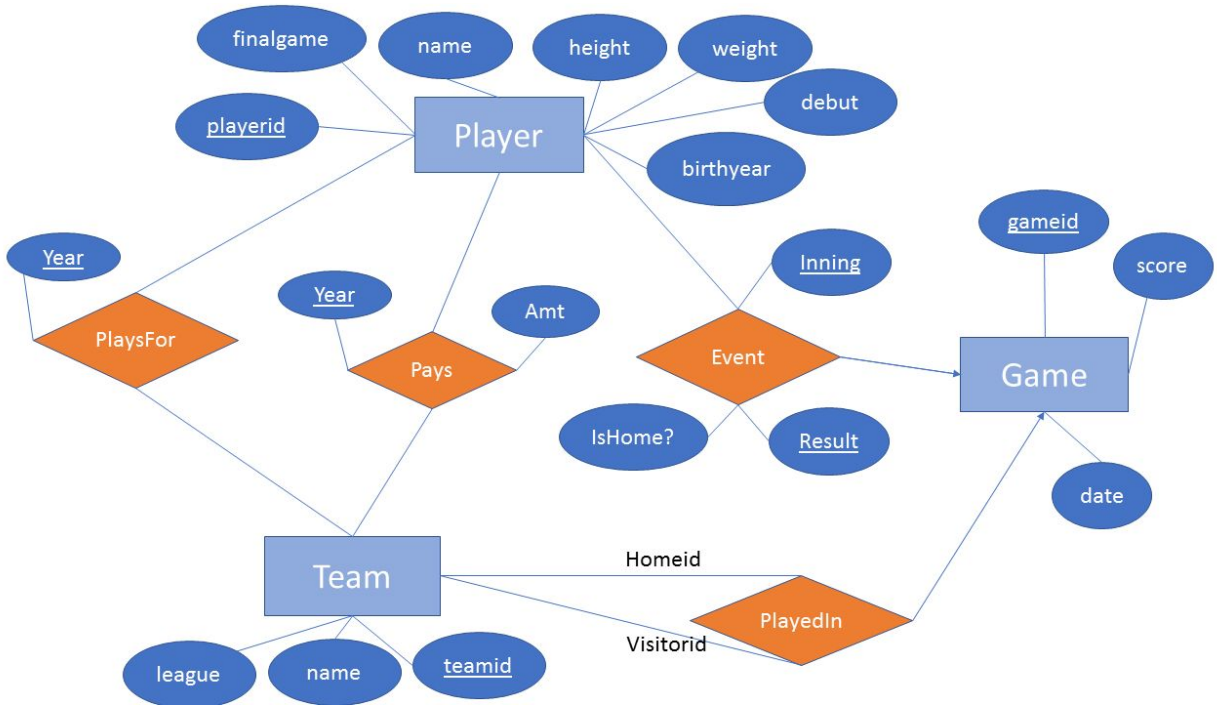
Each team will have a unique team id and a name, encompassed in the Team entity: Team(teamid, league, name). Each player will have a unique player id, name, and biographical information which will be encompassed in the Player entity: Player(playerid, name, height, weight, birthyear, debut, finalgame) where the playerid in this case will correspond to the Retrosheet playerid (not the Lahman database playerid--matching up the data will be taken care of in the cleaning of our data). Similarly, the teamid will be the Retrosheet teamid, not the Lahman database id.

A player can play for a team, which will be represented by the relation PlaysFor(teamid, playerid, year) (because players often play for the same team multiple times, and many players will only play for one team per year, we think this schema will actually be more space efficient than using the schema (teamid, playerid, from, to)).

A player's statistics will be computed from the Event Files and thus we have the table Event(playerid, gameid, isHome, inning, result) where isHome is true or ("T" or 1 depending on how we choose to implement it) if this is a home game. Because we hope to provide splits for arbitrary date ranges, we may aggregate these events by game/date instead of having multiple tuples for each player for each game. However, for now we plan to leave the schema as it is defined above so that we have more freedom in how we define our splits-tools.

We plan to have materialized views for a team's roster (so that we don't have to recompute it every time) for any given year (this will be built upon the PlaysFor relation).

Lastly we will have a relationship for the Games, which contains information about the winner and score: Game(gameid, date, score) and a relation PlayedIn(gameid, visitorid, homeid) where homeid and visitorid are foreign keys referencing the Team table.



This leads to the following DDL:

```
CREATE TABLE Team (
        Teamid VARCHAR(10),
        Name VARCHAR(20),
        League VARCHAR(10),
```

```
        PRIMARY KEY(Teamid)
)
CREATE TABLE Player (
        Playerid VARCHAR(20),
        Name VARCHAR(10),
        Birthyear INTEGER,
        Height INTEGER,
        Weight INTEGER,
        Debut VARCHAR(20),
        Finalgame VARCHAR(20),
        PRIMARY KEY (Playerid)
)

CREATE TABLE Game (
        Gameid VARCHAR(20),
        Score VARCHAR(10),
        Date VARCHAR(10),
        PRIMARY KEY(Gameid)
)
CREATE TABLE PlaysFor(
        Playerid VARCHAR(20),
        Teamid VARCHAR(20),
        Year INTEGER,
        FOREIGN KEY (Playerid) REFERENCES (Player.Playerid),
        FOREIGN KEY (Teamid) REFERENCES (Team.Teamid),
        PRIMARY KEY (Playerid, Teamid, Year)
)
CREATE TABLE Pays (
        Playerid VARCHAR(20),
```

```
        Teamid VARCHAR(20),
        Year INTEGER,
        Amount INTEGER,
        FOREIGN KEY (Playerid) REFERENCES (Player.Playerid),
        FOREIGN KEY (Teamid) REFERENCES (Team.Teamid),
        PRIMARY KEY (Playerid, Teamid, Year)
)
CREATE TABLE PlayedIn (
        Gameid VARCHAR(20),
        Homeid VARCHAR(20),
        Visitorid VARCHAR(20),
        FOREIGN KEY (Playerid) REFERENCES (Player.Playerid),
        FOREIGN KEY (Teamid) REFERENCES (Team.Teamid),
        FOREIGN KEY (Gameid) REFERENCES (Game.Gameid),
        PRIMARY KEY (Gameid)
)
CREATE TABLE Event (
        Playerid VARCHAR(20),
        Gameid VARCHAR(20),
        Result VARCHAR(20),
        Inning INTEGER,
        IsHome INTEGER, // must be 0 or 1
        PRIMARY KEY (Playerid, Gameid, Result, Inning),
        FOREIGN KEY (Playerid) REFERENCES (Player.Playerid),
        FOREIGN KEY (Gameid) REFERENCES (Game.Gameid),
)
```

Lastly, we may introduce a few more tables if time permits us to expand our application or we find the above relations too restrictive. For example, we may provide postseason

data, all star game data, awards data (HOF, Cy-Young, Silver SLugger, MVP, etc.), and managerial data.

4. Features that will definitely be implemented in the application

A home page consisting of a leaderboard of player statistics, a team page for each team (containing aggregate team stats and payroll information), a player page for each player (containing salary information, player statistics, and age/height/etc. information), Splits tools (see player statistics by month, date range, year, team), prediction model (using either linear regression or machine learning) for all standard statistics available on each player page.

5. Features that might be implemented in the application, given enough time

A live stream of ongoing game information (from espn or a similar source). User accounts that allow leaderboard customization and perhaps choosing certain "pinned players" on the homepage (so that you can easily see the statistics of the pinned players)

6. Technology and tools to be used

Node.js for the front-end. Neo4J database for the salary information/payroll data. SQLite (or the Oracle database provided by SEAS like we used in homework2) to store our relational databases. R to clean/normalize/process data and get it ready to import into our databases. Potentially Python/Pandas if we are going to use SQLite (like we did in hw1 and hw0). Also, I used the Retrosheet parser applications to help parse the retrosheet data and pipe the output into CSV files.

7. Member responsibility for project components

Responsibilities

JOHN: clean up event-file CSVs and set up the relational database with this information according to the schema given in the milestone (try to be done with this by Nov. 12). Also read payroll data into CSV files

JUAN/OMAR: begin with the front end design of the webpage. Using Node.js, create homepage, team pages, and player pages according to the rough template on our git-repo. (try to be done or close to done by Nov. 12)

OMAR: Look into Neo4J and how to incorporate it into the Node.js framework.

MAX: Look at how to integrate some sort of linear regression/machine learning/prediction algorithm into the web application (try to be done or close to done by Nov. 12)

Let's plan to meet up around Nov. 12 to discuss how we will integrate the various components together into a unified application.

3. One group member should then upload a PDF document via Canvas with the information above, along with the relational schema and noSQL description.

It is important to establish early on specific project component responsibilities – each group member should have aspects of the project that they "own" and are responsible for. "Own" does not necessarily mean they will be doing all of the coding/development, but rather that they are responsible for making sure the feature is complete.

You should also design a relational schema for your application, and develop a description of the noSQL component. Your schema should be based on the application rather than a straightforward copy of the datasets used. The relational schema should be represented as an ER diagram, as well as through (normalized) SQL DDL.

For the web technology, we prefer you to use Node.js – however, if your team feels this is too difficult, then you may use something simpler (e.g. PHP). It is more important for you to be able to complete the project rather than use Node.js!