COVID-19 Government Response Tracker

Group Members: Joseph Polk, Nathan Logan, Nikhilesh Venkatasubramanian

Introduction:

Motivation: The COVID-19 outbreak came as a surprise to all of us, shutting down businesses, the economy and countries all around the world. Many measures like the enforcement of social distancing and the restriction of international travel were enforced in an effort to contain the spread of the virus. In order to understand the effectiveness of these measures, one has to study a tremendous amount of data, drawing parallels and comparisons between different sets of data. This was the main motivation behind building the COVID-19 Government Response Tracker. Using this application, we can study how different countries responded to the outbreak of the virus, what policies they implemented and investments they made in order to contain the spread of the virus and whether ultimately these measures proved effective against the virus.

The application: In order to increase accessibility to users, the COVID-19 Government Response Tracker is a web based application. It studies the response of the governments of 177 countries over the course of 6 months (starting from 1st January to 27th June 2020). Apart from studying the number of confirmed cases and deaths in the country, it also looks at various policies that each country implemented (such as closures of schools and workplace, emergency investment in healthcare) and the overall stringency of the implementation of policies on every particular day. By studying these statistics, one can draw important connections between the enforcement of different policies and how effective they were in the containment of the virus in that particular country.

Organization of Report:
2.1 : Logan
2.2: Nikhilesh
2.3, 2.4 : Joseph
2.5 : Logan
2.6 : Nikhilesh

Conclusion: Joseph
References: Nikhilesh

Implementation:

2.1 System Architecture:

For our dataset, we used the MySQL DBMS. Our interface was primarily constructed using HTML and CSS, using JQuery functions to fill in for the aspects HTML did not natively support like multi-valued sliders or easy to edit tooltips that activate on hover. For user interaction, we implemented functions with Javascript and JQuery that could parse JS variables to PHP scripts using JQuery's AJAX function. Then, for the server-side functionality, we wrote PHP scripts that could generate and execute desired SQL queries.

We used Java briefly to do some preprocessing on our dataset, culling redundant rows and separating columns into their own tables. Our final schema resulted in nine tables, each with no more than four columns, two of which are generally used for our composite key (more on the model in 2.2-2.4).

Our data is a combination of integers, BIGINTs (the dates), and VARCHARs (country codes/country names). As stated, our primary keys are generally a composite of date and country code, the exceptions being the tables revolving around government response (Closures, Policies, Investments) which use the closure, policy, or investment name respectively as well to uniquely identify a tuple as unique. The general design vision being that tables should be easily joined, sorted, and filtered without excess intermediaries.

2.2 Dataset:

We worked with a dataset that was created by the University of Oxford in collaboration with the Blavatnik School of Government. The dataset consists of data that lists the responses of the governments of 177 countries to the novel COVID-19 virus everyday between 01/01/2020 till the current date. While the original dataset is updated on a daily basis, we decided to keep things simple by having a static dataset with the date ranging from 01/01/2020 to 06/27/2020. It has information on several different common policy responses that governments have taken to respond to the pandemic on 17 indicators such as school closures and travel restrictions. Of the 17 indicators, we chose to include only the following from the following categories:

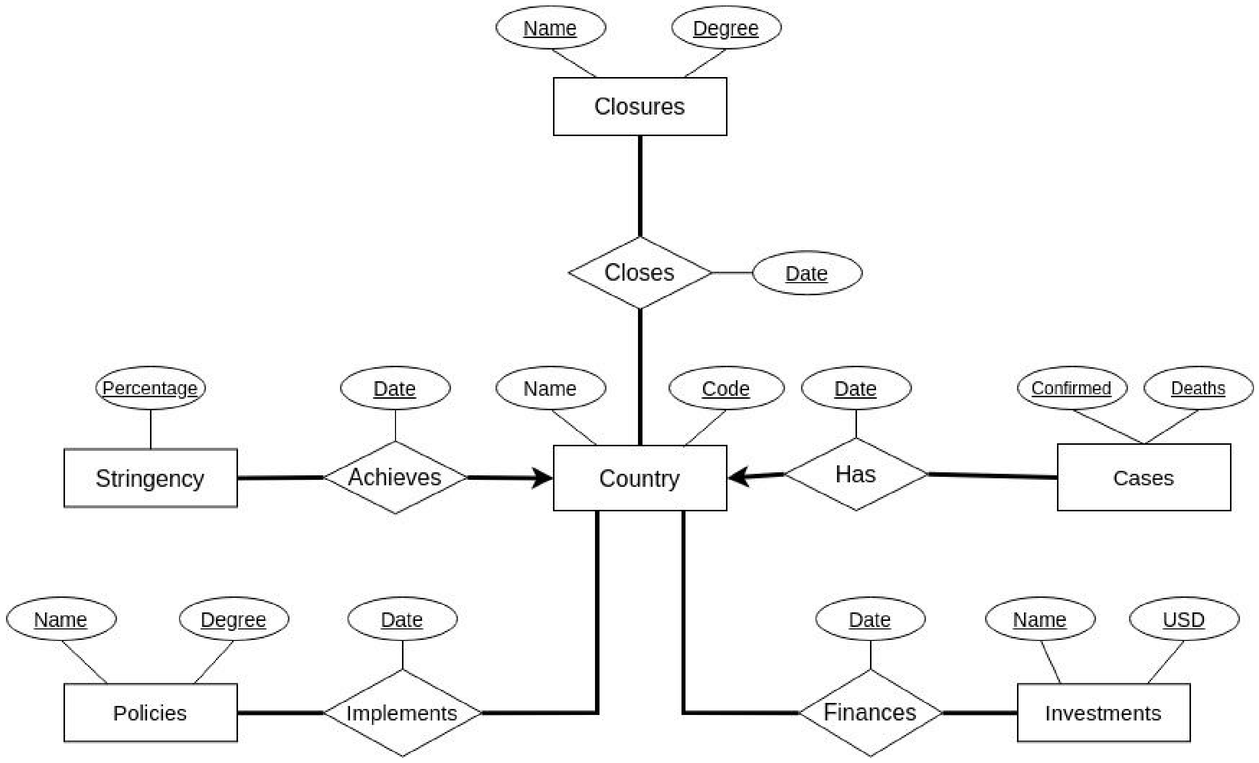
- Closures:
    - School
    - Work
    - Gatherings
    - Public transport
    - Stay At Home orders
    - International travel

- Investments:
    - Emergency health care investment
    - Vaccine investment
- Policies:
    - Income Support
    - Testing

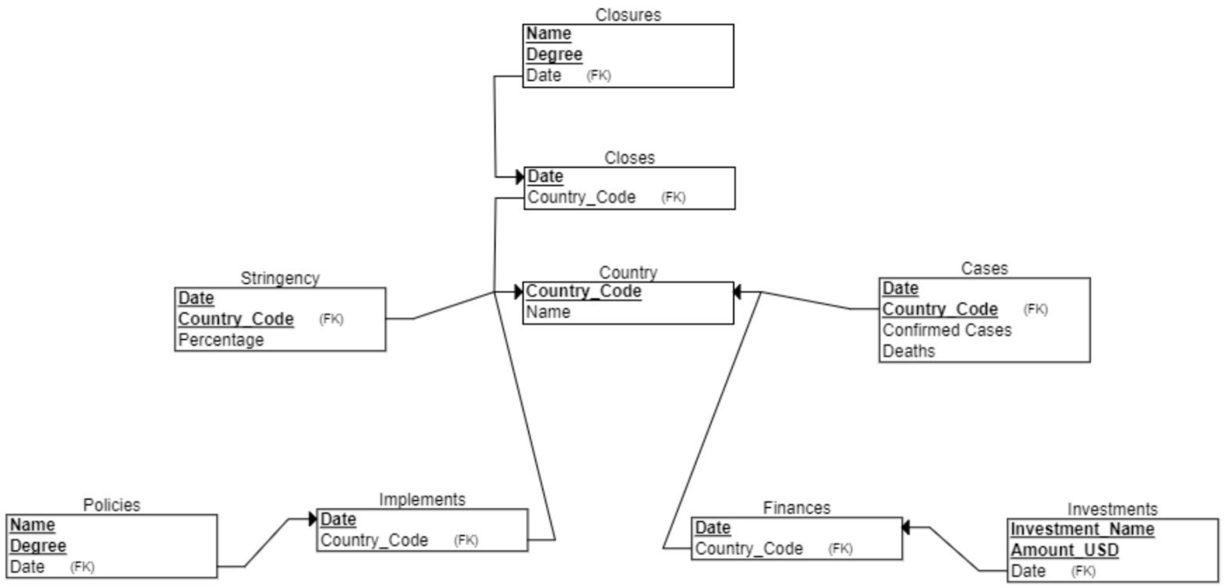
Every indicator has an accompanying degree (excluding investments, which have a corresponding amount in USD) that explains the extent to which the government of that particular country implemented a policy or a closure on a given date.
Degree:
0.0 - No measures
1.0 - Recommended
2.0- Required on some levels
3.0 - Required on all levels

Ensuring that the data fit the requirements of our design model meant performing quite some data preprocessing and cleaning. The original data was available in the form of a CSV file. The first step was to extract the required columns from the original data. The second step was to remove all those tuples which didn't meet the requirements (Blank or invalid data, date beyond limit). Once this was done, the last step was to put each relation from the relational model into separate CSV files that could then be loaded into their corresponding tables in the MySQL database.

2.3 ER Diagram:



2.4 Relational Model:



The different tables and the number of tuples in each are:

1) Country - 177 tuples
2) Cases - 32467 tuples
3) Stringency - 32467 tuples
4) Closes - 194801 tuples
5) Closures - 25 tuples
6) Implements - 64934 tuples
7) Policies - 7 tuples
8) Finances - 64934 tuples
9) Investments - 305 tuples

The reason for the Closes table being so large is because we combined all the different kinds of closures into a single table. While this may be an issue with memory, it was a better design option and allows to view all the different closures simultaneously.

2.5 Implementation:

Our web-app uses a total of six php scripts to handle the utilization of our dataset. Two handle dynamically updating data outside the input of the user: the list of countries and the global data. This way the fields refresh with the page, so adding new tuples is reflected in the country list and the global statistics. The other four PHP scripts run upon receiving user input through the sliders or country selectors.

Of these four scripts, two handle getting data based on user input, and the other two either insert (update on duplicate) or delete data based on the passed input. Javascript is used to handle any get requests, and a php file is directly called on post requests (insert/update and delete). HTML input types are then used for detecting user input as well as sending the input as a parameter.

The general flow of the page is: upon entering the main page, the global data is loaded (uses a table, `cases` for confirmed cases/deaths and another table, `stringency` for stringency percentage [they are unjoined]) and then the countries are generated (one table, `country` used). When the user checks a country in the list, the country's code is sent to a Javascript function which uses the JQuery AJAX function to pass the country's code as well as the current slider values to a PHP file.

This file contains the primary query of our application. It performs a natural join on six tables: `country`, `implements` (twice), `cases`, `closes`, and `stringency` and then finds all tuples with country name like the country code passed, or all countries in the case of no input. A similar sequence occurs if a slider is moved and its value changed. When released, a JQuery function updates the values of the slider, storing them in an array. Then, a query is performed using all slider values and country code (an empty string if none selected) as variables. The same query as specified above is performed and the values updated to the main <div> element.

When we first finished our query, the result was very slow, we were required to cap the output at 250 tuples or risk freezing the webpage. The query was functioning as planned, so changing the query didn't seem like an ideal course of action. Debugging the query using the EXPLAIN SQL function, it was evident that both closes and implements (194,801 and 64,934 tuples respectively) were scanning their entire files for each query. This led us to creating an index in our database for the country code and date on both implements and closes.

The result is that it now takes less time to load in 10,000 tuples than it did to load in 250 without indexing. It is worth noting however, that our query is set to limit results to the first 2,000 or so tuples since stricter filtering is more ideal than manually parsing through hundreds of rows of information.

The remaining functionality of the web-app falls in the insert/update and delete functions. Using the insert field, the form requires a country name, a country code, a date, and then allows parameters for each of the primary tables of the database. Across the twelve text-inputs, 6 different queries are performed on the following: ***`country`{country name, country code};*** **<u>`cases`{country code, date, confirmed cases, deaths};</u>** ***`closes`{country code, date, closure name, closure degree};*** **<u>`implements`{country code, date, policy name, policy degree};</u>** ***`finances`{country code, date, investment name, investment amount};*** **<u>`stringency`{country code, date, stringency}</u>.** So even given invalid info on one, as long as one of the above sets is valid, that tuple will be added. If a tuple already exists for a valid input, the non-keys will be updated to the newest value.

The delete function works similarly, but instead only requires all of the primary keys of a table be present For those where country code and date are the composite key, a binary value is used to select whether the tuple with that date, code pair should be kept or deleted [0 = keep, 1 = delete; default = 0]. Just as above, queries are separated, so even if one input is not valid, the other tuples can still be deleted. All changes made are reflected either on a new query, or in the case of global data and the country list, on page reload.

2.6 Evaluation:

Query Evaluation:

Our web based application is hosted on an FTP server which supports MySQL. To test the application we would have to test some queries that would demonstrate the basic functionality of the database. Since the application uses the very same data (which is obtained and processed using PHP and javascript), this would ensure the functionality of the application. Some of the basic queries which demonstrate this are:

1. **Check the total number of countries in the database:**
   There are a total of 177 countries in the dataset. Running a query counting the number of countries should give us the same result.

   Query: select count(*) from country;
   Result: 177 (as expected)

2. **Print out unique closures**
   We selected the following aspects of the closures to implement in our dataset:
   - School
   - Work
   - Gatherings
   - Public transport
   - Stay At Home orders
   - International travel

   A query that returns the the types of closures should give us the same result

   Query: select distinct cloName from closures;

   Result:
```
+-----------+
| cloName   |    (As expected)
+-----------+
| gathering |
| inttravel |
| school    |
| stayhome  |
| transport |
| workplace |
+-----------+
```

3. **Print our unique policies and investments:**
We do the same thing for the investments and policies to see if it fits the dataset that we used to implement the application

The types of policies are:
- Income support
- Testing

An additional test we could do is look at the degrees and see if they are within the permissible values. The degree for each ranges from 0 to 3 indicating how stringently the policy was implemented
Query : select (*) from policies;
Result:

```
+---------------+-----------+
| polName       | polDegree |
+---------------+-----------+
| incomesupport |         0 |
| incomesupport |         1 |
| incomesupport |         2 |
| testing       |     0     |
| testing       |     1     |
| testing       |     2     |
| testing       |     3     |
+---------------+-----------+
```

The types of investments are:
- Healthcare
- Vaccines

Query: select distinct invName from investments;
Result:

```
+------------+
| invName    |
+------------+
| healthcare |
| vaccines   |
+------------+
```

4. **Print the total deaths/confirmed cases for each country**

   We could also count the total number of deaths for all countries and see if it's a plausible number.

   Query:
   ```
   select sum(f.deaths)
   from
   (select couCode as code, max(deaths) as deaths from cases group by code) as f;
   ```

   Result:
   ```
   +---------------+
   | sum(f.deaths) |
   +---------------+
   |        511045 |
   +---------------+  which is a plausible number
   ```

5. **Print the date the first case occurs for each country.**

   Most countries didn't get their first coronavirus case until at least January and when the countries confirmed their first case it was publicized in the media, so an expected result can be found easily in an internet search. We can then perform the query below and compare the date to the expected date.

   Select min date where cases > 0

6. **Check that confirmed deaths is less than or equal to confirmed cases**

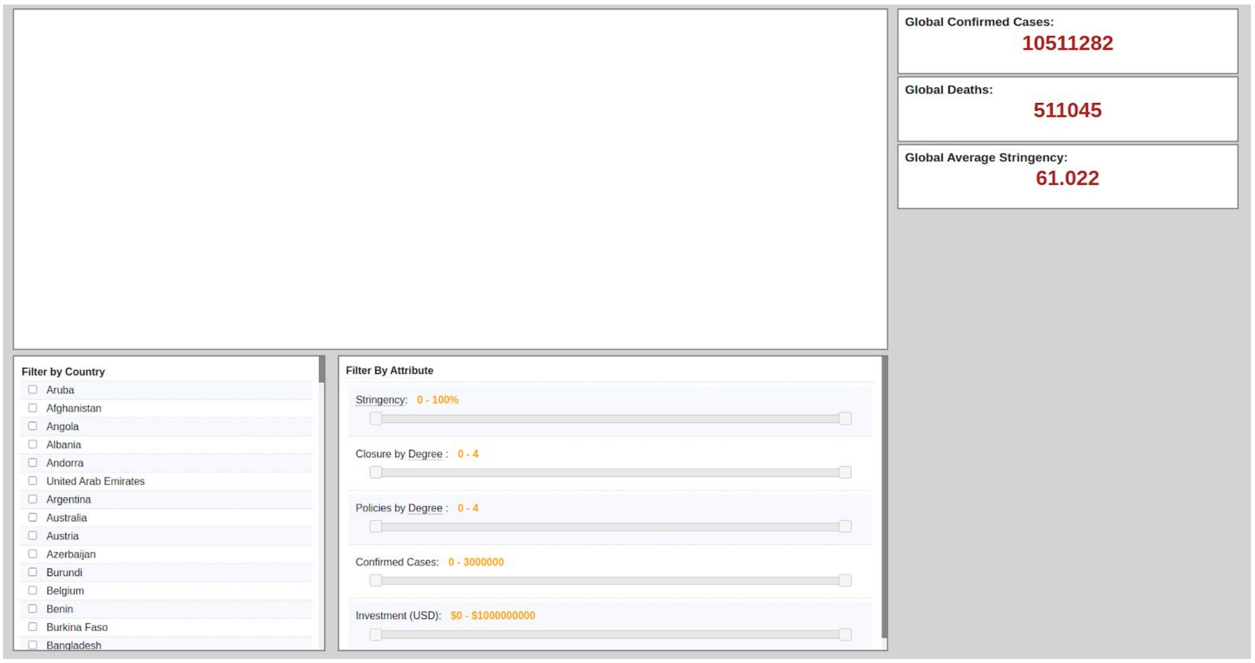
   Logically speaking, someone can't die of COVID-19 if they haven't contracted COVID-19, so the number of total deaths for a country is higher than the number of total confirmed cases, then something is wrong with the database.

   Query:
   ```
   "SELECT DISTINCT SUM(a.conCases), SUM(a.conDeaths)
   FROM country c, cases a
   WHERE c.couName = $var1 AND c.couCode = a.couCode"
   ```

GUI Evaluation:

The GUI is functional as long as it presents all the necessary information from the database accurately. So, the only means of testing the GUI is to put it to use and see if it does what it is supposed to.

1) Upon loading



Global Confirmed Cases:
10511282

Global Deaths:
511045

Global Average Stringency:
61.022

Filter by Country

- ☐ Aruba
- ☐ Afghanistan
- ☐ Angola
- ☐ Albania
- ☐ Andorra
- ☐ United Arab Emirates
- ☐ Argentina
- ☐ Australia
- ☐ Austria
- ☐ Azerbaijan
- ☐ Burundi
- ☐ Belgium
- ☐ Benin
- ☐ Burkina Faso
- ☐ Bangladesh

Filter By Attribute

Stringency: 0 - 100%

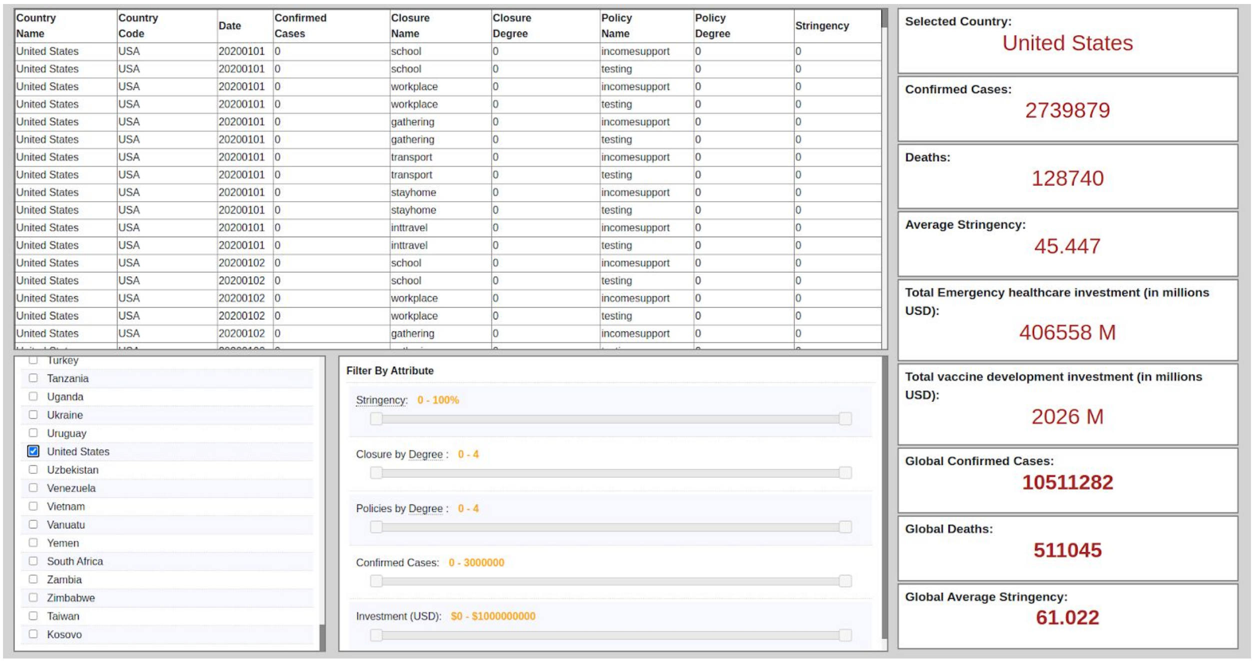
Closure by Degree : 0 - 4

Policies by Degree : 0 - 4

Confirmed Cases: 0 - 3000000

Investment (USD): $0 - $1000000000

2) Selecting a particular country



Country Name	Country Code	Date	Confirmed Cases	Closure Name	Closure Degree	Policy Name	Policy Degree	Stringency
United States	USA	20200101	0	school	0	incomesupport	0	0
United States	USA	20200101	0	school	0	testing	0	0
United States	USA	20200101	0	workplace	0	incomesupport	0	0
United States	USA	20200101	0	workplace	0	testing	0	0
United States	USA	20200101	0	gathering	0	incomesupport	0	0
United States	USA	20200101	0	gathering	0	testing	0	0
United States	USA	20200101	0	transport	0	incomesupport	0	0
United States	USA	20200101	0	transport	0	testing	0	0
United States	USA	20200101	0	stayhome	0	incomesupport	0	0
United States	USA	20200101	0	stayhome	0	testing	0	0
United States	USA	20200101	0	inttravel	0	incomesupport	0	0
United States	USA	20200101	0	inttravel	0	testing	0	0
United States	USA	20200102	0	school	0	incomesupport	0	0
United States	USA	20200102	0	school	0	testing	0	0
United States	USA	20200102	0	workplace	0	incomesupport	0	0
United States	USA	20200102	0	workplace	0	testing	0	0
United States	USA	20200102	0	gathering	0	incomesupport	0	0

- ☐ Turkey
- ☐ Tanzania
- ☐ Uganda
- ☐ Ukraine
- ☐ Uruguay
- ☑ United States
- ☐ Uzbekistan
- ☐ Venezuela
- ☐ Vietnam
- ☐ Vanuatu
- ☐ Yemen
- ☐ South Africa
- ☐ Zambia
- ☐ Zimbabwe
- ☐ Taiwan
- ☐ Kosovo

Filter By Attribute

Stringency: 0 - 100%

Closure by Degree : 0 - 4

Policies by Degree : 0 - 4

Confirmed Cases: 0 - 3000000

Investment (USD): $0 - $1000000000

Selected Country: United States

Confirmed Cases: 2739879

Deaths: 128740

Average Stringency: 45.447

Total Emergency healthcare investment (in millions USD): 406558 M

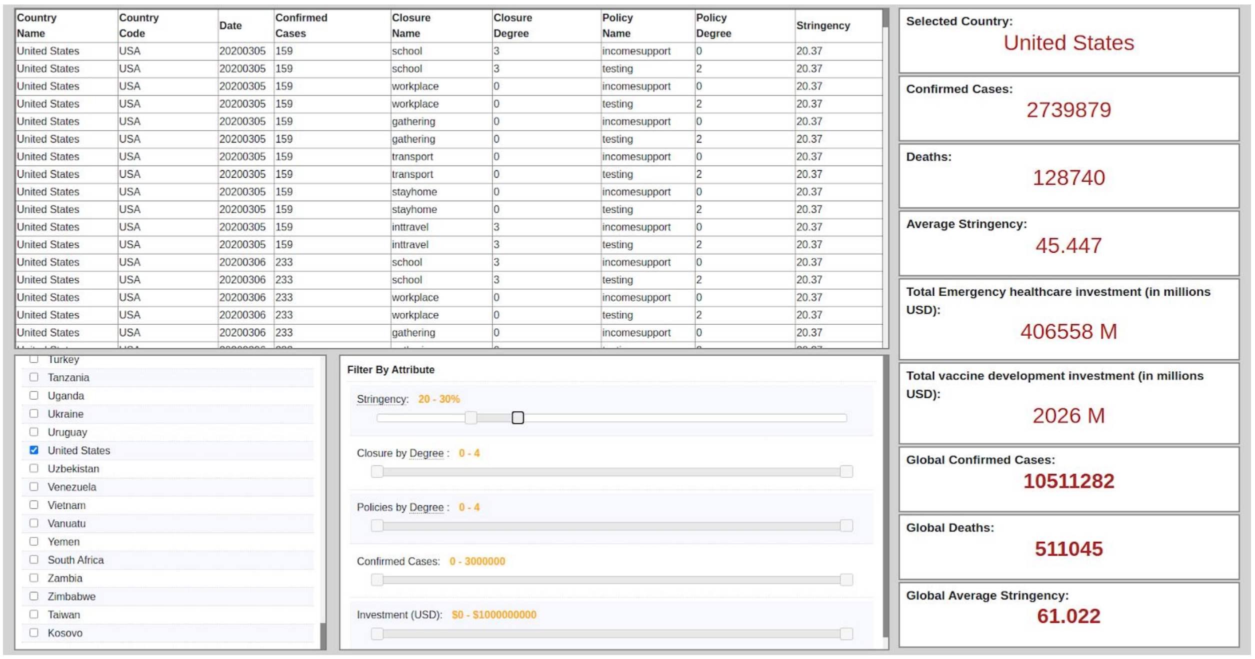
Total vaccine development investment (in millions USD): 2026 M

Global Confirmed Cases: **10511282**

Global Deaths: **511045**

Global Average Stringency: **61.022**

3) Applying a filter on stringency



Country Name	Country Code	Date	Confirmed Cases	Closure Name	Closure Degree	Policy Name	Policy Degree	Stringency
United States	USA	20200305	159	school	3	incomesupport	0	20.37
United States	USA	20200305	159	school	3	testing	2	20.37
United States	USA	20200305	159	workplace	0	incomesupport	0	20.37
United States	USA	20200305	159	workplace	0	testing	2	20.37
United States	USA	20200305	159	gathering	0	incomesupport	0	20.37
United States	USA	20200305	159	gathering	0	testing	2	20.37
United States	USA	20200305	159	transport	0	incomesupport	0	20.37
United States	USA	20200305	159	transport	0	testing	2	20.37
United States	USA	20200305	159	stayhome	0	incomesupport	0	20.37
United States	USA	20200305	159	stayhome	0	testing	2	20.37
United States	USA	20200305	159	inttravel	3	incomesupport	0	20.37
United States	USA	20200305	159	inttravel	3	testing	2	20.37
United States	USA	20200306	233	school	3	incomesupport	0	20.37
United States	USA	20200306	233	school	3	testing	2	20.37
United States	USA	20200306	233	workplace	0	incomesupport	0	20.37
United States	USA	20200306	233	workplace	0	testing	2	20.37
United States	USA	20200306	233	gathering	0	incomesupport	0	20.37

- ☐ Turkey
- ☐ Tanzania
- ☐ Uganda
- ☐ Ukraine
- ☐ Uruguay
- ☑ United States
- ☐ Uzbekistan
- ☐ Venezuela
- ☐ Vietnam
- ☐ Vanuatu
- ☐ Yemen
- ☐ South Africa
- ☐ Zambia
- ☐ Zimbabwe
- ☐ Taiwan
- ☐ Kosovo

Filter By Attribute

Stringency: 20 - 30%

Closure by Degree : 0 - 4

Policies by Degree : 0 - 4

Confirmed Cases: 0 - 3000000

Investment (USD): $0 - $1000000000

Selected Country: United States

Confirmed Cases: 2739879

Deaths: 128740

Average Stringency: 45.447

Total Emergency healthcare investment (in millions USD): 406558 M

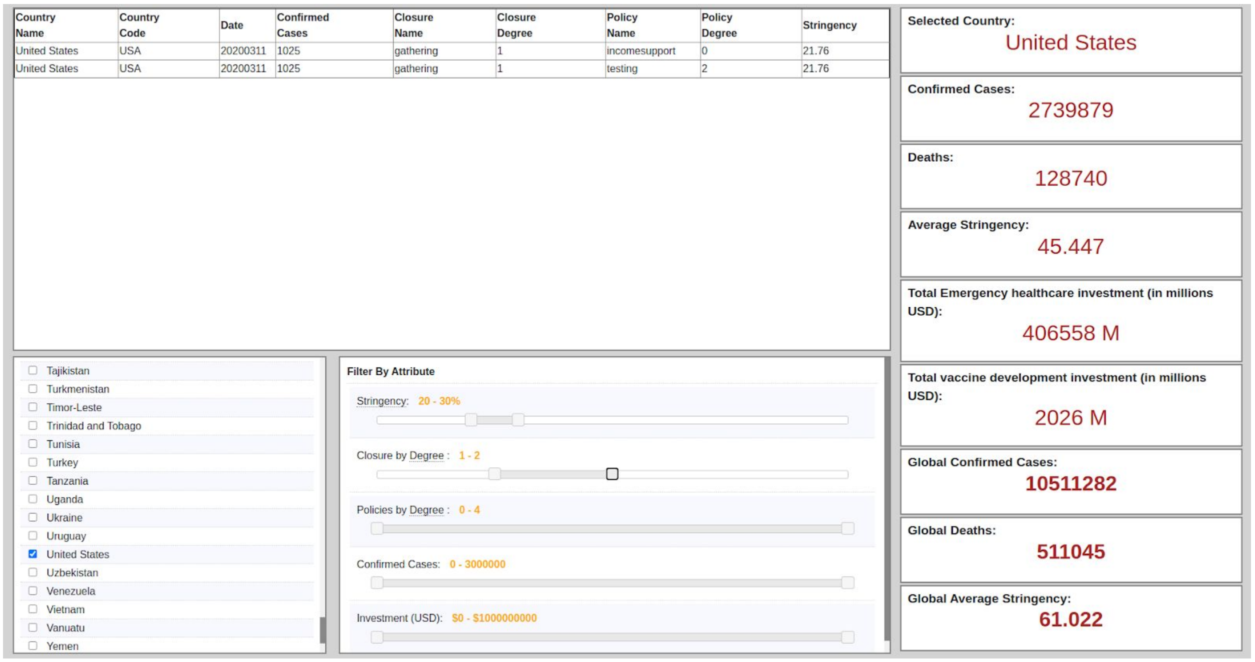
Total vaccine development investment (in millions USD): 2026 M

Global Confirmed Cases: **10511282**

Global Deaths: **511045**

Global Average Stringency: **61.022**

4) Applying a filter on closure degree



Country Name	Country Code	Date	Confirmed Cases	Closure Name	Closure Degree	Policy Name	Policy Degree	Stringency
United States	USA	20200311	1025	gathering	1	incomesupport	0	21.76
United States	USA	20200311	1025	gathering	1	testing	2	21.76

- ☐ Tajikistan
- ☐ Turkmenistan
- ☐ Timor-Leste
- ☐ Trinidad and Tobago
- ☐ Tunisia
- ☐ Turkey
- ☐ Tanzania
- ☐ Uganda
- ☐ Ukraine
- ☐ Uruguay
- ☑ United States
- ☐ Uzbekistan
- ☐ Venezuela
- ☐ Vietnam
- ☐ Vanuatu
- ☐ Yemen

Filter By Attribute

Stringency: 20 - 30%

Closure by Degree : 1 - 2

Policies by Degree : 0 - 4

Confirmed Cases: 0 - 3000000

Investment (USD): $0 - $1000000000

Selected Country: United States

Confirmed Cases: 2739879

Deaths: 128740

Average Stringency: 45.447

Total Emergency healthcare investment (in millions USD): 406558 M

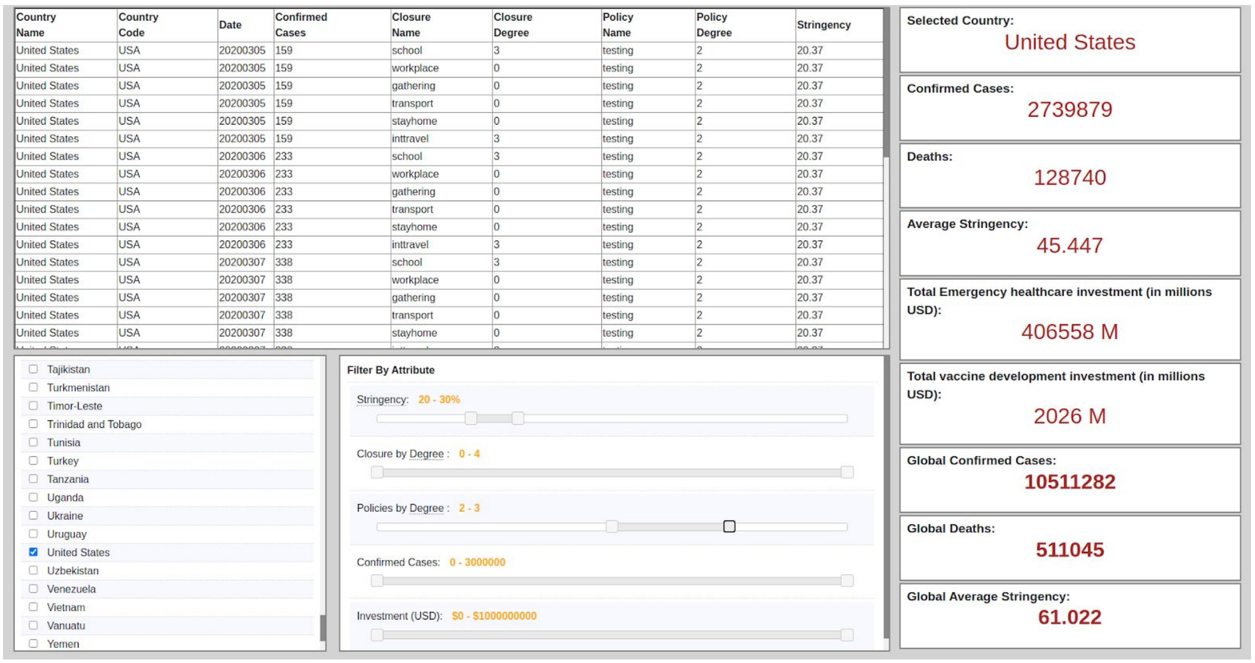
Total vaccine development investment (in millions USD): 2026 M

Global Confirmed Cases: 10511282

Global Deaths: 511045

Global Average Stringency: 61.022

5) Applying a filter on policy degree



Country Name	Country Code	Date	Confirmed Cases	Closure Name	Closure Degree	Policy Name	Policy Degree	Stringency
United States	USA	20200305	159	school	3	testing	2	20.37
United States	USA	20200305	159	workplace	0	testing	2	20.37
United States	USA	20200305	159	gathering	0	testing	2	20.37
United States	USA	20200305	159	transport	0	testing	2	20.37
United States	USA	20200305	159	stayhome	0	testing	2	20.37
United States	USA	20200305	159	inttravel	3	testing	2	20.37
United States	USA	20200306	233	school	3	testing	2	20.37
United States	USA	20200306	233	workplace	0	testing	2	20.37
United States	USA	20200306	233	gathering	0	testing	2	20.37
United States	USA	20200306	233	transport	0	testing	2	20.37
United States	USA	20200306	233	stayhome	0	testing	2	20.37
United States	USA	20200306	233	inttravel	3	testing	2	20.37
United States	USA	20200307	338	school	3	testing	2	20.37
United States	USA	20200307	338	workplace	0	testing	2	20.37
United States	USA	20200307	338	gathering	0	testing	2	20.37
United States	USA	20200307	338	transport	0	testing	2	20.37
United States	USA	20200307	338	stayhome	0	testing	2	20.37

- ☐ Tajikistan
- ☐ Turkmenistan
- ☐ Timor-Leste
- ☐ Trinidad and Tobago
- ☐ Tunisia
- ☐ Turkey
- ☐ Tanzania
- ☐ Uganda
- ☐ Ukraine
- ☐ Uruguay
- ☑ United States
- ☐ Uzbekistan
- ☐ Venezuela
- ☐ Vietnam
- ☐ Vanuatu
- ☐ Yemen

Filter By Attribute

Stringency: 20 - 30%

Closure by Degree : 0 - 4

Policies by Degree : 2 - 3

Confirmed Cases: 0 - 3000000

Investment (USD): $0 - $1000000000

Selected Country: United States

Confirmed Cases: 2739879

Deaths: 128740

Average Stringency: 45.447

Total Emergency healthcare investment (in millions USD): 406558 M

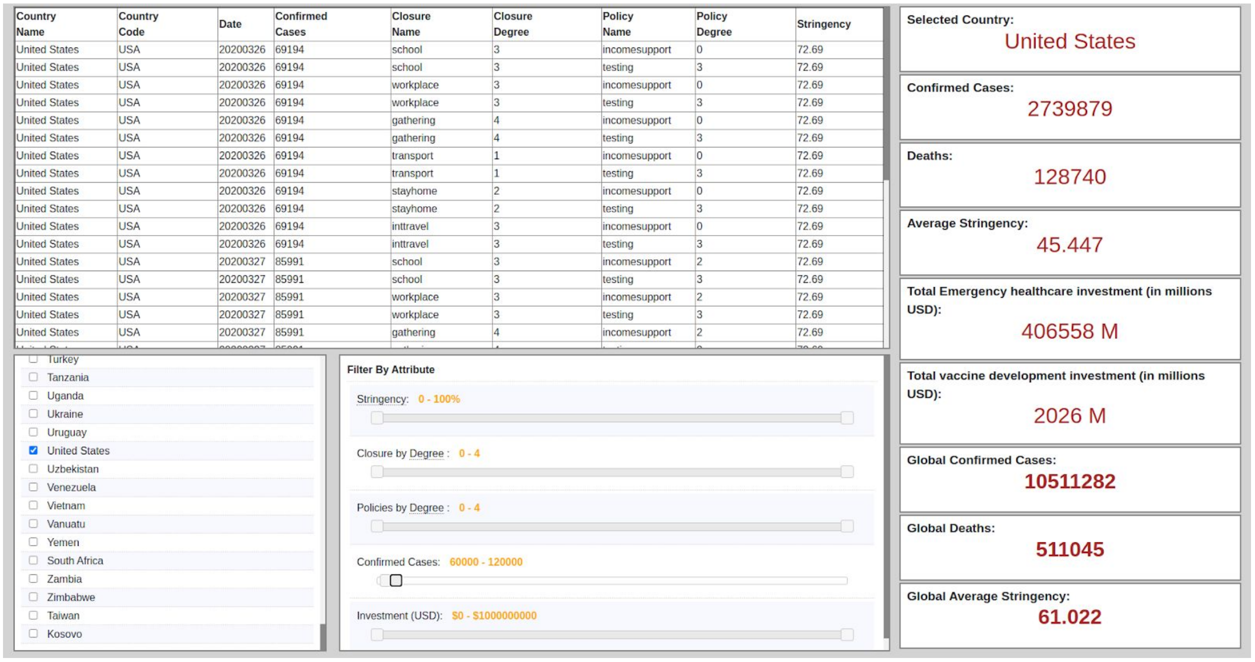
Total vaccine development investment (in millions USD): 2026 M

Global Confirmed Cases: 10511282

Global Deaths: 511045

Global Average Stringency: 61.022

6) Applying a filter on the number of confirmed cases



Country Name	Country Code	Date	Confirmed Cases	Closure Name	Closure Degree	Policy Name	Policy Degree	Stringency
United States	USA	20200326	69194	school	3	incomesupport	0	72.69
United States	USA	20200326	69194	school	3	testing	3	72.69
United States	USA	20200326	69194	workplace	3	incomesupport	0	72.69
United States	USA	20200326	69194	workplace	3	testing	3	72.69
United States	USA	20200326	69194	gathering	4	incomesupport	0	72.69
United States	USA	20200326	69194	gathering	4	testing	3	72.69
United States	USA	20200326	69194	transport	1	incomesupport	0	72.69
United States	USA	20200326	69194	transport	1	testing	3	72.69
United States	USA	20200326	69194	stayhome	2	incomesupport	0	72.69
United States	USA	20200326	69194	stayhome	2	testing	3	72.69
United States	USA	20200326	69194	inttravel	3	incomesupport	0	72.69
United States	USA	20200326	69194	inttravel	3	testing	3	72.69
United States	USA	20200327	85991	school	3	incomesupport	2	72.69
United States	USA	20200327	85991	school	3	testing	3	72.69
United States	USA	20200327	85991	workplace	3	incomesupport	2	72.69
United States	USA	20200327	85991	workplace	3	testing	3	72.69
United States	USA	20200327	85991	gathering	4	incomesupport	2	72.69

- [ ] Turkey
- [ ] Tanzania
- [ ] Uganda
- [ ] Ukraine
- [ ] Uruguay
- [x] United States
- [ ] Uzbekistan
- [ ] Venezuela
- [ ] Vietnam
- [ ] Vanuatu
- [ ] Yemen
- [ ] South Africa
- [ ] Zambia
- [ ] Zimbabwe
- [ ] Taiwan
- [ ] Kosovo

Filter By Attribute

Stringency: 0 - 100%

Closure by Degree : 0 - 4

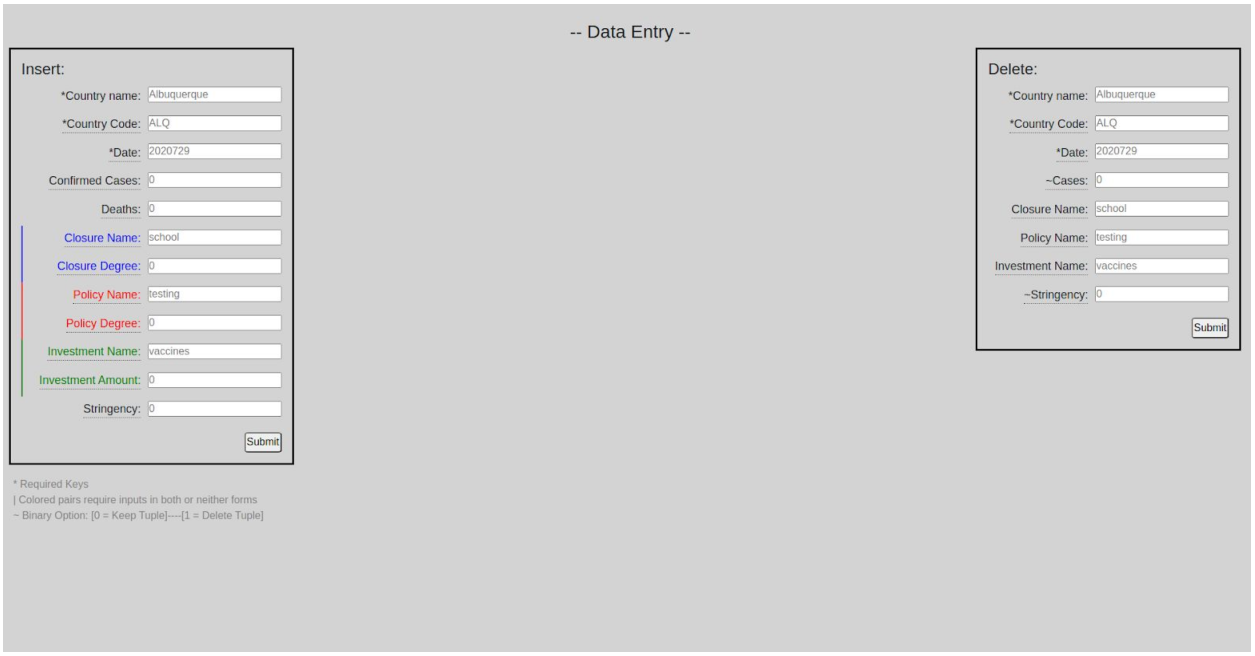
Policies by Degree : 0 - 4

Confirmed Cases: 60000 - 120000

Investment (USD): $0 - $1000000000

Selected Country:	United States
Confirmed Cases:	2739879
Deaths:	128740
Average Stringency:	45.447
Total Emergency healthcare investment (in millions USD):	406558 M
Total vaccine development investment (in millions USD):	2026 M
Global Confirmed Cases:	10511282
Global Deaths:	511045
Global Average Stringency:	61.022

7) Data Entry



-- Data Entry --

Insert:

*Country name:	Albuquerque
*Country Code:	ALQ
*Date:	2020729
Confirmed Cases:	0
Deaths:	0
Closure Name:	school
Closure Degree:	0
Policy Name:	testing
Policy Degree:	0
Investment Name:	vaccines
Investment Amount:	0
Stringency:	0

Submit

Delete:

*Country name:	Albuquerque
*Country Code:	ALQ
*Date:	2020729
~Cases:	0
Closure Name:	school
Policy Name:	testing
Investment Name:	vaccines
~Stringency:	0

Submit

* Required Keys
| Colored pairs require inputs in both or neither forms
~ Binary Option: [0 = Keep Tuple]----[1 = Delete Tuple]

Conclusion:

This turned out to be a very insightful project for many reasons. The first thing we learned, although rather trivial, is that hosting a web server for the application isn't free. We initially used Google Cloud as our server, but we started getting billed for it (only $18 which was covered by the student credit). We decided to use our team member Nathan's parent's server so the cost wasn't an issue. But in future pursuits it is helpful to know the costs involved.

Another thing we learned is how important it is to have a good ER diagram and then a relational model set before starting. Before this class, I would have haphazardly uploaded the CSV files to the MySQL server without much thought on whether I'm putting the data into the correct tables. This would have taken me hours of troubleshooting down the road when trying to query from tables that aren't based on the correct relations. This becomes especially relevant when level 3 queries are required.

An issue we came across that was dealt with in class was query optimization during insert and delete processes. When we first implemented insert and delete it would take an unacceptable amount of time to get there the 190,000 tuples in the closes table. Taking the idea of hashing, we applied indexes to all the tuples so that we could jump to the location of inserting and deleting instead of sorting through. This improved the runtime of these operations significantly.

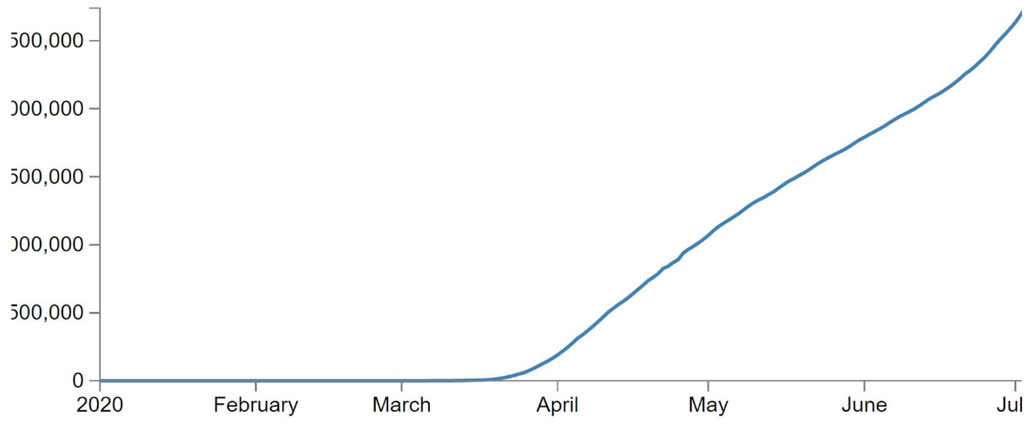
An issue we came across that was dealt with in class was how to handle null values in queries. A lot of the 195 countries in the world didn't have perfect data from the start of COVID -19 coming into existence, so there was a bit of missing data. We couldn't place the missing values as zero for cases and confirmed deaths because that would result in inaccurate averages, so we placed them as null so that they wouldn't be counted in the queries.

Additional Features that could be included:

- Graphic Visualization of Data
- Machine learning techniques to predict the outcome (confirmed cases/deaths) based on training set

Graphic Visualization of Data:

The purpose of our data basing being to help the users see correlations between case rates and policies implemented which is hard to do just looking at raw data. So we planned to use PHP to obtain the number of cases by date for whichever country is selected and then use d3.js to plot that data. We figured out how to do this at the last minute, but we didn't have time to include it in the final project. Below is an example of how used the aforementioned method to plot the confirmed cases vs date for the United States:



An example of implementing Machine Learning: Say we want to predict the number of deaths based on the degree of the different kinds of closures. The training data for such a problem would be all the known closure degrees for the different types of closures along with the number of deaths for each country. Our feature vector would be the vector of the all closure degrees for a given country on a given date and the label would be the number of deaths on that particular day. We could combine all the feature vectors into a feature matrix, let's call this **A** and the corresponding label vector would be **d**. Now, our machine learning problem is to find a set of weights **w** such that **Aw = d**. There are many means of solving this problem (Least-Squares, Ridge-Regression, etc). Now let's say we have a new feature vector **x** for a new city on a given date, all one has to do is to multiply **x** with our vector of weights **w** to obtain the deaths for the new feature. This is just an instance of how we could apply machine learning to our data. We could also dive deeper into this by creating a neural network that calculates the deaths or confirmed cases based on multiple parameters such as investments and policies.

Our Take:

Nikhilesh : I mainly dealt with the design and database side of things in this project. Once we decided on the dataset, we got started on getting the ER diagram and the relational model ready. The relational model needed quite some tweaking before we were satisfied with the result. Once we had the ER diagram ready, I got to cleaning the data, picking the parameters we wanted to include and separating each table into separate CSVs. At first, we wanted to host the database on Google Cloud using CloudSQL. I figured out how to get started with Google Cloud and even had the database loaded in before we realized that there were two problems, one, we began incurring costs to host the database (luckily, these were covered by the student credit) and two, we weren't sure how we could link the database to our backend. So instead, we decided to use an FTP server to host the database and this was relatively easy to connect to the backend and ultimately to the frontend. Although we didn't end up using CloudSQL, I did learn how to work my way around

CloudSQL and Google Cloud. Loading the data took considerable time, especially because some of the tables were rather large. But we are able to do this without any major difficulties. Once the database was loaded in, we began to work on the backend and the frontend. Given my lack of experience in web development, I wasn't sure how we would go about doing this. Luckily, Nathan seemed to have some experience and guided us through the process. While he still ended up doing a relatively large portion of the development, I definitely picked up on some important concepts in PHP, javascript, HTML, CSS and was able to contribute as much as I could to building the application. While Nathan was busy with the development, Joe and I worked on the BPlus Index project. It took quite some time to figure out, but we managed to get it working without a major hitch. Overall, I think the project was very fruitful. Although we couldn't include all the features we would have liked in the application because of shortage of time, I learnt many new concepts in database systems, Google Cloud and web development.

Nathan : I handled a lot of the web development for this project. I have previous experience with HTML, CSS, and Javascript, so I was somewhat comfortable working on the front-end, but I am in no way proficient in web development. Though I am glad to have the experience that I do, as it made formatting and implementing design ideas with HTML and CSS a lot simpler. The greatest difficulty was working on the back-end of our website. I had used PHP before, but not extensively, and definitely not with a database, so it was a bit overwhelming. There were countless moments where I would run into a problem, take, at times, hours to study up on any possible solution, only to run into another unique issue immediately after. The biggest moment of this was when we were trying to figure out how to use Javascript variables in our PHP files. The first issue: how do I take the values generated by my sliders? It turned out to be somewhat simple thanks to the powerful functions allowed through JQuery, but then I couldn't figure out how to call a PHP script with Javascript, this part took a while to research as there is a lot of info on this, but eventually, JQuery saved me again and I learned how to use AJAX calls to make http requests, allowing me to call the PHP script and send in my slider values as well. Of course more hang ups followed: I had no clue how to make SQL queries in PHP, after that, how do I add country code to the filter search, then after that, how do I stop my queries from running as the slider is moved because it was making an absurd amount of calls, especially on large values like confirmed cases, then ultimately, how do I speed up our query? Tackling this problem was probably one of the greatest boons for our website, we could previously only load 250 or so tuples before our web server crashed, but after some research (and guidance from lectures) I tried creating an index on the tables causing problems. This instantly allowed us to load thousands of tuples in seconds. All this is to say that even with a seemingly endless stream of hurdles we were able to figure out a solution. We couldn't quite implement every feature we wanted to, but it takes failing as many times as we did making this web-app to force creative problem solving and expedite learning.

Joe: With the B+ tree I was in charge of implementing the delete function, and along with coming up with the initial user interface of the application. Working on the delete function helped me develop some very useful habits in java development. With previous projects I was able to just start coding and chip away at it. With the recursive nature required to merge and redistribute up the tree, I was hitting a ton of road blocks and ended up just staring at the screen for hours. From there I realized I would have to take a step back and look at the pseudo code in the cow textbook before I do anything. This helped instill the habit of getting my pseudocode down first and a plan in place before I start. This will save a lot of time and errors in future projects.

I really enjoyed the fact that we had to build our own application from scratch and make all of the implementation decisions. It was almost overwhelming at first because I had never heard of PHP before this class. This took a bit of struggling and not being sure what was going on in the beginning, but it taught me a ton and made me a better developer. I did feel bad as I wasn't able to contribute as much in the website development as my other team members because of the time it took for me to get over the learning curve.

I was able to get the data visualization working which should be very useful in future projects. Once the data is encoded into a json object using PHP, it can be visualized using the d3.js library. D3 is a very powerful library for data visualization, so I plan to continue working with it and learning more about it for future projects.

References:

Dataset: *Hale, Thomas, Sam Webster, Anna Petherick, Toby Phillips, and Beatriz Kira (2020). Oxford COVID-19 Government Response Tracker, Blavatnik School of Government. Data use policy: Creative Commons Attribution CC BY standard.*