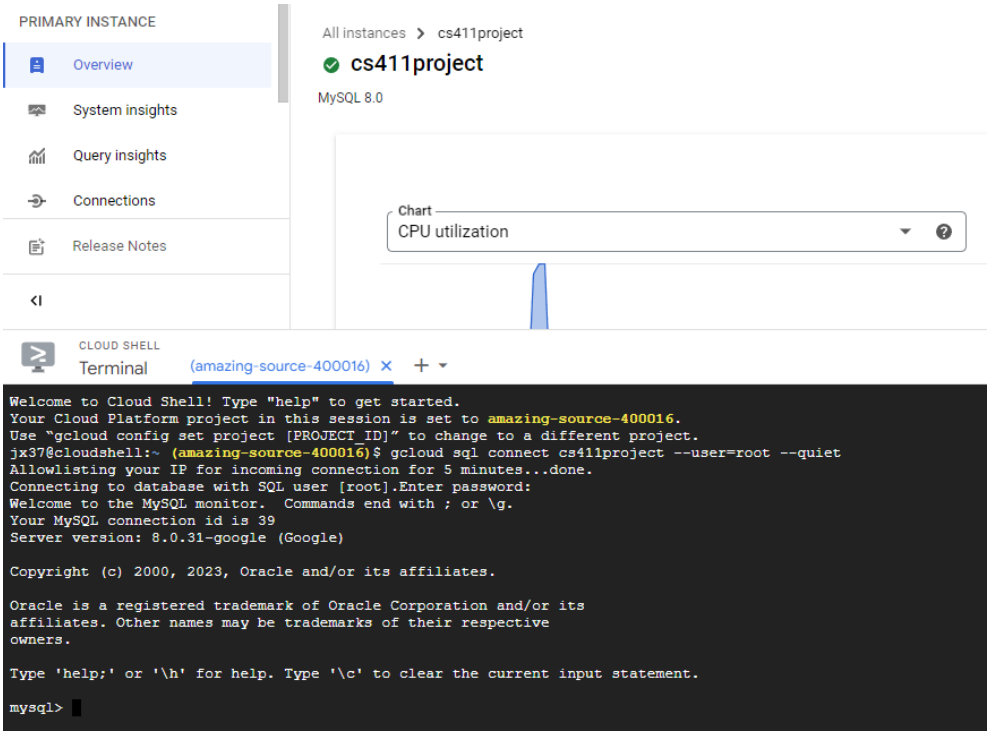
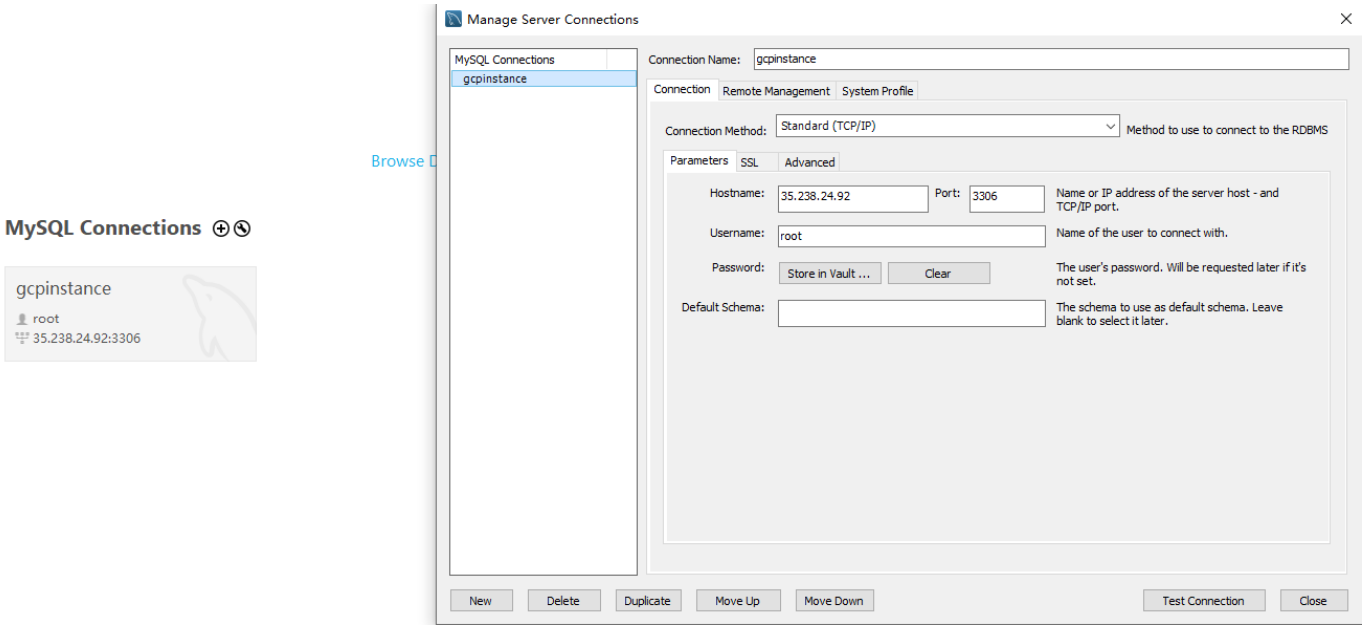
Stage 3: Database Implementation and Indexing

We implement our tables on GCP connecting with the Mysql Workbench. Here is a screenshot of the terminal on GCP and the server connection on Workbench:





The DDL commands for creating the tables are shown below:

```sql
USE cs411;
DROP TABLE IF EXISTS ToAndFrom;
DROP TABLE IF EXISTS Booking;
DROP TABLE IF EXISTS Trip;
DROP TABLE IF EXISTS Flight;
DROP TABLE IF EXISTS Airport;
DROP TABLE IF EXISTS Airline;
DROP TABLE IF EXISTS User;

CREATE TABLE IF NOT EXISTS User (
    USER_ID VARCHAR(255) PRIMARY KEY,
    USERNAME VARCHAR(255),
    PASSWORD VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS Airline (
    AIRLINE_NAME VARCHAR(255),
    IATA_CODE_LINE VARCHAR(255) PRIMARY KEY
);

CREATE TABLE IF NOT EXISTS Airport (
    AIRPORT_NAME VARCHAR(255),
    IATA_CODE_PORT VARCHAR(255) PRIMARY KEY,
    CITY VARCHAR(255),
    STATE VARCHAR(255),
    COUNTRY VARCHAR(255),
    LATITUDE VARCHAR(255),
    LONGITUDE VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS Flight (
    FLIGHT_ID VARCHAR(255) PRIMARY KEY,
    FLIGHT_NUMBER INT,
    AIRLINE_IATA VARCHAR(255),
    TAIL_NUMBER VARCHAR(255),
    CONSTRAINT fk_airline FOREIGN KEY (AIRLINE_IATA) REFERENCES Airline(IATA_CODE_LINE) ON DELETE CASCADE
);
```

```sql
CREATE TABLE IF NOT EXISTS Trip (
    FLIGHT_ID VARCHAR(255),
    FLIGHT_NUMBER INT,
    YEAR INT,
    MONTH INT,
    DAY INT,
    DAY_OF_WEEK INT,
    SCHEDULED_DEPARTURE INT,
    DEPARTURE_TIME INT,
    DEPARTURE_DELAY INT,
    TAXI_OUT INT,
    WHEELS_OFF INT,
    SCHEDULED_TIME INT,
    ELAPSED_TIME INT,
    AIR_TIME INT,
    DISTANCE INT,
    WHEELS_ON INT,
    TAXI_IN INT,
    SCEHDULED_ARRIVAL INT,
    ARRIVAL_TIME INT,
    ARRIVAL_DELAY INT,
    DIVERTED INT,
    CANCELED INT,
    CANCELLATION_REASON VARCHAR(255),
    AIR_SYSTEM_DELAY INT,
    SECURITY_DELAY INT,
    AIRLINE_DELAY INT,
    LATE_AIRCRAFT_DELAY INT,
    WEATHER_DELAY INT,
    ORIGIN_AIRPORT VARCHAR(255),
    TAIL_NUMBER VARCHAR(255),
    PRIMARY KEY (YEAR,DAY,MONTH, FLIGHT_NUMBER,ORIGIN_AIRPORT,TAIL_NUMBER),
    CONSTRAINT fk_flight FOREIGN KEY (FLIGHT_ID) REFERENCES Flight(FLIGHT_ID) ON DELETE CASCADE
);
```

```sql
CREATE TABLE IF NOT EXISTS Booking (
    BOOKING_ID VARCHAR(255),
    USER_ID VARCHAR(255) NOT NULL,
    PRICE INT,
    FLIGHT_ID VARCHAR(255),
    BOOKED_DATE DATETIME,
    PRIMARY KEY(USER_ID, BOOKING_ID),
    FOREIGN KEY (USER_ID) REFERENCES User (USER_ID) ON DELETE CASCADE,
    FOREIGN KEY (FLIGHT_ID) REFERENCES Trip(FLIGHT_ID) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS ToAndFrom (
    FLIGHT_ID VARCHAR(255),
    ORIGIN_AIRPORT VARCHAR(255),
    DESTINATION_AIRPORT VARCHAR(255),
    PRIMARY KEY (FLIGHT_ID, ORIGIN_AIRPORT, DESTINATION_AIRPORT),
    CONSTRAINT fk_f  FOREIGN KEY (FLIGHT_ID) REFERENCES Flight(FLIGHT_ID) ON DELETE CASCADE,
    CONSTRAINT fk_orgin FOREIGN KEY (ORIGIN_AIRPORT) REFERENCES Airport(IATA_CODE_PORT) ON DELETE CASCADE,
    CONSTRAINT fk_dest FOREIGN KEY (DESTINATION_AIRPORT) REFERENCES Airport(IATA_CODE_PORT) ON DELETE CASCADE
);
```

For each table, we selected the count of tuples:

```
77
78 • Select Count(*) from Flight;
79
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Count(*)
▸	2668

```
77
78 • Select Count(*) from Trip;
79
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Count(*)
▸	2668

```
77
78 • Select Count(*) from ToAndFrom;
79
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Count(*)
▸	2668

Here are the two advanced queries:

Advanced Query1:

```
SELECT
   AVG(Trip.DEPARTURE_DELAY) as dp_delay,
   AVG(Trip.ARRIVAL_DELAY) as ar_delay,
   f.AIRLINE_IATA
FROM
   Trip
JOIN
   ToAndFrom ON Trip.FLIGHT_ID = ToAndFrom.FLIGHT_ID
JOIN
   Flight f ON Trip.FLIGHT_ID = f.FLIGHT_ID
WHERE
   Trip.Year = 2015
   AND ToAndFrom.ORIGIN_AIRPORT = "ANC"
   AND ToAndFrom.DESTINATION_AIRPORT = "SEA"
GROUP BY
   f.AIRLINE_IATA
ORDER BY
   ar_delay DESC, dp_delay DESC;
```

```
SELECT
    AVG(Trip.DEPARTURE_DELAY) as dp_delay,
    AVG(Trip.ARRIVAL_DELAY) as ar_delay,
    f.AIRLINE_IATA
FROM
    Trip
JOIN
    ToAndFrom ON Trip.FLIGHT_ID = ToAndFrom.Flight_ID
JOIN
    Flight f ON Trip.FLIGHT_ID = f.FLIGHT_ID
WHERE
    Trip.Year = 2015
    AND ToAndFrom.ORIGIN_AIRPORT = "ANC"
    AND ToAndFrom.DESTINATION_AIRPORT = "SEA"
GROUP BY
    f.AIRLINE_IATA
ORDER BY
    ar_delay DESC, dp_delay DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dp_delay	ar_delay	AIRLINE_IATA
-9.3333	-18.6667	AS
-14.0000	-24.0000	DL

The reason there is only two data is because we only import 3000 data into the table which is only the flight at 2015.1.1. And there are only two corporations operating from ANC to SEA.

Indexing On Advanced Query 1:

```sql
use cs411;
Explain Analyze
SELECT
    AVG(Trip.DEPARTURE_DELAY) as dp_delay,
    AVG(Trip.ARRIVAL_DELAY) as ar_delay,
    f.AIRLINE_IATA
FROM
    Trip
JOIN
    ToAndFrom ON Trip.FLIGHT_ID = ToAndFrom.Flight_ID
JOIN
    Flight f ON Trip.FLIGHT_ID = f.FLIGHT_ID
WHERE
    Trip.Year = 2015
    AND ToAndFrom.ORIGIN_AIRPORT = "LAX"
    AND ToAndFrom.DESTINATION_AIRPORT = "ORD"
GROUP BY
    f.AIRLINE_IATA
ORDER BY
    ar_delay DESC, dp_delay DESC;
```

Form Editor | Navigate: 1 / 1

EXPLAIN:

```
-> Sort: ar_delay DESC, dp_delay DESC  (actual time=4.246..4.246 rows=2 loops=1)
  -> Table scan on <temporary>  (actual time=4.232..4.233 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=4.230..4.230 rows=2 loops=1)
      -> Nested loop inner join  (cost=25.34 rows=7) (actual time=1.737..4.200 rows=5 loops=1)
```

This is the result of performance of query1 before adding index.

After I added an index on Trip(year), the performance became better.
Setting index on Trip(year)

```sql
use cs411;
-- Drop Index idx_year On Trip;
CREATE INDEX tripyear ON Trip(Year);
Explain Analyze
SELECT
    AVG(Trip.DEPARTURE_DELAY) as dp_delay,
    AVG(Trip.ARRIVAL_DELAY) as ar_delay,
    f.AIRLINE_IATA
FROM
    Trip
JOIN
    ToAndFrom ON Trip.FLIGHT_ID = ToAndFrom.Flight_ID
JOIN
    Flight f ON Trip.FLIGHT_ID = f.FLIGHT_ID
WHERE
    Trip.Year = 2015
    AND ToAndFrom.ORIGIN_AIRPORT = "LAX"
    AND ToAndFrom.DESTINATION_AIRPORT = "ORD"
GROUP BY
    f.AIRLINE_IATA
ORDER BY
    ar_delay DESC, dp_delay DESC;
```

Form Editor | Navigate: 1 / 1

EXPLAIN:

```
-> Sort: ar_delay DESC, dp_delay DESC  (actual time=0.237..0.237 rows=2 loops=1)
  -> Table scan on <temporary>  (actual time=0.214..0.215 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=0.213..0.213 rows=2 loops=1)
      -> Nested loop inner join  (cost=20.09 rows=7) (actual time=0.059..0.191 rows=5 loops=1)
```

The second Strategy is add index in the attribute of group by

```sql
use cs411;
SHOw index From ToAndFrom;

CREATE INDEX index_iata ON Flight(AIRLINE_IATA);

explain analyze
SELECT
    AVG(Trip.DEPARTURE_DELAY) as dp_delay,
    AVG(Trip.ARRIVAL_DELAY) as ar_delay,
    f.AIRLINE_IATA
FROM
    Trip
JOIN
    ToAndFrom ON Trip.FLIGHT_ID = ToAndFrom.Flight_ID
JOIN
    Flight f ON Trip.FLIGHT_ID = f.FLIGHT_ID
WHERE
    Trip.Year = 2015
    AND ToAndFrom.ORIGIN_AIRPORT = "ANC"
    AND ToAndFrom.DESTINATION_AIRPORT = "SEA"
GROUP BY
    f.AIRLINE_IATA
ORDER BY
    ar_delay DESC, dp_delay DESC;
```

Form Editor | Navigate: 1 / 1

EXPLAIN:

```
-> Sort: ar_delay DESC, dp_delay DESC  (actual time=0.171..0.171 rows=2 loops=1)
  -> Table scan on <temporary>  (actual time=0.161..0.161 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=0.160..0.160 rows=2 loops=1)
      -> Nested loop inner join  (cost=3.59 rows=0.3) (actual time=0.032..0.140 rows=7 loops=1)
```

The Third Strategy is add index on attribute of Join clause, FLIGHT_ID from Trip and ToAndFrom

```
CREATE INDEX index_flight_id ON ToAndFrom(Flight_ID);
CREATE INDEX index_flight_id ON Trip(Flight_ID);
-- Drop index index_iata on Flight;
explain analyze
SELECT
    AVG(Trip.DEPARTURE_DELAY) as dp_delay,
    AVG(Trip.ARRIVAL_DELAY) as ar_delay,
    f.AIRLINE_IATA
FROM
    Trip
JOIN
    ToAndFrom ON Trip.FLIGHT_ID = ToAndFrom.Flight_ID
JOIN
    Flight f ON Trip.FLIGHT_ID = f.FLIGHT_ID
WHERE
    Trip.Year = 2015
    AND ToAndFrom.ORIGIN_AIRPORT = "ANC"
    AND ToAndFrom.DESTINATION_AIRPORT = "SEA"
GROUP BY
    f.AIRLINE_IATA
ORDER BY
    ar_delay DESC, dp_delay DESC;
```

Form Editor | Navigate: 1 / 1

EXPLAIN:

```
-> Sort: ar_delay DESC, dp_delay DESC  (actual time=1.152..1.153 rows=2 loops=1)
  -> Table scan on <temporary>  (actual time=1.140..1.141 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=1.139..1.139 rows=2 loops=1)
      -> Nested loop inner join  (cost=3.76 rows=0.3) (actual time=0.041..1.117 rows=7 loops=1)
```

I choose to create an index on the attribute of Join which is FLIGHT_ID from Trip and ToAndFromm, attribute of Where clause which is Year from Trip, and attribute of Group by which is AIRLINE_IATA from Flight. The reason that I set the index at Join is when we join tables with conditions, the database needs to search through Trip, Flight,ToAndFrom to find match rows and join together. After I create an index on those attributes, the database can find those match rows more quickly, which enhances the performance. Also the result of EXPLAIN ANALYZE shows that the cost has dropped from 25.34 to 3.76.

When we use the where clause to set limitations to find the row we want, the database needs to check each row to determine which one satisfies the requirements. After we add index on attribute of where clause, the database does not need to search through all the rows, which reduces the time it consumes. By checking the result of EXPLAIN ANALYZE shows that the cost has dropped from 25.34 to 20.09, it can show the enhancement of speed on the query.

The reason that choosing a set index on group by is that the group by will search through each row to find the columns with the same value and group them together. By setting an index on it, which already sorts rows with the same value of column, it will save a lot of time. And the result of EXPLAIN ANALYZE shows that the cost has dropped from 25.34 to 3.59, which reflects that the set index in group by is a good strategy to enhance the query performance.

Advanced Query2:

<u>Calculating the non-delay rate where DEPARTURE_DELAY is not positive, grouped by the origin airport (and **the final optimized version** is shown in indexing):</u>

```
select delay.NonDelay, delay.DepartureCount,  (delay.NonDelay / delay.DepartureCount) as
NonDelayRate, delay.ORIGIN_AIRPORT
FROM
(select count(a.DEPARTURE_DELAY) as NonDelay, (
        select count(DEPARTURE_DELAY)
   FROM Flight NATURAL JOIN ToAndFrom NATURAL JOIN Trip
   GROUP by ORIGIN_AIRPORT
   HAVING ORIGIN_AIRPORT = a.ORIGIN_AIRPORT
) as DepartureCount, ORIGIN_AIRPORT
FROM Flight NATURAL JOIN ToAndFrom NATURAL JOIN Trip as a
WHERE a.DEPARTURE_DELAY <= 0   -- early departure has a negative DEPARTURE_DELAY
GROUP BY a.ORIGIN_AIRPORT) AS delay
HAVING delay.DepartureCount > 50      -- for popular airport
order by NonDelayRate DESC
LIMIT 15
```

```
use cs411;
select delay.NonDelay, delay.DepartureCount,  (delay.NonDelay / delay.DepartureCount) as NonDelayRate, delay.ORIGIN_AIRPORT
FROM
(select count(a.DEPARTURE_DELAY) as NonDelay, (
    select count(DEPARTURE_DELAY)
    FROM Flight NATURAL JOIN ToAndFrom NATURAL JOIN Trip
    GROUP by ORIGIN_AIRPORT
    HAVING ORIGIN_AIRPORT = a.ORIGIN_AIRPORT
) as DepartureCount, ORIGIN_AIRPORT
FROM Flight NATURAL JOIN ToAndFrom NATURAL JOIN Trip as a
WHERE a.DEPARTURE_DELAY <= 0   -- early departure has a negative DEPARTURE_DELAY
GROUP BY a.ORIGIN_AIRPORT) AS delay
HAVING delay.DepartureCount > 50      -- for popular airport
order by NonDelayRate DESC
LIMIT 15;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

NonDelay	DepartureCount	NonDelayRate	ORIGIN_AIRPORT
91	113	0.8053	LAX
57	71	0.8028	LAS
51	65	0.7846	MCO
53	69	0.7681	BOS
40	54	0.7407	MSP
40	55	0.7273	FLL
52	72	0.7222	ATL
54	77	0.7013	JFK
64	92	0.6957	SFO
38	55	0.6909	DTW
38	57	0.6667	EWR
54	83	0.6506	SEA
38	62	0.6129	IAH
45	75	0.6000	DEN
42	75	0.5600	ORD

Indexing On Advanced Query 2:

- The performance before editing and indexing is shown below:

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=2789.292..2789.293 rows=15 loops=1)
  -> Sort: NonDelayRate DESC, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=2789.291..2789.292 rows=15 loops=1)
    -> Filter: (delay.DepartureCount > 50)  (cost=2.50..2.50 rows=0) (actual time=2789.237..2789.273 rows=17 loops=1)
      -> Table scan on delay  (cost=2.50..2.50 rows=0) (actual time=2789.234..2789.257 rows=224 loops=1)
```

- After finding the execution speed being so slow, I looked through the code again and found that joining tables is unnecessary. Hence, I edited line 6 and line 10 so that the data is selected from the sole table **Trip**. As we can see, without joining operations, the execution speed increases a lot from about **2.789s** to **0.427s**.

```
1  explain analyze
2  select delay.NonDelay, delay.DepartureCount,  (delay.NonDelay / delay.DepartureCount) as NonDelayRate, delay.ORIGIN_AIRPORT
3  FROM
4  (select count(a.DEPARTURE_DELAY) as NonDelay, (
5      select count(DEPARTURE_DELAY)
6      FROM Trip
7      GROUP by ORIGIN_AIRPORT
8      HAVING ORIGIN_AIRPORT = a.ORIGIN_AIRPORT
9  ) as DepartureCount, ORIGIN_AIRPORT
10 FROM Trip as a
11 WHERE a.DEPARTURE_DELAY <= 0   -- early departure has a negative DEPARTURE_DELAY
12 GROUP BY a.ORIGIN_AIRPORT) AS delay
13 HAVING delay.DepartureCount > 50    -- for popular airport
14 order by NonDelayRate DESC
15 LIMIT 15;
```

Form Editor | Navigate:

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=427.780..427.782 rows=15 loops=1)
  -> Sort: NonDelayRate DESC, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=427.780..427.781 rows=15 loops=1)
    -> Filter: (delay.DepartureCount > 50)  (cost=2.50..2.50 rows=0) (actual time=427.730..427.763 rows=17 loops=1)
      -> Table scan on delay  (cost=2.50..2.50 rows=0) (actual time=427.729..427.751 rows=224 loops=1)
```

- Then, since the query wants to find rows with non-negative values of DEPARTURE_DELAY, I tried to create an index on this column.

```
CREATE INDEX idx_dep_delay
ON Trip(DEPARTURE_DELAY);
```

- After executing it using index, the execution time improves slightly however the cost of each step remains the same. The reason is that the table scan on delay (order by NonDelayRate DESC), where delay is a alias for a subquery, can't be optimized with this index on Trip.

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=380.351..380.353 rows=15 loops=1)
  -> Sort: NonDelayRate DESC, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=380.350..380.351 rows=15 loops=1)
    -> Filter: (delay.DepartureCount > 50)  (cost=2.50..2.50 rows=0) (actual time=380.302..380.333 rows=17 loops=1)
      -> Table scan on delay  (cost=2.50..2.50 rows=0) (actual time=380.298..380.321 rows=224 loops=1)
```

- Therefore, I'm thinking about deleting one subquery and using "when… then…" to count the "NonDelay". After changing the format, the execution time improves a lot from **0.41s** to **0.031s**

```
explain analyze
select delay.NonDelay, delay.DepartureCount,  (delay.NonDelay / delay.DepartureCount) as NonDelayRate, delay.ORIGIN_AIRPORT
FROM (
SELECT
    COUNT(CASE WHEN Trip.DEPARTURE_DELAY <= 0 THEN 1 ELSE NULL END) as NonDelay,
    COUNT(*) as DepartureCount,
    ORIGIN_AIRPORT
FROM
    Trip
GROUP BY
    Trip.ORIGIN_AIRPORT
HAVING
    DepartureCount > 50
) as delay
ORDER BY
    NonDelayRate DESC
LIMIT 15;
```

Form Editor | Navigate: 1 / 1

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=2.171..2.173 rows=15 loops=1)
  -> Sort: NonDelayRate DESC, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=2.171..2.171 rows=15 loops=1)
    -> Table scan on delay  (cost=2.50..2.50 rows=0) (actual time=2.153..2.155 rows=17 loops=1)
      -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.153..2.153 rows=17 loops=1)
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.03100000

Timing (as measured by the server):
Execution time: 0:00:0.00258391
Table lock wait time: 0:00:0.00000200

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 15
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 1

- I tried to add the index on Departure_Delay:

EXPLAIN:

```
-> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=2.121..2.123 rows=15 loops=1)
   -> Sort: NonDelayRate DESC, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=2.121..2.122 rows=15 loops=1)
      -> Table scan on delay  (cost=2.50..2.50 rows=0) (actual time=2.104..2.106 rows=17 loops=1)
         -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.104..2.104 rows=17 loops=1)
```

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.01500000

Timing (as measured by the server):
Execution time: 0:00:0.00231746
Table lock wait time: 0:00:0.00000200

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:

Joins per Type:
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 15
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 1

As we can see, although the execution time slightly reduced from **0.031s** to **0.015s,** the cost remains the same since the sorting with table scans on delay (order by NonDelayRate DESC) can't be optimized by indexing created on delay.

//// followed are development code, not in formal use

```
Select f.AIRLINE_IATA ,Avg(t.DEPARTURE_DELAY) as dp_delay,  Avg(t.ARRIVAL_DELAY) as
ar_delay
From
(select *
from ToAndFrom
Where ORIGIN_AIRPORT = “X1”(required) and DESTINATION_AIRPORT = “X2”(required)
) as taf
join
Flight f
Join
(select * from Trip
Where date = “xxx”(optional)
) as t
Group by f.AIRLINE_IATA
Order by ar_delay desc, dp_delay desc
```

```
Select count(t.FLIGHT_ID) as total_run, sum(t.DIVERTED), sum(t.CANCELED),
sum(t.AIR_SYSTEM_DELAY), sum(t.SECURITY_DELAY ), sum(t.AIRLINE_DELAY
),sum(t.LATE_AIRCRAFT_DELAY),sum(t.WEATHER_DELAY)
from
(select *
from ToAndFrom
Where ORIGIN_AIRPORT = “X1”(optional) and DESTINATION_AIRPORT = “X2”(optional)
) as taf
join
From
(select * from Trip
Where date = “xxx”(optional)
) as t
Join
Flight f
Group by f.AIRLINE_IATA
Order by ar_delay desc, total_run desc
```

// the followed one has not changed yet

Further development possible for broadcast the ori_airport and dest_airport into the country to country using the table airport

Like changing the Where `ORIGIN_AIRPORT` = “X1” to

Where `FLIGHT_ID` in(

```
Select f2.FLIGHT_ID
From ToAndFrom
Where ToAndFrom.ORIGIN_AIRPORT in
        (select ap1.IATA_CODE
        From Airport ap1
        Where ap1.COUNTRY(or CITY) = “X1”)
and ToAndFrom.DESTINATION_AIRPORT  in
        (select ap1.IATA_CODE
        From Airport ap1
        Where ap1.COUNTRY(or CITY) = “X1”)
)
```