CSC343 - Project Phase 2

Roger Lam and Wayne Zhu

Due November 19th, 2021 at 8pm EST

Design Decisions

We want to examine the popular trends from the new generation of NBA basketball. More specifically, we will investigate the following 3 open-ended questions:

- Have players gotten better at shooting 3s? How has shot selection changed over the years?
- Are offenses truly more efficient nowadays? How much have points scored per game changed? Are players scoring more points per shot attempt? Are teams playing with faster pace in terms of number of possessions per game?
- Are more fouls being called? Has it become easier for players to draw free throws?

Our overall design focused on building around a shot relation. It logically followed that the shot relation should have a corresponding player of which is associated with a team relation. Moreover, every shot is taken in some game, making it also necessary to have a game relation. The connected nature of these relations allowed our design to be straightforward and easy to follow:

- Player($\underline{\text{playerID, teamID}}$, year, GP, PTS, FGA, FGM, FG%, eFG%, 3PTA, 3PTM, 3PT%, FTA, FTM, FT%, AST, REB, STL, BLK, TOV, PF)
- Team($\underline{\text{teamID, teamName}}$)
- Shot($\underline{\text{shotID, playerID}}$, shotDistance, gameID, clutchTime, shotResult)
- Game($\underline{\text{gameID, teamID, oppTeamID}}$, homeScore, awayScore)

TA feedback was mostly positive, however there were a couple obvious flaws that they brought to attention.

Firstly, we could have more widely considered what data were required for our inquiries. For instance, player positions may play a strong role in investigating whether or not offenses are more efficient and if the organization has generally become 'soft'. Additionally, players' experience level may help answer why 3-pointers have supposedly become so dominant in the league. Even further, our questions could be more thoroughly answered by involving NBA teams' trade decisions and their long-term goals.

Secondly, the keys in every relation except for the Player relation are not minimal. Any given tuple can be uniquely identified by the team id, therefore team name should not be part of the key. Similarly for the Game relation, any two games must reasonably have different gameIDs from each other, hence home and away teamIDs are unnecessary. Finally, a basketball shot can only be taken by a single player; making playerID part of the key is redundant.

Outside of the feedback, we also found that our Player relation was very poorly designed. One reason for this is that we did not include the player names. Having names to players is quintessential to our analysis as it would provide more relevant information to work with. Having an incorrectly defined key also contributed to its bad design. More specifically, we did not include the year attribute as part of the key. This left out the possibility that a player could not be on more than one team in a year. Teams make dozens of trades throughout the year, it would be completely unreasonable to exclude this from the data. Lastly, the integrity constraint $Shot[playerID] \subseteq Player[playerID]$ we defined in phase 1 cannot hold, since $playerID$ itself is not a key.

Our renewed relations are as follows:

- Player(playerID, teamID, year, playerName, GP, PTS, FGA, FGM, FG%, eFG%, 3PTA, 3PTM, 3PT%, FTA, FTM, FT%, AST, REB, STL, BLK, TOV, PF)
- Team(teamID, teamName)
- Shot(shotID, playerID, shotDistance, gameID, clutchTime, shotResult)
- Game(gameID, teamID, oppTeamID, homeScore, awayScore)

Cleaning Process

For context, the official NBA website has a comprehensive stats portal which uses an undocumented internal API. The nba_api Python package is a community-authored module that provides a wrapper for accessing the official NBA API. Unfortunately, the nba_api package is also very badly documented. These factors heavily informed our data collection process, which typically went as follows:

- Go to stats.nba.com, navigate to a page that delivers data that we want to grab. Use our browser inspector to find the name of the endpoint being used by the internal NBA API
- With our target endpoint, look through the source code of nba_api to find the correct wrapper function that will allow us to mass call and format data
- In the case that no one endpoint could satisfy our data requirements, grab data from two or more endpoints and manually merge relevant data together in Python.

Luckily, we can trust that the data from the official NBA API is internally consistent, which means that certain fields from the data that we extract i.e. game_id, player_id, team_id are keys with consistent foreign relationships across our queries. Since our schema use these exact same keys, we never had any problems with key violations. Finally, since the data that we take is from a small subset of data returned by our API calls, we are certain that there are no invalid NULL references anywhere. We're lucky that our source data was compliant with all of the constraints that we had in place.