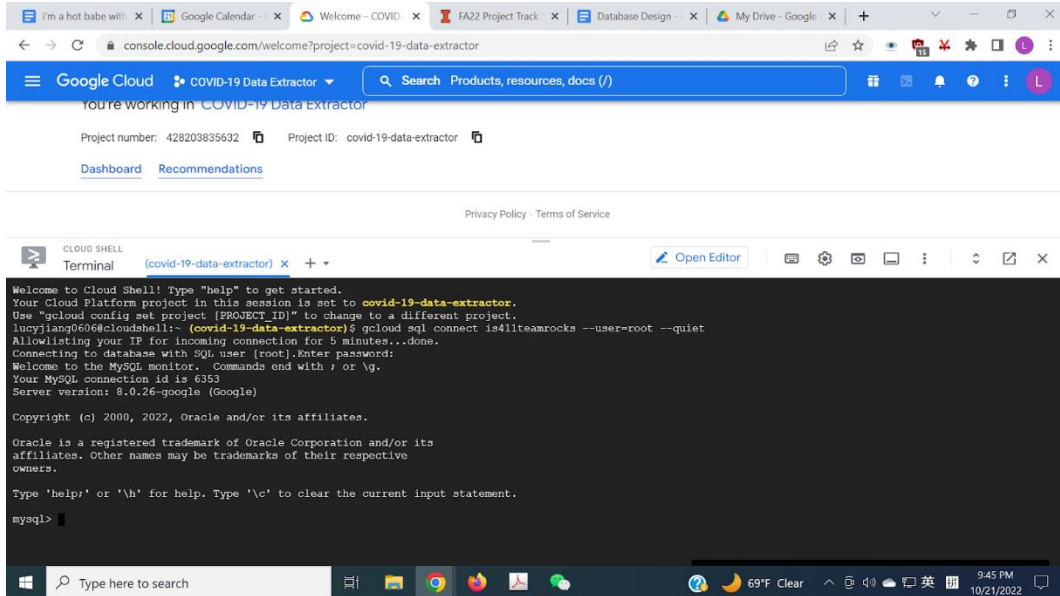


Stage 3: Database Implementation and Indexing

- Create the Database on Google Cloud Platform (GCP)

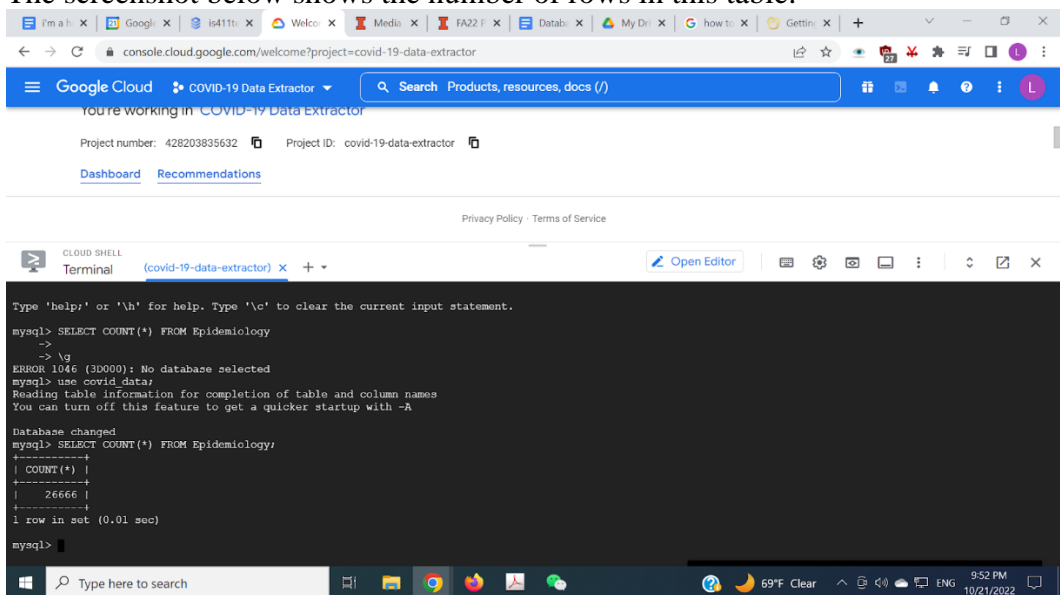
Connection Screenshot



DDL Commands we used to create the each table:

```
CREATE TABLE Epidemiology(  
    Region VARCHAR(25),  
    Date VARCHAR(20),  
    Cumulative_confirmed REAL,  
    Cumulative_deceased REAL,  
    Cumulative_recovered REAL,  
    Cumulative_tested REAL,  
    PRIMARY KEY(Region, Date)  
);
```

The screenshot below shows the number of rows in this table:



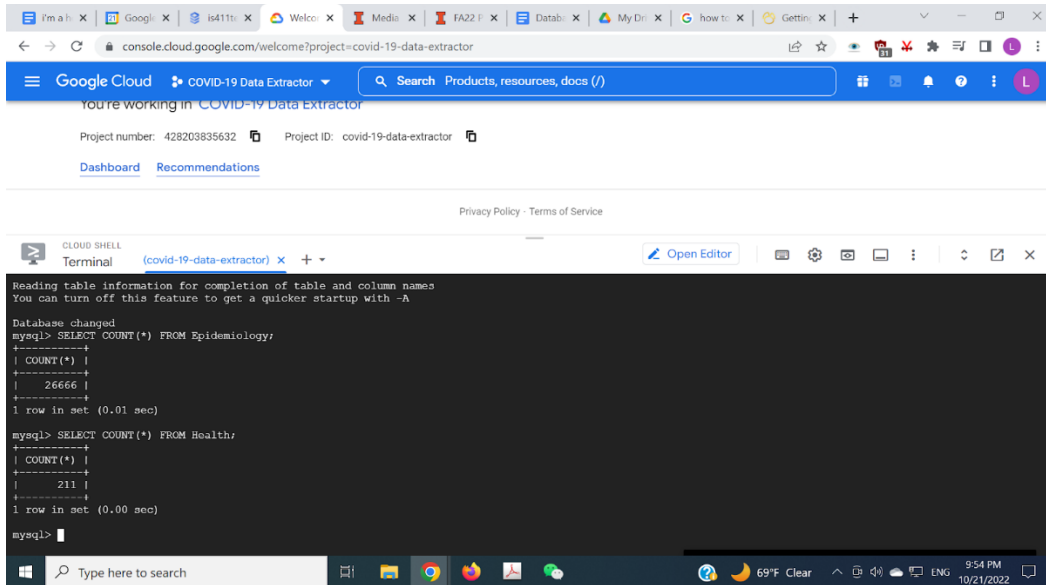
```
CREATE TABLE Health(  
    Region VARCHAR(25),  
    Life_expectancy REAL,  
    Smoking_prevalence REAL,
```

```

Diabetes_prevalence REAL,
Health_expenditure REAL,
PRIMARY KEY (Region)
);

```

The screenshot below shows the number of rows in this table:



The screenshot shows the Google Cloud console interface for the 'COVID-19 Data Extractor' project. A terminal window is open, displaying the following SQL queries and their results:

```

mysql> SELECT COUNT(*) FROM Epidemiology;
+-----+
| COUNT(*) |
+-----+
| 26666 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Health;
+-----+
| COUNT(*) |
+-----+
| 211 |
+-----+
1 row in set (0.00 sec)

mysql>

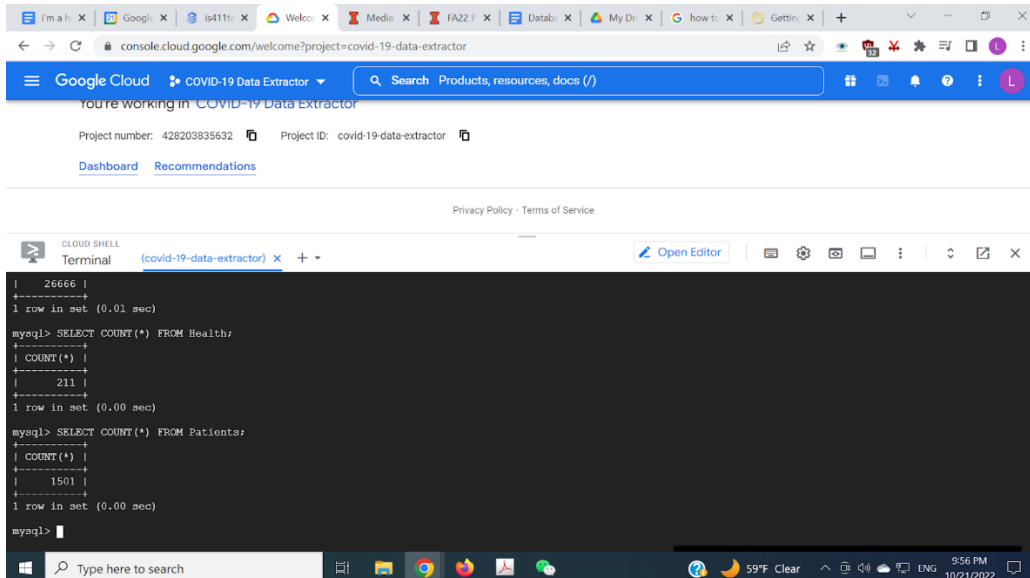
```

```

CREATE TABLE Patients (
  PatientName VARCHAR(255),
  PatientId INT,
  Region VARCHAR(25),
  PRIMARY KEY (PatientId)
);

```

The screenshot below shows the number of rows in this table:



The screenshot shows the Google Cloud console interface for the 'COVID-19 Data Extractor' project. A terminal window is open, displaying the following SQL queries and their results:

```

mysql> SELECT COUNT(*) FROM Health;
+-----+
| COUNT(*) |
+-----+
| 211 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Patients;
+-----+
| COUNT(*) |
+-----+
| 1501 |
+-----+
1 row in set (0.00 sec)

mysql>

```

```
CREATE TABLE Hospitalizations(
  Region VARCHAR(25),
  Date VARCHAR(20),
  Current_ventilator_patients INT,
  Current_hospitalized_patients INT,
  Current_intensive_care_patients INT,
  PRIMARY KEY (Region, Date)
);
```

The screenshot below shows the number of rows in this table:

The screenshot shows the Google Cloud console interface for the 'COVID-19 Data Extractor' project. A terminal window is open, displaying the following MySQL queries and results:

```
mysql> SELECT COUNT(*) FROM Patients;
+-----+
| COUNT(*) |
+-----+
| 1501 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Hospitalizations;
+-----+
| COUNT(*) |
+-----+
| 1768486 |
+-----+
1 row in set (0.47 sec)
```

```
CREATE TABLE Vaccinations(
  PatientId INT,
  Fully_vaccinated INT,
  Have_booster INT,
  PRIMARY KEY (PatientId)
);
```

The screenshot shows the Google Cloud console interface for the 'COVID-19 Data Extractor' project. A terminal window is open, displaying the following MySQL queries and results:

```
mysql> SELECT COUNT(*) FROM Hospitalizations;
+-----+
| COUNT(*) |
+-----+
| 1768486 |
+-----+
1 row in set (0.47 sec)

mysql> SELECT COUNT(*) FROM Vaccinations;
+-----+
| COUNT(*) |
+-----+
| 1501 |
+-----+
1 row in set (0.00 sec)
```

• Advanced Query

Advanced Query1:

1:Description:

Find the number of patients who are fully vaccinated by region

2:SQL query:

```
select count(p.PatientId), p.Region
from Patients as p natural join Vaccinations as v
where v.Fully_vaccinated = 1 group by p.Region limit 15;
```

3:Query Result:

Top 15 data:

```
mysql> select count(p.PatientId), p.Region from Patients as p natural join Vaccinations as v where v.Fully_vaccinated = 1 group by p.Region limit 15;
+-----+-----+
| count(p.PatientId) | Region |
+-----+-----+
| 4 | MH |
| 6 | CV |
| 5 | CZ |
+-----+-----+
15 rows in set (0.01 sec)
```

First Try: Use region (Improved)

The reason of choosing region is that region is an attribution used in "group by" function

The performance improved, this is because we index the region, so the timer aggregation decreased, which lead a small improvement.

```
+-----+-----+
| 4 | MH |
| 6 | CV |
| 5 | CZ |
+-----+-----+
1 row in set (0.01 sec)
```

Second Try: Use Fully_vaccinated (Worse)

The reason of choosing Fully_vaccinated is that it used in "where" function

The performance is worse, this is because we index the fully vaccinated attributed, so it used more time in aggregation and nested loop part.

```
+-----+-----+
| 4 | MH |
| 6 | CV |
| 5 | CZ |
+-----+-----+
1 row in set (0.00 sec)
```

Advanced Query2:

1:Description:

Find maximum number of COVID-19 positive cases which require a ventilator in areas whose smoking prevalence is greater than average.

2:SQL query:

```
select a1.Region, a1.Smoking_prevalence ,b1. ans as Max_ventilator_number
from
(select Region,Smoking_prevalence from Health where Smoking_prevalence > (select
avg(Smoking_prevalence)
from Health where Smoking_prevalence<>0))as a1
NATURAL JOIN
( select Region,max(Current_ventilator_patients) as ans from Hospitalizations group by
Region )as b1
limit 15;
```

3:Query Result:

Top 15 data:

```
mysql> select a1.Region, a1.Smoking_prevalence ,b1. ans as Max_ventilator_nu
from Health where Smoking_prevalence<>0))as a1 NATURAL JOIN ( select Region
+-----+-----+-----+-----+
| Region | Smoking_prevalence | Max_ventilator_number |
+-----+-----+-----+-----+
| AR     | 21.8               | 0                     |
| BE     | 28.2               | 7461                  |
| CH     | 25.7               | 0                     |
| CZ     | 34.3               | 9551                  |
| ES     | 29.3               | 0                     |
| FR     | 32.7               | 33466                 |
| GB     | 22.3               | 39254                 |
| IT     | 23.7               | 38507                 |
| JP     | 22.1               | 0                     |
| LU     | 23.5               | 0                     |
| PH     | 24.3               | 0                     |
| PT     | 22.7               | 1302                  |
| RO     | 29.7               | 0                     |
| SI     | 22.5               | 1324                  |
| US     | 21.8               | 154513                |
+-----+-----+-----+-----+
15 rows in set (0.75 sec)
```

4:Index

Before Index:

```
+-----+
| -> Limit: 15 row(s) (cost=12451426.86 rows=15) (actual time=786.080..786.231 rows=15 loops=1)
|   -> Nested loop inner join (cost=12451426.86 rows=123056184) (actual time=786.079..786.229 rows=15 loops=1)
|     -> Filter: (Health.Smoking_prevalence > (select #3)) (cost=7.28 rows=70) (actual time=0.189..0.255 rows=72 loops=1)
|       -> Table scan on Health (cost=7.28 rows=211) (actual time=0.052..0.097 rows=198 loops=1)
|       -> Select #3 (subquery in condition; run only once)
|         -> Aggregate: avg(Health.Smoking_prevalence) (cost=40.34 rows=190) (actual time=0.129..0.130 rows=1 loops=1)
|           -> Filter: (Health.Smoking_prevalence <> 0) (cost=21.35 rows=190) (actual time=0.030..0.114 rows=146 loops=1)
|             -> Table scan on Health (cost=21.35 rows=211) (actual time=0.030..0.091 rows=211 loops=1)
|   -> Index lookup on b1 using <auto_key0> (Region=Health.Region) (actual time=0.001..0.001 rows=0 loops=72)
|     -> Materialize (cost=526500.20..526500.20 rows=1749789) (actual time=785.956..785.960 rows=7043 loops=1)
|       -> Group aggregate: max(Hospitalizations.Current_ventilator_patients) (cost=351521.30 rows=1749789) (actual time=0.694..771.286 rows=7043)
|         -> Index scan on Hospitalizations using PRIMARY (cost=176542.40 rows=1749789) (actual time=0.045..498.498 rows=1768486 loops=1)
```

First Try: Use Hospitalizations.Region (Improved)

The reason of choosing region is that region is an attribution used in "group by" function

We can notice that the time of Group aggregate decreased from 0.694 to 0.357, which means that this index improve the performance of aggregate and also lead to an overall improved.

```
| -> Limit: 15 row(s) (cost=12451426.86 rows=15) (actual time=739.301..739.445 rows=15 loops=1)
-> Nested loop inner join (cost=12451426.86 rows=123056184) (actual time=739.300..739.443 rows=15 loops=1)
-> Filter: (Health.Smoking_prevalence > (select #3)) (cost=7.28 rows=70) (actual time=0.119..0.181 rows=72 loops=1)
-> Table scan on Health (cost=7.28 rows=211) (actual time=0.037..0.077 rows=198 loops=1)
-> Select #3 (subquery in condition; run only once)
-> Aggregate: avg(Health.Smoking_prevalence) (cost=40.34 rows=190) (actual time=0.076..0.077 rows=1 loops=1)
-> Filter: (Health.Smoking_prevalence <> 0) (cost=21.35 rows=190) (actual time=0.016..0.066 rows=146 loops=1)
-> Table scan on Health (cost=21.35 rows=211) (actual time=0.015..0.049 rows=211 loops=1)
-> Index lookup on b1 using <auto_key0> (Region=Health.Region) (actual time=0.001..0.001 rows=0 loops=72)
-> Materialize (cost=526500.20..526500.20 rows=1749789) (actual time=739.245..739.249 rows=7043 loops=1)
-> Group aggregate: max(Hospitalizations.Current_ventilator_patients) (cost=351521.30 rows=1749789) (actual time=0.357..724.569 rows=7043 loops=1)
-> Index scan on Hospitalizations using PRIMARY (cost=176542.40 rows=1749789) (actual time=0.026..468.265 rows=1768486 loops=1)
|
```

Second Try: Use Health.Region (No difference)

Set a index on the region of Health do not change the performance obviously. Actually, we can use show index from Health to look up all the index of table Health, and we will find that Region is the primary key of Health and MySQL will automatically create a primary key index. This makes our duplicate creation meaningless.

```
-> Limit: 15 row(s) (cost=12451426.86 rows=15) (actual time=736.123..736.272 rows=15 loops=1)
-> Nested loop inner join (cost=12451426.86 rows=123056184) (actual time=736.123..736.270 rows=15 loops=1)
-> Filter: (Health.Smoking_prevalence > (select #3)) (cost=7.28 rows=70) (actual time=0.123..0.185 rows=72 loops=1)
-> Table scan on Health (cost=7.28 rows=211) (actual time=0.036..0.078 rows=198 loops=1)
-> Select #3 (subquery in condition; run only once)
-> Aggregate: avg(Health.Smoking_prevalence) (cost=40.34 rows=190) (actual time=0.081..0.081 rows=1 loops=1)
-> Filter: (Health.Smoking_prevalence <> 0) (cost=21.35 rows=190) (actual time=0.015..0.068 rows=146 loops=1)
-> Table scan on Health (cost=21.35 rows=211) (actual time=0.015..0.049 rows=211 loops=1)
-> Index lookup on b1 using <auto_key0> (Region=Health.Region) (actual time=0.001..0.001 rows=0 loops=72)
-> Materialize (cost=526500.20..526500.20 rows=1749789) (actual time=736.067..736.071 rows=7043 loops=1)
-> Group aggregate: max(Hospitalizations.Current_ventilator_patients) (cost=351521.30 rows=1749789) (actual time=0.376..721.959 rows=7043 loops=1)
-> Index scan on Hospitalizations using PRIMARY (cost=176542.40 rows=1749789) (actual time=0.026..472.271 rows=1768486 loops=1)
-----+
```

Third Try: Use Health. Smoking_prevalence (Worse)

Set a index on the Smoking_prevalence of Health made the performance worse. This may be because we compare the value of Smoking_prevalence in the expression, which makes the index lose its effect, and the introduction of the index also leads to the deterioration of the overall performance, which leads to such a worse result.

```
-----+
| -> Limit: 15 row(s) (cost=13101871.37 rows=15) (actual time=775.858..776.005 rows=15 loops=1)
-> Nested loop inner join (cost=13101871.37 rows=129484386) (actual time=775.856..776.002 rows=15 loops=1)
-> Filter: (Health.Smoking_prevalence > (select #3)) (cost=15.25 rows=74) (actual time=0.015..0.044 rows=58 loops=1)
-> Index range scan on Health using emp_name_index1 (cost=15.25 rows=74) (actual time=0.015..0.031 rows=58 loops=1)
-> Select #3 (subquery in condition; run only once)
-> Aggregate: avg(Health.Smoking_prevalence) (cost=44.74 rows=147) (actual time=0.082..0.082 rows=1 loops=1)
-> Filter: (Health.Smoking_prevalence <> 0) (cost=30.04 rows=147) (actual time=0.034..0.070 rows=146 loops=1)
-> Index range scan on Health using emp_name_index1 (cost=30.04 rows=147) (actual time=0.032..0.056 rows=146 loops=1)
-> Index lookup on b1 using <auto_key0> (Region=Health.Region) (actual time=0.002..0.002 rows=0 loops=58)
-> Materialize (cost=526500.20..526500.20 rows=1749789) (actual time=775.941..775.946 rows=7043 loops=1)
-> Group aggregate: max(Hospitalizations.Current_ventilator_patients) (cost=351521.30 rows=1749789) (actual time=0.468..760.751 rows=7043 loops=1)
-> Index scan on Hospitalizations using PRIMARY (cost=176542.40 rows=1749789) (actual time=0.035..490.683 rows=1768486 loops=1)
|
-----+
```