**FanDuel Coding Challenge**

Please answer the below questions to the best of you ability. If you have any questions do not hesitate to reach out to us for clarification. You may use any language/framework you feel comfortable with to implement your solutions but **we heavily prefer you to use Java or Python.** Please provide us with any files / instructions we will need to run your code, including installation/download instructions for anything used other than the language sdks. Please try to get this back to us within a week. If you need more time than that, please let us know.

**1.) API Creation**

See **attached** for data.

Build a server that takes the following data sources and implements the following HTTP API calls. You may store / access this data however you want (in-memory / file / etc…)

We are looking at correctness of code / handling of edge cases / how well it is factored (reusable / extensible) / commenting.

**API Calls**

We want to create a usable REST based API given the data source.

// API Calls

Teams

* **GET /nba/teams** 
  + Returns all teams
* **GET /nba/teams/1**
  + Returns just the team object of the id specified

Players

* **GET /nba/players**
  + Returns all player
* **GET /nba/players?date=01012016** 
  + Returns all players who played (i.e. had player stats) on a specific date (date format: MMDDYYYY)
* **GET /nba/players/1**
  + Returns the player object of the id specified
* **GET /nba/players/1/stats**
  + Returns all player stats objects for the specified player

Games

* **GET /nba/games**
  + Returns all games with scores (game + game state)
* **GET /nba/games?date=01012016**
  + Returns all games with scores (game + game state) for a given date (date format: MMDDYYYY)
* **GET /nba/games/1**
  + Returns the game objects with scores (game + game state) of the id specified

**Follow Up**

Explain how your code would change if you were to implement same calls for multiple sports.

**2.) Data Analysis**

We've provided a SQLite 3 database file, "sqlite.db" containing six tables:

***send\_event***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Notes** |
| batch\_id | INTEGER |  |
| to\_email | TEXT |  |
| sent\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |
| successful | INTEGER |  |

***open\_event***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Notes** |
| batch\_id | INTEGER |  |
| to\_email | TEXT |  |
| open\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |
| ip\_address | TEXT |  |
| user\_agent | TEXT |  |

***bounce\_event***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Notes** |
| batch\_id | INTEGER |  |
| to\_email | TEXT |  |
| bounce\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |

***click\_event***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Notes** |
| batch\_id | INTEGER |  |
| to\_email | TEXT |  |
| click\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |
| ip\_address | TEXT |  |
| user\_agent | TEXT |  |
| url | TEXT |  |

***unsub\_event***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Notes** |
| email | TEXT |  |
| unsub\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |
| ip\_address | TEXT |  |
| user\_agent | TEXT |  |

***event\_summary***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Notes** |
| batch\_id | INTEGER |  |
| to\_email | TEXT |  |
| sent\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |
| bounce\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |
| open\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |
| click\_date | TEXT | Timestamp in YYYY-mm-dd HH:MM:SS format |

**Background**

With the exception of *event\_summary*, each table contains data about a specific type of event in an email marketing campaign, recorded by the system that sends the emails and tracks responses. The process flow of the email marketing campaign starts with a batch send to many email addresses. The batch is identified by a common *batch\_id* number which is shared across all email addresses that were sent to in the single batch. All recipients receive similar content in the email body, although it may be slightly customised for each recipient. Each send to each individual email address is recorded in *send\_event*.

Some of the emails may later generate a "bounce" response if the email cannot be delivered - for example if the email address does not exist. These are recorded in *bounce\_event*.

The emails have embedded image tags which point to a unique URL on the email service provider's servers, so whenever the recipient opens an email and their email client attempts to load the image an event is recorded in *open\_event*. This can only be recorded if the user has their email client set to automatically load and display images.

There are links in the email body which point to URLs on the email service provider's servers, so that when a recipient clicks on one of the links, they are brought to a tracking page which records that they have clicked on the link in *click\_event* and then redirects them to the intended destination.

The *event\_summary* table contains a summary of when each type of event was triggered for each recipient email address in the batch. It has been generated by starting with *send\_event* and doing a simple (left outer) join to the other *\_event* tables to find the timestamps of the other events if they occurred.

**The task**

The task is split into the two parts detailed below. These can be done in SQL, or a combination of SQL and another programming language e.g. Python or Java (other languages are OK as well, as long as you can explain how the code works and how to run it).

**Part A**

An analyst looking at the email marketing campaign data has reported finding multiple records in the *event\_summary* table for some recipient emails. Identify the probable cause of these duplicates, and suggest and implement a fix. This may require new tables or changes to the existing one.

**Part B**

Since the *event\_summary* table was last updated, new open and click events have been logged in the other *\_event* tables. Implement some code to update the *event\_summary* table to take in to account new open and click events, which can be run repeatedly to update *event\_summary* as new events are logged over time.

**What to submit as a response**

Please provide the source code that can be run to complete the tasks along with instructions for building (if applicable) and running it. You don't need to provide the generated summary data, just your code to generate the data from the database file supplied. Please also provide notes explaining any design or implementation choices and reasoning behind them.

### SQLITE DB FILE

(Attached)

# **3.) Depth Charts:**

Most team sports have a depth chart (a ranking of each player) for each position they have. For Example in NFL: Ben Roethlisberger is listed as the starting QB and first on the QB depth chart. Landry Jones, his backup is listed as the 2nd person on that depth chart. We want to implement functionality that will manage these depth charts.

Note: Players can be on the depth chart for positions that are not their own.

**Functions To Implement:**

* **addPlayerToDepthChart(player, position, depth)**
  + Adds a player to a depth chart for a given position (at a specific spot). If you are entering two players into the same slot, the last player entered gets priority and bumps the existing player down a depth spot.
* **addPlayerToDepthChart(player, position)**
  + Adds a player at the end of the depth chart for a given position.
* **removePlayerFromDepthChart(player, position)**
  + Removes a player from the depth chart for a position
* **getFullDepthChart()**
  + Prints out all depth chart positions
* **getPlayersUnderPlayerInDepthChart(player, position)**
  + For a given player find all players below them on the depth chart

**a.) Implement the 4 functions above for NFL supporting positions (QB, WR, RB, TE, K, P, KR, PR)**

**b.) Implement the 4 functions above for MLB supporting positions (SP, RP, C, 1B, 2B, 3B, SS, LF, RF, CF, DH).**

**Attached** is some sample Java code to get you started, it can be used, adjusted, or thrown away as you see fit.

Keep in mind we potentially might add more sports in the future and we want to make it as easy as possible to add new ones.