Ariel Gutierrez

Prof. Rao Ali

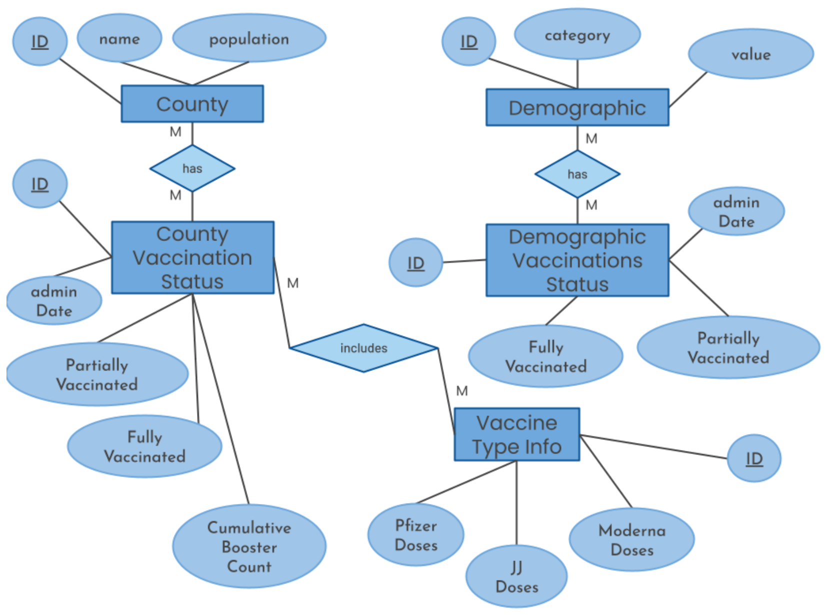
CPSC 408-01

18 December 2021

COVID Vaccine Dashboard for California

My final project was on making a COVID Vaccine Dashboard for the state of California. COVID is a major health problem in our society that does not seem to be going away any time soon. With the whole of the United States lagging behind in COVID vaccination progress, I believe that creating a COVID Vaccine Dashboard for California could help in educating the public and possibly incentivize people who are not yet vaccinated to get the vaccine. Moreover, creating a COVID Vaccine Dashboard can help those that are already vaccinated to gauge their safety while traveling throughout California, since the number of vaccinated people per county drastically changes. Because COVID cases and vaccines are a major topic in the media, there are also many COVID vaccine dashboards. However, my application also allows you to download csv generated reports that summarize the data regarding COVID vaccines.

Not only does my project address the number of vaccinated individuals in each county of California, it also provides information on the demographics of age group, gender and race/ethnicity. I achieved this by using two datasets found on the California Department of Public Health's website and creating normalized tables from them and then uploaded the data into their appropriate tables onto my Cloud Instance of MySQL. The Public IP address for the cloud instance is 35.222.150.170, the username is: rao, and the password is password1. From the two datasets I found, I was able to create the following Entity Relationship Diagram which has a total of 5 entities: County, Demographic, County Vaccination Status and Demographic Vaccination Status.



I could not get demographic data to be on the same records as county vaccination status because the data on COVID vaccines in California is very limited in the way it is recorded. It would have been helpful to have the specific demographic data for each county, but since that was not the case, I kept the data separated in the MySQL database and in the application.

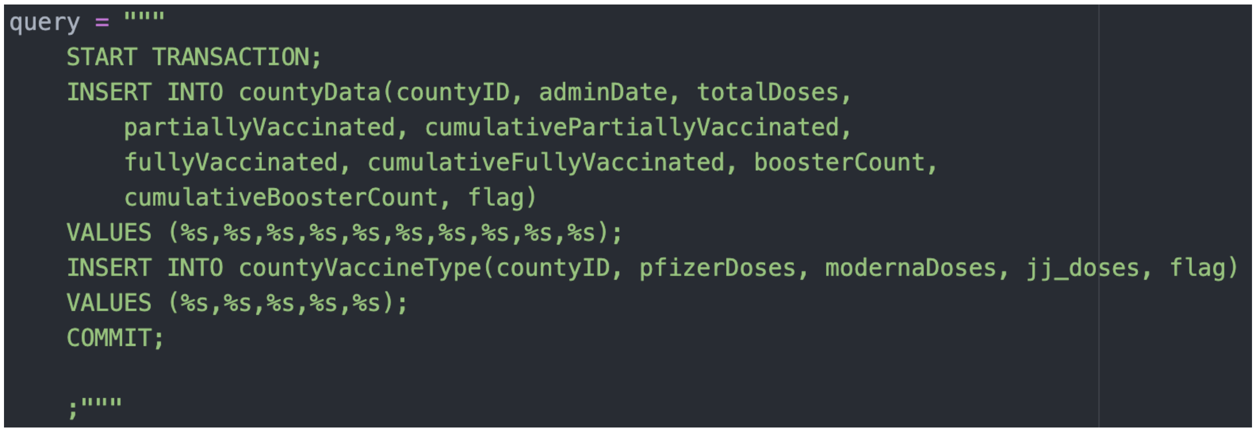
For the application design of my project, I decided to use the python command line since I am the sole member of my group and it is something that I am familiar with because of how often python is used in my classes. When starting the application, you are introduced to a menu of options which include the following: find county record, modify county total doses, delete county record, create county record information, view demographic statistics, and view all county statistics. My application allows users to create, modify, delete, and create records. The last two options are focused on generating csv file reports that could be used in a variety of ways, but I also display the contents of the csv file in readable tabular format to the user. Moreover, in my application, I used a “soft” delete, which means that I used a flag to delete something. If the flag is equal to 1 then it is considered deleted and if it is equal to 0 then it is considered as an existing record. Thus, to do queries on the database, it is necessary to filter out the records which have the flag equal to 1.

Furthermore, for the last two options of the menu for my application it was useful to use an aggregate clause since I had to group the certain demographic subcategories and I had to group the county information together. Recall that in my Entity Relationship Diagram, I have three entities that are related to each other: County, County Vaccination Status, and County Vaccine Type. So to get a lot of information with these three entities, I had to join three tables. Therefore, I created a view so that I would not have to constantly write a large join for the three tables.

```
CREATE VIEW vCumulativeCountyVaccineTotals AS
    SELECT c.name, MAX(cd.cumulativePartiallyVaccinated),
    MAX(cd.cumulativeFullyVaccinated),
    MAX(cd.cumulativeBoosterCount),
    SUM(cv.pfizerDoses), SUM(cv.modernaDoses), SUM(cv.jj_doses)
    FROM county AS c
    INNER JOIN countyData AS cd
    ON c.countyID = cd.countyID
    INNER JOIN countyVaccineType as cv
    ON cv.countyID = c.countyID AND
    cv.countyID = cd.countyID AND
    cv.id = cd.id
    WHERE cd.flag = 0
    GROUP BY c.countyID;
```

Another benefit that arose from making a view was that it also provides security benefits because I am only including things that I wanted the user to see. Any other information is left out, so it increases readability and in some cases can protect sensitive information from landing in the wrong hands.

Since I have three tables that are related to each other, I decided to make use of transactions in my project. If a system crash or failure happens, I want my application to be durable especially during phases where there are more than one MySQL query happening.

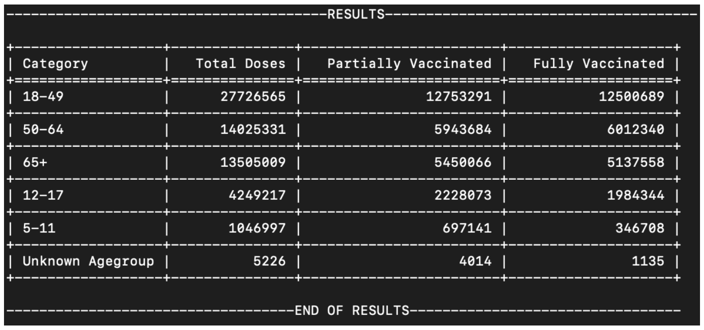


```
query = """
    START TRANSACTION;
    INSERT INTO countyData(countyID, adminDate, totalDoses,
        partiallyVaccinated, cumulativePartiallyVaccinated,
        fullyVaccinated, cumulativeFullyVaccinated, boosterCount,
        cumulativeBoosterCount, flag)
    VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);
    INSERT INTO countyVaccineType(countyID, pfizerDoses, modernaDoses, jj_doses, flag)
    VALUES (%s,%s,%s,%s,%s);
    COMMIT;

    ;"""
```

This query is used when I am inserting records in option 4. Because I am inserting information into two tables, I do not want a system crash to only do the first insert; I would either want that nothing happens or that both inserts happen. I also use another transaction when I am deleting a record by the date and county from two tables.

In order to provide readable results to the user, I used the package tabulate, which can be imported using the command: pip3 import tabulate. This allowed me to create beautiful looking graphs with minimal effort as shown below.



```
----------------------------------------RESULTS----------------------------------------

+------------------+---------------+------------------------+--------------------+
| Category         |   Total Doses |   Partially Vaccinated |   Fully Vaccinated |
+==================+===============+========================+====================+
| 18-49            |      27726565 |               12753291 |           12500689 |
+------------------+---------------+------------------------+--------------------+
| 50-64            |      14025331 |                5943684 |            6012340 |
+------------------+---------------+------------------------+--------------------+
| 65+              |      13505009 |                5450066 |            5137558 |
+------------------+---------------+------------------------+--------------------+
| 12-17            |       4249217 |                2228073 |            1984344 |
+------------------+---------------+------------------------+--------------------+
| 5-11             |       1046997 |                 697141 |             346708 |
+------------------+---------------+------------------------+--------------------+
| Unknown Agegroup |          5226 |                   4014 |               1135 |
+------------------+---------------+------------------------+--------------------+

------------------------------------END OF RESULTS------------------------------------
```