Basic information

```
ERROR 1045 (28000): Access denied for user 'root'@'34.133.137.92' (using password: YES)
soyyo011224@cloudshell:~ (bunniebunnie)$ gcloud sql connect bunniebunnie --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 28
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use tutu;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------+
| Tables_in_tutu |
+----------------+
| Article        |
| City           |
| Dailydata      |
| Vaccination    |
+----------------+
4 rows in set (0.00 sec)

mysql>
```

```
4 rows in set (0.01 sec)

mysql> select count(*) from Article;
+----------+
| count(*) |
+----------+
|     2823 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from City;
+----------+
| count(*) |
+----------+
|      531 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from Dailydata;
+----------+
| count(*) |
+----------+
|     2000 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from Vaccination;
+----------+
| count(*) |
+----------+
|     1055 |
+----------+
1 row in set (0.00 sec)

mysql>
```

1.

```
mysql> SELECT d.Country, avg(d.new_cases) as avg_num
    -> FROM Dailydata d NATURAL JOIN City c
    -> Group by d.Country
    -> limit 15;
+---------------+-----------+
| Country       | avg_num   |
+---------------+-----------+
|  Japan        |  112.8667 |
|  China        |   28.7000 |
|  Mexico       | 2890.0762 |
|  Pakistan     |  289.8537 |
|  Malaysia     |  112.2813 |
|  Spain        | 4038.9561 |
|  Morocco      |   89.0566 |
|  Afghanistan  |  197.4838 |
|  Ghana        |  111.2454 |
|  Denmark      |  476.5714 |
|  Greece       | 4863.0010 |
|  Slovakia     |   36.8750 |
|  Lebanon      |   18.1200 |
|  Albania      |  335.9235 |
|  Iraq         |  475.8889 |
+---------------+-----------+
15 rows in set (0.02 sec)
```

Before we add index:

```
mysql> EXPLAIN ANALYZE
    -> SELECT d.Country, avg(d.new_cases) AS avg_num
    -> FROM Dailydata d NATURAL JOIN City c
    -> GROUP BY d.Country
    -> LIMIT 15;
+--------------------------------------------------------------------------------------------
--------+
| EXPLAIN           |
+--------------------------------------------------------------------------------------------
--------+
| -> Limit: 15 row(s)  (actual time=17.460..17.463 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.003 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=17.459..17.462 rows=15 loops=1)
            -> Filter: (c.Country = d.Country)  (cost=311203.69 rows=310582) (actual time=2.620..8.197 rows=13625 loops=1)
                -> Inner hash join (<hash>(c.Country)=<hash>(d.Country))  (cost=311203.69 rows=310582) (actual time=2.616..4.363 rows=13625 loops=1)
                    -> Table scan on c  (cost=0.01 rows=531) (actual time=0.027..0.313 rows=531 loops=1)
                    -> Hash
                        -> Table scan on d  (cost=591.90 rows=5849) (actual time=0.035..1.682 rows=5718 loops=1)
 |
+--------------------------------------------------------------------------------------------
--------+
1 row in set (0.02 sec)
```

We add index1 on City(Country) considering it will be frequently searched against.

```
mysql> EXPLAIN ANALYZE
    -> SELECT d.Country, avg(d.new_cases) AS avg_num
    -> FROM Dailydata d NATURAL JOIN City c
    -> GROUP BY d.Country
    -> LIMIT 15;
+--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------+
| EXPLAIN

                                                                          |
+--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=29.088..29.140 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.012 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=29.087..29.138 rows=15 loops=1)
            -> Nested loop inner join  (cost=5296.90 rows=26099) (actual time=0.062..20.010 rows=13625 loops=1)
                -> Filter: (d.Country is not null)  (cost=591.90 rows=5849) (actual time=0.039..2.472 rows=5718 loops=1)
                    -> Table scan on d  (cost=591.90 rows=5849) (actual time=0.038..2.028 rows=5718 loops=1)
                -> Index lookup on c using id1 (Country=d.Country)  (cost=0.36 rows=4) (actual time=0.002..0.003 rows=2 loops=5718)
 |
+--------------------------------------------------------------------------------------------
--------------------------------------------------------------------------+
1 row in set (0.04 sec)
```

Then we delete index1 and add index2 on Dailydata(Country)

```
mysql> EXPLAIN ANALYZE
    -> SELECT d.Country, avg(d.new_cases) AS avg_num
    -> FROM Dailydata d NATURAL JOIN City c
    -> GROUP BY d.Country
    -> LIMIT 15;
+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| EXPLAIN



                                                                                         |
+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=28.558..28.562 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.004 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=28.557..28.560 rows=15 loops=1)
            -> Nested loop inner join  (cost=21915.07 rows=107097) (actual time=0.154..20.135 rows=13625 loops=1)
                -> Filter: (c.Country is not null)  (cost=54.35 rows=531) (actual time=0.047..0.296 rows=531 loops=1)
                    -> Table scan on c  (cost=54.35 rows=531) (actual time=0.046..0.244 rows=531 loops=1)
                -> Index lookup on d using id2 (Country=c.Country)  (cost=21.04 rows=202) (actual time=0.016..0.035 rows=26 loops=531)
 |
+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------+
```

Finally we make the combination of the 2 indexes.

```
+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
------+
| -> Limit: 15 row(s)  (cost=7763.86 rows=15) (actual time=4.315..28.538 rows=15 loops=1)
    -> Group aggregate: avg(d.New_cases)  (cost=7763.86 rows=25515) (actual time=4.314..28.534 rows=15 loops=1)
        -> Nested loop inner join  (cost=5212.39 rows=25515) (actual time=0.175..25.177 rows=13625 loops=1)
            -> Index range scan on d using id2, with index condition: (d.Country is not null)  (cost=565.84 rows=5718) (actual time=0.151..8.145 rows
=5718 loops=1)
            -> Index lookup on c using id (Country=d.Country)  (cost=0.36 rows=4) (actual time=0.002..0.003 rows=2 loops=5718)
 |
+--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
------+
1 row in set (0.04 sec)
```

Analysis: We found that the original one is the best. Because the index needs to be maintained as well as the original data. Indexes should not be used on columns that return a high percentage of data rows when used as a filter condition in a query's where clause. We can see that when using only a separate index, the time is optimized in the join process, reducing the number of rows to be processed. But when aggregating, the time is much slower. I guess it is because the duplicate data of the country to which the city belongs is too large and not very recognizable, so it wastes more time to load the index each time. And the join process is not optimized in the process of using two indexes. I think this shows that adding more indexes in inappropriate places will instead degrade the performance of the whole query even more.

2.

```
mysql> (SELECT COUNT(VacId) AS cntv, Company
    -> FROM Vaccination
    -> GROUP BY Company
    -> HAVING cntv < 3)
    -> UNION
    -> (SELECT COUNT(VacId) AS cntv, Company
    -> FROM Vaccination
    -> GROUP BY Company
    -> HAVING cntv > 10)
    -> LIMIT 15;
+------+-----------------------------------------------------+
| cntv | Company                                             |
+------+-----------------------------------------------------+
|    2 |  State Research Center of Virology & Biotechnology  |
|    1 |  Shenzhen GenoImmune Medical Institute              |
|    1 |  Insitute of Medical Biology                        |
|    1 |  Zydus Cadila                                       |
|    1 |  Biological E                                       |
|    1 |  Chumakov                                           |
|  146 |  AstraZeneca                                        |
|  127 |  Moderna                                            |
|  171 |  Pfizer BioNTech                                    |
|   64 |  Sinovac                                            |
|   20 |  nan                                                |
|   99 |  Gamaleya Research Institute                        |
|   99 |  Beijing Bio-Institute Biological Products (CNBG)   |
|   28 |  CanSino Biologicals                                |
|   33 |  Bharat Biotech                                     |
+------+-----------------------------------------------------+
15 rows in set (0.01 sec)
```

Before we add the index:

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.001..0.003 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.002 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=1.966..1.969 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: (cntv < 3)  (actual time=1.021..1.024 rows=6 loops=1)
                    -> Table scan on <temporary>  (actual time=0.001..0.003 rows=23 loops=1)
                        -> Aggregate using temporary table  (actual time=1.016..1.019 rows=23 loops=1)
                            -> Table scan on Vaccination  (cost=108.00 rows=1055) (actual time=0.052..0.363 rows=1055 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: (cntv > 10)  (actual time=0.922..0.926 rows=9 loops=1)
                    -> Table scan on <temporary>  (actual time=0.000..0.002 rows=20 loops=1)
                        -> Aggregate using temporary table  (actual time=0.921..0.924 rows=20 loops=1)
                            -> Table scan on Vaccination  (cost=108.00 rows=1055) (actual time=0.025..0.347 rows=1055 loops=1)
 |
1 row in set (0.01 sec)
```

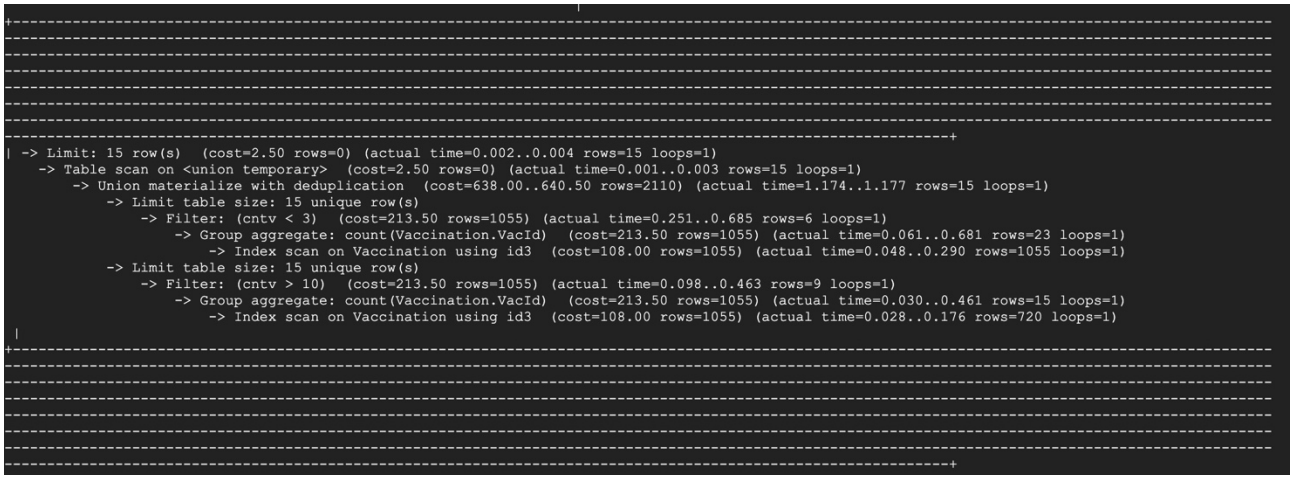
Then we add index 3 on vac(company):

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.002..0.004 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.002 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=638.00..640.50 rows=2110) (actual time=1.200..1.203 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: (cntv < 3)  (cost=213.50 rows=1055) (actual time=0.270..0.698 rows=6 loops=1)
                    -> Group aggregate: count(Vaccination.VacId)  (cost=213.50 rows=1055) (actual time=0.056..0.694 rows=23 loops=1)
                        -> Index scan on Vaccination using id3  (cost=108.00 rows=1055) (actual time=0.041..0.303 rows=1055 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: (cntv > 10)  (cost=213.50 rows=1055) (actual time=0.087..0.460 rows=9 loops=1)
                    -> Group aggregate: count(Vaccination.VacId)  (cost=213.50 rows=1055) (actual time=0.018..0.458 rows=15 loops=1)
                        -> Index scan on Vaccination using id3  (cost=108.00 rows=1055) (actual time=0.016..0.165 rows=720 loops=1)
 |
1 row in set (0.01 sec)
```

Then we delete index3 and add index4 on vacid

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.001..0.011 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.000..0.002 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=1.856..1.867 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: (cntv < 3)  (actual time=0.966..0.970 rows=6 loops=1)
                    -> Table scan on <temporary>  (actual time=0.001..0.003 rows=23 loops=1)
                        -> Aggregate using temporary table  (actual time=0.963..0.966 rows=23 loops=1)
                            -> Table scan on Vaccination  (cost=108.00 rows=1055) (actual time=0.045..0.337 rows=1055 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: (cntv > 10)  (actual time=0.868..0.872 rows=9 loops=1)
                    -> Table scan on <temporary>  (actual time=0.000..0.002 rows=20 loops=1)
                        -> Aggregate using temporary table  (actual time=0.867..0.870 rows=20 loops=1)
                            -> Table scan on Vaccination  (cost=108.00 rows=1055) (actual time=0.017..0.293 rows=1055 loops=1)
 |
1 row in set (0.01 sec)
```

Finally we add both indexes together (company and vacid):



Analysis: We found that all of them are better than the original one. Relatively speaking, the combination works the best. This might because vacid and company columns are used in the union and group by operations. But we can see that the effect of adding indexes in different locations is different, adding on company is more effective than adding on vacid, adding on company is much better than the default index, adding on id is only a little time optimization. I think it is because here is the use of count vacid instead of vacid, so the use of vacid as an index is not very direct, it is not as direct as the operation of the index by company. We can see that the use of two indexes at the same time, the effect is worse than the use of index3 alone, than the use of index4 alone is good, can also support the above law.