There are just two problems this week, relating to normal forms, regular expressions, and searching lexemes. Owing to the (very) late arrival of this assignment, it is both fairly straightforward and it will not be due until the end of the semester.

You can accept the assignment and initial materials at the link below:

Assignment link: https://classroom.github.com/a/2dQoYGsl

1. One of the more common ways that you will receive unnormalized data is through various JSON formats. Consider then the example JSON response shown below, which might depict a response from an internal API containing information about student assignment submissions and corresponding feedback and rubric items.

```
[
  {
  "student_id": 2948294,
  "name": "Janet",
  "submitted_date": "2022-09-22",
  "assignment": "Homework 1",
  "total_available_points": 20,
  "deductions": [
    {
      "id": 1,
      "comment": "Didn't read instructions well",
      "deduction": -1
    },
    {
      "id": 2,
      "comment": "Missed a negative",
      "deduction": -0.5
    }
  ]
}, ...
```

The repository has a more complete example that you can look through, which includes multiple submissions. *Note that you can have the same deduction item showing up multiple times for a given assignment.* Your task here is to take this information and progressively create from it tables of various normal forms. I will not ask you to show the specific contents within these tables, but am instead just interested in the structure of the tables. As such, the easiest way to represent table structure (and potential relations between tables) is through ER Diagrams. For each of the below parts, I'd ask you to upload an ER diagram created by a tool such as DrawSQL to the repository.

   (a) The first step is always to get the data / table into the first normal form. As a reminder, in this form you just need to have 1 piece of information per cell, and have some primary or compound primary key that uniquely identifies each row of the table. Create an ERD for a table that would hold the above data. This will be just a single table in this case, so you really just need to worry about specifying the column names and types, as well as determining what your primary key will be.

(b) To put the table into 2NF, you need to ensure that there are no partial dependencies present, where some columns only depend on a subset of the columns that comprise a compound primary key. If you do not have a compound primary key, then your table is already in 2NF. Break the above into more tables as necessary to ensure that you no longer have any partial dependencies in any table. Be sure to include in the ERD which tables and columns are related to others through foreign key dependencies.

(c) Finally, to ensure that your tables are in 3NF, you must ensure that there are no transitive dependencies, where one column actually depends on another (non-primary key) column, instead of the primary key. Further break apart your tables as needed to ensure that there are no transitive dependencies between any non-primary key columns. Again, be sure to include in your ERD the foreign key constraints that indicate what rows are relate.

Don't forget to include an image of the ERD for *each step* of the process above! If you could upload them back to GitHub with names like `Prob1_1NF.png`, `Prob1_2NF.png`, and `Prob1_3NF.png`, that would be very appreciated!

2. Growing up, my family had illustrated versions of many of the classics, of which one of my favorites was *The Count of Monte Cristo* (apparently I had a taste for elaborate revenge stories...). You can access plain-text versions of many of the classics from the Project Gutenberg website, and here I have grabbed the full text and read it into a Postgres table. *I have already done several pieces of the parsing for you,* breaking the text up into volumes and chapters. The SQL file `CountMC.sql` is a table dump that will recreate the table in your database of choice if you use `psql` to run it. I have also included `CountMC_Slow.sql` which uses `INSERT INTO` statements that you can copy over and run if `psql` still isn't working for you, as well as `CountOG_OPTIONAL.sql` with the original book read into a single cell if you wanted to try parsing out the chapters yourself.

   After running `CountMC.sql` to create the table `count_of_mc`, use it to execute the following tasks or answer the corresponding questions.

   (a) Currently the chapter headings contain both the chapter number as well as the chapter title. It would be more convenient to separate those pieces of information. Add and popuulate two new columns to the table, one called `chapter_number` and one called `chapter_title`, with types of `INT` and `TEXT`, respectively. Be a bit careful with chapter titles that also include numbers!

   (b) Add a new column called `vector_contents` which is a `tsvector` of the chapter contents. Add an appropriate index to this column for fast searching.

   (c) The Count of Monte Cristo is, inherently, a tale of revenge. What is the *sum* of all the chapter numbers whose content's contain mention of the word "revenge"?