CSE 460 Sona Milestone 2

Jacob Goldverg

Univeristy at Buffalo

UB# 50218299

Team: Sona 460
Buffalo, US

jacobgol@buffalo.edu

***Abstract*—The overview of the importer.py which is a python CLI that provides simple access to the John Hopkins Covid-19 dataset. The idea is to load many days into a single table, using a strict primary key.**

I. Problem Statement

Johns Hopkins exposes daily covid reports on their GitHub, the issue we have found is that it is extremely difficult to manipulate this data inside as the files are all CSV's. The project will construct a database out of these files that will enable users to use this data in ways they were unable to in a simple Excel sheet. As there are new reports published for every country in the world every day it quickly becomes impossible to just use a simple Excel sheet to manage and manipulate these reports. Data surrounding COVID-19 varies heavily and changes very frequently which is why a database would be great to house the data, as well as manipulate it. This project will be a CLI application that provides simple arguments to build powerful queries that increase interpretability and manageability of the database.

II. Target User

The users of our database will be the general public, this means people who are already aware of how to connect into a database and manipulate some tables will be direct users. An example of a user is the frontend for our project as it will be able to do all of the operations specified above. Another example of an administrator is a Linux hobbyist that is trying to understand the data surrounding COVID-19. This would enable this person to study how covid affects their state to the entire United States.

The administration of the database will be done through a Python CLI application. The Docopt framework provides a simple CLI creation recipe and using Postgres SQL I was able to access the local deployment of Postgres and create a CLI application that makes it easy to insert/create/destroy/delete/update/alter operations on the database. The CLI is a completely public application on my Github account so anyone is able to use my code to learn more about the data. The CLI has checks to ensure that the operation cannot be harmful to the DB and the data within it. So, we are essentially configuring our Python CLI to be the sole administrator that many people can sign up to use and push all their requests through.

A. Indicate the Primary key and the Foreign keys for each relation.

The database will have one tables as of right now, the core table is called "coviddays" and will house the daily reports provided by Johns Hopkins, and in this table the primary key is the UID of every row with the Last_Update timestamp. This way we can store multiple reports from the day in varying regions and not have to worry about a potential conflict between having many reports of the same day as a UID is completely unique. Any overlap in the primary key will result in that row being updated.

B. Write the detailed description of each attribute(for each table), its purpose and datatype.

- Coviddays Table:
  - UID is an Integer, the unique row of the table
    - Cannot be null, part of primary key
  - Index: this is a new column in the table and represents the index of any given row and it is not the primary key
    - Cannot be null
  - Province_State varchar, the state or province the row is corresponding to.
    - Cannot be null and no default is available
  - Last_Update timestamp using UTF-8, the last time this measurement was updated
    - Cannot be null and no default is available and is part of the primary key
  - Lat: is latitude and is a Double Precision, initially we assumed we could use a Long.
    - Can be null and no default value
  - Long_: longitude: Double Precision, I initially assumed that I could use a Long.
    - Can be null and no default value
  - Confirmed: Integer, total number of cases this row(day).
    - Cannot be Null and has a default value of 0.
  - Deaths: Integer, total number of deaths due to covid.
    - Default is set to 0 and this cannot be null.
  - Recovered_Integer: total number of recovered cases this row(day).

        - Default value is 0 and this value cannot be null.
    - Active: Integer, total number of active cases this row(day).
        - Default value is 0 and this value cannot be null.
    - Incident_Rate: Double Precision
        - Default value is 0 and this value cannot be null.
    - Total_test_results Double Precision.
        - Default value is 0 and this value cannot be null.
    - People_Hospitalized Double Precision.
        - Default value is 0 and this value cannot be null.
    - Case_Fatality_Ratio Double Precision.
        - Default value is 0 and this value cannot be null.
    - Testing_Rate Double Precision.
        - Default value is 0 and this value cannot be null.
    - Hospitalization_Rate Double Precision.
        - Default value is 0 and this value cannot be null.
    - ISO3: varchar that holds the three letters representing the ISO.
        - Cannot be null
    - People_Hospitalized: A Double Precision value that holds the number of people hospitalized.
        - Can be null and has no default value.
    - People_Tested: Double Precision value that holds the number of people tested on that day.
        - Cannot be null and has no default

The CLI application when importing csv files will automatically fill any null values with 0's, this way every single row will never have null values.

III. Indicate each attributes default value

This was done in step 2.

IV. Explain the actions taken on any foreign key when the primary key is deleted.

For every entry we have a UID and Last_Updated date that allows every single row to be a totally unique row inside of our "coviddays" table. This way we can store many days regarding how states/provinces have been doing without worrying about uniqueness. The (UID, Last_Updated) is the primary key for the "coviddays" table. The Index column is a Secondary key which allows me to maintain the order of rows in the csv file as it is simply modular 50 for every entry in the file. Dropping any row from this table will have a cascading effect on the Views that the CLI creates. Provided below we have our "coviddays" table with a single day from the United States loaded into it. The Foreign key problem is not applicable to our schema design, the reason being is we have no foreign keys being used. But our schema does not support null and will "fix" every null to be 0, including for string data.

V. The CLI commands supported

I decided that we only needed one table for this project called "coviddays". This table is responsible for housing all of the rows needed for the CLI to provide the user with clear and simple interface to manipulating and understanding the data from John Hopkins University.

The CLI focuses on providing a SQL free API that allows the user to still manage the database, and gain some statistics about the table. All the of SQL this project used can be found in the importer.py file where SQL statements are written into strings and then executed using a database connection. This is done by having a large number of commands where each command facilitates some new piece of functionality for the user.

Here is a list of all the commands the CLI supports with the corresponding functionality.

1. load_csv: Reads the csv file into a panda's DataFrame and then writes it to the coviddays table. Where if the table exists, it does a copy and concatenates the previous table to the csv data so it always preserves the data contained as well as add any extra columns if necessary.
2. get_date: Takes in three optional parameters (year, month, day) and returns all rows with the given combination of parameters.
3. drop_table: An administrator function that drops a table with the supplied name.
4. drop_column: Drops a column from the given table name as column name
5. alter_covid_table: This function checks that there is a primary key and unique constraint on the coviddays table. If there is no private key then it creates one using the columns (uid, last_updated) and creates a unique constraint on the exact same columns.
6. Stats: Takes in state, and column name and returns the average amount found in that column given the state
7. Print_table_stats: This takes in a table name with a default for coviddays. Ouputs: [(oid, schemaname, relname, heap_blks_read, heap_blks_hit, heap_blks_icache_hit, idx_blks_read, idx_blks_hit, idx_blks_icache_hit, toast_blks_read, toast_blks_hit, toast_blks_icache_hit, tidx_blks_read, tidx_blks_hit, tidx_blks_icache_hit] which are all different statistcis regarding the current state of the table. It also outputs each column, name, type and default value.
8. Sign_up: Takes in a username and password to create a postgres user that can access the coviddays table.
9. Sign_in: Takes in a username and password and uses them in the connection string to postgres. This ensures that every

user is authenticated with the DB before they can execute queries. If the user decides not to log in then the default user postgres will be used.

VI. The SQL statements

To find the SQL statements used for any command please navigate to the GitHub repository and look at the importer.py file. There you will find functions with respective implementations.

To show some of the queries that I used with the respective functionality:

1. Create Primary Key Constraint: ALTER TABLE coviddays ADD CONSTRAINT pk_coviddays PRIMARY KEY (uid, last_update)
2. Create Unique constraint: ALTER TABLE coviddays ADD CONSTRAINT uni_coviddays UNIQUE(uid, last_update)
3. Sign up: CREATE USER name WITH PASSWORD pwd
4. Get rows on Date by day, month or year: SELECT * FROM coviddays WHERE EXTRACT(day FROM 'last_update') = day AND EXTRACT(month FROM 'last_update') = month AND EXTRACT(year FROM 'last_update') = year
5. How I check if a User already exists in the database: SELECT 1 FROM pg_roles WHERE rolname='username'

GitHub link: https://github.com/jgoldverg/dqmlProject1

VII. Loading Large Data

The table is relatively trivial with only 17 dimensions but the large size comes from the number of day.csv files we can load into it. The queries done by the cli are not intensive as they do not apply large transformations.

Most of the operations are trivial in that they do simple create/update/delete/alter. This means that with a large number of days loaded into Postgres SQL the query time drops but by a logarithmic amount. The largest amount of files I added into the table is 30 which

VIII. The conclusion

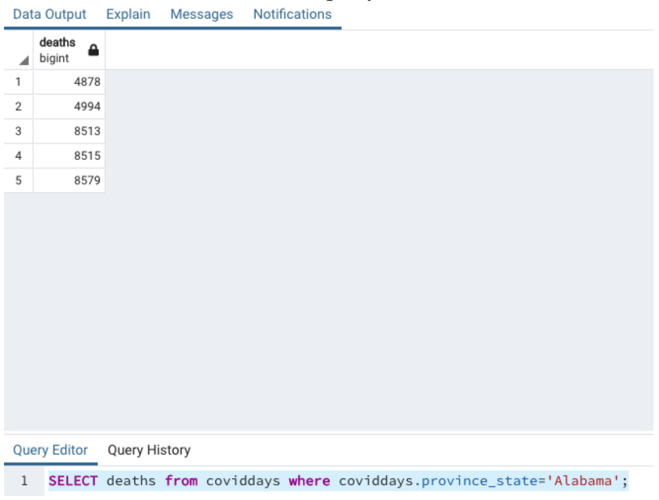
The project I created is for the use of managing the John Hopkins COVID-19 day to day files. This entailed creating a python script that handled all cli side which is the frontend of the project. It's a way for a user to directly get involved with my work. Then the CLI exposes various commands that execute different routines with arguments and flags. The cli follow POSIX style use, which means it is extremely standardized for arguments and flags.

The database work for this assignment revolved around creating a key system that was simple and unique resistant this way not two rows from various kinds of files will collide. This allows the table to retain completely unique entries across any number of years while providing a bucketing system using UID. The functionality my cli offers is provided above but does cover the general requirement of: create/load/alter/update/destroy. I also ensured security of the assignment by utilizing PostgresDB user/role system. The power of this, is every query ran by any user is logged so I am able to always find what queries users have done.

The future of this project would be to expand more on the kinds of statistics the cli is capable of showing. It would be nice to expand it to show maybe the std deviation and other kinds of metrics as data gets put in. With the goal to continue simplifying the way this data is used in a transparent way.

IX. The images

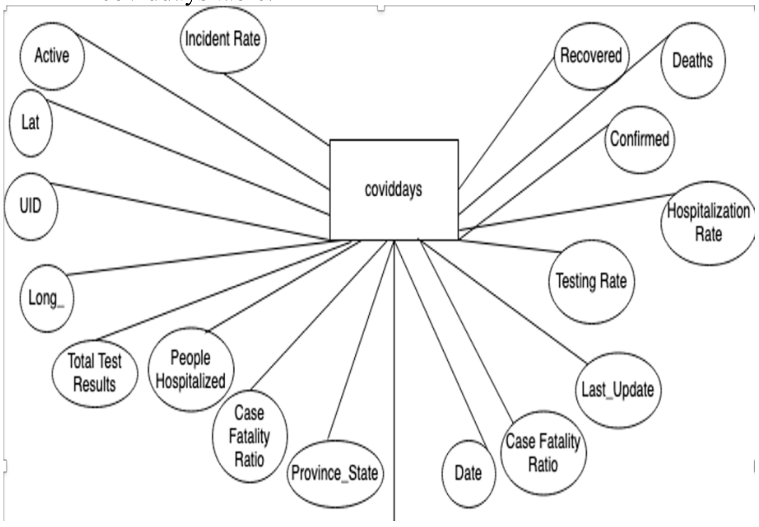
1. Here we have an image that shows the output from all deaths in Alabama for any day. We can tell that because we have 5 rows then are 5 days in the database from when this query was ran.



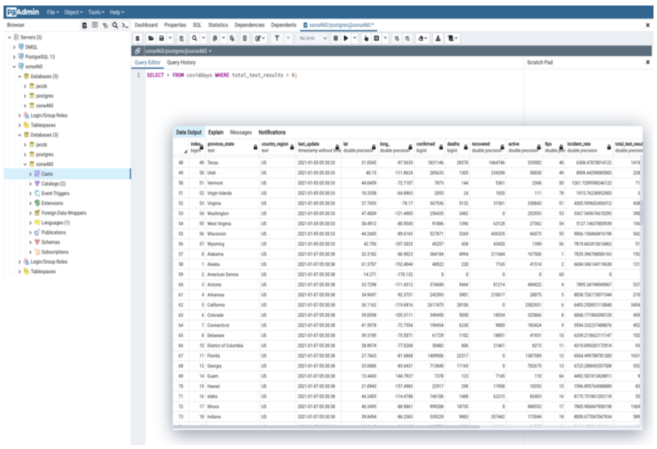
```
   deaths
   bigint
1    4878
2    4994
3    8513
4    8515
5    8579
```

```
SELECT deaths from coviddays where coviddays.province_state='Alabama';
```

2. Here we have the E/R Diagram used for our coviddays table.

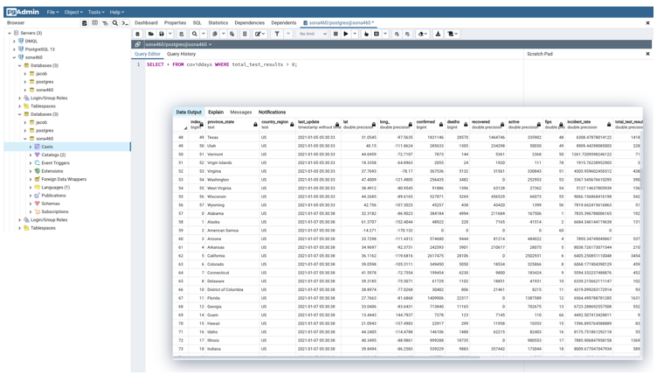


3. This image the database with a single day of covid testing results.



```
SELECT * FROM coviddays WHERE total_test_results > 0;
```

	index bigint	province_state text	country_region text	last_update timestamp without time	lat double precision	long_ double precision	confirmed bigint	deaths bigint	recovered double precision	active double precision	fips double pre	incident_rate double precision	total_test_resul double precisio
48	49	Texas	US	2021-01-05 05:30:33	31.0545	-97.5635	1831146	28570	1464746	335902	48	6308.47878014122	1418
49	50	Utah	US	2021-01-05 05:30:33	40.15	-111.8624	285633	1305	234298	50030	49	8909.44298085003	228
50	51	Vermont	US	2021-01-05 05:30:33	44.0459	-72.7107	7873	144	5361	2368	50	1261.7209598246122	71
51	52	Virgin Islands	US	2021-01-05 05:30:33	18.3358	-64.8963	2055	24	1920	111	78	1915.76238952903	3
52	53	Virginia	US	2021-01-05 05:30:33	37.7693	-78.17	367536	5132	31561	330843	51	4305.959602456512	438
53	54	Washington	US	2021-01-05 05:30:33	47.4009	-121.4905	256435	3482	0	252953	53	3367.545676610295	390
54	55	West Virginia	US	2021-01-05 05:30:33	38.4912	-80.9545	91886	1396	63128	27362	54	5127.14637805939	156
55	56	Wisconsin	US	2021-01-05 05:30:33	44.2685	-89.6165	527871	5269	456529	66073	55	9066.156868416198	542
56	57	Wyoming	US	2021-01-05 05:30:33	42.756	-107.3025	45257	438	43420	1399	56	7819.662415616863	51
57	0	Alabama	US	2021-01-07 05:30:38	32.3182	-86.9023	384184	4994	211684	167506	1	7835.396788006165	192
58	1	Alaska	US	2021-01-07 05:30:38	61.3707	-152.4044	48922	220	7165	41514	2	6684.346144119638	131
59	2	American Samoa	US	2021-01-07 05:30:38	-14.271	-170.132	0	0	0	0	60	0	
60	3	Arizona	US	2021-01-07 05:30:38	33.7298	-111.4312	574680	9444	81214	484022	4	7895.34749049867	537
61	4	Arkansas	US	2021-01-07 05:30:38	34.9697	-92.3731	242593	3901	210617	28075	5	8038.726173071544	210
62	5	California	US	2021-01-07 05:30:38	36.1162	-119.6816	2617475	28106	0	2502931	6	6405.250851110048	3454
63	6	Colorado	US	2021-01-07 05:30:38	39.0598	-105.3111	349450	5050	18534	325866	8	6068.171904390129	459
64	7	Connecticut	US	2021-01-07 05:30:38	41.5978	-72.7554	199454	6230	9800	183424	9	5594.332237488876	452
65	8	Delaware	US	2021-01-07 05:30:38	39.3185	-75.5071	61729	1102	18851	41931	10	6339.215662111147	102
66	10	District of Columbia	US	2021-01-07 05:30:38	38.8974	-77.0268	30482	806	21461	8215	11	4319.099283172914	93
67	11	Florida	US	2021-01-07 05:30:38	27.7663	-81.6868	1409906	22317	0	1387589	12	6564.499788781285	1631
68	12	Georgia	US	2021-01-07 05:30:38	33.0406	-83.6431	713840	11165	0	702675	13	6723.288692557508	552
69	14	Guam	US	2021-01-07 05:30:38	13.4443	144.7937	7378	123	7145	110	66	4492.507413428811	9
70	15	Hawaii	US	2021-01-07 05:30:38	21.0943	-157.4983	22917	299	11958	10353	15	1596.895764588889	83
71	16	Idaho	US	2021-01-07 05:30:38	44.2405	-114.4788	146106	1488	62215	82403	16	8175.751861292118	55
72	17	Illinois	US	2021-01-07 05:30:38	40.3495	-88.9861	999288	18735	0	980553	17	7885.906847958158	1369
73	18	Indiana	US	2021-01-07 05:30:38	39.8494	-86.2583	539229	9883	357442	173044	18	8009.677047047934	589



```
SELECT * FROM coviddays WHERE total_test_results > 0;
```

4. The row shere show a simple query that gets all the values with a total_test_results > 0.