DDL Commands:

```
CREATE TABLE UserProfile(
userFirstName VARCHAR(100) NOT NULL,
userLastName VARCHAR(100) NOT NULL,
destinationCity VARCHAR(100) NOT NULL,
email VARCHAR(100) NOT NULL,
password VARCHAR(100) NOT NULL,
PRIMARY KEY(email),
FOREIGN KEY(destinationCity) REFERENCES AirportData(airportCity)
);

CREATE TABLE CountryData(
country VARCHAR(100) NOT NULL,
countryCode VARCHAR(3) NOT NULL,
population INT,
region VARCHAR(100),
PRIMARY KEY(country)
);

CREATE TABLE AirportData(
country VARCHAR(100) NOT NULL,
airportCity VARCHAR(100) NOT NULL,
airportName VARCHAR(100) NOT NULL,
airportCode VARCHAR(3) NOT NULL,
PRIMARY KEY(airportCode),
FOREIGN KEY(country) REFERENCES CountryData(country),
UNIQUE(airportCity),
UNIQUE(airportName)
);

CREATE TABLE CovidCases(
country VARCHAR(100) NOT NULL,
countryCode VARCHAR(3),
date TIMESTAMP NOT NULL,
newCaseNumber INT,
newDeathNumber INT,
PRIMARY KEY(date, country),
FOREIGN KEY(country) REFERENCES CountryData(country)
);

CREATE TABLE Vaccination(
country VARCHAR(100) NOT NULL,
countryCode VARCHAR(3),
```

```
date TIMESTAMP NOT NULL,
dailyVaccinationNumber INT,
PRIMARY KEY(date, country),
FOREIGN KEY(country) REFERENCES CountryData(country)
 );

CREATE TABLE Hospitalization(
country VARCHAR(100) NOT NULL,
countryCode VARCHAR(3),
date TIMESTAMP NOT NULL,
patientNumber INT,
PRIMARY KEY(date, country),
FOREIGN KEY(country) REFERENCES CountryData(country)
);

CREATE TABLE Testing(
country VARCHAR(100) NOT NULL,
countryCode VARCHAR(3),
date TIMESTAMP NOT NULL,
newTestNumber INT,
PRIMARY KEY(date, country),
FOREIGN KEY(country) REFERENCES CountryData(country)
);

CREATE TABLE Ratings(
airportName VARCHAR(100) NOT NULL,
email VARCHAR(100) NOT NULL,
rating INT,
review TEXT,
PRIMARY KEY(airportName, email),
FOREIGN KEY(email) REFERENCES UserProfile(email),
FOREIGN KEY(airportName) REFERENCES AirportData(airportName)
);
```

Proof of Database:

```
mysql> SELECT table_name, table_rows FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'AbDB';
+-----------------+------------+
| TABLE_NAME      | TABLE_ROWS |
+-----------------+------------+
| AirportData     |         50 |
| CountryData     |        224 |
| CovidCases      |     158812 |
| Hospitalization |      30595 |
| Ratings         |          0 |
| Testing         |      61187 |
| UserProfile     |          0 |
| Vaccination     |      74315 |
+-----------------+------------+
8 rows in set (0.02 sec)
```

Advanced SQL Commands:

```
mysql> SELECT country, SUM(newCaseNumber)/population as rate FROM CountryData NATURAL JOIN CovidCases GROUP BY country ORDER BY rate DESC LIMIT 15;
+-----------------+--------+
| country         | rate   |
+-----------------+--------+
| Bahrain         | 0.7844 |
| Israel          | 0.6007 |
| Iceland         | 0.5885 |
| Andorra         | 0.5578 |
| Denmark         | 0.5551 |
| San Marino      | 0.5155 |
| Cyprus          | 0.4998 |
| Seychelles      | 0.4905 |
| Maldives        | 0.4865 |
| Aruba           | 0.4708 |
| Slovenia        | 0.4707 |
| Netherlands     | 0.4681 |
| Liechtenstein   | 0.4606 |
| Switzerland     | 0.4492 |
| Cayman Islands  | 0.4476 |
+-----------------+--------+
15 rows in set (0.26 sec)
```

```
mysql> SELECT airportName as 'Airport', country as 'Country', rate/3 AS 'Vaccination Rate' FROM (SELECT country as c, SUM(dailyVaccinationNumber)/population as rat
e FROM CountryData NATURAL JOIN Vaccination GROUP BY country) AS temp, AirportData WHERE rate > 0.5 AND country = c LIMIT 15;
+-----------------------------------------------+-----------+------------------+
| Airport                                       | Country   | Vaccination Rate |
+-----------------------------------------------+-----------+------------------+
| Sydney Kingsford-Smith Airport                | Australia |       0.91413333 |
| Toronto Pearson International Airport         | Canada    |       0.82426667 |
| Guangzhou Baiyun International Airport        | China     |       0.81830000 |
| Chongqing Jiangbei International Airport      | China     |       0.81830000 |
| Chengdu Shuangliu International Airport       | China     |       0.81830000 |
| Kunming Changshui International Airport       | China     |       0.81830000 |
| Beijing Capital International Airport         | China     |       0.81830000 |
| Shanghai Pudong International Airport         | China     |       0.81830000 |
| Shanghai Hongqiao International Airport       | China     |       0.81830000 |
| Shenzhen Bao'an International Airport         | China     |       0.81830000 |
| Xi'an Xianyang International Airport          | China     |       0.81830000 |
| Hong Kong International Airport               | Hong Kong |       0.68953333 |
| Chhatrapati Shivaji Maharaj International Airport | India     |       0.55183333 |
| Indira Gandhi International Airport           | India     |       0.55183333 |
| Soekarno-Hatta International Airport          | Indonesia |       0.49683333 |
+-----------------------------------------------+-----------+------------------+
15 rows in set (0.14 sec)
```

Indexing:

<u>First advanced SQL command:</u>

PRIMARY index:

```
| EXPLAIN




                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=350.024..350.026 rows=15 loops=1)
    -> Sort: (sum(CovidCases.newCaseNumber) / CountryData.population) DESC, limit input to 15 row(s) per chunk  (actual time=350.023..350.024 rows=15 loops=1)
        -> Stream results  (cost=54752.55 rows=15203) (actual time=2.140..349.488 rows=196 loops=1)
            -> Group aggregate: sum(CovidCases.newCaseNumber)  (cost=54752.55 rows=15203) (actual time=2.135..349.076 rows=196 loops=1)
                -> Nested loop inner join  (cost=53232.29 rows=15203) (actual time=0.314..317.711 rows=140504 loops=1)
                    -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.050..0.282 rows=224 loops=1)
                    -> Filter: (CovidCases.countryCode = CountryData.countryCode)  (cost=169.70 rows=68) (actual time=0.265..1.369 rows=627 loops=224)
                        -> Index lookup on CovidCases using country (country=CountryData.country)  (cost=169.70 rows=679) (actual time=0.172..1.267 rows=689 loops=
224)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
1 row in set (0.35 sec)
```

PRIMARY and population index:

```
---------------------------------------------------------+
| EXPLAIN




                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=279.917..279.919 rows=15 loops=1)
    -> Sort: (sum(CovidCases.newCaseNumber) / CountryData.population) DESC, limit input to 15 row(s) per chunk  (actual time=279.916..279.917 rows=15 loops=1)
        -> Stream results  (cost=54752.55 rows=15203) (actual time=2.177..279.544 rows=196 loops=1)
            -> Group aggregate: sum(CovidCases.newCaseNumber)  (cost=54752.55 rows=15203) (actual time=2.172..279.250 rows=196 loops=1)
                -> Nested loop inner join  (cost=53232.29 rows=15203) (actual time=0.341..255.006 rows=140504 loops=1)
                    -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.041..0.205 rows=224 loops=1)
                    -> Filter: (CovidCases.countryCode = CountryData.countryCode)  (cost=169.70 rows=68) (actual time=0.218..1.098 rows=627 loops=224)
                        -> Index lookup on CovidCases using country (country=CountryData.country)  (cost=169.70 rows=679) (actual time=0.137..1.019 rows=689 loops=
224)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
1 row in set (0.29 sec)
```

PRIMARY and newCaseNumber index:

```
| EXPLAIN




                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=276.868..276.870 rows=15 loops=1)
    -> Sort: (sum(CovidCases.newCaseNumber) / CountryData.population) DESC, limit input to 15 row(s) per chunk  (actual time=276.867..276.868 rows=15 loops=1)
        -> Stream results  (cost=54752.55 rows=15203) (actual time=2.185..276.532 rows=196 loops=1)
            -> Group aggregate: sum(CovidCases.newCaseNumber)  (cost=54752.55 rows=15203) (actual time=2.180..276.270 rows=196 loops=1)
                -> Nested loop inner join  (cost=53232.29 rows=15203) (actual time=0.312..252.291 rows=140504 loops=1)
                    -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.034..0.192 rows=224 loops=1)
                    -> Filter: (CovidCases.countryCode = CountryData.countryCode)  (cost=169.70 rows=68) (actual time=0.216..1.086 rows=627 loops=224)
                        -> Index lookup on CovidCases using country (country=CountryData.country)  (cost=169.70 rows=679) (actual time=0.135..1.006 rows=689 loops=
224)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
1 row in set (0.28 sec)
```

PRIMARY, population, and newCaseNumber index:

```
| EXPLAIN




                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=280.504..280.506 rows=15 loops=1)
    -> Sort: (sum(CovidCases.newCaseNumber) / CountryData.population) DESC, limit input to 15 row(s) per chunk  (actual time=280.504..280.505 rows=15 loops=1)
        -> Stream results  (cost=54752.55 rows=15203) (actual time=2.369..280.164 rows=196 loops=1)
            -> Group aggregate: sum(CovidCases.newCaseNumber)  (cost=54752.55 rows=15203) (actual time=2.364..279.902 rows=196 loops=1)
                -> Nested loop inner join  (cost=53232.29 rows=15203) (actual time=0.357..255.278 rows=140504 loops=1)
                    -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.046..0.200 rows=224 loops=1)
                    -> Filter: (CovidCases.countryCode = CountryData.countryCode)  (cost=169.70 rows=68) (actual time=0.217..1.090 rows=627 loops=224)
                        -> Index lookup on CovidCases using country (country=CountryData.country)  (cost=169.70 rows=679) (actual time=0.136..1.010 rows=689 loops=
224)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------+
1 row in set (0.28 sec)
```

We tried two types of indexing, trying them separately and together. In short, both of them helped, but one more than the other

Our first index was on the population in the Country Data table. This affected only 224 rows however when we compared the result of this indexing to no indexing, our performance had increased by 6 seconds. Instead of taking 0.35 seconds, it took 0.29 seconds.

Our second index was on the newCaseNumber in the Covid Cases table - this is the largest table in our query, with approximately 158000 rows so we expected to see some results. We implemented this index after dropping our first index on population. Just by itself, it had increased our performance significantly, from 0.35 seconds to 0.28 seconds

We tried both the indices together and the result was actually the same as just the second index on newCaseNumber - 0.28 seconds.

Both of them helped increase our performance however newCaseNumber helped more, presumably because of how large the Covid Cases table is. Additionally, because we are doing a sum on the newCaseNumbers, it is fairly complex, meaning indexing would help greatly. The index on population also helped because our query was just that large.

Our query groups by country which is the primary key for CountryData. This means MySQL already created the PRIMARY index for it so our original query was already fast.

In conclusion, we will index on just newCaseNumber, as that provided the best results.

<u>Second advanced SQL command:</u>

PRIMARY index:

```
mysql> show index from CountryData;
+-------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---
------+------------+
| Table       | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Vi
sible | Expression |
+-------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---
------+------------+
| CountryData |          0 | PRIMARY  |            1 | country     | A         |         224 |     NULL |   NULL |      | BTREE      |         |               | YE
S     | NULL       |
+-------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---
------+------------+
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN ANALYZE SELECT airportName as 'Airport', country as 'Country', rate/3 AS 'Vaccination Rate' FROM (SELECT country as c, SUM(dailyVaccinationNumber)/p
opulation as rate FROM CountryData NATURAL JOIN Vaccination GROUP BY country) AS temp, AirportData WHERE rate > 0.5 AND country = c LIMIT 15;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                               |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=116.55 rows=15) (actual time=148.342..148.444 rows=15 loops=1)
    -> Nested loop inner join  (cost=116.55 rows=22) (actual time=148.341..148.442 rows=15 loops=1)
        -> Table scan on temp  (cost=0.12..2.77 rows=22) (actual time=0.002..0.008 rows=59 loops=1)
            -> Materialize  (cost=106.15..108.81 rows=22) (actual time=148.316..148.326 rows=163 loops=1)
                -> Filter: ((sum(Vaccination.dailyVaccinationNumber) / CountryData.population) > 0.5)  (cost=103.79 rows=22) (actual time=1.893..148.074 rows=163 l
oops=1)
                    -> Group aggregate: sum(Vaccination.dailyVaccinationNumber)  (cost=103.79 rows=22) (actual time=1.075..147.902 rows=199 loops=1)
                        -> Nested loop inner join  (cost=101.55 rows=22) (actual time=0.099..134.311 rows=76149 loops=1)
                            -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.043..0.165 rows=224 loops=1)
                            -> Filter: (Vaccination.countryCode = CountryData.countryCode)  (cost=0.25 rows=0) (actual time=0.060..0.573 rows=340 loops=224)
                                -> Index lookup on Vaccination using country (country=CountryData.country)  (cost=0.25 rows=1) (actual time=0.016..0.530 rows=371 l
oops=224)
        -> Index lookup on AirportData using country (country=temp.c)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0 loops=59)
```

```
1 row in set (0.15 sec)
```

—--

PRIMARY and population index:

```
mysql> CREATE INDEX popindex ON CountryData(population);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT airportName as 'Airport', country as 'Country', rate/3 AS 'Vaccination Rate' FROM (SELECT country as c, SUM(dailyVaccinationNumber)/p
opulation as rate FROM CountryData NATURAL JOIN Vaccination GROUP BY country) AS temp, AirportData WHERE rate > 0.5 AND country = c LIMIT 15;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
```

```
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=37588.80 rows=15) (actual time=150.872..150.903 rows=15 loops=1)
    -> Nested loop inner join  (cost=37588.80 rows=366669) (actual time=150.871..150.901 rows=15 loops=1)
        -> Table scan on AirportData  (cost=5.25 rows=50) (actual time=0.019..0.026 rows=16 loops=1)
        -> Index lookup on temp using <auto_key0> (c=AirportData.country)  (actual time=0.001..0.001 rows=1 loops=16)
            -> Materialize  (cost=27156.68..27156.68 rows=7333) (actual time=150.865..150.868 rows=163 loops=1)
                -> Filter: ((sum(Vaccination.dailyVaccinationNumber) / CountryData.population) > 0.5)  (cost=26423.34 rows=7333) (actual time=2.128..150.372 rows=1
63 loops=1)
                    -> Group aggregate: sum(Vaccination.dailyVaccinationNumber)  (cost=26423.34 rows=7333) (actual time=1.187..150.124 rows=199 loops=1)
                        -> Nested loop inner join  (cost=25690.00 rows=7333) (actual time=0.253..136.443 rows=76149 loops=1)
                            -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.032..0.171 rows=224 loops=1)
                            -> Filter: (Vaccination.countryCode = CountryData.countryCode)  (cost=81.86 rows=33) (actual time=0.168..0.586 rows=340 loops=224)
                                -> Index lookup on Vaccination using country (country=CountryData.country)  (cost=81.86 rows=327) (actual time=0.132..0.540 rows=37
1 loops=224)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.16 sec)
```

———---

PRIMARY and dailyVaccinationNumber index:

```
mysql> CREATE INDEX dailyvacc ON Vaccination(dailyVaccinationNumber);
Query OK, 0 rows affected (0.50 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW INDEX FROM Vaccination;
+-------------+------------+-----------+--------------+------------------------+-----------+-------------+----------+--------+------+------------+---------+-------
-------+---------+------------+
| Table       | Non_unique | Key_name  | Seq_in_index | Column_name            | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_
comment | Visible | Expression |
+-------------+------------+-----------+--------------+------------------------+-----------+-------------+----------+--------+------+------------+---------+-------
-------+---------+------------+
| Vaccination |          0 | PRIMARY   |            1 | date                   | A         |         473 |     NULL |   NULL |      | BTREE      |         |
       | YES     | NULL       |
| Vaccination |          0 | PRIMARY   |            2 | country                | A         |       74316 |     NULL |   NULL |      | BTREE      |         |
       | YES     | NULL       |
| Vaccination |          1 | country   |            1 | country                | A         |         227 |     NULL |   NULL |      | BTREE      |         |
       | YES     | NULL       |
| Vaccination |          1 | dailyvacc |            1 | dailyVaccinationNumber | A         |       38359 |     NULL |   NULL | YES  | BTREE      |         |
       | YES     | NULL       |
+-------------+------------+-----------+--------------+------------------------+-----------+-------------+----------+--------+------+------------+---------+-------
-------+---------+------------+
4 rows in set (0.01 sec)
```

```
| EXPLAIN
                                                |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=37588.80 rows=15) (actual time=147.513..147.544 rows=15 loops=1)
    -> Nested loop inner join  (cost=37588.80 rows=366669) (actual time=147.512..147.542 rows=15 loops=1)
        -> Table scan on AirportData  (cost=5.25 rows=50) (actual time=0.019..0.026 rows=16 loops=1)
        -> Index lookup on temp using <auto_key0> (c=AirportData.country)  (actual time=0.001..0.001 rows=1 loops=16)
            -> Materialize  (cost=27156.68..27156.68 rows=7333) (actual time=147.506..147.509 rows=163 loops=1)
                -> Filter: ((sum(Vaccination.dailyVaccinationNumber) / CountryData.population) > 0.5)  (cost=26423.34 rows=7333) (actual time=1.948..147.085 rows=1
63 loops=1)
                    -> Group aggregate: sum(Vaccination.dailyVaccinationNumber)  (cost=26423.34 rows=7333) (actual time=1.156..146.894 rows=199 loops=1)
                        -> Nested loop inner join  (cost=25690.00 rows=7333) (actual time=0.257..133.898 rows=76149 loops=1)
                            -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.029..0.161 rows=224 loops=1)
                            -> Filter: (Vaccination.countryCode = CountryData.countryCode)  (cost=81.86 rows=33) (actual time=0.166..0.576 rows=340 loops=224)
                                -> Index lookup on Vaccination using country (country=CountryData.country)  (cost=81.86 rows=327) (actual time=0.131..0.531 rows=37
1 loops=224)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.15 sec)
```

—-----

Index on PRIMARY, population and dailyVaccinationNumber:

```
| EXPLAIN
                                                   |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=37588.80 rows=15) (actual time=179.994..180.026 rows=15 loops=1)
    -> Nested loop inner join  (cost=37588.80 rows=366669) (actual time=179.993..180.024 rows=15 loops=1)
        -> Table scan on AirportData  (cost=5.25 rows=50) (actual time=0.022..0.029 rows=16 loops=1)
        -> Index lookup on temp using <auto_key0> (c=AirportData.country)  (actual time=0.001..0.001 rows=1 loops=16)
            -> Materialize  (cost=27156.68..27156.68 rows=7333) (actual time=179.983..179.987 rows=163 loops=1)
                -> Filter: ((sum(Vaccination.dailyVaccinationNumber) / CountryData.population) > 0.5)  (cost=26423.34 rows=7333) (actual time=2.065..179.389 rows=1
63 loops=1)
                    -> Group aggregate: sum(Vaccination.dailyVaccinationNumber)  (cost=26423.34 rows=7333) (actual time=1.195..179.099 rows=199 loops=1)
                        -> Nested loop inner join  (cost=25690.00 rows=7333) (actual time=0.257..162.101 rows=76149 loops=1)
                            -> Index scan on CountryData using PRIMARY  (cost=23.15 rows=224) (actual time=0.030..0.208 rows=224 loops=1)
                            -> Filter: (Vaccination.countryCode = CountryData.countryCode)  (cost=81.86 rows=33) (actual time=0.200..0.697 rows=340 loops=224)
                                -> Index lookup on Vaccination using country (country=CountryData.country)  (cost=81.86 rows=327) (actual time=0.159..0.643 rows=37
1 loops=224)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.18 sec)
```

We tried two types of indexing, trying them separately and together. In short, neither of them helped, and one of them actually made it worse.

Our first index was on the population in the Country Data table. This affected only 224 rows so when we compared the result of this indexing to no indexing, our performance had actually decreased! Instead of taking 0.15 seconds, it took 0.16 seconds.

Our second index was on the dailyVaccinationNumber in the Vaccination table - this was the second largest table in our query, which approximately 74000 rows. We implemented this index after dropping our first index on population. There was absolutely no difference between the result of this index and no index - both of them took 0.15 seconds.

We tried both the indices together and the result actually took the longest - 0.18 seconds! This was by far the most interesting observation.

Our current theory for why none of them helped reduce the time was that our tables are too small. Indexing small tables is not necessarily optimal and that is what we saw here.

Our query groups by country which is the primary key for CountryData. This means MySQL already created the PRIMARY index for it so our original query was already very fast.

In conclusion, we will only index on the PRIMARY key, which is what we had originally, as it is either faster or the same as our other indices.