CS 411

Project 1-Stage 3

Qi Long, Bingjun Guo, Jiayu Zhou, Haotian Zhang

qilong2, bingjun3, jiayu9, hz75

0* Stage 2 Update

New Stage 2 pdf is already submitted on github, in doc folder, named Stage 2 - Project Conceptual Design team028-TBD-new.pdf.

0.1 ER Diagram Modified

In Section 1, Entities States' Case & Death, Covid-19 Vaccine Distribution, Counties' Case & Death are drawn as weak entities.

0.2 Relational Schema Added

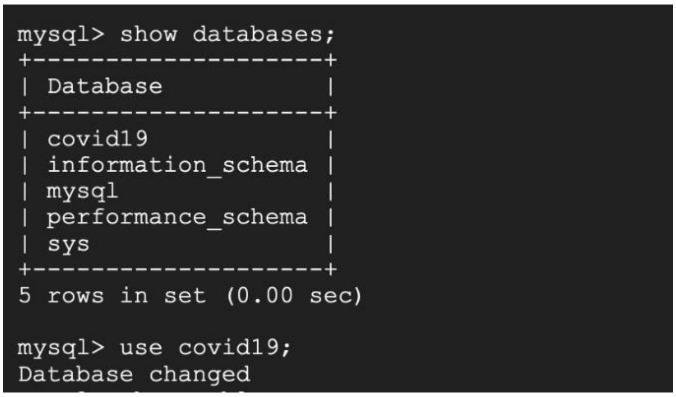
In Section 2.3, Relational Schema translated before normalization is added.

0.3 FDs Corrected

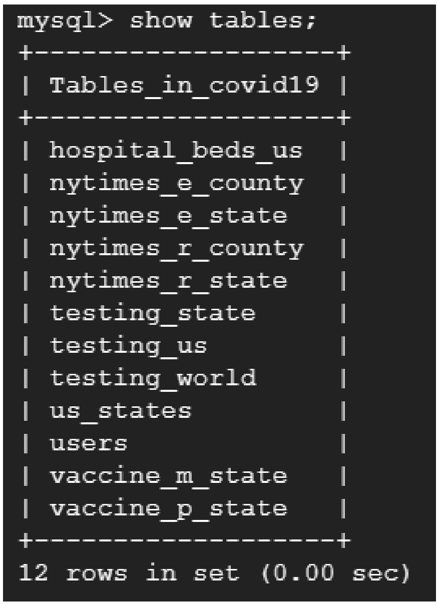
In Section 3, FDs for Entity User is modified. So are Section 3.2 Process and Section 3.3 Relational Schema.

1 Database Implementation

1.1 Implement database tables on GCP



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| covid19            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> use covid19;
Database changed
```



```
mysql> show tables;
+-------------------+
| Tables_in_covid19 |
+-------------------+
| hospital_beds_us  |
| nytimes_e_county  |
| nytimes_e_state   |
| nytimes_r_county  |
| nytimes_r_state   |
| testing_state     |
| testing_us        |
| testing_world     |
| us_states         |
| users             |
| vaccine_m_state   |
| vaccine_p_state   |
+-------------------+
12 rows in set (0.00 sec)
```

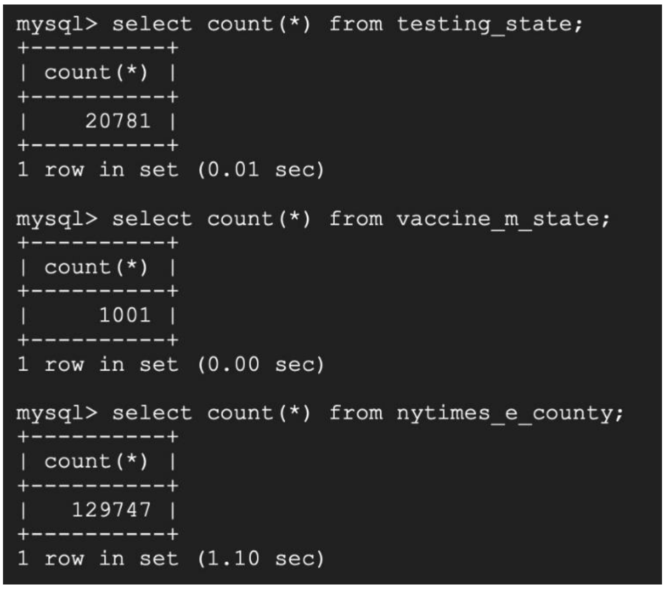
1.2 DDL commands

```
CREATE TABLE us_states (
State CHAR(40) PRIMARY KEY,
Abbreviation CHAR (2)
);

CREATE TABLE users (
ID primary key,
name CHAR(40),
status CHAR(40),
password CHAR(40)
);

CREATE TABLE vaccine_m_state (
jurisdiction CHAR(40),
week_of_allocations DATETIME,
_1st_dose_allocations INT,
_2nd_dose_allocations INT,
PRIMARY KEY (jurisdiction, week_of_allocations)
);

CREATE TABLE vaccine_p_state (
jurisdiction CHAR(40),
week_of_allocations DATETIME,
_1st_dose_allocations INT,
_2nd_dose_allocations INT,
PRIMARY KEY (jurisdiction, week_of_allocations)
```

```
);

CREATE TABLE nytimes_e_state (
date DATE,
state CHAR(40),
fips INT,
cases INT,
deaths INT,
PRIMARY KEY (date, state)
);

CREATE TABLE nytimes_e_county (
date DATE,
county CHAR(40),
state CHAR(40),
fips INT,
cases INT,
deaths INT,
PRIMARY KEY (date, county, state)
);

CREATE TABLE nytimes_r_state (
date DATE,
state CHAR(40),
fips INT,
cases INT,
deaths INT,
PRIMARY KEY (date, state)
);

CREATE TABLE nytimes_r_county (
date DATE,
county CHAR(40),
state CHAR(40) FOREIGN KEY REFERENCES us_states(State),
fips INT,
cases INT,
deaths INT,
PRIMARY KEY (date, county, state)
);

CREATE TABLE testing_world (
iso_code CHAR(20),
continent CHAR(40),
location CHAR(40),
date DATE,
total_cases INT,
new_cases INT,
new_cases_smoothed FLOAT,
total_deaths INT,
new_deaths INT,
new_deaths_smoothed FLOAT,
total_cases_per_million FLOAT,
new_cases_per_million FLOAT,
new_cases_smoothed_per_million FLOAT,
total_deaths_per_million FLOAT,
new_deaths_per_million FLOAT,
new_deaths_smoothed_per_million FLOAT,
reproduction_rate FLOAT,
icu_patients INT,
icu_patients_per_million FLOAT,
hosp_patients INT,
hosp_patients_per_million FLOAT,
weekly_icu_admissions INT,
weekly_icu_admissions_per_million FLOAT,
weekly_hosp_admissions INT,
weekly_hosp_admissions_per_million FLOAT,
new_tests INT,
total_tests INT,
total_tests_per_thousand FLOAT,
new_tests_per_thousand FLOAT,
new_tests_smoothed FLOAT,
new_tests_smoothed_per_thousand FLOAT,
positive_rate FLOAT,
tests_per_case FLOAT,
tests_units CHAR(40),
total_vaccinations INT,
people_vaccinated INT,
people_fully_vaccinated INT,
total_boosters INT,
new_vaccinations INT,
new_vaccinations_smoothed INT,
total_vaccinations_per_hundred FLOAT,
people_vaccinated_per_hundred FLOAT,
people_fully_vaccinated_per_hundred FLOAT,
total_boosters_per_hundred FLOAT,
new_vaccinations_smoothed_per_million INT,
stringency_index FLOAT,
population INT,
population_density FLOAT,
median_age FLOAT,
aged_65_older FLOAT,
aged_70_older FLOAT,
gdp_per_capita FLOAT,
extreme_poverty FLOAT,
cardiovasc_death_rate FLOAT,
diabetes_prevalence FLOAT,
female_smokers INT,
male_smokers INT,
handwashing_facilities FLOAT,
hospital_beds_per_thousand FLOAT,
life_expectancy FLOAT,
human_development_index FLOAT,
excess_mortality_cumulative_absolute FLOAT,
excess_mortality_cumulative FLOAT,
excess_mortality FLOAT,
excess_mortality_cumulative_per_million FLOAT,
PRIMARY KEY (iso_code, date, total_cases, new_cases)
);

CREATE TABLE testing_state (
date DATE,
state CHAR(2),
positive INT,
probableCases INT,
negative INT,
pending INT,
totalTestResultsSource CHAR(40),
totalTestResults INT,
hospitalizedCurrently INT,
hospitalizedCumulative INT,
inIcuCurrently INT,
inIcuCumulative INT,
onVentilatorCurrently INT,
onVentilatorCumulative INT,
recovered INT,
lastUpdateEt date,
dateModified date,
checkTimeEt date,
death INT,
hospitalized INT,
hospitalizedDisCHARged INT,
dateChecked date,
totalTestsViral INT,
positiveTestsViral INT,
negativeTestsViral INT,
positiveCasesViral INT,
deathConfirmed INT,
deathProbable INT,
totalTestEncountersViral INT,
totalTestsPeopleViral INT,
totalTestsAntibody INT,
positiveTestsAntibody INT,
negativeTestsAntibody INT,
totalTestsPeopleAntibody INT,
positiveTestsPeopleAntibody INT,
negativeTestsPeopleAntibody INT,
totalTestsPeopleAntigen INT,
positiveTestsPeopleAntigen INT,
totalTestsAntigen INT,
positiveTestsAntigen INT,
fips INT,
positiveIncrease INT,
negativeIncrease INT,
total INT,
totalTestResultsIncrease INT,
posNeg INT,
dataQualityGrade INT,
deathIncrease INT,
hospitalizedIncrease INT,
hash CHAR(40),
commercialScore INT,
negativeRegularScore INT,
negativeScore INT,
positiveScore INT,
score INT,
grade INT,
PRIMARY KEY (date, state)
);

CREATE TABLE testing_us (
date DATE PRIMARY KEY,
states INT,
positive INT,
negative INT,
pending INT,
hospitalizedCurrently INT,
hospitalizedCumulative INT,
inIcuCurrently INT,
inIcuCumulative INT,
onVentilatorCurrently INT,
onVentilatorCumulative INT,
dateChecked DATETIME,
death INT,
hospitalized INT,
totalTestResults INT,
lastModified DATETIME,
recovered INT,
total INT,
posNeg INT,
deathIncrease INT,
hospitalizedIncrease INT,
negativeIncrease INT,
positiveIncrease INT,
totalTestResultsIncrease INT,
hash CHAR(40)
);

CREATE TABLE hospital_beds_us (
Objectid INT PRIMARY KEY,
Hospital_name CHAR (100),
Hospital_type CHAR (40),
HQ_address CHAR (100),
HQ_address1 CHAR (100),
HQ_city CHAR(40),
HQ_state CHAR(2),
HQ_zip_code CHAR(5),
County_name CHAR(40),
State_name CHAR(40) FOREIGN KEY REFERENCES us_states(State),
State_fips INT,
CNTY_fips INT,
Fips INT,
Num_licensed_beds INT,
Num_staffed_beds INT,
Num_icu_beds INT,
Adult_icu_beds INT,
Pedi_icu_beds INT,
Bed_utilization FLOAT,
Avg_ventilator_usage INT,
Potential_increase_in_bed_capac INT,
Latitude FLOAT,
```

Longtitude FLOAT
);

1.3 Count Queries (3 of Tables)



```
mysql> select count(*) from testing_state;
+----------+
| count(*) |
+----------+
|    20781 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from vaccine_m_state;
+----------+
| count(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from nytimes_e_county;
+----------+
| count(*) |
+----------+
|   129747 |
+----------+
1 row in set (1.10 sec)
```

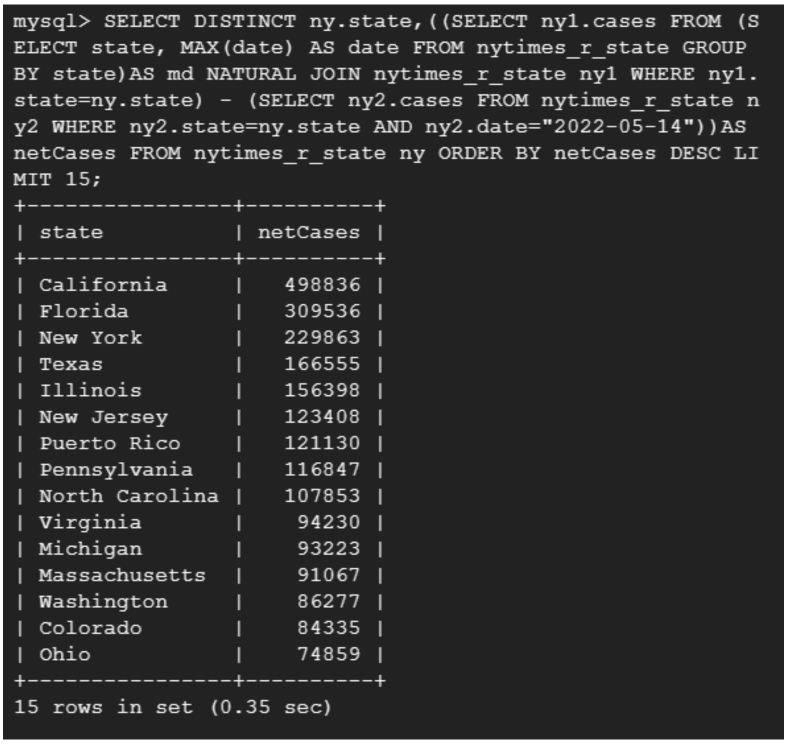
2 Advanced Queries

2.1 Advanced Query 1

1. Motivation: Provide risk of getting COVID19 within US. Try to order the states of US by the risk of getting COVID19 in that state.
2. Logic flow: calculate cases increment in the latest one month (available in table) for each state, return states and cases increment in order.
3. SQL:

SELECT DISTINCT ny.state,((SELECT ny1.cases FROM (SELECT state, MAX(date) AS date FROM nytimes_r_state GROUP BY state)AS md NATURAL JOIN nytimes_r_state ny1 WHERE ny1.state=ny.state) - (SELECT ny2.cases FROM nytimes_r_state ny2 WHERE ny2.state=ny.state AND ny2.date="2022-05-14"))AS netCases FROM nytimes_r_state ny ORDER BY netCases DESC LIMIT 15;

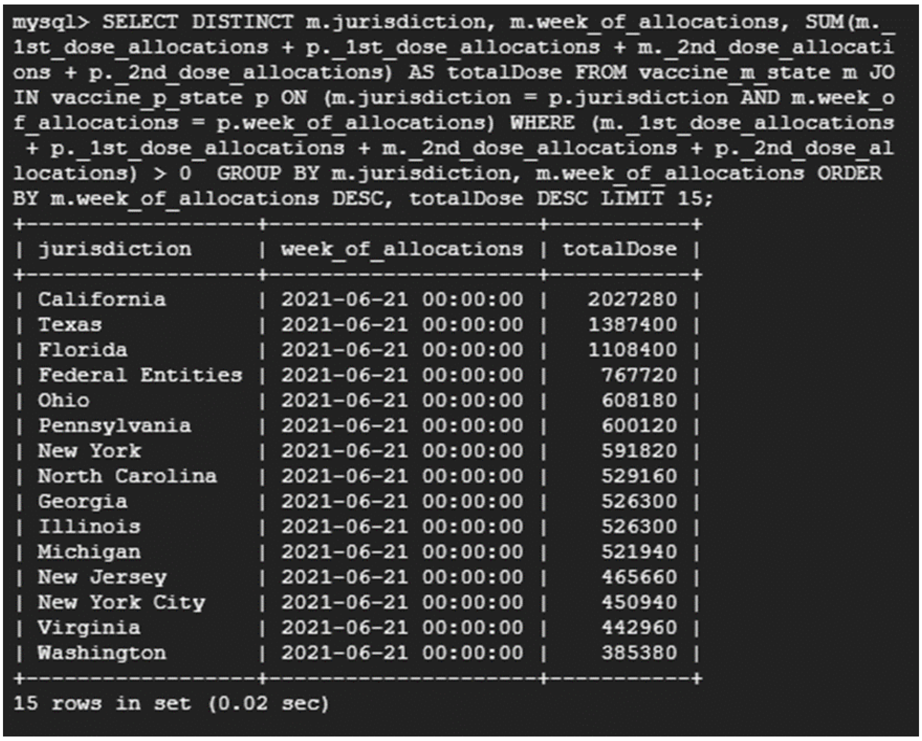
4. SQL concepts:
   1) Aggregation via GROUP BY.
   2) Subqueries.
5. Result on GCP:



```
mysql> SELECT DISTINCT ny.state,((SELECT ny1.cases FROM (S
ELECT state, MAX(date) AS date FROM nytimes_r_state GROUP
BY state)AS md NATURAL JOIN nytimes_r_state ny1 WHERE ny1.
state=ny.state) - (SELECT ny2.cases FROM nytimes_r_state n
y2 WHERE ny2.state=ny.state AND ny2.date="2022-05-14"))AS
netCases FROM nytimes_r_state ny ORDER BY netCases DESC LI
MIT 15;
+----------------+----------+
| state          | netCases |
+----------------+----------+
| California     |   498836 |
| Florida        |   309536 |
| New York       |   229863 |
| Texas          |   166555 |
| Illinois       |   156398 |
| New Jersey     |   123408 |
| Puerto Rico    |   121130 |
| Pennsylvania   |   116847 |
| North Carolina |   107853 |
| Virginia       |    94230 |
| Michigan       |    93223 |
| Massachusetts  |    91067 |
| Washington     |    86277 |
| Colorado       |    84335 |
| Ohio           |    74859 |
+----------------+----------+
15 rows in set (0.35 sec)
```

2.2 Advanced Query 2

1. Motivation: Provide total doses of vaccine allocated to each state in US. Order by number of doses.
2. Logic Flow: Calculate sum of 1st and 2nd doses for each state for each brand of vaccine, return states and number of total doses in order.
3. SQL:
   SELECT DISTINCT m.jurisdiction, m.week_of_allocations, SUM(m._1st_dose_allocations + p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocations) AS totalDose FROM vaccine_m_state m JOIN vaccine_p_state p ON (m.jurisdiction = p.jurisdiction AND m.week_of_allocations = p.week_of_allocations) WHERE (m._1st_dose_allocations + p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocations) > 0 GROUP BY m.jurisdiction, m.week_of_allocations ORDER BY m.week_of_allocations DESC, totalDose DESC LIMIT 15;
4. SQL concepts:
   1) Join of multiple relations.
   2) Aggregation via GROUP BY.
5. Result on GCP:

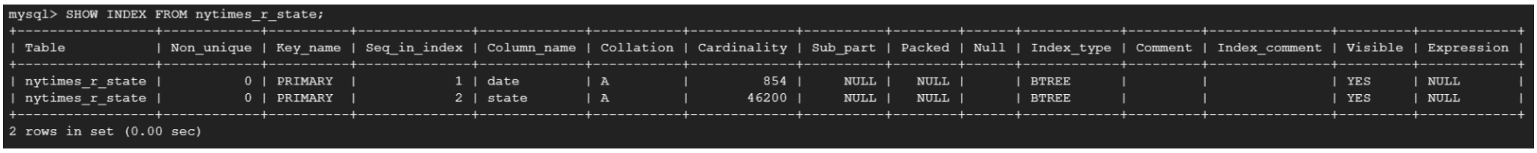


```
mysql> SELECT DISTINCT m.jurisdiction, m.week_of_allocations, SUM(m._
1st_dose_allocations + p._1st_dose_allocations + m._2nd_dose_allocati
ons + p._2nd_dose_allocations) AS totalDose FROM vaccine_m_state m JO
IN vaccine_p_state p ON (m.jurisdiction = p.jurisdiction AND m.week_o
f_allocations = p.week_of_allocations) WHERE (m._1st_dose_allocations
 + p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_al
locations) > 0  GROUP BY m.jurisdiction, m.week_of_allocations ORDER
BY m.week_of_allocations DESC, totalDose DESC LIMIT 15;
+------------------+---------------------+-----------+
| jurisdiction     | week_of_allocations | totalDose |
+------------------+---------------------+-----------+
| California       | 2021-06-21 00:00:00 |   2027280 |
| Texas            | 2021-06-21 00:00:00 |   1387400 |
| Florida          | 2021-06-21 00:00:00 |   1108400 |
| Federal Entities | 2021-06-21 00:00:00 |    767720 |
| Ohio             | 2021-06-21 00:00:00 |    608180 |
| Pennsylvania     | 2021-06-21 00:00:00 |    600120 |
| New York         | 2021-06-21 00:00:00 |    591820 |
| North Carolina   | 2021-06-21 00:00:00 |    529160 |
| Georgia          | 2021-06-21 00:00:00 |    526300 |
| Illinois         | 2021-06-21 00:00:00 |    526300 |
| Michigan         | 2021-06-21 00:00:00 |    521940 |
| New Jersey       | 2021-06-21 00:00:00 |    465660 |
| New York City    | 2021-06-21 00:00:00 |    450940 |
| Virginia         | 2021-06-21 00:00:00 |    442960 |
| Washington       | 2021-06-21 00:00:00 |    385380 |
+------------------+---------------------+-----------+
15 rows in set (0.02 sec)
```

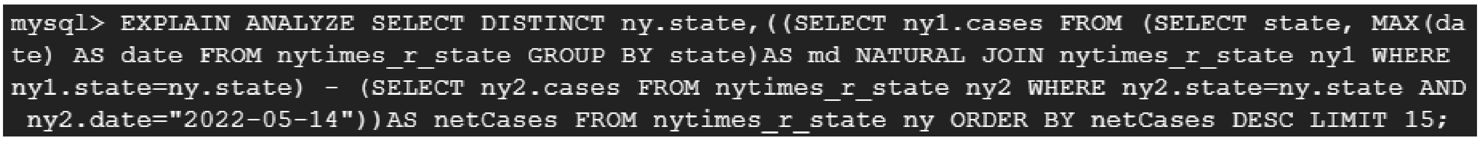
3 Indexing Analysis

3.1 Query 1 (2.1)

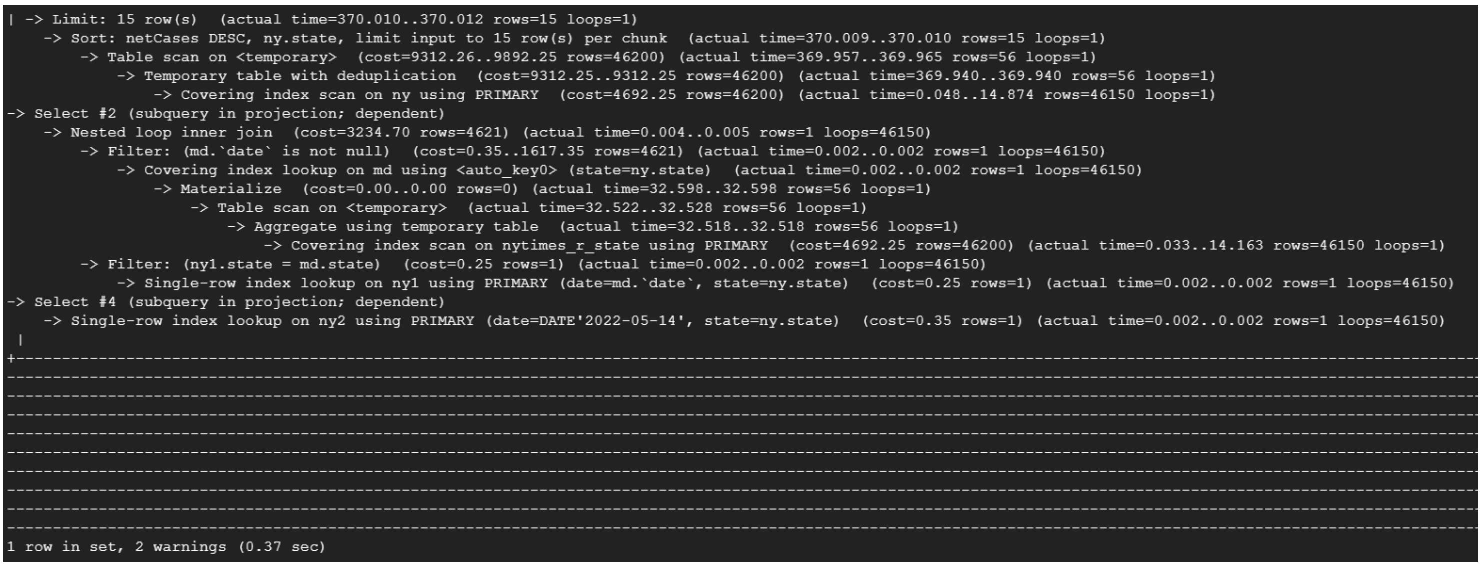
Table nytimes_r_state is used. Without adding new index, analyze:



```
mysql> SHOW INDEX FROM nytimes_r_state;
+-----------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table           | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| nytimes_r_state |          0 | PRIMARY  |            1 | date        | A         |         854 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| nytimes_r_state |          0 | PRIMARY  |            2 | state       | A         |       46200 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-----------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
2 rows in set (0.00 sec)
```



```
mysql> EXPLAIN ANALYZE SELECT DISTINCT ny.state,((SELECT ny1.cases FROM (SELECT state, MAX(da
te) AS date FROM nytimes_r_state GROUP BY state)AS md NATURAL JOIN nytimes_r_state ny1 WHERE
ny1.state=ny.state) - (SELECT ny2.cases FROM nytimes_r_state ny2 WHERE ny2.state=ny.state AND
 ny2.date="2022-05-14"))AS netCases FROM nytimes_r_state ny ORDER BY netCases DESC LIMIT 15;
```



```
| -> Limit: 15 row(s)  (actual time=370.010..370.012 rows=15 loops=1)
    -> Sort: netCases DESC, ny.state, limit input to 15 row(s) per chunk  (actual time=370.009..370.010 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=9312.26..9892.25 rows=46200) (actual time=369.957..369.965 rows=56 loops=1)
            -> Temporary table with deduplication  (cost=9312.25..9312.25 rows=46200) (actual time=369.940..369.940 rows=56 loops=1)
                -> Covering index scan on ny using PRIMARY  (cost=4692.25 rows=46200) (actual time=0.048..14.874 rows=46150 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Nested loop inner join  (cost=3234.70 rows=4621) (actual time=0.004..0.005 rows=1 loops=46150)
        -> Filter: (md.`date` is not null)  (cost=0.35..1617.35 rows=4621) (actual time=0.002..0.002 rows=1 loops=46150)
            -> Covering index lookup on md using <auto_key0> (state=ny.state)  (actual time=0.002..0.002 rows=1 loops=46150)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=32.598..32.598 rows=56 loops=1)
                    -> Table scan on <temporary>  (actual time=32.522..32.528 rows=56 loops=1)
                        -> Aggregate using temporary table  (actual time=32.518..32.518 rows=56 loops=1)
                            -> Covering index scan on nytimes_r_state using PRIMARY  (cost=4692.25 rows=46200) (actual time=0.033..14.163 rows=46150 loops=1)
        -> Filter: (ny1.state = md.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
            -> Single-row index lookup on ny1 using PRIMARY (date=md.`date`, state=ny.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
-> Select #4 (subquery in projection; dependent)
    -> Single-row index lookup on ny2 using PRIMARY (date=DATE'2022-05-14', state=ny.state)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set, 2 warnings (0.37 sec)
```

3.1.1 Indexing Design 1

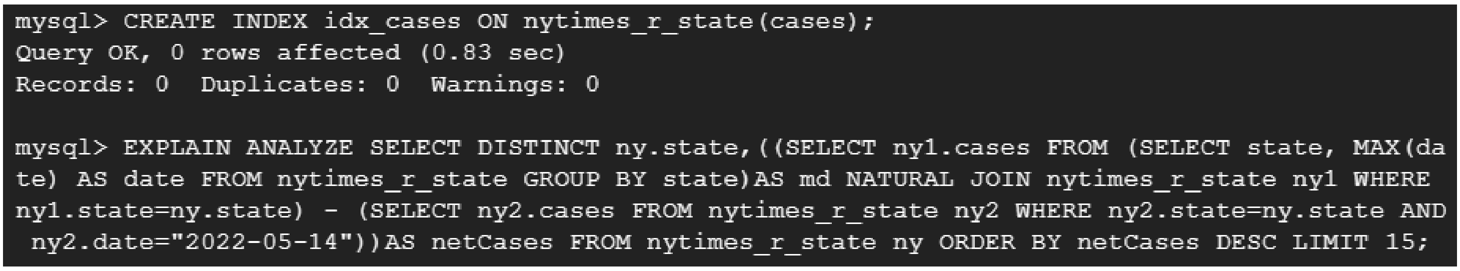
1. Motivation:

   The subqueries SELECT by state and date, which is primary key having BTREE INDEX already. Observing that we have order by on cases attributes, we try inserting index on cases.

2. Design:

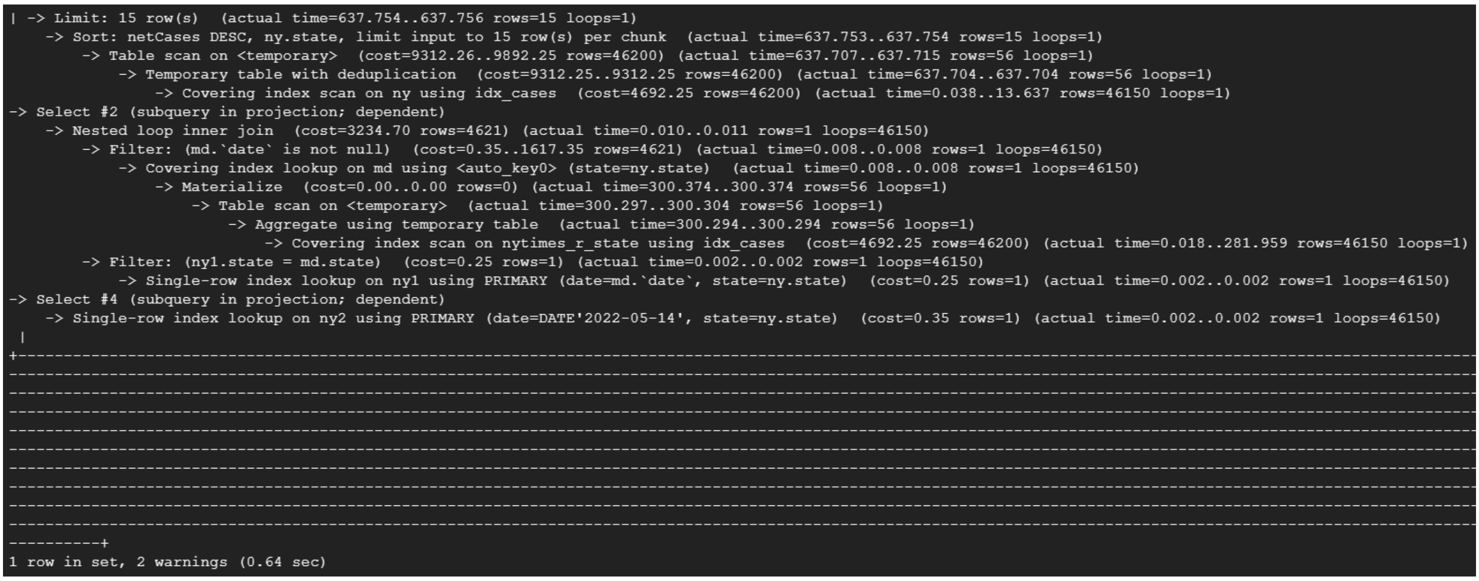
   CREATE INDEX idx_cases ON nytimes_r_state(cases);

3. Implementation:



```
mysql> CREATE INDEX idx_cases ON nytimes_r_state(cases);
Query OK, 0 rows affected (0.83 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT DISTINCT ny.state,((SELECT ny1.cases FROM (SELECT state, MAX(da
te) AS date FROM nytimes_r_state GROUP BY state)AS md NATURAL JOIN nytimes_r_state ny1 WHERE
ny1.state=ny.state) - (SELECT ny2.cases FROM nytimes_r_state ny2 WHERE ny2.state=ny.state AND
 ny2.date="2022-05-14"))AS netCases FROM nytimes_r_state ny ORDER BY netCases DESC LIMIT 15;
```

4. Result:



```
| -> Limit: 15 row(s)  (actual time=637.754..637.756 rows=15 loops=1)
    -> Sort: netCases DESC, ny.state, limit input to 15 row(s) per chunk  (actual time=637.753..637.754 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=9312.26..9892.25 rows=46200) (actual time=637.707..637.715 rows=56 loops=1)
            -> Temporary table with deduplication  (cost=9312.25..9312.25 rows=46200) (actual time=637.704..637.704 rows=56 loops=1)
                -> Covering index scan on ny using idx_cases  (cost=4692.25 rows=46200) (actual time=0.038..13.637 rows=46150 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Nested loop inner join  (cost=3234.70 rows=4621) (actual time=0.010..0.011 rows=1 loops=46150)
        -> Filter: (md.`date` is not null)  (cost=0.35..1617.35 rows=4621) (actual time=0.008..0.008 rows=1 loops=46150)
            -> Covering index lookup on md using <auto_key0> (state=ny.state)  (actual time=0.008..0.008 rows=1 loops=46150)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=300.374..300.374 rows=56 loops=1)
                    -> Table scan on <temporary>  (actual time=300.297..300.304 rows=56 loops=1)
                        -> Aggregate using temporary table  (actual time=300.294..300.294 rows=56 loops=1)
                            -> Covering index scan on nytimes_r_state using idx_cases  (cost=4692.25 rows=46200) (actual time=0.018..281.959 rows=46150 loops=1)
        -> Filter: (ny1.state = md.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
            -> Single-row index lookup on ny1 using PRIMARY (date=md.`date`, state=ny.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
-> Select #4 (subquery in projection; dependent)
    -> Single-row index lookup on ny2 using PRIMARY (date=DATE'2022-05-14', state=ny.state)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------
----------+
1 row in set, 2 warnings (0.64 sec)
```

5. Analysis:

   Inserting index on cases has an adverse effect on the overall cost of the query. This is because we always take states and netCases into consideration and running this query needs additional time to scanning the index keys, which delay the overall query speed.

3.1.2 Indexing Design 2

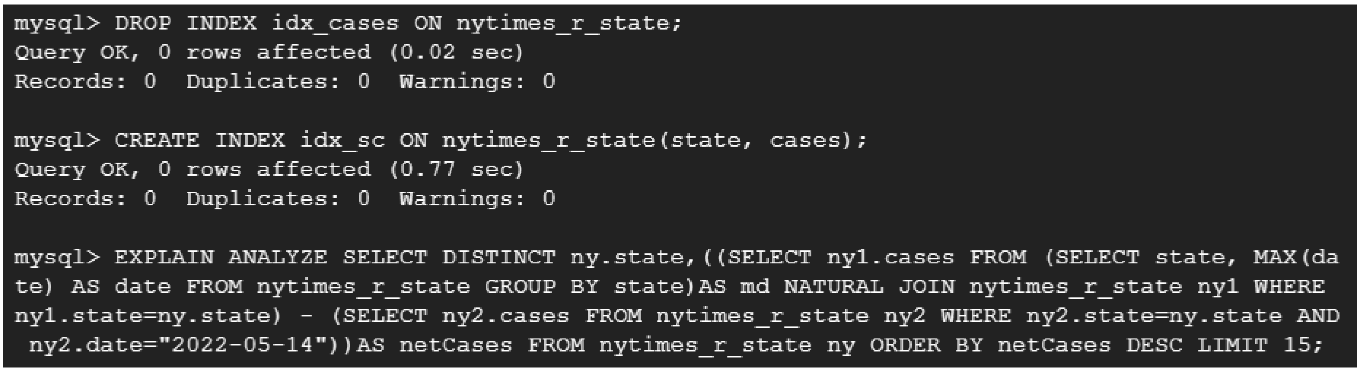
1. Motivation:

   The order by clause is on state and netCases. Therefore, we try adding index on (state, cases) so that the sort can be based on a BTREE data structure.

2. Design:

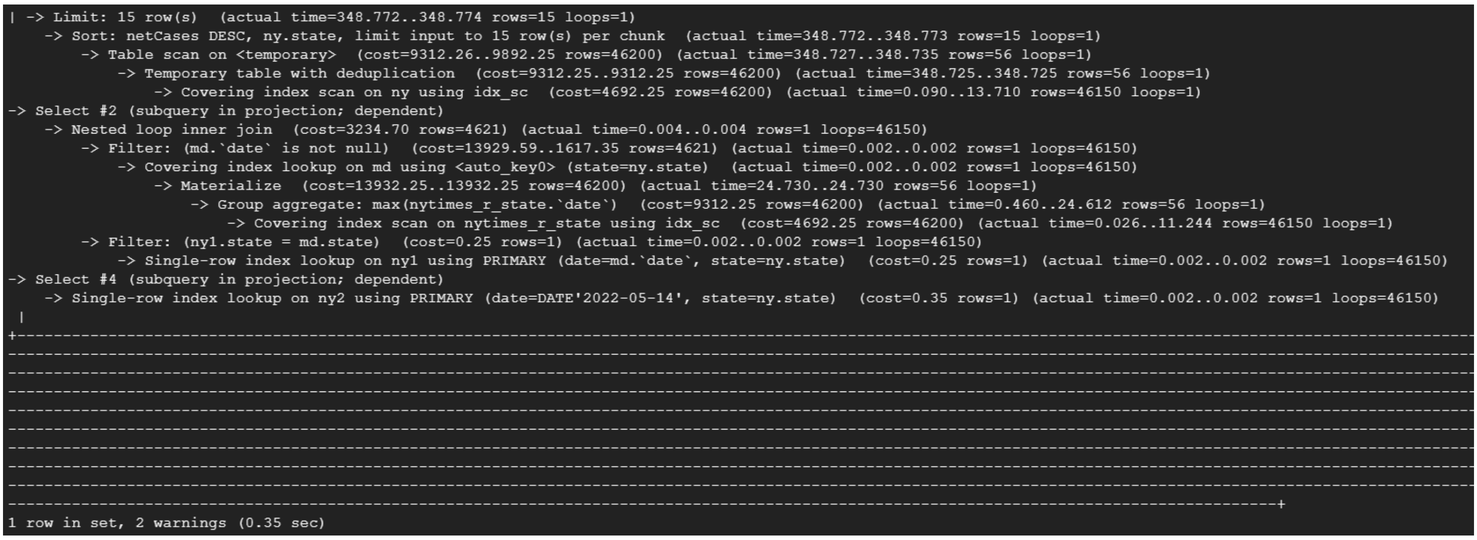
   CREATE INDEX idx_sc ON nytimes_r_state(state, cases);

3. Implementation:



```
mysql> DROP INDEX idx_cases ON nytimes_r_state;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_sc ON nytimes_r_state(state, cases);
Query OK, 0 rows affected (0.77 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT DISTINCT ny.state,((SELECT ny1.cases FROM (SELECT state, MAX(da
te) AS date FROM nytimes_r_state GROUP BY state)AS md NATURAL JOIN nytimes_r_state ny1 WHERE
ny1.state=ny.state) - (SELECT ny2.cases FROM nytimes_r_state ny2 WHERE ny2.state=ny.state AND
 ny2.date="2022-05-14"))AS netCases FROM nytimes_r_state ny ORDER BY netCases DESC LIMIT 15;
```

4. Result:



```
| -> Limit: 15 row(s)  (actual time=348.772..348.774 rows=15 loops=1)
    -> Sort: netCases DESC, ny.state, limit input to 15 row(s) per chunk  (actual time=348.772..348.773 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=9312.26..9892.25 rows=46200) (actual time=348.727..348.735 rows=56 loops=1)
            -> Temporary table with deduplication  (cost=9312.25..9312.25 rows=46200) (actual time=348.725..348.725 rows=56 loops=1)
                -> Covering index scan on ny using idx_sc  (cost=4692.25 rows=46200) (actual time=0.090..13.710 rows=46150 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Nested loop inner join  (cost=3234.70 rows=4621) (actual time=0.004..0.004 rows=1 loops=46150)
        -> Filter: (md.`date` is not null)  (cost=13929.59..1617.35 rows=4621) (actual time=0.002..0.002 rows=1 loops=46150)
            -> Covering index lookup on md using <auto_key0> (state=ny.state)  (actual time=0.002..0.002 rows=1 loops=46150)
                -> Materialize  (cost=13932.25..13932.25 rows=46200) (actual time=24.730..24.730 rows=56 loops=1)
                    -> Group aggregate: max(nytimes_r_state.`date`)  (cost=9312.25 rows=46200) (actual time=0.460..24.612 rows=56 loops=1)
                        -> Covering index scan on nytimes_r_state using idx_sc  (cost=4692.25 rows=46200) (actual time=0.026..11.244 rows=46150 loops=1)
        -> Filter: (ny1.state = md.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
            -> Single-row index lookup on ny1 using PRIMARY (date=md.`date`, state=ny.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
-> Select #4 (subquery in projection; dependent)
    -> Single-row index lookup on ny2 using PRIMARY (date=DATE'2022-05-14', state=ny.state)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
1 row in set, 2 warnings (0.35 sec)
```

5. Analysis:

This design reduces running time. Since we need to fetch the cases by checking corresponding state and date, during order by, if we insert (state, cases) index, it can sort based on BTREE index, which is faster than index on state only.

3.1.3 Indexing Design 3

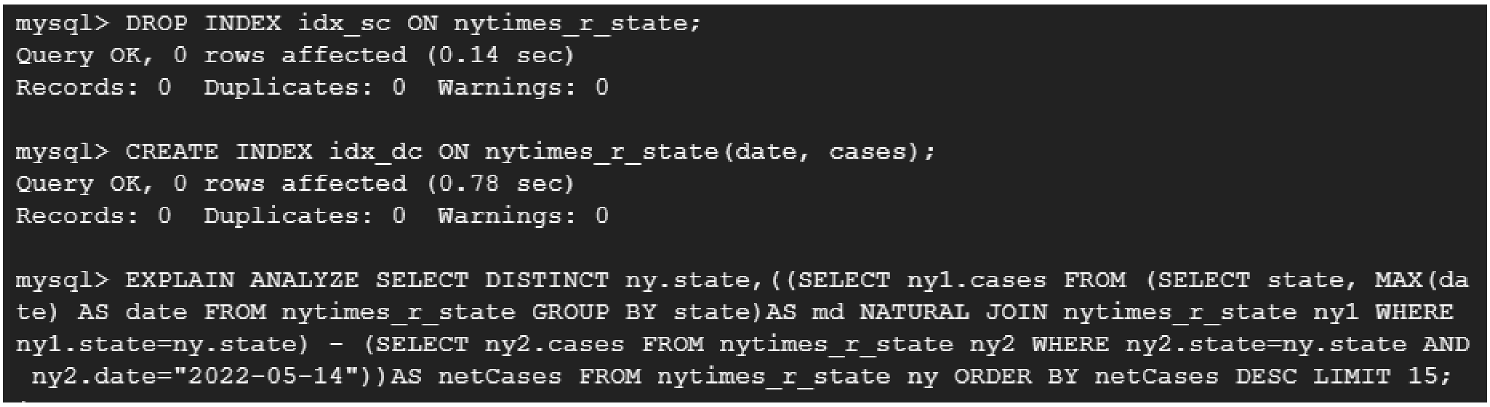
1. Motivation:

For two subqueries, since SELECT is based on state, date, and cases, we try adding index on (date, cases) so that the index scan can be based on (date, cases) together instead of date and cases separately.

2. Design:

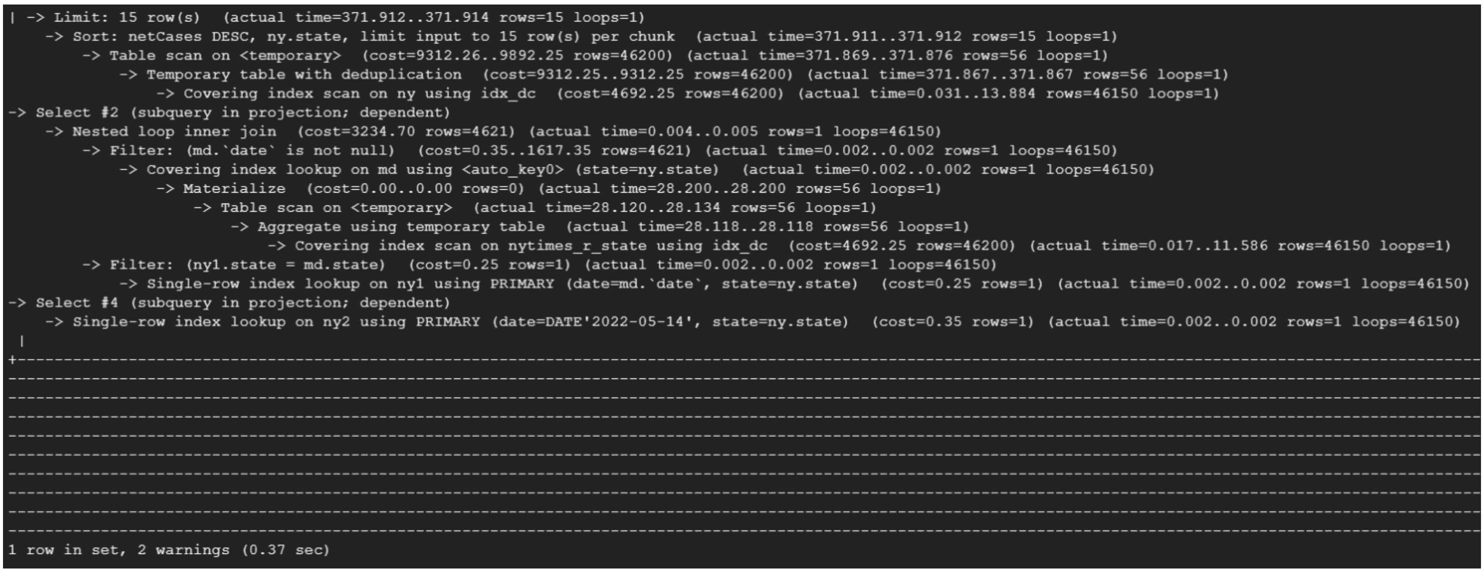
CREATE INDEX idx_dc ON nytimes_r_state(date, cases);

3. Implementation:



```
mysql> DROP INDEX idx_sc ON nytimes_r_state;
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_dc ON nytimes_r_state(date, cases);
Query OK, 0 rows affected (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT DISTINCT ny.state,((SELECT ny1.cases FROM (SELECT state, MAX(da
te) AS date FROM nytimes_r_state GROUP BY state)AS md NATURAL JOIN nytimes_r_state ny1 WHERE
ny1.state=ny.state) - (SELECT ny2.cases FROM nytimes_r_state ny2 WHERE ny2.state=ny.state AND
 ny2.date="2022-05-14"))AS netCases FROM nytimes_r_state ny ORDER BY netCases DESC LIMIT 15;
```

4. Result:

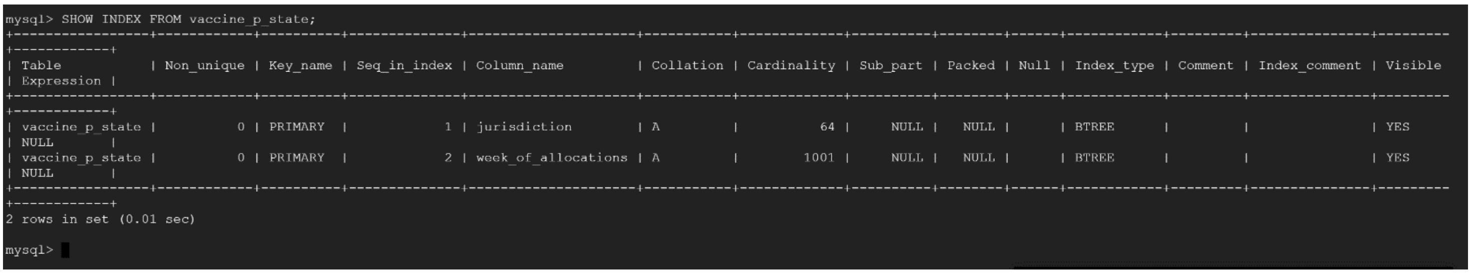


```
| -> Limit: 15 row(s)  (actual time=371.912..371.914 rows=15 loops=1)
    -> Sort: netCases DESC, ny.state, limit input to 15 row(s) per chunk  (actual time=371.911..371.912 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=9312.26..9892.25 rows=46200) (actual time=371.869..371.876 rows=56 loops=1)
            -> Temporary table with deduplication  (cost=9312.25..9312.25 rows=46200) (actual time=371.867..371.867 rows=56 loops=1)
                -> Covering index scan on ny using idx_dc  (cost=4692.25 rows=46200) (actual time=0.031..13.884 rows=46150 loops=1)
-> Select #2 (subquery in projection; dependent)
    -> Nested loop inner join  (cost=3234.70 rows=4621) (actual time=0.004..0.005 rows=1 loops=46150)
        -> Filter: (md.`date` is not null)  (cost=0.35..1617.35 rows=4621) (actual time=0.002..0.002 rows=1 loops=46150)
            -> Covering index lookup on md using <auto_key0> (state=ny.state)  (actual time=0.002..0.002 rows=1 loops=46150)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=28.200..28.200 rows=56 loops=1)
                    -> Table scan on <temporary>  (actual time=28.120..28.134 rows=56 loops=1)
                        -> Aggregate using temporary table  (actual time=28.118..28.118 rows=56 loops=1)
                            -> Covering index scan on nytimes_r_state using idx_dc  (cost=4692.25 rows=46200) (actual time=0.017..11.586 rows=46150 loops=1)
        -> Filter: (ny1.state = md.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
            -> Single-row index lookup on ny1 using PRIMARY (date=md.`date`, state=ny.state)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
-> Select #4 (subquery in projection; dependent)
    -> Single-row index lookup on ny2 using PRIMARY (date=DATE'2022-05-14', state=ny.state)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=46150)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set, 2 warnings (0.37 sec)
```

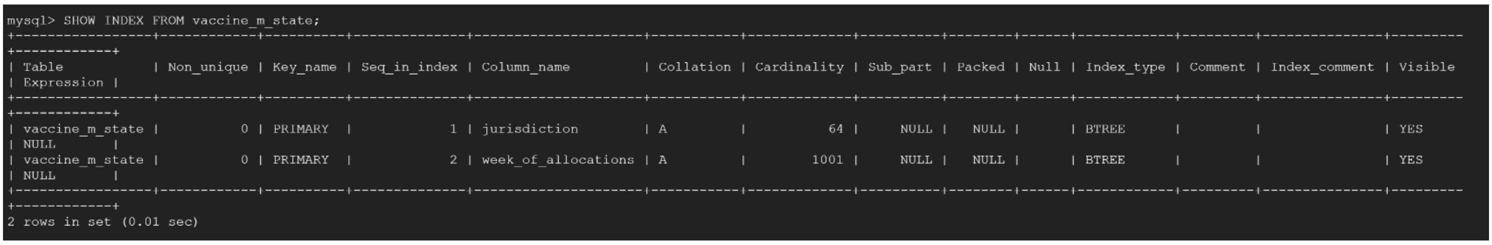
5. Analysis:
   It shows that inserting index (date, cases) do not have the improvement on running speed as we expected, which actually make sense. The subquery only select based on state and date, which already have BTREE index. Besides, the ORDER BY and GROUP BY within and out of subqueries only deal with state and netCases, which also lead to no improvement by index (date, cases).

3.2 Query 2

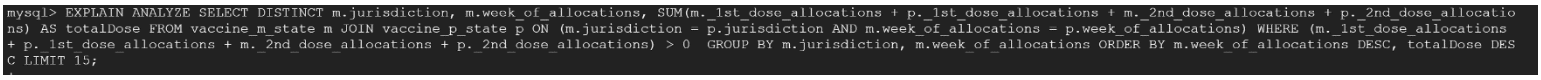
Table vaccine_p_state and vaccine_m_state are used. Without adding new index, analyze:



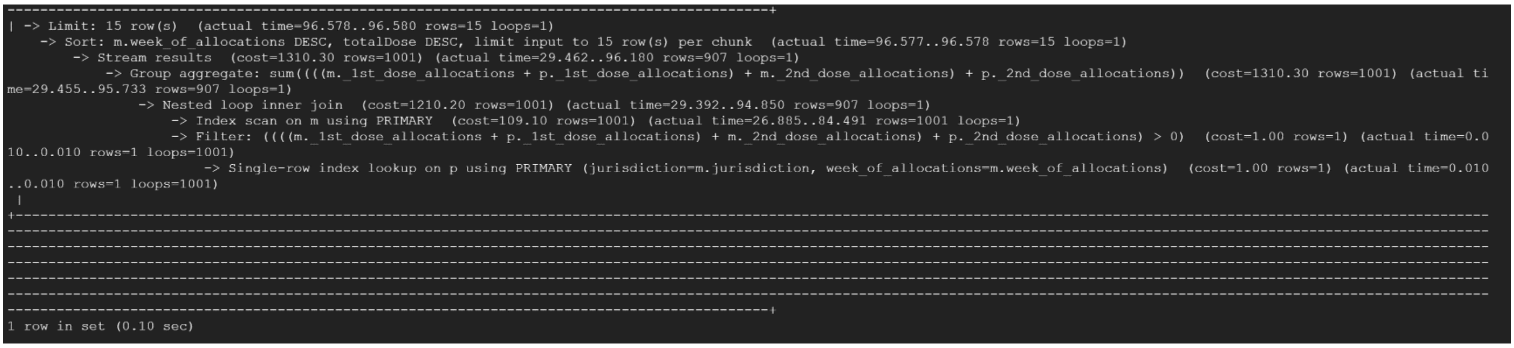
```
mysql> SHOW INDEX FROM vaccine_p_state;
+-----------------+------------+----------+--------------+---------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------
+------------+
| Table           | Non_unique | Key_name | Seq_in_index | Column_name         | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible
| Expression |
+-----------------+------------+----------+--------------+---------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------
+------------+
| vaccine_p_state |          0 | PRIMARY  |            1 | jurisdiction        | A         |          64 |     NULL |   NULL |      | BTREE      |         |               | YES
| NULL       |
| vaccine_p_state |          0 | PRIMARY  |            2 | week_of_allocations | A         |        1001 |     NULL |   NULL |      | BTREE      |         |               | YES
| NULL       |
+-----------------+------------+----------+--------------+---------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------
+------------+
2 rows in set (0.01 sec)

mysql> 
```



```
mysql> SHOW INDEX FROM vaccine_m_state;
+-----------------+------------+----------+--------------+---------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------
+------------+
| Table           | Non_unique | Key_name | Seq_in_index | Column_name         | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible
| Expression |
+-----------------+------------+----------+--------------+---------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------
+------------+
| vaccine_m_state |          0 | PRIMARY  |            1 | jurisdiction        | A         |          64 |     NULL |   NULL |      | BTREE      |         |               | YES
| NULL       |
| vaccine_m_state |          0 | PRIMARY  |            2 | week_of_allocations | A         |        1001 |     NULL |   NULL |      | BTREE      |         |               | YES
| NULL       |
+-----------------+------------+----------+--------------+---------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------
+------------+
2 rows in set (0.01 sec)
```



```
mysql> EXPLAIN ANALYZE SELECT DISTINCT m.jurisdiction, m.week_of_allocations, SUM(m._1st_dose_allocations + p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocatio
ns) AS totalDose FROM vaccine_m_state m JOIN vaccine_p_state p ON (m.jurisdiction = p.jurisdiction AND m.week_of_allocations = p.week_of_allocations) WHERE (m._1st_dose_allocations
+ p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocations) > 0  GROUP BY m.jurisdiction, m.week_of_allocations ORDER BY m.week_of_allocations DESC, totalDose DES
C LIMIT 15;
```



```
------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=96.578..96.580 rows=15 loops=1)
    -> Sort: m.week_of_allocations DESC, totalDose DESC, limit input to 15 row(s) per chunk  (actual time=96.577..96.578 rows=15 loops=1)
        -> Stream results  (cost=1310.30 rows=1001) (actual time=29.462..96.180 rows=907 loops=1)
            -> Group aggregate: sum((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_allocations))  (cost=1310.30 rows=1001) (actual ti
me=29.455..95.733 rows=907 loops=1)
                -> Nested loop inner join  (cost=1210.20 rows=1001) (actual time=29.392..94.850 rows=907 loops=1)
                    -> Index scan on m using PRIMARY  (cost=109.10 rows=1001) (actual time=26.885..84.491 rows=1001 loops=1)
                    -> Filter: ((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_allocations) > 0)  (cost=1.00 rows=1) (actual time=0.0
10..0.010 rows=1 loops=1001)
                        -> Single-row index lookup on p using PRIMARY (jurisdiction=m.jurisdiction, week_of_allocations=m.week_of_allocations)  (cost=1.00 rows=1) (actual time=0.010
..0.010 rows=1 loops=1001)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
1 row in set (0.10 sec)
```

3.2.1 Indexing Design 1

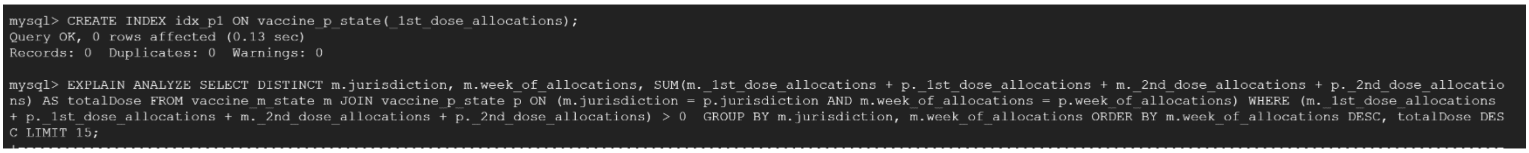
1. Motivation:

   To speed up scanning through first dose for each state and date, we try inserting index on vaccine_p_state(_1st_dose_allocations).

2. Design:

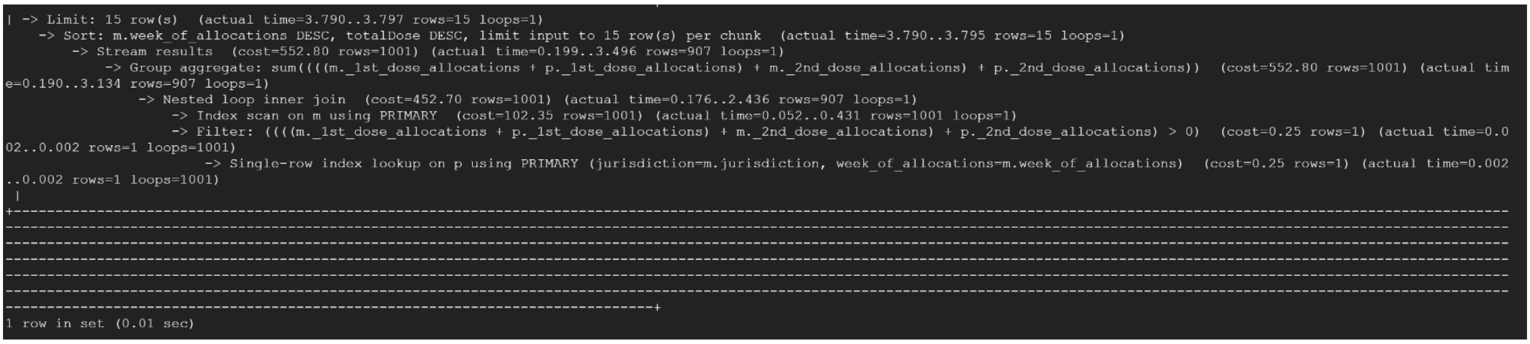
   CREATE INDEX idx_p1 ON vaccine_p_state(_1st_dose_allocations);

3. Implementation:



```
mysql> CREATE INDEX idx_p1 ON vaccine_p_state(_1st_dose_allocations);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT DISTINCT m.jurisdiction, m.week_of_allocations, SUM(m._1st_dose_allocations + p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocatio
ns) AS totalDose FROM vaccine_m_state m JOIN vaccine_p_state p ON (m.jurisdiction = p.jurisdiction AND m.week_of_allocations = p.week_of_allocations) WHERE (m._1st_dose_allocations
+ p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocations) > 0  GROUP BY m.jurisdiction, m.week_of_allocations ORDER BY m.week_of_allocations DESC, totalDose DES
C LIMIT 15;
```

4. Result:



```
| -> Limit: 15 row(s)  (actual time=3.790..3.797 rows=15 loops=1)
    -> Sort: m.week_of_allocations DESC, totalDose DESC, limit input to 15 row(s) per chunk  (actual time=3.790..3.795 rows=15 loops=1)
        -> Stream results  (cost=552.80 rows=1001) (actual time=0.199..3.496 rows=907 loops=1)
            -> Group aggregate: sum((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_allocations))  (cost=552.80 rows=1001) (actual tim
e=0.190..3.134 rows=907 loops=1)
                -> Nested loop inner join  (cost=452.70 rows=1001) (actual time=0.176..2.436 rows=907 loops=1)
                    -> Index scan on m using PRIMARY  (cost=102.35 rows=1001) (actual time=0.052..0.431 rows=1001 loops=1)
                    -> Filter: ((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_allocations) > 0)  (cost=0.25 rows=1) (actual time=0.0
02..0.002 rows=1 loops=1001)
                        -> Single-row index lookup on p using PRIMARY (jurisdiction=m.jurisdiction, week_of_allocations=m.week_of_allocations)  (cost=0.25 rows=1) (actual time=0.002
..0.002 rows=1 loops=1001)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

5. Analysis:

   This index makes a great contribution to speed up searching because the number of first dose taking by company p is part of the group-by clause, which needs to be scan through when the query is executed.

3.2.2 Indexing Design 2

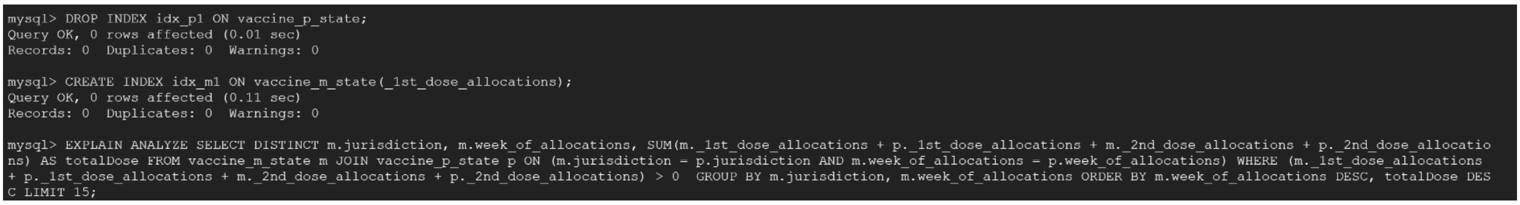
1. Motivation:

   Since a Theta-JOIN is implemented on tables vaccine_p_state and vaccine_m_state, we have the same idea from Index Design 1 that try index on vaccine_m_state(_1st_dose_allocations) can speed up the query.

2. Design:

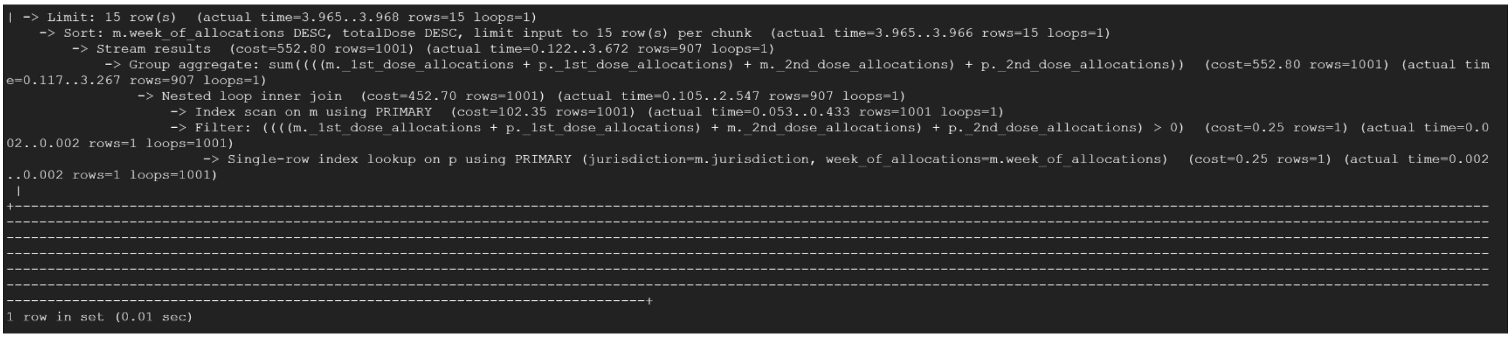
   CREATE INDEX idx_m1 ON vaccine_m_state(_1st_dose_allocations);

3. Implementation:



```
mysql> DROP INDEX idx_p1 ON vaccine_p_state;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_m1 ON vaccine_m_state(_1st_dose_allocations);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT DISTINCT m.jurisdiction, m.week_of_allocations, SUM(m._1st_dose_allocations + p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocatio
ns) AS totalDose FROM vaccine_m_state m JOIN vaccine_p_state p ON (m.jurisdiction = p.jurisdiction AND m.week_of_allocations = p.week_of_allocations) WHERE (m._1st_dose_allocations
+ p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocations) > 0  GROUP BY m.jurisdiction, m.week_of_allocations ORDER BY m.week_of_allocations DESC, totalDose DES
C LIMIT 15;
```

4. Result:

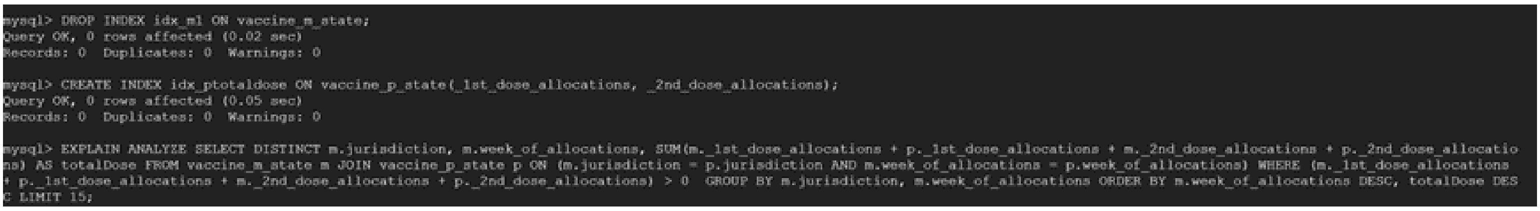


```
| -> Limit: 15 row(s)  (actual time=3.965..3.968 rows=15 loops=1)
    -> Sort: m.week_of_allocations DESC, totalDose DESC, limit input to 15 row(s) per chunk  (actual time=3.965..3.966 rows=15 loops=1)
        -> Stream results  (cost=552.80 rows=1001) (actual time=0.122..3.672 rows=907 loops=1)
            -> Group aggregate: sum((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_allocations))  (cost=552.80 rows=1001) (actual tim
e=0.117..3.267 rows=907 loops=1)
                -> Nested loop inner join  (cost=452.70 rows=1001) (actual time=0.105..2.547 rows=907 loops=1)
                    -> Index scan on m using PRIMARY  (cost=102.35 rows=1001) (actual time=0.053..0.433 rows=1001 loops=1)
                    -> Filter: ((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_allocations) > 0)  (cost=0.25 rows=1) (actual time=0.0
02..0.002 rows=1 loops=1001)
                        -> Single-row index lookup on p using PRIMARY (jurisdiction=m.jurisdiction, week_of_allocations=m.week_of_allocations)  (cost=0.25 rows=1) (actual time=0.002
..0.002 rows=1 loops=1001)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

5. Analysis:
   This index makes a great contribution to speed up searching because the number of first dose taking by company m is also part of the group-by clause, which needs to be scan through when the query is executed.

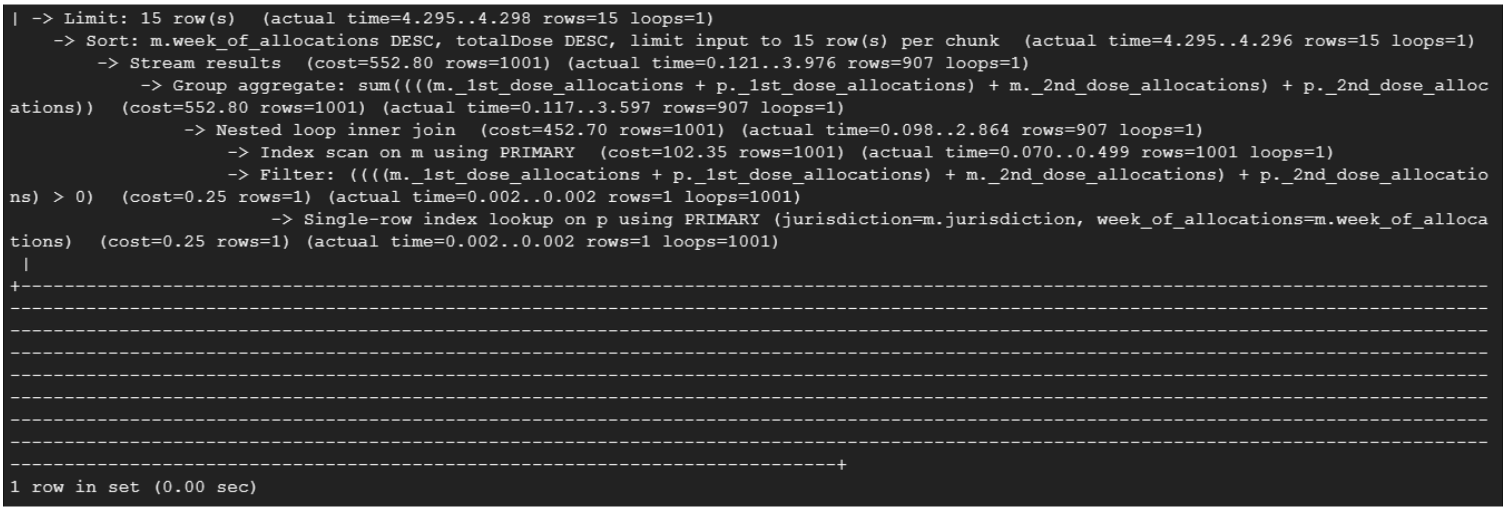
3.2.3 Indexing Design 3

1. Motivation:
   To speed up summation clause, since 1st dose and 2nd dose are always added together according to state and date, we try adding index on vaccine_p_state(_1st_dose_allocations, _2nd__dose_allocation) so that the scanning through SELECT (_1st_dose_allocations, _2nd__dose_allocation) can be faster scanned.
2. Design:
   CREATE INDEX idx_ptotaldose ON vaccine_p_state(_1st_dose_allocations, _2nd__dose_allocation);
3. Implementation:



```
mysql> DROP INDEX idx_m1 ON vaccine_m_state;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_ptotaldose ON vaccine_p_state(_1st_dose_allocations, _2nd_dose_allocations);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT DISTINCT m.jurisdiction, m.week_of_allocations, SUM(m._1st_dose_allocations + p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocatio
ns) AS totalDose FROM vaccine_m_state m JOIN vaccine_p_state p ON (m.jurisdiction = p.jurisdiction AND m.week_of_allocations = p.week_of_allocations) WHERE (m._1st_dose_allocations
+ p._1st_dose_allocations + m._2nd_dose_allocations + p._2nd_dose_allocations) > 0  GROUP BY m.jurisdiction, m.week_of_allocations ORDER BY m.week_of_allocations DESC, totalDose DES
C LIMIT 15;
```

4. Result:



```
| -> Limit: 15 row(s)  (actual time=4.295..4.298 rows=15 loops=1)
    -> Sort: m.week_of_allocations DESC, totalDose DESC, limit input to 15 row(s) per chunk  (actual time=4.295..4.296 rows=15 loops=1)
        -> Stream results  (cost=552.80 rows=1001) (actual time=0.121..3.976 rows=907 loops=1)
            -> Group aggregate: sum((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_alloc
ations))  (cost=552.80 rows=1001) (actual time=0.117..3.597 rows=907 loops=1)
                -> Nested loop inner join  (cost=452.70 rows=1001) (actual time=0.098..2.864 rows=907 loops=1)
                    -> Index scan on m using PRIMARY  (cost=102.35 rows=1001) (actual time=0.070..0.499 rows=1001 loops=1)
                    -> Filter: ((((m._1st_dose_allocations + p._1st_dose_allocations) + m._2nd_dose_allocations) + p._2nd_dose_allocatio
ns) > 0)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1001)
                        -> Single-row index lookup on p using PRIMARY (jurisdiction=m.jurisdiction, week_of_allocations=m.week_of_alloca
tions)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1001)
 |
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

5. Analysis:
   This index makes even greater contribution to speed up searching. Since in the query, for each state and date selected, two doses are always added. An index on both of them can avoid scanning them separately. Besides, both of them are part of totalDose in group-by clause, which also speeds up GROUP BY step.