NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

# BC2402 Designing and Developing Databases

# The New Normal - Vaccinations and Reopening

AY 2021/22 Semester 1

Seminar Group 5, Team 4

Tutor: Mr. Chan Wai Xin


| Name: | Matriculation Number: |
|---|---|
| SHENAL DEVINDA BANDARA RAJAGURU | U2023670B |
| TAN SHU HUA, SAMANTHA | U2021180J |
| TERESA ZHANG HAN YU | U2022886C |
| TOH YI CHENG | U2010698A |
| VENKATARAMAN SIDHAARTH | U2021808J |

# 1. Executive Summary

Our project is targeted at the World Health Organisation (WHO). WHO is the United Nations agency that acts as the bridge between nations, partners and people. Its ultimate goal is for everyone in the world to attain the highest level of health. WHO spearheads global initiatives to grow worldwide health coverage, focussing on coordinating the world's response to health crises. WHO is a reliable provider of data as well, managing and maintaining a wide range of data collections related to global health and well-being. We will focus on data related to Covid-19 in particular for this project.

Our project investigates 2 types of databases, relational and non relational schema-less databases.

For the relational databases, the relationship between each attribute is elaborated using the Entity Relationship Diagram (ERD) and our data model is demonstrated through the 3rd Normal Form as well as a focus on simplicity. After this, our implementation for the relational database was shown by deployment and development of SQL queries addressing the questions given.

Next the non-relational schema-less database was implemented and the same set of queries was performed using NoSQL as well in an attempt to paint a clearer picture for comparison purposes.

We aimed to differentiate key differences and inconsistencies in design between the two implementations. Our in-depth analysis was performed with the following perspectives in mind, namely performance and speed and CAP (Consistency, Availability and Partition Tolerance) theorem of our model. Fundamentally, we intended to gather insights so that we could determine the better model between the two to the WHO. We have further evaluated and supplemented this recommendation with an eye on its efficacy and viability in terms of the WHO's needs during the Covid-19 pandemic. The shortfalls of our recommendation were assessed in an attempt to provide the WHO with an entirely holistic picture in order to help them continue building on our recommendations in the future.

In the end, our team decided that a non-relational database would be more suitable for WHO, due to the nature of the data we are dealing with.

## 2. Business Problem

The Covid-19 pandemic is something that has struck a devastating blow to almost every single aspect of our day-to-day lives ever since the first confirmed case on 31 Dec 2019. Ever since then, there have been many variants of Covid-19 identified, including the Alpha, Gamma and Delta variants which each have different transmission rates (Roberts, 2021).

At the same time, we have entered the age of data and analytics, with the use of analytics no longer limited to large corporations with deep pockets. Many companies as well as governments today base many of their strategies and decision making entirely off insights driven from data. The WHO is a key consolidator and provider of this data, especially when it comes to the Covid-19 pandemic. As part of extensive efforts to tackle the COVID-19 situation, WHO is working tirelessly with partners to develop, manufacture and deploy safe and effective vaccines. This is to ensure fair and equitable access to safe and effective vaccines, starting with the most vulnerable.

There are 2 main implications on the WHO based on the above mentioned issues.

1. Data on the pandemic is undoubtedly pivotal to an entity's survival and progress. However, too much data is a serious problem as well, as managing and going through the sheer volume coming in becomes almost an impossible chore. The WHO thus needs a database to manage this information flow.

2. Simply managing information in a database is insufficient. While the WHO is definitely one of the biggest entities in its field, it does have several large and more than competent competitors. Thus, in today's world it is essential for WHO to continue to research, innovate and create, pushing the boundaries on database technologies and implementation. This provides the perfect blend of speed, efficiency and accuracy of data to the entities who patronise it.

Databases are a collection of organised information that need to be easily accessed, managed and updated. Considering this, selecting and effectively deploying the right database in an efficient and precise manner is absolutely crucial.

With a good database, the data that comes out from it will be more accurate. In this report, we derived Herd Immunity estimation, Vaccination Drivers, and Vaccination Effects among other things from the data we got from the database. Countries will base many of their strategies and decision making off insights driven from this information, thus accurate data and good databases are of utmost importance.

There are many databases with different capabilities. This project's objective is to look into and elaborate on two key databases, the relational and non-relational database. We will be implementing the relational database first, followed by the non-relational database and to compare, we will be performing the same queries using both SQL and NoSQL.

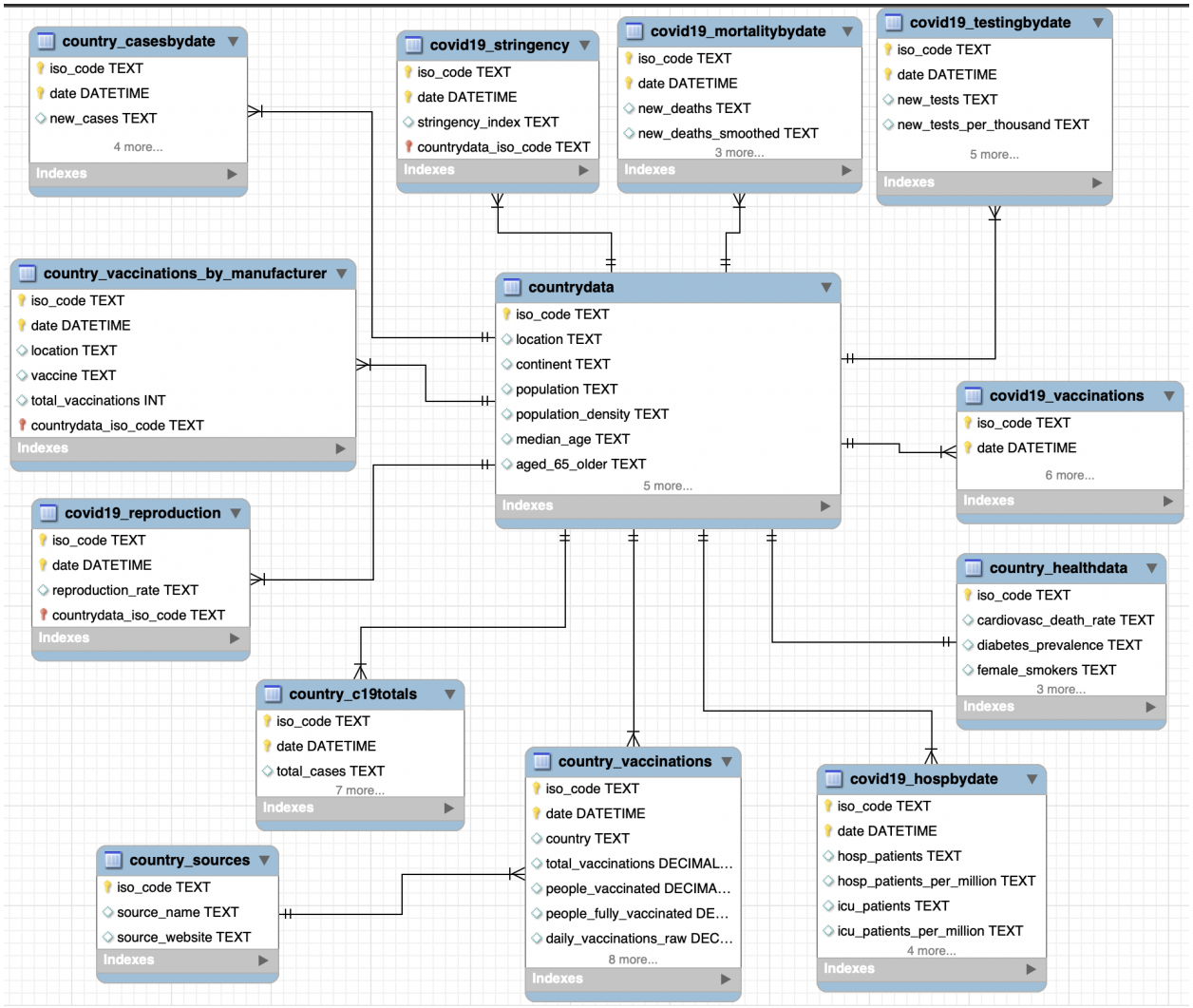
Ultimately through our analysis, we aim to siphon out the inconsistencies, pros and cons, as well as potential limitations of both implementations so as to fully understand the capabilities of each one. This will guide us in finally offering valid and suitable recommendations to the WHO.

# 3. Relational Data Model

A relational database is a type of database that uses relations to store data. It provides access to data points that are related to one another through common characteristics. To communicate with our relational database, we used Structured Query Language (SQL). SQL is a domain-specific programming language used to query and manipulate data.

## 3.1 Entity-Relationship Diagram (ERD)

In the aspect of relational database management system, we have developed an entity-relational diagram (ERD) derived from the 3 datasets provided - covid19data, country_vaccinations and country_vaccinations_by_manufacturer.



Based on the 3 datasets, they have been further decomposed into multiple entities.

The first dataset, covid19data, has been split into multiple entities which consist of static and dynamic data. Static data (e.g. population, median age, female smokers etc.) is constant and remains unchanged throughout the timeframe of data collection. We split these static data into 2 entities, countrydata and country_healthdata for easier management, placing health related data into the latter. Dynamic Covid-19 related data (e.g. new cases, total cases, total tests etc) is updated on a routine basis. We split the dynamic data into Understanding that a one-to-many relationship exists, where one country could update multiple records of Covid-19 related data while each record must be updated by only one country, many other weak entities were created. The primary key of countrydata and country_healthdata is iso_code which is the shorthand code to identify each country, while the composite primary key of the many dynamic tables include both iso_code and date. To ensure referential integrity, the foreign keys in other tables are iso_code, which is also the primary key of *countrydata.*

For the second dataset, country_vaccinations_by_manufacturer, in order to avoid update, deletion and insertion anomalies, we created a column for iso_code and matched the location column with the location column from countrydata. Essentially, the primary key of country_vaccinations_by_manufacturer is iso_code. Since the entities countrydata and country_vaccinations_by_manufacturer exist in a one-to-many relationship (i.e. One country could have many vaccines and vaccination records, while each vaccination is recorded by a single country), a composite primary key consisting of both iso_code and date exists.

In the last dataset, country_vaccinations, we extracted the 2 columns source_name and source_website into a new table with distinct iso_code and then dropped the 2 columns from country_vaccinations. This allowed us to separate the static data into country_sources.

## 3.2 Relational Database Design Considerations

We designed the relational database model based on a few fundamental concepts. Firstly, commitment and atomicity. Relational databases handle things in a very permanent, granular manner when it comes to making changes to the database.

As mentioned in the previous section, we ensured that static data will not be unnecessarily present in our tables to minimise redundancy. Since dynamic data contains records that are updated on a daily basis, they are more likely to be queried. Therefore, we separated these data into their own tables with headers that are easily identifiable. This allows the data to be more available for use as only relevant data will be present and will result in better understandability and faster querying speed.

## 3.3 Relational Schemas

Our team has determined that 3rd Normal Form (3NF) Schema is the most suitable with regards to the data provided and also for querying purposes. We have hence removed any partial and transitive dependencies in order to attain 3NF.

| Entity | Attributes |
|---|---|
| countrydata | [**iso_code**, location, continent, population, population_density, median_age, age_65_older, aged_70_older, gdp_per_capita, extreme_poverty, handwashing_facilities, human_development_index, hospital_beds_per_thousand] |
| country_healthdata | [**iso_code**, cardiovasc_death_rate, diabetes_prevalence, female_smokers, male_smokers, life_expectancy] |
| country_c19totals | [**iso_code**, **date**, total_cases, total_cases_per_million, total_deaths, total_deaths_per_million, total_tests, total_tests_per_thousand, tests_units] |
| country_casesbydate | [**iso_code**, date, new_cases, new_cases_smoothed, new_cases_per_million, new_cases smoothed_per_million] |
| covid19_mortalitybydate | [**iso_code**, **date**, new_deaths, new_deaths_smoothed, new_deaths_per_million, new_deaths_smoothed_per_million, excess_mortality] |
| covid19_hospbydate | [**iso_code**, **date**, hosp_patients, hosp_patients_per_million, icu_patients, icu_patients_per_million, weekly_hosp_admissions, weekly_hosp_admissions_per_million, weekly_icu_admissions, weekly_icu_admissions_per_million] |
| covid19_testingbydate | [**iso_code**, **date**, new_tests, new_tests_per_thousand, new_tests_smoothed, new_tests_smoothed_per_thousand, tests_per_care, positive_rate] |
| covid19_vaccinations | [**iso_code**, **date**, new_vaccinations, new_vaccinations_smoothed, new_vaccinations_smoothed_per_million, total_vaccinations, total_vaccinations_per_hundred] |
| covid19_stringency | [**iso_code**, **date**, stringency_index] |
| covid19_reproductionrate | [**iso_code**, **date**, reproduction_rate] |
| country_vaccinations | [**iso_code, date**, country, total_vaccinations, people_vaccinated, people_fully_vaccinated, daily_vaccinations_raw, daily_vaccinations, total_vaccinations_per_hundred, people_vaccinated_per_hundred, |

| | |
|---|---|
| | people_fully_vaccinated_per_hundred, daily_vaccinations_per_million, vaccines] |
| country_sources | [**iso_code**, source_name, source_website] |
| country_vaccinations_by_ manufacturer | [**iso_code**, location, date, vaccine, total_vaccinations] |

## 3.4 Assumptions

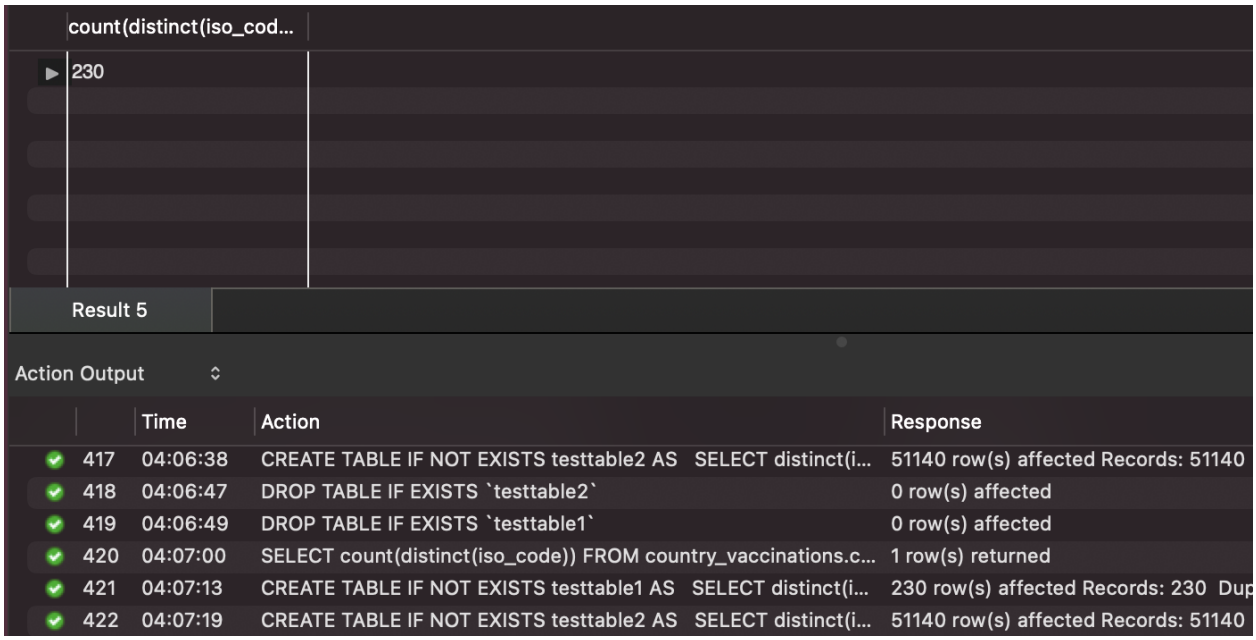
In the process of improving the initial database design, some assumptions were made.
The assumptions are as follow:

1. Each country has at least one recorded Covid-19 data.
2. Each country has one and only one recorded health data.
3. Each of the vaccinations only has one data source.
4. The population of each country did not change across the dates, thus we can use a distinct statement when creating countrydata since there is only 1 value for population for each country.

To elaborate on Assumption 4, the use of distinct(iso_code) to create tables will allow us to confirm that this assumption holds true.

```
SELECT count(distinct(iso_code))
FROM country_vaccinations.covid19data;

#Test Case
CREATE TABLE IF NOT EXISTS testtable1 AS
  SELECT distinct(iso_code), location, continent, population
  FROM country_vaccinations.covid19data;

#Test Case 2
CREATE TABLE IF NOT EXISTS testtable2 AS
  SELECT distinct(iso_code), location, continent, population, new_cases
  FROM country_vaccinations.covid19data;
```

The first code allows us to count the number of distinct countries that are in the covid19data dataset. After running the second line of code, we will obtain the number of distinct rows where the values in the rows of chosen columns are unique. If this is equal to the number obtained from the first query, it shows the population only has 1 unique value per country. The third line of code runs as a test to show what would happen when there is non-unique data. If there are more rows when adding ‘new_cases’, it shows that creating a table with the above code only returns unique data.




The above screenshot is the results obtained when the 3 codes are run. As the results show, the count for the number of distinct countries is 230, which is also the same number of rows generated when testtable1 is created. Since testtable2 has 51140 rows, it proves that population values remain unique for each country, and it does not change over the timeframe of data collected.

## 3.5 Instructions on Deployment of mySQL Database

Pre-installed Applications & Servers:

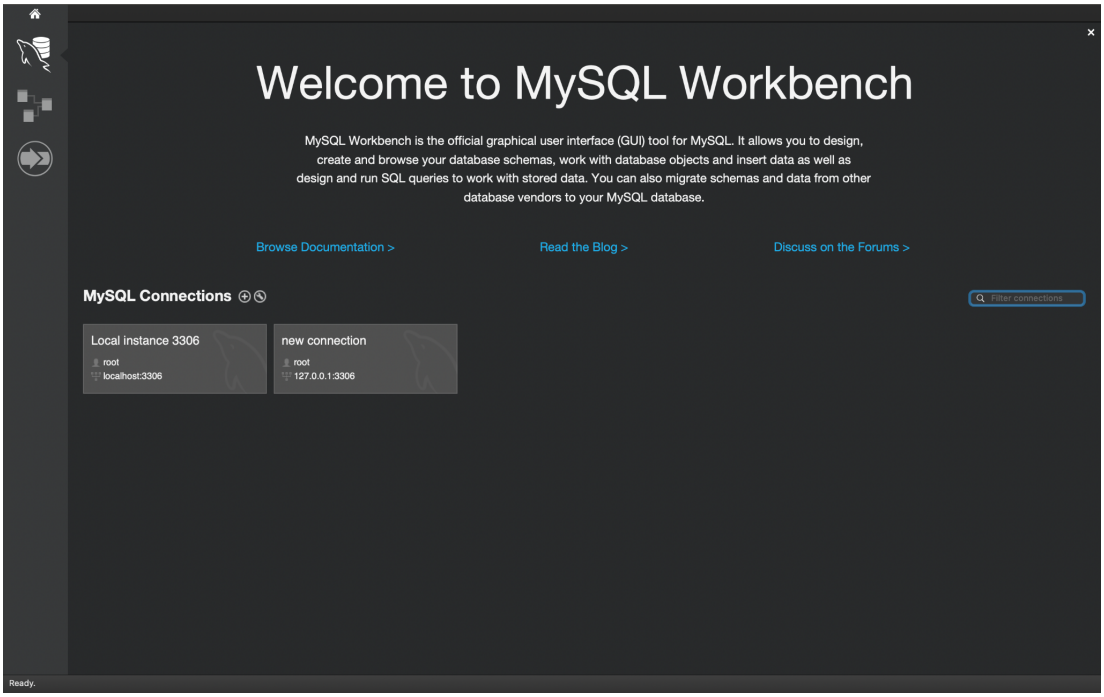
- MySQL Server
- MySQL Workbench

SQL Packages Provided:

- Database_dump.sql
- Table_creation.sql
- BC2402_S05_T04_MySQL.sql

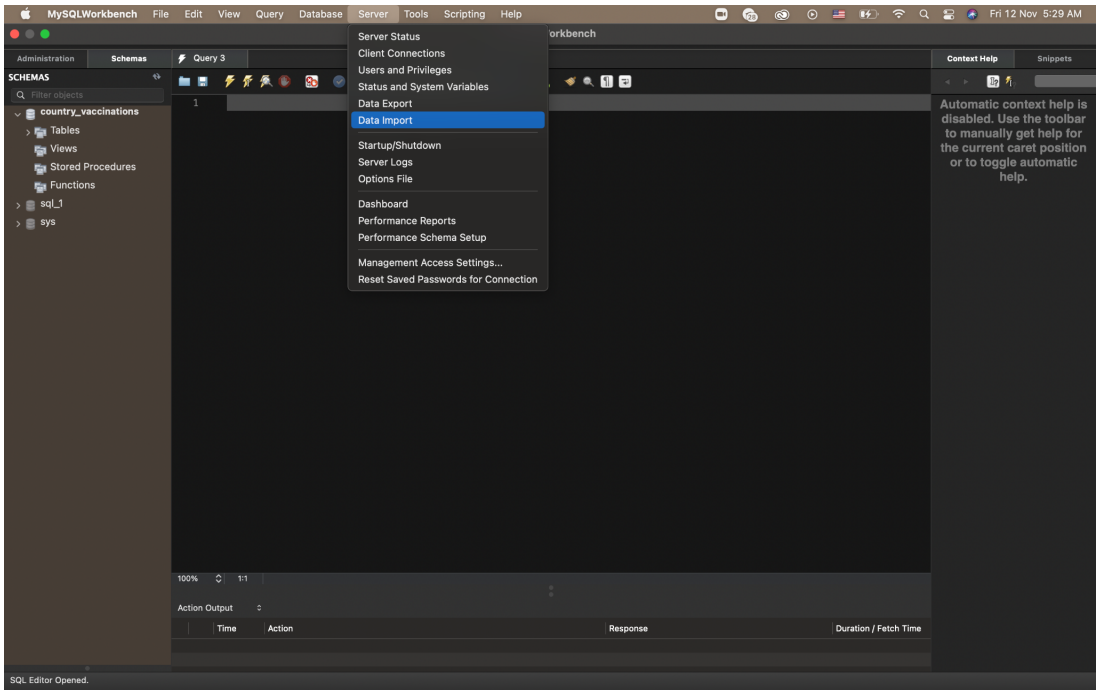
**Step 1: Booting up MySQL Workbench**

Run MySQL Workbench on the local operating system. Click on your local instance to connect to the local MySQL server.

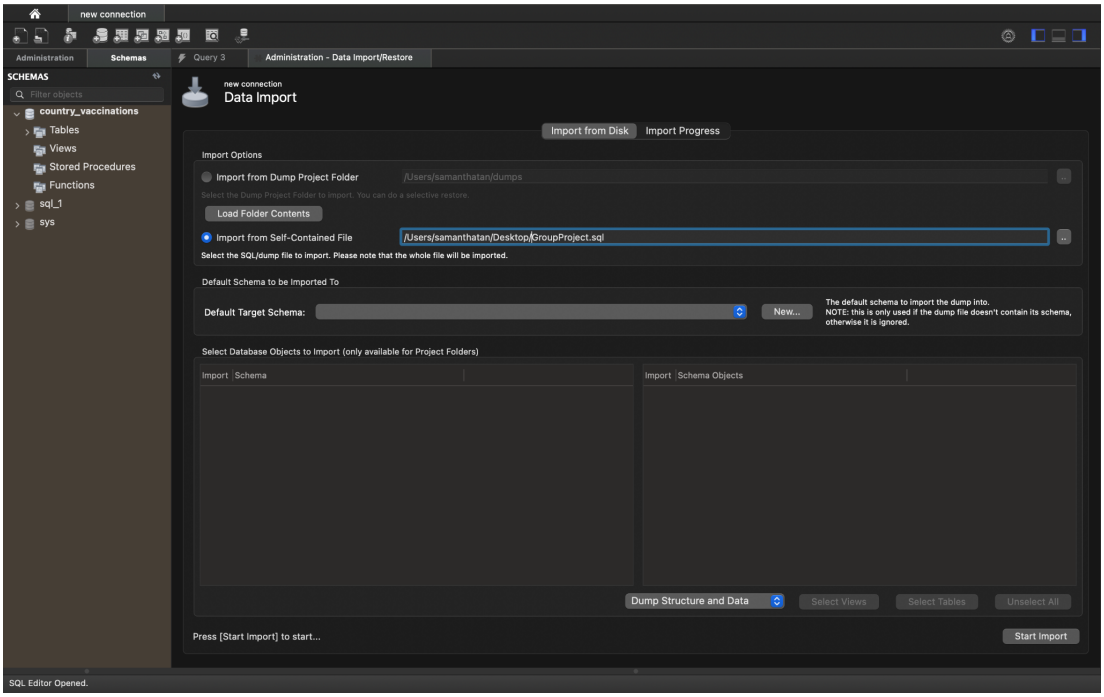



**Step 2: Importing Database Schema for Database_dump_1.sql**

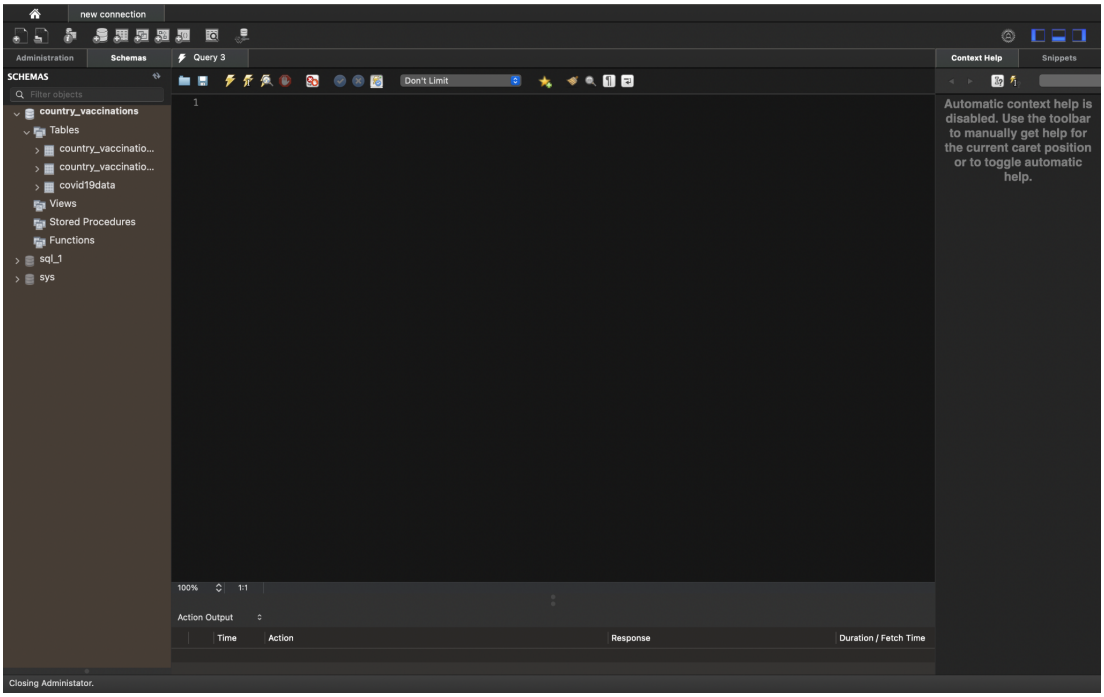
Click on the ‘Server’ Tab, followed by the ‘Data Import’ option.



Under the ‘Import from Disk’ tab, check the option ‘Import from Self-Contained File’. Choose the file ‘Database_dump.sql’ from the saved directory. Under the ‘Import Progress’ tab, click ‘Start Import’ at the bottom-right.

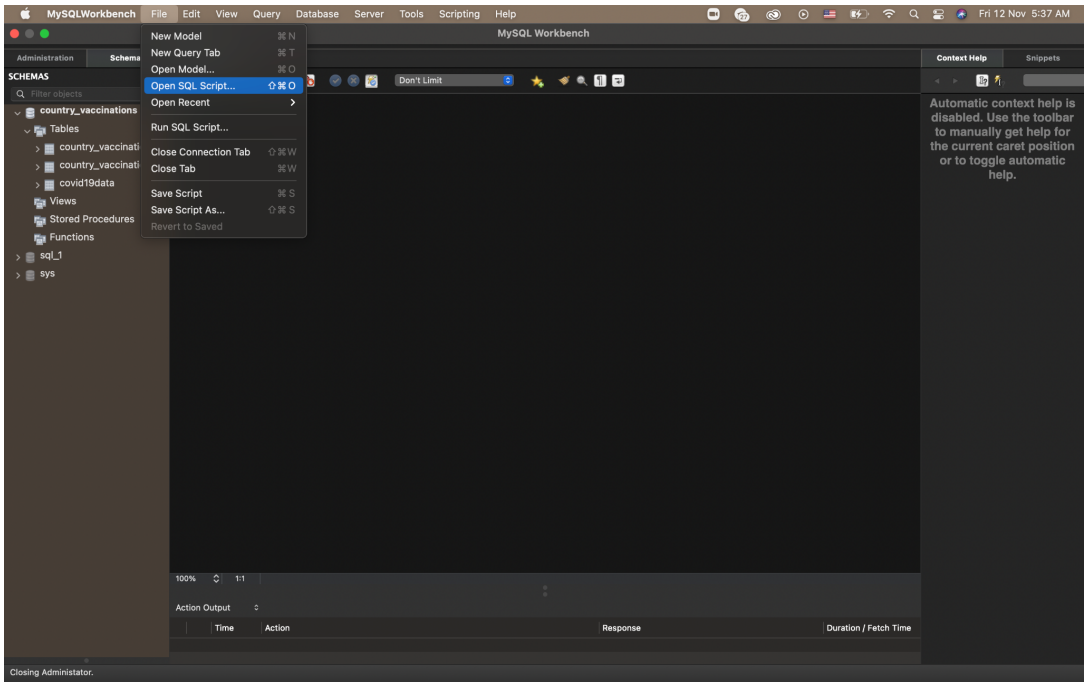



Once the import has been completed, click on the refresh icon below the ‘Navigator - Schemas’ tab to see the schema imported.

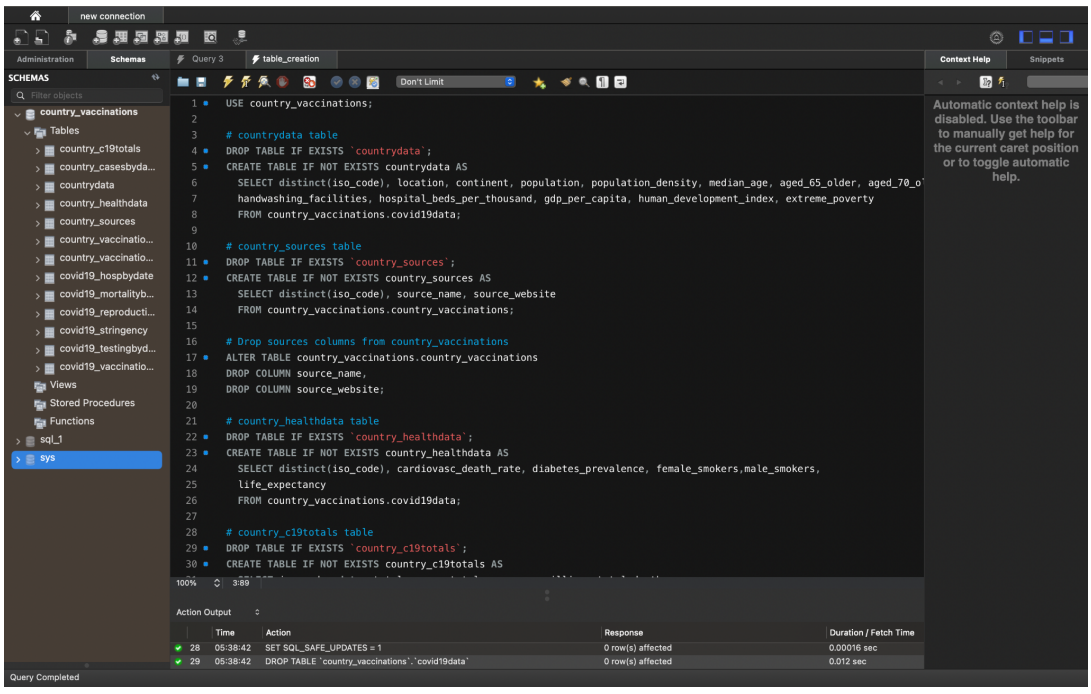


**Step 3: Deploying Normalisation from table_creation.sql**

Click on the ‘File’ Tab, followed by the ‘Open SQL Script’ option.



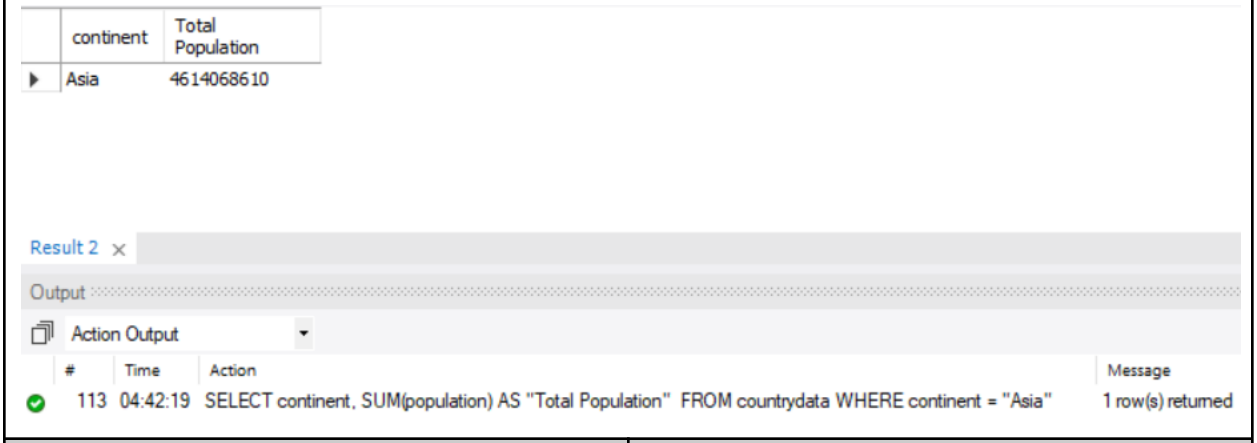

Choose the file ‘table_creation.sql’ from the saved directory. Once you run the whole script in table_creation.sql, refresh the schema again and ensure your tables are as follows.




Open BC2402_S05_T04_MySQL.sql as before and the queries can then be run by highlighting and executing the selected portion of the script.

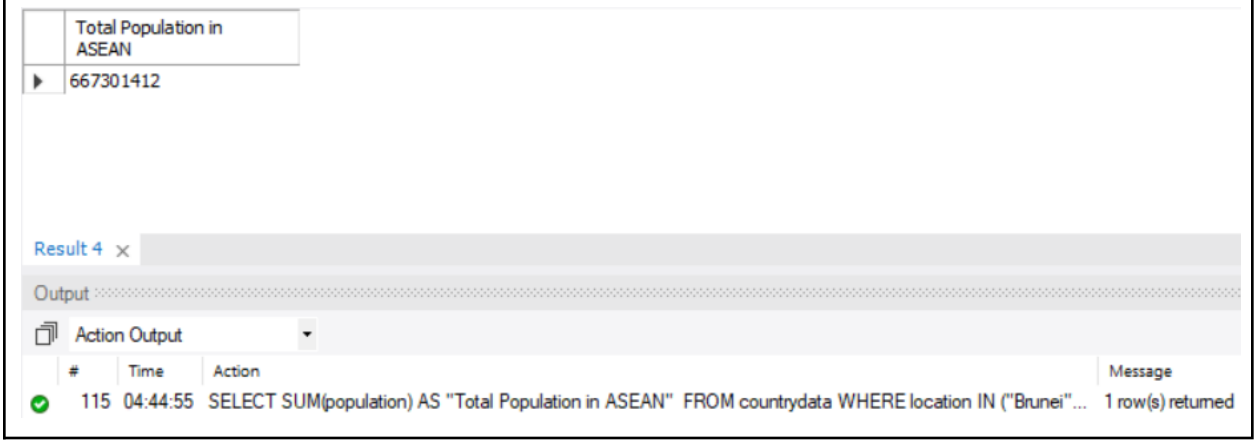
## 3.6 Query Results from Relational Database

**1. What is the total population in Asia?**




| Query | Explanation |
|---|---|
| SELECT continent, SUM(population) AS "Total Population"<br>FROM countrydata<br>WHERE continent = "Asia";<br><br>**1 Row** | To get the total Asian population we first got the latest date for each asian country from the covid19data table, then we got the respective population for each country on that date. Finally we added it all up.<br><br>After normalisation:<br>When countrydata was created, it already ensured that each country will have one population value, thus, to get the total Asian population, we can just sum up the countries whose continent is Asia. |

**2. What is the total population among the ten ASEAN countries?**

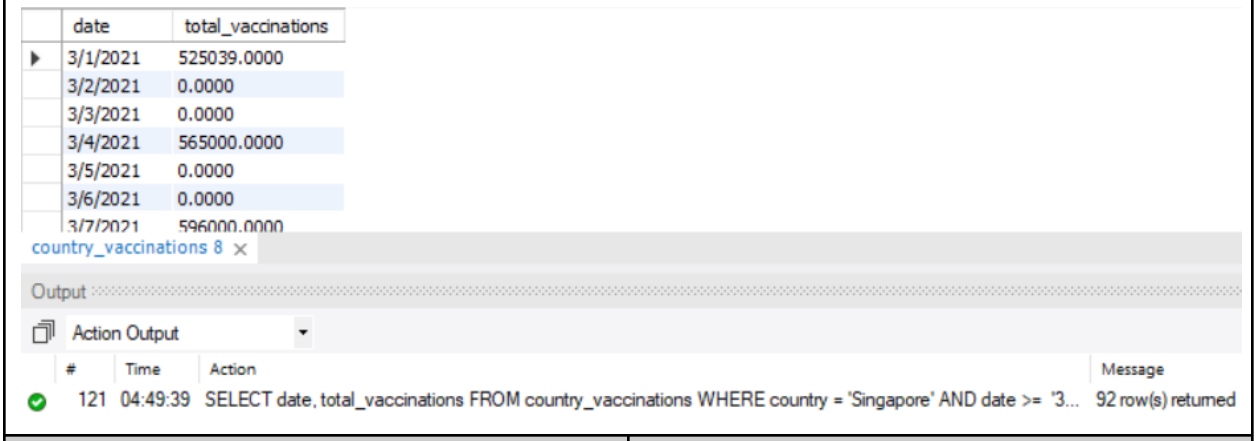


| Query | Explanation |
|---|---|
| SELECT SUM(population) AS "Total Population in ASEAN"<br>FROM countrydata<br>WHERE location IN ("Brunei", "Myanmar", "Cambodia", "Indonesia", "Laos", "Malaysia", "Philippines", "Singapore", "Thailand", "Vietnam");<br><br>**1 Row** | To get the total ASEAN population we first got the latest date for each asian country from the covid19data table, then we got the respective population for each country on that date. Finally we added it all up.<br><br><u>After normalisation:</u><br>To get the total ASEAN population, we can just sum up the countries who are in the list of countries we provided.<br><br>We assumed that the countries in the list we provided ("Brunei", "Myanmar", "Cambodia", "Indonesia", "Laos", "Malaysia", "Philippines", "Singapore", "Thailand", "Vietnam") are part of ASEAN, and that Timor-Leste is not part of ASEAN. |

**3. Generate a list of unique data sources (source_name).**

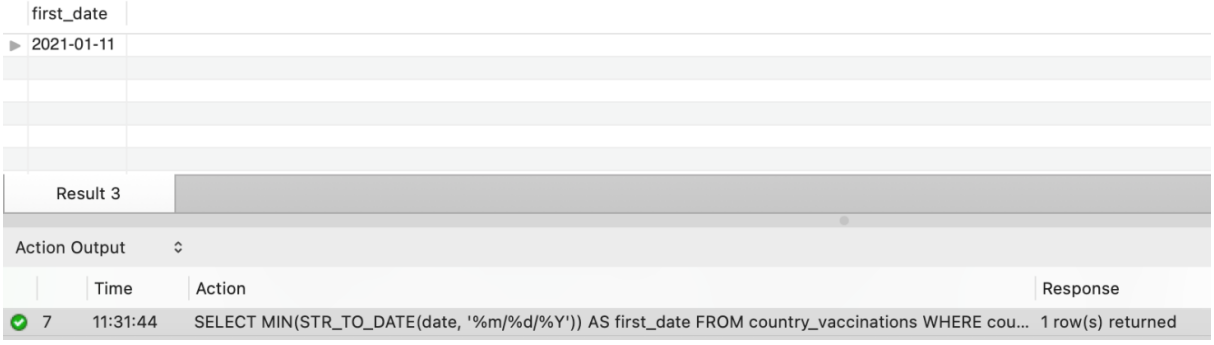

| Query | Explanation |
|---|---|
| SELECT DISTINCT source_name<br>FROM country_sources;<br><br>**92 Rows** | We selected all the distinct source_names from the country_sources table |

**4. Specific to Singapore, display the daily total_vaccinations starting (inclusive) March-1 2021 through (inclusive) May-31 2021.**




| Query | Explanation |
|---|---|
| SELECT date, total_vaccinations<br>FROM country_vaccinations<br>WHERE country = 'Singapore'<br>AND date >= '3/1/2021' AND date < '6/1/2021';<br><br>**92 Rows** | We selected the date and total_vaccinations column from country_vaccinations, specify the country as Singapore and the date as after and equal to March-1 2021 and before June-1 2021 so that the entire May-31 2021 will be included. |

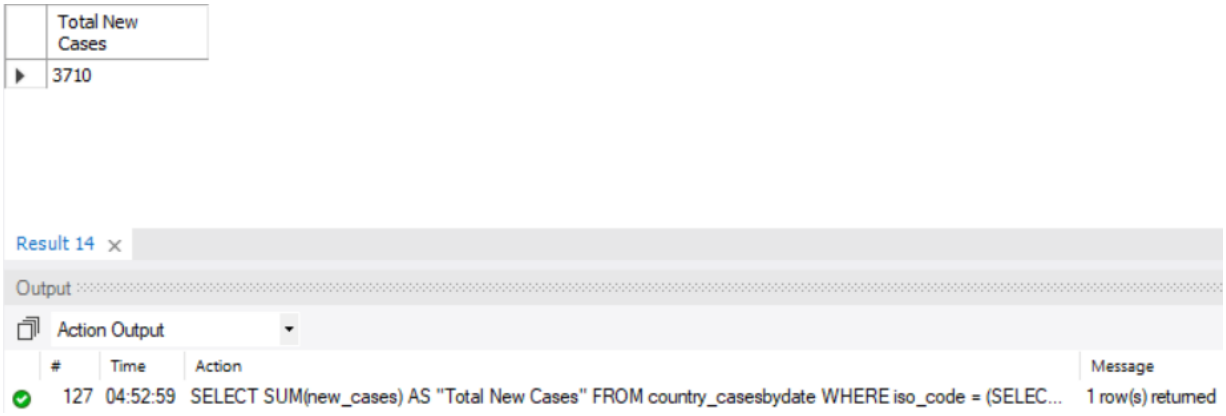
**5. When is the first batch of vaccinations recorded in Singapore?**




| Query | Explanation |
|---|---|
| SELECT MIN(STR_TO_DATE(date, '%m/%d/%Y')) AS first_date FROM country_vaccinations<br>WHERE country = "Singapore"<br>AND total_vaccinations>0;<br><br>**1 Row** | To find the first batch of vaccination, we look for the earliest date, by using min(date), where total_vaccinations is more than 0. We also specified the country as Singapore.<br><br>We converted the date from string to date format so that we are able to effectively use |

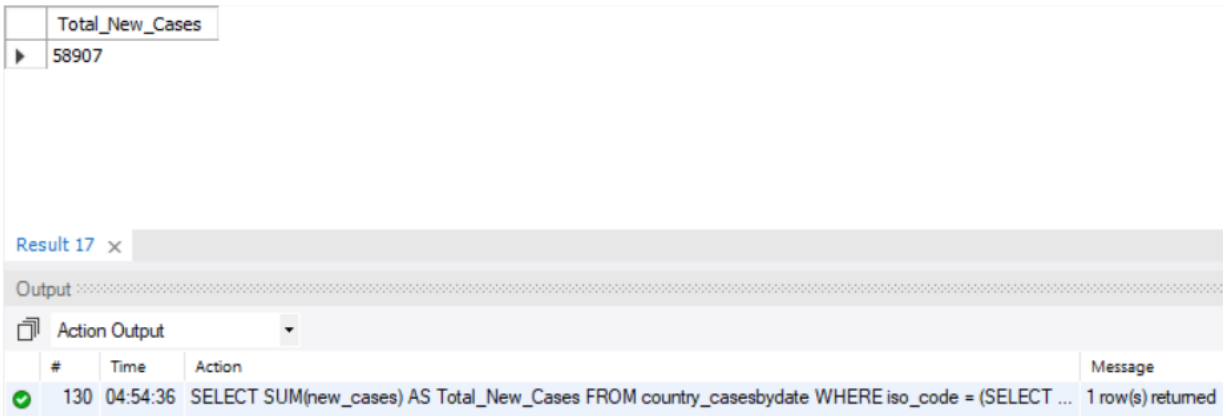
| | |
|---|---|
| | the logical operators in question 6 and 7.<br><br>We assumed that when total_vaccination is 0, there were no vaccinations at all, even prior to the first recorded date. Thus, the first date where total_vaccination becomes a non-zero value is the first batch of vaccination. |

**6. Based on the date identified in (5), specific to Singapore, compute the total number of new cases thereafter. For instance, if the date identified in (5) is Jan-1 2021, the total number of new cases will be the sum of new cases starting from (inclusive) Jan-1 to the last date in the dataset.**



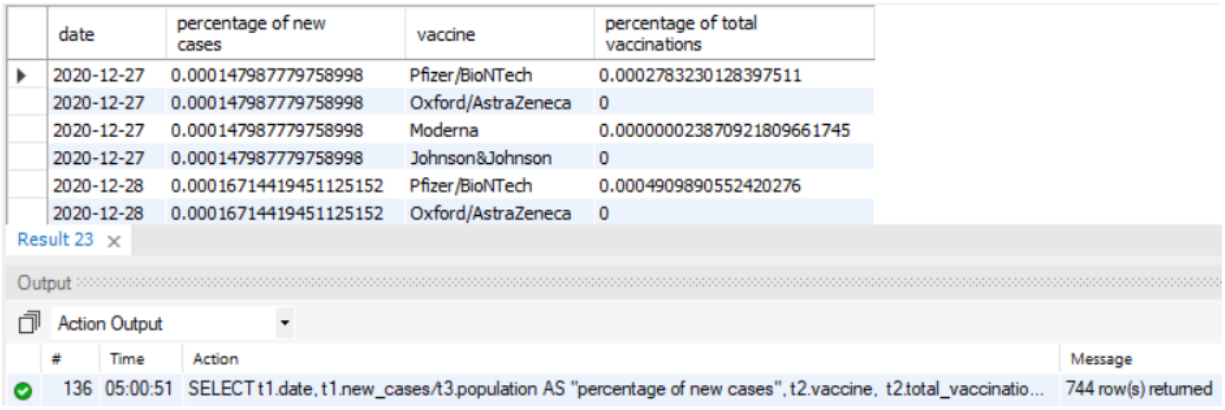

| Query | Explanation |
|---|---|
| SELECT SUM(new_cases) AS "Total New Cases" FROM country_casesbydate<br>WHERE iso_code = (SELECT iso_code FROM countrydata WHERE location = "Singapore")<br>AND date >= (SELECT MIN(STR_TO_DATE(date, '%m/%d/%Y')) AS first_date FROM country_vaccinations<br>WHERE country = "Singapore"<br>AND total_vaccinations>0);<br><br>**1 Row** | Since the question wants the number of new cases starting from the date identified in question 5 to the last date in the dataset, we find all dates after and equal to it. To find that date, we put the query in question 5 as a subquery for this question. We sum the new_cases from country_casesbydate. Since location is not in country_casesbydate, we get iso_code of the location “Singapore” by selecting it from countrydata. |

**7. Compute the total number of new cases in Singapore before the date identified in (5). For instance, if the date identified in (5) is Jan-1 2021 and the first date recorded (in Singapore) in the dataset is Feb-1 2020, the total number of new cases will be the sum of new cases starting from (inclusive) Feb-1 2020 through (inclusive) Dec-31 2020.**




| Query | Explanation |
|---|---|
| SELECT SUM(new_cases) AS Total_New_Cases FROM country_casesbydate<br>WHERE iso_code = (SELECT iso_code FROM countrydata WHERE location = "Singapore")<br>AND date <<br>(SELECT MIN(STR_TO_DATE(date, '%m/%d/%Y')) AS first_date FROM country_vaccinations<br>WHERE country = "Singapore"<br>AND total_vaccinations>0);<br><br>**1 Row** | Since the question wants the number of new cases before the date identified in question 5, we find all dates before it. To find that date, we put the query in question 5 as a subquery for this question. We sum the new_cases from country_casesbydate. Since location is not in country_casesbydate, we get iso_code of the location “Singapore” by selecting it from countrydata. |

**8. Herd immunity estimation. On a daily basis, specific to Germany, calculate the percentage of new cases and total vaccinations on each available vaccine in relation to its population.**



| date | percentage of new cases | vaccine | percentage of total vaccinations |
|---|---|---|---|
| 2020-12-27 | 0.0001479877797589 98 | Pfizer/BioNTech | 0.0002783230128397511 |
| 2020-12-27 | 0.0001479877797589 98 | Oxford/AstraZeneca | 0 |
| 2020-12-27 | 0.0001479877797589 98 | Moderna | 0.000000023870921809661745 |
| 2020-12-27 | 0.0001479877797589 98 | Johnson&Johnson | 0 |
| 2020-12-28 | 0.0001671441945112 5152 | Pfizer/BioNTech | 0.0004909890552420276 |
| 2020-12-28 | 0.0001671441945112 5152 | Oxford/AstraZeneca | 0 |

Result 23

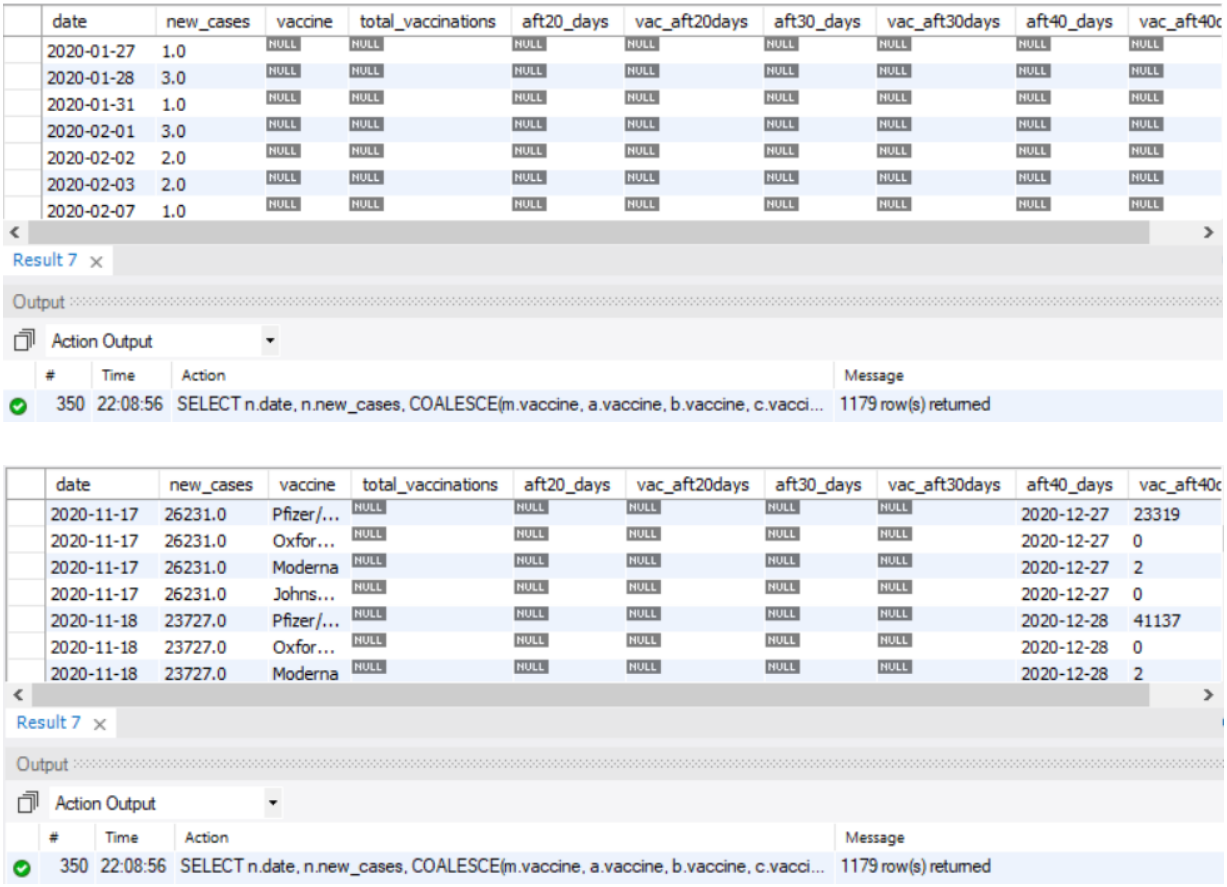
Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 136 | 05:00:51 | SELECT t1.date, t1.new_cases/t3.population AS "percentage of new cases", t2.vaccine, t2.total_vaccinatio... | 744 row(s) returned |

| Query | Explanation |
|---|---|
| SELECT t1.date, t1.new_cases/t3.population AS "percentage of new cases", t2.vaccine, t2.total_vaccinations/t3.population AS "percentage of total vaccinations"<br>FROM<br>(<br>SELECT new_cases, date FROM country_casesbydate<br>WHERE iso_code = (SELECT iso_code FROM countrydata WHERE location = "Germany")<br>) t1<br>INNER JOIN<br>(<br>SELECT vaccine, total_vaccinations, date<br>FROM country_vaccinations_by_manufacturer<br>WHERE location = "Germany"<br>) t2<br>ON t1.date = t2.date,<br>(SELECT population FROM countrydata WHERE location = "Germany") t3;<br><br>**744 Rows** | For this question, we need four values - date from either country_casesbydate or country_vaccinations_by_manufacturer as the question stated “on a daily basis", new_cases from country_casesbydate, total_vaccinations from country_vaccinations_by_manufacturer, and population from countrydata as these are needed for the calculations. Three different tables are needed, thus we created three subqueries to get the values from the three tables. As usual, since location is not in country_casesbydate, we get iso_code of the location “Germany” by selecting it from countrydata. |

**9. Vaccination Drivers. Specific to Germany, based on each daily new case, display the total vaccinations of each available vaccines after 20 days, 30 days, and 40 days.**



| date | new_cases | vaccine | total_vaccinations | aft20_days | vac_aft20days | aft30_days | vac_aft30days | aft40_days | vac_aft40c |
|---|---|---|---|---|---|---|---|---|---|
| 2020-01-27 | 1.0 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 2020-01-28 | 3.0 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 2020-01-31 | 1.0 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 2020-02-01 | 3.0 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 2020-02-02 | 2.0 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 2020-02-03 | 2.0 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 2020-02-07 | 1.0 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Result 7

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 350 | 22:08:56 | SELECT n.date, n.new_cases, COALESCE(m.vaccine, a.vaccine, b.vaccine, c.vacci... | 1179 row(s) returned |

| date | new_cases | vaccine | total_vaccinations | aft20_days | vac_aft20days | aft30_days | vac_aft30days | aft40_days | vac_aft40c |
|---|---|---|---|---|---|---|---|---|---|
| 2020-11-17 | 26231.0 | Pfizer/... | NULL | NULL | NULL | NULL | NULL | 2020-12-27 | 23319 |
| 2020-11-17 | 26231.0 | Oxfor... | NULL | NULL | NULL | NULL | NULL | 2020-12-27 | 0 |
| 2020-11-17 | 26231.0 | Moderna | NULL | NULL | NULL | NULL | NULL | 2020-12-27 | 2 |
| 2020-11-17 | 26231.0 | Johns... | NULL | NULL | NULL | NULL | NULL | 2020-12-27 | 0 |
| 2020-11-18 | 23727.0 | Pfizer/... | NULL | NULL | NULL | NULL | NULL | 2020-12-28 | 41137 |
| 2020-11-18 | 23727.0 | Oxfor... | NULL | NULL | NULL | NULL | NULL | 2020-12-28 | 0 |
| 2020-11-18 | 23727.0 | Moderna | NULL | NULL | NULL | NULL | NULL | 2020-12-28 | 2 |

Result 7

Output

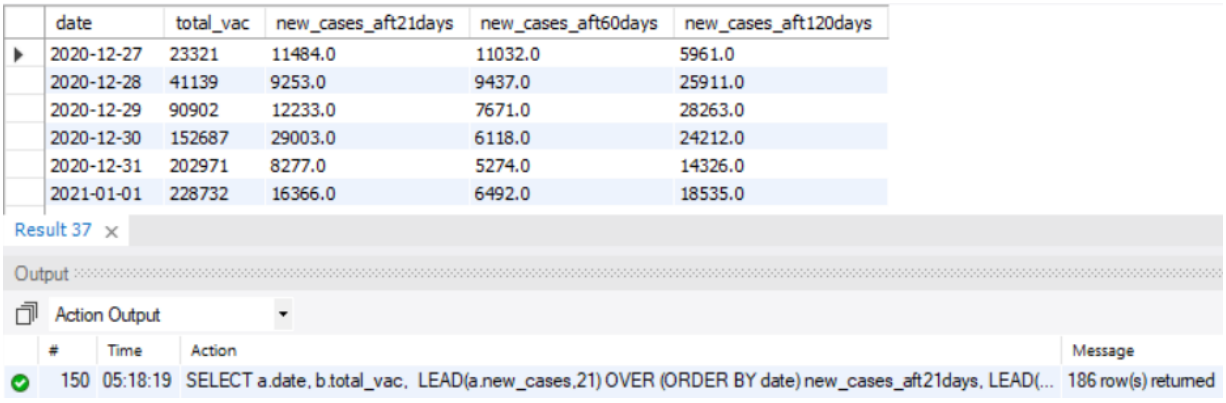
Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 350 | 22:08:56 | SELECT n.date, n.new_cases, COALESCE(m.vaccine, a.vaccine, b.vaccine, c.vacci... | 1179 row(s) returned |

| Query | Explanation |
|---|---|
| SELECT n.date, n.new_cases,<br>COALESCE(m.vaccine, a.vaccine, b.vaccine, c.vaccine) AS vaccine,<br>m.total_vaccinations,<br>a.date AS aft20_days, a.total_vaccinations AS vac_aft20days,<br>b.date AS aft30_days, b.total_vaccinations AS vac_aft30days,<br>c.date AS aft40_days, c.total_vaccinations AS vac_aft40days<br>FROM country_casesbydate AS n<br>LEFT JOIN country_vaccinations_by_manufacturer AS m<br>ON n.date = m.date AND n.iso_code = m.iso_code<br>LEFT JOIN country_vaccinations_by_manufacturer AS c<br>ON date_add(n.date, INTERVAL 40 DAY) = | For this question, 2 original tables are involved, country_casesbydate and country_vaccinations_by_manufacturer. However since there are additional requirements (total vaccinations after 20,30 and 40 days) we need to perform additional joins from country_casesbydate to country_vaccination_by_manufacturer based on a different shifted date accordingly for each of the specified dates (20,30 and 40).<br><br>In the end, there are 5 tables:<br>1. country_casesbydate as n(date new_cases are selected from this)<br>2. country_vaccinations_by_manufacturer as m (this is the first and most basic join, the two tables are joined on date then iso_code) |

| | |
|---|---|
| c.date<br>AND c.iso_code = n.iso_code AND c.vaccine = COALESCE(m.vaccine, c.vaccine)<br>LEFT JOIN<br>country_vaccinations_by_manufacturer AS b<br>ON date_add(n.date, INTERVAL 30 DAY) = b.date AND b.iso_code = n.iso_code<br>AND b.vaccine = COALESCE(m.vaccine, c.vaccine)<br>LEFT JOIN<br>country_vaccinations_by_manufacturer AS a<br>ON date_add(n.date, INTERVAL 20 DAY) = a.date AND a.iso_code = n.iso_code<br>AND a.vaccine = COALESCE(m.vaccine, c.vaccine)<br>WHERE new_cases>0 AND n.iso_code = (SELECT iso_code FROM countrydata WHERE location = "Germany")<br>ORDER BY n.date;<br><br>**1179 Rows** | 3. Country_vaccinations_by_manufacturer as c (the next join, performed to get the date and total vaccinations after 40 days) joined on c's date against n's date 40 days later, as well as the vaccine and iso codes<br>4. Country_vaccinations_by_manufacturer as b (the next join, performed to get the date and total vaccinations after 30 days) joined on b's date against n's date 30 days later, as well as the vaccine and iso codes<br>5. Country_vaccinations_by_manufacturer as a (the next join, performed to get the date and total vaccinations after 20 days) joined on a's date against n's date 20 days later, as well as the vaccine and iso codes<br><br>Left join is used as the country_vaccinations_by_manufacturer table started from a later date but we want the data all the way from the first instance of new_cases more than 0, so we need to keep those rows from the country_casesbydate table. The function COALESCE is used because of the same reason as above, to preserve the earlier data that we need. This function evaluates the arguments provided and returns the non-null values, which is essential for all our joins based on vaccines. The earlier dates don't exist in the country_vaccinations_by_manufacturer table and without this function, joining based on something that does not exist will just return NULL again.<br>In the end the overall query is once again filtered using the WHERE condition, based on Germany's iso code and new_cases > 0. |

**10. Vaccination Effects. Specific to Germany, on a daily basis, based on the total number of accumulated vaccinations (sum of total_vaccinations of each vaccine in a day), generate the daily new cases after 21 days, 60 days, and 120 days.**



| date | total_vac | new_cases_aft21days | new_cases_aft60days | new_cases_aft120days |
|---|---|---|---|---|
| 2020-12-27 | 23321 | 11484.0 | 11032.0 | 5961.0 |
| 2020-12-28 | 41139 | 9253.0 | 9437.0 | 25911.0 |
| 2020-12-29 | 90902 | 12233.0 | 7671.0 | 28263.0 |
| 2020-12-30 | 152687 | 29003.0 | 6118.0 | 24212.0 |
| 2020-12-31 | 202971 | 8277.0 | 5274.0 | 14326.0 |
| 2021-01-01 | 228732 | 16366.0 | 6492.0 | 18535.0 |

Result 37

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 150 | 05:18:19 | SELECT a.date, b.total_vac, LEAD(a.new_cases,21) OVER (ORDER BY date) new_cases_aft21days, LEAD(... | 186 row(s) returned |

| Query | Explanation |
|---|---|
| SELECT a.date, b.total_vac,<br>LEAD(a.new_cases,21) OVER (ORDER BY date) new_cases_aft21days,<br>LEAD(a.new_cases,60) OVER (ORDER BY date) new_cases_aft60days,<br>LEAD(a.new_cases,120) OVER (ORDER BY date) new_cases_aft120days<br>FROM country_casesbydate a,<br>(<br>SELECT location, date,sum(total_vaccinations) total_vac<br>FROM country_vaccinations_by_manufacturer<br>WHERE location = 'Germany'<br>GROUP BY date<br>) b<br>WHERE iso_code = (SELECT iso_code FROM countrydata WHERE location = "Germany")<br>AND a.date = b.date;<br><br>**186 Rows** | We got the sum of total vaccinations for the four vaccines in Germany by date first in subquery b. After that, we did another query and used the lead function to generate the new cases after 21 days, 60 days and 120 days. The lead function gets data by going down the specified number of rows after ordering by the column specified. This query was performed on two tables, the subquery b generated above and the country_casesbydate table. The two tables were inner joined based on date, and the overall query was filtered by iso code for Germany. |

# 4. Non-relational Data Model

Non-relational databases are different from traditional relational databases in that they store their data in a non-tabular form. Instead, non-relational databases might be based on data structures like documents. We used MongoDB, which is a document-oriented noSQL database used for high volume data storage.

## 4.1 Non-relational Database Collections

Collection 1 (**countryVac**)

```
{
  "_id": {
    "$oid": ""
  },
  "country":"",
  "iso_code": "",
  "date": "",
  "total_vaccinations": "",
  "people_vaccinated": "",
  "people_fully_vaccinated": "",
  "daily_vaccinations_raw": “",
  "daily_vaccinations": "",
  "total_vaccinations_per_hundred": "",
  "people_vaccinated_per_hundred": "",
  "people_fully_vaccinated_per_hundred": "",
  "daily_vaccinations_per_million": "",
  "vaccines": "",
  "source_name": "",
  "source_website": ""
}
```

Collection 2 (**countryVacManu**)

```
{
  "_id": {
    "$oid": ""
  } ,
  "location": "",
  "date": "",
  "vaccine": "",
  "total_vaccinations": ""
}
```

Collection 3 (**covid19data**)

```
{
  "_id": {
    "$oid": ""
  },
```

```
  "iso_code": "",
  "continent": "",
  "location": "",
  "date": "",
  "total_cases": "",
  "new_cases": "",
  "total_cases_per_million": "",
  "new_cases_per_million": "",
  "stringency_index": "",
  "population": “",
  "population_density": "",
  "median_age": "",
  "aged_65_older": "",
  "aged_70_older": "",
  "gdp_per_capita": "",
  "cardiovasc_death_rate": "",
  "diabetes_prevalence": "",
  "handwashing_facilities": "",
  "hospital_beds_per_thousand": "",
  "life_expectancy": "",
  "human_development_index": ""
}
```

## 4.2 Indexing

We created indexes for the “date” attribute in both our covid19data and countryVacManu collections to make the execution of queries in MongoDB more efficient. Without indexes, MongoDB must perform a scan throughout every document in each collection to select those documents that match the query statement. After indexes are created, the number of documents to be inspected will be limited, hence significantly decreasing runtime of our queries.

We implemented this in Question 19 and 20. An example showing the significant reduction in runtime is shown below:

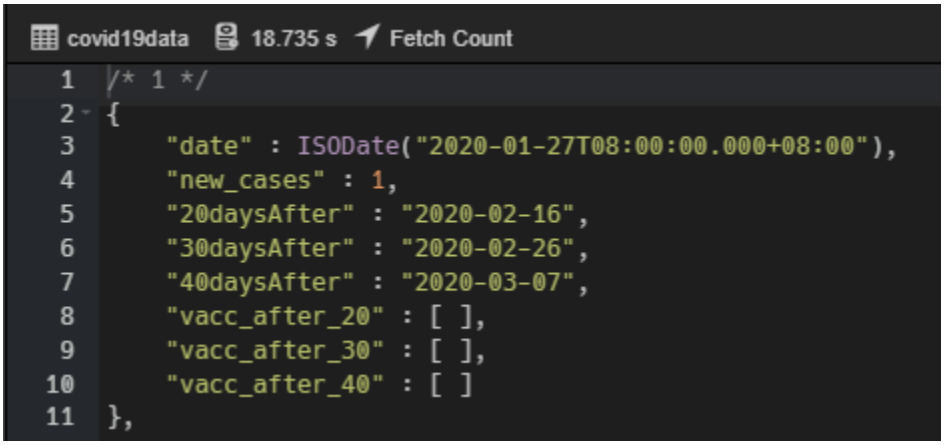



*Figure 4.1 Runtime of Question 19 before Index Implementation*

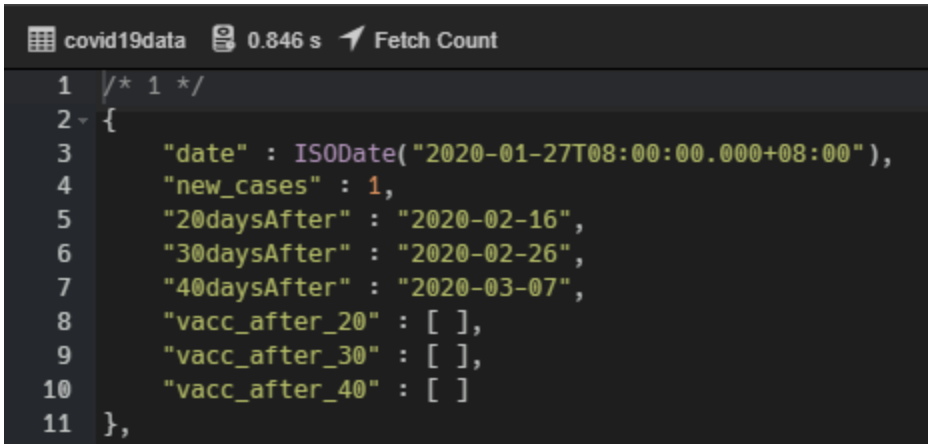



*Figure 4.2 Runtime of Question 19 after Index Implementation*

## 4.3 Instructions on Deployment of NoSQL Database

We have 3 main collections, **country_vaccinations** and **country_vaccinations_by_manufacturer** from Kaggle, and **covid19data** from our world in data, which is consolidated from a variety of sources such as the COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU) and constantly updated throughout the course of the Covid-19 pandemic. We did not make any modifications to these collections as we felt that they were already appropriate enough to match the situation and easy to write understandable and effective queries based on.

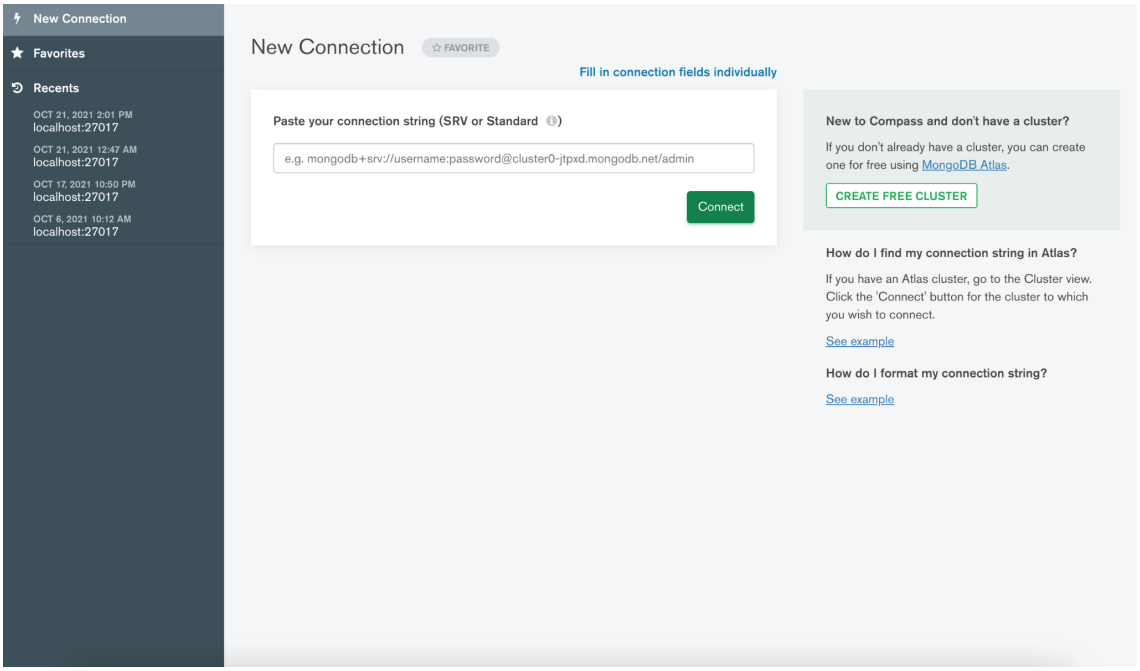
In order to implement the non-relational database, the following applications need to be pre-installed.

- No-SQLBooster for MongoDB
- MongoDB Compass Community

These are the JSON and JavaScript packages provided.

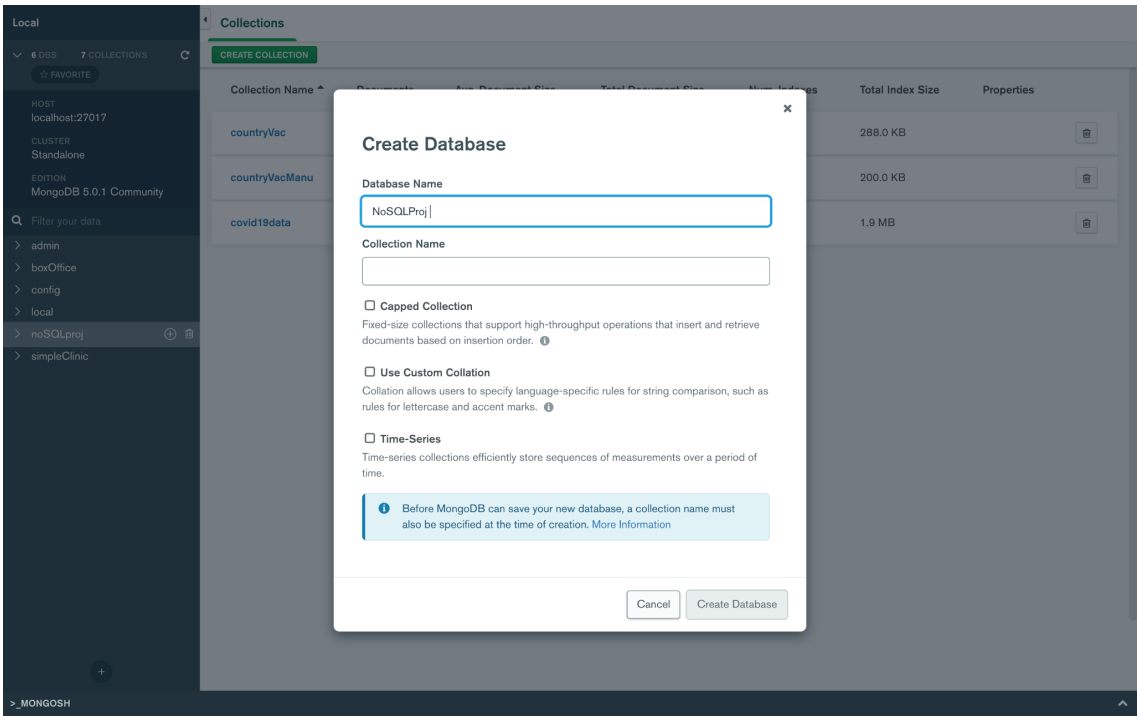
- covid19data_2.json
- country_vaccinations.json
- country_vaccinations_by_manufacturer.json
- BC2402_S05_T04_noSQL.js

## Step 1: Booting up MongoDB Compass Community



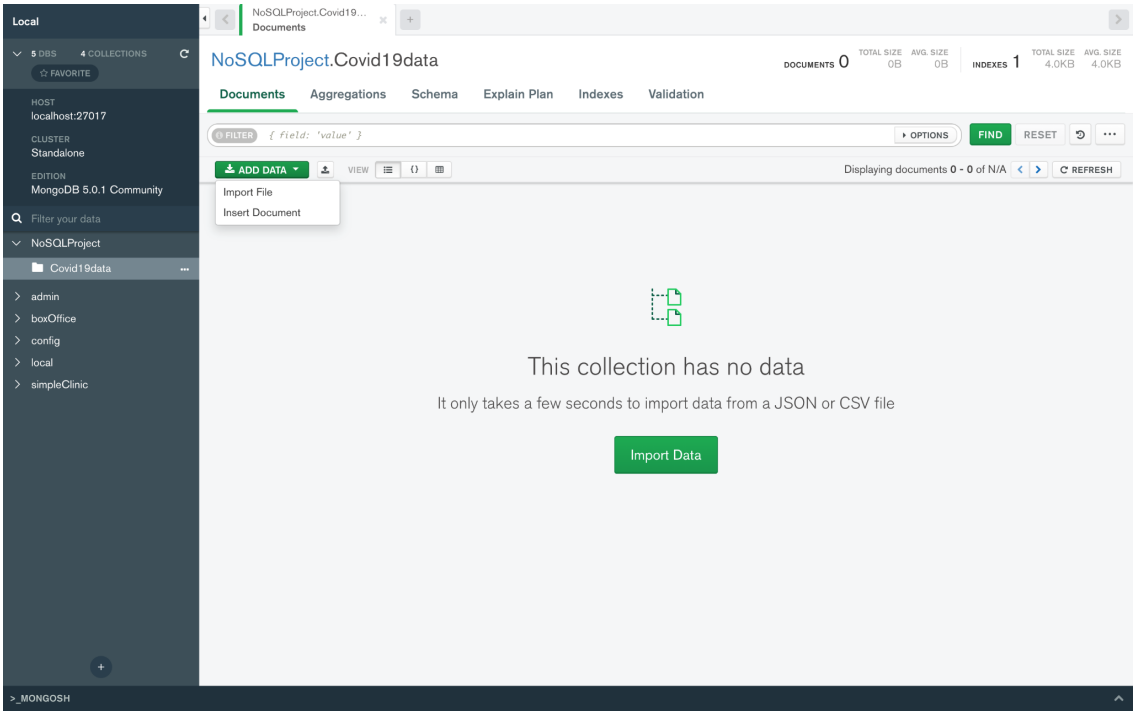

Run MongoDB and Compass Community and click on 'Connect' to connect to the local MongoDB Compass server.

## Step 2: Creating New Database

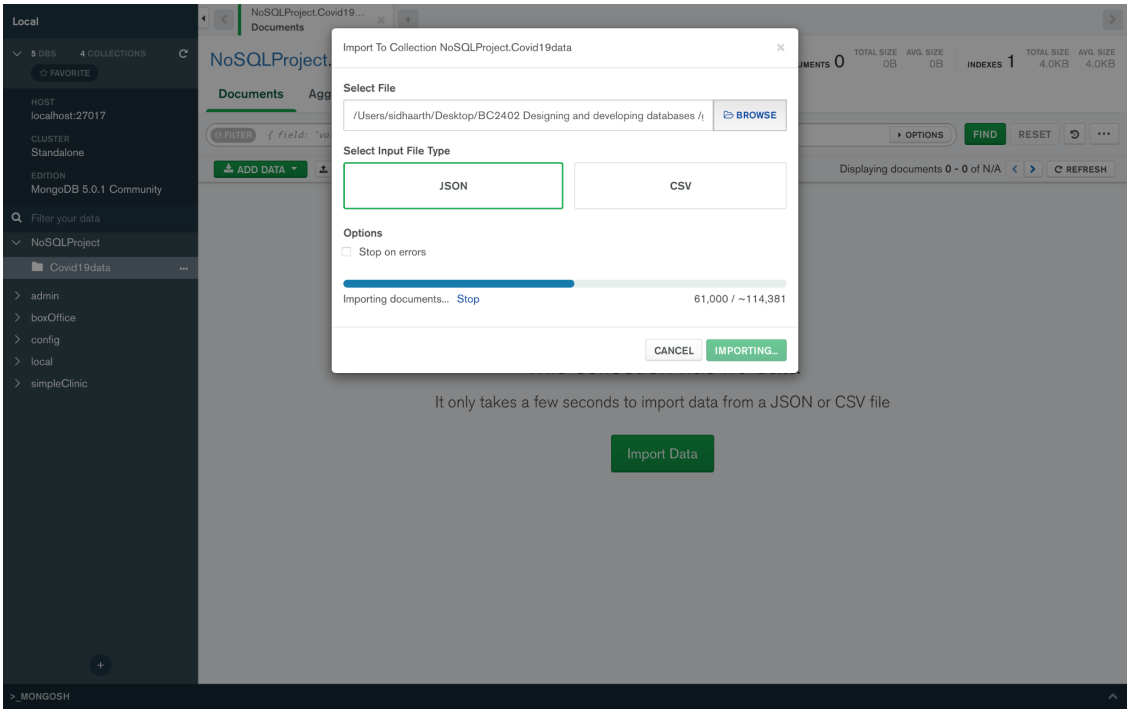



Enter NoSQLProject into the Database Name field.

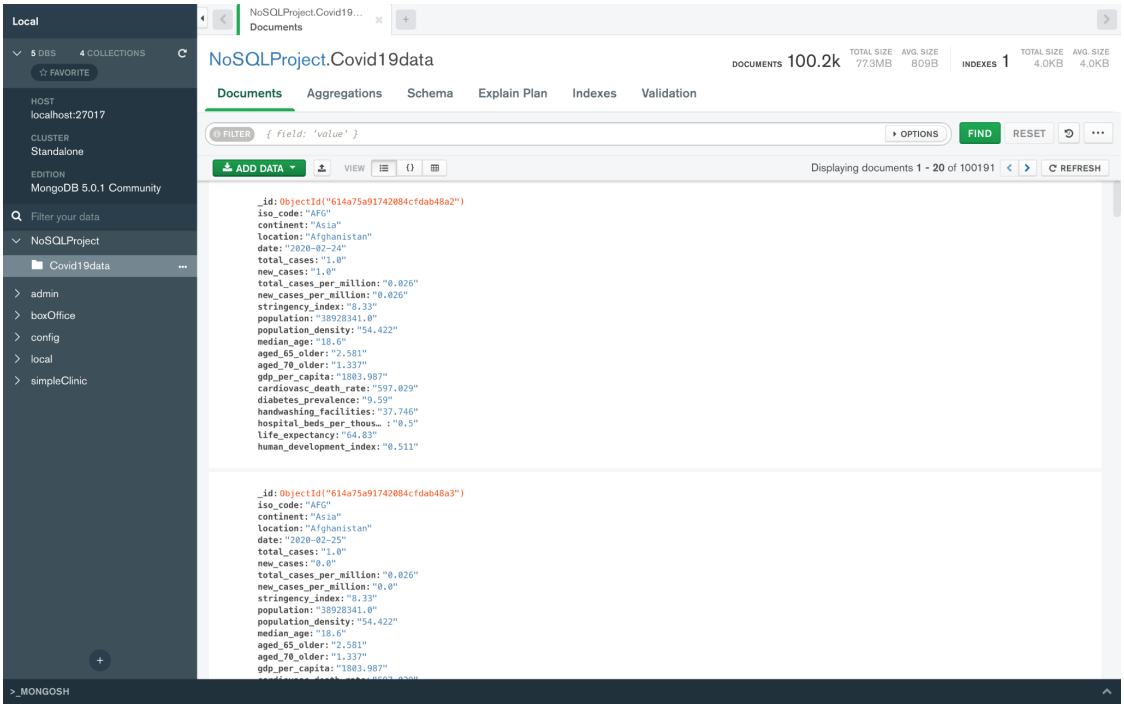
## Step 3: Importing the 3 Collections



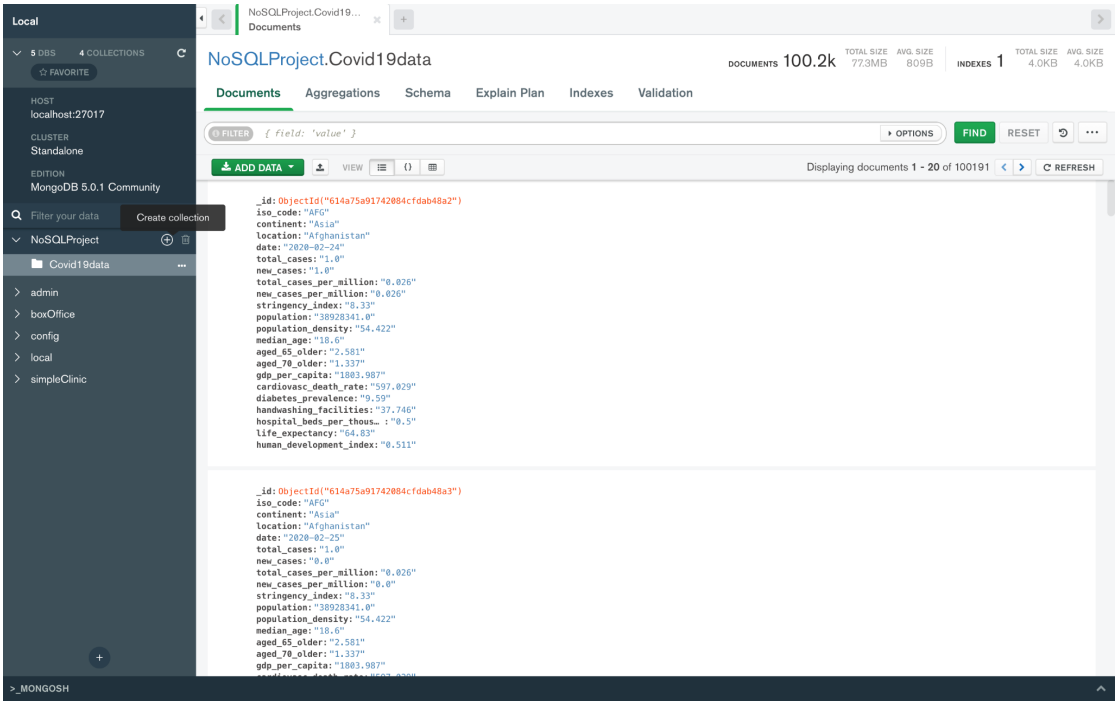

Enter the covid19data collection, and select add data




Click browse and select covid19data_2.json from the local system. Select the input file type as json and click import.



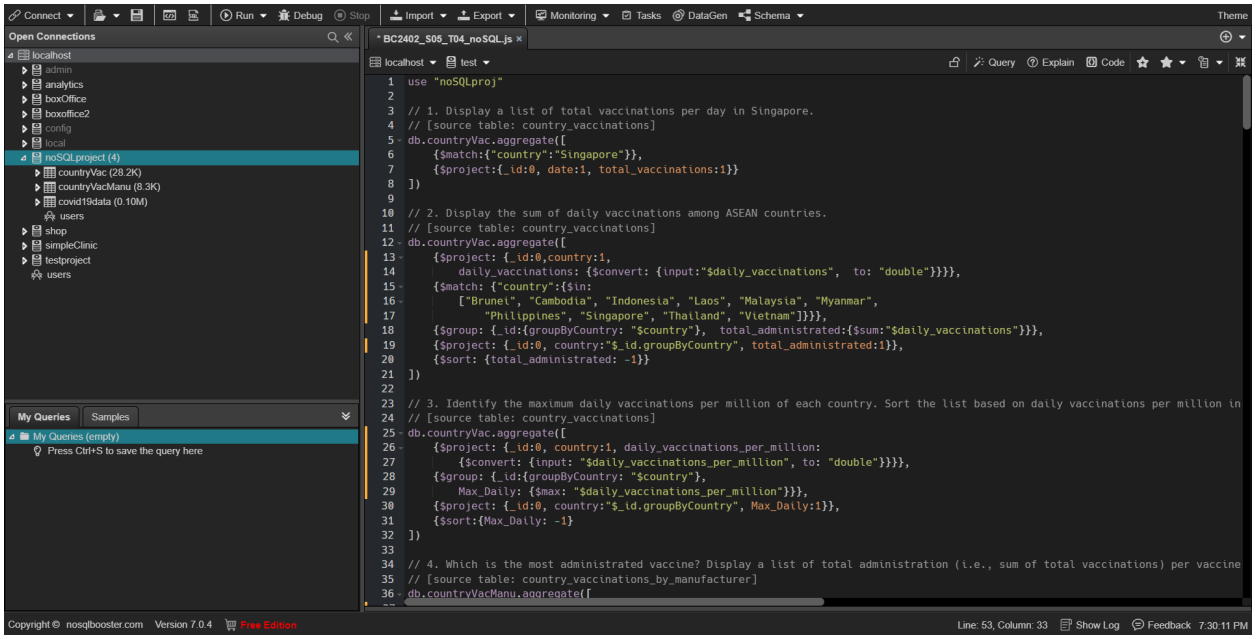

The data will look like the above once the import is completed.




Select create collection on the left and create collections for the other 2 json files, country_vaccinations.json and country_vaccinations_by_manufacturer.json as countryVac and countryVacManu respectively. Follow the steps above to import the files above into their respective collections.

**Step 4: Deploying NoSQL Queries from BC2402_S05_T04_noSQL.js**

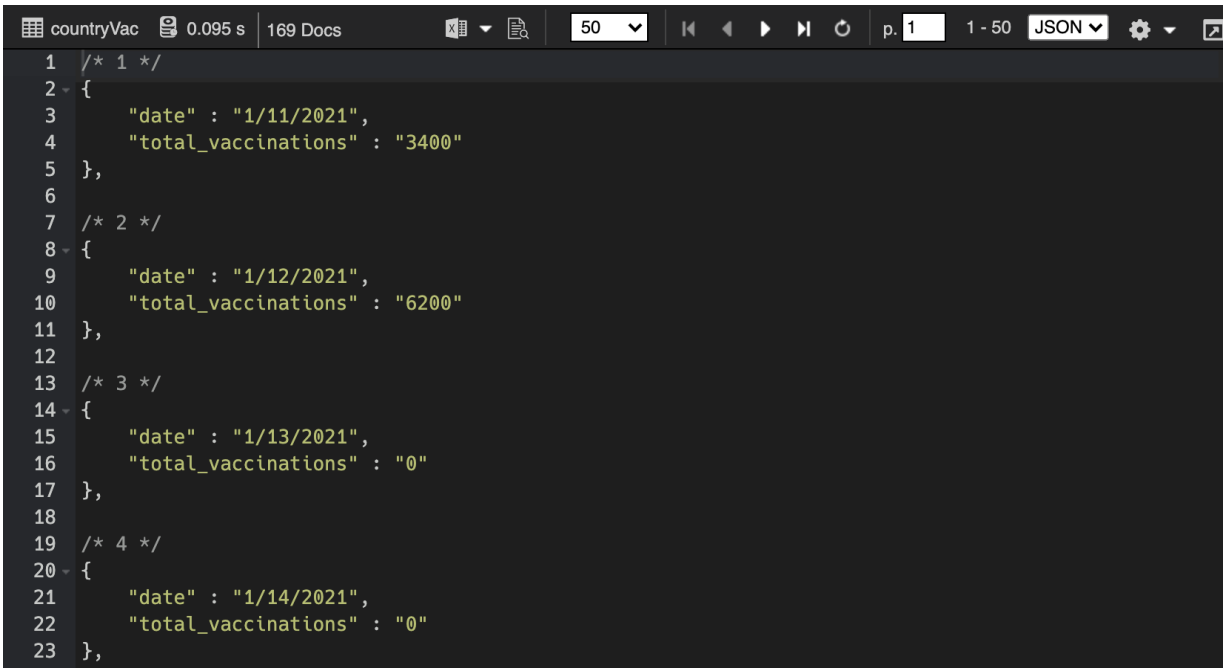
Launch NoSQLBooster for MongoDB and refresh. Check that the 3 collections have been imported. Choose 'BC2402_S05_T04_noSQL.js' from the saved directory.




Proceed to run the queries by highlighting and executing the selected portions of the script for each query.

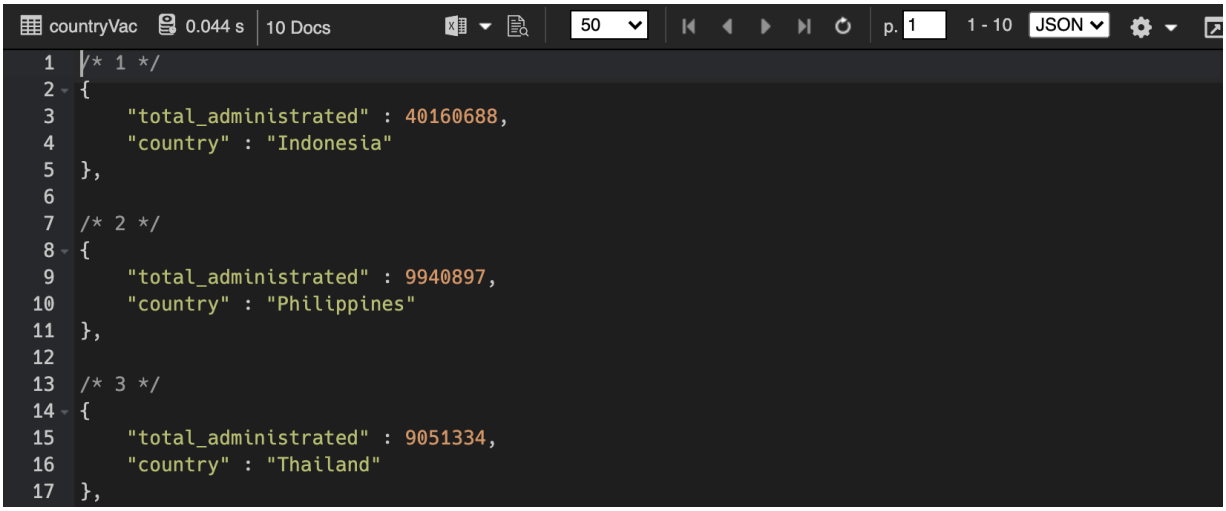
## 4.4 Query Results from Non-relational Database

**1. Display a list of total vaccinations per day in Singapore.**
**[source table: country_vaccinations]**



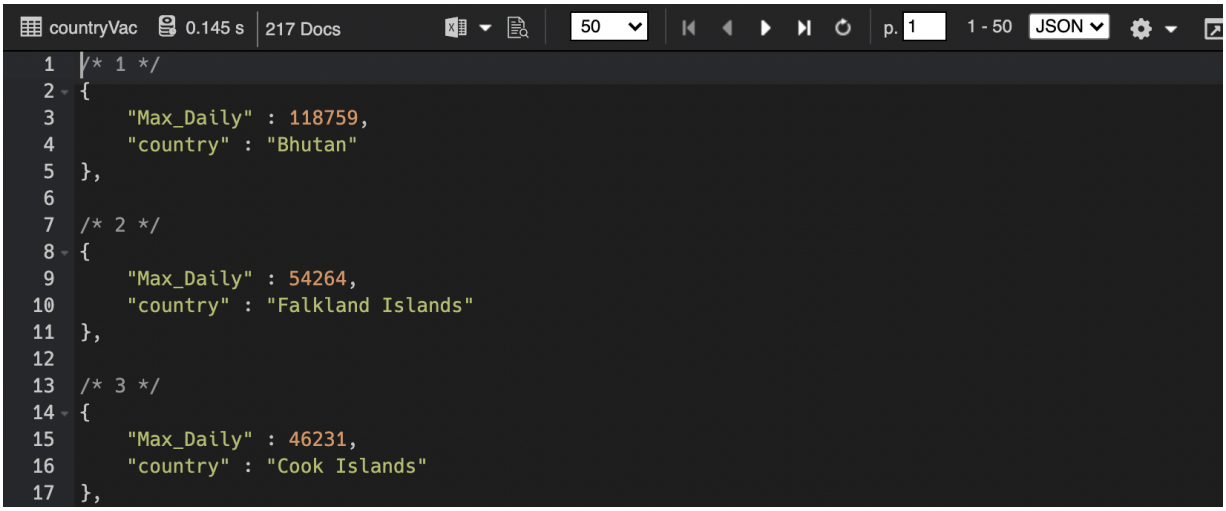

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$match:{"country":"Singapore"}},<br>{$project:{_id:0, date:1, total_vaccinations:1}}<br>])<br><br>**169 Docs** | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Match where country = “Singapore”.<br>● Project the date and corresponding total_vaccinations. |

**2. Display the sum of daily vaccinations among ASEAN countries.**
**[source table: country_vaccinations]**



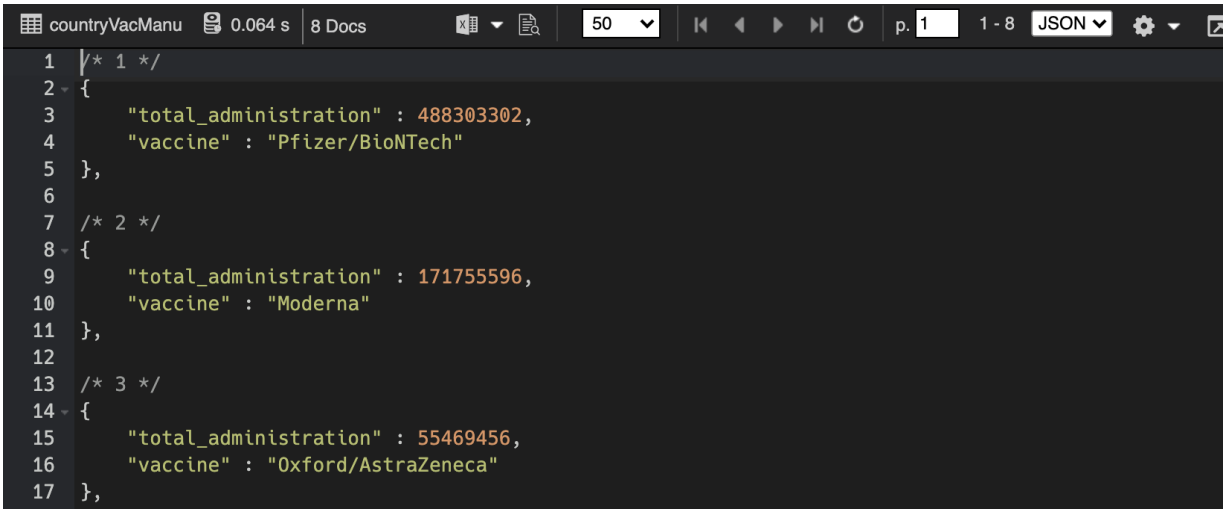

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$project: {_id:0,country:1,<br>daily_vaccinations: {$convert:<br>{input:"$daily_vaccinations", to: "double"}}}},<br>{$match: {"country":{$in: ["Brunei",<br>"Cambodia", "Indonesia", "Laos", "Malaysia",<br>"Myanmar", "Philippines", "Singapore",<br>"Thailand", "Vietnam"]}}},<br>{$group: {_id:{groupByCountry:<br>"$country"},<br>total_administrated:{$sum:"$daily_vaccination<br>s"}}},<br>{$project: {_id:0,<br>country:"$_id.groupByCountry",<br>total_administrated:1}},<br>{$sort: {total_administrated: -1}}<br>])<br><br>**10 Docs** | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Project country and also convert the data type of daily_vaccinations to “double”.<br>● Match countries with a specified array filled with all 10 ASEAN countries using the $in operator.<br>● Group by country and retrieve the summation of daily_vaccinations [total_adminstrated].<br>● Project the individual ASEAN country and the respective total_adminstrated.<br>● Sort by total_administrated in descending order. |

**3. Identify the maximum daily vaccinations per million of each country. Sort the list based on daily vaccinations per million in a descending order.**
**[source table: country_vaccinations]**



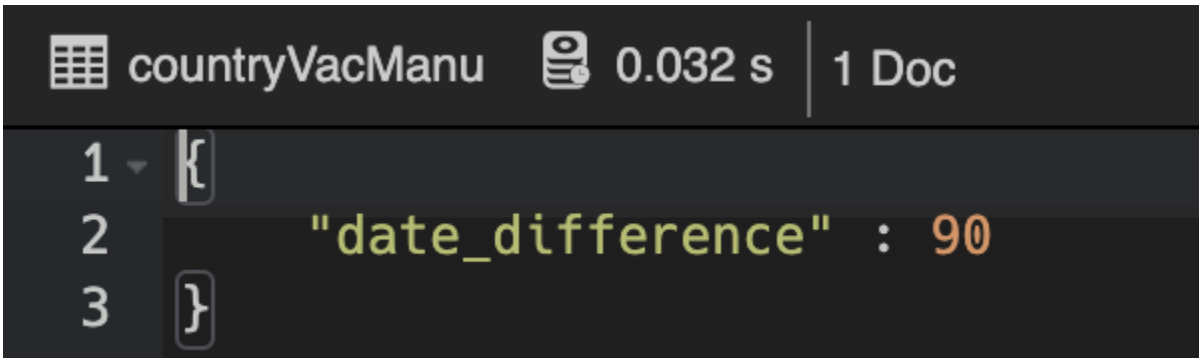

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$project: {_id:0, country:1, daily_vaccinations_per_million: {$convert: {input: "$daily_vaccinations_per_million", to: "double"}}}},<br>{$group: {_id:{groupByCountry: "$country"}, Max_Daily: {$max: "$daily_vaccinations_per_million"}}},<br>{$project: {_id:0, country:"$_id.groupByCountry", Max_Daily:1}},<br>{$sort:{Max_Daily: -1}<br>])<br><br>**217 Docs** | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Project country and also convert the data type of daily_vaccinations_per_million to “double”.<br>● Group by country and retrieve the maximum of daily_vaccinations_per_million [Max_Daily].<br>● Project country and the respective Max_Daily.<br>● Sort by Max_Daily in descending order. |

**4. Which is the most administrated vaccine? Display a list of total administration (i.e., sum of total vaccinations) per vaccine.**
**[source table: country_vaccinations_by_manufacturer]**



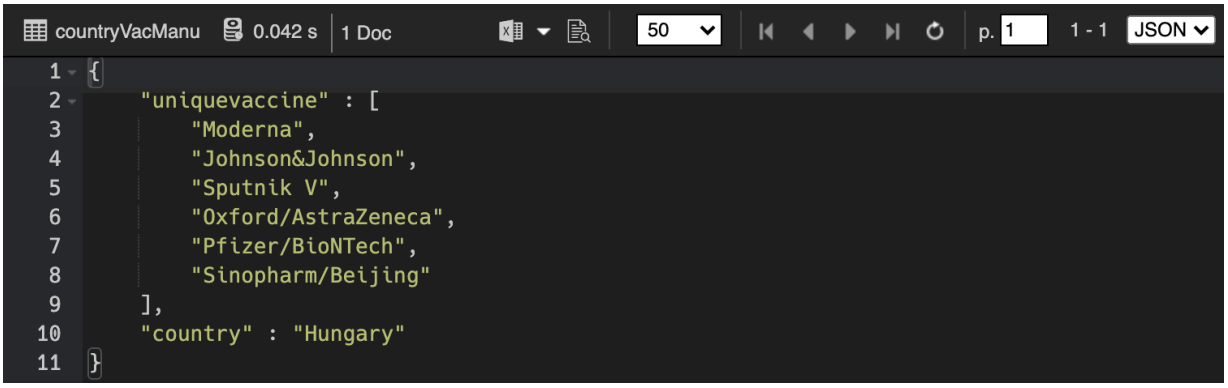

| Query | Explanation |
|---|---|
| db.countryVacManu.aggregate([<br>{$project: {_id:0, location:1, vaccine:1, total_vaccinations: {$convert:{input: "$total_vaccinations", to: "double"}}}},<br>{$group: {_id: {location: "$location", vaccine: "$vaccine"}, max_administration_per_country: {$max: "$total_vaccinations"}}},<br>{$group: {_id: {vaccine : "$_id.vaccine"}, total_administration: {$sum: "$max_administration_per_country"}}},<br>{$project: {_id:0, vaccine: "$_id.vaccine", total_administration:1}},<br>{$sort: {total_administration:-1}}<br>])<br><br>**8 Docs** | Using the aggregate function on the countryVacManu collection, pipeline the following functions:<br>● Project location, vaccine and also convert the data type of total_vaccinations to “double”.<br>● Group by location and vaccine and retrieve the maximum of total_vaccinations [max_administration_per_country].<br>● Group by vaccine and retrieve the summation of max_administration_per_country [total_administration].<br>● Project vaccine and the respective total_administration.<br>● Sort by total_administration in descending order. |

**5. Italy has commenced administrating various vaccines to its populations as a vaccine becomes available. Identify the first dates of each vaccine being administrated, then compute the difference in days between the earliest date and the 4th date.**
**[source table: country_vaccinations_by_manufacturer]**



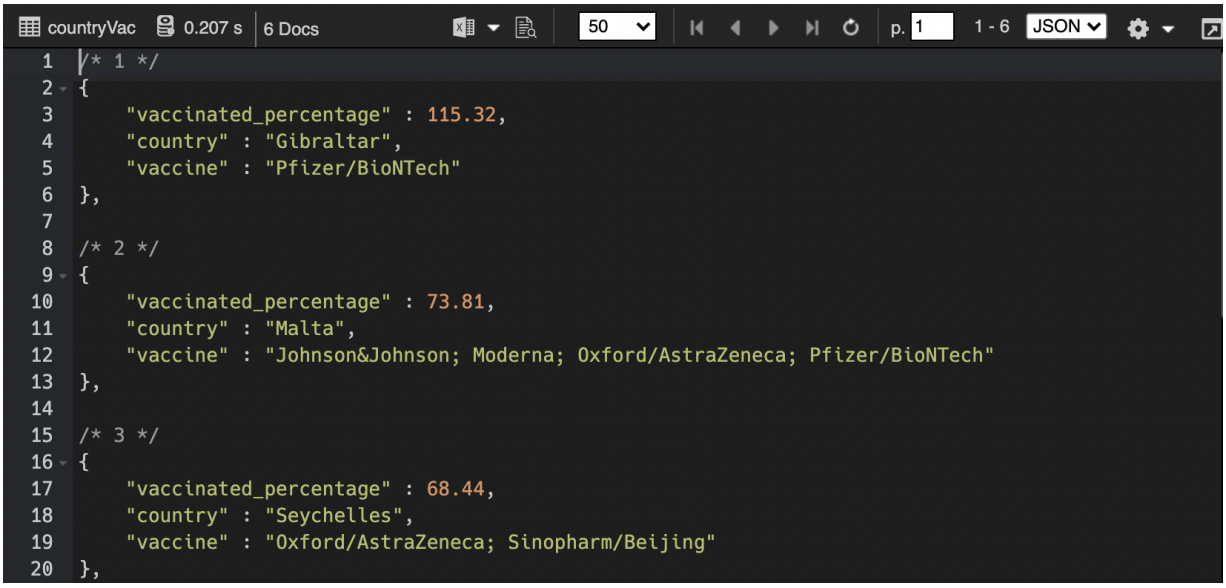

| Query | Explanation |
|---|---|
| db.countryVacManu.aggregate([<br>{$match: {location: "Italy"}},<br>{$project: {_id: 0, vaccine: 1, date: {$convert: {input: "$date", to: "date"}}}},<br>{$group: {_id: {vaccine: "$vaccine"}, min_date: {$min: "$date"}}},<br>{$project: {_id:0, vaccine: "$_id.vaccine", min_date:1}},<br>{$sort: {min_date: -1}},<br>{$group: {_id: {}, max_date: {$max: "$min_date"}, min_date: {$min: "$min_date"}}},<br>{$project: {_id: 0, date_difference: {$divide: [{$subtract: ["$max_date", "$min_date"]}, 1000*60*60*24]}}}<br>])<br><br>**1 Doc** | Using the aggregate function on the countryVacManu collection, pipeline the following functions:<br>● Match where location = “Italy”.<br>● Project vaccine and also convert data type of date to “date”.<br>● Group by vaccine and retrieve the minimum date [min_date].<br>● Project vaccine and the respective min_date.<br>● Sort by min_date in descending order.<br>● Using the group function, we obtained the earliest date, as well as the latest date.<br>● Project the date_difference by subtracting the earliest date from the latest date. |

**6. What is the country with the most types of administrated vaccine? [source table: country_vaccinations_by_manufacturer]**




| Query | Explanation |
|---|---|
| db.countryVacManu.aggregate([<br>{$group: {_id: {location: "$location"}, uniquevaccine: {$addToSet: "$vaccine"}}},<br>{$project: {_id:0, country: "$_id.location", uniquevaccine:1, vaccineCount: {$size: "$uniquevaccine"}}},<br>{$sort: {vaccineCount: -1}},<br>{$limit:1},<br>{$project: {country:1, uniquevaccine:1}}<br>])<br><br>**1 Doc** | Using the aggregate function on the countryVacManu collection, pipeline the following functions:<br>● Group by location and retrieve arrays of unique vaccines by applying the $addtoset operator on vaccine [unqiuevaccine].<br>● Project location [country], unique vaccine and also count the number of unique vaccines in each uniquevaccine array by using the $size operator [vaccineCount].<br>● Sort by vaccineCount in descending order.<br>● Limit the results to only show the first document.<br>● Project country and the respective uniquevaccine array. |

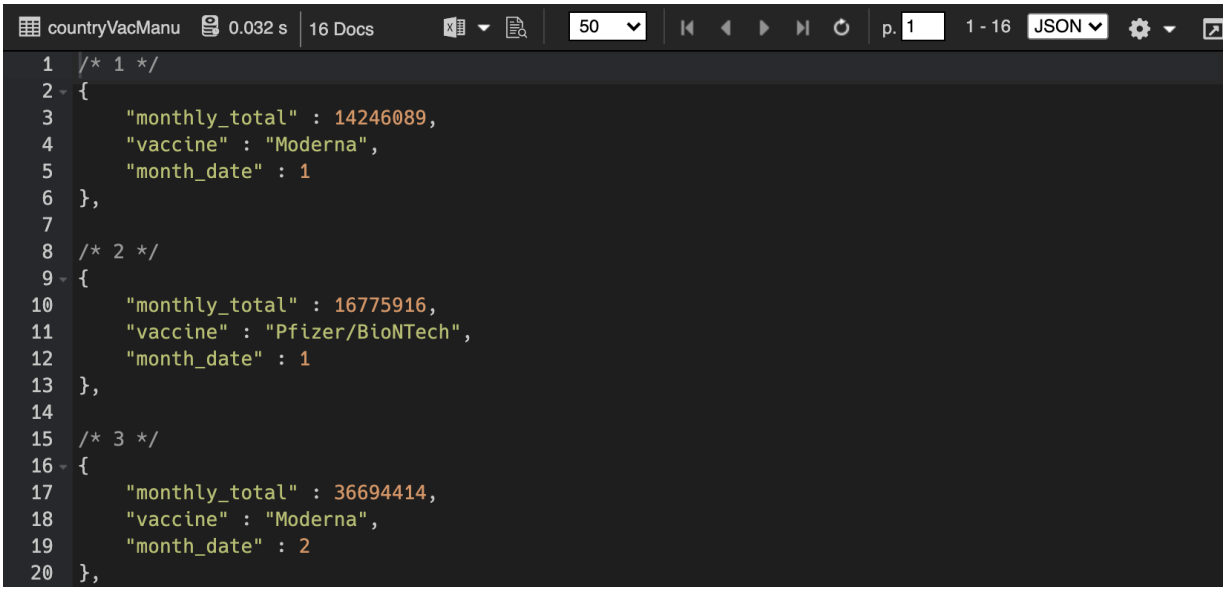
**7. What are the countries that have fully vaccinated more than 60% of its people? For each country, display the vaccines administrated.**
**[source table: country_vaccinations]**




```
/* 1 */
{
    "vaccinated_percentage" : 115.32,
    "country" : "Gibraltar",
    "vaccine" : "Pfizer/BioNTech"
},

/* 2 */
{
    "vaccinated_percentage" : 73.81,
    "country" : "Malta",
    "vaccine" : "Johnson&Johnson; Moderna; Oxford/AstraZeneca; Pfizer/BioNTech"
},

/* 3 */
{
    "vaccinated_percentage" : 68.44,
    "country" : "Seychelles",
    "vaccine" : "Oxford/AstraZeneca; Sinopharm/Beijing"
},
```

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$project: {_id: 0, country: 1, vaccines: 1, people_fully_vaccinated_per_hundred: {$convert: {input: "$people_fully_vaccinated_per_hundred", to: "double"}}}},<br>{$group: {_id: {country: "$country", vaccine: "$vaccines"}, vaccinated_percentage: {$max: "$people_fully_vaccinated_per_hundred"}}},<br>{$match:{vaccinated_percentage:{$gt:60}}},<br>{$project: {_id: 0,country:"$_id.country", vaccine: "$_id.vaccine", vaccinated_percentage:1}},<br>{$sort: {vaccinated_percentage: -1}}<br>])<br><br>**6 Docs** | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Project country, vaccines and also convert the data type of people_fully_vaccinated_per_hundred to “double”.<br>● Group by country and vaccines and retrieve the maximum of people_fully_vaccinated_per_hundred [vaccinated_percentage].<br>● Match where vaccinated_percentage > 60.<br>● Project country, vaccine and vaccinated_percentage.<br>● Sort by vaccinated_percentage in descending order. |

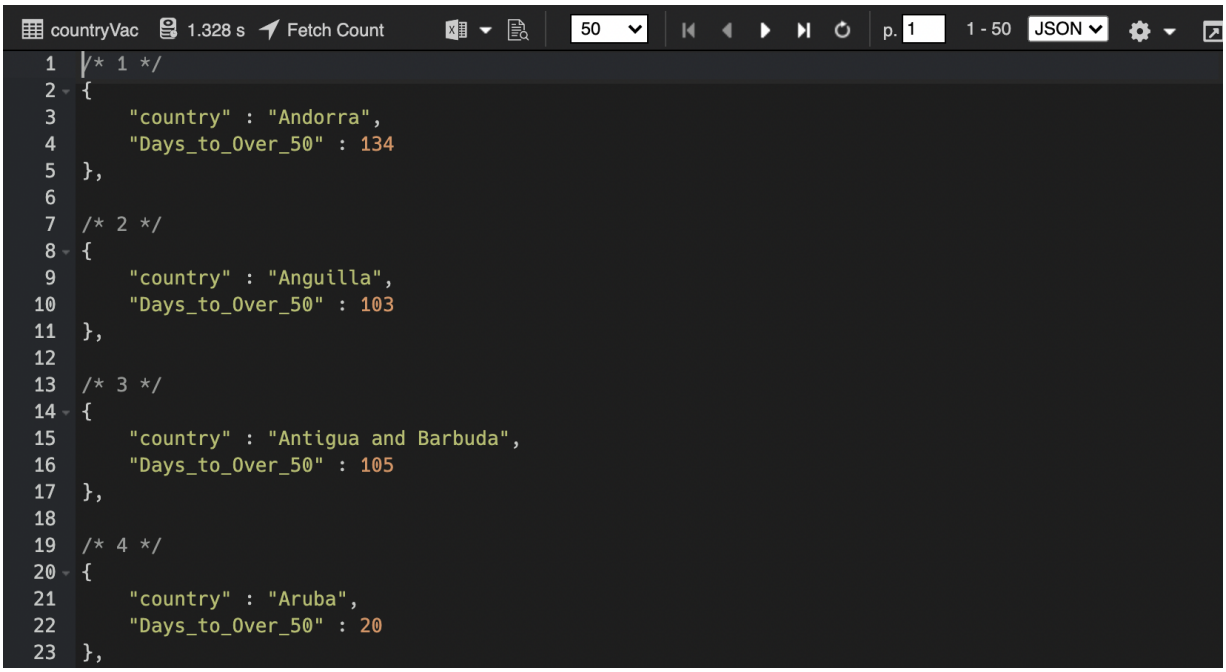
**8. Monthly vaccination insight – display the monthly total vaccination amount of each vaccine per month in the United States.**
**[source table: country_vaccinations_by_manufacturer]**




```
/* 1 */
{
    "monthly_total" : 14246089,
    "vaccine" : "Moderna",
    "month_date" : 1
},

/* 2 */
{
    "monthly_total" : 16775916,
    "vaccine" : "Pfizer/BioNTech",
    "month_date" : 1
},

/* 3 */
{
    "monthly_total" : 36694414,
    "vaccine" : "Moderna",
    "month_date" : 2
},
```

| Query | Explanation |
|---|---|
| db.countryVacManu.aggregate([<br>{$match: {location: "United States"}},<br>{$project: {_id: 0, vaccine: 1, total_vaccinations: {$convert: {input: "$total_vaccinations", to: "double"}}, date: {$convert: {input: "$date", to: "date"}}}},<br>{$group: {_id: {vaccine: "$vaccine", month_date: {$month: "$date"}}, monthly_total: {$max: "$total_vaccinations"}}},<br>{$project: {_id: 0, vaccine: "$_id.vaccine", month_date: "$_id.month_date", monthly_total:1}},<br>{$sort: {month_date: 1}}<br>])<br><br>**16 Docs** | Using the aggregate function on the countryVacManu collection, pipeline the following functions:<br>● Match where location = “United States”.<br>● Project vaccine and convert the data type of total_vaccinations and date to “double” and “date” respectively.<br>● Group by vaccine and month (obtained by using the $month operator on date), and retrieve the maximum of total_vaccinations [monthly_total].<br>● Project vaccine, month_date and monthly_total.<br>● Sort by month_date in ascending order. |

**9. Days to 50 percent. Compute the number of days (i.e., using the first available date on records of a country) that each country takes to go above the 50% threshold of vaccination administration (i.e., total_vaccinations_per_hundred > 50) [source table: country_vaccinations]**

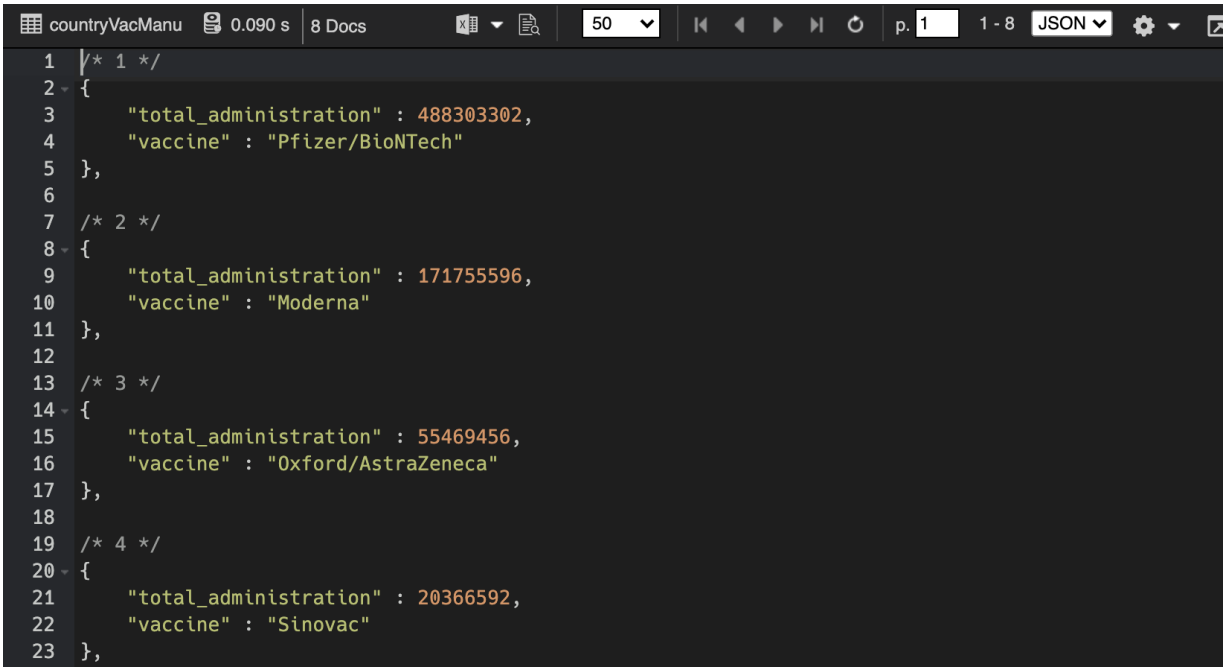



```
/* 1 */
{
    "country" : "Andorra",
    "Days_to_Over_50" : 134
},

/* 2 */
{
    "country" : "Anguilla",
    "Days_to_Over_50" : 103
},

/* 3 */
{
    "country" : "Antigua and Barbuda",
    "Days_to_Over_50" : 105
},

/* 4 */
{
    "country" : "Aruba",
    "Days_to_Over_50" : 20
},
```

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$project: {_id:0, country:1, date: {$convert: {input:"$date",to:"date"}},<br>total_vaccinations_per_hundred: {$convert: {input:"$total_vaccinations_per_hundred",to:"double"}}}},<br>{$match: {total_vaccinations_per_hundred: {$gt:50}}},<br>{$group: {_id: {"country": "$country"}, threshold_date: {$min: "$date"}}}<br>{$lookup: {<br>from: "countryVac",<br>localField: "_id.country",<br>foreignField: "country",<br>pipeline: [<br>{$project: {_id:0, country:1, date: {$convert: {input:"$date",to:"date"}}}},<br>{$group: {_id: {"country": "country"}, min_date: {$min: "$date"}}}],<br>as: "VacInfo"<br>}},<br>{$unwind: {path: "$VacInfo"}}, | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Project country, and convert the data type of date and total_vaccinations_per_hundred to “date” and “double” respectively.<br>● Match where total_vaccinations_per_hundred > 50.<br>● Group by country and retrieve the minimum date [threshold_date].<br>● Lookup from countryVac.<br>○ Project country, and convert the data type of date to “date”.<br>○ Group by country, and retrieve the first available date on records of a country [min_date].<br>● Unwind VacInfo.<br>● Project country and Days_to_Over_50 by subtracting min_date from threshold_date. |

| | |
|---|---|
| {$project: {_id:0, country: "$_id.country", Days_to_Over_50: {$sum: [{$divide: [{$subtract: ["$threshold_date","$VacInfo.min_date"]},1000*60*60*24]},1]}}},<br>{$sort: {country:1}}<br>])<br><br>**85 Docs** | • Sort the countries by alphabetical order. |

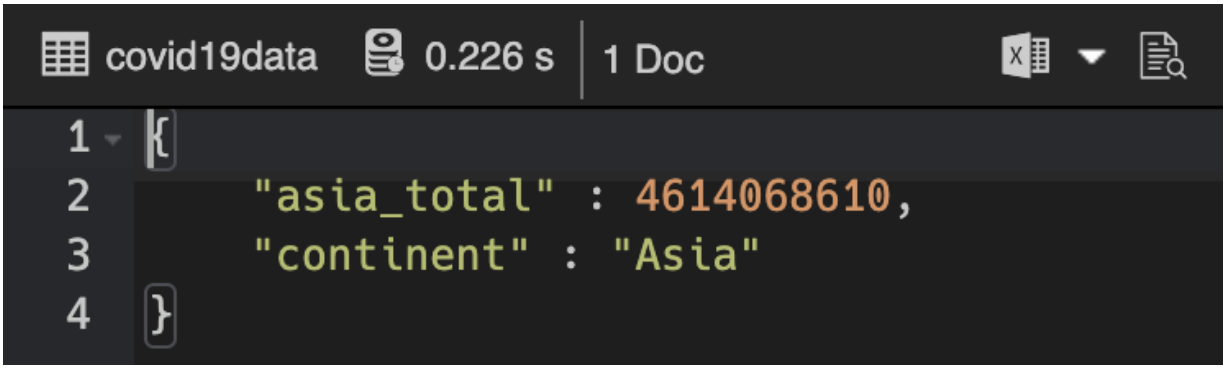
**10. Compute the global total of vaccinations per vaccine.**
**[source table: country_vaccinations_by_manufacturer]**




| Query | Explanation |
|---|---|
| db.countryVacManu.aggregate([<br>{$project: {_id: 0, location: 1, vaccine: 1, total_vaccinations: {$convert:{input: "$total_vaccinations", to: "double"}}}},<br>{$group: {_id: {location: "$location", vaccine: "$vaccine"}, max_administrationpc: {$max: "$total_vaccinations"}}},<br>{$group: {_id: {vaccine : "$_id.vaccine"}, total_administration: {$sum: "$max_administrationpc"}}},<br>{$project: {_id: 0, vaccine: "$_id.vaccine", total_administration: 1}},<br>{$sort: {total_administration: -1}} | Using the aggregate function on the countryVacManu collection, pipeline the following functions:<br>• Match where location = “Italy”.<br>• Project vaccine and also convert data type of date to “date”.<br>• Group by vaccine and retrieve the minimum date [min_date].<br>• Project vaccine and the respective min_date.<br>• Sort by min_date in descending order.<br>• Using the group function, we obtained the earliest date, as well as the latest |

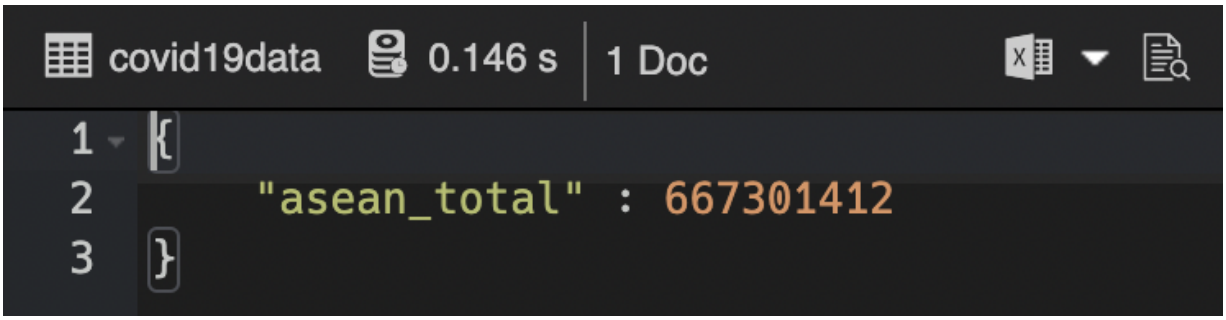
| | |
|---|---|
| ])<br><br>**8 Docs** | date.<br>● Project the date_difference by subtracting the earliest date from the latest date. |

**11. What is the total population in Asia?**



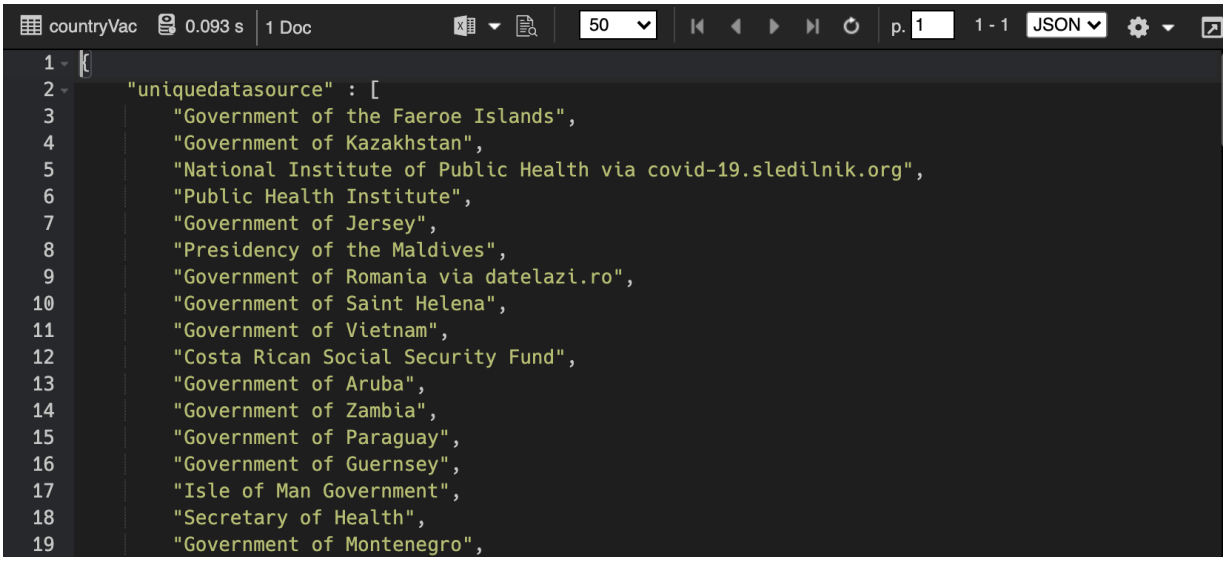

| Query | Explanation |
|---|---|
| db.covid19data.aggregate([<br>{$match: {continent: "Asia"}},<br>{$project: {_id: 0, continent: 1, location: 1, date: {$convert: {input: "$date", to: "date"}}, population: {$convert: {input: "$population", to: "double"}}}},<br>{$sort: {date:1}},<br>{$group: {_id: {location: "$location",continent:"$continent"}, latest_pop: {$last:"$population"}}},<br>{$group: {_id: {continent:"$_id.continent"},asia_total:{$sum:"$latest_pop"}}},<br>{$project:{_id:0, continent:"$_id.continent", asia_total:1}}<br>])<br><br>**1 Doc** | Using the aggregate function on the covid19data collection, pipeline the following functions:<br>● Match where continent = “Asia”.<br>● Project continent, location and also convert the data type of date and population to “date” and “double” respectively.<br>● Sort by date in ascending order.<br>● Group by location and continent and retrieve the latest population of each group by using the $last operator on population [population].<br>● Group by continent and retrieve the summation of population [asia_total].<br>● Project continent and asia_total. |

**12. What is the total population among the ten ASEAN countries?**



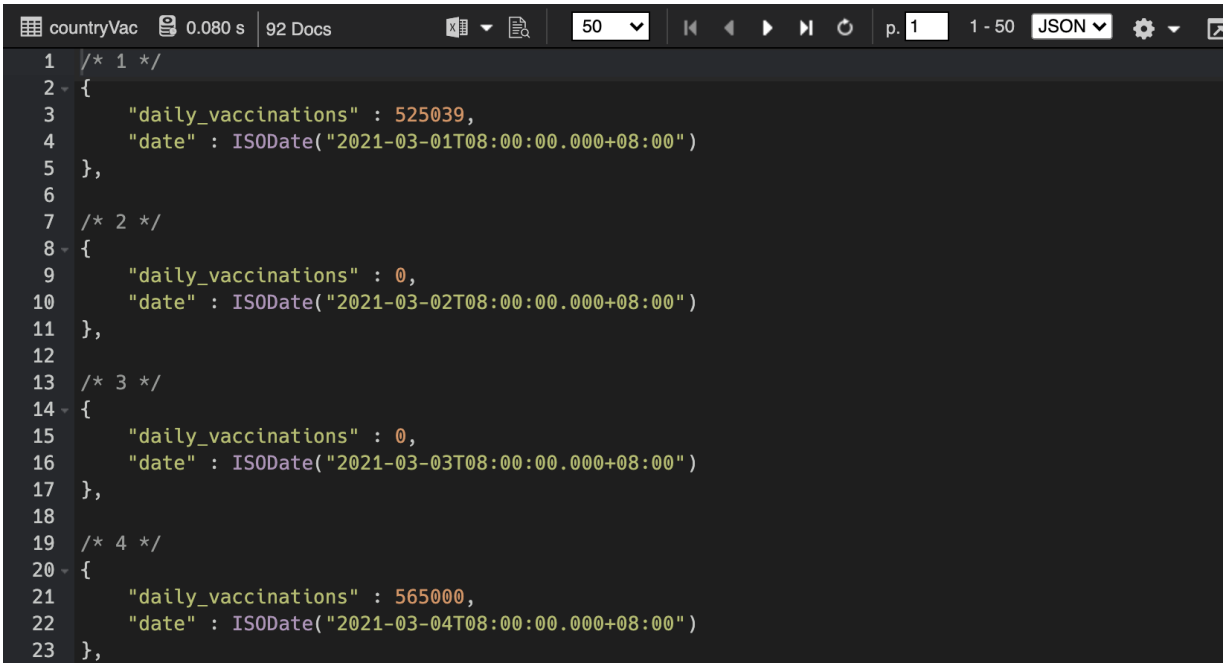

| Query | Explanation |
|---|---|
| db.covid19data.aggregate([<br>{$match: {"location":{$in: ["Brunei", "Cambodia", "Indonesia", "Laos", "Malaysia", "Myanmar", "Philippines", "Singapore", "Thailand", "Vietnam"]}}},<br>{$project: {_id:0, location:1, date:{$convert:{input:"$date",to:"date"}}, population: {$convert: {input: "$population", to: "double"}}}},<br>{$sort: {date:1}},<br>{$group: {_id: {location: "$location"}, latest_pop: {$last: "$population"}}},<br>{$group: {_id: {}, asean_total:{$sum:"$latest_pop"}}},<br>{$project: {_id:0, asean_total:1}}<br>])<br><br>**1 Doc** | Using the aggregate function on the covid19data collection, pipeline the following functions:<br>● Match the location with a specified array filled with all 10 ASEAN countries using the $in operator.<br>● Project location and also convert the data type of date and population to “date” and “double” respectively.<br>● Sort by date in ascending order.<br>● Group by location and retrieve the latest population of each group by using the $last operator on population [population].<br>● Using the group function, find the summation of population among all 10 countries [asean_total].<br>Project asean_total. |

**13. Generate a list of unique data sources (source_name).13. Generate a list of unique data sources (source_name).**



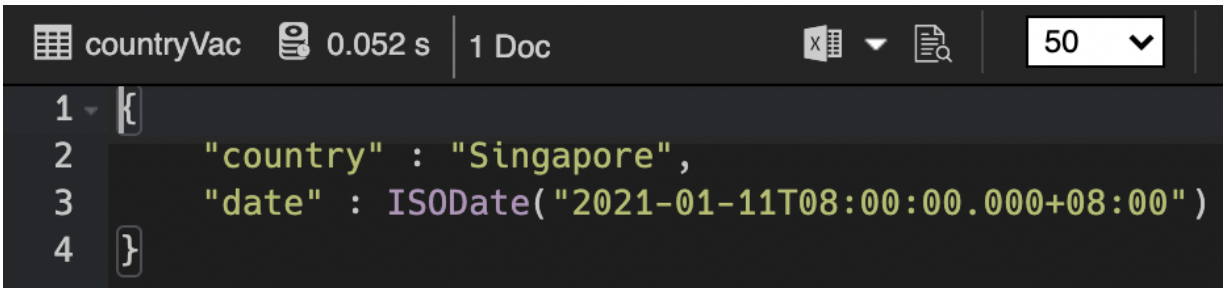

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$group: {_id:{}, uniquedatasource: {$addToSet: "$source_name"}}},<br>{$project: {_id:0, uniquedatasource:1}}<br>])<br><br>**1 Doc** | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Using the group function, add all the unique source_name into an array using the $addToSet operator [uniquedatasource].<br>● Project uniquedatasource. |

**14. Specific to Singapore, display the daily total_vaccinations starting (inclusive) March-1 2021 through (inclusive) May-31 2021.**



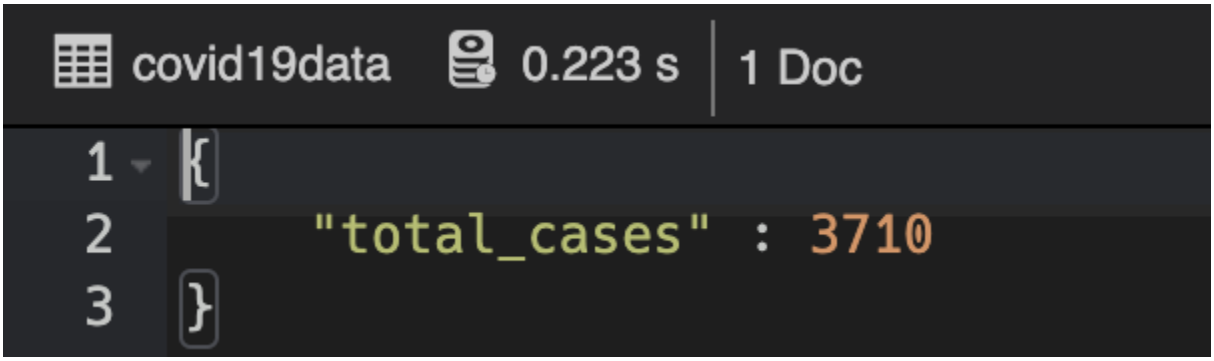

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$match: {country: "Singapore"}},<br>{$project: {_id:0, daily_vaccinations: {$convert: {input: "$total_vaccinations", to: "double"}}, date: {$convert: {input: "$date", to: "date"}}}},<br>{$match: {date: {$gte: ISODate("2021-03-01"), $lte: ISODate("2021-05-31")}}}<br>{$sort: {date: 1}}<br>])<br><br>**92 Docs** | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Match where country = “Singapore”.<br>● Project and convert the data type of total_vaccinations and date to “double” and “date” respectively.<br>● Match where the date is between “2021-03-01” and “2021-05-31”. |

**15. When is the first batch of vaccinations recorded in Singapore?**



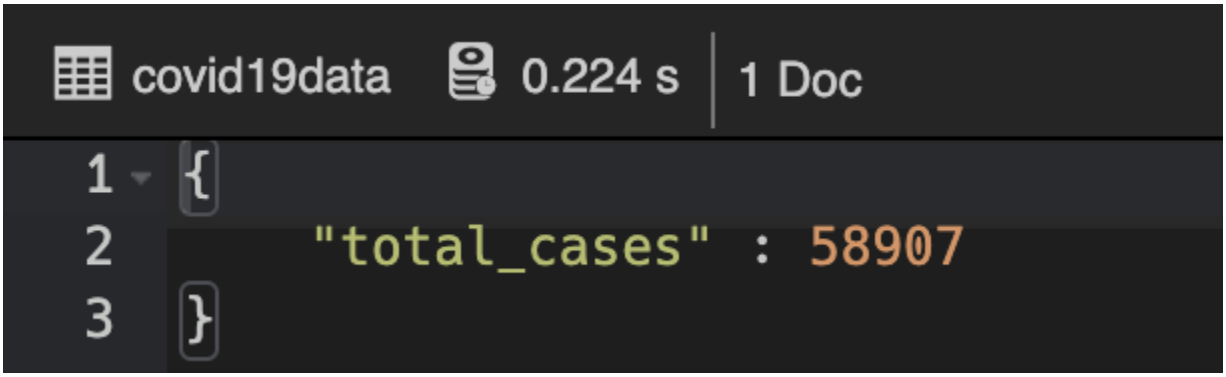

| Query | Explanation |
|---|---|
| db.countryVac.aggregate([<br>{$project: {_id:0, country:1, date: {$convert: {input: "$date", to: "date"}}, total_vaccinations: {$convert: {input: "$total_vaccinations", to: "double"}}}},<br>{$match: {country: "Singapore"}},<br>{$match: {total_vaccinations: {$gt:0}}},<br>{$sort: {date:1}},<br>{$limit: 1},<br>{$project: {_id:0, total_vaccinations: 0}}<br><br>])<br><br>**1 Doc** | Using the aggregate function on the countryVac collection, pipeline the following functions:<br>● Project country and also convert the data type of date and total_vaccinations to “date” and “double” respectively.<br>● Match where country = “Singapore” and total_vaccinations > 0.<br>● Sort by date in ascending order.<br>● Limit the results to only show the first document. |

**16. Based on the date identified in (5), specific to Singapore, compute the total number of new cases thereafter. For instance, if the date identified in (5) is Jan-1 2021, the total number of new cases will be the sum of new cases starting from (inclusive) Jan-1 to the last date in the dataset.**



| Query | Explanation |
|---|---|
| db.covid19data.aggregate([<br>{$match: {location: "Singapore"}},<br>{$project: {_id:0, date: {$convert: {input: "$date", to: "date"}}, new_cases: {$convert: {input: "$new_cases", to: "double"}}}},<br>{$match: {date: {$gte: ISODate("2021-01-11")}}},<br>{$group: {_id: {}, total_cases:{$sum:"$new_cases"}}},<br>{$project: {_id:0, total_cases:1}}<br>])<br><br>**1 Doc** | Using the aggregate function on the covid19data collection, pipeline the following functions:<br>● Project location and also convert the data type of date and new_cases to “date” and “double” respectively.<br>● Match where location = “Singapore” and date >= “2021-01-11”.<br>● Using the group function, find the total number of new_cases [total_new_cases].<br>● Project total_new_cases. |

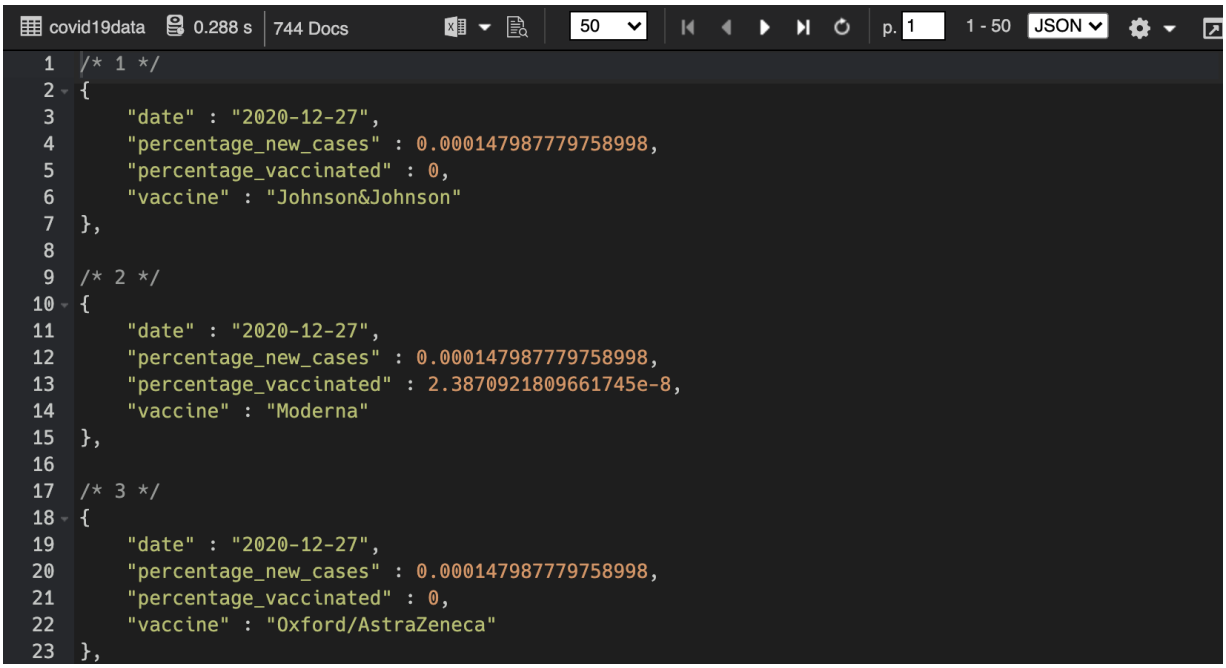
**17. Compute the total number of new cases in Singapore before the date identified in (5). For instance, if the date identified in (5) is Jan-1 2021 and the first date recorded (in Singapore) in the dataset is Feb-1 2020, the total number of new cases will be the sum of new cases starting from (inclusive) Feb-1 2020 through (inclusive) Dec-31 2020.**




| Query | Explanation |
|---|---|
| db.covid19data.aggregate([<br>{$match: {location: "Singapore"}},<br>{$project: {_id:0, date: {$convert: {input: "$date", to: "date"}}, new_cases: {$convert: {input: "$new_cases", to: "double"}}}},<br>{$match: {date: {$lt: ISODate("2021-01-11")}}},<br>{$group: {_id: {}, total_cases:{$sum:"$new_cases"}}},<br>{$project: {_id:0, total_cases:1}} | Using the aggregate function on the covid19data collection, pipeline the following functions:<br>● Project location and also convert the data type of date and new_cases to “date” and “double” respectively.<br>● Match where location = “Singapore” and date < “2021-01-11”.<br>● Using the group function, find the total number of new_cases [total_new_cases].<br>● Project total_new_cases. |

| | |
|---|---|
| ])<br><br>**1 Doc** | |

**18. Herd immunity estimation. On a daily basis, specific to Germany, calculate the percentage of new cases and total vaccinations on each available vaccine in relation to its population.**




```
/* 1 */
{
    "date" : "2020-12-27",
    "percentage_new_cases" : 0.000147987779758998,
    "percentage_vaccinated" : 0,
    "vaccine" : "Johnson&Johnson"
},

/* 2 */
{
    "date" : "2020-12-27",
    "percentage_new_cases" : 0.000147987779758998,
    "percentage_vaccinated" : 2.3870921809661745e-8,
    "vaccine" : "Moderna"
},

/* 3 */
{
    "date" : "2020-12-27",
    "percentage_new_cases" : 0.000147987779758998,
    "percentage_vaccinated" : 0,
    "vaccine" : "Oxford/AstraZeneca"
},
```

| Query | Explanation |
|---|---|
| db.covid19data.aggregate([<br>{$match: {"location" :"Germany"}},<br>{$project: {_id:0, date: 1,<br>new_cases: {$convert: {input:"$new_cases", to: "double"}},<br>population: {$convert: {input:"$population", to: "double"}}}},<br>{$lookup:<br>{<br>from: "countryVacManu",<br>localField: "date",<br>foreignField: "date",<br>pipeline:<br>[<br>{$match: {"location" :"Germany"}}, | Using the aggregate function on the covid19data collection, pipeline the following functions:<br>● Match where location = “Germany”.<br>● Project date and also convert the data type of new_cases and population to “double”.<br>● Lookup from countryVacManu.<br>○ Match location = “Germany”.<br>○ Project date, vaccine and also convert the data type of total_vaccinations to “double”.<br>● Unwind vaccInfo.<br>● Project date, percentage of new cases in relation to population by dividing new_cases by population and percentage of total vaccinations in relation to population by dividing |

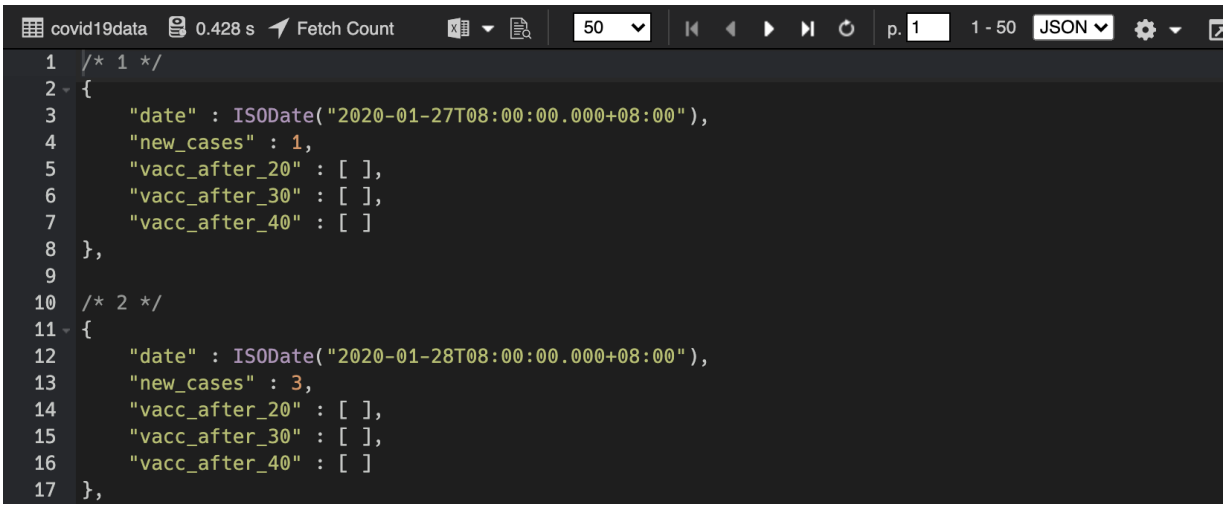
```
            {$project: {_id:0, date: 1, vaccine:1,
                        total_vaccinations:
{$convert: {input:"$total_vaccinations", to:
"double"}}}},
          ]
          as: "vaccInfo"
       }
   }
   {$unwind:"$vaccInfo"},
   {$project: {date:1, percentage_new_cases:
{$multiply: [{$divide: ["$new_cases",
"$population"]}, 100]},
              percentage_vaccinated: {$multiply:
[{$divide: ["$vaccInfo.total_vaccinations",
"$population"]}, 100]},
vaccine:"$vaccInfo.vaccine"}},
   {$sort: {date:1, vaccine:1}}
])
```

**744 Docs**

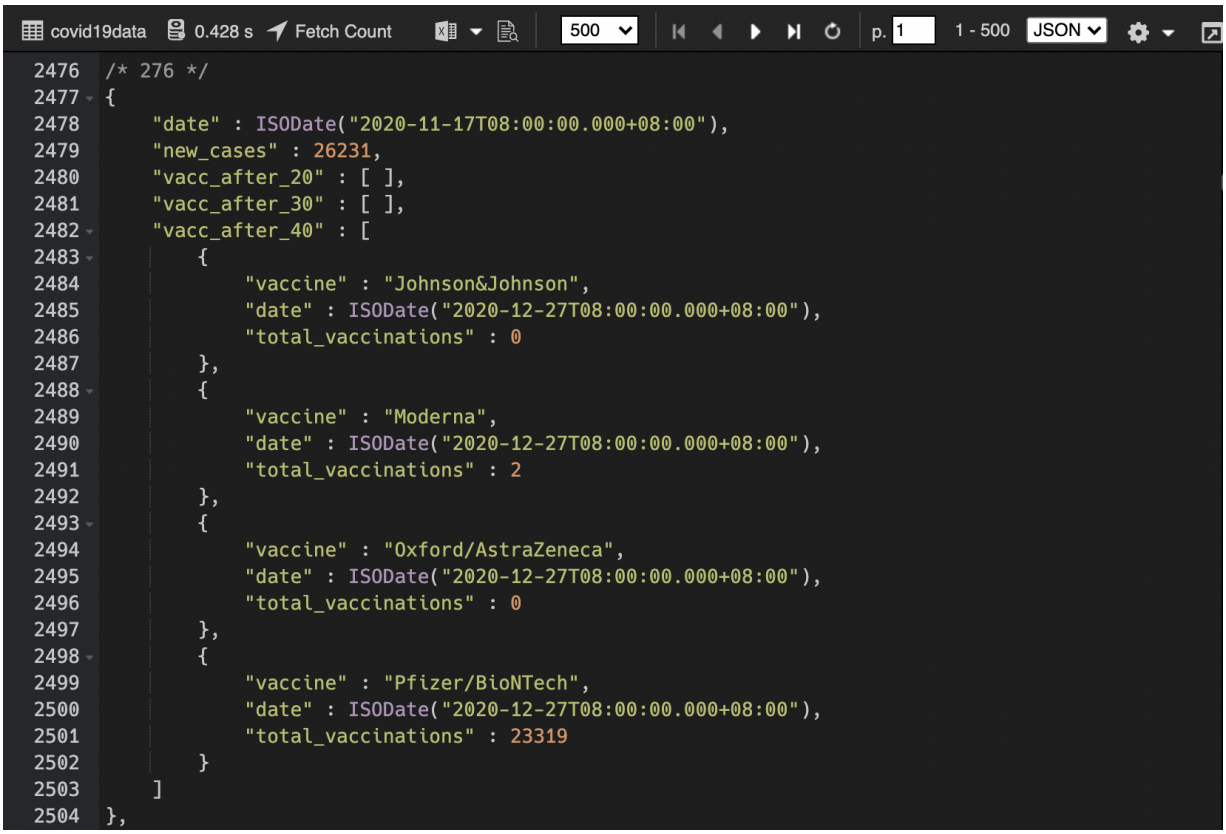
total_vaccinations by population.

- Sort by date in chronological order.

**19. Vaccination Drivers. Specific to Germany, based on each daily new case, display the total vaccinations of each available vaccines after 20 days, 30 days, and 40 days.**




```
/* 1 */
{
    "date" : ISODate("2020-01-27T08:00:00.000+08:00"),
    "new_cases" : 1,
    "vacc_after_20" : [ ],
    "vacc_after_30" : [ ],
    "vacc_after_40" : [ ]
},

/* 2 */
{
    "date" : ISODate("2020-01-28T08:00:00.000+08:00"),
    "new_cases" : 3,
    "vacc_after_20" : [ ],
    "vacc_after_30" : [ ],
    "vacc_after_40" : [ ]
},
```




| Query | Explanation |
|---|---|
| db.covid19data.createIndex({"date":1})<br>db.countryVacManu.createIndex({"date":1})<br><br>db.covid19data.aggregate([<br>{$project: {_id: 0, location:1, date: {$convert: {input: "$date", to: "date"}}, new_cases: {$convert: {input:"$new_cases", to: "double"}}}},<br>{$match: {location: "Germany", new_cases: {$gt:0}}},<br>{$project: {date: 1, "20daysAfter": {$add: ["$date",20*1000*60*60*24]},<br>"30daysAfter": {$add: ["$date",30*1000*60*60*24]},<br>"40daysAfter": {$add: ["$date",40*1000*60*60*24]}, new_cases: 1}},<br>{$project: {date: 1, "20daysAfter": {$substr: [{$convert: {input: "$20daysAfter", to: "string"}}, 0, 10]}, | Using the aggregate function on the covid19data collection, pipeline the following functions:<br>● Project location and also convert the data type of date and new_cases to “date” and “double” respectively.<br>● Match where location = “Germany” and new_cases > 0.<br>● Project date and new_cases, and find dates 20, 30 & 40 days after the initial date. [20daysAfter, 30daysAfter, 40daysAfter].<br>● Project date and new_cases, and find dates 20, 30 & 40 days after the initial date. [20daysAfter, 30daysAfter, 40daysAfter].<br>● Project date and also convert the datatype of 20daysAfter, 30daysAfter and 40daysAfter to “string”. At the same time, using $substr, obtain the first 10 characters of 20daysAfter, 30daysAfter and 40daysAfter, to |

```
            "30daysAfter": {$substr: [{$convert:
{input: "$30daysAfter", to: "string"}}, 0, 10]},
            "40daysAfter": {$substr: [{$convert:
{input: "$40daysAfter", to: "string"}}, 0, 10]},
new_cases: 1}},
   {$lookup:
      {
         from: "countryVacManu",
         localField: "20daysAfter"
         foreignField: "date"
         pipeline:
         [
            {$match: {"location" :"Germany"}},
            {$project: {_id:0, vaccine:1, date:
{$convert: {input: "$date", to: "date"}},
                        total_vaccinations:
{$convert: {input:"$total_vaccinations", to:
"double"}}}}
         ]
         as: "vacc_after_20"
      }
   },
   {$lookup:
      {
         from: "countryVacManu",
         localField: "30daysAfter"
         foreignField: "date"
         pipeline:
         [
            {$match: {"location" :"Germany"}},
            {$project: {_id:0, vaccine:1, date:
{$convert: {input: "$date", to: "date"}},
                        total_vaccinations:
{$convert: {input:"$total_vaccinations", to:
"double"}}}}
         ]
         as: "vacc_after_30"
      }
   }
   {$lookup:
      {
         from: "countryVacManu",
         localField: "40daysAfter"
```

ensure that each string only contains the date, excluding the time.

- Lookup from countryVacManu 3 times, once for 20daysAfter, once for 30daysAfter, and once for 40daysAfter.
    - Match location = “Germany”.
    - Project vaccine and also convert the data type of date and total_vaccinations to “date” and “double” respectively.
- Project date, new_cases, vacc_after_20, vacc_after_30 and vacc_after_40.

```
        foreignField: "date"
        pipeline:
        [
            {$match: {"location" :"Germany"}},
            {$project: {_id:0, vaccine:1, date:
{$convert: {input: "$date", to: "date"}},
                       total_vaccinations:
{$convert: {input:"$total_vaccinations", to:
"double"}}}}
        ]
        as: "vacc_after_40"
      }
   }
   {$project: {date:1, new_cases:1,
vacc_after_20:1, vacc_after_30:1,
vacc_after_40:1}}
])
```

**504 Docs**

**20. Vaccination Effects. Specific to Germany, on a daily basis, based on the total number of accumulated vaccinations (sum of total_vaccinations of each vaccine in a day), generate the daily new cases after 21 days, 60 days, and 120 days.**

countryVacManu 0.289 s 186 Docs 500 p. 1 1 - 186 JSON

```
/* 1 */
{
    "accumulated_vaccinations" : 23321,
    "date" : ISODate("2020-12-27T08:00:00.000+08:00"),
    "new_cases_after_21" : [
        {
            "date" : ISODate("2021-01-17T08:00:00.000+08:00"),
            "new_cases" : 11484
        }
    ],
    "new_cases_after_60" : [
        {
            "date" : ISODate("2021-02-25T08:00:00.000+08:00"),
            "new_cases" : 11032
        }
    ],
    "new_cases_after_120" : [
        {
            "date" : ISODate("2021-04-26T08:00:00.000+08:00"),
            "new_cases" : 5961
        }
    ]
},
```

| Query | Explanation |
| --- | --- |
| db.covid19data.createIndex({"date":1})<br>db.countryVacManu.createIndex({"date":1})<br><br>db.countryVacManu.aggregate([<br>{$project: {_id: 0, location:1, date: {$convert: {input: "$date", to: "date"}}, total_vaccinations: {$convert: {input:"$total_vaccinations", to: "double"}}}},<br>{$match: {location: "Germany"}},<br>{$group: {_id: {date: "$date"}, accumulated_vaccinations:{$sum:"$total_vaccinations"}}},<br>{$sort: {"_id.date":1}}<br>{$project: {_id:0, date: "$_id.date", "21daysAfter": {$add: ["$_id.date",21*1000*60*60*24]},<br>"60daysAfter": {$add: ["$_id.date",60*1000*60*60*24]},<br>"120daysAfter": {$add: ["$_id.date",120*1000*60*60*24]}, accumulated_vaccinations:1}},<br>{$project: {date: 1, "21daysAfter": {$substr: [{$convert: {input: "$21daysAfter", to: "string"}}, 0, 10]},<br>"60daysAfter": {$substr: [{$convert: {input: "$60daysAfter", to: "string"}}, 0, 10]},<br>"120daysAfter": {$substr: [{$convert: {input: "$120daysAfter", to: "string"}}, 0, 10]}, accumulated_vaccinations: 1}},<br>{$lookup:<br>{<br>from: "covid19data",<br>localField: "21daysAfter"<br>foreignField: "date"<br>pipeline:<br>[<br>{$match: {"location" :"Germany"}},<br>{$project: {_id:0, date: {$convert: {input: "$date", to: "date"}},<br>new_cases: {$convert: {input:"$new_cases", to: "double"}}}} | Using the aggregate function on the countryVacManu collection, pipeline the following functions:<br>● Project location and also convert the data type of date and total_vaccinations to “date” and “double” respectively.<br>● Match where location = “Germany”.<br>● Group by date and retrieve total number of accumulated vaccinations by finding sum of total_vaccinations. [accumulated_vaccinations].<br>● Sort by date in chronological order.<br>● Project date and accumulated_vaccinations, and find dates 21, 60 & 120 days after initial date. [21daysAfter, 60daysAfter, 120daysAfter].<br>● Project date and find dates 20, 30 & 40 days after initial date. [20daysAfter, 30daysAfter, 40daysAfter].<br>● Project date and also convert the datatype of 21daysAfter, 60daysAfter and 120daysAfter to “string”. At the same time, using $substr, obtain the first 10 characters of 21daysAfter, 60daysAfter and 120daysAfter, to ensure that each string only contains the date, excluding the time.<br>● Lookup from covid19data 3 times, once for 21daysAfter, once for 60daysAfter, and once for 120daysAfter.<br>○ Match location = “Germany”.<br>○ Project and convert the data type of date and total_vaccinations to “date” and “double” respectively.<br>● Project date, accumulated_vaccinations, new_cases_after_21, new_cases_after_60 and new_cases_after_120.<br>○ Match location = “Germany”.<br>○ Project and convert the data type of date and total_vaccinations to “date” and “double” respectively. |

```
        ]
        as: "new_cases_after_21"
      }
   },
   {$lookup:
      {
        from: "covid19data",
        localField: "60daysAfter"
        foreignField: "date"
        pipeline:
        [
          {$match: {"location" :"Germany"}},
          {$project: {_id:0, date: {$convert:
{input: "$date", to: "date"}},
                     new_cases: {$convert:
{input:"$new_cases", to: "double"}}}}
        ]
        as: "new_cases_after_60"
      }
   }
   {$lookup:
      {
        from: "covid19data",
        localField: "120daysAfter"
        foreignField: "date"
        pipeline:
        [
          {$match: {"location" :"Germany"}},
          {$project: {_id:0, date: {$convert:
{input: "$date", to: "date"}},
                     new_cases: {$convert:
{input:"$new_cases", to: "double"}}}}
        ]
        as: "new_cases_after_120"
      }
   },
   {$project: {_id:0, date: 1,
new_cases_after_21: 1,
new_cases_after_60: 1,
new_cases_after_120: 1,
accumulated_vaccinations:1}}
])
```

| | |
|---|---|
| **186 Docs** | |

# 5. Comparison between Models

## 5.1 Construction

The relational data model contains 13 different schemas, consisting of 3 static and 10 dynamic tables, which were all constructed from the initial 3 datasets provided. 10 of the schemas are in a one to many relation with countrydata, with country_healthdata having a one to one relation. There is also country_sources which is in a one to many relation with country_vaccinations. The non-relational data model contains three collections.

## 5.2 Comparison between Relational and Non-relational Database Models

There are numerous distinctions between relational and non-relational databases. These differences are crucial when deciding on the best data management system for WHO. The table below summarizes the main differences between the models.

| **Relational Database (MySQL)** | **Non-relational Database (MongoDB)** |
|---|---|
| Schema/Structure | |
| Has predefined schema.<br><br>It is table based, with clearly defined rows and columns. Every row in the table must have the same columns.<br><br>Data records make up the finite rows of the table. | Has dynamic schema for unstructured data.<br><br>It can be document based, key-value pairs, graph databases or wide-column stores.<br><br>No standard schema definitions to adhere to, very flexible. Could potentially lead to data inconsistency. |
| Scalability | |
| Mostly vertically scalable, which means that the load on a single server can be increased by increasing components like RAM, SSD, or CPU.<br><br>Data can be replicated via servers, where data from one server can be copied onto another replica Server. This enhances performance and provides backups of different databases. | Horizontally scalable, which means that they can handle increased traffic by adding more servers to the database.<br><br>Have the ability to become larger and much more powerful. This makes them the preferred choice for large or constantly evolving data sets. Scaled by increasing the database servers in the pool of resources to reduce the load. |

<table>
<tr><td colspan="2">Flexibility</td></tr>
<tr><td>SQL databases are rigid and must be defined before use.<br><br>They are inflexible and modifications are often difficult and labor-intensive.</td><td>NoSQL databases are less structured and confined.<br><br>They are dynamic in their abilities to handle different types of data. This flexibility allows users to continuously add new features and functionality into their systems.</td></tr>
<tr><td colspan="2">Performance & Speed</td></tr>
<tr><td>Gets slower when the database size gets larger.<br><br>It is best used for databases with a smaller volume.</td><td>Able to handle large unstructured data much faster.<br><br>Has the ability to query in a manner that is capable of handling a greater workload.</td></tr>
<tr><td colspan="2">CAP (Consistency, Availability and Partition Tolerance) Theorem</td></tr>
<tr><td colspan="2"><u>Consistency</u><br>The nodes in the system will have the same copies of a replicated data item visible for various transactions.<br><br><u>Availability</u><br>Each read or write operation for a data item will either be processed successfully or will receive a message that the operation cannot be completed.<br><br><u>Partition Tolerance</u><br>The system can continue operating and functioning, while upholding its consistency in spite of network partitions between the nodes.<br><br>CAP theorem states that it is not possible to fulfil all three of the desirable properties at the same time in a distributed system with data replication.</td></tr>
<tr><td>Delivers consistency and availability across all nodes, but not partition tolerance.<br><br>The data presented will be consistent between all working nodes. This allows users to make any read or write request from any node, while ensuring the data is consistent.</td><td>Delivers consistency and partition tolerance at the expense of availability.<br><br>Consistent view of the database will be available for all the users to see.<br><br>With Partition tolerance, this means that the database is able to continue functioning despite any number of breakdowns between the nodes.</td></tr>
</table>

# 6. Recommendations

Selecting the right database is a relatively long-term commitment and hence is a pivotal decision for any organisation. To determine which database is the most suitable for WHO, we will have to consider a few fundamental factors:

1. What type of data will we be analysing?
2. How much data are we dealing with?
3. How much resources can we devote to the setup and maintenance of the database?
4. Do we need real-time data?

WHO provides an abundance of data on COVID-19 such as each country's new cases, confirmed cases, deaths and many more. On top of that, as part of extensive efforts to tackle the COVID-19 situation, WHO is working tirelessly with partners to develop, manufacture and deploy safe and effective vaccines. This is to ensure fair and equitable access to safe and effective vaccines, starting with the most vulnerable. WHO's COVID-19 dashboard features the number of vaccine doses administered globally, with more detail provided on the dedicated COVID-19 vaccination dashboard. After much consideration, we recommend that WHO implement non-relational databases as it is better suited to their needs.

## 6.1 Data Types

When it comes to the nature of data, we have to consider whether our data fits comfortably in rows and columns, or is it better suited in a more flexible space. WHO manages and maintains a wide range of data from many different sources. This might result in inconsistent rows and columns for different datasets. NoSQL is dynamic in its ability to handle different types of data. This flexibility will allow WHO to continuously add new features and functionality into their dashboards.

## 6.2 Amount of Data

The bigger the data set, the more likely a non-relational database is a better fit. WHO deals with a large volume of data, since information on vaccinations and COVID-19 cases are updated on a daily basis. The data available is also all encompassing, with information about all countries and all vaccines. Non-relational databases have the capacity to accommodate such volumes of such data as it can store unlimited sets of data with any type and have the flexibility to change the data type. Since they are horizontally scalable, they can handle increased traffic simply by adding more servers to the database. They possess the ability to become larger and much more powerful. Therefore, it is simple to add new analytic approaches or metrics during this pandemic. This scalability is also largely inexpensive, where WHO can scale out architecture efficiently without expensive overhead.

## 6.3 Capacity

To reap the benefits of using non-relational databases, it requires more extensive management. Non-relational databases may require more programming knowledge, which in turn means that the management team may have to possess proficient coding skills. Non-relational databases also take more time to manage, and hence requires a bigger team.

However, we feel that this is not a problem for WHO. WHO is a reputable organisation with extremely meaningful vision and goals. There are many talented professionals around the world who are always looking to contribute to this important mission. Currently, WHO employs 7,000 people in 149 countries and regions to carry out its principles. Overall, our team believes that WHO has the capacity and manpower necessary to employ non-relational databases.

## 6.4 Real-time Data

When it comes to COVID-19 data, real-time data is paramount for decision making by many stakeholders including world leaders, healthcare organisations...etc. Non-relational databases are able to store and process real-time data. They support ad-hoc queries for real-time analytics. An ad-hoc query is a short-lived command that provides different results for queries being executed. MongoDB supports ad-hoc queries that can be updated in real-time using MongoDB Query Language (MQL). Hence, MongoDB, a document-oriented and flexible schema database, is the best choice for enterprise applications that require real-time analytics. Since the dashboards are updated daily, using MongoDB allows WHO to query in a manner that is capable of handling a greater workload.

## 7. Limitations and Future Developments

To help WHO paint a more holistic picture, our team recognises that there are inevitable tradeoffs in our proposed recommendation that have to be properly evaluated.

Though non-relational databases are designed to tackle large amounts of data of varying nature, it is not as efficient or extensive as compared to relational databases. This could be mainly attributed to the lack of consistency found in non-relational databases. As a result, more work has to be performed to custom the queries for data retrieval, manipulation and exploration. Furthermore, since it is relatively new and with a lack of support, there are limited features which are usually present in relational databases.

As a result, perhaps WHO could look into taking on a multi-faceted approach once a strong foundation is laid down. This entails implementing a mixture of both relational and non-relational databases to fully capitalize on both benefits. Hybrid systems focus on combining the two databases. This allows for joins and SQL systems to be added onto NoSQL features. The capabilities of horizontal scaling are now potentially possible while achieving consistency at the same time. SQL is an incredibly strong language for expressing data manipulation instructions. NoSQL stores could implement a SQL-based query engine if necessary. We could thus tap on the ability to store large amounts of unstructured data from different sources. However, we should keep in mind that this can only be executed after the necessary infrastructures and systems are put in place as it might potentially be confusing to adopt more than one database model.

Additionally, WHO could consider implementing cloud computing database systems. There are a myriad of benefits that come with cloud computing. Firstly, cloud databases are efficient and flexible. There are no restrictions on the ability to expand, and hence, this makes scaling and managing the database easier. Secondly, the cost of adoption is cheaper than expanding existing on-site server capability. Lastly, the cloud also offers more robust and comprehensive security which WHO could rely on.

## 8. Conclusion

To sum up, our team has decided that a non-relational database would be more suitable for WHO during this global pandemic, due to the nature of the data we are dealing with. With the provided solutions for queries, we can sieve out vital information necessary for decision making by various stakeholders.

On top of looking at relational and non-relational databases as separate entities, WHO could look at them in unison and implement a hybrid approach. Also, WHO could also consider adopting cloud computing database systems because of the extensive benefits they offer.

# 9. References


Benefits and challenges of cloud databases. MongoDB. (n.d.). Retrieved November 11, 2021, from https://www.mongodb.com/cloud-database/benefits.

Higginbotham, J. (2020, November 5). 3 tips for selecting The right database for your app. Cloud 66 blog on Rails, Docker, Kubernetes and Node.js. Retrieved November 11, 2021, from https://blog.cloud66.com/3-tips-for-selecting-the-right-database-for-your-app/.

How to Choose the Right Database. Amazon. (2019). Retrieved November 11, 2021, from https://aws.amazon.com/startups/start-building/how-to-choose-a-database/

IBM Cloud Education. (n.d.). Cap Theorem. IBM. Retrieved November 11, 2021, from https://www.ibm.com/sg-en/cloud/learn/cap-theorem.

Issac, L. P. (2014, January 14). SQL VS NoSQL database differences explained with few example DB. The Geek Stuff. Retrieved November 11, 2021, from https://www.thegeekstuff.com/2014/01/sql-vs-nosql-db.

MongDB. (n.d.). Indexes¶. Indexes - MongoDB Manual. Retrieved November 11, 2021, from https://docs.mongodb.com/manual/indexes/.

MongoDB vs mysql: Compare the differences based on parameters. Simform Blog. (2021, October 25). Retrieved November 11, 2021, from https://www.simform.com/blog/mongodb-vs-mysql-databases/#section5.

NoSQL advantages and disadvantages. Blue Claw Database Developer Resource. (n.d.). Retrieved November 11, 2021, from https://blueclawdb.com/nosql/nosql-advantages-disadvantages/.

Pawlan, D. (2021, April 29). Relational vs. Non-Relational Database: Pros & Cons: The aloa blog. RSS. Retrieved November 11, 2021, from https://aloa.co/blog/relational-vs-non-relational-database-pros-cons.

Rich Edwards, & Edwards, R. (2019, September 29). SQL versus NOSQL: Pros and cons. Datastax. Retrieved November 11, 2021, from https://www.datastax.com/blog/sql-vs-nosql-pros-cons.

Shiff, L., & Rowe, W. (2018, March 5). SQL VS nosql databases: What's the difference? BMC Blogs. Retrieved November 11, 2021, from https://www.bmc.com/blogs/sql-vs-nosql/.

SQL vs. NoSQL databases: What's the difference? IBM. (n.d.). Retrieved November 11, 2021, from https://www.ibm.com/cloud/blog/sql-vs-nosql.

The Cap Theorem in DBMS. GeeksforGeeks. (2021, June 15). Retrieved November 11, 2021, from https://www.geeksforgeeks.org/the-cap-theorem-in-dbms/.

Waterman, T. (2020, May 3). SQL vs no-SQL: Which is better for analytics? Corbett Analytics. Retrieved November 11, 2021, from https://www.corbettanalytics.com/no-sql-vs-sql/.

What is a non-relational database? MongoDB. (n.d.). Retrieved November 11, 2021, from https://www.mongodb.com/databases/non-relational.

What is a relational database? What Is a Relational Database | Oracle Singapore. (n.d.). Retrieved November 11, 2021, from https://www.oracle.com/sg/database/what-is-a-relational-database/.

What's the difference? relational vs non-relational databases. Logi Analytics. (n.d.). Retrieved November 11, 2021, from https://www.logianalytics.com/relational-vs-non-relational-databases/.

Wikimedia Foundation. (2021, November 9). World Health Organization. Wikipedia. Retrieved November 11, 2021, from https://en.wikipedia.org/wiki/World_Health_Organization.

World Health Organization. (n.d.). About WHO. World Health Organization. Retrieved November 11, 2021, from https://www.who.int/about/.

Wu, J. (2019, November 28). Choosing the right database. Medium. Retrieved November 11, 2021, from https://towardsdatascience.com/choosing-the-right-database-c45cd3a28f77.