**COVID-19 Database Application Final Report**

Currently in the world, there are over two hundred million cases of COVID-19 with around five million deaths. Of those cases, there are approximately fifty million cases within the United States and are sorted into categories mainly being confirmed, death, active, and recovered cases. As of the moment, there are a great number of large spreadsheets and data files that are quite difficult and tedious to keep track of and search. The datasets that I am using are a perfect example of this as there are five different spreadsheets with all of them including hundreds and tens of thousands of records. With this amount, it is quite impossible and unreasonable to manually search through each dataset and find the necessary data, which is quite important for researchers and those who simply desire the information for their safety.

To solve this problem, I’ve decided to create an application that will make querying the dataset much simpler and efficient for the user. The program will be created through python and SQL using two files, app.py to hold the main program and helper.py to hold the helper class which includes functions used in the program. The databases will be uploaded and stored within the cloud database with the function upload\_create\_tables() which is within the helper class. The main framework of the app.py program is the use of a while-loop that includes an initial if statement of which feature the user chooses and functions from the helper class to obtain the desired results. Then the while-loop will end when the user chooses to exit, signaling the end of the program. The helper class consists of the app.py’s main functions such as creating the database, creating views and indexes, and all the main features used in the program. As for the features, the user will be able to search for whole records, search for attributes of each dataset, insert, delete, and update records quickly without needing to look at the datasets. For each of these options, the user will start by choosing which table they want to query from which include the total cases per every country in the world, the latest cases per country, the cases that occurred per day within each country from the timeline of January 21, 2020 to July 26, 2020, the total cases that occurred each day from the same timeline, and the coordinates of longitude and latitude of each country and some provinces if there are any. Some additional features of the application include getting a description of each attribute in case the user is unaware of them, being able to import another dataset into the database, exporting a data table to csv format, rollbacking an action, and committing an action.

So first, I have implemented a search function that will print records that follow the user’s conditions based on filters of the attributes. Two more tables have been added for this function which are the combinations of the world cases, latest cases, and country coordinates and the combination of total cases per day and total cases per day per country. These tables are created from forming views with the create\_view() function in the helper class and through multiple joins of the main tables. After they choose their desired table, the user will be given options to filter the data with non-numerical filters and print only certain records, print all the records, or print a specified number of records starting from the top. For example, if the user wanted to print from the world total data table and desired to see the total amount of all types of cases within just the United States, they would first choose option one for the table, then choose option two for the attribute filtering with “Country”, then enter the country name, “USA.” These inputs would allow the program to print all the records with their country listed as “USA.”

The next feature within the application is a search function that more specifically searches through the records through attributes and prints specified columns. The user will again have to choose one of the five main tables to query from. Then, they will choose if they want to print a non-numerical(ID is categorized as non-numerical) or numerical attribute within the tables. If they choose non-numerical attributes, then the program will simply output the whole column of the chosen attribute. If numerical attributes are chosen, then the user will have to choose if they want any groupings with non-numerical attributes and if they want to use any aggregates which include getting the max, min, sum, average, count, and count of records that have more or less of a specified number for that attribute. For the count feature that takes the user’s specified number, the program uses subqueries to count the number of records that are within the filtered table with records that have a greater or less of the specified number with attribute. The user can also choose to have no aggregates and just the column as well. Also, if the user chooses to include grouping, they must have chosen the option of no filtering/aggregating.

More main features include the insert, update, and delete options. For the insert feature, the user will choose a table and be able to insert a record by inputting the specified values for each attribute listed by the program. The update feature works similarly to the initial search feature where the user chooses a table and an attribute from their corresponding attributes to update. The user will then also choose which attribute to use to filter the changing the area. Then the user will specify what to change the record or record’s attributes to and where they want to update. Lastly, the delete feature is similar to the update feature where the user will once again choose a table and attribute to filter with. Then the user will specify the attribute’s value to delete the records matching that value. There is also an option to delete records with a null value within a certain attribute that the user will specify.

Lastly, the additional features include giving the descriptions of the attributes for each table, importing a new dataset into the database, exporting a table from the database to a csv format, committing, and rollbacking. First, the descriptions option will provide a definition of the chosen attribute within the chosen table for the user to understand what each column represents. Second, the importing feature allows the user to specify a new csv dataset and create a new table to insert its values, which is important due the fact that COVID-19 is still ongoing and more data will surely be created. This feature requires the user to have SQL knowledge as they must input queries for the creation of the table and insertion of the dataset. The third feature is the option to choose a table from the database and create a csv file with its records. The user will specify which table and input a name for the file with a .csv, then the program will copy the records to the newly generated csv file. Next, I have included a rollback feature that will essentially undo any an uncommitted action by the user that affects the database, which would include insert, delete, and update. Then lastly, the commit feature causes all actions that affect the database to become permanent and undoable unless another action takes place to specifically change the action back.

The program is activated within the command prompt console and has proved to be working and complete. The program successfully returns records the user searches for with the attribute filters, properly inserts, updates, and deletes records the user specifies, and perfectly carries out the rest of the features as well. Ultimately, the application successfully solves the problem of tedious, manual work in managing and querying datasets with its ability to search and modify the data by simply asking the user for their goals and criteria.

The schema for this project includes five entities that represent the tables for the dataset which, the descriptions are listed above, include World Data, Country Latest, Day Country, Day Total, and Country Coordinates. As for attributes, all the tables, excluding Day Total, have IDs as their primary key. All of them also, excluding Country Coordinates, have similar attributes with Confirmed, Death, Active, and Recovered cases with some differences in each table, such as Continent, Population in World Data and Date in Day Country and Day Total. Country Coordinates includes separate attributes with Latitude and Longitude. As for relationships between the entities, there are some relationships such as Country Coordinates showing the location for most of the countries in Country Latest, Day Country lists the cases per day for most countries in Country Latest, Country Latest holds only the latest cases which are a part of the cases held in World Data, and Day Total holds the total cases per day within Day Country. Although there are relationships, there are different numbers and names of Countries within each table and do not match completely. Therefore, I have deemed that it is inefficient to use foreign keys for this project as the tables will be queried from separately.