BC2402: Designing and Developing Databases

Project Title: Tackling COVID-19 Infodemic

Submission Date:

15 November 2020

Seminar Group:

Seminar Group 1, Group 7

Names of group members:

Name:	**Matriculation Number:**
Chan Wen Xin	U1910993F
Ernest Ang Cheng Han	U1921210H
Ernest Tan Yan Heng	U1920436K
Goh Tse Yinn, Sheryl	U1922180C
Ng Xue Bing Linda	U1910143G

Table of Contents

1 Introduction

The COVID-19 pandemic, also known as the Corona-virus, is an ongoing global pandemic that was first identified in December 2019 in Wuhan, China. The World Health Organisation (WHO) declared the outbreak as a pandemic in March 2020. As of November 2020, there have been more than 51 million confirmed cases, with more than 1.2 million deaths reported.

Coupled with the spike in cases, we have observed a significant rise in the amount of information related to COVID-19. Sieving through the plethora of information is an essential part in responding to the pandemic. The key focus should thus be on how to efficiently obtain the correct and essential information, in order to gain meaningful insights from the analysis of relevant information. These insights gained will then be helpful for countries to implement solutions to reduce the effect and intensity of the virus on their population effectively.

In this project, we utilised three sources of data which gave us daily information on how COVID-19 has affected the various countries in the World and measures taken by countries to tackle the ongoing pandemic.

2 Data Models

In this project, we designed and implemented both a relational database and non-relational database. The relational database is implemented using MySQL while the non-relational database uses NoSQL together with MongoDB.

Relational Database Management System (RDBMS) is the basis for MySQL and data stored in RDBMS are called tables. These tables are collections of related data entries, consisting of columns and rows.

Non-relational databases were specifically developed for top internet companies such as Google, Yahoo, Amazon, etc as existing relational databases were not able to cope with the increasing data processing requirements. In this project, we focused our efforts on NoSQL and non-relational databases.

After the design and implementation to capture the three datasets as mentioned above, the two databases were compared based on results and performance and recommendations were crafted for WHO to tackle the COVID-19 situation.

2.1 Database Creation

To begin, we first took a look at the 3 datasets that we were given, namely "owid_covid_data", "global_mobility_report" and "response_graphs_2020-08-13". We realised that the datasets had certain issues that we wanted to resolve before we started on the queries.

Firstly, we realised that "owid_covid_data" and "global_mobility_report" had additional columns that added no value towards the queries. Therefore, we decided that we wanted to only pick out the columns which were needed to complete the queries, which would both reduce the computation required for each query, as well as only display the results that were necessary.

Next, for "response_graphs-2020-08-13", the first row of the dataset actually contained the column headers, which was deleted so that we would not affect the results of any of our queries.

2.1.1 Data Exploration & Cleaning

For rows which contained "World" and "International" in the location column, we removed them from the table as they were just a summation of all the countries, which interferes with our queries and are not required for obtaining our results.

For "global_mobility_report", we found that the records without "sub_region_1", "sub_region_2" and "metro_area" were just a combination of the entire country's mobility numbers. Therefore, we removed records which had value in any of those fields, so that we would only be getting the results by country instead of by smaller subregions or metro areas.

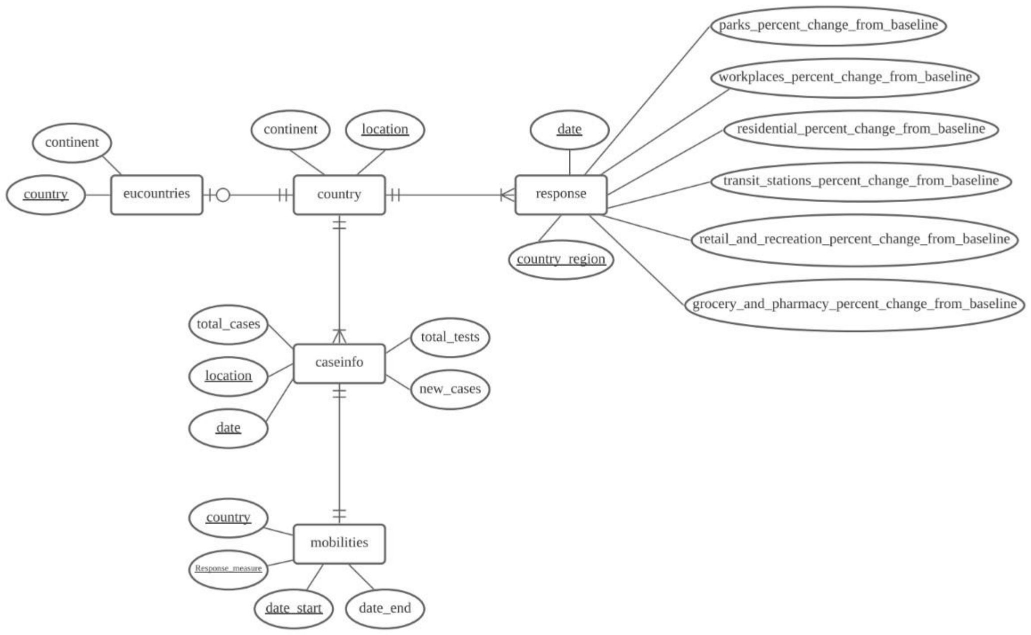
These changes were applied to both the relational and non-relational database implementations.

2.2 Relational Database

After we imported the data, we made the changes that we identified earlier, e.g. removing unnecessary columns, deleting unneeded rows. After this, we analysed the columns and realised that we needed to conduct data normalisation in order to achieve 3NF.

Data normalisation was performed on the three datasets to avoid the unnecessary duplication of data. We aimed to achieve Third Normal Form (3NF). 3NF schema is achieved when there are no non-key fields dependent on any other non-key fields, no dependent non-key fields spawned as a separate relation and non-key with functional dependency will become a new key.

In the original datasets, there were some many-to-many relations and duplicates of data. Data normalisation was performed by creating two new tables, namely “country” and “eucountries”. “country” would remove the duplicated data of continents in “caseinfo”. “eucountries” sieves out countries in the European Union (EU) in the “country” table for ease of implementation in MySQL as they were not explicitly stated to be EU countries in the “country” table.


continent
country
eucountries
continent
location
country
date
response
country_region
parks_percent_change_from_baseline
workplaces_percent_change_from_baseline
residential_percent_change_from_baseline
transit_stations_percent_change_from_baseline
retail_and_recreation_percent_change_from_baseline
grocery_and_pharmacy_percent_change_from_baseline
total_cases
location
date
caseinfo
total_tests
new_cases
country
Response_measure
mobilities
date_start
date_end

Entity-Relationship model between the three datasets in 3NF

2.2.1 Data Dictionary for Relational Database

Table	**Description**
caseinfo	Contains *total cases*, *new cases* and *total tests* for each *location* and *date*
country	Contains names of countries and their respective continents
eucountries	Contains list of all countries within the EU (27 countries)
mobilities	Contains data of the percentage changes for 6 different place categories in relation to their baseline for all the countries
response	Contains each country's *response measures*, and their respective *start* and *end dates*

2.3 Non-Relational Database

For our non-relational database, we created a new database and loaded in the JSON files containing the datasets into 3 separate collections. The table below shows the dataset and the collection name we specified for them:

Dataset	**Collection name**
global_mobility_report	mobility
owid_covid_data	covid
response_graphs	response

Unlike a relational database, there is no need to normalise the data in a non-relational database as the basis of normalisation is to make sure that data is stored in a relational manner. Therefore, after the JSON files were uploaded onto MongoDB, we could proceed straight with the data cleansing and writing of queries.

3 Data Model Analysis

Inconsistencies in relational and non-relational databases occur as they are fundamentally different in performance, reliability, flexibility, consistency, data storage and scalability.

3.1 Performance

In terms of writing speed, MySQL is slower than NoSQL but gradually becomes faster than NoSQL when the data collection becomes huge. This is true in this case as the datasets have 39.6K, 25.6K and 420 rows and documents for covid, mobility and response respectively in MySQL and NoSQL.

Based on 100 runs, the average time taken for each question for each database is as follows:

	Average time taken (s) to generate output	
Question	Relational database (MySQL)	Non-relational database (NoSQL)
1	0.016s	0.613s
2	0.032s	0.716s
3	0.031s	0.773s
4	0.062s	1.707s
5	0.078s	1.215s
6	0.140s	1.425s
7	<0.000...s	0.798s
8	0.031s	0.536s
9	0.078s	1.431s
10	0.219s	9.429s
11	1.095s	9.312s
12	30.562s	49.378s
13	<0.000..s.	0.587s

As seen from the timings as shown in the table, the average time taken using MySQL is significantly faster than NoSQL.

3.2 Reliability

Relational databases (MySQL) are more rigid in its structure of tables with columns and rows and limits to one entry per row and for every column, it contains a distinct piece of information. For such a neatly organised database that requires normalisation of up to 3NF, data redundancy is reduced and thus, would improve the reliability of the data. As compared to NoSQL which is based on denormalized data, it is less reliable as it is a BASE and values availability over any other values.

3.3 Flexibility

In terms of flexibility, relational databases (MySQL) are more rigid as compared to non-relational databases (NoSQL). Despite being considered as one of the most versatile and widely used options available, it can be rather restrictive as compared to NoSQL. In SQL, it will only be effective only when the data is structured in a very organised way required to predefined schemas before working on it. Also, all the data must follow the same structure. This generally requires significant up-front preparation and any change to the structure would be extremely difficult and disruptive for the entire system. Whereas for NoSQL, it has a more dynamic schema for unstructured data. The data can also be stored in various ways - column-oriented, document-oriented, graph-based or organized as a KeyValue store. This flexibility of NoSQL means that one can create documents without needing to define the structure beforehand and each document can have their unique structure. In addition, one can vary the syntax between databases and fields can be added along the way.

3.4 Consistency

There are two consistency models: ACID and BASE. First off, the ACID model stands for Atomic, Consistent, Isolation and Durable. Properties of this model ensures that the data is consistent and stable on the disk once a transaction is completed. Also, ACID follows a strong consistency model. While BASE represents Basic Availability, Soft-state and Eventual Consistency and this provides eventual consistency over immediate consistency for scalability, availability and resiliency. For NoSQL databases, they utilise BASE consistency model and such databases prefer availability over consistency of replicated data at write time. Since BASE is less strict than ACID, data for NoSQL will be consistent in the future either at read time or it will be consistently the same. Different from SQL which is configured for strong consistency, NoSQL depends on the DBMS used as some offer strong consistency (e.g. MongoDB) while others offer eventual consistency (Singh, V. K., 2019).

3.5 Data Storage

SQL uses highly available storage (SAN, RAID) and NoSQL uses commodity drives storage (standard HDDs, JBOD). Storing document-form data in bulk in NoSQL requires additional processing effort and more storage than highly organised MySQL data (Singh, V. K., 2019). In SQL, data storage is optimised due to normalisation and other optimisation opportunities which often leads to better performance and more efficient use of resources (Anderson, B., 2020).

3.6 Scalability

SQL databases scale vertically, meaning that to scale the database, the capacity of the server (e.g. RAM, CPU, SSD) would need to be increased. This is because the data is highly structured where the rows and columns of each table are related and linked to each other.

However, NoSQL scales horizontally, meaning that more servers can be added to scale the database. This is possible due to the lack of structure in NoSQL as the documents are independently stored and can be stored on different servers without being linked. This is what makes NoSQL the preferred choice for large and constantly changing databases (Chan, M., 2019).

4 Recommendation to WHO

After the design and implementation of both our relational (SQL) and non-relational databases (NoSQL), we found that each had their own unique strengths and weaknesses.

	Relational	Non-relational
Strengths	- Shorter run time - Easier and more intuitive to write queries	- Faster set-up time - Easier to add and remove new entries
Weakness	- Slow set-up time - Difficult to remodel structure of database	- Longer run time - Less intuitive queries

Therefore, based on the strengths and weaknesses listed above, as well as taking into account the context of the COVID-19 pandemic, we would recommend the use of a non-relational database to the WHO.

Globally, the COVID-19 pandemic has not yet become stable, and this means that the data collected to track the movements of the virus fluctuates on a day-to-day basis.

Taking into account this fact, we feel that the most important attribute of the database should be its flexibility, so that the new data can be added without delay. This will favour a flexible schema which allows changes to the database to be made easily and efficiently. In this case, users will be able to iterate quickly and continuously integrate new application features to provide value to users faster.

Non-relational databases achieve this flexibility through its fluid and document-oriented structure. When new data is recorded and received, the database can simply be updated by inserting these documents into the existing database. The data received does not need to conform to any structure, and can have certain inconsistencies such as missing values, without triggering an error.

This process is much harder to accomplish if a relational database is used. Relational databases have a rigid structure, which means that the data that is added must conform to the previously defined structure set earlier. Otherwise, the new data would not be accepted into the new database due to the data quality.

Another important attribute of the database is that it must be able handle and store large amounts of data. With how extensive the data collection with regards to this pandemic, the dataset is expected to be extremely large. Non-relational databases, compared to relational databases, are able to handle and store larger amounts of data. Therefore, our team has decided that using a non-relational database would be more appropriate for this specific situation.

This is not to say that a non-relational database is free of problems. Queries in non-relational databases are more difficult to write compared to relational databases. NoSQL is less intuitive compared to SQL, and requires a more experienced database manager in order to get the correct output from the query. However, we feel that this problem can be easily resolved by engaging experienced database managers to manage and perform operations on their database.

Also, since non-relational databases do not follow the ACID properties, they are less reliable when it comes to consistency in performance and scalability. However, based on the situation, we still believe that the benefits of using a non-relational database, such as its flexibility and ability to store large data, will offset its minor drawbacks.

5 Conclusion

Given the uncertainty of how the coronavirus will continue to affect us in the future, people from different disciplines have to come together to fight this uncertainty and emerge victorious from the situation.

Tackling the infodemic is also an extremely important part of tackling the pandemic as we are able to gain meaningful insights from the analysis of relevant information. These insights gained will then be helpful for countries to implement solutions to reduce the effect and intensity of the virus on their population effectively.

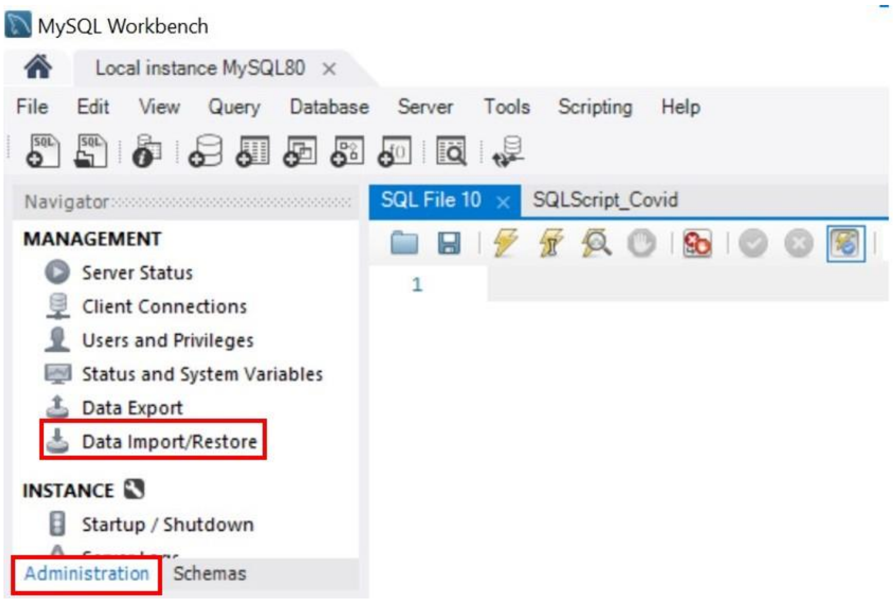
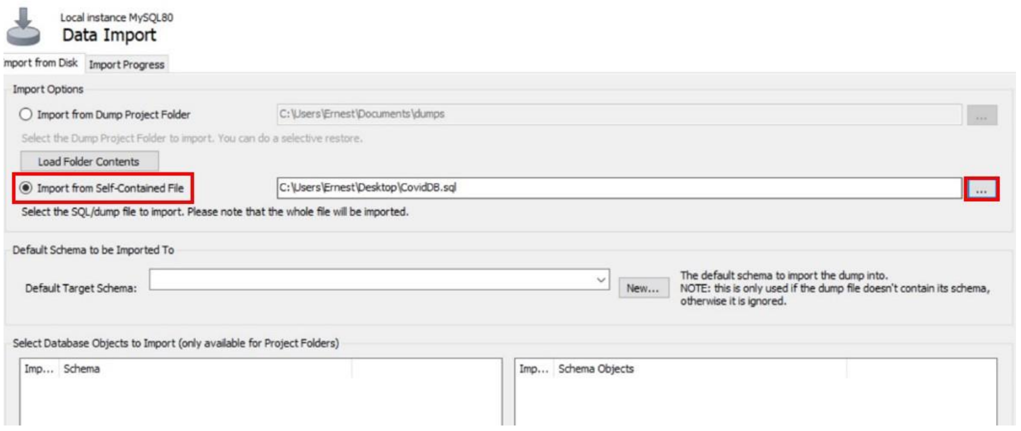
Our recommendation to the WHO on using non-relational databases to sieve through important information in their datasets would significantly help WHO find valuable statistics to make paramount decisions. Given that time is of essence, as everyday there are thousands more people getting affected by the virus and dying from it, the more efficient and flexible method is preferred and non-relational databases will be the best fit in this situation.

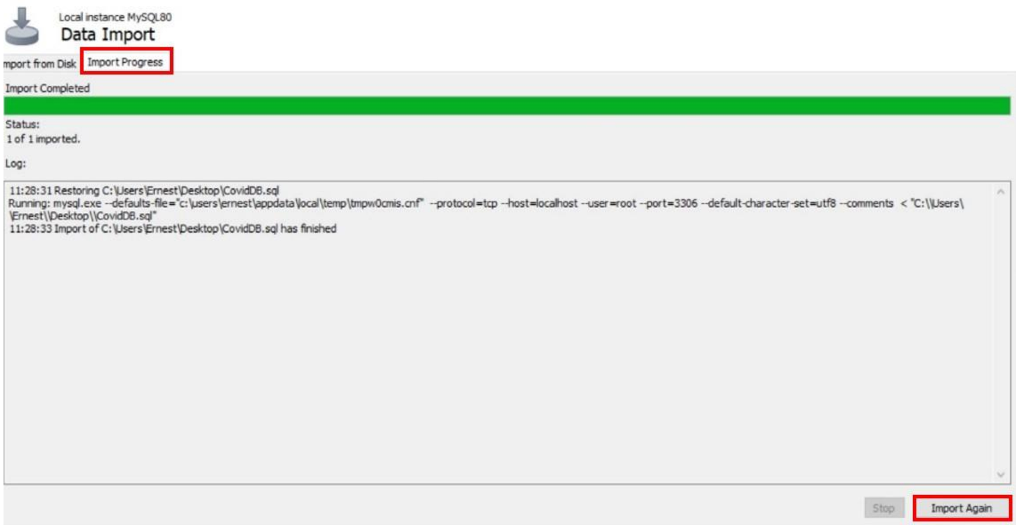
Furthermore, the information we obtain and lessons learnt from the COVID-19 situation could serve as a learning point to prevent such a pandemic from happening again or in the worst case, prepares us for when another pandemic is to hit us again.

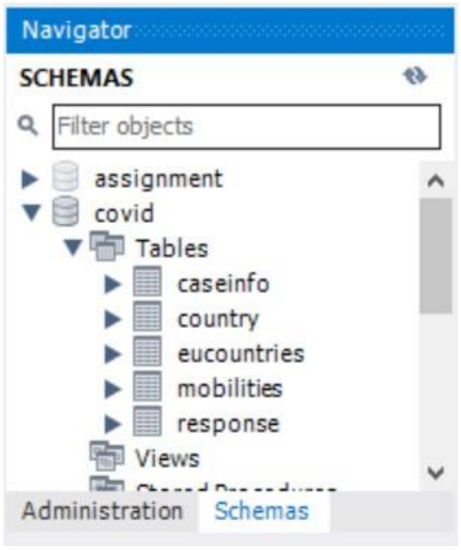
6 Appendix

6.1 Relational Database

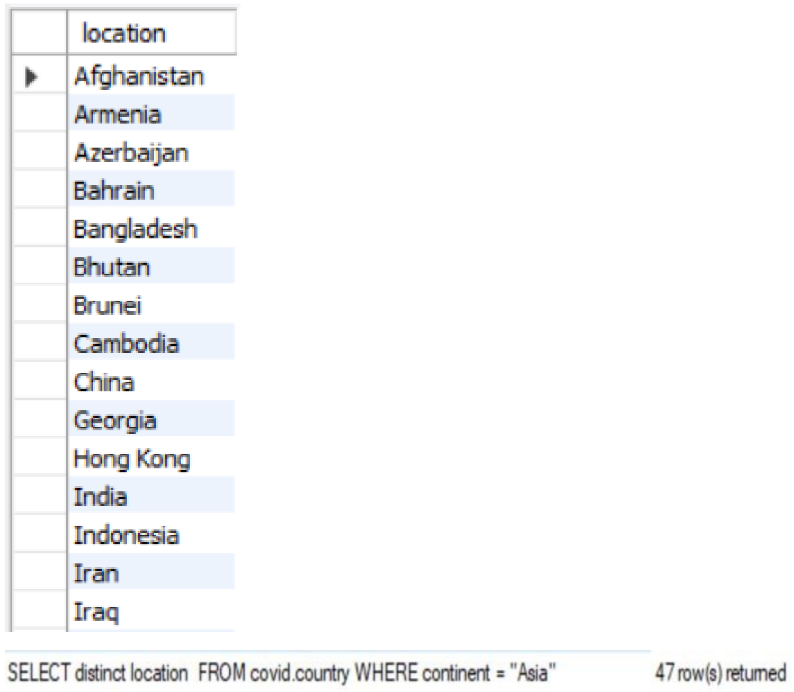
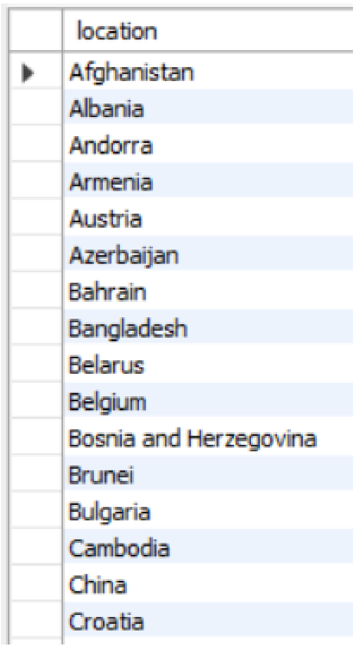
6.1.1 Import Database into MySQL

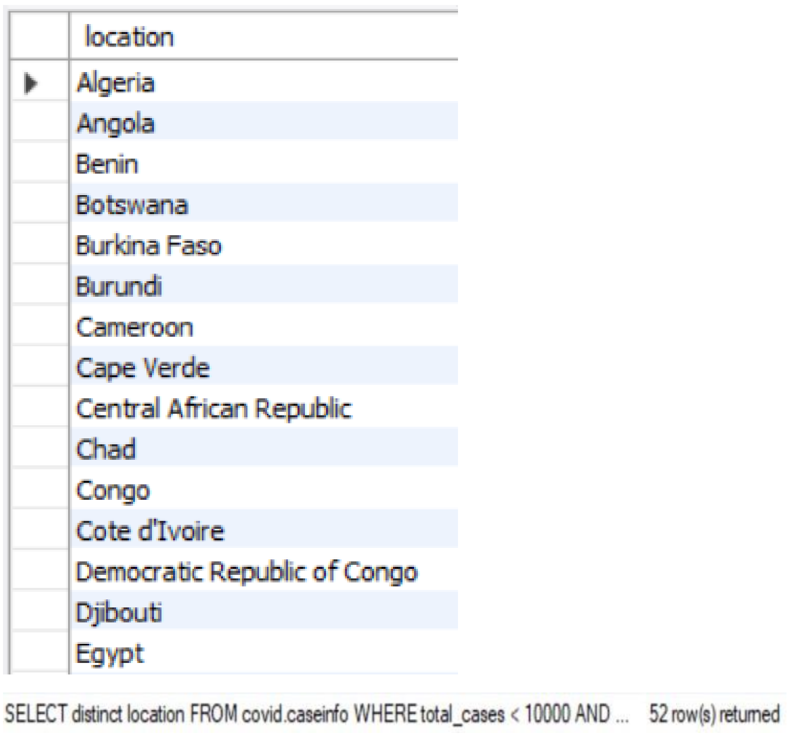
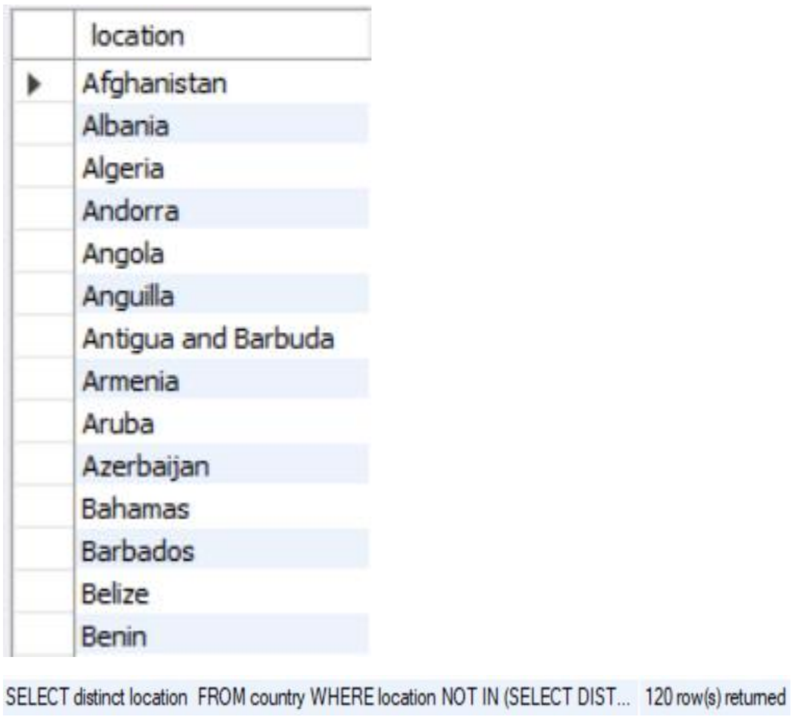
1. In the Navigator Window, select the Administration tab, select the Data Import/Restore option	 MySQL Workbench Local instance MySQL80 File Edit View Query Database Server Tools Scripting Help Navigator SQL File 10 SQLScript_Covid MANAGEMENT Server Status Client Connections Users and Privileges Status and System Variables Data Export Data Import/Restore INSTANCE Startup / Shutdown Administration Schemas
2. Select the database dump, “CovidDB” using the file explorer	 Local instance MySQL80 Data Import; mport from Disk, Import Progress; Import Options: ○ Import from Dump Project Folder C:\Users\Ernest\Documents\dumps ... Select the Dump Project Folder to import. You can do a selective restore. Load Folder Contents; ● Import from Self-Contained File C:\Users\Ernest\Desktop\CovidDB.sql ... Select the SQL/dump file to import. Please note that the whole file will be imported. Default Schema to be Imported To: Default Target Schema: New... The default schema to import the dump into. NOTE: this is only used if the dump file doesn't contain its schema, otherwise it is ignored. Select Database Objects to Import (only available for Project Folders): Imp... Schema; Imp... Schema Objects

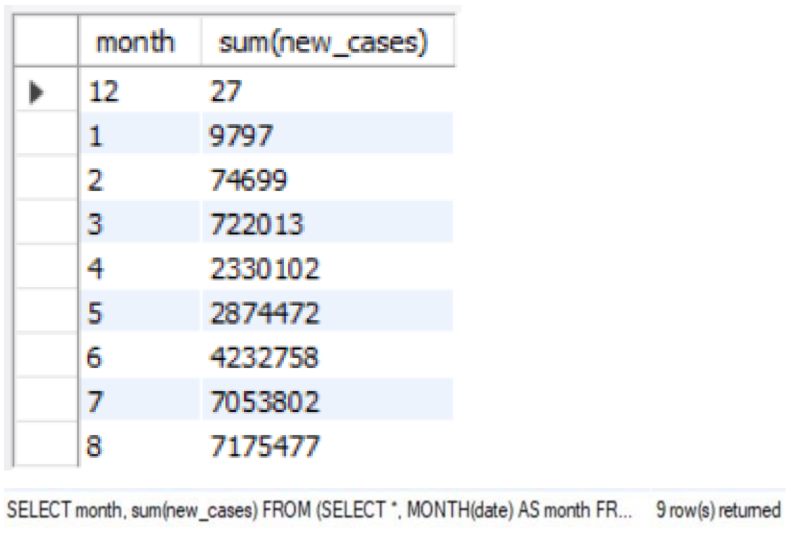
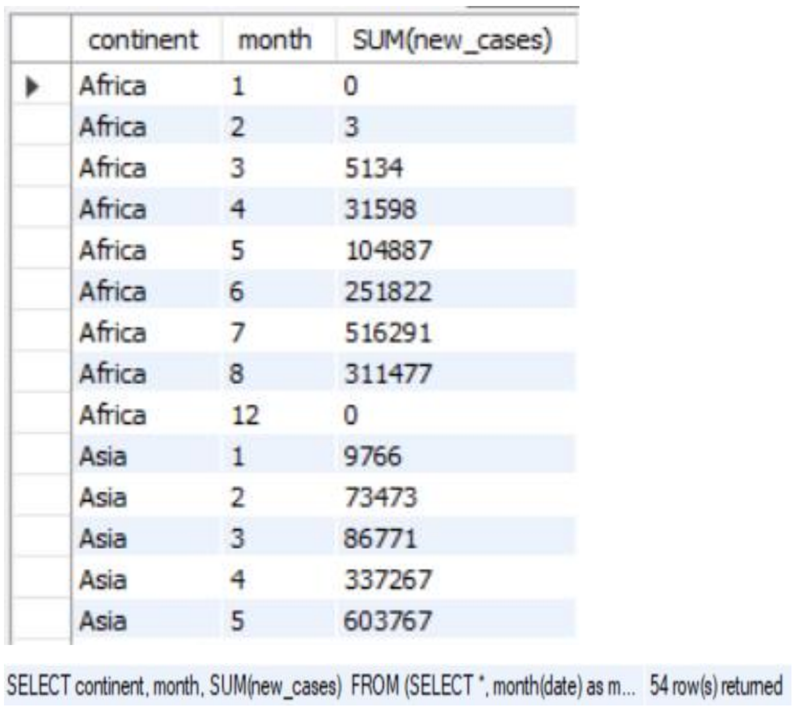
3. Select Import Progress tab and select Import located at the bottom left	 Local instance MySQL80
	Data Import
	mport from Disk
	Import Progress
	Stop
	Import AgainImport Completed
	Status:
	1 of 1 imported.
	Log:
	`11:28:31 Restoring C:\Users\Ernest\Desktop\CovidDB.sql`
	`Running: mysql.exe --defaults-file="c:\users\ernest\appdata\local\temp\tmpw0cmis.cnf" --protocol=tcp --host=localhost --user=root --port=3306 --default-character-set=utf8 --comments < "C:\\Users\\Ernest\\Desktop\\CovidDB.sql"`
	`11:28:33 Import of C:\Users\Ernest\Desktop\CovidDB.sql has finished`

4. Return to the Navigator window and refresh the Schemas tab to see the imported databases	 Navigator SCHEMAS Filter objects assignment covid Tables caseinfo country eucountries mobilities response Views Administration Schemas

6.1.2 Results for MySQL

No.	Question	Results
1	Generate a list of unique locations (countries) in Asia	 location: Afghanistan, Armenia, Azerbaijan, Bahrain, Bangladesh, Bhutan, Brunei, Cambodia, China, Georgia, Hong Kong, India, Indonesia, Iran, Iraq. SELECT distinct location FROM covid.country WHERE continent = "Asia" 47 row(s) returned
2	Generate a list of unique locations (countries) in Asia and Europe, with more than 10 total cases on 2020-04-01.	 location: Afghanistan, Albania, Andorra, Armenia, Austria, Azerbaijan, Bahrain, Bangladesh, Belarus, Belgium, Bosnia and Herzegovina, Brunei, Bulgaria, Cambodia, China, Croatia

3	Generate a list of unique locations (countries) in Africa, with less than 10,000 total cases between 2020-04-01 and 2020-04-20 (inclusive of the start date and end date).	 location: Algeria, Angola, Benin, Botswana, Burkina Faso, Burundi, Cameroon, Cape Verde, Central African Republic, Chad, Congo, Cote d'Ivoire, Democratic Republic of Congo, Djibouti, Egypt `SELECT distinct location FROM covid.caseinfo WHERE total_cases < 10000 AND ...` 52 row(s) returned
4	Generate a list of unique locations (countries) without any data on total tests.	 location: Afghanistan, Albania, Algeria, Andorra, Angola, Anguilla, Antigua and Barbuda, Armenia, Aruba, Azerbaijan, Bahamas, Barbados, Belize, Benin `SELECT distinct location FROM country WHERE location NOT IN (SELECT DIST...` 120 row(s) returned

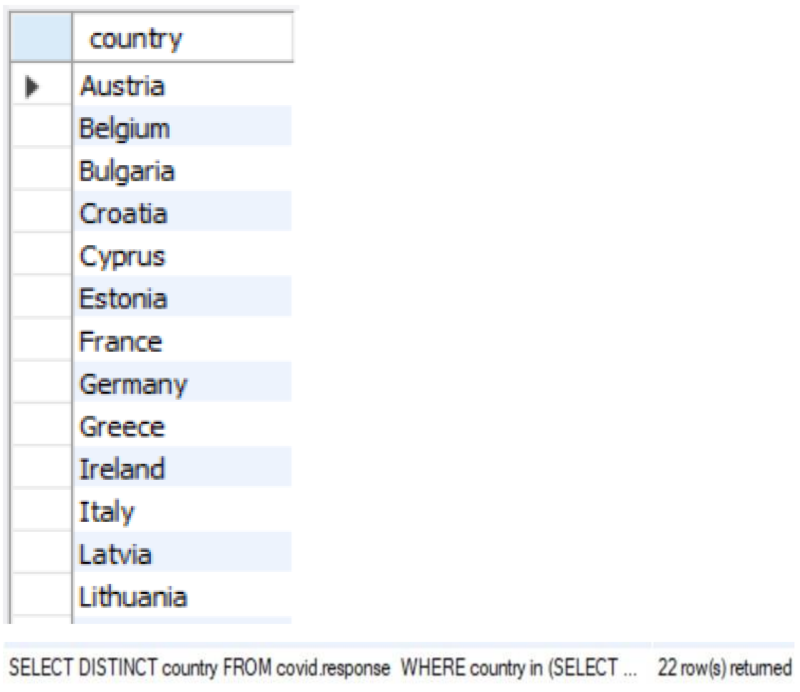
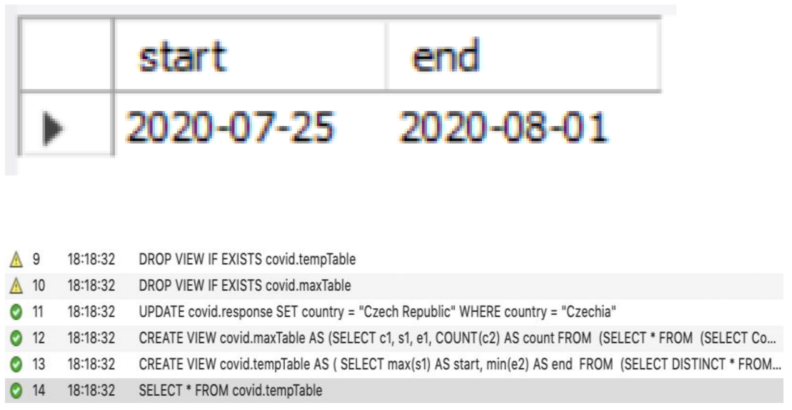
5	Conduct trend analysis, i.e., for each month, compute the total number of new cases globally.	
6	Conduct trend analysis, i.e., for each month, compute the total number of new cases in each continent.	

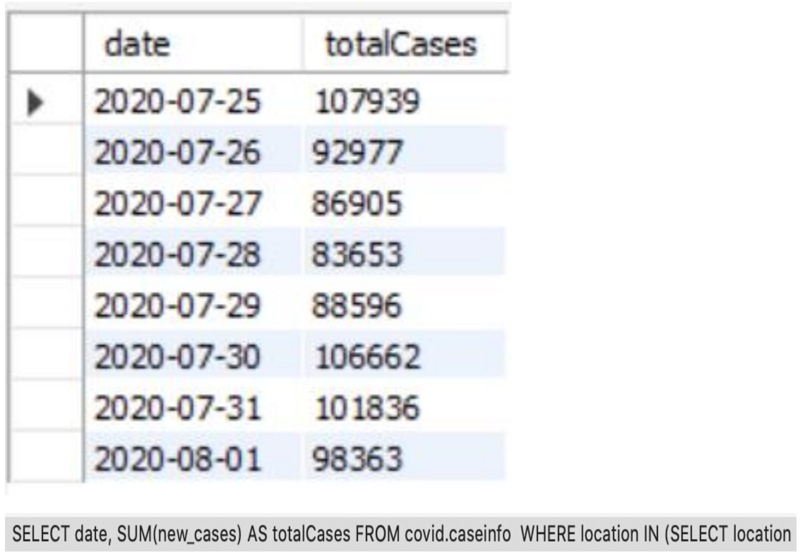
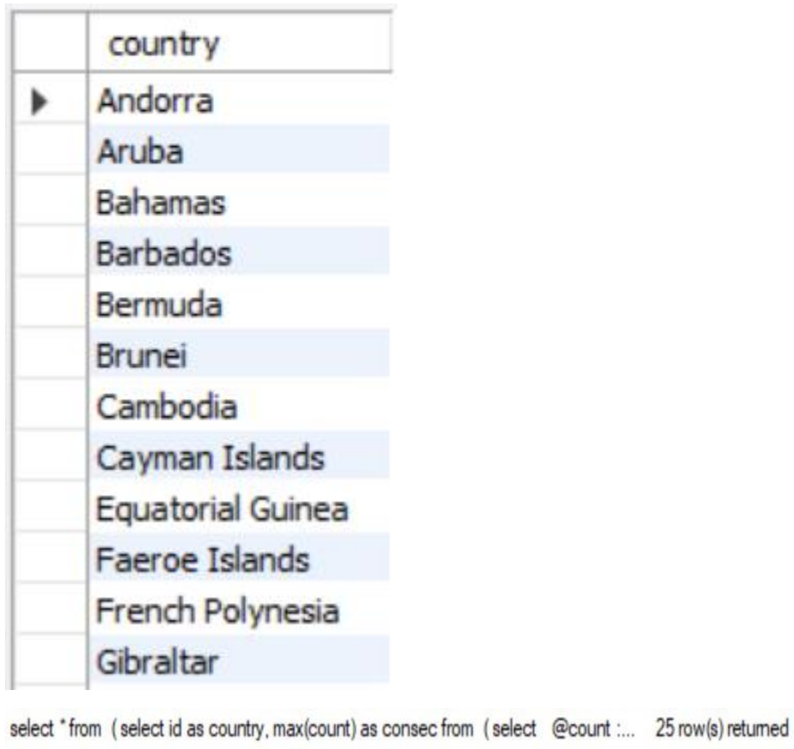
month	sum(new_cases)
12	27
1	9797
2	74699
3	722013
4	2330102
5	2874472
6	4232758
7	7053802
8	7175477

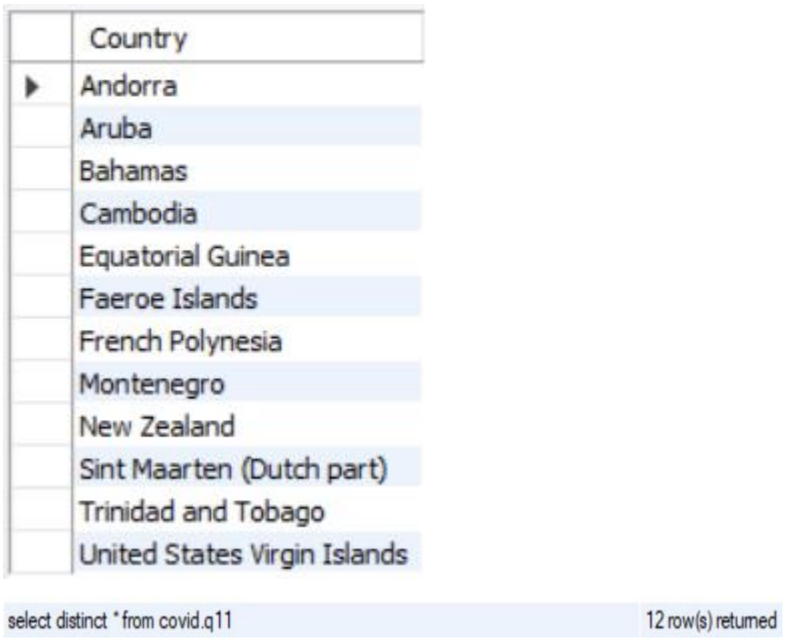
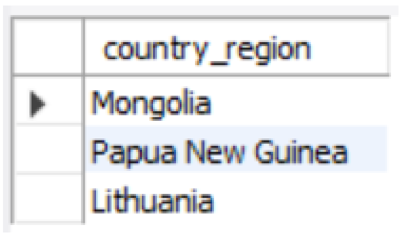
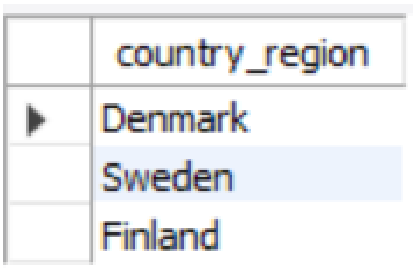
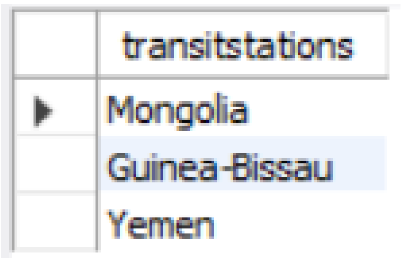
```
SELECT month, sum(new_cases) FROM (SELECT *, MONTH(date) AS month FR...     9 row(s) returned
```

continent	month	SUM(new_cases)
Africa	1	0
Africa	2	3
Africa	3	5134
Africa	4	31598
Africa	5	104887
Africa	6	251822
Africa	7	516291
Africa	8	311477
Africa	12	0
Asia	1	9766
Asia	2	73473
Asia	3	86771
Asia	4	337267
Asia	5	603767

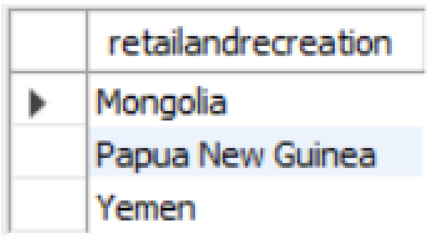
```
SELECT continent, month, SUM(new_cases)  FROM (SELECT *, month(date) as m...   54 row(s) returned
```

7	Generate a list of EU countries that have implemented mask related responses (i.e., response measure contains the word “mask”).	 country Austria Belgium Bulgaria Croatia Cyprus Estonia France Germany Greece Ireland Italy Latvia Lithuania SELECT DISTINCT country FROM covid.response WHERE country in (SELECT ... 22 row(s) returned
8	Compute the period (i.e., start date and end date) in which most EU countries have implemented MasksMandatory as the response measure. For NA values, use 1-August 2020.	 start 2020-07-25 end 2020-08-01 9 18:18:32 DROP VIEW IF EXISTS covid.tempTable 10 18:18:32 DROP VIEW IF EXISTS covid.maxTable 11 18:18:32 UPDATE covid.response SET country = "Czech Republic" WHERE country = "Czechia" 12 18:18:32 CREATE VIEW covid.maxTable AS (SELECT c1, s1, e1, COUNT(c2) AS count FROM (SELECT * FROM (SELECT Co... 13 18:18:32 CREATE VIEW covid.tempTable AS (SELECT max(s1) AS start, min(e2) AS end FROM (SELECT DISTINCT * FROM... 14 18:18:32 SELECT * FROM covid.tempTable

9	Based on the period above, conduct trend analysis for Europe and North America, i.e., for each day during the period, compute the total number of new cases.	 date | totalCases 2020-07-25 | 107939 2020-07-26 | 92977 2020-07-27 | 86905 2020-07-28 | 83653 2020-07-29 | 88596 2020-07-30 | 106662 2020-07-31 | 101836 2020-08-01 | 98363 SELECT date, SUM(new_cases) AS totalCases FROM covid.caseinfo WHERE location IN (SELECT location
10	Generate a list of unique locations (countries) that have successfully flattened the curve (i.e., achieved more than 14 days of 0 new cases, after recording more than 50 cases).	 country Andorra Aruba Bahamas Barbados Bermuda Brunei Cambodia Cayman Islands Equatorial Guinea Faeroe Islands French Polynesia Gibraltar select * from (select id as country, max(count) as consec from (select @count :... 25 row(s) returned

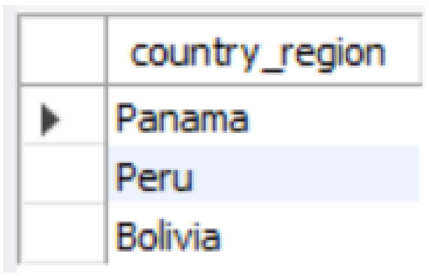
11	Second wave detection – generate a list of unique locations (countries) that have flattened the curve (as defined above) but suffered upticks in new cases (i.e., after >= 14 days, registered more than 50 cases in a subsequent 7-day window).	 Country Andorra Aruba Bahamas Cambodia Equatorial Guinea Faeroe Islands French Polynesia Montenegro New Zealand Sint Maarten (Dutch part) Trinidad and Tobago United States Virgin Islands select distinct * from covid.q11 12 row(s) returned
12	Display the top 3 countries in terms of changes from baseline in each of the place categories (i.e., grocery and pharmacy, parks, transit stations, retail and recreation, residential, and workplaces).	Grocery and Pharmacy:  country_region Mongolia Papua New Guinea Lithuania Parks:  country_region Denmark Sweden Finland Transit Stations:  transitstations Mongolia Guinea-Bissau Yemen

Retail and Recreation:



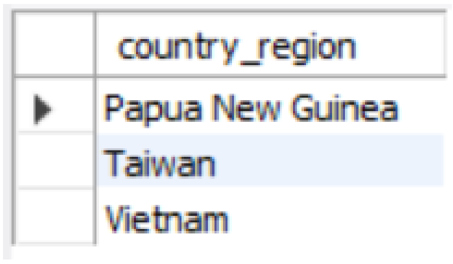
retailandrecreation
Mongolia
Papua New Guinea
Yemen

Residential:

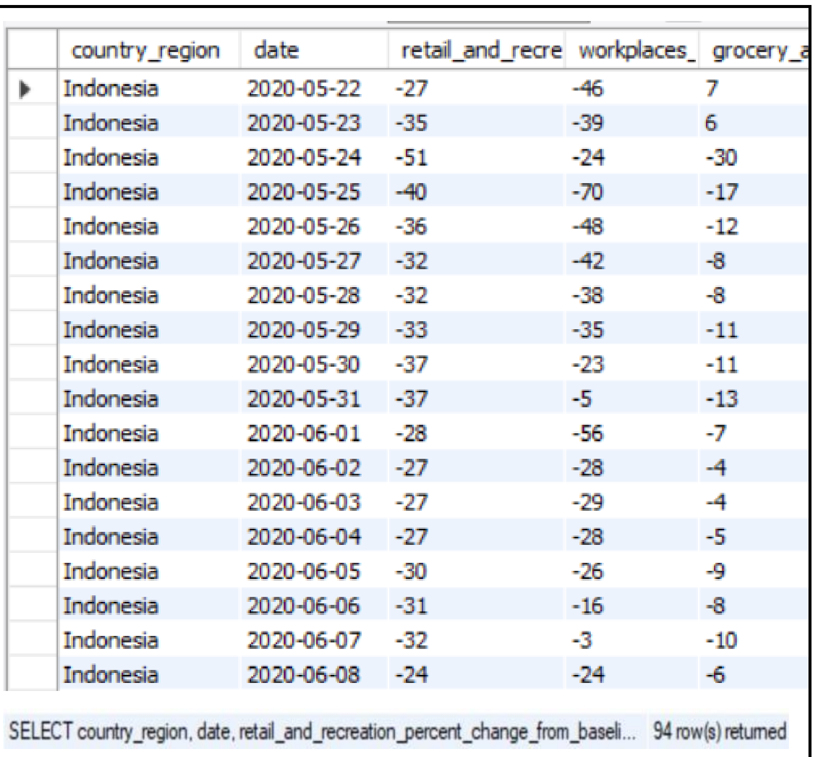


country_region
Panama
Peru
Bolivia

Workplaces:



country_region
Papua New Guinea
Taiwan
Vietnam

13	Conduct mobility trend analysis, i.e., in Indonesia, identify the date where more than 20,000 cases were recorded (D-day). Based on D-day, show the daily changes in mobility trends for the 3 place categories (i.e., retail and recreation, workplaces, and grocery and pharmacy).	

country_region	date	retail_and_recre	workplaces_	grocery_a
Indonesia	2020-05-22	-27	-46	7
Indonesia	2020-05-23	-35	-39	6
Indonesia	2020-05-24	-51	-24	-30
Indonesia	2020-05-25	-40	-70	-17
Indonesia	2020-05-26	-36	-48	-12
Indonesia	2020-05-27	-32	-42	-8
Indonesia	2020-05-28	-32	-38	-8
Indonesia	2020-05-29	-33	-35	-11
Indonesia	2020-05-30	-37	-23	-11
Indonesia	2020-05-31	-37	-5	-13
Indonesia	2020-06-01	-28	-56	-7
Indonesia	2020-06-02	-27	-28	-4
Indonesia	2020-06-03	-27	-29	-4
Indonesia	2020-06-04	-27	-28	-5
Indonesia	2020-06-05	-30	-26	-9
Indonesia	2020-06-06	-31	-16	-8
Indonesia	2020-06-07	-32	-3	-10
Indonesia	2020-06-08	-24	-24	-6

SELECT country_region, date, retail_and_recreation_percent_change_from_baseli... 94 row(s) returned

6.1.3 Implementation of MySQL

Qn	Query	Explanation

1	SELECT distinct location
	FROM covid.country
	WHERE continent = "Asia";
		To find the unique locations (countries) that are in Asia, we select distinct locations which can be found in covid.country table with their value in the “continent” column to be “Asia”.

2	SELECT location
	FROM covid.caseinfo
	WHERE date="2020-04-01" AND
	total_cases >10 AND
	location in
	(SELECT location
	FROM covid.country
	WHERE continent = "Asia" or continent
	="Europe");
		We will be taking the distinct data of the location(countries) from the caseinfo table where we set the conditions to be on the specific date: “2020-04-01” and limit the total_cases to be more than 10. Since the question wants these countries to either be in Asia or Europe, we will use nested subquery where we use “IN” to get the information of the continents and set the condition for the continent values to be either Asia or Europe for this table.

3	SELECT distinct location
	FROM covid.caseinfo
	WHERE total_cases < 10000 AND
	date>="2020-04-01" AND date<="2020-
	04-20"
	AND location in
	(SELECT location
	FROM covid.country
	WHERE continent = "Africa");
		We will filter out the distinct countries from the caseinfo table with 2 conditions: Total Cases < 10,000 and the dates to be between 2020-04-01 and 2020-04-20 and ensure that we include the two dates as well. To ensure that these countries are in Africa, we will use a nested subquery that has the condition that the continent for these countries is Africa.

4	SELECT distinct location
	FROM covid.country
	WHERE location NOT IN
	(SELECT DISTINCT location
	FROM covid.caseinfo
	WHERE total_tests != " ");
		We will first be finding the table with locations that have values in their total tests column and use it as a nested subquery along with “NOT IN” to filter out the countries that are not in this table, meaning countries who actually have no data in their total tests column.

5	SELECT month, sum(new_cases)
	FROM (SELECT *, MONTH(date) AS month FROM covid.caseinfo) as newtable1
	GROUP BY month;
		To conduct a trend analysis, we base it on the monthly trend to find out the total global new cases per month. To find the total global new cases, we will be using aggregate sum().

6	SELECT continent, month, sum(new_cases)
	FROM (SELECT *, MONTH(date) AS month FROM covid.caseinfo) as newtable2 NATURAL JOIN country
	GROUP BY continent, month
	ORDER BY continent, month;
		This question is similar to question 5 trend analysis, but since we have to find the new cases in each continent, we join the tables using NATURAL JOIN. This enables us to sort the output base on the new cases in each continent as well.

7	SELECT DISTINCT country
	FROM covid.response
	WHERE country in (SELECT country
	FROM covid.eucountries)
	AND Response_measure LIKE
	"%mask%";
		Since this question specifically targets EU Countries and we do not have any columns that represent it, we will create a new table that consists of all the EU Countries. This new table is called eucountries. Next, we will use the “LIKE” operator which allows us to compare the strings using wildcards. The “%”wildcard in “%mask%” means any sequence of characters that consist of “mask” will be filtered out. Then we use nested

		subquery to find which of these countries are EU countries.
8	DROP VIEW IF EXISTS covid.tempTable; DROP VIEW IF EXISTS covid.maxTable; UPDATE covid.response SET country = "Czech Republic" WHERE country = "Czechia"; CREATE VIEW covid.maxTable AS (SELECT c1, s1, e1, COUNT(c2) AS count FROM (SELECT * FROM (SELECT Country AS c1, Response_measure AS m1, date_start AS s1, date_end AS e1 FROM covid.response WHERE Response_measure = "MasksMandatory" AND Country IN (SELECT country FROM covid.eucountries)) AS windows INNER JOIN (SELECT Country AS c2, Response_measure AS m2, date_start AS s2, date_end AS e2 FROM covid.response WHERE Response_measure = "MasksMandatory" AND Country IN (SELECT country FROM covid.eucountries)) AS overlapping_windows	First, to find out whether EU countries have overlapping periods for the implementation of masks mandatory, we joined 2 tables with the same attributes of country, response measure, start date of implementation and end date of implementation so that we can compare a country’s implementation date individually with all other countries. We filtered out the irrelevant rows where country c1 equals to country c2 as we only want to compare between different countries. We also filtered out countries where their start date is in between the start and end date of the second country in each row. This shows that there are overlapping periods of implementation between the two countries. Next, we group the output by the country c1, its start and end date, and proceed with a count. This count represents the number of countries whose implementation overlaps with country c1’s implementation period. We sort the table to order by count starting from the highest number of countries. This table is stored as a view maxTable. Similarly, we did the same join function with the specific filters as before. From the view maxTable, we retrieved rows with the first country and the respective start date. By

	WHERE date(s1) BETWEEN date(s2) AND date(e2) AND windows.c1 != overlapping_windows.c2 ORDER BY windows.s1, overlapping_windows.c2) AS table0 GROUP BY c1, s1, e1 ORDER BY COUNT DESC); CREATE VIEW covid.tempTable AS (SELECT max(s1) AS start, min(e2) AS end FROM (SELECT DISTINCT * FROM (SELECT Country AS c1, Response_measure AS m1, date_start AS s1, date_end AS e1 FROM covid.response WHERE Response_measure = "MasksMandatory" AND Country IN (SELECT country FROM covid.eucountries)) AS windows INNER JOIN (SELECT Country AS c2, Response_measure AS m2, date_start AS s2, date_end AS e2 FROM covid.response WHERE Response_measure = "MasksMandatory" AND Country in (select country from covid.eucountries)) AS overlapping_windows WHERE date(s1) BETWEEN date(s2) AND date(e2) AND windows.c1 != overlapping_windows.c2	doing so, all the countries with the implementation period overlapping with the selected country will be shown. To find the maximum number of countries overlapping, we filter out the latest start date of country c1 and the earliest end date of country c2. These dates are retrieved and stored in the view tempTable, which is used in the next question.

	AND c1 LIKE (SELECT c1 FROM (SELECT * FROM covid.maxTable) AS table1 LIMIT 1) AND s1 LIKE (SELECT s1 FROM (SELECT * FROM covid.maxTable) AS table1 LIMIT 1)) AS table2);
	
	SELECT * FROM covid.tempTable;	

9	SELECT date, SUM(new_cases) AS totalCases FROM covid.caseinfo
	WHERE location IN (SELECT location FROM covid.country WHERE continent="Europe" OR continent="North America")
	AND date BETWEEN (SELECT start FROM covid.tempTable) AND (SELECT end FROM covid.tempTable)
	GROUP BY date;
		Using the time period obtained in Qn 8, we will only filter out Europe and North America continents and find the total new cases for each day during the period.
		This is done by using aggregate sum() to find the total new cases for each day. Followed by setting the condition by using nested subquery with "IN" to limit the location to places within Europe or North America only. Lastly, ensure that it is with the time period obtained from Qn8 and put this query into the WHERE clause as well.

10	SET @test = 0, @id=0, @count=0; SELECT * FROM (SELECT id AS country, MAX(count) as CONSEC from (SELECT @count := if(new_cases = 0 and Location = @id, @count+1, 0) as count, @test := new_cases, @id := Location as id FROM	In here, we understand the meaning of "flattened the curve" to be consecutively 14 days of 0 new cases with the condition that it has total cases of more than 50. Firstly, the countries are from another subquery whereby the countries have the condition of total cases > 50. Next, for each of these countries and initialize a counter variable whereby we +1 every time

	(SELECT location, date, total_cases, new_cases
	FROM covid.caseinfo
	WHERE total_cases > 50) AS table_1)
	AS table_2
	GROUP BY id
	) as table_2
	WHERE consec > 14;
		new cases equals 0 for that country consecutively and the moment the new cases does not equal to 0, the counter will be set back to 0. After which, we will select the maximum value of the counter variables for each country and select countries whose maximum number of consecutively 0 new cases are more than 14.

11	-- Set sql mode if not it will not work --
	SET sql_mode = '';
	
	-- Ensure the following does not exist --
	DROP TABLE IF EXISTS covid.flattenCurve;
	
	DROP TABLE IF EXISTS covid.q11;
	
	DROP PROCEDURE IF EXISTS covid.insertIfUptick;
	DROP PROCEDURE IF EXISTS covid.secondWave;
	
	-- Create flattenCurve table which has list of countries which flatten the curve --
	set @test = 0, @id=0, @count=0, @date=0, @total=0;
	create table covid.flattenCurve as (
	select id, min(date) as date
	from (select *
	from
	(select
		Here we would create a table to find which countries have successfully flatten the curve.This is similar to the query in Qn 10.
		
		Giving each row an ID for usage later on, to find the countries with upticks.
		
		Creating a procedure to add country into another table name “q11” when uptick is observed.
		
		Now, we Create a table “q11”. In this query, we will use procedure to find out if a certain rowID hass uptick.
		If the country with rowID i has flatten the curve, we will then see if there is an uptick. IF it is true, we will add one to the counter.
		
		Finally, we find the countries who suffered the second wave.

```
@count := if(new_cases = 0 and Location
= @id, @count+1, 0) as count,
@test := new_cases,  @id := Location as
id,  @date := date as date,
@total := total_cases as total
from
(select distinct location, date,
total_cases, new_cases
from covid.caseinfo where total_cases >
50)
as table_1) as table_2
where count = 14) as table_3 group by
id);

-- Add rowID to index flattenCurve table -
-
ALTER TABLE `covid`.`flattenCurve`
ADD COLUMN `rowID` INT NOT NULL
AUTO_INCREMENT AFTER `date`,
ADD PRIMARY KEY (`rowID`);

-- Procedure to insert country into a table
q11 if uptick is observed --
DELIMITER //
CREATE PROCEDURE
covid.insertIfUptick(
        IN inputDate date,
   IN inputLoc varchar(255)
)
BEGIN
        INSERT INTO covid.q11 (Country)
        select location from (
```

	SELECT x.location, x.date, x.new_cases, SUM(y.new_cases) as rollingSum FROM (select * from caseinfo where Location = inputLoc and date >= date(inputDate) order by date desc) x JOIN (select * from caseinfo where Location = inputLoc and date >= date(inputDate) order by date desc) y ON y.date BETWEEN x.date - INTERVAL 7-1 DAY AND x.date GROUP BY x.date, x.new_cases, x.location) as table_1 where rollingSum > 50 limit 1; END // DELIMITER ; -- Create q11 table which is the answer to q11 -- CREATE TABLE covid.q11 (Country VARCHAR(255) NULL); -- Create procedure that will run through the entire flattenCurve table to check for upticks with the checkIfUptick function -- DELIMITER //	

	CREATE PROCEDURE covid.secondWave()
	BEGIN
	DECLARE n INT DEFAULT 0;
	DECLARE i INT DEFAULT 1;
	DECLARE tempDate date;
	DECLARE tempLoc varchar(255);
	SELECT COUNT(*) FROM covid.flattenCurve INTO n;
	WHILE i<=n DO
	    select id from covid.flattenCurve where rowID = i into tempLoc;
	    select date from covid.flattenCurve where rowID = i into tempDate;
	    Call insertIfUptick(tempDate, tempLoc);
	    SET i = i + 1;
	END WHILE;
	END //
	DELIMITER ;
	
	call covid.secondWave();
	select distinct * from covid.q11;	

12	SELECT country_region AS retailandrecreation FROM
	(SELECT country_region, sum(retail_and_recreation_percent_change_from_baseline) AS rr
	FROM covid_world.mobility GROUP BY country_region
	ORDER BY rr DESC
	LIMIT 3) AS table1;
		According to the different place categories, use the aggregate sum() to find the changes from the baseline and we have to group them by country. To find the top 3 countries with the most changes from baseline, we order by the sum obtained previously and obtain the top 3 using LIMIT 3. Repeat the same procedures for the different place categories.

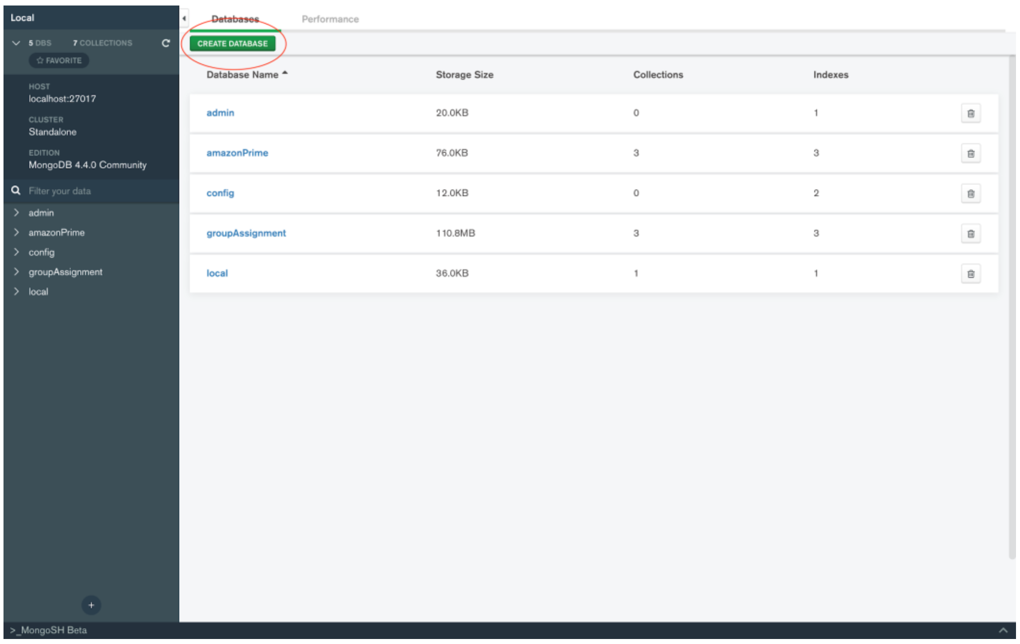
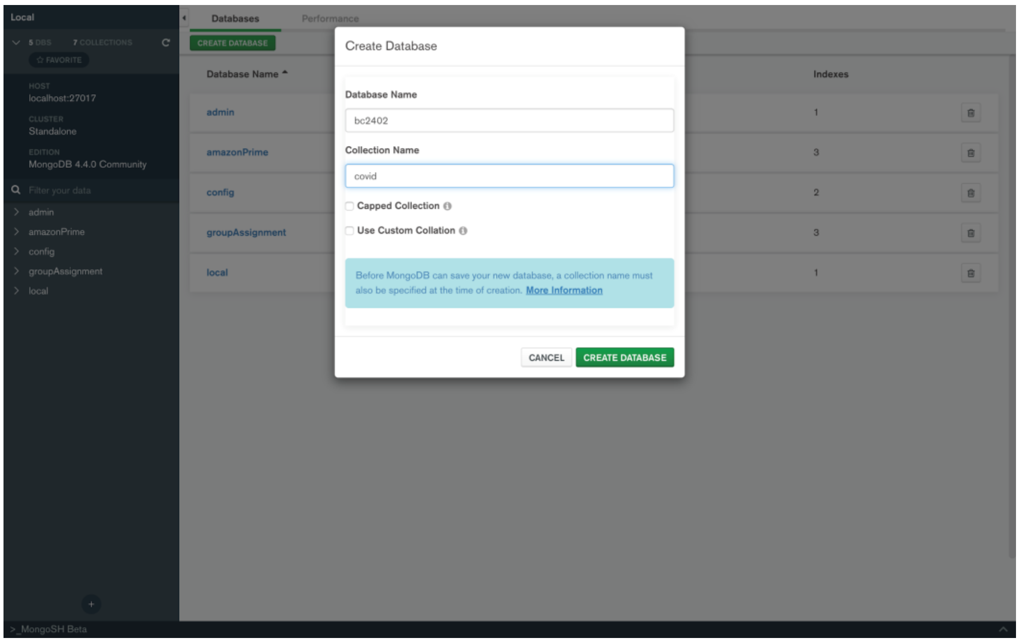
	SELECT country_region AS groceryandpharmacy FROM (SELECT country_region, sum(grocery_and_pharmacy_percent_change_from_baseline) AS gp FROM covid_world.mobility GROUP BY country_region ORDER BY gp DESC LIMIT 3) AS table1;
	
	SELECT country_region AS parks FROM (SELECT country_region, sum(parks_percent_change_from_baseline) AS p FROM covid_world.mobility GROUP BY country_region ORDER BY p DESC LIMIT 3) AS table1;
	
	SELECT country_region AS transitstations FROM (SELECT country_region, sum(transit_stations_percent_change_from_baseline) AS ts FROM covid_world.mobility GROUP BY country_region ORDER BY ts DESC LIMIT 3) AS table1;
	
	SELECT country_region AS workplaces FROM	

	(SELECT country_region,
	sum(workplaces_percent_change_from_baseline) AS w
	FROM covid_world.mobility
	GROUP BY country_region
	ORDER BY w DESC
	LIMIT 3) AS table1;
	
	SELECT country_region AS residential
	FROM
	(SELECT country_region,
	sum(residential_percent_change_from_baseline) AS r
	FROM covid_world.mobility
	GROUP BY country_region
	ORDER BY r DESC
	LIMIT 3) AS table1;	

13	SELECT country_region, date,
	retail_and_recreation_percent_change_from_baseline,
	workplaces_percent_change_from_baseline,
	grocery_and_pharmacy_percent_change_from_baseline
	from mobilities
	WHERE country_region = "Indonesia"
	AND
	date >= (SELECT date FROM caseinfo
	WHERE total_cases >= 20000 AND
	location = "Indonesia"
	LIMIT 1);
		After obtaining the date where total cases exceed 20,000 in Indonesia, we find the 3 percentage change required for all the subsequent dates.

6.2 Non-Relational Database

6.2.1 Import JSON into MongoDB

1. In the Navigator Window, select the Administration tab, select the create database option	
2. Name the database and give the name of the first collection within the database	

Local
5 DBS 7 COLLECTIONS
HOST localhost:27017
CLUSTER Standalone
EDITION MongoDB 4.4.0 Community

Database Name	Storage Size	Collections	Indexes
admin	20.0KB	0	1
amazonPrime	76.0KB	3	3
config	12.0KB	0	2
groupAssignment	110.8MB	3	3
local	36.0KB	1	1

Create Database

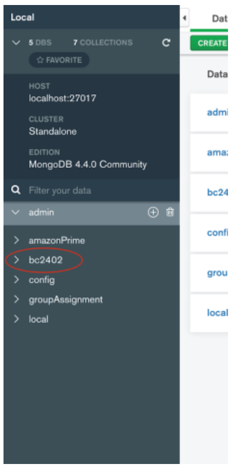
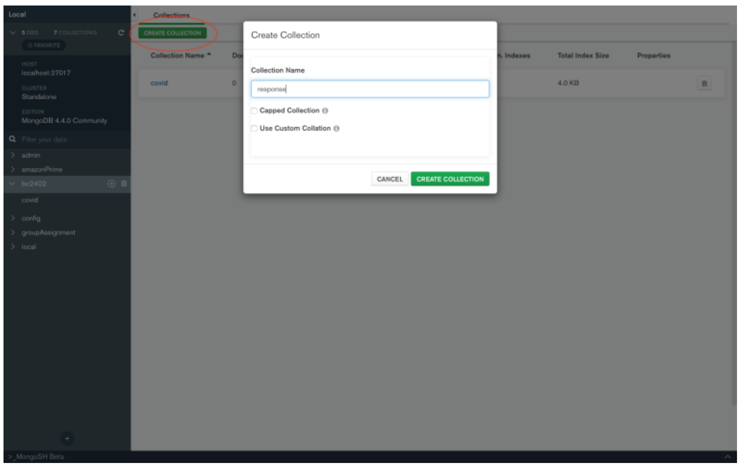
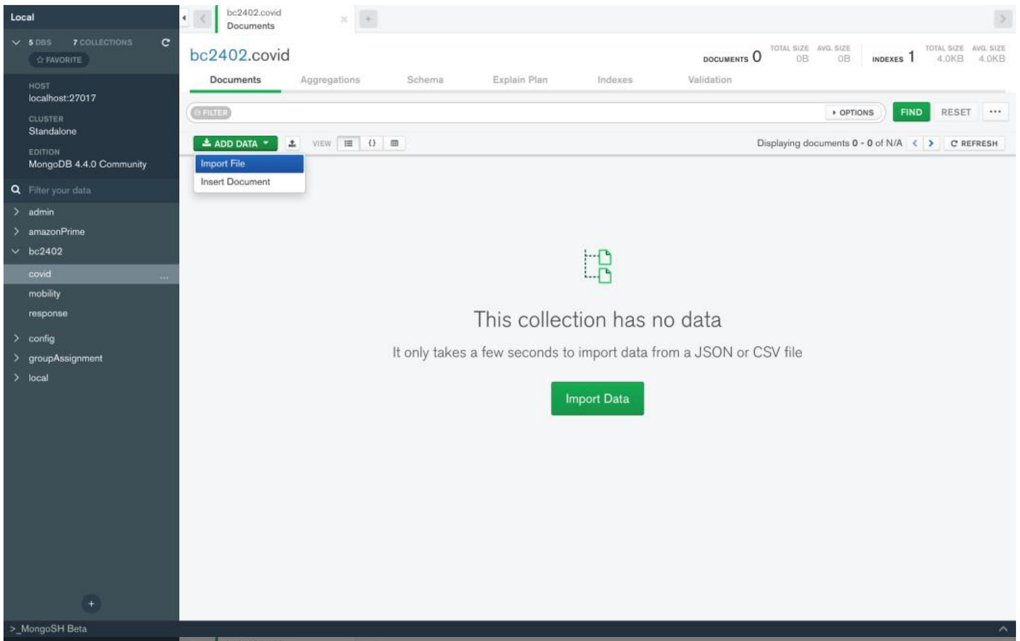
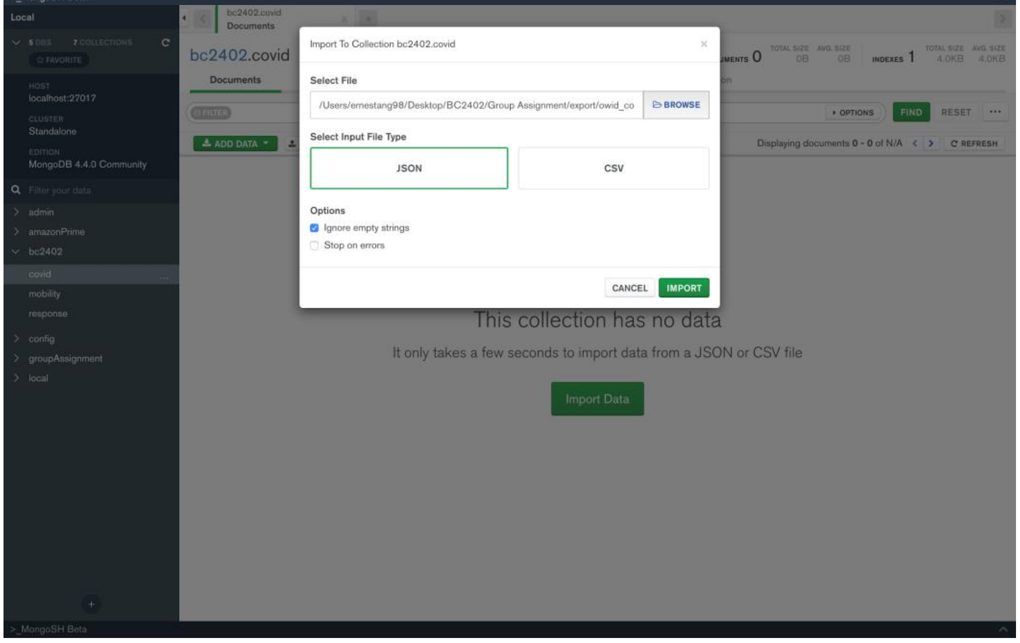
Database Name: bc2402

Collection Name: covid

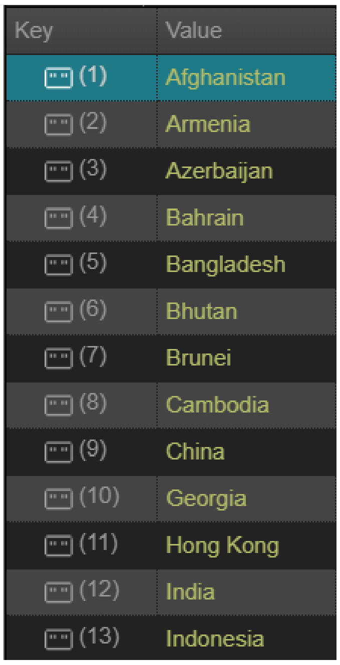
Capped Collection

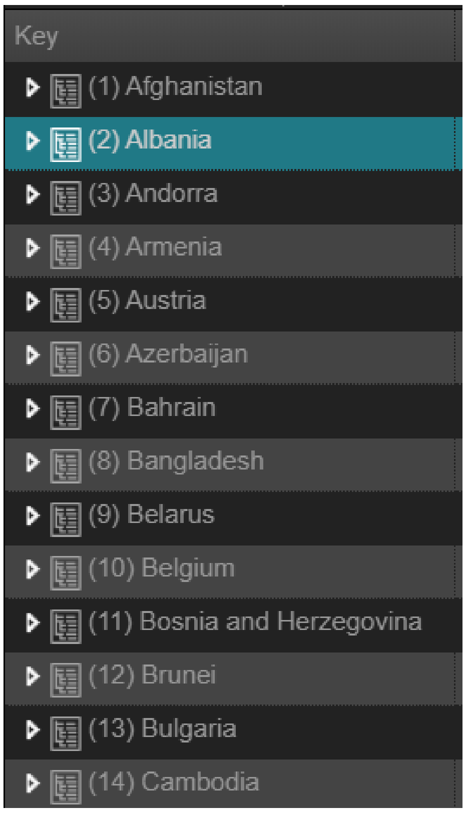
Use Custom Collation

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. More Information

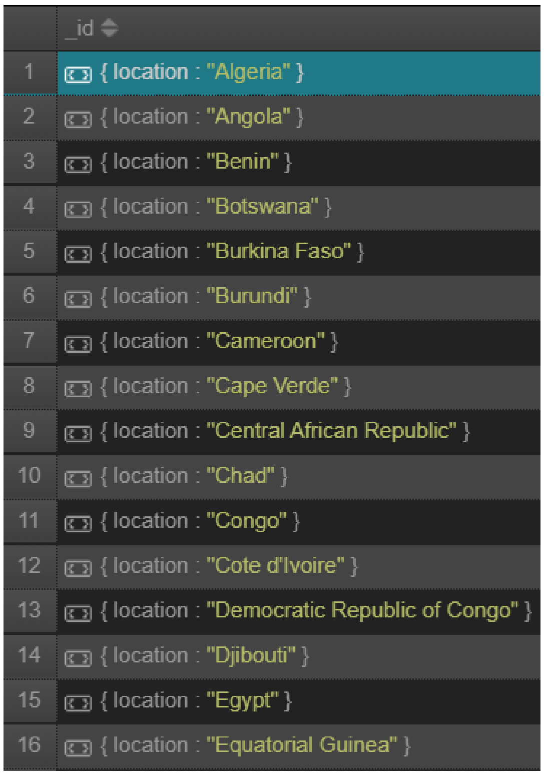
3. Navigate to your created database under the administration tab and create 2 additional collections	 Local 5 DBS 7 COLLECTIONS FAVORITE HOST localhost:27017 CLUSTER Standalone EDITION MongoDB 4.4.0 Community Filter your data admin amazonPrime bc2402 config groupAssignment local CREATE  Collections CREATE COLLECTION Create Collection Collection Name response Capped Collection Use Custom Collation CANCEL CREATE COLLECTION
4. Import the json files to the relevant collection. Repeat this 2 more times.	 bc2402.covid Documents Aggregations Schema Explain Plan Indexes Validation ADD DATA Import File Insert Document FIND RESET covid mobility response This collection has no data It only takes a few seconds to import data from a JSON or CSV file Import Data  Import To Collection bc2402.covid Select File /Users/ernestang98/Desktop/BC2402/Group Assignment/export/owid_co BROWSE Select Input File Type JSON CSV Options Ignore empty strings Stop on errors CANCEL IMPORT

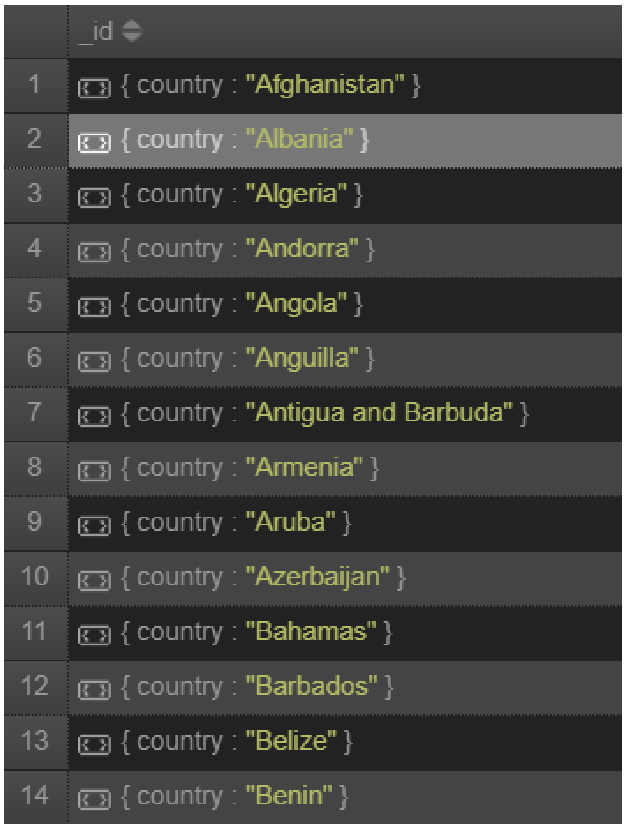
6.2.2 Results for NoSQL

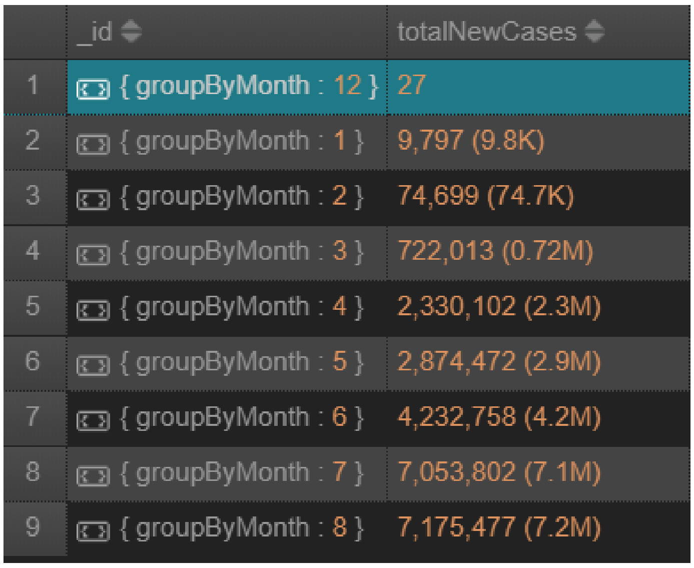
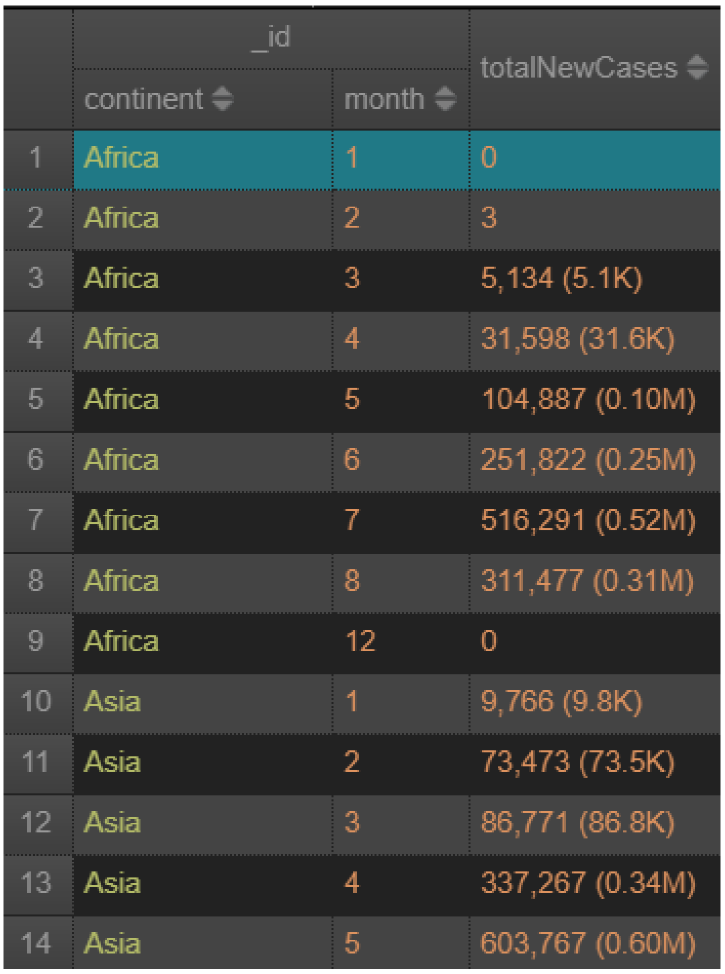
<table>
<tr><th>No.</th><th>Question</th><th>Results</th></tr>
<tr><td>1</td><td>Generate a list of unique locations (countries) in Asia</td><td>

```
Key     Value
(1)     Afghanistan
(2)     Armenia
(3)     Azerbaijan
(4)     Bahrain
(5)     Bangladesh
(6)     Bhutan
(7)     Brunei
(8)     Cambodia
(9)     China
(10)    Georgia
(11)    Hong Kong
(12)    India
(13)    Indonesia
```
</td></tr>
<tr><td>2</td><td>Generate a list of unique locations (countries) in Asia and Europe, with more than 10 total cases on 2020-04-01.</td><td>

```
Key
(1) Afghanistan
(2) Albania
(3) Andorra
(4) Armenia
(5) Austria
(6) Azerbaijan
(7) Bahrain
(8) Bangladesh
(9) Belarus
(10) Belgium
(11) Bosnia and Herzegovina
(12) Brunei
(13) Bulgaria
(14) Cambodia
```
</td></tr>
</table>

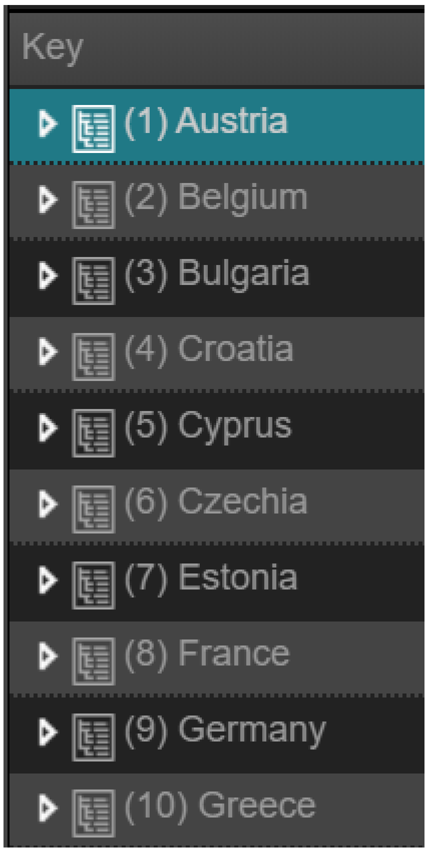
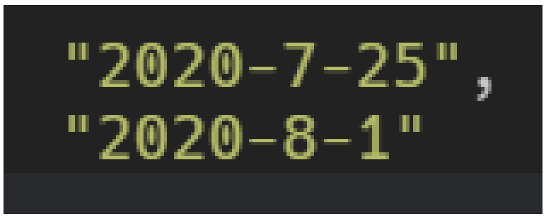
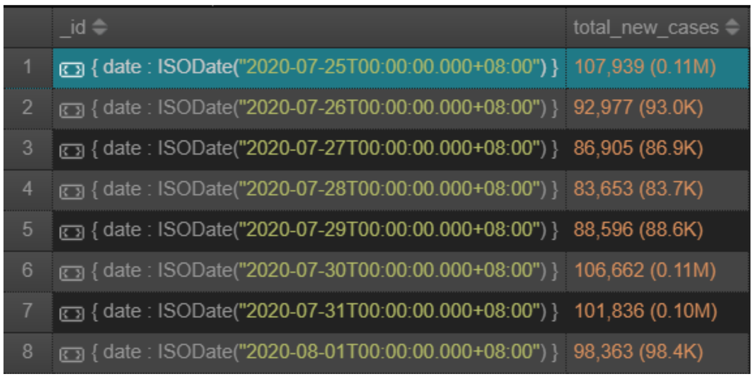
3	Generate a list of unique locations (countries) in Africa, with less than 10,000 total cases between 2020-04-01 and 2020-04-20 (inclusive of the start date and end date).	
		_id
		1 { location : "Algeria" }
		2 { location : "Angola" }
		3 { location : "Benin" }
		4 { location : "Botswana" }
		5 { location : "Burkina Faso" }
		6 { location : "Burundi" }
		7 { location : "Cameroon" }
		8 { location : "Cape Verde" }
		9 { location : "Central African Republic" }
		10 { location : "Chad" }
		11 { location : "Congo" }
		12 { location : "Cote d'Ivoire" }
		13 { location : "Democratic Republic of Congo" }
		14 { location : "Djibouti" }
		15 { location : "Egypt" }
		16 { location : "Equatorial Guinea" }

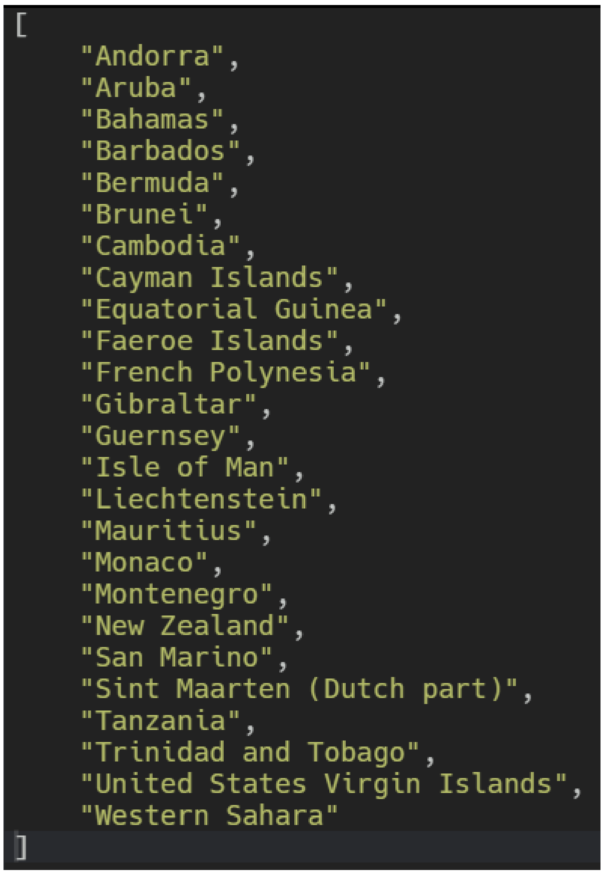
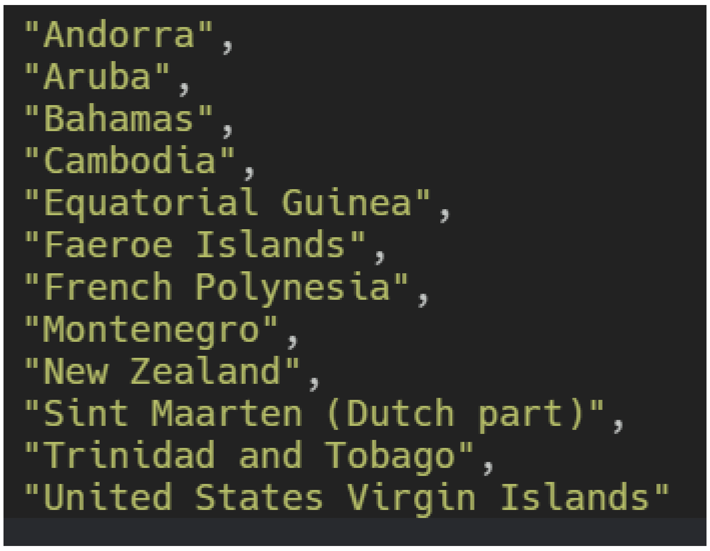
4	Generate a list of unique locations (countries) without any data on total tests.	
		_id
		1 { country : "Afghanistan" }
		2 { country : "Albania" }
		3 { country : "Algeria" }
		4 { country : "Andorra" }
		5 { country : "Angola" }
		6 { country : "Anguilla" }
		7 { country : "Antigua and Barbuda" }
		8 { country : "Armenia" }
		9 { country : "Aruba" }
		10 { country : "Azerbaijan" }
		11 { country : "Bahamas" }
		12 { country : "Barbados" }
		13 { country : "Belize" }
		14 { country : "Benin" }

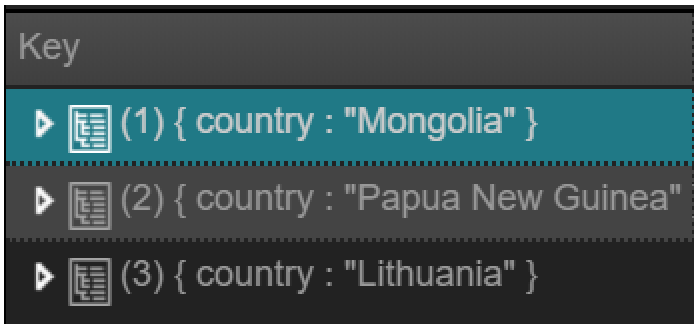
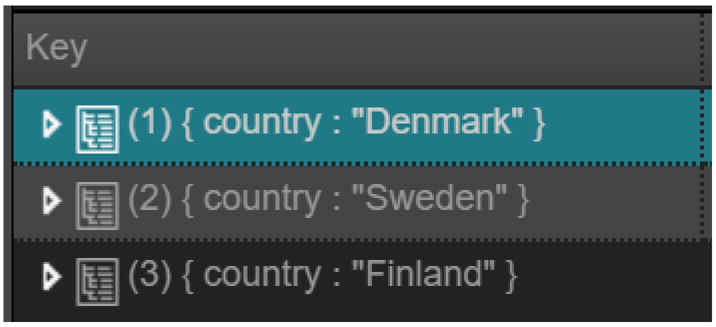
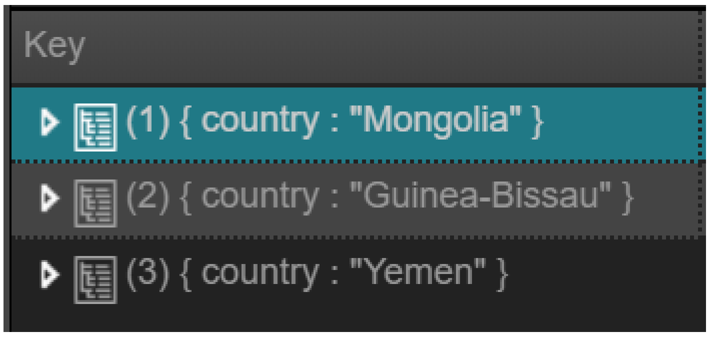
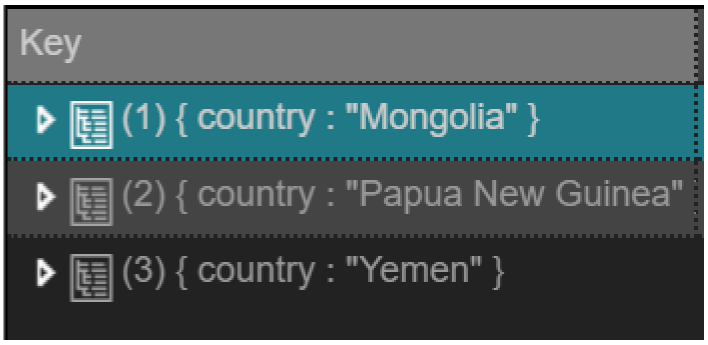
5	Conduct trend analysis, i.e., for each month, compute the total number of new cases globally.	
6	Conduct trend analysis, i.e., for each month, compute the total number of new cases in each continent.	

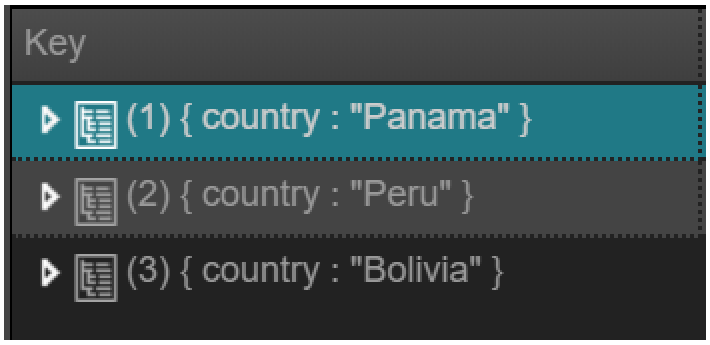
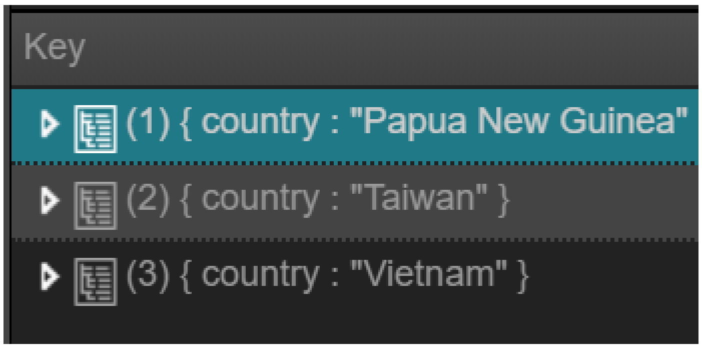
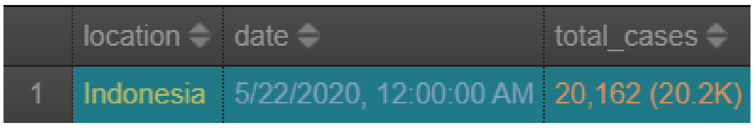
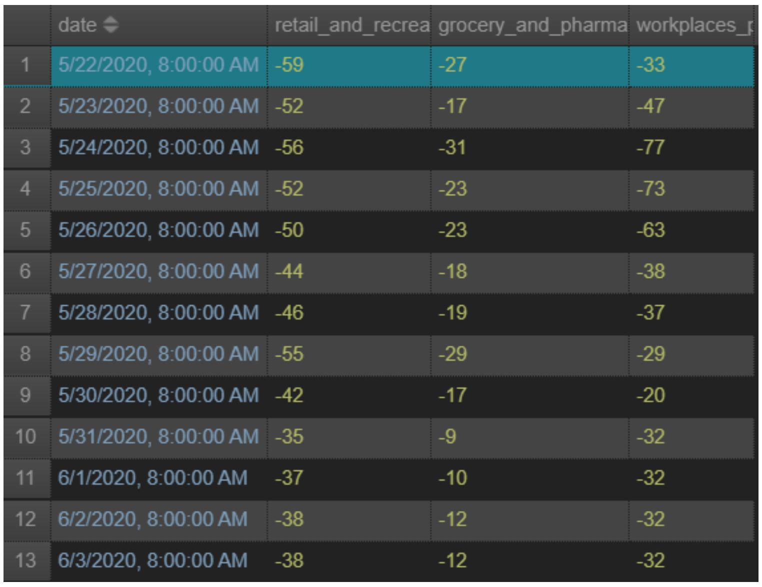
	_id	totalNewCases
1	{ groupByMonth : 12 }	27
2	{ groupByMonth : 1 }	9,797 (9.8K)
3	{ groupByMonth : 2 }	74,699 (74.7K)
4	{ groupByMonth : 3 }	722,013 (0.72M)
5	{ groupByMonth : 4 }	2,330,102 (2.3M)
6	{ groupByMonth : 5 }	2,874,472 (2.9M)
7	{ groupByMonth : 6 }	4,232,758 (4.2M)
8	{ groupByMonth : 7 }	7,053,802 (7.1M)
9	{ groupByMonth : 8 }	7,175,477 (7.2M)

	_id		totalNewCases
	continent	month	
1	Africa	1	0
2	Africa	2	3
3	Africa	3	5,134 (5.1K)
4	Africa	4	31,598 (31.6K)
5	Africa	5	104,887 (0.10M)
6	Africa	6	251,822 (0.25M)
7	Africa	7	516,291 (0.52M)
8	Africa	8	311,477 (0.31M)
9	Africa	12	0
10	Asia	1	9,766 (9.8K)
11	Asia	2	73,473 (73.5K)
12	Asia	3	86,771 (86.8K)
13	Asia	4	337,267 (0.34M)
14	Asia	5	603,767 (0.60M)

7	Generate a list of EU countries that have implemented mask related responses (i.e., response measure contains the word “mask”).	 Key (1) Austria (2) Belgium (3) Bulgaria (4) Croatia (5) Cyprus (6) Czechia (7) Estonia (8) France (9) Germany (10) Greece
8	Compute the period (i.e., start date and end date) in which most EU countries have implemented MasksMandatory as the response measure. For NA values, use 1-August 2020.	 "2020-7-25", "2020-8-1"
9	Based on the period above, conduct trend analysis for Europe and North America, i.e., for each day during the period, compute the total number of new cases.	 | | _id | total_new_cases | | 1 | { date : ISODate("2020-07-25T00:00:00.000+08:00") } | 107,939 (0.11M) | | 2 | { date : ISODate("2020-07-26T00:00:00.000+08:00") } | 92,977 (93.0K) | | 3 | { date : ISODate("2020-07-27T00:00:00.000+08:00") } | 86,905 (86.9K) | | 4 | { date : ISODate("2020-07-28T00:00:00.000+08:00") } | 83,653 (83.7K) | | 5 | { date : ISODate("2020-07-29T00:00:00.000+08:00") } | 88,596 (88.6K) | | 6 | { date : ISODate("2020-07-30T00:00:00.000+08:00") } | 106,662 (0.11M) | | 7 | { date : ISODate("2020-07-31T00:00:00.000+08:00") } | 101,836 (0.10M) | | 8 | { date : ISODate("2020-08-01T00:00:00.000+08:00") } | 98,363 (98.4K) |

10	Generate a list of unique locations (countries) that have successfully flattened the curve (i.e., achieved more than 14 days of 0 new cases, after recording more than 50 cases).	 ```["Andorra", "Aruba", "Bahamas", "Barbados", "Bermuda", "Brunei", "Cambodia", "Cayman Islands", "Equatorial Guinea", "Faeroe Islands", "French Polynesia", "Gibraltar", "Guernsey", "Isle of Man", "Liechtenstein", "Mauritius", "Monaco", "Montenegro", "New Zealand", "San Marino", "Sint Maarten (Dutch part)", "Tanzania", "Trinidad and Tobago", "United States Virgin Islands", "Western Sahara"]```
11	Second wave detection – generate a list of unique locations (countries) that have flattened the curve (as defined above) but suffered upticks in new cases (i.e., after >= 14 days, registered more than 50 cases in a subsequent 7-day window).	 ```"Andorra", "Aruba", "Bahamas", "Cambodia", "Equatorial Guinea", "Faeroe Islands", "French Polynesia", "Montenegro", "New Zealand", "Sint Maarten (Dutch part)", "Trinidad and Tobago", "United States Virgin Islands"```

12	Display the top 3 countries in terms of changes from baseline in each of the place categories (i.e., grocery and pharmacy, parks, transit stations, retail and recreation, residential, and workplaces)	Grocery and Pharmacy:  Key (1) { country : "Mongolia" } (2) { country : "Papua New Guinea" (3) { country : "Lithuania" } Parks:  Key (1) { country : "Denmark" } (2) { country : "Sweden" } (3) { country : "Finland" } Transit Stations:  Key (1) { country : "Mongolia" } (2) { country : "Guinea-Bissau" } (3) { country : "Yemen" } Retail and Recreation:  Key (1) { country : "Mongolia" } (2) { country : "Papua New Guinea" (3) { country : "Yemen" }

		Residential:  Workplaces: 
13	Conduct mobility trend analysis, i.e., in Indonesia, identify the date where more than 20,000 cases were recorded (D-day). Based on D-day, show the daily changes in mobility trends for the 3 place categories (i.e., retail and recreation, workplaces, and grocery and pharmacy).	 

Residential:

```
Key
(1) { country : "Panama" }
(2) { country : "Peru" }
(3) { country : "Bolivia" }
```

Workplaces:

```
Key
(1) { country : "Papua New Guinea"
(2) { country : "Taiwan" }
(3) { country : "Vietnam" }
```

	location	date	total_cases
1	Indonesia	5/22/2020, 12:00:00 AM	20,162 (20.2K)

	date	retail_and_recrea	grocery_and_pharma	workplaces_
1	5/22/2020, 8:00:00 AM	-59	-27	-33
2	5/23/2020, 8:00:00 AM	-52	-17	-47
3	5/24/2020, 8:00:00 AM	-56	-31	-77
4	5/25/2020, 8:00:00 AM	-52	-23	-73
5	5/26/2020, 8:00:00 AM	-50	-23	-63
6	5/27/2020, 8:00:00 AM	-44	-18	-38
7	5/28/2020, 8:00:00 AM	-46	-19	-37
8	5/29/2020, 8:00:00 AM	-55	-29	-29
9	5/30/2020, 8:00:00 AM	-42	-17	-20
10	5/31/2020, 8:00:00 AM	-35	-9	-32
11	6/1/2020, 8:00:00 AM	-37	-10	-32
12	6/2/2020, 8:00:00 AM	-38	-12	-32
13	6/3/2020, 8:00:00 AM	-38	-12	-32

6.2.3 Implementation for NoSQL

Qn	Query	Explanation

1	db.covid.deleteMany({location:"World"})
	 db.covid.deleteMany({location:"International"})
	 db.mobility.deleteMany({sub_region_1:{$ne:""}})
	 db.mobility.deleteMany({sub_region_2:{$ne:""}})
	 db.mobility.deleteMany({metro_area:{ $ne:""}})
		Performance of data cleaning by deleting rows in covid and mobility collection as it is not needed and will only interfere with our queries and results.

2	db.covid.distinct ("location", {"continent":"Asia"})	To get the distinct countries in Asia, we used the distinct function which would get the unique countries from the continent Asia.
3	db.covid.aggregate([{$match:{$or: [{continent:"Asia"}, {continent:"Europe"}]}}, {$match:{date:ISODate("2020-04-01T00:00:00.000+08:00")}}, {$project:{_id:0, location:1 , total_cases: {$convert: { input: "$total_cases", to: "double"}}}}, {$match:{"total_cases":{$gt:10.0}}}, {$group:{_id:"$location"}},	The aggregate function was used, where the documents belonging to the countries in Asia and Europe were first obtained. Then, the documents on 2020-04-01 were obtained. Then, the location and total cases in double data type from all the documents were obtained. Then, the documents having more than 10 total cases were

	{$sort:{_id:1}}])	obtained. Lastly, the list of distinct countries were obtained and sorted in alphabetical order.
4	db.covid.aggregate([{$project:{_id:0, location:1, continent:1, date:1, total_cases:{$convert: { input: "$total_cases", to: "double", onError:0.0, onNull:0.0}}}}, {$match:{$and:[{continent:"Africa"}, {total_cases:{$lt:10000}}, {date:{$gte:ISODate("2020-04-01T00:00:00.000+08:00")}}, {date:{$lt:ISODate("2020-04-21T00:00:00.000+08:00")}}]}}, {$group:{_id:"$location"}}, {$sort:{_id:1}}])	The aggregate function is used where the location, continent, date and total cases in double data type were obtained from all documents. Documents from Africa which have total cases less than 10,000 between 2020-04-01 and 2020-04-21 were obtained, using “and” function which ensures that the documents fulfill all conditions. Unique locations from the documents were obtained. Lastly, the list of countries were sorted in alphabetical order.
5	db.covid.aggregate([{$project:{_id:0, total_tests:{$convert: { input: "$total_tests", to: "double", onError:" "}},location:1}}, {$group:{_id:{country:"$location"},totalTests:{$sum:"$total_tests"}}},	Aggregate function is used to obtain total tests in double data type and location of all documents. Overall total tests done by countries are found by summing

	{$match:{totalTests:{$eq:0}}}, {$sort:{_id:1}}])	total tests of each day. Countries who have total tests equal to 0 means that they do not have any data on total tests. Countries are sorted in alphabetical order.
6	db.covid.aggregate([{$project:{_id:0, "month":{$month:{date:"$date", timezone:"+08:00"}}, "day":{$dayOfMonth:{date:"$date", timezone:"+08:00"}}, location:1, totalNewCases:{$convert:{input:"$new_cases", to:"double", onError:0.0, onNull:0.0}}}}, {$group:{_id:{groupByMonth:"$month"}, totalNewCases:{$sum:"$totalNewCases"}}}, {$sort:{totalNewCases:1}}])	The aggregate function is used where month and day of date, location and number of new cases in double data type were obtained from all documents. The sum of new cases for each month globally is obtained by using the “sum” function. The documents were sorted in ascending order for total new cases. It can be seen that the total new cases increases every month.
7	db.covid.aggregate([{$project:{_id: 0, continent: 1 , "month":{$month:{date:"$date", timezone:"+08:00"}}, newCases:{$convert:{input:"$new_cases", to:"double", onError:0.0, onNull:0.0}}}},	The aggregate function was used to obtain the continent, month and new cases in double data type. The total new cases are obtained by summing the new cases for

	{$group:{_id:{continent:"$continent", month:"$month"}, totalNewCases:{$sum:"$newCases"}}}, {$sort:{"_id.continent":1, "_id.month":1}}])	each country in each continent for each month. The results are sorted by continent and month.
8	db.response.aggregate([{$match:{$or: [{Country:"Austria"}, {Country:"Belgium"}, {Country:"Bulgaria"}, {Country:"Croatia"}, {Country:"Cyprus"}, {Country:"Denmark"}, {Country:"Estonia"}, {Country:"Finland"}, {Country:"France"}, {Country:"Germany"}, {Country:"Greece"}, {Country:"Hungary"}, {Country:"Ireland"}, {Country:"Italy"}, {Country:"Latvia"}, {Country:"Lithuania"}, {Country:"Luxembourg"}, {Country:"Malta"}, {Country:"Netherlands"}, {Country:"Poland"}, {Country:"Portugal"}, {Country:"Romania"}, {Country:"Slovakia"}, {Country:"Slovenia"}, {Country:"Spain"},	The aggregate function is used to obtain countries that are in the European Union (EU). Next, the countries which implemented responses that include “Masks” are obtained. The documents are grouped by countries to obtain unique countries.

	{Country:"Sweden"}]}}, {$match: {"Response_measure":{$regex: "Masks"}}}, {$group:{_id:"$Country"}}])	
8	db.response.updateMany({date_end: "NA"}, {$set: {date_end: "2020-08-01"}}) var countryListLength = 0 var countryList = [] var dateStartList = [] var dateEndList = [] var countryOverlapCounter = {} var obj = db.response.aggregate([{$match:{$or: [{Country:"Austria"}, {Country:"Belgium"}, {Country:"Bulgaria"}, {Country:"Croatia"}, {Country:"Cyprus"}, {Country:"Czechia"}, {Country:"Denmark"}, {Country:"Estonia"}, {Country:"Finland"}, {Country:"France"},	Aggregate function is used to obtain the start date and end date of EU countries who have implemented “MasksMandatory” measures. We will then store this data into a Javascript array. From this, we will pull the list of countries, their start date and end date into another 3 separate arrays. After which we will initialise another counter object which will then store the number of overlaps a certain country’s start date has with another country’s start and end date. Via a series of for loops, we derive the number of overlaps. Afterwhich, we can find the country who has the most number of overlaps via a for loop

	{Country:"Germany"}, {Country:"Greece"}, {Country:"Hungary"}, {Country:"Ireland"}, {Country:"Italy"}, {Country:"Latvia"}, {Country:"Lithuania"}, {Country:"Luxembourg"}, {Country:"Malta"}, {Country:"Netherlands"}, {Country:"Poland"}, {Country:"Portugal"}, {Country:"Romania"}, {Country:"Slovakia"}, {Country:"Slovenia"}, {Country:"Spain"}, {Country:"Sweden"},]}}, {$match:{Response_measure: "MasksMandatory"}}]).project({Country:1, date_start:1, date_end:1, _id:0}).toArray(); countryListLength = obj.length for (var i = 0; i < countryListLength; i++) { countryList.push(obj[i]["Country"])	iterating over the counter object. That country's start date is the start date of the period with the most number of overlaps. After which, we will find the end date of the period with the most number of overlaps by querying the countries which are part of the most overlapping period and taking the minimum end date amongst this list of countries.

```
    var temp = new
ISODate(obj[i]["date_start"].toString())

    temp = temp.getFullYear()+'-
'+(temp.getMonth()+1)+'-
'+temp.getDate();

    dateStartList.push(temp)

    temp = new
ISODate(obj[i]["date_end"].toString())

    temp = temp.getFullYear()+'-
'+(temp.getMonth()+1)+'-
'+temp.getDate();

    dateEndList.push(temp)

}


var add = 0


for (var z = 0; z < countryListLength;
z++) {


countryOverlapCounter[countryList[z]
+ add] = 0

    add++

}


for (var a = 0; a < countryListLength;
a++) {

    var country = countryList[a]

    var dateC = dateStartList[a]

    for (var b = 0; b <
countryListLength; b++) {

        var comparedc = countryList[b]

        var compareds = dateStartList[b]
        var comparede = dateEndList[b]

        if (dateC >= compareds &&
dateC <= comparede &&
comparedc != country) {

countryOverlapCounter[country+a] +=
1

        }

    }

}


var max = 0;

var maxC = "";

var change = 0;


var copy = [];


for (var i = 0; i < countryListLength;
i++) {

    copy[i] = countryList[i] + change;

    change++;

}


for (var i = 0; i < countryListLength;
i++) {

    if (countryOverlapCounter[copy[i]] >
max) {

        max =
countryOverlapCounter[copy[i]]

        maxC = copy[i]

    }
```

```
}


var index = 0;


for (var i = 0; i < countryListLength; i++) {
    if (copy[i] == maxC) {
        index = i
        break
    }
}


var temp = dateStartList[index].toString()


var findMinDateArr = []
var findMinDate = ""


for (var a = 0; a < countryListLength; a++) {


    var com1 = new Date(dateStartList[a])
    var com2 = new Date(temp)
    var com3 = new Date(dateEndList[a])


    if (com1 <= com2 && com2 <= com3) {
```

	`findMinDateArr.push(dateEndList[a])`
	`    }`
	`}`
	
	`var sizeArr = findMinDateArr.length`
	
	`findMinDate = findMinDateArr[0]`
	
	`for (var b = 0; b < sizeArr; b++) {`
	`    if (findMinDate > findMinDateArr[b])`
	`{`
	`        findMinDate = findMinDateArr[b]`
	`    }`
	`}`
	
	`var minMax = [temp, findMinDate]`
	`print(minMax)`	

9	`db.covid.aggregate([`
	`{$match:`
	`    {$and:[`
	`        {$or:[`
	`        {continent:"Europe"},`
	`        {continent:"North America"}]},`
	`       {date:{$gte:ISODate("2020-07-`
	`25T00:00:00.000+08:00")}},`
	`        {date:{$lte:ISODate("2020-08-`
	`01T00:00:00.000+08:00")}}]}},`
		Aggregate function is used to get documents from Europe and North America which fall between 2020-07-25 and 2020-08-01, the period obtained in Question 8.
		
		Date, number of new cases in double data type and location are obtained from the documents.
		
		Total new cases are obtained by

	{$project:{_id:0, date:1, newCases:{$convert:{input:"$new_cases", to:"double", onError:0.0, onNull:0.0}}, location:1}}, {$group:{_id:{date:"$date"}, total_new_cases:{$sum:"$newCases"}}}, {$sort:{"_id.date":1}}])	summing new cases of each country by dates. Lastly, the values were sorted by date.

10	function addIfConsec(array, country, final) {
	for (var i = 0; i < array.length-14; i++) {
	if ((array[i]+array[i+1]+array[i+2]+array[i+3]+array[i+4]+array[i+5]+array[i+6]+array[i+7]+array[i+8]+array[i+9]+array[i+10]+array[i+11]+array[i+12]+array[i+13]+array[i+14]) == 0) {
	final.push(country)
	break;
	}
	}
	}
	
	var q10 = [];
	var countries = []
	
	var obj = db.covid.aggregate([
	{$project:{_id:0, location:1 , date:1, total_cases:{$convert: { input:
		Function is used to calculate If there are 14 values of 0 in the array, it means that there are at least 14 days of no new cases. This country is then pushed to the array to be printed.
		
		Distinct function is used to obtain documents of all countries who have total cases more than 50.
		
		For each country that has recorded more than 50 total cases, if the new cases recorded are 0, the value 0 is pushed into the array.
		
		Function is called to check if there are more than 14 days where there are no new cases.
		
		Countries that fit the condition

	"$total_cases", to: "double", onError:0.0, onNull:0.0}}}},
	{$match:{$and:[
	 {total_cases:{$gt:50}},
	},
	{$group:{_id:"$location"}},
	{$sort:{_id:1}}
	]).toArray()
	
	for (var i = 0; i < obj.length; i++) {
	    countries.push(obj[i]["_id"])
	}
	var len = countries.length;
	
	for (var a = 0; a < len; a++) {
	    // --- country to test --- //
	    var test = countries[a];
	    var array = [];
	    var temp = db.covid.aggregate([
	        {$project:{_id:0, location:1 , date:1, total_cases:{$convert: { input: "$total_cases", to: "double", onError:0.0, onNull:0.0}}, new_cases:{$convert: { input: "$new_cases", to: "double", onError:0.0, onNull:0.0}}}},
	        {$match:{$and:[
	         {location:test},
		are printed.

	```
	            {total_cases:{$gt:50}},
	          }).toArray()
	      for (var o = 0; o < temp.length; o++)
	  {
	          if (temp[o]["new_cases"] == 0) {
	              array.push(0)
	          }
	          else {
	              array.push(1)
	          }
	      }
	      addIfConsec(array, test, q10)
	  }

	  print(q10)
	```

11	
	```
	var countries = [
	        "Andorra",
	        "Aruba",
	        "Bahamas",
	        "Barbados",
	        "Bermuda",
	        "Brunei",
	        "Cambodia",
	        "Cayman Islands",
	        "Equatorial Guinea",
	        "Faeroe Islands",
	        "French Polynesia",
	        "Gibraltar",
	        "Guernsey",
	        "Isle of Man",
	```
		We will store the results from question 10 into an array.
		
		Using a for loop which will iterate over every country identified above, we will first filter the covid collection based on the country and and perform a predefined function checkIfUptick which will return a boolean value based on if an uptick is observed from that country, i.e. total cases is more than 50 within any 7 day period
		
		Based on this, we will add

	```
	        "Liechtenstein",
	        "Mauritius",
	        "Monaco",
	        "Montenegro",
	        "New Zealand",
	        "San Marino",
	        "Sint Maarten (Dutch part)",
	        "Tanzania",
	        "Trinidad and Tobago",
	        "United States Virgin Islands",
	        "Western Sahara"
	];

	var clen = countries.length;

	var date = {}

	for (var w = 0; w < clen; w++) {
	   date[countries[w]] = ""
	}

	var dlen = date.length;

	var arr = {};

	for (var v = 0; v < clen; v++) {
	   arr[countries[v]] = 0
	}

	for (var b = 0; b < clen; b++) {
	   var tempCountry = countries[b]
	   var temp = db.covid.aggregate([
	```
		countries which observes upticks into an array

```
        {$project:{_id:0, location:1 , date:1,
total_cases:{$convert: { input:
"$total_cases", to: "double", onError:0.0,
onNull:0.0}}, new_cases:{$convert:
{ input: "$new_cases", to: "double",
onError:0.0, onNull:0.0}}}},
        {$match:{$and:[
         {location:tempCountry},
         {total_cases:{$gt:50}},
        }).toArray()
    for (var a = 0; a < temp.length; a++) {
        if (temp[a]["location"] ==
tempCountry) {
            if (temp[a]["new_cases"] == 0) {
                arr[tempCountry] += 1;
                if (arr[tempCountry] == 14) {
                    date[tempCountry] =
temp[a]["date"]
                    break;
                    }
                }
            }
            else {
                arr[tempCountry] = 0;
            }
        }
    }
}

var final = [];

function checkIfUptick(array) {
    var len = array.length
    for (var d = 0; d < len - 6; d++) {
        var temp = array[d] + array[d+1] +
array[d+2] + array[d+3] + array[d+4] +
array[d+5] + array[d+6]
        if (temp > 50) {
            return true;
        }
    }
}



for (var c = 0; c < clen; c++) {
    var theCounter = 0;
    var newArray = []
    var tempCountry = countries[c]
    var tempDate = date[tempCountry]
    var temp = db.covid.aggregate([
        {$project:{_id:0, location:1 , date:1,
total_cases:{$convert: { input:
"$total_cases", to: "double", onError:0.0,
onNull:0.0}}, new_cases:{$convert:
{ input: "$new_cases", to: "double",
onError:0.0, onNull:0.0}}}},
        {$match:{$and:[
         {location:tempCountry},
         {total_cases:{$gt:50}},
        }).toArray()

    for (var q = 0; q < temp.length; q++) {
        var conDate = temp[q]["date"]
```

	if (temp[q]["location"] ==
	tempCountry && conDate >= tempDate) {
	var value = temp[q]["new_cases"]
	newArray.push(value)
	}
	var check =
	checkIfUptick(newArray);
	if (check && theCounter == 0) {
	final.push(tempCountry)
	theCounter++
	}
	}
	}
	
	print(final)	

12	//retail and recreational db.mobility.aggregate([{$project:{_id:0, country_region:1, retail_and_recreation_percent_change_from_baseline:{$convert: { input: "$retail_and_recreation_percent_change_from_baseline", to: "double", onError: 0.0}}}}, {$group:{_id:{country:"$country_region"}, totalrr:{$sum:"$retail_and_recreation_percent_change_from_baseline"}}}, {$sort:{totalrr:-1}}, {$limit:3},	For each of the place categories (i.e. grocery and pharmacy, parks, transit stations, retail and recreation, residential, and workplace), the countries and percent changes of place categories from baseline in double data type are first obtained. Next, the percent change from baselines each day is summed according to countries. Countries are sorted according to descending order of the sum of percent change from baseline of

	{$project:{country_region:1}}
	])
	
	//grocery and pharmacy
	db.mobility.aggregate([
	{$project:{_id:0, country_region:1, grocery_and_pharmacy_percent_change_from_baseline:{$convert: { input: "$grocery_and_pharmacy_percent_change_from_baseline", to: "double", onError: 0.0}}}},
	
	{$group:{_id:{country:"$country_region"}, totalgh:{$sum:"$grocery_and_pharmacy_percent_change_from_baseline"}}},
	{$sort:{totalgh:-1}},
	
	{$limit:3},
	
	{$project:{country_region:1}}
	])
	
	//parks
	db.mobility.aggregate([
	{$project:{_id:0, country_region:1, parks_percent_change_from_baseline:{$convert: { input: "$parks_percent_change_from_baseline", to: "double", onError: 0.0}}}},
		place categories.
		
		A limit of 3 countries is used such that the top 3 countries are obtained since the countries are listed in descending order.
		
		Lastly, the top 3 countries are printed.

	{$group:{_id:{country:"$country_region"}, totalp:{$sum:"$parks_percent_change_fr om_baseline"}}}, {$sort:{totalp:-1}}, {$limit:3}, {$project:{country_region:1}}]) //transit stations db.mobility.aggregate([{$project:{_id:0, country_region:1, transit_stations_percent_change_from_b aseline:{$convert: { input: "$transit_stations_percent_change_from_ baseline", to: "double", onError: 0.0}}}}, {$group:{_id:{country:"$country_region"}, totalts:{$sum:"$transit_stations_percent_ change_from_baseline"}}}, {$sort:{totalts:-1}}, {$limit:3}, {$project:{country_region:1}}])	

	//workplace
	db.mobility.aggregate([
	{$project:{_id:0, country_region:1,
	workplaces_percent_change_from_baseli
	ne:{$convert: { input:
	"$workplaces_percent_change_from_bas
	eline", to: "double", onError: 0.0}}}},
	
	{$group:{_id:{country:"$country_region"},
	totalwp:{$sum:"$workplaces_percent_cha
	nge_from_baseline"}}},
	
	{$sort:{totalwp:-1}},
	
	{$limit:3},
	
	{$project:{country_region:1}}
	])
	
	//residential
	db.mobility.aggregate([
	{$project:{_id:0, country_region:1,
	residential_percent_change_from_baseli
	ne:{$convert: { input:
	"$residential_percent_change_from_base
	line", to: "double", onError: 0.0}}}},
	
	{$group:{_id:{country:"$country_region"},
	totalr:{$sum:"$residential_percent_chang
	e_from_baseline"}}},	

	{$sort:{totalr:-1}}, {$limit:3}, {$project:{country_region:1}}])	
13	db.covid.aggregate([{$match:{location:"Indonesia"}}, {$project:{_id:0, location:1, date:1, total_cases:{$convert: { input: "$total_cases", to: "double", onError: 0.0}}}}, {$match:{total_cases:{$gt:20000}}} {$sort:{date:1}}, {$limit:1}]) db.mobility.aggregate([{$project: {_id:0, retail_and_recreation_percent_change_from_baseline:1, workplaces_percent_change_from_baseline:1, grocery_and_pharmacy_percent_change_from_baseline:1, date:{$convert:{input:"$date", to:"date"}}}},	The aggregate function is used to obtain all documents from Indonesia. Next, country, date and total cases for each day in double data type are obtained. Next, days where total cases exceed 20000 were obtained. The documents were sorted according to date and the first date where the total cases exceed 20000 was obtained. Aggregate function is used to obtain the 3 different place categories (i.e., retail and recreation, workplaces, and grocery and pharmacy). Documents which dates are greater than the date obtained

	{$match:{"date":{$gt:ISODate("2020-05-22T00:00:00.000+08:00")}}}])	from first aggregate function are obtained.

7 References

Chan, M. (2019, 5 March). *SQL vs. NoSQL – what's the best option for your database needs?* Retrieved 14 November 2020, from https://www.thorntech.com/2019/03/sql-vs-nosql/

Singh, V. K. (2019, March 15). *SQL vs noSQL Databases.* Retrieved 10 November 2020, from https://medium.com/system-design-blog/sql-vs-nosql-databases-6896a8cb1800

Anderson, B. (2020, 18 June). *SQL vs NoSQL Databases: What's the difference?* Retrieved 15 November 2020, from https://www.ibm.com/cloud/blog/sql-vs-nosql