Background & Instructions

Mighty works extensively with various stakeholders in legal funding. The primary users of Mighty are companies (hereafter referred to as “funders”) that provide cash advances to plaintiffs in exchange for the rights to a portion of the proceeds of the plaintiff’s civil lawsuit.

1. Funders can provide one or more advances per legal claim (“case”).
2. All plaintiffs have attorneys who represent their claims; otherwise, they cannot be provided legal funding.
3. A cash advance contains various financial details including: the advance amount, an agreement date and a due date.

Please push all answers to Github and share links to your solutions with us. If instructions are unclear or you get stuck, do your best to show us your thought process about the problem.

SQL Section

In Figure 1 in the Appendix, you can see a depiction of the database described in the prompt above. It may be valuable to answering questions below. If you get stumped, please write out your thoughts.

- We have decided to expand our data model by adding a new **Case Type** table. Each case in Mighty has exactly one “Case Type” (such as Motor Vehicle Accident or Other). Assuming the only information on the **Case Type** table is the primary key and the name of the case type. Please add this table to Figure 1 in the appendix. You do not need to turn this in, but you will have to write queries using this new table below.

- Using the schema described by the diagram above and your addition, construct SQL queries to answer the following questions. Use whatever flavor of SQL you are comfortable with:
    a. For each funding, return the funding_id and the agreement date.
    b. Show me all attorneys that do NOT have a phone_number
    c. What is the total amount advanced all time?
    d. For every case, return the attorney’s phone number and email.
    e. For every case, return the total amount funded.
    f. Return all cases that have attorney name is exactly “Sal Goodman”.
    g. Return all case types, sorted alphabetically by name.
    h. Return all case types with the name of the case type and the number of cases of that type, each in its own row, without duplicates.
    i. Rank the case types by the most frequent (most number of cases) to the least frequent with the number of cases of each type.

Python Section

Please answer the following using your preferred version of Python (or R).

Social Security Number Cleaning

When migrating client data, we often have to clean the data before inputting it into our database. In this example, we are attempting to clean social security numbers ("SSN"s). We have included a number of test cases your function should pass in the appendix below, but please ensure the function meets the following conditions:

- You should not use an existing library designed for this purpose.
- The function should be written to clean one SSN at a time, assuming both input and output values are strings. If passed in an empty string, return an empty string.
- The database expects that there are 11 characters in outputted SSN string in "XXX-XX-XXXX" format, where X is a integer 0-9.
- Sometimes, input data can have only a few digits of a social security number OR too many digits.
    - By convention, if the number of digits provided is less than 9, we add zeros in the front until they have 9 integers in the string (see Figure 2, row #1).
    - By convention, if there are more than 9 integers provided, we assume the first 9 are the SSN (see Figure 2, row #3).
- Hint: You do **NOT** have to use regular expressions.

Appendix

Figure 1 - Simplified ER Diagram

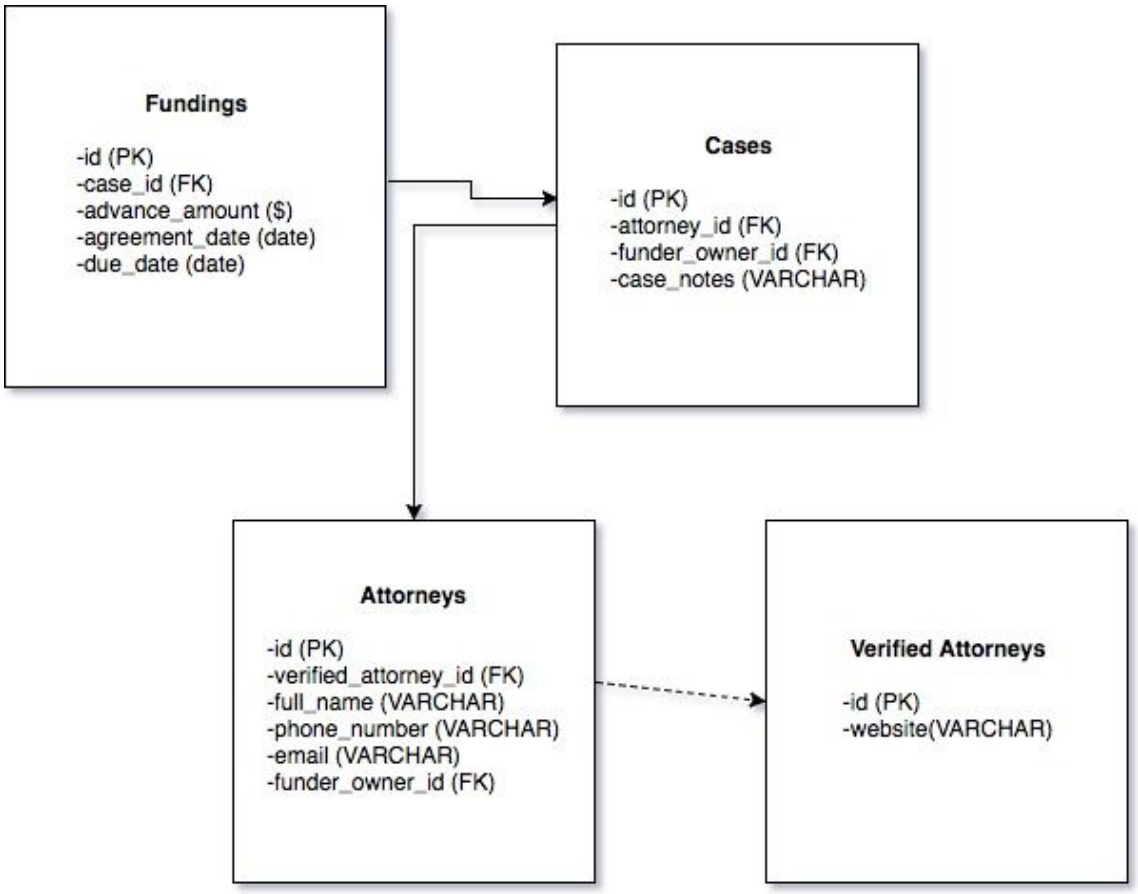

Figure 2 - Inputs & Outputs for Social Security Number Cleaning Function

Input Value **(str)**	Expected Output **(str)**
9876	000-00-9876
987654321	987-65-4321
1234567891	123-45-6789
789526345 -- client ssn	789-52-6345
XXX-12-3456	000-12-3456