Assignment #7

👀 Click on this link: https://classroom.github.com/a/PwXuZTmJ (https://classroom.github.com/a/PwXuZTmJ) to accept this assignment in GitHub classroom. This will create your homework repository. Clone your new repository.

In this homework, you'll:

1. Create an ER Diagram by inspecting the tables
2. Run queries that involve:
    - joins
    - subqueries
    - aggregation and `HAVING`
3. Bring csv data into postgresql and create a normalized data model representing the data
4. Examine a data set and create a normalized data model to store the data
5. Manually write DDL commands (`CREATE TABLE` statements) to implement your data model

Part 1: Create an ER Diagram by inspecting tables

Overview

The sample data, `nation_subset.sql` contains information about countries (such as population, languages, etc.). This data is sourced from a site for another relational database (https://www.mariadbtutorial.com/), so the import statements have been adapted to postgres. The data is contained in multiple tables.

This is a **practice** data set:

- the original site does not provide any details on the provenance of the data nor does it have a license
- do not use this data for research purposes
- we are using it to practice joins and subqueries
- it's small enough to easily understand, but big enough that simply looking at the data wouldn't be a replacement for writing proper sql queries
- the subject matter is straightforward (countries, continents, etc.), allowing you to focus on the mechanics of writing SQL

Import

Run the sql in `nation_subset.sql` in any way that you like:

- in psql: `\i`
- using psql with redirection: `<`
- using the tools that is provided by your IDE (for example, if you're using pgadmin or DataGrip, the user interface allows for importing data)

ER Diagram Requirements

⚠️ Please make sure to check that your diagram follows these requirements

Create a directory called `img` and place your ER diagram in that folder.

1. examine the tables created by the import script. Create an **crow's foot** ER Diagram – **as an image file** called `img/part1_nation_subset_er_diagram` (use png, jpg, etc. as the extension) – that shows:
    - tables
    - each table's columns and types of columns
    - the relationships between tables
2. again, the **resulting diagram should be in an image format**
3. if the tool that you you use auto generates foreign key columns, you do not have to explicitly add foreign keys yourself
4. **explicitly show any "join" tables required for many-to-many relationships**

    - in class, we did not create an explicit table
    - instead we let the modelling tool that we used generate the join table when we exported
    - so rather than doing this, add the intermediary join table explicitly in the diagram
5. 👀 the **actual** ER diagram for this dataset is available online, but please refrain from viewing it until you've attempted to create your own ER diagram
6. note that there are some tables that do not have foreign keys, but there is a *likely* relationship between them
7. ⚠️ ⚠️ ⚠️ in the `README.md` file, document the following:
    - under a heading, `## Part 1: Create an ER Diagram by inspecting tables` ...
    - describe the kind of relationship between each table (for example: some_table has a one to many relationship with another_table)
    - *why* you think this kind of relationship exists (for example: ... this relationship exists because of a foreign key in some_table that references another_table)

You can use **any drawing tool that you'd like to do this →**

1. pen and paper: hand-draw your digram and take a picture of it
2. pgAdmin (https://www.pgadmin.org/) also has a beta ER diagram tool
3. online drawing tools: you can use any online drawing tools that you like, such as google draw, lucidchart, etc.
4. traditional diagramming tools: dia (http://dia-installer.de/), visio, etc.
5. ...or even vector drawing tools such as illustrator or inkscape 🐙 there are multiple ways to model the scenario described, and there are clearly ambiguities in the description

Part 2: Run Queries

In `src/part2_queries.sql`, answer the following questions by writing queries. Above each query, write a comment containing the question number and the first line of question text. For example:

```
-- 1. Show a report containing a and b...

SELECT a, b
FROM foo
INNER JOIN baz ON baz.foo_id = foo.foo_id
WHERE c = 'd';
```

1. Show the possible values of the `year` column in the `country_stats` table sorted by most recent year first.
    - only show years
    - sort by year descending
    - partial example output:

    ```
    +----+
    |year|
    +----+
    |2018|
    |2017|
    |2016|
    ```

2. Show the names of the first 5 countries in the database when sorted in alphabetical order by name.
    - only show country names
    - sort results in alphabetical order
    - example output:

```
+--------------+
|name          |
+--------------+
|Afghanistan   |
|Albania       |
|Algeria       |
|American Samoa|
|Andorra       |
+--------------+
```

3. Adjust the previous query to show both the country name and the gdp from 2018, but this time show the top 5 countries by gdp.
    - show the country name and gdp
    - sort in descending order of gdp
    - only show the first 5 rows
    - example output:

```
+--------------+--------------+
|name          |gdp           |
+--------------+--------------+
|United States |20494100000000|
|China         |13608200000000|
|Japan         |4970920000000 |
|Germany       |3996760000000 |
|United Kingdom|2825210000000 |
+--------------+--------------+
```

4. How many countries are associated with each region id?
    - show both the region id and count
    - label the count column country_count
    - order the report by the country_count descending
    - partial example output:

```
+---------+-------------+
|region_id|country_count|
+---------+-------------+
|1        |24           |
|11       |20           |
|5        |18           |
```

5. What is the average area of countries in each region id?
    - only show the region id and average area (labelled as avg_area) of countries in that region
    - round the result to the nearest whole number
    - sort from least to greatest avg_area
    - partial example output:

```
+---------+--------+
|region_id|avg_area|
+---------+--------+
|25       |16      |
|23       |443     |
|7        |846     |
|1        |9768    |
|21       |58372   |
|4        |87759   |
|22       |108155  |
|10       |123162  |
|24       |156587  |
|19       |188843  |
|5        |267811  |
```

6. Use the same query as above, but only show the groups with an average country area less than 1000
   - use HAVING (https://www.postgresql.org/docs/13/sql-select.html#SQL-HAVING) to do this
   - example output:

```
+---------+--------+
|region_id|avg_area|
+---------+--------+
|25       |16      |
|23       |443     |
|7        |846     |
+---------+--------+
```

7. Create a report displaying the name and population of every **continent** in the database from the year 2018 in millions.
   - show only name and total population (in a field called `tot_pop`)
   - sort from greatest `tot_pop` to least `tot_pop`
   - example output:

```
+-------------+-------+
|name         |tot_pop|
+-------------+-------+
|Asia         |4376.91|
|Africa       |1259.76|
|Europe       |717.22 |
|North America|569.63 |
|South America|394.53 |
|Oceania      |40.94  |
+-------------+-------+
```

8. List the names of all of the countries that do not have a language.
   - only display the country names
   - no specific ordering is required
   - example output:

```
+--------------------------------------------+
|name                                        |
+--------------------------------------------+
|Antarctica                                  |
|French Southern territories                 |
|Bouvet Island                               |
|Heard Island and McDonald Islands           |
|British Indian Ocean Territory              |
|South Georgia and the South Sandwich Islands|
+--------------------------------------------+
```

9. Show the country name and number of associated languages of the top 10 countries with most languages
    - only display the name of the country and the count (name this field `lang_count`)
    - sort by most languages to least languages
    - only show the top 10
    - example output:

```
+------------------+----------+
|name              |lang_count|
+------------------+----------+
|Canada            |12        |
|China             |12        |
|India             |12        |
|Russian Federation|12        |
|United States     |12        |
|South Africa      |11        |
|Tanzania          |11        |
|Iran              |10        |
|Kenya             |10        |
|Mozambique        |10        |
+------------------+----------+
```

10. Repeat your previous query, but display a comma separated list of spoken languages rather than a count (use the aggregate function for strings, string_agg (https://www.postgresql.org/docs/13/functions-aggregate.html). A single example row (note that results before and above have been omitted for formatting):
    - no alias is required for the string aggregation function
    - example output:

```
+------------------+-----------------------------------------------------------+
|name              |string_agg                                                 |
+------------------+-----------------------------------------------------------+
| ...              | ...                                                       |
|Kenya             |Gusii,Kalenjin,Kamba,Kikuyu,Luhya,Luo,Masai,Meru,Nyika,Turkana  |
| ...              | ...                                                       |
```

11. What's the average number of languages in every country in a region in the dataset? Show both the region's name and the average. Make sure to include countries that don't have a language in your calculations. (Hint: using your previous queries and additional subqueries may be useful)
    - only include name and average (as a field called `avg_lang_count_per_country`)
    - at most, `avg_lang_count_per_country` should have one decimal place
    - sort by greatest `avg_lang_count_per_country` to least `avg_lang_count_per_country`
    - example output:

```
+-------------------------+--------------------------+
|name                     |avg_lang_count_per_country|
+-------------------------+--------------------------+
|Eastern Europe           |6.1                       |
|Central Africa           |6.1                       |
|Western Africa           |6                         |
|Southern and Central Asia|5.9                       |
|Southeast Asia           |5.9                       |
|Southern Africa          |5.8                       |
|North America            |5.6                       |
|Eastern Asia             |5.4                       |
|Baltic Countries         |5.3                       |
|Eastern Africa           |5.1                       |
|Western Europe           |5.1                       |
|Central America          |4.4                       |
```

12. Show the country name and its "national day" for the country with the most recent national day and the country with the oldest national day. Do this with a *single query*. (Hint: both subqueries and UNION (https://www.postgresql.org/docs/current/typeconv-union-case.html) may be helpful here). The output may look like this:

```
   name      | national_day
-------------+--------
East Timor  | 2002-05-20
Switzerland | 1291-08-01
```

Part 3: Examine a data set and create a normalized data model to store the data

Download / Documentation

Download and read documentation about the .csvs linked to from the FDA CAERS website (https://www.fda.gov/food/compliance-enforcement-food/cfsan-adverse-event-reporting-system-caers):

- Find and download the **csv** version of the data form the link above
- Read the documentation about the fields contained in these files (https://www.fda.gov/media/97035/download)

Create a Staging Table

Write a short `.sql` script to create a temporary staging table called `staging_caers_event_product` to store the downloaded data. This will house your initial data import, but you'll eventually create new tables to store this same data (though you won't have to populate them).

- In `src/part3_01_create_staging.sql` …
- Write a `DROP TABLE IF EXISTS staging_caers_event_product` statement
- Then write a `CREATE TABLE staging_caers_event_product` with appropriate columns
  - The types can be loose… for example, using `text` rather than `varchar`; you can fix these later
  - 👀 It will be **extremely** helpful to immediately add an artificial primary key to the rows right on import
  - If you need some help with this, you can grab the `CREATE` statement from the slides on conditionals (../slides/db/conditionals.html) (just make sure the table name being created is prefixed with `staging_`
- Finally, in `src/part3_02_import.sql` add a `COPY` statement that actually adds the data to the staging table from the downloaded csv

Exploring the Data

In `src/part3_03_explore.sql` write at least 4 sql statements to explore the data that you have to:

- Reveal how columns may be related to each other
- View the kinds of values that exist in each field
- Determine what column or combination of columns provide unique values / candidate keys

Write statements and comments in `src/part3_03_explore.sql`, and document their output in comments underneath the query

- Add a comment above each query describing what the query is trying to achieve.
  - Example format:

```
-- 2. this query tries to determine whether or not report id is unique
YOUR QUERY UNDERNEATH;
-- +-----------+
-- | report_id |
-- +-----------+
-- | 1234      |
-- | 5678      |
-- +-----------+
```

- Then add a comment showing the partial output of the query
  - Paste the output of each query; try to limit the queries to under 10 rows (or truncate what you've pasted), and be judicious about which columns you include
    - If you're using DataGrip to do this, then you'll have to:
      1. go to the upper right of the markdown window
      2. click on the dropdown to the left of the down and up arrow icons
      3. select `Pretty`
      4. copy the output table
      5. paste into your file and use `--` to comment (as shown above)
    - Of course, if using psql, you can simply copy and paste from the commandline into a markdown code black (fence with 3 backticks)
  - Write a sentence interpreting the results of your queries underneath the pasted output (in `README.md`)

Normalization, ER Diagram

Using the results of your data exploration, as well as the documentation on the data set...

Create an ER Diagram showing a normalized data model to store the same data in the staging table.

- Each table must be in third normal form
- ⚠️ **Note that even if you have a surrogate primary key, like `serial`, you'll have to find candidate keys that are composite to continue with the normalization process**
- Some design decisions are based on *your* interpretation of the data, so there may be multiple solutions
- For practicality, allow nulls where appropriate despite what some resources dictate for normalization

As in the previous part, the ER digaram can be created in any tool that you'd like and should be placed in the `img` folder.

- Export it to an image named `img/part3_03_caers_er_diagram` (use `png`, `jpg`, etc. as the extension)
- Add the image to your repository
- Underneath a header `## Part 3: Examine a data set and create a normalized data model to store the data`, show the image in your `README.md` using markdown
- Underneath the image, add an unordered list in markdown that describes some of your design decisions

Part 4: Manually write DDL commands (`CREATE TABLE` statements) to implement your data model

In `src/part4_ddl.sql`, write the create table statements to implement the data model that was created in the previous part.

- note that some tools can generate sql for you based on your ER diagram
- please try hand-writing your `CREATE` statements first rather than relying on the tool
- (of course, you can check against the sql that the tool generates)