This week's data heralds largely from the Museum of Modern Art, which makes their collection data available publically here, though I have already included everything you need in this assignment's repository. The data initially comes in two pieces, with one containing information about the works of art and the other containing information about the artists. As you will see though, some work can be done to clean things up and make the data easier to work with, so this assignment will initially be an exercise in two parts (and problems): cleaning up the data and then gaining some insight from it. **There is a lot of text here, but most of it is explanatory to try to give you plenty of guidance, so don't be intimidated.**

As per usual, you should follow the link here to accept the assignment and get access to the repository:

Assignment link: https://classroom.github.com/a/Cf40BY49

1. One of the more common ways that you will receive non-normalized 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. The next two problems are dealing with the MoMA data set. You are starting mostly from scratch here, but I have included template code in the repository that you can copy and run to create the necessary starting tables with data types that will successfully import the data. I will detail the rest of the parts of the clean-up that you should do in the below parts. *For each clean-up step, include the commands you used to perform that operation.*

   (a) (1 point) Create the initial `artists` and `artworks` tables in whatever database you like, and copy the information in from the included CSV files into the appropriate tables. If you are using the template code I provided for you, this should be simple, and you need to only include the code you used to copy in the table information. If you made your own tables with your own chosen names, please include the code for creating them as well.

   (b) (2 points) Perusing through the tables, you will likely notice one thing fairly quickly: there is a lot of duplicated information in the `artworks` table that actually describes the artists (and is also present in the `artists` table. Moreover, if you look around a bit more in the `artworks` table, you will see something unfortunate: some pieces of

art have multiple artists, and so all of that information is smashed into compound strings within each column of the `artworks` table. Good table design in relational databases dictates that, in almost all circumstances, you want to have one (and only one) value in each column of a record. In order to accomplish this, we are going to need to generate a third table, which you might call `artwork_artists`, which will have only 2 columns: one containing the artwork `object_id`s, and the other containing the artist `constituent_id`s. If multiple artists worked on the same artwork, then that same artwork `object_id` should simply appear on multiple rows with a different `constituent_id` on each. This is the same idea as to how we combined students and classes earlier in the semester.

The difficulty lies in how to accomplish this easily. We have all the needed information, but it is buried inside `artworks.constituent_id`. If you have worked with strings in other programming languages, you might want to do something like splitting apart the current `constituent_id` field at the commas, to get each individual constituent id. We can actually do the same in Postgres! The function `string_to_array(some_text, text_to_split_on)` will return an array of the individual pieces of `some_text` split at each occurrence of `text_to_split_on`. The issue then is that you have an array of the constituent id numbers (which are still actually strings). What you really want are these same numbers not in an array, but split across rows. And we've actually already seen a method of doing this, though it only has come up once in the text of Ch 6: `UNNEST`. Official documentation here.

Create the `artwork_artists` table such that it has one row for each artwork id and contributing artists.

(c) (1 point) You might think your `artwork_artists` table looks great! You are probably wrong, unfortunately. In theory, each piece of art with a corresponding artist should show up *once* in your created table. Is that the case? Can you figure out why this happened? Fix it by removing the duplicates (or recreating the table with only unique combinations). While you are there, the `constituent_id`s are just increasing integers, so it would be nicer to work with them as actual integers instead of the strings they came in as when you split the original string. (And importantly, this removes any extra spaces that might be still lurking around as well) Change that column data type to be an integer (you may need to use something like `USING constituent_id::INT` at the end of your `ALTER TABLE` statement to get it to go through).

(d) (2 points) You now have three related tables, but no constraints are in place. Alter the tables to add primary keys `object_id` and `constituent_id` to `artworks` and `artists`, respectively. For `artwork_artists`, create a compound primary key that includes both columns, which we can do since we just confirmed that those combinations are all unique. Add in the necessary foreign keys as well to link the `artwork_artists` table columns back to the necessary reference table columns.

(e) (1 point) Now that we have better related the tables, there is really no need for the `artworks` table to still have all the columns containing information that is already

in the `artists` table. Remove those 7 columns from `artworks`.

(f) (1 point) If you check for missing information in the `artists` table, you'll realize that there is plenty, but much of it is information that we probably do not have. One thing you might notice though is that there are artists with no recorded `nationality`, but who *do* have something about a nationality mentioned in their bio. These probably could be fixed, but there are a lot of them and no good automatic way to do so, so I'm not going to ask that of you! One other aspect of this table though is that missing or "not yet occurring" birth and death dates are recorded with 0's, which seems potentially problematic if you wanted to do calculations with those columns. Change all instances of 0 in the birth year (`begin_date`) or death year (`end_date`) to `NULL` instead.

(g) (2 points) Moving over to the `artworks` table, take a look at the `date` column, which is when the artwork was supposed created. This column is an absolute mess of formatting. Some are years, some are ranges of years, some are specific days, and everything in between. For the most part though, each record contains **at least one** 4 digit date somewhere amidst the text. While grabbing just this 4 digit date may not be perfectly precise, it would surely be more useful than the current state of things. How can we accomplish that date extraction though? We'll be talking about working with text and regular expressions more soon, but for now, you can extract the first 4 digit date from any string using:

```
substring(date_text FROM '\d{4}')::INT
```

This extracts the part of the string that matches the regular expression `'\d{4}'`, which stands for "four numeric digits adjacent to one another". Add a new column named `date_int` to `artworks` and populate its rows with the first 4 digit date that appears in the row's original `date` column. If you want a quick check that this worked, if I sum all those years in the entire column, I get a value of 263617360.

3. (10 points) Clean up time is over! Now you can proceed to use the data set to answer the following questions. As per usual, show all queries that you used to arrive at your solution.

   (a) What is the title of the piece of art with the most collaborators/contributors at the Museum of Modern Art?

   (b) Who are the top 5 *painters* (those whose work falls within the Painting classification) to have the most artwork acquired by the MoMA?

   (c) What is the median age of a piece of art when the museum acquires it?

   (d) What artist has pieces in the MoMA collection across the greatest number of different classifications?