Clippers Full-Stack Developer Assessment

Alex Nakagawa

Skip to:
Questionnaire
Developer Test

Questionnaire

Back to Title Slide

Questionnaire

1. How did you end up at your undergraduate school? How did you choose your major? What appealed to you about it? What Mathematics, Statistics, Computer Science, or related subject(s) did you take while in school?

I went to UC Berkeley for a multitude of very different reasons:

- ➔ Taking computer science was a great privilege in high school. It was the class in which I found the most joy given the creative freedom to build stuff.
- ➔ Berkeley was widely considered one of the best colleges to study CS, it was my first choice.
- ➔ I was chosen as a Regents and Chancellor's Scholar (top 2% of admitants), which lessened the financial burden considerably.
- ➔ One semester later, I switched to Data Science for the opportunity it provided me to take interdisciplinary classes from other departments (such as statistics).

Questionnaire

1. How did you end up at your undergraduate school? How did you choose your major? What appealed to you about it? What Mathematics, Statistics, Computer Science, or related subject(s) did you take while in school?

B.A. Data Science

Minor in Industrial Engineering and Operations Research

Data science became a new major in Fall 2018. I was drawn to the idea of being able to take more classes outside of just computer science, along with devoting myself towards reaching a more specific concentration of study. I landed on a “Cognition” domain emphasis, specializing in neurological and psychological aspects of data.

It was in the major where I found my passions in democratizing education for all students and public health research.

The major also provided me with the technical competency to begin my journey into the sports analytics world.

Questionnaire

1. How did you end up at your undergraduate school? How did you choose your major? What appealed to you about it? What Mathematics, Statistics, Computer Science, or related subject(s) did you take while in school?

Mathematics/Statistics

- ➔ Linear Algebra
- ➔ Stochastic Processes
- ➔ Linear Programming & Network Flows (AMPL)
- ➔ Engineering Stat., Quality Control, Forecasting (Python)
- ➔ Probability for Data Science (Python)
- ➔ Discrete Mathematics

Computer Science

- ➔ Intro CS Class (Python, Scheme)
- ➔ Data Structures and Algorithms (Java)
- ➔ Industrial Databases and Design (Access, Python, SQL)
- ➔ Theory of Databases (Java, SQL)
- ➔ Computer Arch. (C, Spark)

Related Subjects

- ➔ Inferential Thinking in Data Science (Python)
- ➔ Data Science in Venture Applications (Python)
- ➔ Computational Models of Cognition

** (___) → programming language/software used

Questionnaire

2. Proficiencies with: Languages, Frameworks, Computational Tools

Questionnaire

2. Proficiencies with: Languages, Frameworks, Computational Tools

Python

Level: ● ● ● ● ○

4 years in school for all data science related analyses (ML, stats) and for back-ends; especially APIs. Used in internship at NBA

SQL/NoSQL

Level: ● ● ● ○ ○

Familiar with both relational and key-value store paradigms. Used in Postgres, MongoDB, FaunaDB, etc. Used in internship at NBA

Javascript

Level: ● ● ● ○ ○

Self-taught vanilla Javascript, alongside with some other frameworks for front-end related work such as JSX w/ React Components, Svelte compiler, etc

Kotlin

Level: ● ● ○ ○ ○

Self-taught Kotlin on JetBrains Academy for developing websites after falling in love with it as a replacement to the verbosity of Java. Made tools in the command line.

Questionnaire

2. Proficiencies with: Languages, Frameworks, Computational Tools

My favorite stack: Svelte + GraphQL + Hasura

GraphQL / Level: ● ● ● ○ ○

Querying language for backend development, used when creating the backend of multi-platform apps.

Svelte / Level: ● ● ○ ○ ○

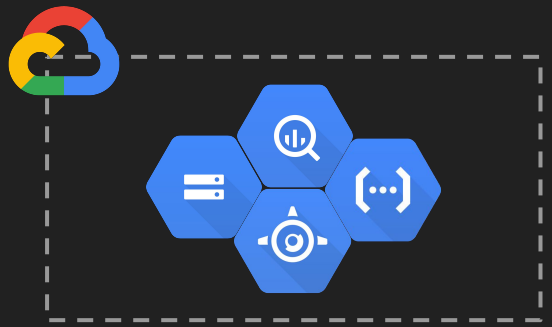
Front-End Compiler that removes the verbosity of React and makes stateful components easy to build. Still relatively new to it, used when quickly creating frontend sites

Hasura / Level: ● ● ○ ○ ○

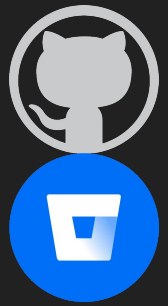
Engine that converts PostgresQL to GraphQL

Questionnaire

2. Proficiencies with: Languages, Frameworks, Computational Tools
The following tools are indispensable to my workflow as a software/data engineer, but I would still consider myself proficient/beginner level with most of them.



Google Cloud: Storage, BigQuery, Functions, App Engine, Airflow (Composer)



Version Control: Github, Bitbucket

Relational: Airtable, PostgresQL

Non-Relational: FaunaDB

Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

#1: Spotify

I love Spotify (user since 2011), but I honestly feel that they can do so much more when it comes to bringing the SOCIAL aspect of music sharing to the consumer.

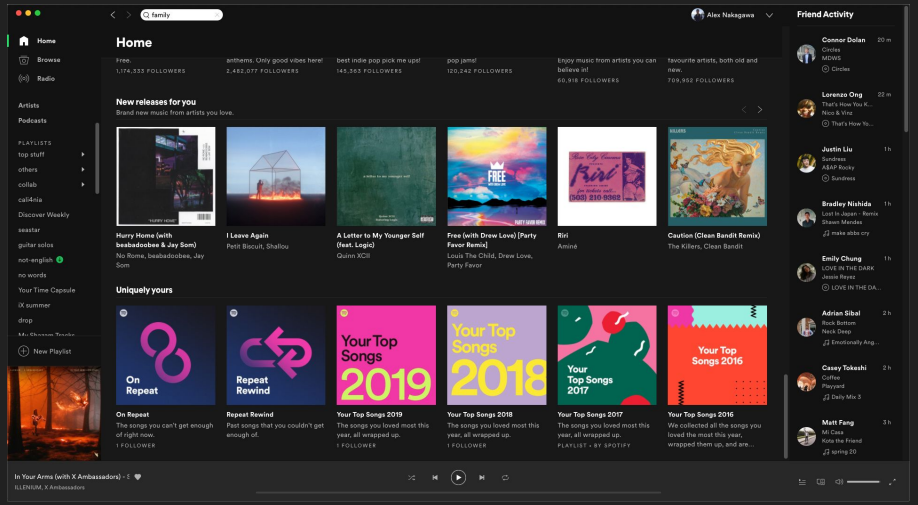
When discovering new music, it almost always comes from the recommendation of my friends.

Creating an entire playlist to share to your friend doesn't give the "live" feel that social media provides and misses out on the chance to discuss or give feedback.

Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

#1: Spotify My fix: re-inventing the side-social bar:



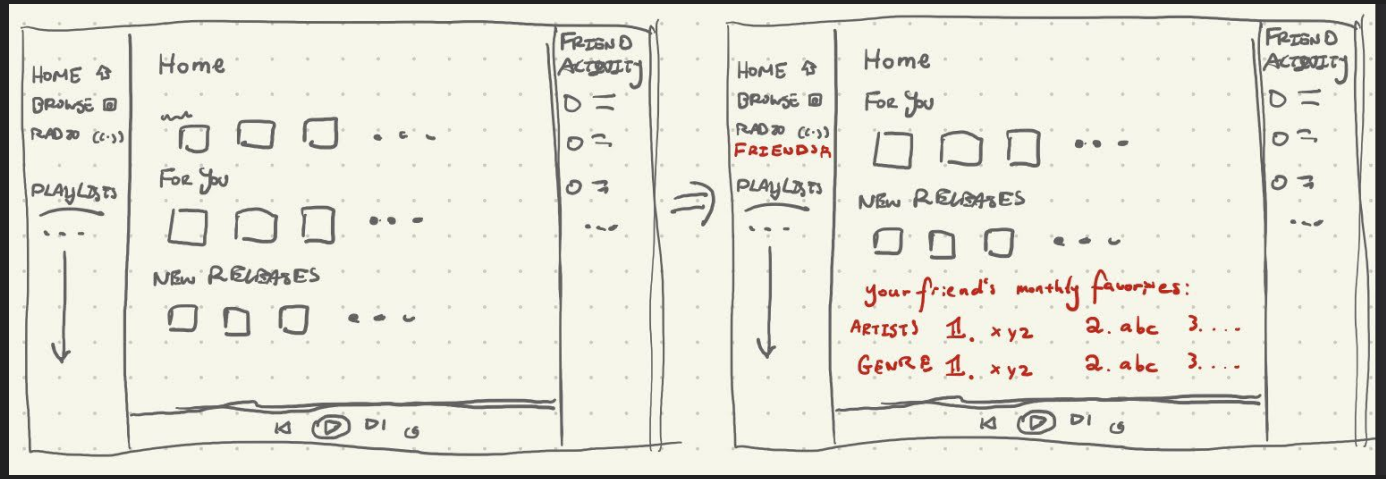
1. Why can't I view who follows my playlists?
2. Noticeable 3 minute delay between switching songs

Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

#1: Spotify

My fix: re-inventing the side-social bar:



Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

#2: Yogurt Park

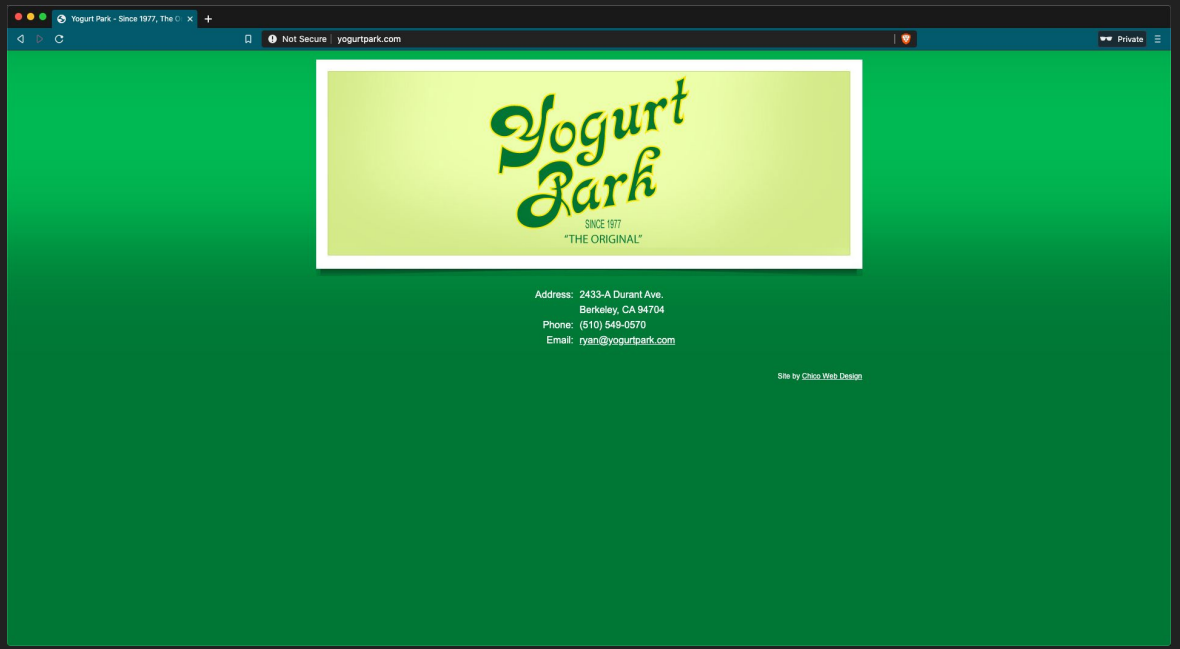
I had a favorite small business in college that sold the best frozen yogurt on the planet. However, their website is atrocious. It provides zero functionality other than to list an address and email.

This particular store changes their flavors every single day at random. The only way to keep up with what the flavors of the day are, you have to visit their Twitter, which rarely gets updated before they open.

Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

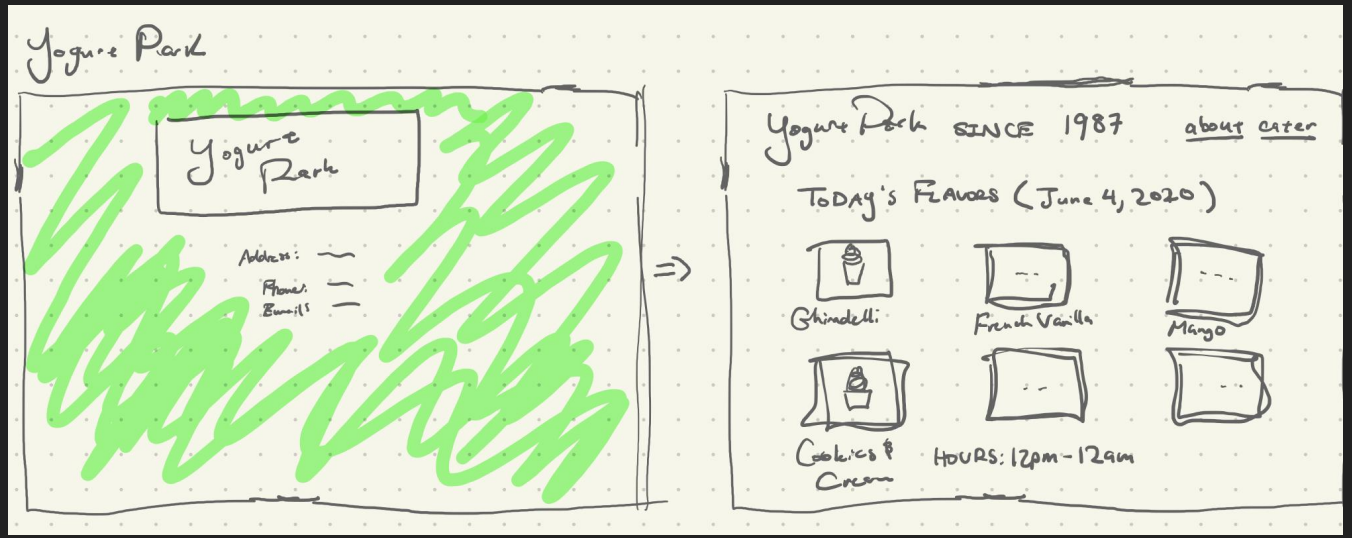
#2: Yogurt Park



Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

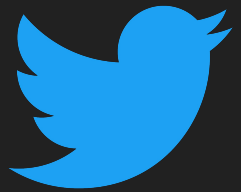
#2: Yogurt Park



Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

#3: Twitter



Okay, there are only two things that irk me constantly about Twitter.

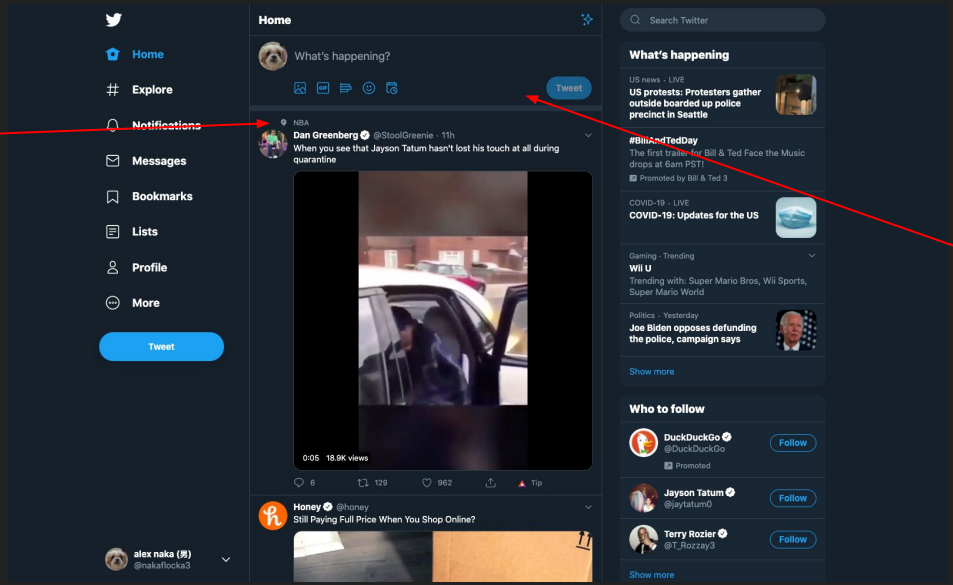
1. No access to Tweet drafts on twitter.com
2. Clicking topics on home page will not redirect you to the topic, but just tell you you're following the topic (duh).

Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

#3: Twitter A simple feature add for the Desktop experience

Clicking this has no effect.

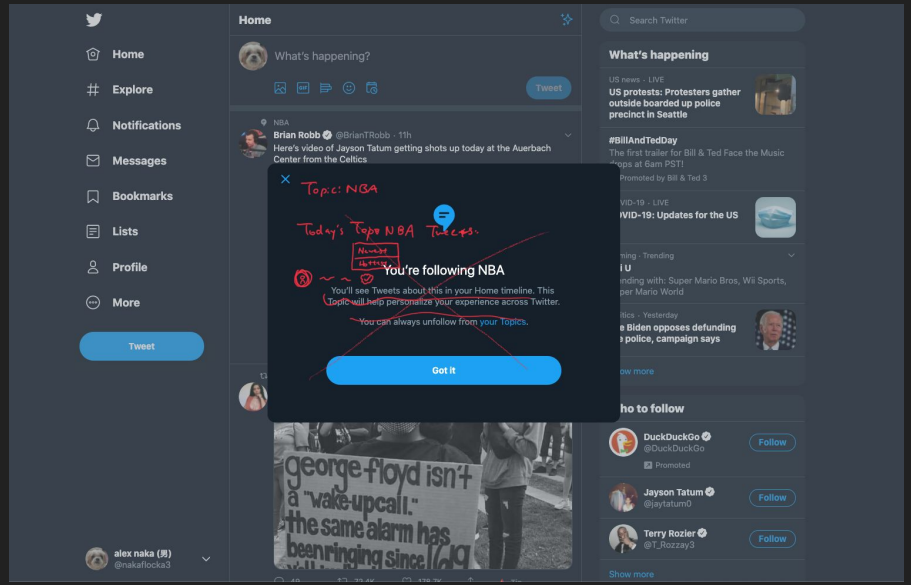


No drafts...

Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

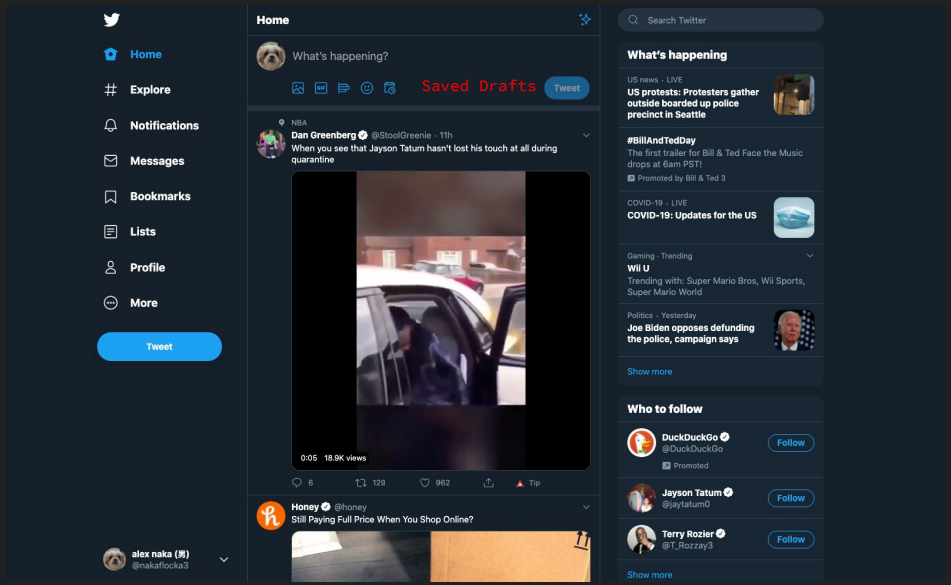
#3: Twitter A simple feature add for the Desktop experience



Questionnaire

3. List 3 websites or apps that annoy you (functionally/aesthetically). In your opinion what's wrong with them? How would you fix them?

#3: Twitter A simple feature add for the Desktop experience



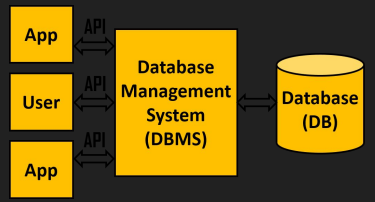
Questionnaire

4. What is the hardest thing you ever programmed? Why was it difficult? How did you overcome the difficulty?

In the spring semester of freshman year, I was taking the (in)famous Data Structures course at my university. As a way of spurring further innovation in class teaching, our new professor decided he would give us a very, very daunting task:

Build a database management service (DBMS) from scratch

Intimidated but determined, and with very little starter code, I began my journey towards a fully implemented DBMS. The reason for the level of difficulty was that we were given almost no direction on the specifics on how to implement it, with the only constraints being accuracy and time limits on query execution.



Questionnaire

4. What is the hardest thing you ever programmed? Why was it difficult? How did you overcome the difficulty?

The very first step was to scope the problem: what were the bare minimum products required to run a DBMS?

- A command parser
- A query executer
- A schema generator

I now had my business logic set. These three items would be the makeup of the final deliverable.

The second step was to break each product down into the required data models and structures to hold our information:

I.e. schema generator would require an Array to maintain the ordering of columns as specified, but only need a HashMap to keep the columns in-tact, as ordering is maintained through primary keys

Questionnaire

4. What is the hardest thing you ever programmed? Why was it difficult? How did you overcome the difficulty?

Doing the previous two steps was *essential* to map the possible solution in my head before writing a single line of Java code.

This careful consideration and calculated approach towards building products has helped me countless times towards reaching solutions to problems faster. I'm not perfect (I often get sidetracked with trying shiny, new methods to get to answers), but I am getting better.

Questionnaire

5. What online (sports/programming/statistics) communities do you read and/or participate in on a regular basis?

I'm a dedicated subscriber to The Athletic, particularly my favorite journalists:

- Seth Partnow
  - Quite adept at writing somewhat objectively, and shows in his analytical and stats-heavy writing-style
- Zach Harper
  - Has the thankless job at doing the weekly power rankings for the NBA
- Mina Kimes (ESPN)
  - Her stories always feel very personal

I also engage occasionally in Twitter discourse about sports stats, and had the privilege of attending the Sloan Sports Analytics Conf. I got into sports statistics thanks to my internship at the NBA.

@TheKendallBaker, @chuurveg, @nickwan, @CausalKathy, many others.

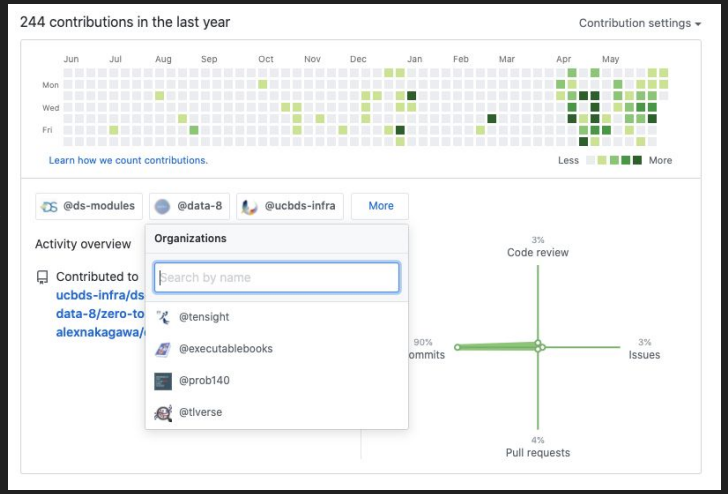
Questionnaire

5. What online (sports/programming/statistics) communities do you read and/or participate in on a regular basis?

I've dedicated a lot of my senior year in college contributing to open source projects at the Division of Data Science at my school as part of my part-time gig there, along with helping the Jupyter Notebook team with some of their open source beta testing and documentation.

A snapshot of some orgs I've contributed to are listed. Most of them are affiliated with UC Berkeley and creating curriculum for classes.

Besides that, I'm a big fan of the Kotlin language from JetBrains and follow their progress. Most recently, I've began to follow development tips on dev.to



Questionnaire

5. What online (sports/programming/statistics) communities do you read and/or participate in on a regular basis?

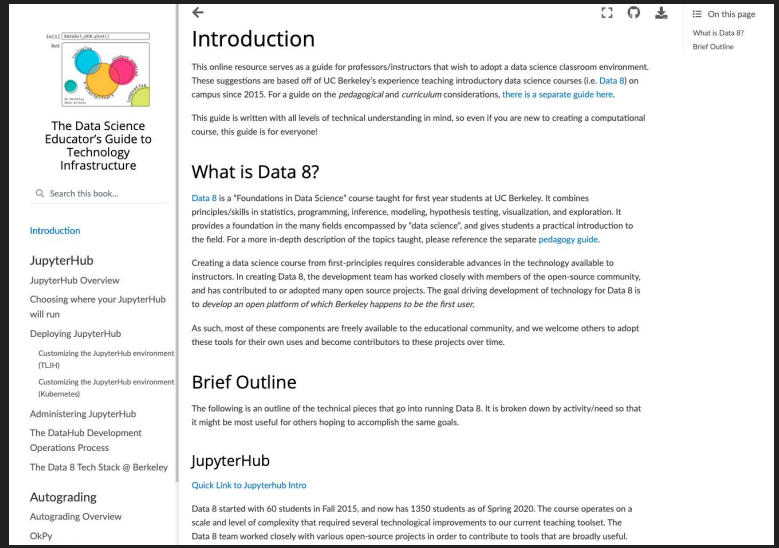
Like previously mentioned, I've been a large founding open source contributor to the data science programs of UC Berkeley as an undergrad.

[DS-MODULES] [TECH-GUIDE]

In addition to this, the research paper I was working on was set to be submitted to conferences such as UC Tech, PyData, SciPy, and more before COVID-19 hit. An example proposal is linked below.

[PROPOSAL]

Finally, I'm a follower (not so much a contributor) of open-source analytics in sports, such as ncaahoopR by Luke Benz.



Questionnaire

6. Have you ever taught yourself something? What was it? Why did you learn it? How did you learn it?

As a freshman, I took my very first internship *setting up a classification model* that could take bank transaction names and classify them into certain categories. While I had a theoretical understanding of how to do so using vanilla Python, it was a whole other beast to be able to deploy the model itself.

In 2017, AWS released the first beta version of *AWS Sagemaker*, which was an environment that provided serverless runtimes to deploy models to production. I didn't know it at the time, but after hours of researching the problem, I was able to come to the realization that I would be able to leverage the features of the new product to complete the task.

It took me *3 months* of trial and error (with some very bad coding conventions and styling), but when it finally deployed, it gave me my very first taste of building features. Working with a product that had barely been used meant that I had no online resources to help me through solving the issues and errors I ran into. Fortunately, I contacted the support team at AWS, who was very helpful in pointing me to the right people to speak to, and, with the help of AWS engineers, I was one of the first people to beta test the product to completion.

Questionnaire

7. During your time in school, what single skill did you develop that you think would be the most valuable for this role and what single skill did you develop that you think would be the least valuable for this role? Why?

TIME MANAGEMENT is by far the most single important transferable skill I am developing. I learned how I needed to invest my time towards myself, not stretch myself too thin (w/ Crew, student orgs, classes, part-time work, etc.), and overall get things done in as timely of a manner as possible.

The least valuable skill I learned for this role within college was a knack for eating while working. It was a nasty habit I developed in school, and there's no way I want to continue doing that at the Clippers.

Questionnaire

8. What interests you about working for a basketball team? How do you think a Basketball Front Office could maximize your development?

To be fully transparent, and as you may have seen at this point, I am *not* trained to be a “fullstack” developer by way of my data science-heavy course-load. However, I’ve taken strides towards learning the ins-and-outs of fullstack throughout my senior year, and was able to complete projects such as a nationally-used technology guide for educators. This sudden push was kickstarted by my data engineering and backend work at the NBA as an intern, and I feel so fortunate to have been offered the chance to show what I’ve self-taught myself since then here in this challenge.

For me, I see a huge amount of potential for the sports industry to become a disrupter in creating technological and digital experiences for consumers. As a fan, I’m starting to see it as more teams try to out-maneuver each other not just physically, but by also running the numbers in the front-office. My initial reasoning for wanting to enter the world of sports was to aid in the research for *solutions to injury prevention*.

When speaking to Gregg, I really felt how impactful the position was when he described how the work directly influences the abilities of the directors of the team. That type of level of responsibility is a valuable opportunity, and I’m up for the challenge.

Questionnaire

8. What interests you about working for a basketball team? How do you think a Basketball Front Office could maximize your development?

*BONUS

On a more personal note, working for the Clippers would mean a tremendous amount to me. In middle school, I was part of the band as a trumpet player, and we would be invited to perform before the game began on the court.

The Clippers organization also employs many Japanese Americans - an identity I value very strongly. I've looked up to Natalie Nakase for so long throughout high school. She's been an inspiration of perseverance for me.

Developer Test

Jump to:

Lineups

Two-Way Full Stack

Back to Title Slide

Lineups

The file structure of the repository is as follows:

- ➔ lineups
  - dev_test_data
  - images
  - lineups.ipynb ← **This is the file you want
  - requirements.txt

Lineups

➔ Technologies Used:

Lineups

➔ Query 2a: Win-Loss Records and Win Percentage

We need a simple way to view the home games and away games for each steam

Lineups

➔ Query 2a: Win-Loss Records and Win Percentage

We need a simple way to view the home games and away games for each steam

```
CREATE VIEW home_games AS
    SELECT home_id AS team_id, COUNT(*) AS total_home_games,
         SUM(CASE WHEN home_score > away_score THEN 1 ELSE 0 END) AS home_wins,
         SUM(CASE WHEN home_score < away_score THEN 1 ELSE 0 END) AS home_losses
    FROM game_schedule
    GROUP BY home_id
```

CASE WHEN...THEN...ELSE...END
is an Ansi SQL syntax that allows for predicate cases.

Lineups

➔ Query 2a: Win-Loss Records and Win Percentage

We need a simple way to view the home games and away games for each steam

```
CREATE VIEW home_games AS
    ...
CREATE VIEW away_games AS
    ...
```

away_games follows very similar syntax, just counting for the away games this time.

Lineups

➔ Query 2a: Win-Loss Records and Win Percentage

We need a simple way to view the home games and away games for each steam

```
CREATE VIEW home_games AS
    ...
CREATE VIEW away_games AS
    ...
CREATE VIEW total_records AS
    SELECT home_games.team_id,
           home_wins + away_wins AS wins,
           home_losses + away_losses AS losses
    FROM home_games JOIN away_games ON home_games.team_id = away_games.team_id
```

total_records now combines the two views we created together. The final call is now much simpler.

Lineups

➔ Query 2a: Win-Loss Records and Win Percentage

We need a simple way to view the home games and away games for each steam

```
CREATE VIEW home_games AS
    ...
CREATE VIEW away_games AS
    ...
CREATE VIEW total_records AS
    ...
SELECT *,
       CAST(CAST(wins AS DECIMAL)/(wins + losses) AS DECIMAL(6,5)) AS
                                                          win_percentage
FROM team JOIN total_records ON team.team_id = total_records.team_id \
ORDER BY win_percentage DESC, team.team_id ASC
```

➔ Ordered by descending win percentage
➔ Ties are ordered next by ascending team_id

Lineups

➔ Query 2a: Win-Loss Records and Win Percentage

FINAL RESULT

- ➔ Bucks have 100% win-percent
- ➔ Lowest 3 are the Cavs, Suns, Wizards with ~14.3%
- ➔ Clippers went 4-3, ~57.1%

Out[208]:

	team_id	name	city	abrv	team_id	wins	losses	win_percentage
0	6	Cavaliers	Cleveland	CLE	6	1	6	0.14286
1	24	Suns	Phoenix	PHX	24	1	6	0.14286
2	30	Wizards	Washington	WAS	30	1	6	0.14286
3	11	Rockets	Houston	HOU	11	1	5	0.16667
4	5	Bulls	Chicago	CHI	5	2	6	0.25000
5	7	Mavericks	Dallas	DAL	7	2	6	0.25000
6	20	Knicks	New York	NYK	20	2	6	0.25000
7	1	Hawks	Atlanta	ATL	1	2	5	0.28571
8	22	Magic	Orlando	ORL	22	2	5	0.28571
9	21	Thunder	Oklahoma City	OKC	21	2	4	0.33333
10	3	Nets	Brooklyn	BKN	3	3	5	0.37500
11	14	Lakers	Los Angeles	LAL	14	3	5	0.37500
12	16	Heat	Miami	MIA	16	3	4	0.42857
13	4	Hornets	Charlotte	CHA	4	4	4	0.50000
14	18	Timberwolves	Minnesota	MIN	18	4	4	0.50000
15	23	76ers	Philadelphia	PHI	23	4	4	0.50000
16	9	Pistons	Detroit	DET	9	4	3	0.57143
17	13	Clippers	Los Angeles	LAC	13	4	3	0.57143
18	19	Pelicans	New Orleans	NOP	19	4	3	0.57143
19	29	Jazz	Utah	UTA	29	4	3	0.57143
20	12	Pacers	Indiana	IND	12	5	3	0.62500
21	26	Kings	Sacramento	SAC	26	5	3	0.62500
22	15	Grizzlies	Memphis	MEM	15	4	2	0.66667
23	2	Celtics	Boston	BOS	2	5	2	0.71429
24	25	Trail Blazers	Portland	POR	25	5	2	0.71429
25	27	Spurs	San Antonio	SAS	27	5	2	0.71429
26	8	Nuggets	Denver	DEN	8	6	1	0.85714
27	28	Raptors	Toronto	TOR	28	7	1	0.87500
28	10	Warriors	Golden State	GSW	10	8	1	0.88889
29	17	Bucks	Milwaukee	MIL	17	7	0	1.00000

Lineups

➔ Query 2b: Rank of Games Played

```
SELECT RANK () OVER (
       ORDER BY h.total_home_games + a.total_away_games DESC,
                h.total_home_games DESC,
                a.total_away_games DESC) rank,
       h.team_id,
       t.name,
       h.total_home_games + a.total_away_games as total_games,
       h.total_home_games,
       a.total_away_games
FROM home_games h JOIN away_games a
        ON h.team_id = a.team_id
    JOIN team t ON h.team_id = t.team_id
ORDER BY total_games DESC, h.total_home_games DESC,
    a.total_away_games DESC
```

RANK() OVER (...)
returns a rank (w/ ties) of the returned rows with a specified order

Lineups

➔ Query 2b: Rank of Games Played

```
In [19]:  1 read_query(query_2_b)
```

executed in 228ms, finished 20:17:52 2020-05-30

Out[19]:

	rank	team_id	name	total_games	total_home_games	total_away_games
0	1	10	Warriors	9	4	5
1	2	28	Raptors	8	6	2
2	3	18	Timberwolves	8	5	3
3	3	20	Knicks	8	5	3
4	5	14	Lakers	8	4	4
5	5	5	Bulls	8	4	4
6	5	23	76ers	8	4	4
7	8	26	Kings	8	3	5
8	8	3	Nets	8	3	5
9	8	4	Hornets	8	3	5
10	8	7	Mavericks	8	3	5
11	8	12	Pacers	8	3	5
12	13	17	Bucks	7	5	2
13	14	9	Pistons	7	4	3
14	14	27	Spurs	7	4	3
15	14	16	Heat	7	4	3
16	14	8	Nuggets	7	4	3
17	14	19	Pelicans	7	4	3
18	14	6	Cavaliers	7	4	3
19	14	13	Clippers	7	4	3
20	14	22	Magic	7	4	3
21	22	24	Suns	7	3	4
22	22	2	Celtics	7	3	4
23	22	25	Trail Blazers	7	3	4
24	25	30	Wizards	7	2	5
25	25	29	Jazz	7	2	5
26	25	1	Hawks	7	2	5
27	28	11	Rockets	6	4	2
28	28	21	Thunder	6	4	2
29	30	15	Grizzlies	6	3	3

Final Result

- ➔ Warriors played most games with 9
- ➔ Lowest Ranked is Grizzlies with 6 games, and only 3 home games
- ➔ Clippers played 7 games with 4 home.

Lineups

➔ Query 3a: Home and Away Back-to-Backs

```
CREATE OR REPLACE FUNCTION
    rest_days (day2 timestamp without time zone, day1 timestamp without time
zone)
RETURNS integer AS $rest_days$
BEGIN
    RETURN date_part('day'::text, day2) - date_part('day'::text, day1) - 1;
END;
$rest_days$ LANGUAGE plpgsql;
```

rest_days is a user-defined function that calculates the rest days between two days. **date_part** is a postgres function that takes the day of a timestamp

**** In hindsight, I realize that this would break if it went from Oct. 31 -> Nov. 1, so this solution would technically break on new data.):**

Lineups

➔ Query 3a: Home and Away Back-to-Backs

```
CREATE OR REPLACE FUNCTION
    rest_days (...);

CREATE OR REPLACE VIEW home_rest AS
SELECT g1.home_id, g1.game_id first_game_id, g1.game_date first_game_date,
       CAST(g2.game_id AS INTEGER) second_game_id, MIN(g2.game_date)
second_game_date,
       rest_days(min(g2.game_date), g1.game_date)
FROM game_schedule g1
    LEFT JOIN game_schedule g2 ON g1.home_id = g2.home_id
                                  AND g2.game_date > g1.game_date
WHERE g2.game_id IS NOT NULL
GROUP BY g1.home_id, first_game_id, second_game_id
```

home_rest finds consecutive games and calculates the amount of rest between.

Lineups

➔ Query 3a: Home and Away Back-to-Backs

```
CREATE OR REPLACE FUNCTION
    rest_days (...);

CREATE OR REPLACE VIEW home_rest
    ...

CREATE OR REPLACE VIEW away_rest
    ...
```

away_rest follows almost same exact logic

Lineups

➔ Query 3a: Home and Away Back-to-Backs

```
CREATE OR REPLACE FUNCTION
    rest_days (...);

CREATE OR REPLACE VIEW home_rest
    ...;

CREATE OR REPLACE VIEW away_rest
    ...;

SELECT home_id, team.name, COUNT(*) num_home_b2b
FROM home_rest JOIN team ON home_id = team.team_id
WHERE rest_days = 0
GROUP BY home_id, team.name;
SELECT away_id, team.name, COUNT(*) num_away_b2b
FROM away_rest JOIN team ON away_id = team.team_id
WHERE rest_days = 0
GROUP BY away_id, team.name;
```

Two separate queries for home and away

Lineups

➔ Query 3a: Home and Away Back-to-Backs

```
In [21]:
query_3_a_home = text("""
SELECT home_id, team.name, COUNT(*) num_home_b2b
FROM home_rest JOIN team ON home_id = team.team_id
WHERE rest_days = 0
GROUP BY home_id, team.name"""
)
read_query(query_3_a_home)
```

executed in 166ms, finished 20:17:52 2020-05-30

Out[21]:

	home_id	name	num_home_b2b
0	8	Nuggets	1
1	19	Pelicans	1

```
In [22]:
query_3_a_away = text("""
SELECT away_id, team.name, COUNT(*) num_away_b2b
FROM away_rest JOIN team ON away_id = team.team_id
WHERE rest_days = 0
GROUP BY away_id, team.name"""
)
read_query(query_3_a_away)
```

executed in 88ms, finished 20:17:52 2020-05-30

Out[22]:

	away_id	name	num_away_b2b
0	1	Hawks	1
1	2	Celtics	1
2	4	Hornets	1
3	5	Bulls	1
4	9	Pistons	1
5	10	Warriors	1
6	11	Rockets	1
7	16	Heat	1
8	23	76ers	1
9	24	Suns	1
10	25	Trail Blazers	1
11	26	Kings	1
12	29	Jazz	1

Final Result

➔ Nuggets and Pelicans had one home b2b each

➔ 13 teams had one away b2b each

Lineups

➔ Query 3b: Most Rest Days

```
CREATE OR REPLACE VIEW games_by_team AS (
SELECT t.team_id, g2.game_id, g2.game_date
FROM team t
    RIGHT JOIN game_schedule g2 ON t.team_id = g2.away_id
UNION ALL
SELECT t.team_id, g1.game_id, g1.game_date
FROM team t
    RIGHT JOIN game_schedule g1 ON t.team_id = g1.home_id
ORDER BY team_id ASC, game_id ASC);
```

games_by_team gets team ids and uses UNION ALL to match each game with all the others. This will be useful to find consecutive total games.

Lineups

➔ Query 3b: Most Rest Days

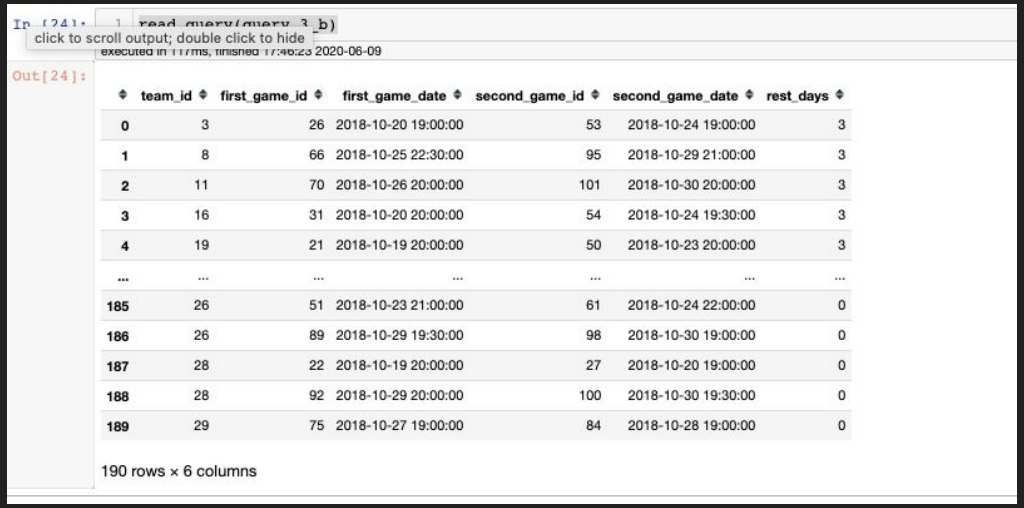
```
CREATE OR REPLACE VIEW games_by_team
    ...

SELECT g1.team_id,
       g1.game_id AS first_game_id,
       g1.game_date AS first_game_date,
       min(g2.game_id) AS second_game_id,
       min(g2.game_date) AS second_game_date,
       rest_days(min(g2.game_date), g1.game_date) AS rest_days
FROM games_by_team g1
    LEFT JOIN games_by_team g2 ON g1.team_id = g2.team_id AND g2.game_date >
g1.game_date
WHERE g2.game_id IS NOT NULL
GROUP BY g1.team_id, g1.game_id, g1.game_date
ORDER BY rest_days DESC, g1.team_id ASC, g1.game_id ASC
HAVING rest_days = MAX(rest_days) ← optional to just return max
```

This query will find the minimum next game from each row (aka next game) and also use **rest_days** to calculate rest days again.

Lineups

➔ Query 3b: Most Rest Days



	team_id	first_game_id	first_game_date	second_game_id	second_game_date	rest_days
0	3	26	2018-10-20 19:00:00	53	2018-10-24 19:00:00	3
1	8	66	2018-10-25 22:30:00	95	2018-10-29 21:00:00	3
2	11	70	2018-10-26 20:00:00	101	2018-10-30 20:00:00	3
3	16	31	2018-10-20 20:00:00	54	2018-10-24 19:30:00	3
4	19	21	2018-10-19 20:00:00	50	2018-10-23 20:00:00	3
...	...	...	...	...	...	...
185	26	51	2018-10-23 21:00:00	61	2018-10-24 22:00:00	0
186	26	89	2018-10-29 19:30:00	98	2018-10-30 19:00:00	0
187	28	22	2018-10-19 20:00:00	27	2018-10-20 19:00:00	0
188	28	92	2018-10-29 20:00:00	100	2018-10-30 19:30:00	0
189	29	75	2018-10-27 19:00:00	84	2018-10-28 19:00:00	0

190 rows × 6 columns

Final Result

There are several instances of 3-rest days. This seems to be the max.

Lineups

➔ Query 4a: Wide Table of Lineups

```
CREATE OR REPLACE VIEW five_player_lineup AS (
SELECT game_id, team_id, lineup_num, period, array_agg(player_id) as
five_players
FROM lineup
GROUP BY game_id, team_id, lineup_num, period
);
```

five_player_lineup finds who is on the court by aggregating player ids into an array **(aray_agg)** by the lineup_num and period for each game.

Lineups

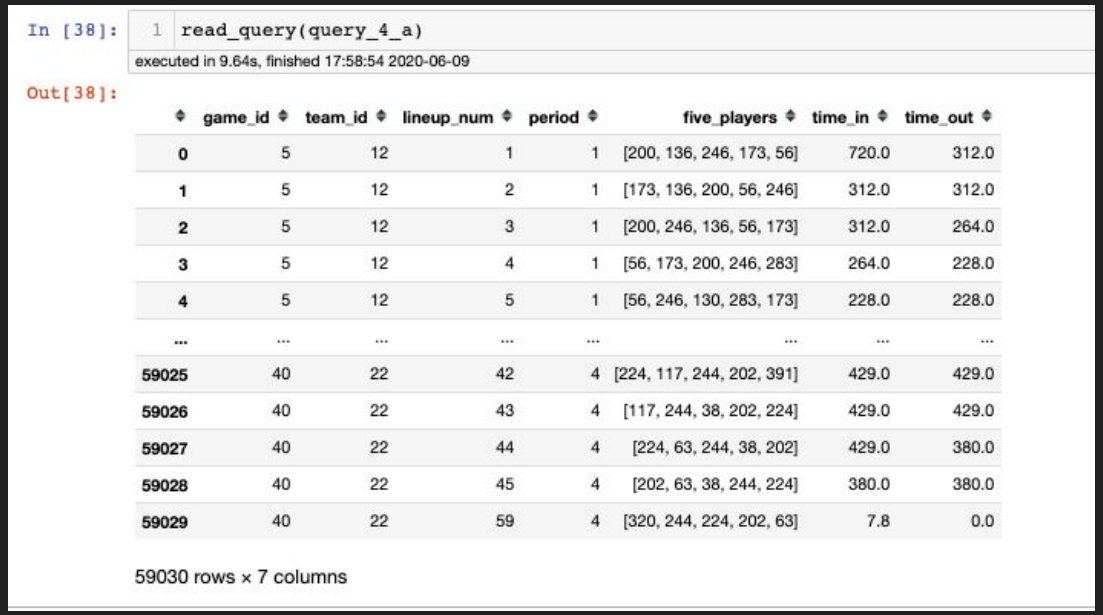
➔ Query 4a: Wide Table of Lineups

```
CREATE OR REPLACE VIEW five_player_lineup
    ...;


SELECT f.*, l.time_in, l.time_out
FROM five_player_lineup f
    JOIN lineup l ON f.game_id = l.game_id
                  AND f.team_id = l.team_id
                  AND f.lineup_num = l.lineup_num
                  AND f.period = l.period;
```

Lineups

➔ Query 4a: Wide Table of Lineups



```
In [38]: 1 read_query(query_4_a)
```

executed in 9.64s, finished 17:58:54 2020-06-09

Out[38]:

	game_id	team_id	lineup_num	period	five_players	time_in	time_out
0	5	12	1	1	[200, 136, 246, 173, 56]	720.0	312.0
1	5	12	2	1	[173, 136, 200, 56, 246]	312.0	312.0
2	5	12	3	1	[200, 246, 136, 56, 173]	312.0	264.0
3	5	12	4	1	[56, 173, 200, 246, 283]	264.0	228.0
4	5	12	5	1	[56, 246, 130, 283, 173]	228.0	228.0
...	...	...	...	...	...	...	...
59025	40	22	42	4	[224, 117, 244, 202, 391]	429.0	429.0
59026	40	22	43	4	[117, 244, 38, 202, 224]	429.0	429.0
59027	40	22	44	4	[224, 63, 244, 38, 202]	429.0	380.0
59028	40	22	45	4	[202, 63, 38, 244, 224]	380.0	380.0
59029	40	22	59	4	[320, 244, 224, 202, 63]	7.8	0.0

59030 rows × 7 columns

Final Result

➔ If we don't need an array, we can always unpack the array column by using the **unnest()** postgres function

Lineups

➔ Query 4b: Player Stints

```
CREATE OR REPLACE VIEW all_next_times AS (
    SELECT l1.team_id, l1.player_id, l1.game_id, l1.period, l1.lineup_num, l1.time_in,
l1.time_out, (CASE WHEN l1.time_out != 0 THEN l2.time_in ELSE 0 END) next_in_or_end
    FROM lineup l1 JOIN lineup l2 ON  l1.team_id = l2.team_id
                                  AND l1.player_id = l2.player_id
                                  AND l1.game_id = l2.game_id
                                  AND l1.period = l2.period
                                  AND l1.lineup_num != l2.lineup_num
                                  AND l1.time_in != l1.time_out
    WHERE l1.time_in >= l2.time_in OR l1.time_out = 0
    GROUP BY l1.team_id, l1.player_id, l1.game_id, l1.period, l1.lineup_num, l1.time_in,
l1.time_out, next_in_or_end
    ORDER BY team_id, player_id, game_id, period, l1.time_in DESC, time_out DESC,
next_in_or_end DESC
);
```

all_next_times uses a join on lineup twice to get all next times a player is in the game following the first row.

Lineups

➔ Query 4b: Player Stints

```
CREATE OR REPLACE VIEW all_next_times
     ...;

CREATE OR REPLACE VIEW next_time_in_or_end AS (
    SELECT team_id, player_id, game_id, period, lineup_num, time_in, time_out,
MAX(next_in_or_end) max_next_in_or_end
    FROM all_next_times
    WHERE time_in != next_in_or_end
    GROUP BY team_id, player_id, game_id, period, lineup_num, time_in, time_out
    ORDER BY player_id, game_id, period, lineup_num, time_in DESC, time_out DESC
);
```

next_time_in_or_end uses the next_in_or_end column from all_next_times to determine the end of the current lineup

Lineups

➔ Query 4b: Player Stints

```
CREATE OR REPLACE VIEW all_next_times
      ...;

CREATE OR REPLACE VIEW next_time_in_or_end
      ...;
```

```
CREATE OR REPLACE VIEW created_group_stints AS (
SELECT team_id, player_id, game_id, period, time_in, time_out, max_next_in_or_end,
       CASE WHEN time_out = max_next_in_or_end THEN 'cont' ELSE 'end' END cont_or_end,
       COALESCE (SUM(CASE WHEN time_out != max_next_in_or_end THEN 1 ELSE 0 END)
           OVER (PARTITION BY team_id, player_id, game_id, period
                 ORDER BY team_id,
                          player_id,
                          game_id,
                          period,
                          time_in DESC,
                          time_out DESC,
                          max_next_in_or_end DESC
                rows between unbounded preceding and 1 preceding),
                0) AS stint_group_num
FROM next_time_in_or_end
GROUP BY team_id, player_id, game_id, period, time_in, time_out, max_next_in_or_end
ORDER BY team_id, player_id, game_id, period, time_in DESC, time_out DESC, max_next_in_or_end DESC
)
```

created_group_stints is the most important view that creates the stints themselves.

Lineups

➔ Query 4b: Player Stints

```
CREATE OR REPLACE VIEW all_next_times
      ...;

CREATE OR REPLACE VIEW next_time_in_or_end
      ...;

CREATE OR REPLACE VIEW created_group_stints
      ...;

CREATE OR REPLACE VIEW stint_times AS
(
SELECT team_id, player_id,
       game_id, period, stint_group_num,
       ROW_NUMBER () OVER (PARTITION BY team_id, player_id, game_id ORDER BY team_id,
                                                                             player_id,
                                                                             game_id,
                                                                             period,
                                                                             stint_group_num)
           AS stint_number,
        MAX(time_in) stint_start_time,
        MIN(time_out) stint_end_time
FROM created_group_stints
GROUP BY team_id, player_id, game_id, period, stint_group_num
)
```

stint_times now has the actual stints by finding the maxes and mins of groups.

Lineups

➔ Query 4b: Player Stints

```
CREATE OR REPLACE VIEW all_next_times
     ...;

CREATE OR REPLACE VIEW next_time_in_or_end
     ...;

CREATE OR REPLACE VIEW created_group_stints
     ...;

CREATE OR REPLACE VIEW stint_times
     ...;

SELECT g.game_date, s.team_id as team,
       CASE WHEN g.home_id != s.team_id THEN g.home_id ELSE g.away_id END as opponent,
       p.first_name || ' ' || p.last_name as player_name,
       s.period, s.stint_number,
       TO_CHAR((ROUND(s.stint_start_time) || ' seconds')::interval , 'MI:SS') AS stint_start_time,
       TO_CHAR((ROUND(s.stint_end_time) || ' seconds')::interval, 'MI:SS') AS stint_end_time
FROM stint_times s
    LEFT JOIN player p ON s.player_id = p.player_id
    LEFT JOIN game_schedule g ON s.game_id = g.game_id;
```

TO_CHAR() takes the casted interval inside and casts it to a char for nice printing.

Lineups

➔ Query 4b: Player Stints

In [29]:

```
query_4_b = text("""
SELECT g.game_date, s.team_id as team,
       CASE WHEN g.home_id != s.team_id THEN g.home_id ELSE g.away_id END as opponent,
       p.first_name || ' ' || p.last_name as player_name,
       s.period, s.stint_number,
       TO_CHAR((ROUND(s.stint_start_time) || ' seconds')::interval , 'MI:SS') AS stint_start_time,
       TO_CHAR((ROUND(s.stint_end_time) || ' seconds')::interval, 'MI:SS') AS stint_end_time
FROM stint_times s
    LEFT JOIN player p ON s.player_id = p.player_id
    LEFT JOIN game_schedule g ON s.game_id = g.game_id;
""")
read_query(query_4_b)
```

executed in 2.40s, finished 17:46:26 2020-06-09

Out[29]:

	game_date	team	opponent	player_name	period	stint_number	stint_start_time	stint_end_time
0	2018-10-17 19:30:00	1	20	Kent Bazemore	1	1	12:00	06:47
1	2018-10-17 19:30:00	1	20	Kent Bazemore	1	2	04:45	02:50
2	2018-10-17 19:30:00	1	20	Kent Bazemore	2	3	08:40	03:09
3	2018-10-17 19:30:00	1	20	Kent Bazemore	3	4	12:00	00:00
4	2018-10-17 19:30:00	1	20	Kent Bazemore	4	5	12:00	04:53
...	...	...	...	...	...	...	...	...
8757	2018-10-30 20:00:00	30	15	Kelly Oubre	3	4	05:18	00:00
8758	2018-10-30 20:00:00	30	15	Kelly Oubre	4	5	12:00	00:00
8759	2018-10-24 22:30:00	30	10	Thomas Bryant	4	1	09:20	00:00
8760	2018-10-28 21:30:00	30	13	Thomas Bryant	4	1	10:39	00:00
8761	2018-10-24 22:30:00	30	10	Troy Brown Jr.	4	1	06:44	00:00

8762 rows × 8 columns

Final Result

➔ Has all of the rows for every stint in the game dataset.

Lineups

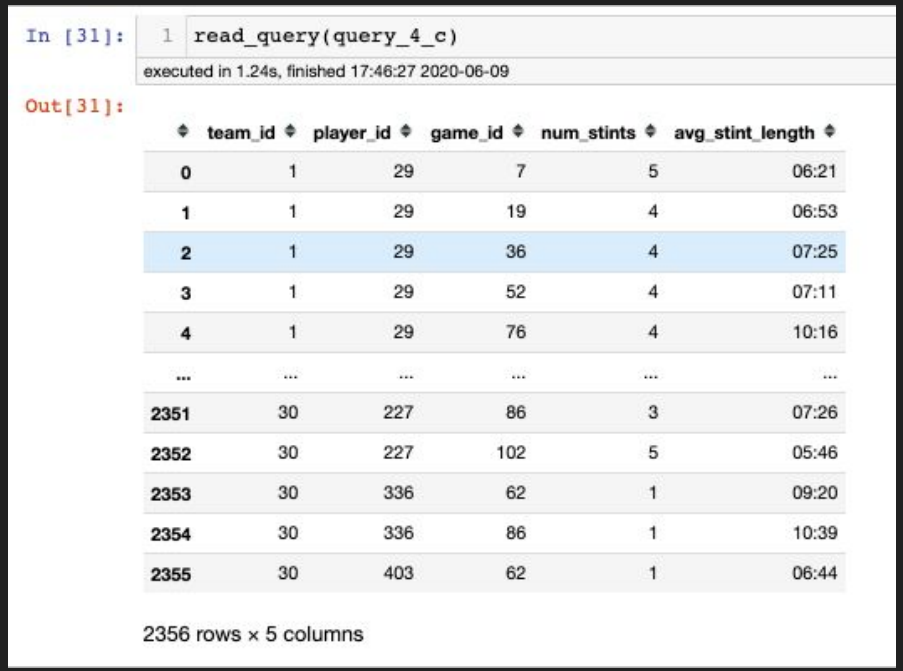
➔ Query 4c: Player Stints (Averaged)

```
SELECT team_id, player_id, game_id,
       COUNT(stint_number) AS num_stints,
       TO_CHAR((ROUND(AVG(stint_start_time - stint_end_time)) || ' seconds')::interval ,
'MI:SS' ) AS avg_stint_length
FROM stint_times
GROUP BY team_id, player_id, game_id;
```

Uses TO_CHAR again to prettify the minutes and seconds.

Lineups

➔ Query 4c: Player Stints (Averaged)



```
In [31]: 1 read_query(query_4_c)
```

executed in 1.24s, finished 17:46:27 2020-06-09

Out[31]:

	team_id	player_id	game_id	num_stints	avg_stint_length
0	1	29	7	5	06:21
1	1	29	19	4	06:53
2	1	29	36	4	07:25
3	1	29	52	4	07:11
4	1	29	76	4	10:16
...	...	...	...	...	...
2351	30	227	86	3	07:26
2352	30	227	102	5	05:46
2353	30	336	62	1	09:20
2354	30	336	86	1	10:39
2355	30	403	62	1	06:44

2356 rows × 5 columns

Final Result

➔ Has all of the rows for every stint averaged for each game per player

Lineups

➔ Query 4d: Differences between wins and losses

```
WITH query_4_c AS (
SELECT team_id, player_id, game_id,
       COUNT(stint_number) AS num_stints,
       TO_CHAR((ROUND(AVG(stint_start_time - stint_end_time)) || ' seconds')::interval , 'MI:SS' ) AS avg_stint_length
FROM stint_times
GROUP BY team_id, player_id, game_id
), game_by_winner AS (
SELECT q.*,
       CASE WHEN q.team_id = g.home_id THEN 'home' ELSE 'away' END AS home_or_away,
       g.home_score, g.away_score
FROM query_4_c q
    LEFT JOIN game_schedule g ON q.game_id = g.game_id
), stints_win_and_losses AS (
SELECT *,
       CASE WHEN home_or_away = 'home' THEN
            CASE WHEN home_score > away_score THEN 'win' ELSE 'loss' END
       ELSE
            CASE WHEN home_score < away_score THEN 'win' ELSE 'loss' END
       END AS win_or_loss
FROM game_by_winner
), running_wins_and_losses AS (
SELECT *, SUM(CASE WHEN win_or_loss = 'win' THEN 1 ELSE 0 END)
            OVER (PARTITION BY team_id, player_id ORDER BY game_id) AS running_wins,
          SUM(CASE WHEN win_or_loss = 'loss' THEN 1 ELSE 0 END)
            OVER (PARTITION BY team_id, player_id ORDER BY game_id) AS running_losses
FROM stints_win_and_losses
)
SELECT team_id, player_id, game_id, num_stints, avg_stint_length,
       home_or_away, home_score, away_score, win_or_loss,
       running_wins || ' - ' || running_losses AS running_record
FROM running_wins_and_losses;
```

Very long, but essentially making 4 sub queries to get the running wins and losses from the table.

Lineups

➔ Query 4d: Differences between wins and losses

```
In [46]: 1 read_query(query_4_d)
executed in 1.21s, finished 20:20:58 2020-06-09
```

Out[46]:

	team_id	player_id	game_id	num_stints	avg_stint_length	home_or_away	home_score	away_score	win_or_loss	running_record
0	1	29	7	5	06:21	away	126	107	loss	0 - 1
1	1	29	19	4	06:53	away	131	117	loss	0 - 2
2	1	29	36	4	07:25	away	111	133	win	1 - 2
3	1	29	52	4	07:11	home	111	104	win	2 - 2
4	1	29	76	4	10:16	home	85	97	loss	2 - 3
...	...	...	...	...	...	...	...	...	...	...
2351	30	227	86	3	07:26	away	136	104	loss	1 - 5
2352	30	227	102	5	05:46	away	107	95	loss	1 - 6
2353	30	336	62	1	09:20	away	144	122	loss	0 - 1
2354	30	336	86	1	10:39	away	136	104	loss	0 - 2
2355	30	403	62	1	06:44	away	144	122	loss	0 - 1

2356 rows × 10 columns

Final Result

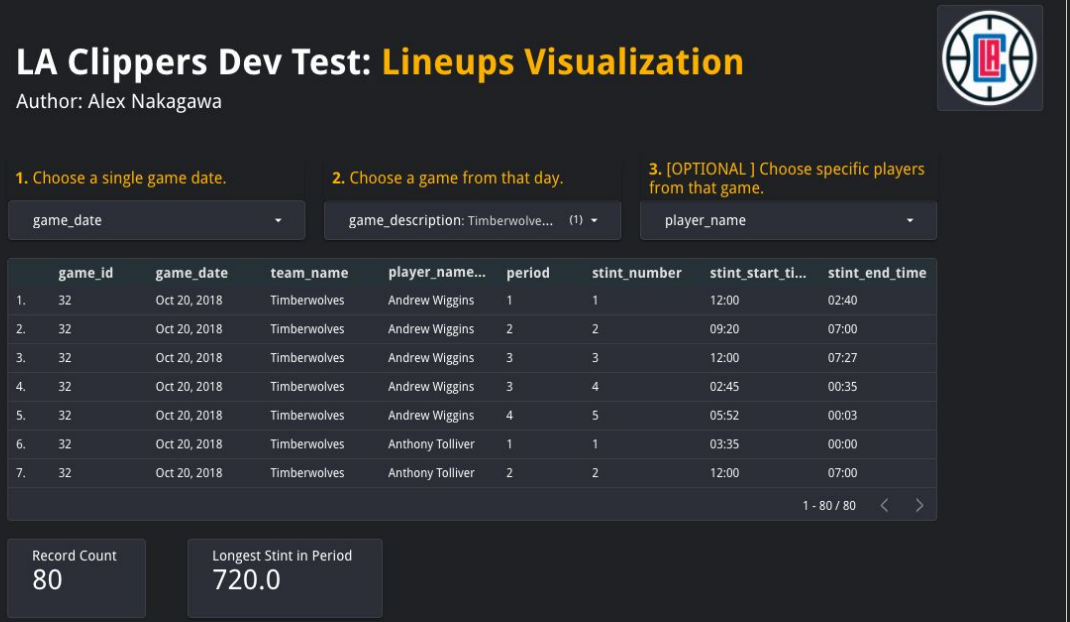
➔ Running record for each player!

Lineups

➔ Data Visualization

https://datastudio.google.com/open/1l6Gf4Qei58CDM-9BY2KXl7Jshmr0jLQ1?usp=sharing

Cloud SQL Data Studio



LA Clippers Dev Test: Lineups Visualization

Author: Alex Nakagawa

1. Choose a single game date.

game_date

2. Choose a game from that day.

game_description: Timberwolve... (1)

3. [OPTIONAL] Choose specific players from that game.

player_name

	game_id	game_date	team_name	player_name...	period	stint_number	stint_start_ti...	stint_end_time
1.	32	Oct 20, 2018	Timberwolves	Andrew Wiggins	1	1	12:00	02:40
2.	32	Oct 20, 2018	Timberwolves	Andrew Wiggins	2	2	09:20	07:00
3.	32	Oct 20, 2018	Timberwolves	Andrew Wiggins	3	3	12:00	07:27
4.	32	Oct 20, 2018	Timberwolves	Andrew Wiggins	3	4	02:45	00:35
5.	32	Oct 20, 2018	Timberwolves	Andrew Wiggins	4	5	05:52	00:03
6.	32	Oct 20, 2018	Timberwolves	Anthony Tolliver	1	1	03:35	00:00
7.	32	Oct 20, 2018	Timberwolves	Anthony Tolliver	2	2	12:00	07:00

1 - 80 / 80

Record Count 80

Longest Stint in Period 720.0

Screenshot

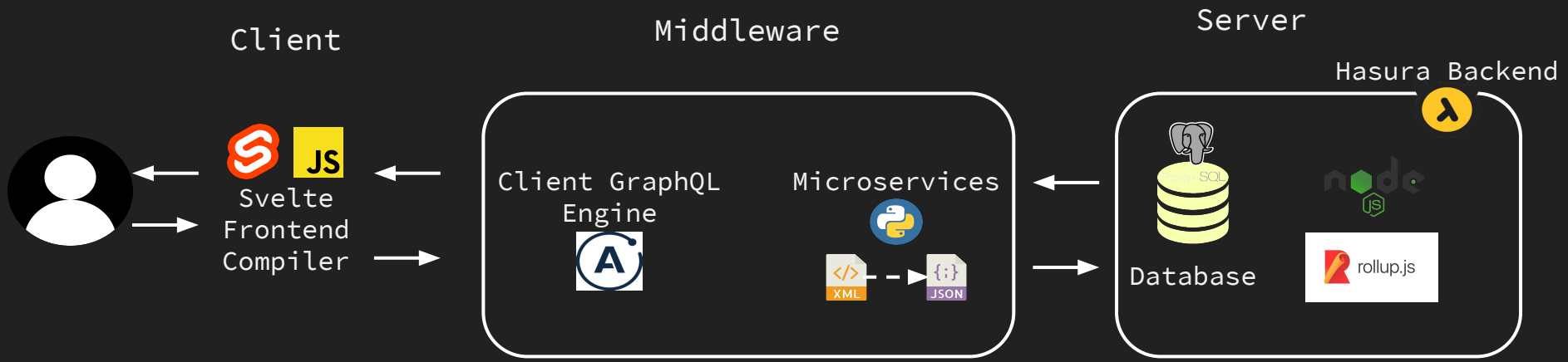
Two-Way Full Stack

Expectations

- ➔ This was one of the very first times I created my own front-end, which proved to be time-consuming, but extremely rewarding.
- ➔ Things I learned how to do:
  - ◆ write extensive GraphQL queries from the client.
  - ◆ learn async and await functionality of node.js
  - ◆ internalize component-like frontend development,
- ➔ The project's current status and shortcomings:
  - ◆ I was unable to implement two of the required pieces:
    - 3d: because this was all learned from scratch, I ran out of time to complete the calendar-like functionality, and cumulative information
    - Login/logout feature is not implemented
  - ◆ The hope is to *continue to improve this even for the following weeks to come*! (:

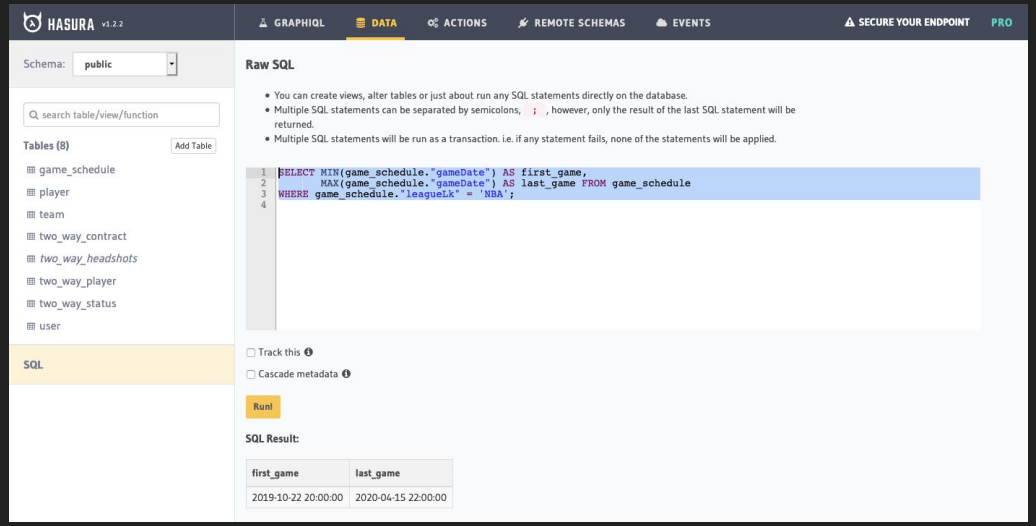
Two-Way Full Stack

Systems Design



Two-Way Full Stack

1. Basic Questions
    a. Based on the season given in game_schedule, what is the first day of the NBA season? What is the last day? How many days are in the season? How about for the GLG?



I ran queries inside of Hasura's SQL Editor, so I didn't have to do queries in Jupyter again.

Final Result

First Game: 2019-10-22
Last Game: 2020-04-15

Two-Way Full Stack

1. Basic Questions
    a. Based on the season given in game_schedule, what is the first day of the NBA season? What is the last day? How many days are in the season? How about for the GLG?

```
WITH first_last AS (
SELECT game_schedule."leagueLk",
       MIN(game_schedule."gameDate") AS first_game,
       MAX(game_schedule."gameDate") AS last_game FROM game_schedule
GROUP BY game_schedule."leagueLk"
)
SELECT *, EXTRACT(days from (last_game - first_game))::int as days_in_season
FROM first_last;
```

Track this

Cascade metadata

Run!

SQL Result:

leagueLk	first_game	last_game	days_in_season
GLG	2019-11-08 19:00:00	2020-03-28 22:00:00	141
NBA	2019-10-22 20:00:00	2020-04-15 22:00:00	176

```
Final Result

NBA
First Game: 2019-10-22
Last Game:  2020-04-15
Days: 176

GLG
First Game: 2019-11-08
Last Game:  2020-03-28
Days: 141
```

Two-Way Full Stack

1. Basic Questions
   a. …
   b. For the season included in game_schedule, how much does the Two-Way make for a day in the G-League? How about for a day in the NBA?

```sql
WITH salaries AS (
SELECT p.*, c."nbaEarnedSalary", c."glgEarnedSalary"
FROM two_way_contract c LEFT JOIN two_way_player p ON c."playerId" = p."playerId"
)
SELECT "playerId", player,
       "nbaEarnedSalary" / NULLIF("activeForNbaGameDays" + "travelWithNbaDays" + "withNbaDays", 0) AS nba_per_day,
       "glgEarnedSalary" / NULLIF("nonNbaGlgDays", 0) AS glg_per_day
FROM salaries
ORDER BY nba_per_day DESC;
```

Track this

Cascade metadata

Run!

SQL Result:

playerId	player	nba_per_day	glg_per_day
1626260	Levi Randolph	14562	1746
1629719	Devontae Cacok	5115	565
1626187	Michael Frazier	5075	560
1627737	Marquese Chriss	5075	560
1627740	Henry Ellenson	5075	560
1627789	Timothe Luwawu-Cabarrot	5075	560
1627814	Damion Lee	5075	560
1627982	Josh Gray	5075	560
1628394	Anzejs Pasecniks	5075	560
1628397	Ivan Rabb	5075	560
1628405	Johnathan Motley	5075	560
1628408	PJ Dozier	5075	560
1628412	Frank Mason	5075	560
1628424	Kobi Simmons	5075	560

Final Result

Levi Randolph makes the most amount of money in the NBA per day based on the data so far in the database. It could be possible he was added right as the season ended, so there's few days to count

Two-Way Full Stack

1. Basic Questions
   a. ...
   b. ...
   c. What is the maximum number of countable NBA days a Two-Way can have? Based on the season given in game_schedule, what is the maximum number of days a Two-Way can be on an NBA active/inactive list? Explain your answer.

Final Result

The maximum number of countable NBA days for a two-way is 45 days. In the dataset, the maximum number we see reached is 59 days, because of inactivity

```
SELECT "playerId", player, "totalDays" - "nonNbaGlgDays" AS nba_days
FROM two_way_player
ORDER BY nba_days DESC;
```

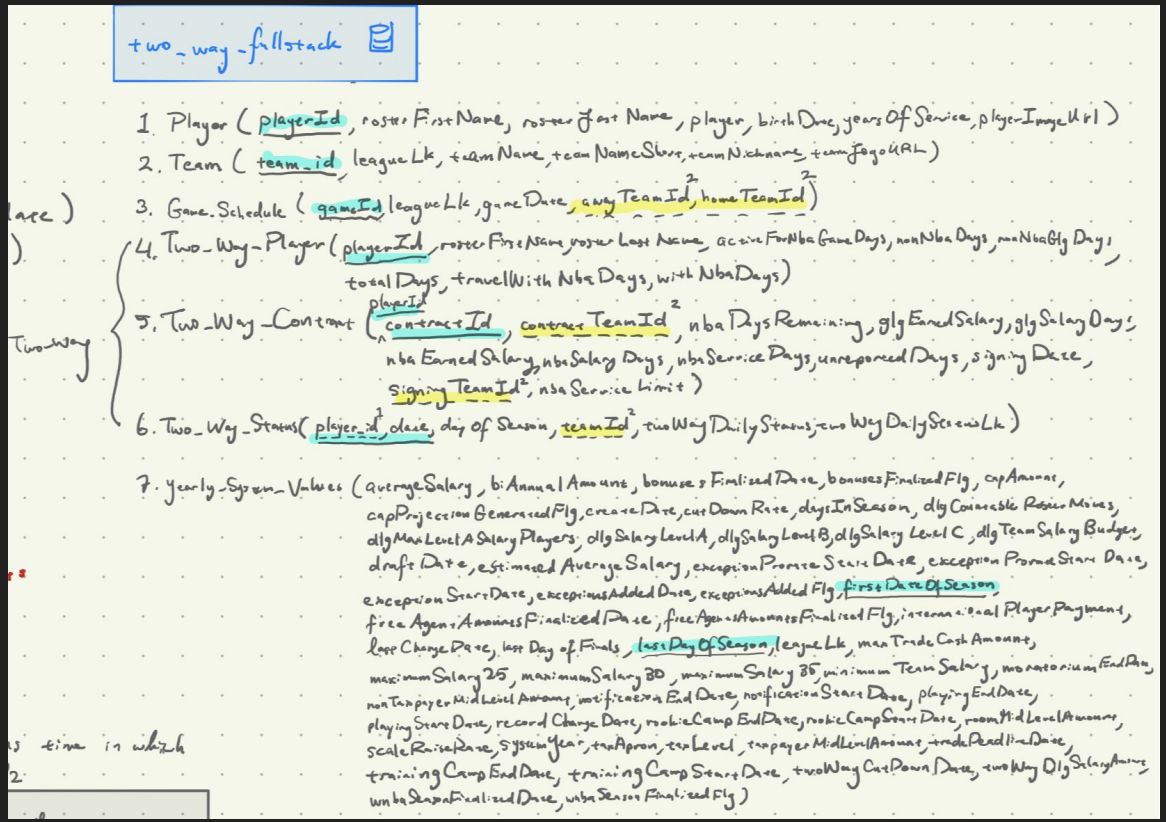
Track this
Cascade metadata
Run!

SQL Result:

playerId	player	nba_days
1629052	Oshae Brissett	59
1628405	Johnathan Motley	55
1629735	Chris Silva	55
1627814	Damion Lee	54
1629164	Brandon Goodwin	54
1629718	Charles Brown Jr.	52
1629598	Chris Clemons	52
1629658	Jaylen Hoard	51
1629065	Ky Bowman	51
203658	Norvel Pelle	51
1628985	Devon Hall	51
1629622	Max Strus	51
1629690	Adam Mokoka	50

Two-Way Full Stack

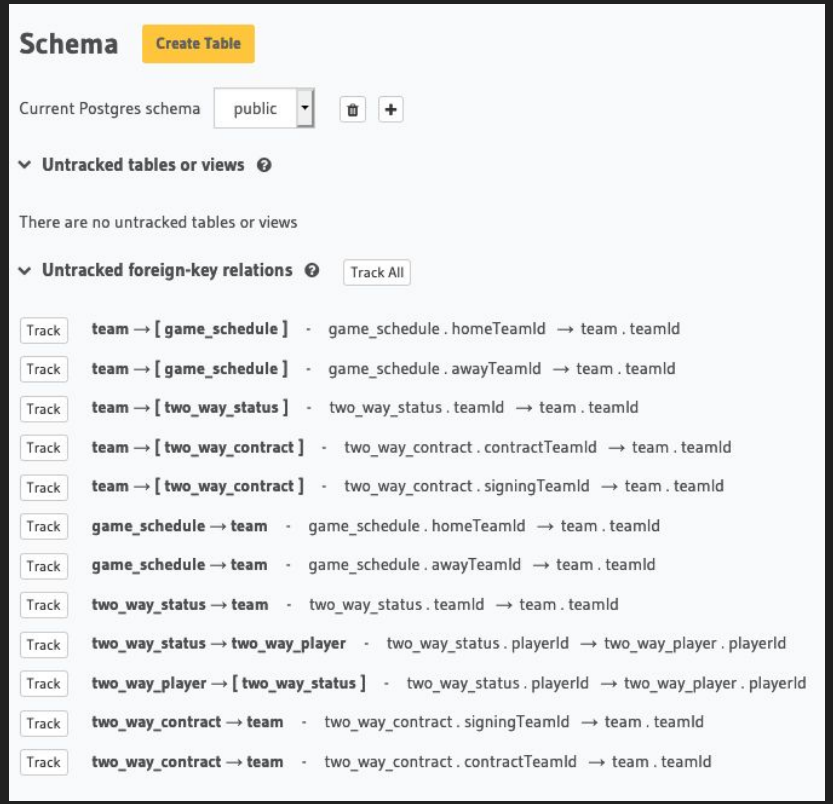
2. Database Creation



Two-Way Full Stack

2. Database Creation

The Foreign Key Table:



Two-Way Full Stack

3. Two-Way Contract Brainstorm

- ➔ Aside from the information presented within the app, more insight can be done on the canonical trends across the league on two-way contracts:
  - For example: Exhibit 10 can be placed on up to 6 NBA player contracts to possibly convert into two-way contracts later down the line. Are teams utilizing all 6 possible positions? Does this provide an incentive for these players to perform? (A dashboard to continually update who has the Exhibit placed on the contract)
  - The amount of time it takes for a two-way player to go from signing the contract to playing in the first NBA game. Because salaries are prorated downwards at the start of the season, is it advantageous (from a business standpoint) to sign players later in the season? (simple SQL join query)

Two-Way Full Stack

3. Two-Way Contract Brainstorm

- If I was asked to handle waiving and signing Two-Way's, there would have to be an consensual agreement towards *evaluating a player's value* based on the lineups pipeline that we created from the lineups.
  - As a very rudimentary start, we can see the win-loss ratio we calculated when a two-way player gets a stint of over ~5:00 in a game.
  - We can make this model more complex by possibly accounting for more advanced statistics such as wins above replacement.
  - Of course to make this as unbiased as possible, there would need to be multiple samples of players being two-way'ed at the same time, getting the same playing time, etc, which is too unrealistic. A more realistic statistical test would be bootstrapping from the entire population of two-ways across the league.

Two-Way Full Stack

3. Two-Way Contract Brainstorm

- ➔ From a technical standpoint, adding a player to the database is simple enough.
  - ◆ Hasura, the engine I used to create the backend, provides a simple UI interface that allows me to insert rows without code.
  - ◆ This allows for really quick data input from anybody, including non-technical people as well

End of Document

Jump to:
Questionnaire
Developer Test

Back to Title Slide