Stage 3 - Database Design

Part 1:

Data Definition Language (DDL) commands:

Airports table

```
CREATE TABLE airports(
      IATA_CODE VARCHAR(3) PRIMARY KEY,
      AIRPORT VARCHAR(255),
      CITY VARCHAR(255),
      STATE VARCHAR(2)
);
```

Airlines table

```
CREATE TABLE airlines(
      IATA_CODE VARCHAR(2) PRIMARY KEY,
      AIRLINE VARCHAR(255)
);
```

AirlineRating table

```
Create table AirlineRating(
      AIRLINE VARCHAR(255) PRIMARY KEY,
      ReliabilityRating INT,
      OnTimePerformanceRating INT,
      FOREIGN KEY(AIRLINE) REFERENCES airlines(IATA_CODE)
);
```

Flight table

```
CREATE TABLE flights(
      DATE VARCHAR(255),
      DAY_OF_WEEK INT,
      AIRLINE VARCHAR(3),
      FLIGHT_NUMBER INT,
      ORIGIN_AIRPORT VARCHAR(3),
```

```
      DESTINATION_AIRPORT VARCHAR(3),
      PRIMARY KEY(DATE, AIRLINE, FLIGHT_NUMBER),
      FOREIGN KEY(ORIGIN_AIRPORT) REFERENCES airports(IATA_CODE),
      FOREIGN KEY(DESTINATION_AIRPORT) REFERENCES
airports(IATA_CODE),
      FOREIGN KEY(AIRLINE) REFERENCES airlines(IATA_CODE)
);
```

Delay table

```
Create Table delays(
      DATE VARCHAR(255),
      AIRLINE VARCHAR(3) ,
      FLIGHT_NUMBER INT ,
      SCHEDULED_DEPARTURE INT,
      DEPARTURE_TIME INT,
      DEPARTURE_DELAY INT,
      SCHEDULED_TIME INT,
      ELAPSED_TIME INT,
      AIR_TIME INT,
      DISTANCE INT,
      SCHEDULED_ARRIVAL INT,
      ARRIVAL_TIME INT,
      ARRIVAL_DELAY INT,
      DIVERTED INT,
      CANCELLED INT,
      CANCELLATION_REASON VARCHAR(1),
      AIR_SYSTEM_DELAY INT,
      SECURITY_DELAY INT,
      AIRLINE_DELAY INT,
      LATE_AIRCRAFT_DELAY INT,
      WEATHER_DELAY INT,
      PRIMARY KEY(DATE, AIRLINE, FLIGHT_NUMBER),
      FOREIGN KEY(DATE, AIRLINE,FLIGHT_NUMBER) REFERENCES
flights(DATE,    AIRLINE,FLIGHT_NUMBER)
);
```

Inserting Data & Table counts

In GCP, we uploaded the CSV data into buckets, and imported the values into SQL directly. The original flights dataset only had ~300 rows for the airports table and 14 rows for the airlines table. We used a Python program to add dummy data to pad our table with the required 1000 rows. We also removed duplicate entries from the delays table, which brought down the count from ~1M to 750K.

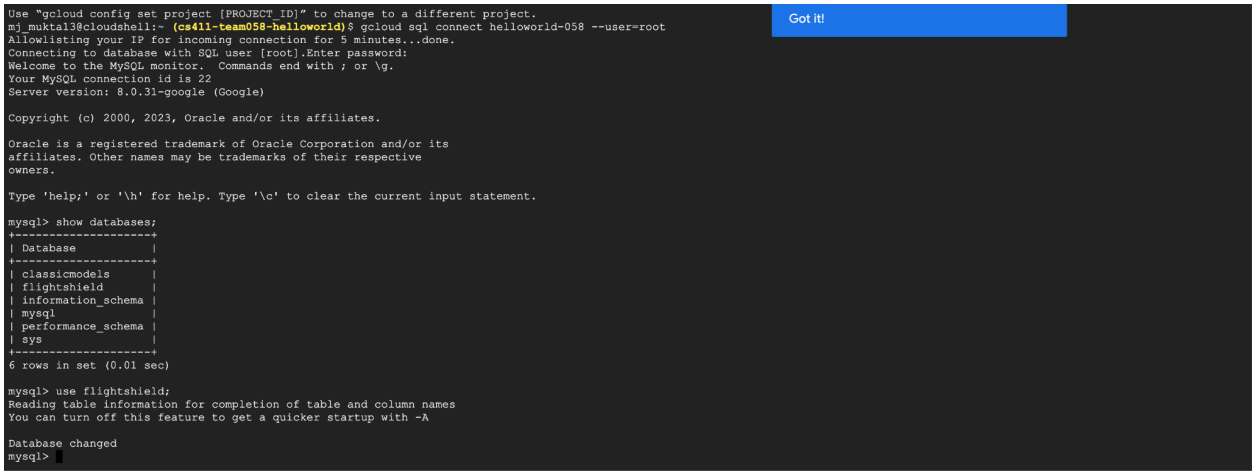
The final count of rows in each table is as follows:

1. airports: 1001 rows
2. Flights: 749794 rows
3. delays: 749794 rows
4. airlines: 1000 rows
5. AirlineRating: 1000 rows

```
mysql> SELECT COUNT(*) FROM airports;
+----------+
| COUNT(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM flights;
+----------+
| COUNT(*) |
+----------+
|   749794 |
+----------+
1 row in set (0.80 sec)

mysql> SELECT COUNT(*) FROM delays;
+----------+
| COUNT(*) |
+----------+
|   749794 |
+----------+
1 row in set (1.29 sec)

mysql> SELECT COUNT(*) FROM airlines;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM AirlineRating;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)
```

Connection Screenshot

Screenshot:



Advanced Queries:

Note: In this section, the value of delay is in minutes. A negative value of -x indicates that the airline was **early** by x minutes. A positive value of y indicates that the airline was **delayed** by y minutes.

1) We are calculating the total average departure delay for every flight belonging to distinct airlines. We are joining the values from delays and airlines tables, and grouping by Airline name to calculate the aggregate average per airline.

```
SELECT a.AIRLINE, AVG(d.DEPARTURE_DELAY) AS Avg_Delay
FROM delays d
JOIN airlines a ON d.AIRLINE = a.IATA_CODE
GROUP BY a.AIRLINE
ORDER BY Avg_Delay DESC;
```

```
mysql> SELECT a.AIRLINE, AVG(d.DEPARTURE_DELAY) AS Avg_Delay
    -> FROM delays d
    -> JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND d.FLIGHT_NUMBER = f.FLIGHT_NUMBER JOIN airlines a ON a.IATA_CODE = f.AIRLINE
    -> GROUP BY a.AIRLINE
    -> ORDER BY Avg_Delay DESC;

+------------------------------+-----------+
| AIRLINE                      | Avg_Delay |
+------------------------------+-----------+
| Frontier Airlines Inc.       |   19.5482 |
| American Eagle Airlines Inc. |   14.2178 |
| Spirit Air Lines             |   13.6022 |
| Skywest Airlines Inc.        |   12.1546 |
| United Air Lines Inc.        |   12.1373 |
| JetBlue Airways              |   10.6637 |
| American Airlines Inc.       |    9.8684 |
| Atlantic Southeast Airlines  |    9.2606 |
| Delta Air Lines Inc.         |    6.8077 |
| Southwest Airlines Co.       |    6.5269 |
| Virgin America               |    6.4416 |
| US Airways Inc.              |    5.2704 |
| Alaska Airlines Inc.         |    3.1394 |
| Hawaiian Airlines Inc.       |    1.0178 |
+------------------------------+-----------+
14 rows in set (2.02 sec)
```

The output is just 14 rows because our original data set only has information about 14 airlines (the remaining dummy airlines data we added does not have corresponding entries in the flights and delays table).

2) We are selecting the top 15 delayed flights due to DEPARTURE DELAYS based on the 4 parameters: Origin, Destination, Depart Date, and Return Date. This query simulates how a user might want to search for delayed flights based on their actual itinerary in our end application.

```
SELECT a.AIRLINE, AVG(d.DEPARTURE_DELAY) AS Avg_Delay
FROM delays d
JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND
d.FLIGHT_NUMBER = f.FLIGHT_NUMBER
JOIN airlines a ON a.IATA_CODE = f.AIRLINE
WHERE f.ORIGIN_AIRPORT = 'JFK'
AND f.DESTINATION_AIRPORT = 'SFO'
AND f.DATE = '1-Jan'
GROUP BY a.AIRLINE
ORDER BY Avg_Delay ASC;
```

```
mysql> SELECT a.AIRLINE, AVG(d.DEPARTURE_DELAY) AS Avg_Delay FROM delays d JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND d.FLIGHT_NUMBER = f.FLIGHT_NUMBER
JOIN airlines a ON a.IATA_CODE = f.AIRLINE WHERE f.ORIGIN_AIRPORT = 'JFK' AND f.DESTINATION_AIRPORT = 'SFO' AND f.DATE = '1-Jan' GROUP BY a.AIRLINE ORDER BY Avg_Delay ASC;
+------------------------+-----------+
| AIRLINE                | Avg_Delay |
+------------------------+-----------+
| American Airlines Inc. |   -5.7500 |
| JetBlue Airways        |   -1.6667 |
| United Air Lines Inc.  |   -0.6000 |
| Delta Air Lines Inc.   |   -0.2500 |
| Virgin America         |   12.3333 |
+------------------------+-----------+
5 rows in set (0.00 sec)
```

The output has just 5 rows because there are only 5 distinct airlines that serve the JFK to SFO route in our dataset.

3) Finally, similar to queries #2, we are calculating the average delay based on the total delay (i.e. departure delay plus arrival delay). We also simulated a real-world use case where a customer might want to check the anticipated delay for each airline, close to their holiday bookings. We tried superbowl weekend (30 Jan) and Valentine's day (14 Feb). We were able to assert that the delay values were significantly larger on these specific dates as anticipated.

```
SELECT a.AIRLINE, AVG(d.ARRIVAL_DELAY + d.DEPARTURE_DELAY) AS
Avg_Delay
FROM delays d
JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND
d.FLIGHT_NUMBER = f.FLIGHT_NUMBER
JOIN airlines a ON a.IATA_CODE = f.AIRLINE
WHERE f.ORIGIN_AIRPORT = 'JFK'
AND f.DESTINATION_AIRPORT = 'PHX'
AND f.DATE = '30-JAN'
GROUP BY a.AIRLINE;
```

```
mysql> SELECT a.AIRLINE, AVG(d.ARRIVAL_DELAY + d.DEPARTURE_DELAY) AS Avg_Delay
    -> FROM delays d
    -> JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND d.FLIGHT_NUMBER = f.FLIGHT_NUMBER
    -> JOIN airlines a ON a.IATA_CODE = f.AIRLINE
    -> WHERE f.ORIGIN_AIRPORT = 'JFK'
    -> AND f.DESTINATION_AIRPORT = 'PHX'
    -> AND f.DATE = '30-JAN'
    -> GROUP BY a.AIRLINE;
+---------------------+-----------+
| AIRLINE             | Avg_Delay |
+---------------------+-----------+
| JetBlue Airways     |   68.5000 |
| Delta Air Lines Inc. |  182.0000 |
| US Airways Inc.     |  140.0000 |
+---------------------+-----------+
3 rows in set (0.02 sec)
```

```
mysql> SELECT a.AIRLINE, AVG(d.ARRIVAL_DELAY + d.DEPARTURE_DELAY) AS Avg_Delay
    -> FROM delays d
    -> JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND d.FLIGHT_NUMBER = f.FLIGHT_NUMBER
    -> JOIN airlines a ON a.IATA_CODE = f.AIRLINE
    -> WHERE f.ORIGIN_AIRPORT = 'JFK'
    -> AND f.DESTINATION_AIRPORT = 'SFO'
    -> AND f.DATE = '14-Feb'
    -> GROUP BY a.AIRLINE;
+-----------------------+-----------+
| AIRLINE               | Avg_Delay |
+-----------------------+-----------+
| American Airlines Inc. |  127.2000 |
| JetBlue Airways       |   -2.5000 |
| Delta Air Lines Inc.  |   60.2500 |
| United Air Lines Inc. |   37.8000 |
| Virgin America        |   -7.5000 |
+-----------------------+-----------+
5 rows in set (0.03 sec)
```

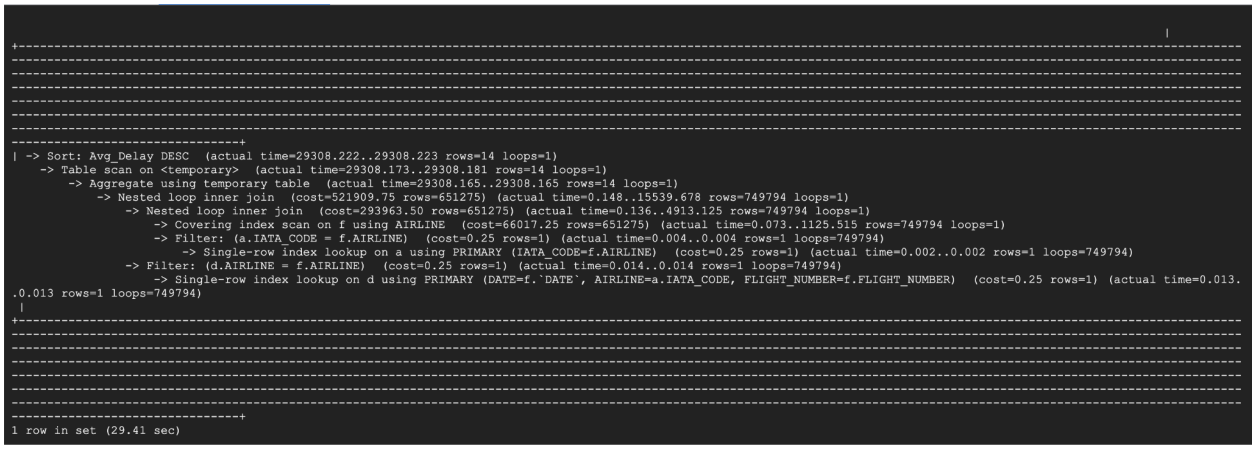
Similar to the output of query #2, this query also has just a few rows returned because there are only 5 distinct airlines that serve the JFK to SFO route in our dataset, and 3 distinct airlines that serve JFK to PHX route.

Part 2: Explain-Analyze and Indexing

Query #1:

```
SELECT a.AIRLINE, AVG(d.DEPARTURE_DELAY) AS Avg_Delay
FROM delays d
JOIN airlines a ON d.AIRLINE = a.IATA_CODE
GROUP BY a.AIRLINE
ORDER BY Avg_Delay DESC;
```

EXPLAIN ANALYZE without indexing:



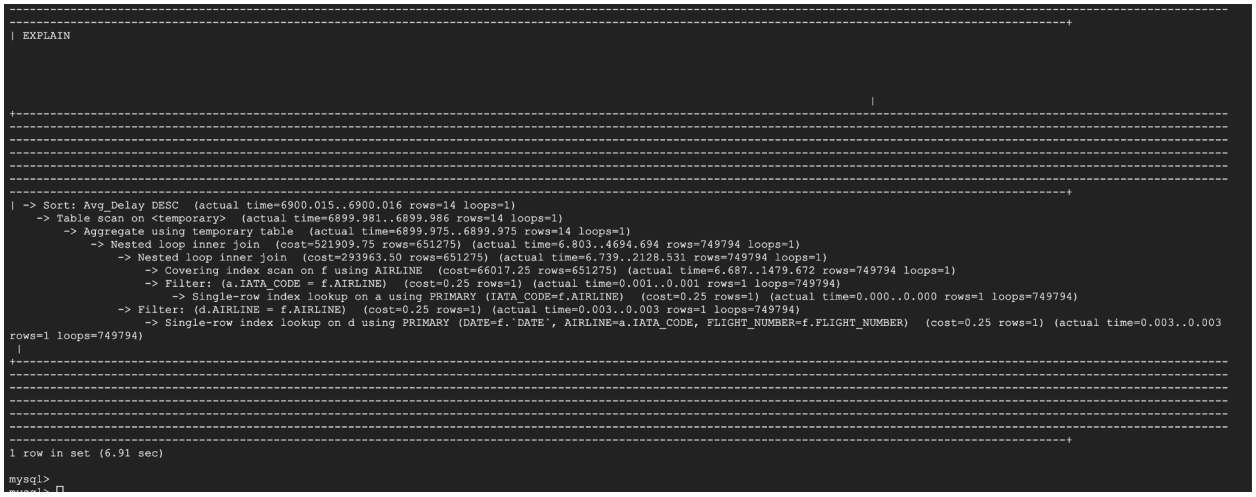
```
| -> Sort: Avg_Delay DESC  (actual time=29308.222..29308.223 rows=14 loops=1)
    -> Table scan on <temporary>  (actual time=29308.173..29308.181 rows=14 loops=1)
        -> Aggregate using temporary table  (actual time=29308.165..29308.165 rows=14 loops=1)
            -> Nested loop inner join  (cost=521909.75 rows=651275) (actual time=0.148..15539.678 rows=749794 loops=1)
                -> Nested loop inner join  (cost=293963.50 rows=651275) (actual time=0.136..4913.125 rows=749794 loops=1)
                    -> Covering index scan on f using AIRLINE  (cost=66017.25 rows=651275) (actual time=0.073..1125.515 rows=749794 loops=1)
                    -> Filter: (a.IATA_CODE = f.AIRLINE)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=749794)
                        -> Single-row index lookup on a using PRIMARY (IATA_CODE=f.AIRLINE)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=749794)
                -> Filter: (d.AIRLINE = f.AIRLINE)  (cost=0.25 rows=1) (actual time=0.014..0.014 rows=1 loops=749794)
                    -> Single-row index lookup on d using PRIMARY (DATE=f.`DATE`, AIRLINE=a.IATA_CODE, FLIGHT_NUMBER=f.FLIGHT_NUMBER)  (cost=0.25 rows=1) (actual time=0.013.
.0.013 rows=1 loops=749794)
 |
1 row in set (29.41 sec)
```

The original query is found to have a cost of **0.25** with a timing of **0.013s.**

- First we created an index based on the Departure Delays

Create index:

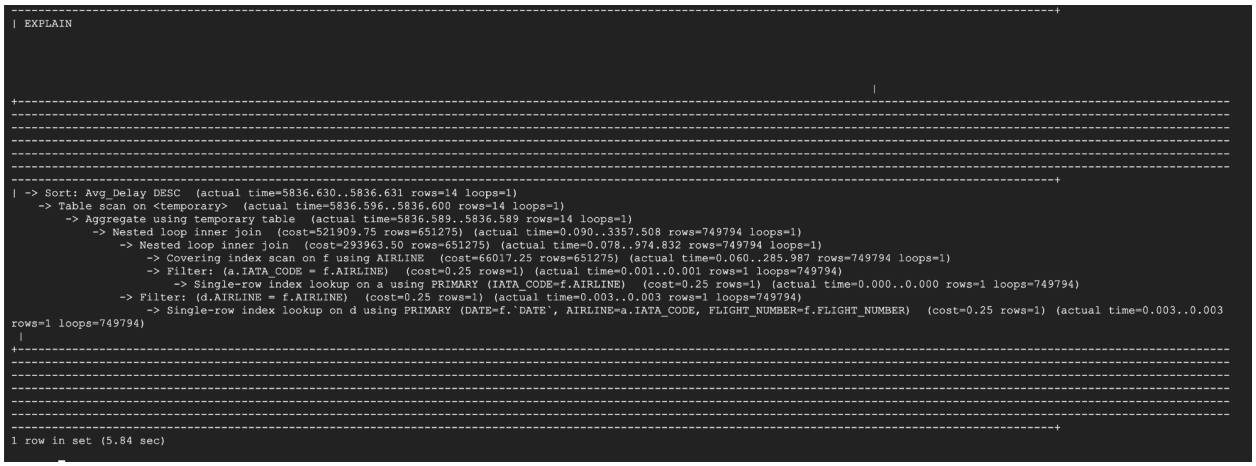
```
create index delay_idx on delays(DEPARTURE_DELAY);
```



The indexed query on delay_idx on flights is found to have an unchanged cost of **0.25** with a timing of **0.003s.** Hence while there is an improvement in timing, the cost was unchanged, hence we continued to try other options.

- Next, we attempted to index based on Airline names.

```
create index airlines_idx on airlines(AIRLINE(5));
```



For the indexed query on Airline Names, there is found to be an unchanged cost of **0.25** with a timing of **0.003s.** The reason for the cost remaining unchanged and the time improving is probably because the database only has 14 airlines (the remaining dummy airlines data we added does not have corresponding entries in the flights and delays table).**The second reason is that since airline is a primary key, it is already indexed.**
Hence as there are no cost improvements, it is determined that indexing based on Airlines is not an optimal indexing.

- Then, we attempted to index based on both Airline names and Departure Delays.

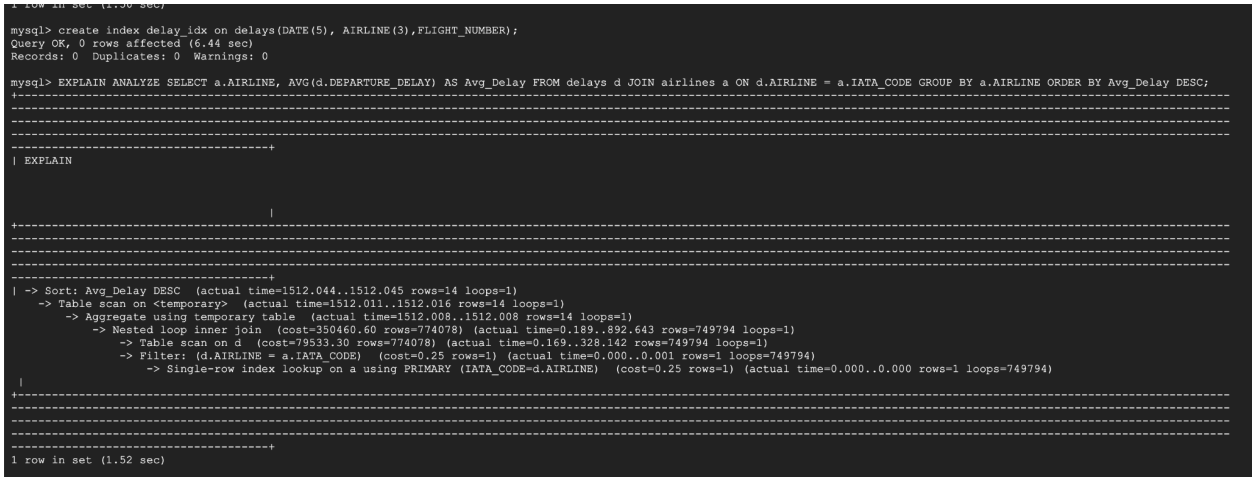
```
create index airlines_idx on airlines(AIRLINE(5));
create index delay_idx on delays(DEPARTURE_DELAY);
```

```
--------------------------------------------------------------------------------+
| EXPLAIN
                                                                                |
+-------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| -> Sort: Avg_Delay DESC  (actual time=5125.312..5125.313 rows=14 loops=1)
    -> Table scan on <temporary>  (actual time=5125.279..5125.284 rows=14 loops=1)
        -> Aggregate using temporary table  (actual time=5125.273..5125.273 rows=14 loops=1)
            -> Nested loop inner join  (cost=521909.75 rows=651275) (actual time=0.086..3026.006 rows=749794 loops=1)
                -> Nested loop inner join  (cost=293963.50 rows=651275) (actual time=0.075..877.169 rows=749794 loops=1)
                    -> Covering index scan on f using AIRLINE  (cost=66017.25 rows=651275) (actual time=0.052..253.896 rows=749794 loops=1)
                    -> Filter: (a.IATA_CODE = f.AIRLINE)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=749794)
                        -> Single-row index lookup on a using PRIMARY (IATA_CODE=f.AIRLINE)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=749794)
                -> Filter: (d.AIRLINE = f.AIRLINE)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=749794)
                    -> Single-row index lookup on d using PRIMARY (DATE=f.`DATE`, AIRLINE=a.IATA_CODE, FLIGHT_NUMBER=f.FLIGHT_NUMBER)  (cost=0.25 rows=1) (actual time=0.002..0.002
rows=1 loops=749794)
 |
+-------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
1 row in set (5.12 sec)
```

For the indexed query on both delay_idx and airlines_idx, there is found to be an unchanged cost of **0.25** with a timing improvement of **0.002s. Which can be attributed to the database only having 14 airlines as seen in the earlier indexing and airline being the primary key is already indexed.**

- Then, we attempted to index based on Date, Airline names and Flight Number.

```
create index delay_idx on delays(DATE(5), AIRLINE(3),
FLIGHT_NUMBER);
```



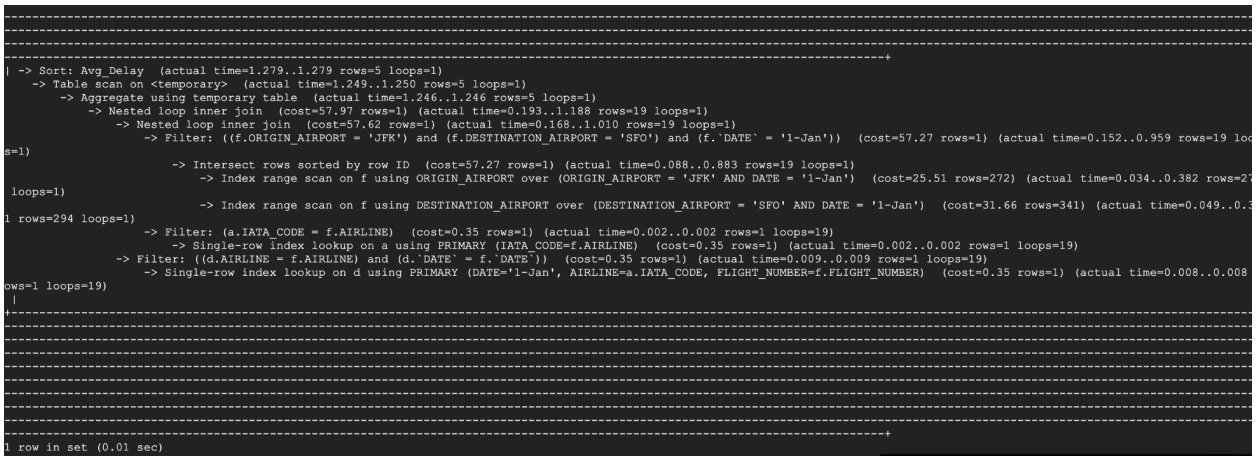
For the indexed query on both delay_idx and airlines_idx, there is found to be an unchanged cost of **0.25** with a timing improvement to **0.000s. Since all the three attributes are primary keys, Indexing has no effect.**

Hence we decided to go with the non-indexed or default query, since adding an index does not seem to be very helpful in this situation, and is not reducing cost and timing both in any instances.

Query #2:

```
SELECT a.AIRLINE, AVG(d.DEPARTURE_DELAY) AS Avg_Delay
FROM delays d
JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND
d.FLIGHT_NUMBER = f.FLIGHT_NUMBER
JOIN airlines a ON a.IATA_CODE = f.AIRLINE
WHERE f.ORIGIN_AIRPORT = 'JFK'
AND f.DESTINATION_AIRPORT = 'SFO'
AND f.DATE = '1-Jan'
GROUP BY a.AIRLINE
ORDER BY Avg_Delay ASC;
```

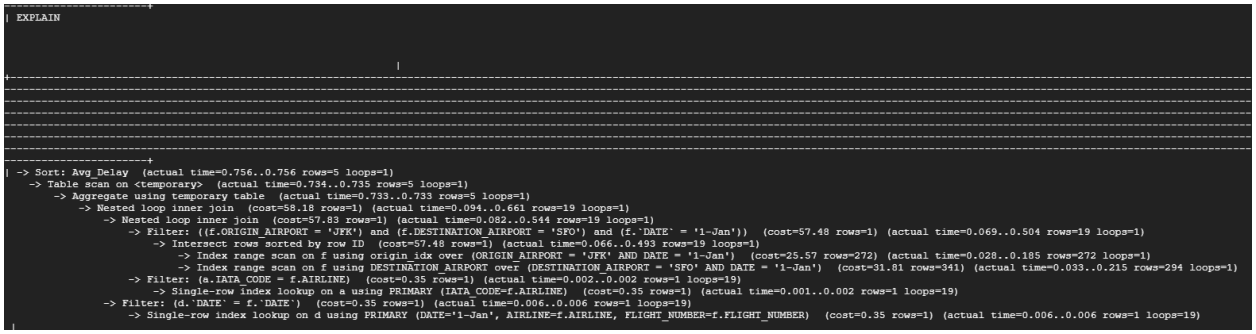
Explain Analyze without Indexing:



The original query is found to have a cost of **0.35** with a timing of **0.008s.**

- **Since the query calculates the average departure delay, an index on this column could potentially improve the speed of the aggregation**

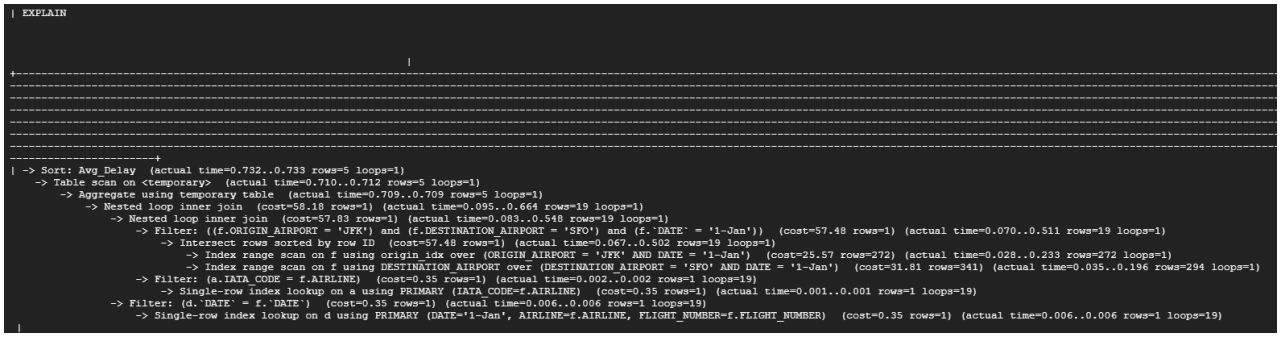
```
CREATE INDEX idx_delays_departure_delay ON delays
(DEPARTURE_DELAY);
```



The indexed query on flights (origin_airport , destination_airport) is found to have an unchanged cost of **0.35** with a timing of **0.006s.** Hence as the cost was unchanged with an insignificant improvement in timing, we continued to try other options.

- **Since the query filters results based on specific origin and destination airports, an index on these columns would help the database quickly find relevant rows.**

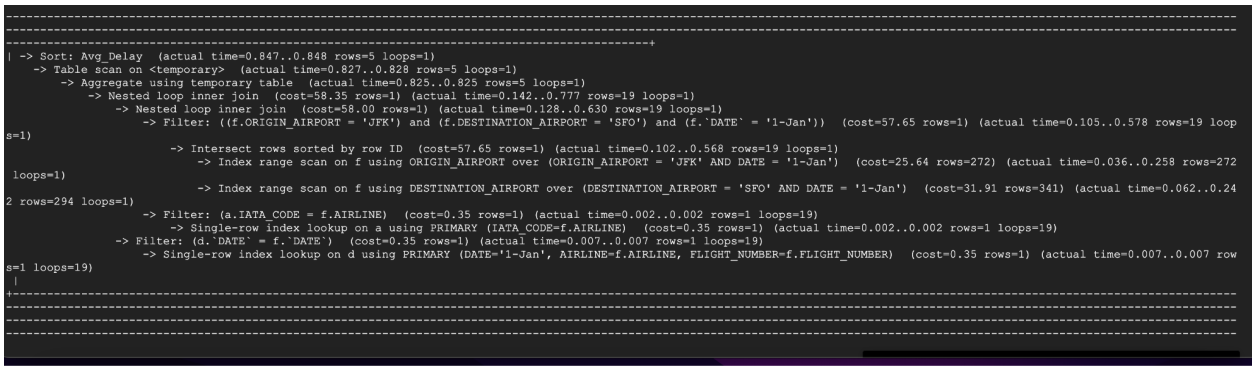
```
CREATE INDEX idx_flights_origin_destination ON flights (ORIGIN_AIRPORT, DESTINATION_AIRPORT);
```



The indexed query on flights (origin_airport , destination_airport) is found to have an unchanged cost of **0.35** with a timing of **0.006s.** Hence as the cost was unchanged with an insignificant improvement in timing, we continued to try other options.

- As there are many dates first we tried to create an index on the date attribute and compare it to the original query:

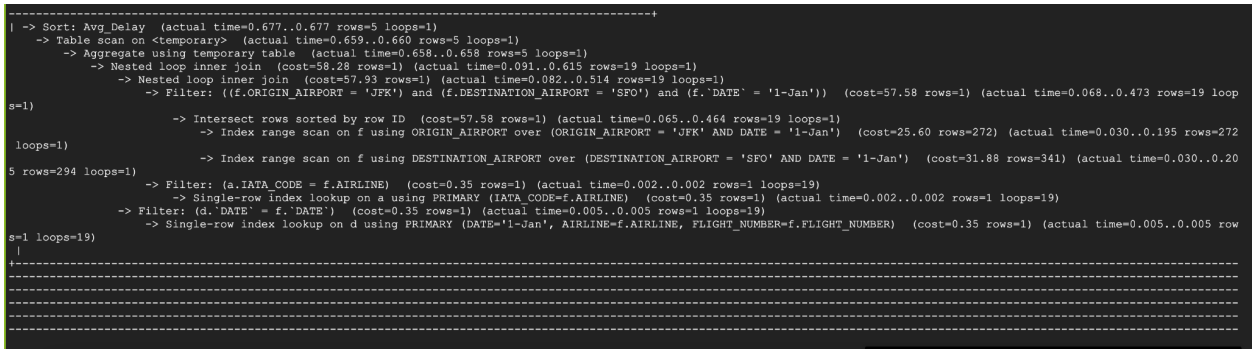
```
create index date_idx on flights(DATE(5));
```



The indexed query on flight Dates is found to have an unchanged cost of **0.35** with a timing of **0.007s.** Hence as the cost was unchanged with an insignificant improvement in timing, this is because Date is already indexed due to it being the primary key.

- We tried to create an index on the Origin Airport attribute.

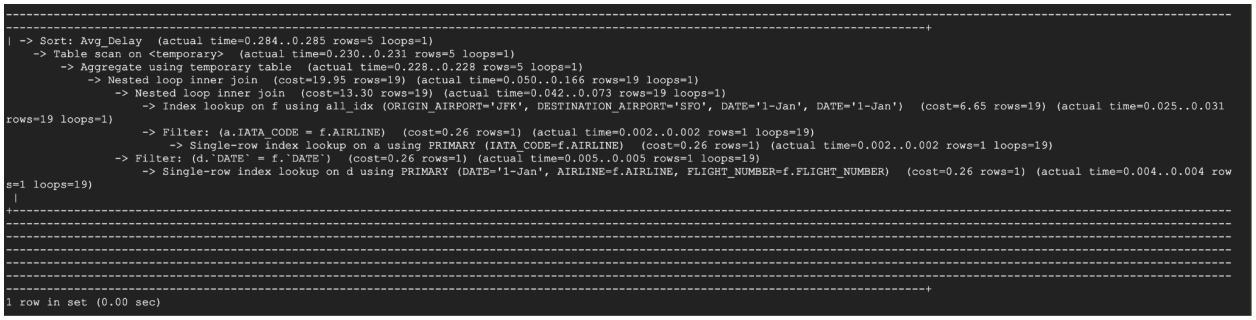
```
create index origin_idx on flights(ORIGIN_AIRPORT(3));
```



The indexed query on Origin Airports was also found to have an unchanged cost of **0.35** with a timing of **0.005s.** Thus the cost is unchanged with an insignificant improvement in timing, we try another option.

- This time we combined both the above attributes, together with destination, to see if that would optimize performance.

```
create index all_idx on flights(ORIGIN_AIRPORT(3),
DESTINATION_AIRPORT(3), DATE(5));
```



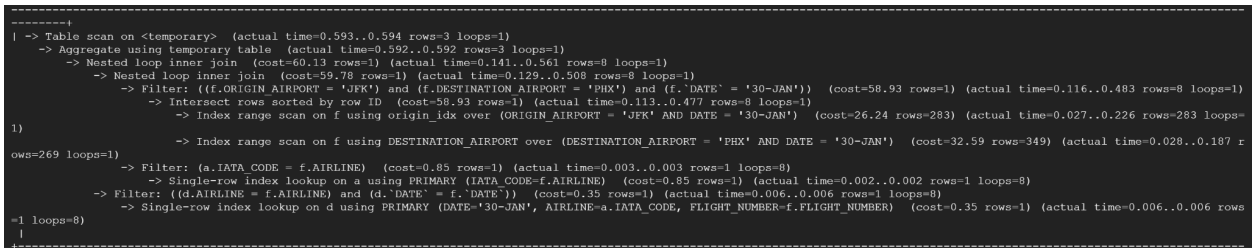
```
-> Sort: Avg_Delay  (actual time=0.284..0.285 rows=5 loops=1)
    -> Table scan on <temporary>  (actual time=0.230..0.231 rows=5 loops=1)
        -> Aggregate using temporary table  (actual time=0.228..0.228 rows=5 loops=1)
            -> Nested loop inner join  (cost=19.95 rows=19) (actual time=0.050..0.166 rows=19 loops=1)
                -> Nested loop inner join  (cost=13.30 rows=19) (actual time=0.042..0.073 rows=19 loops=1)
                    -> Index lookup on f using all_idx (ORIGIN_AIRPORT='JFK', DESTINATION_AIRPORT='SFO', DATE='1-Jan', DATE='1-Jan')  (cost=6.65 rows=19) (actual time=0.025..0.031
rows=19 loops=1)
                    -> Filter: (a.IATA_CODE = f.AIRLINE)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=19)
                        -> Single-row index lookup on a using PRIMARY (IATA_CODE=f.AIRLINE)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=19)
                -> Filter: (d.`DATE` = f.`DATE`)  (cost=0.26 rows=1) (actual time=0.005..0.005 rows=1 loops=19)
                    -> Single-row index lookup on d using PRIMARY (DATE='1-Jan', AIRLINE=f.AIRLINE, FLIGHT_NUMBER=f.FLIGHT_NUMBER)  (cost=0.26 rows=1) (actual time=0.004..0.004 row
s=1 loops=19)

1 row in set (0.00 sec)
```

The indexed query on the Origin and Destination airports and the Date was found to also have an improved cost of **0.26** with a timing of **0.004s.** Thus the cost reduced significantly with an improvement in timing, as it's faster to pinpoint exact flights. We choose this indexing for this query due to significant improvement in the cost and time.

Query #3:

```
SELECT a.AIRLINE, AVG(d.ARRIVAL_DELAY + d.DEPARTURE_DELAY) AS
Avg_Delay
FROM delays d
JOIN flights f ON d.DATE = f.DATE AND d.AIRLINE = f.AIRLINE AND
d.FLIGHT_NUMBER = f.FLIGHT_NUMBER
JOIN airlines a ON a.IATA_CODE = f.AIRLINE
WHERE f.ORIGIN_AIRPORT = 'JFK'
AND f.DESTINATION_AIRPORT = 'PHX'
AND f.DATE = '30-JAN'
GROUP BY a.AIRLINE;
```

EXPLAIN ANALYZE without indexing:

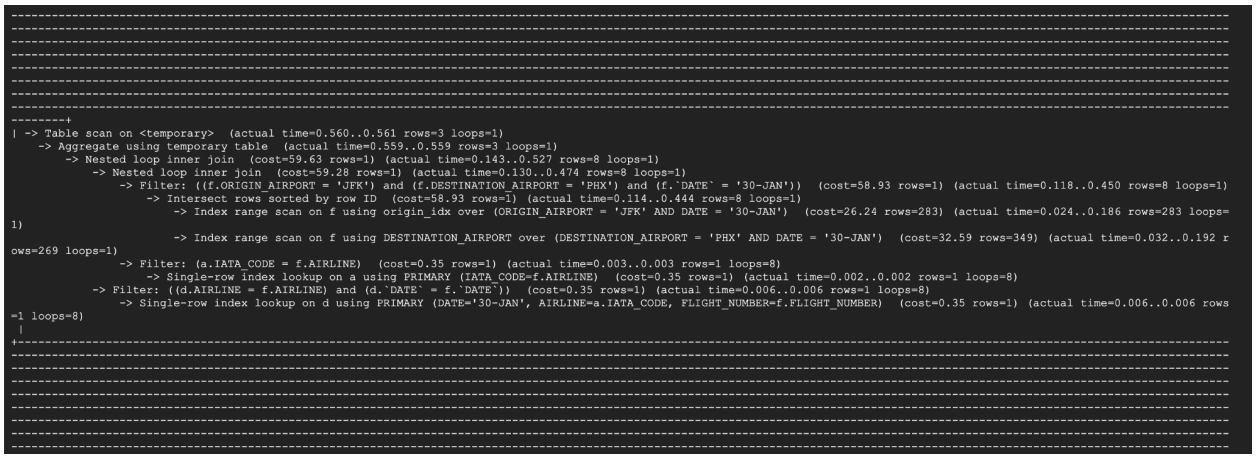


The original query is found to have a cost of **0.35** with a timing of **0.006s.**

With indexing:

- For this query we tried to create an index on the Arrival Delays and compare it to the original query:

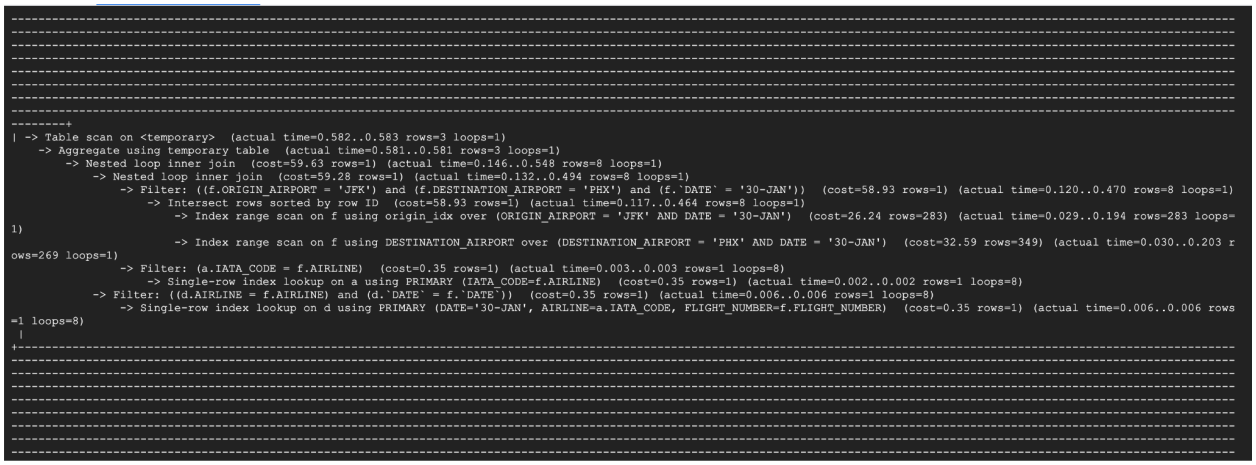
```
create index average_idx on delays(ARRIVAL_DELAY);
```



The indexed query on Arrival Delays was found to have an unchanged cost of **0.35** with a timing of **0.006s.** Thus as the cost is unchanged with an insignificant change in timing, we try another option.

- We tried to create an index on the Departure Delays attribute.

```
create index dept_idx on delays(DEPARTURE_DELAY);
```



The indexed query on Departure Delays was found to have an unchanged cost of **0.35** with a timing of **0.006s.** Thus as the cost is unchanged with no change in timing, we try another option.

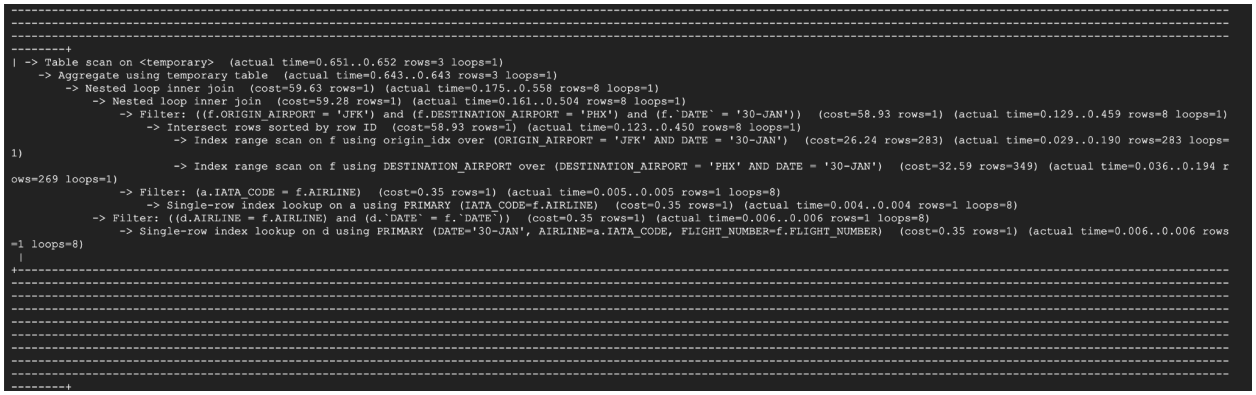
- We tried to create an index using both the Departure and Arrival Delays attribute.

```
create index avg_idx on delays(DEPARTURE_DELAY, ARRIVAL_DELAY);
```

The indexed query using both Departure and Arrival Delays was also found to have an unchanged cost of **0.35** with a timing of **0.006s.** Thus as the cost is unchanged with no change in timing, we try another option.

- Hence we tried to create an index combining both the Departure and Arrival Delays attribute.

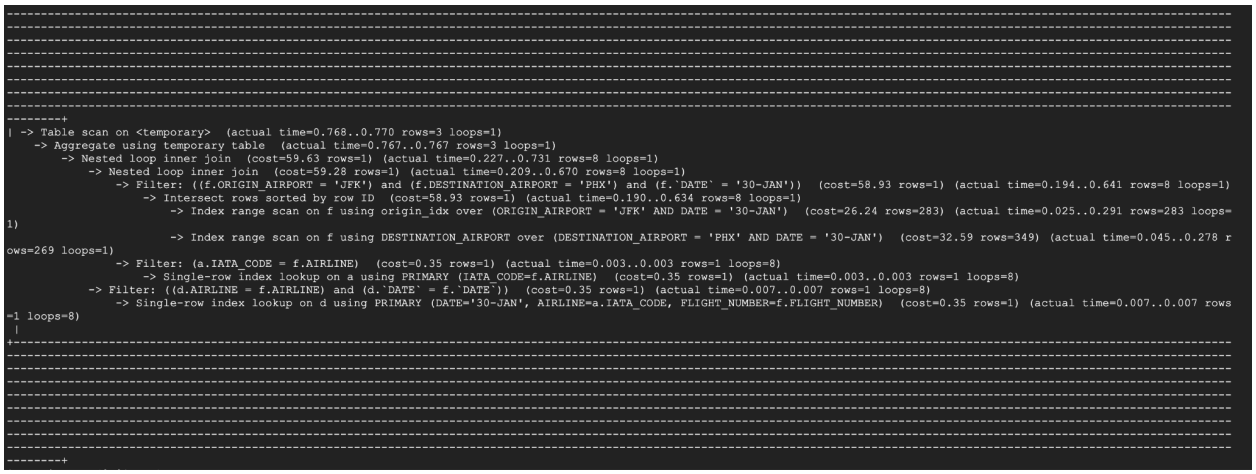
```
create index avg_idx on delays(DEPARTURE_DELAY+ ARRIVAL_DELAY);
```



The indexed query combining both Departure and Arrival Delays was also found to have an unchanged cost of **0.35** with a timing of **0.006s.** Thus as the cost is unchanged with no change in timing, we try another option.

- We tried to create an index using both the Departure and Arrival Delays attribute as well as the Flight Dates.

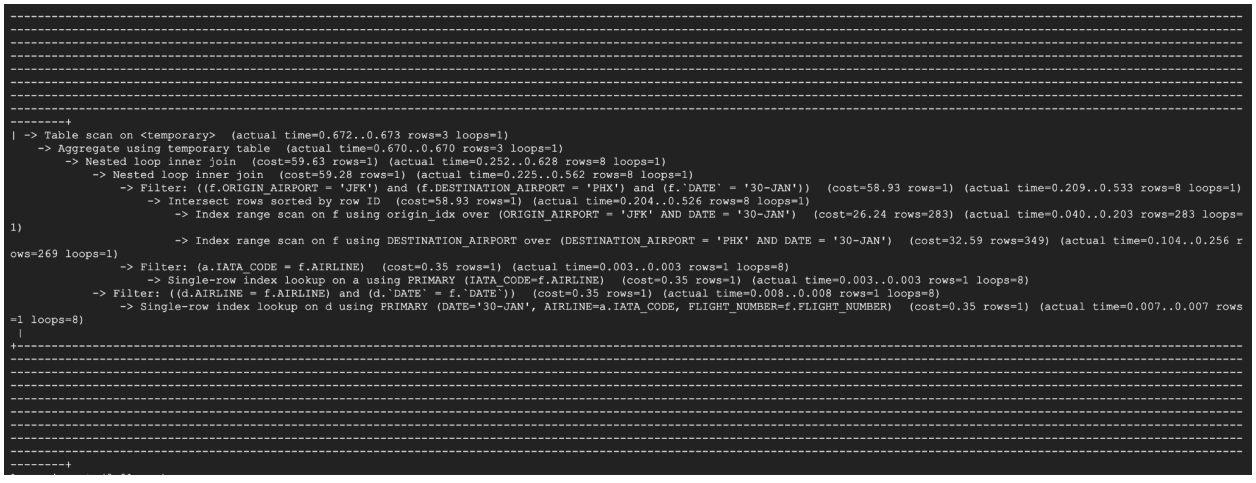
```
CREATE INDEX avg_idx ON delays(DATE(5), DEPARTURE_DELAY,
ARRIVAL_DELAY);
```



The indexed query combining both Departure and Arrival Delays was also found to have an unchanged cost of **0.35** with an insignificantly changed time of **0.007s. Date is a primary key which is already Indexed.** Thus as the cost is unchanged with almost no change in timing, we try another option.

- We tried to create an index using the average of the Departure and Arrival Delays

```
create index avg_idx on delays(
((DEPARTURE_DELAY+ ARRIVAL_DELAY)/2));
```



The indexed query averaging the Departure and Arrival Delays was also found to have an unchanged cost of **0.35** with an insignificantly changed time of **0.007s.** Thus as the cost is unchanged with almost no change in timing, we try another option.

- We tried to **combine the indexes** using the Date, Departure and Arrival Delays, as well as the other index using Origin and Destination airports.

```
CREATE INDEX avg_idx ON delays(DATE(5), DEPARTURE_DELAY,
ARRIVAL_DELAY);
CREATE INDEX flg_idx ON flights(ORIGIN_AIRPORT(3),
DESTINATION_AIRPORT(3));
```

```
+------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                        |
+------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=0.183..0.184 rows=3 loops=1)
    -> Aggregate using temporary table  (actual time=0.181..0.181 rows=3 loops=1)
        -> Nested loop inner join  (cost=8.04 rows=8) (actual time=0.087..0.143 rows=8 loops=1)
            -> Nested loop inner join  (cost=5.24 rows=8) (actual time=0.071..0.088 rows=8 loops=1)
                -> Covering index lookup on f using flg_idx (ORIGIN_AIRPORT='JFK', DESTINATION_AIRPORT='PHX', DATE='30-JAN')  (cost=2.44 rows=8) (actual time=0.029..0.034 rows=8 lo
ops=1)
                -> Filter: (a.IATA_CODE = f.AIRLINE)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=8)
                    -> Single-row index lookup on a using PRIMARY (IATA_CODE=f.AIRLINE)  (cost=0.26 rows=1) (actual time=0.002..0.003 rows=1 loops=8)
            -> Filter: ((d.AIRLINE = f.AIRLINE) and (d.`DATE` = f.`DATE`))  (cost=0.26 rows=1) (actual time=0.007..0.007 rows=1 loops=8)
                -> Single-row index lookup on d using PRIMARY (DATE='30-JAN', AIRLINE=a.IATA_CODE, FLIGHT_NUMBER=f.FLIGHT_NUMBER)  (cost=0.26 rows=1) (actual time=0.006..0.006 rows
=1 loops=8)
 |
+------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

The combined query finally resulted in a reduced cost of **0.26** with an insignificantly changed time of **0.006s.** Thus as the cost is reduced with slight change in timing, we treat this as the optimized query and the choice for selecting this as our index on this query.