Team Lead:

Josh Slocum

Members:

Dillon Walker

Ernesto Barajas

Kevin Hinojosa

Kevin Esswein

NBA Statistics Technical Report

Introduction

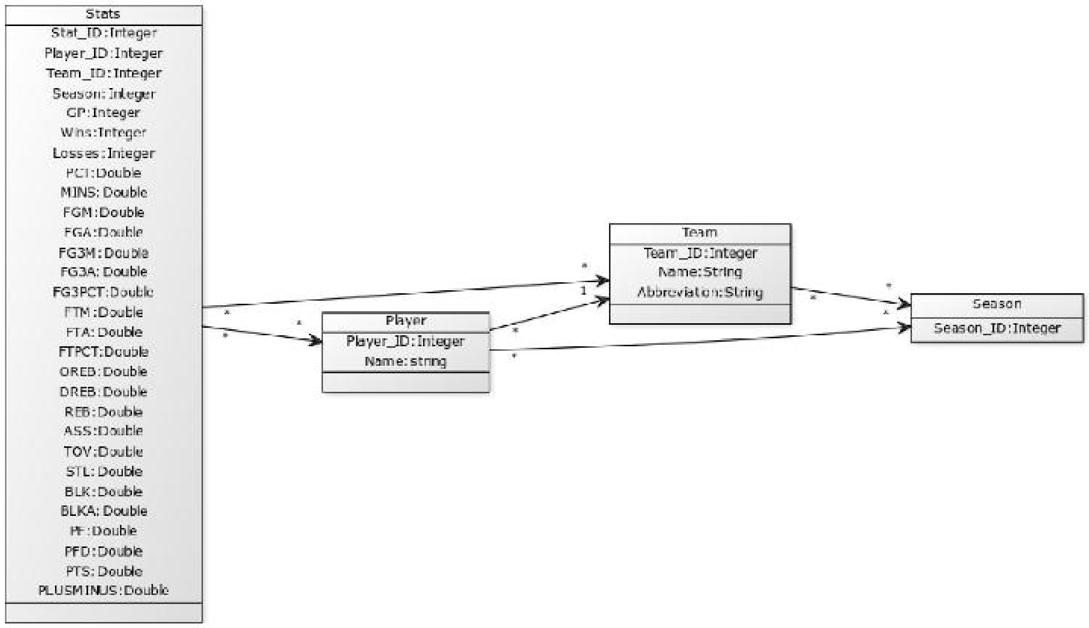
Problem

The National Basketball Association has hundreds of players and hundreds of basketball games each year. It has recorded interesting stats on them for almost the entirety of its existence.

Until recently, these statistics were unavailable to the general public. In recent years, the NBA created a complicated, undocumented REST API to access different types of statistics they have recorded in their vast databases. They also created a website to display this information, but it is mostly a leaderboard-type website for the casual fan. It is very difficult to do any analysis or useful computation on their website. Furthermore, requests to the API are obfuscated by numerous extraneous parameters that are not clearly relevant to the request.

Solution

We created our service, NBA Stats, to improve the use case of this data. Instead of an undocumented, complicated API with many different query parameters, we made a couple improvements. First, we simplified the data model to be per season, not per game. This drastically reduced both the information we needed to store, and the complexity of the information to process. Second, we simplified the API to three common categories: teams, players, and seasons. We made a page on our website for each of these “pillars”, which displays statistics about each of them for a given season. We believe that this will make the data much more accessible to the average developer as it will save them a lot of time that would be otherwise spent figuring out which query parameters mean what, and when they are optional.



Design

Flask Models

Since the NBA keeps track of the same stats for both players and teams, we chose to store the stats as their own entity, and build everything else as an abstraction around these statistics. Each “stat line” has a number of basketball statistics, as well as the player id, team id, and season. We also have a separate Player, Team, and Season models so that we can join each of these with the stats table to produce the stats for a given player/team for a given season. Since the stat lines contain both a player and team id, we had to distinguish between a team’s stat line and a player’s stat line. To do so, we chose to make the playerid null in the team’s stat line. This makes sense because there is no specific player to associate the stat line to. But, it also made sense to not remove the team id for a player’s stat line, since this would prevent us from getting the roster for a team and doing any aggregate calculations on a team’s players.

One thing to be aware of when viewing a player’s statistics in a season is that players can change teams throughout the season. Our model is able to handle this in a convenient way. A player that changes teams during the season will simply have two stat lines in our database: one line for the team they were originally on at the start of the season, that includes their season statistics up until the trade took place; and a second line for the team that they were traded to, including their season statistics through the remainder of the season.

Seasons

The Season model just contains the season_id, which is just the year of the start of the season. The main reason for having these seasons all in one table is to be able to quickly identify what seasons we have data for in the database.

Teams

The Team model contains the information on a given team, independent of the season. This consists of the team_id, which is a number generated by the nba for each team, the team name, and the three letter abbreviation for the team.

Players

The player model holds the information for each player, independent of the team they played on or the season they played in. This information is the player id, which is a number generated by the nba for each player, and their name.

Stats

The stats model is the key component of the NBA stats website and API. It contains a large number of statistics of the per-game average of the performance of an NBA player or team over the course of the season.

All of the stats are:

- Wins (wins)
- Losses (losses)
- Win percentage (pct)
- Minutes played (min)
- Field goals made (fgm)
- Field goals attempted (fga)

- Three-point field goals made (fg3m)
- Three-point field goals attempted (fg3a)
- Three-point field goal percentage (fg3pct)
- Free throws made (ftm)
- Free throws attempted (fta)
- Free throw percentage (ftpct)
- Offensive rebounds (oreb)
- Defensive rebounds (dreb)
- Total rebounds (reb)
- Assists (ass)
- Turnovers (tov)
- Steals (stl)
- Blocks (blk)
- Blocks attempted (blka)
- Fouls (pf)
- Defensive fouls (pfd)
- Points (pts)
- Plus-minus (plusminus)

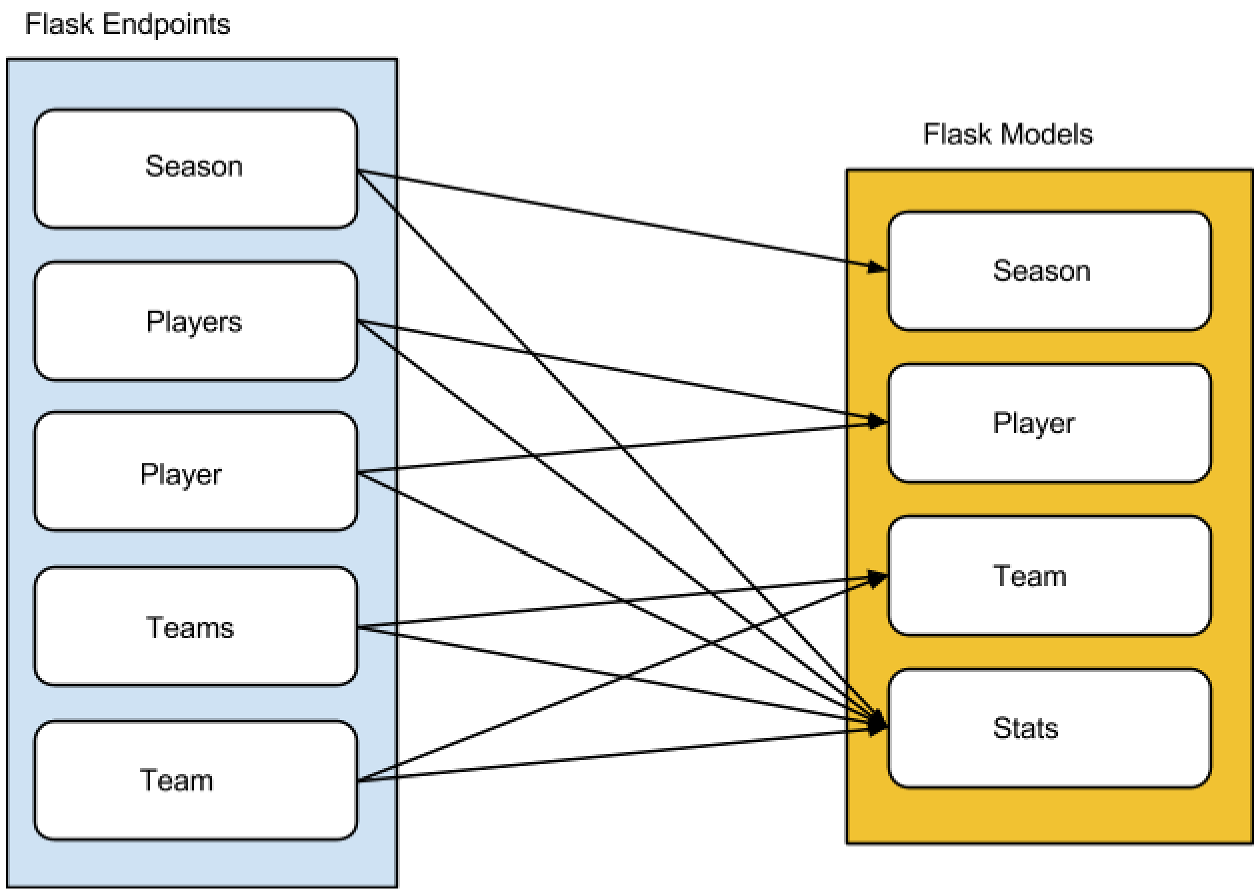
Most of the stats for a team are the sum of the stats for all of the players that played in that game. Thus, a couple of the fields, like minutes played, are mostly meaningless for a team, since the number of minutes played, barring overtime periods, is the same every game. Conversely, some of the stats for the player are just a function of the team's, such as wins and losses.

Stat errors

The NBA only began collecting these statistics in 1974. As a result, they (and by extension, we) only have statistics dating back to that year. Furthermore, when they initially began collecting these statistics, they collected only a subset of the numbers they are collecting today. For example, the NBA only has aggregated statistics on a team basis players and teams back until the 1996 season. Before then, there are individual player statistics, but as mentioned, they are a much smaller set of statistics than the modern set. As such, queries for older dates will generate data with many values that are not filled in, likely taking the form of "null", "None" or zero values. The user should be prepared to handle these values for such older request dates.

RESTful API

Due to our programming backgrounds and simplicity on the client side, we chose to do the bulk of the computation on the server. As such, our API to access the data is relatively straightforward. Since our three pillars of content are Seasons, Players, and Teams, we created a REST endpoint for each of these. Each endpoint takes in the desired season, player, or team, and generates a json object containing the statistics for that season, player, or team.



Seasons Endpoint

The seasons endpoint generates a json object listing all of the seasons with stat lines in our database. The endpoint has no required parameters as it returns a collection of all of the available seasons.

Season Endpoint

The season endpoint generates a page of team statistics for all teams in the input season. The endpoint takes a season year (where a season's year is the year that the season started in; that is, the 1999-2000 season would have a season year of 1999), and searches the database for statistics entries corresponding to the input year. As our statistics for players and teams are stored in the same table, we then remove player statistics lines for the input season, so that the result is several stat lines for each individual team that existed during the input season.

As previously mentioned, requests for season information for seasons before 1974 have undefined behavior, as the earliest season that we are able to retrieve statistics for is 1974.

Players Endpoint

The players endpoint generates a collection of json objects of all of the players with available stat lines in our database and their corresponding statistics. This endpoint does not have any required inputs as it returns every player with available statistics.

Player Endpoint

The player endpoint generates a json object of player statistics for all seasons that the player has statistics for in their NBA career. The endpoint takes a player ID and retrieves all stat lines for the corresponding player, and generates a series of tables to display the player’s statistics by year for each season. A single season stat line for a player is retrieved when the player ID and a season ID are given. The single season stat line for the player is returned as a json object.

Teams Endpoint

The teams endpoint generates a json object containing the collection of all of the teams in our database with available statistics and each team’s corresponding statistics. This endpoint does not have any required inputs similar to the players endpoint as it returns all of the available teams.

Team Endpoint

The team endpoint generates a json object the of team statistics for all players on the team, and the teams total statistics for each season the team has been in existence. The endpoint takes a team ID and returns the appropriate statistics as described above in a json object. A single season stat line for a team is retrieved by including the team ID and a season ID according to the API required syntax. The team's single season stat line is returned in a json object. Some teams have changed location throughout their existence, for example the Oklahoma City Thunder changed location and name from the Seattle SuperSonics in 2008. For teams such as this, the stats for the team's existence (to the extent that the NBA has stats recorded) will be available regardless of name changes. Simply be aware that their names will always be represented by the name that they are currently known.

Potential Discrepancies

As noted before, multiple stat lines can exist for a single player in the same season. Users may notice that a player that has played on more than one team during a single season has stats on each of those teams that do not match; furthermore, each of those individual lines do not match the individual player's seasonal performance. This is because the player's individual contribution to each team that they play for during the regular season is represented separately in our backend by different stat lines. Then when we construct the player's individual page, we aggregate their statistics across all teams that they played on during the regular season and present the aggregated statistics instead of any of their individual performances on any team they were on.

Flask Application

Our server currently only serves up static html pages for each player, team, and season we have created. The Flask application has routes for players, seasons, and teams. The server also has routes for the splash page, about page, and the javascript we use to create the sortable tables in our statistics pages. Future plans include dynamically creating the pages to replace our current model of serving up static html along with placing our flask application behind nginx.

Test

We use SQL Alchemy's automap feature, which reads from our database and automatically generates the models based on the tables and the relationships contained in the table descriptions. As such, there was not much in the way of models to test, so generally our tests try to verify that several test model instances inserted into the database can be retrieved, and in the resulting instances, we verify that several fields are accessible and contain the expected values.

Seasons

As the Season model is fairly simplistic (one field), we had one test verifying that we could insert a season into the database, and then recover that same season with the expected accessor using the original season ID.

Teams

Similarly to Season, the Team model has only three fields. As such, there we test Teams just like Seasons. We add a testing object to the database, verify that we can retrieve the testing object by ID, and verify that the testing object has the fields we would expect given our database.

Players

We test the player model in two different ways. Similarly to the Team and Season models, we verify that we can get a test Player object by ID from the database, and that the automatically generated models have the expected accessor methods with accurate expected test values. Then we also have a test to verify that we can select multiple players from the database by ID, and verify values in the same way as the other tests.

StatLines

The StatLine model varies from the other models in that there are restrictions on values of the player_id reference, and the team_id reference. As we mentioned before, a StatLine for a player will have a player_id and a team_id along with a season, to indicate that the StatLine represents the referenced player’s contribution to the referenced team in the referenced season. However, team StatLines do not have players associated with them, and so the player_id is null in those lines. Our current database does not enforce that restriction, so we did not expect that the automatically generated StatLine model would enforce. We do the same kind of test as we did for the other Models, to verify that we can get two different test StatLines (player-specific and team-specific StatLines) from the database and verify that they have the expected statistic fields. Then we also verify that we can fetch a group of StatLines.