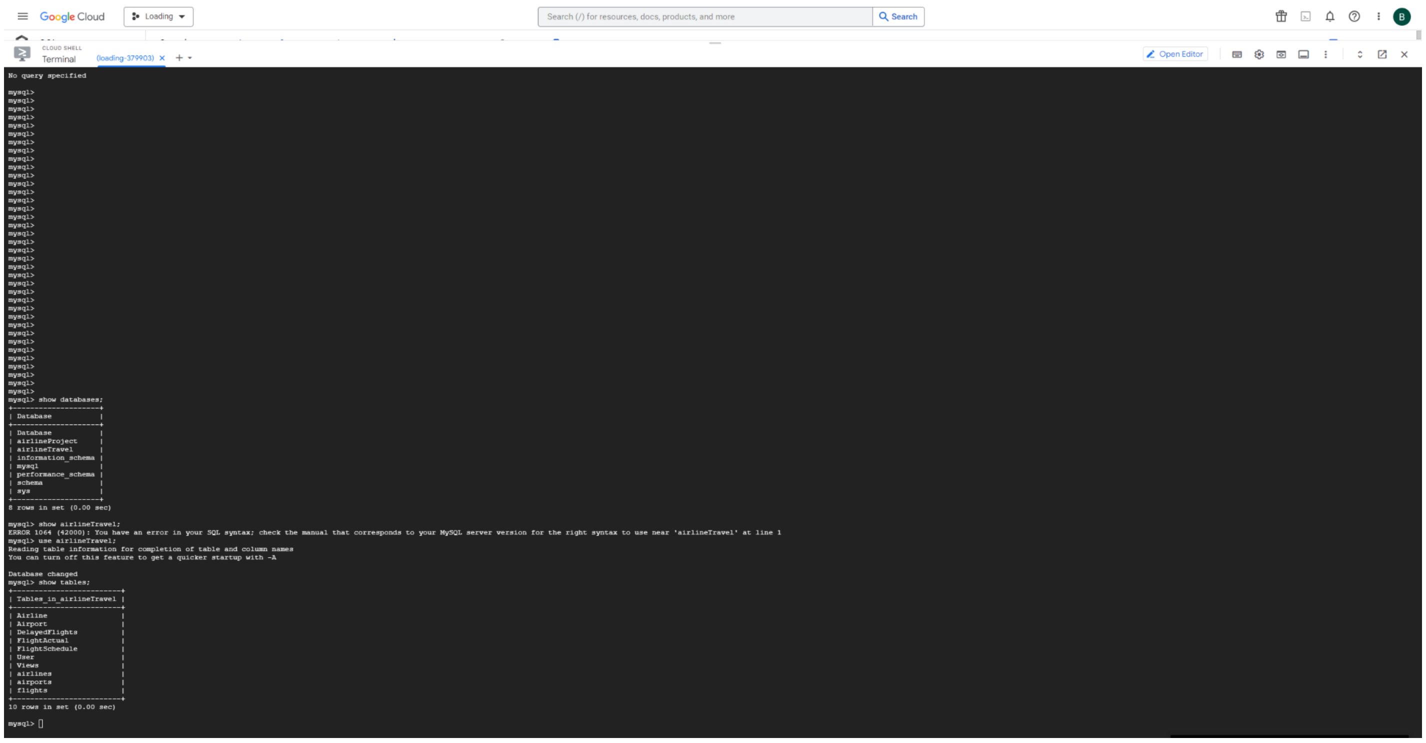
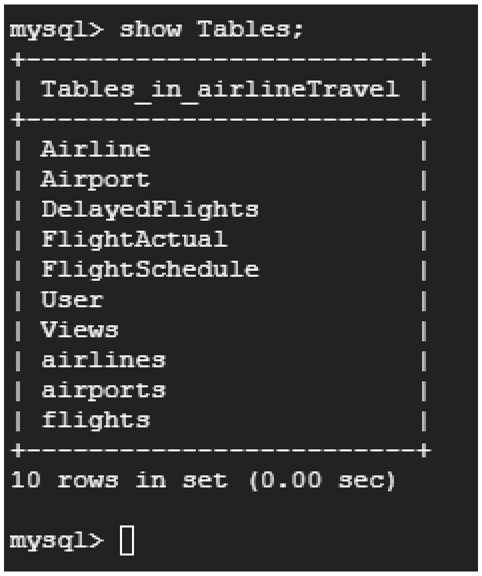
Stage 3: Database Implementation and Indexing

a. MySQL @ GCP



```
No query specified

mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| Database           |
| airlineProject     |
| airlineTravel      |
| information_schema |
| mysql              |
| performance_schema |
| schema             |
| sys                |
+--------------------+
8 rows in set (0.00 sec)

mysql> show airlineTravel;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'airlineTravel' at line 1
mysql> use airlineTravel;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-------------------------+
| Tables_in_airlineTravel |
+-------------------------+
| Airline                 |
| Airport                 |
| DelayedFlights          |
| FlightActual            |
| FlightSchedule          |
| User                    |
| Views                   |
| airlines                |
| airports                |
| flights                 |
+-------------------------+
10 rows in set (0.00 sec)

mysql> ▯
```



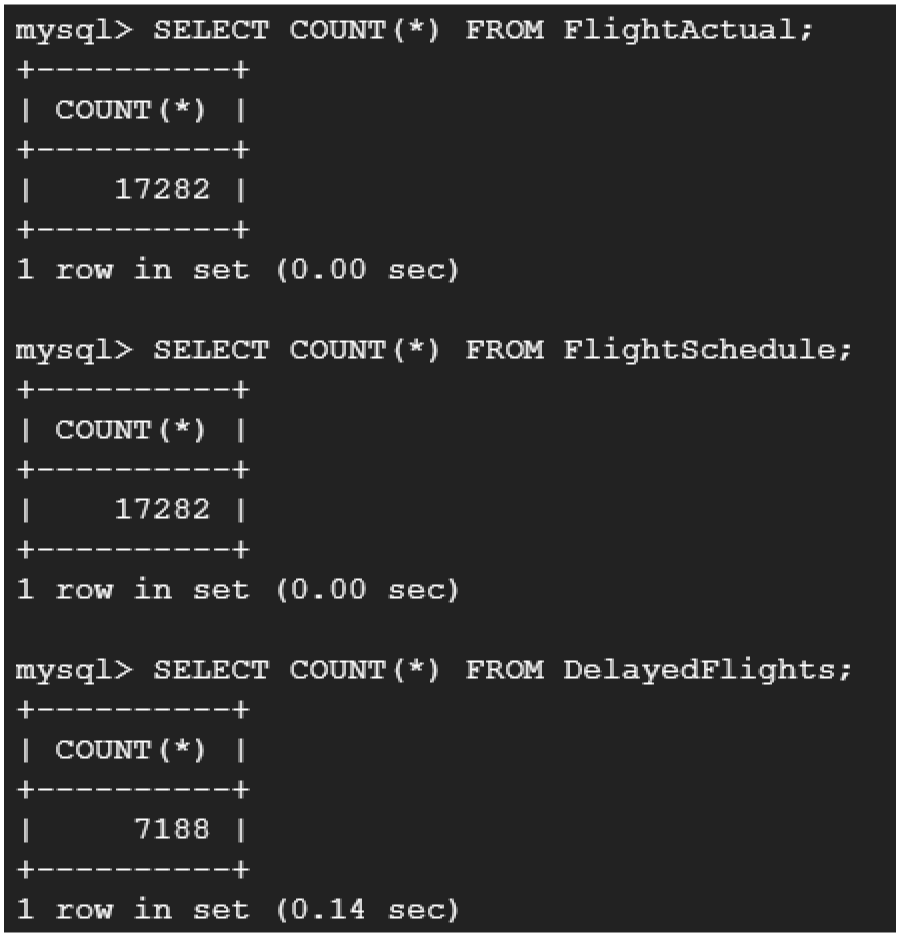
```
mysql> show Tables;
+-------------------------+
| Tables_in_airlineTravel |
+-------------------------+
| Airline                 |
| Airport                 |
| DelayedFlights          |
| FlightActual            |
| FlightSchedule          |
| User                    |
| Views                   |
| airlines                |
| airports                |
| flights                 |
+-------------------------+
10 rows in set (0.00 sec)

mysql> ▯
```

*airlines, airports, flights table imported directly from 2015 Flight Delays and Cancellation dataset

b. Data Definition Language (DDL) Commands

```
CREATE TABLE User(
   userId INT NOT NULL,
   username VARCHAR(250),
   password VARCHAR(250),
   isActive bit,
   phone REAL,
   userType VARCHAR(10),
   PRIMARY KEY (userId)
);

CREATE TABLE Airline(
   airlineCode VARCHAR(2) NOT NULL,
   name VARCHAR(100),
   PRIMARY KEY (airlineCode)
);

CREATE TABLE Airport(
   airportCode VARCHAR(3) NOT NULL,
   name VARCHAR(100),
   city VARCHAR(100),
   state VARCHAR(100),
   latitude REAL,
   longitude REAL,
   PRIMARY KEY (airportCode)
);

CREATE TABLE FlightSchedule(
   flightNumber INT NOT NULL,
   date VARCHAR(10) NOT NULL,
   tailNumber VARCHAR(6) NOT NULL,
   airlineCode VARCHAR(2),
   airportOriginCode VARCHAR(3),
   airportDestinationCode VARCHAR(3),
   scheduledDeparture INT NOT NULL,
   scheduledArrival INT,
   PRIMARY KEY (flightNumber, date, tailNumber, scheduledDeparture),
   FOREIGN KEY (airlineCode) REFERENCES Airline(airlineCode),
   FOREIGN KEY (airportOriginCode) REFERENCES Airport(airportCode),
   FOREIGN KEY (airportDestinationCode) REFERENCES Airport(airportCode)
);

CREATE TABLE FlightActual(
   flightNumber INT NOT NULL,
   date VARCHAR(10) NOT NULL,
   tailNumber VARCHAR(6) NOT NULL,
   arrivalTime INT NOT NULL,
   departureTime INT NOT NULL,
   airlineCode VARCHAR(2),
   arrivalDelayTime INT,
   PRIMARY KEY (flightNumber, date, tailNumber, departureTime, arrivalTime),
   FOREIGN KEY (airlineCode) REFERENCES Airline(airlineCode)
```

```
);

CREATE TABLE DelayedFlights(
   flightNumber INT NOT NULL,
   airlineCode VARCHAR(2) NOT NULL,
   date VARCHAR(10) NOT NULL,
   tailNumber VARCHAR(6) NOT NULL,
   scheduledArrival INT NOT NULL,
   arrivalDelayTime VARCHAR(10),
   airSystemDelay VARCHAR(10),
   securityDelay VARCHAR(10),
   airlineDelay VARCHAR(10),
   lateAircraftDelay VARCHAR(10),
   weatherDelay VARCHAR(10),
   PRIMARY KEY (flightNumber, airlineCode, date, tailNumber, scheduledArrival),
   FOREIGN KEY (airlineCode) REFERENCES Airline(airlineCode)
);


CREATE TABLE Views(
   userId INT NOT NULL,
   flightNumber INT,
   FOREIGN KEY (userId) REFERENCES User(userId),
   FOREIGN KEY (flightNumber) REFERENCES FlightSchedule(flightNumber)
);
```


```
mysql> SELECT COUNT(*) FROM FlightActual;
+----------+
| COUNT(*) |
+----------+
|    17282 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM FlightSchedule;
+----------+
| COUNT(*) |
+----------+
|    17282 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM DelayedFlights;
+----------+
| COUNT(*) |
+----------+
|     7188 |
+----------+
1 row in set (0.14 sec)
```

****WE INPUTTED OUR RAW CSV DATA INTO GCP****

INSERT INTO Airline (airlineCode, name)
SELECT IATA_CODE, AIRLINE FROM airlines;

INSERT INTO Airport(airportCode, name, city, state, latitude, longitude)
SELECT IATA_CODE, AIRPORT, CITY, STATE, LATITUDE, LONGITUDE FROM airports;

INSERT INTO FlightSchedule (flightNumber, date, tailNumber, airlineCode, airportOriginCode, airportDestinationCode, scheduledDeparture, scheduledArrival)
SELECT FLIGHT_NUMBER, concat(YEAR, '-', MONTH, '-', DAY), TAIL_NUMBER, AIRLINE, ORIGIN_AIRPORT, DESTINATION_AIRPORT, SCHEDULED_DEPARTURE, SCHEDULED_ARRIVAL FROM flights;

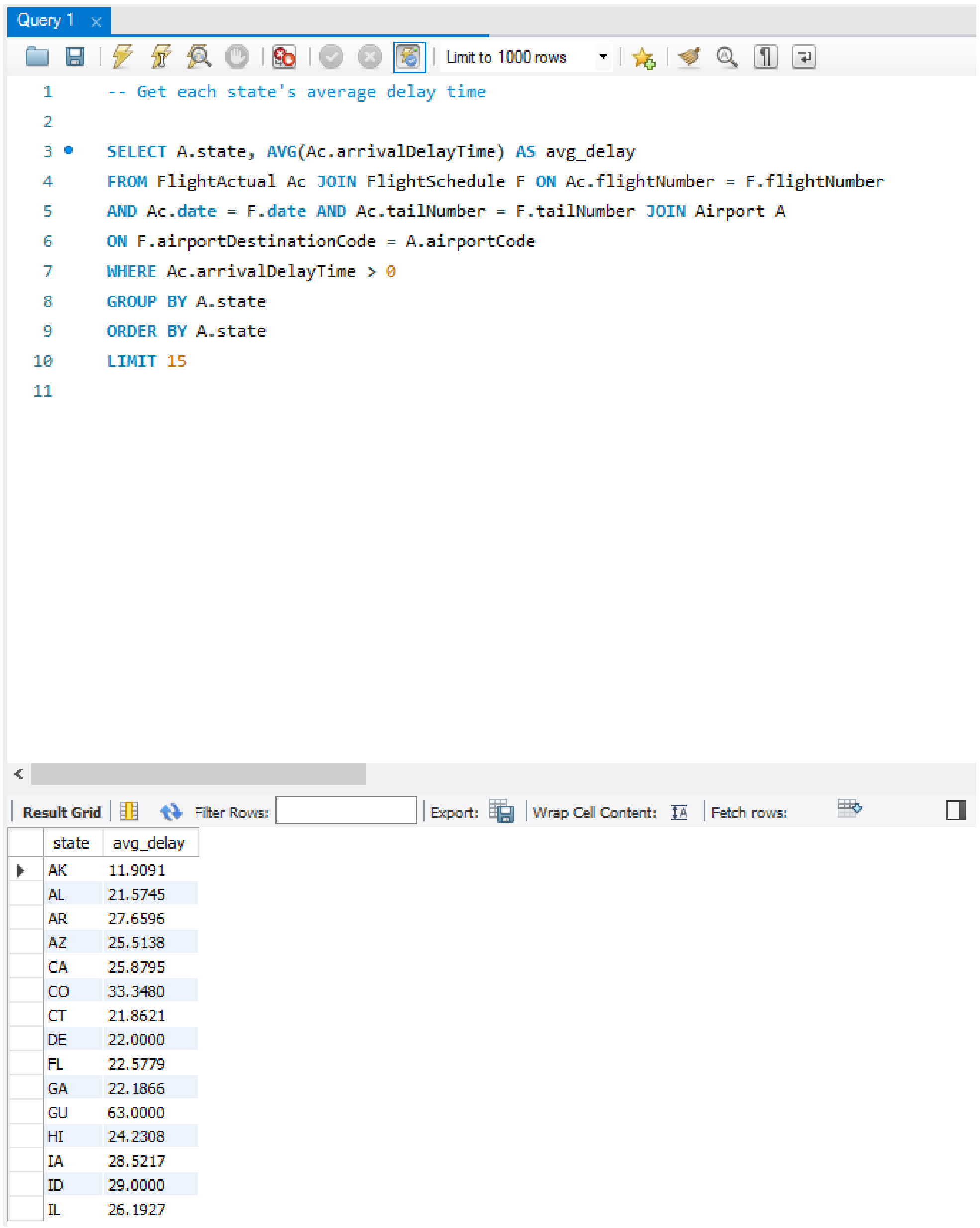
INSERT INTO DelayedFlights (flightNumber, airlineCode, date, tailNumber, arrivalTime, arrivalDelayTime, airSystemDelay, securityDelay, airlineDelay, lateAircraftDelay, weatherDelay)
SELECT FLIGHT_NUMBER, AIRLINE, concat(YEAR, '-', MONTH, '-', DAY), TAIL_NUMBER, SCHEDULED_ARRIVAL, ARRIVAL_DELAY, AIR_SYSTEM_DELAY, SECURITY_DELAY, AIRLINE_DELAY, LATE_AIRCRAFT_DELAY, WEATHER_DELAY

```
FROM flights WHERE ARRIVAL_DELAY > 0;

INSERT INTO FlightActual (flightNumber, date, tailNumber, departureTime, arrivalTime,
airlineCode, arrivalDelayTime)
SELECT FLIGHT_NUMBER, concat(YEAR, '-', MONTH, '-', DAY), TAIL_NUMBER,
DEPARTURE_TIME, ARRIVAL_TIME, AIRLINE, ARRIVAL_DELAY
FROM flights
```

c. Advanced Queries

1) Get each state's average delay time

```
SELECT A.state, AVG(Ac.arrivalDelayTime) AS avg_delay
FROM FlightActual Ac JOIN FlightSchedule F ON Ac.flightNumber = F.flightNumber AND
Ac.date = F.date AND Ac.tailNumber = F.tailNumber JOIN Airport A ON
F.airportDestinationCode = A.airportCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.state
ORDER BY A.state
```

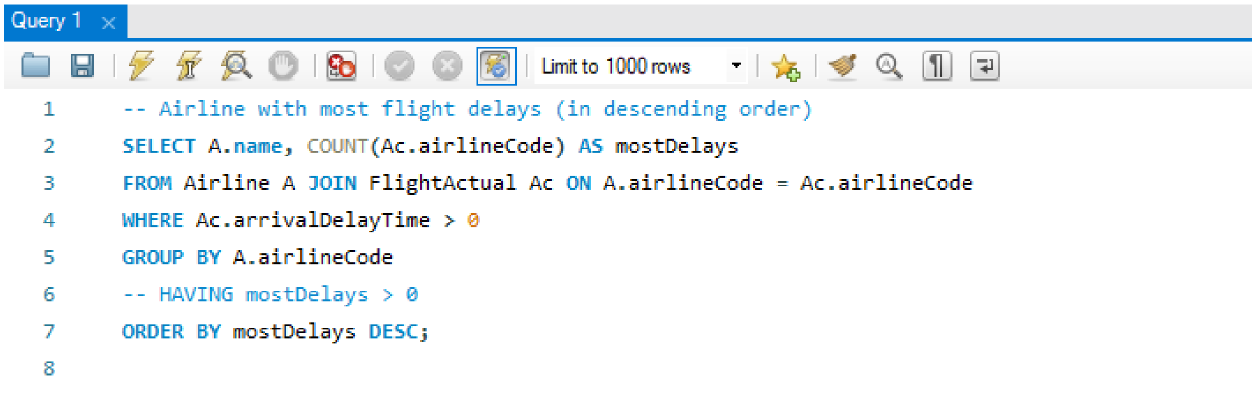


```
-- Get each state's average delay time

SELECT A.state, AVG(Ac.arrivalDelayTime) AS avg_delay
FROM FlightActual Ac JOIN FlightSchedule F ON Ac.flightNumber = F.flightNumber
AND Ac.date = F.date AND Ac.tailNumber = F.tailNumber JOIN Airport A
ON F.airportDestinationCode = A.airportCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.state
ORDER BY A.state
LIMIT 15
```

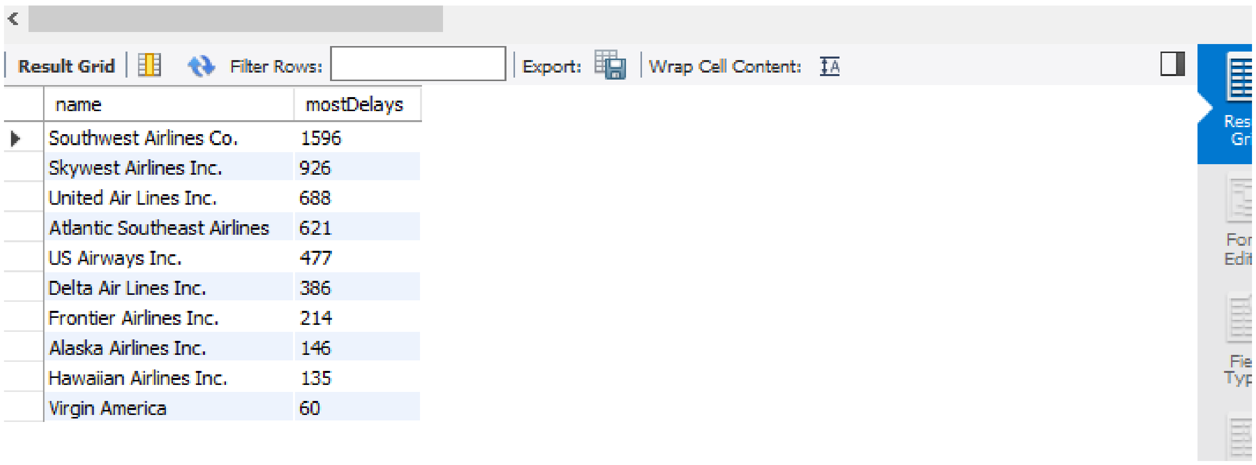
state	avg_delay
AK	11.9091
AL	21.5745
AR	27.6596
AZ	25.5138
CA	25.8795
CO	33.3480
CT	21.8621
DE	22.0000
FL	22.5779
GA	22.1866
GU	63.0000
HI	24.2308
IA	28.5217
ID	29.0000
IL	26.1927

2) Airline with most flight delays (in descending order)

SELECT A.name, COUNT(Ac.airlineCode) AS mostDelays
FROM Airline A JOIN FlightActual Ac ON A.airlineCode = Ac.airlineCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.airlineCode
ORDER BY mostDelays DESC;



```sql
-- Airline with most flight delays (in descending order)
SELECT A.name, COUNT(Ac.airlineCode) AS mostDelays
FROM Airline A JOIN FlightActual Ac ON A.airlineCode = Ac.airlineCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.airlineCode
-- HAVING mostDelays > 0
ORDER BY mostDelays DESC;
```

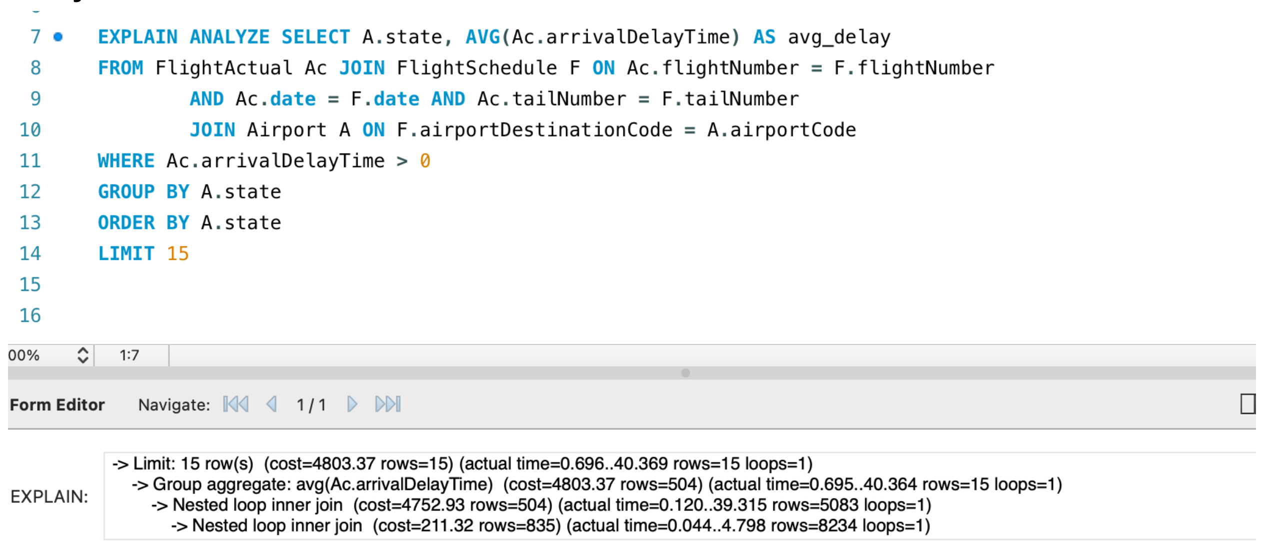


name	mostDelays
Southwest Airlines Co.	1596
Skywest Airlines Inc.	926
United Air Lines Inc.	688
Atlantic Southeast Airlines	621
US Airways Inc.	477
Delta Air Lines Inc.	386
Frontier Airlines Inc.	214
Alaska Airlines Inc.	146
Hawaiian Airlines Inc.	135
Virgin America	60

****OUTPUT has less than 15 rows**

d. <u>Indexing Analysis</u>

Query 1

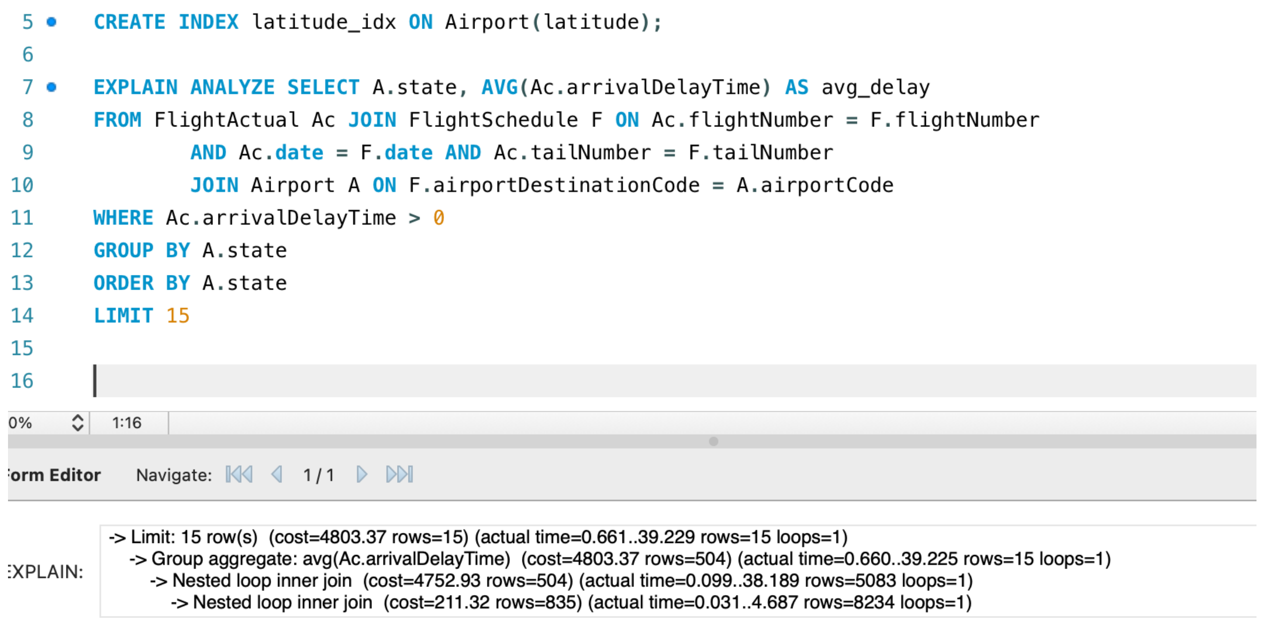


```sql
EXPLAIN ANALYZE SELECT A.state, AVG(Ac.arrivalDelayTime) AS avg_delay
FROM FlightActual Ac JOIN FlightSchedule F ON Ac.flightNumber = F.flightNumber
        AND Ac.date = F.date AND Ac.tailNumber = F.tailNumber
        JOIN Airport A ON F.airportDestinationCode = A.airportCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.state
ORDER BY A.state
LIMIT 15
```

```
-> Limit: 15 row(s)  (cost=4803.37 rows=15) (actual time=0.696..40.369 rows=15 loops=1)
    -> Group aggregate: avg(Ac.arrivalDelayTime)  (cost=4803.37 rows=504) (actual time=0.695..40.364 rows=15 loops=1)
        -> Nested loop inner join  (cost=4752.93 rows=504) (actual time=0.120..39.315 rows=5083 loops=1)
            -> Nested loop inner join  (cost=211.32 rows=835) (actual time=0.044..4.798 rows=8234 loops=1)
```

These are the results from our first advanced query without any index. The aggregate cost is 4803 and the time is around 0.696 to 40.369. We consider these results as the foundation for our index performance analysis.

1. **Created index on *latitude* in *Airport*.**

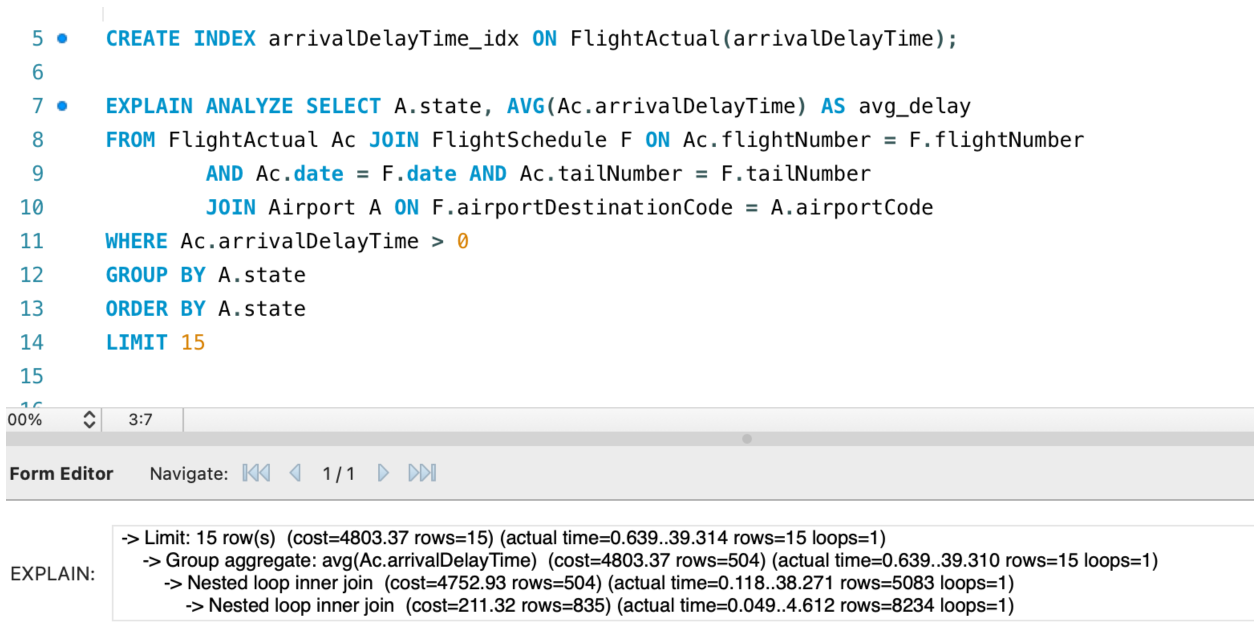


```sql
CREATE INDEX latitude_idx ON Airport(latitude);

EXPLAIN ANALYZE SELECT A.state, AVG(Ac.arrivalDelayTime) AS avg_delay
FROM FlightActual Ac JOIN FlightSchedule F ON Ac.flightNumber = F.flightNumber
        AND Ac.date = F.date AND Ac.tailNumber = F.tailNumber
        JOIN Airport A ON F.airportDestinationCode = A.airportCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.state
ORDER BY A.state
LIMIT 15
```

```
-> Limit: 15 row(s)  (cost=4803.37 rows=15) (actual time=0.661..39.229 rows=15 loops=1)
    -> Group aggregate: avg(Ac.arrivalDelayTime)  (cost=4803.37 rows=504) (actual time=0.660..39.225 rows=15 loops=1)
        -> Nested loop inner join  (cost=4752.93 rows=504) (actual time=0.099..38.189 rows=5083 loops=1)
            -> Nested loop inner join  (cost=211.32 rows=835) (actual time=0.031..4.687 rows=8234 loops=1)
```

First of all we indexed on latitude because latitude varies from airports to airports and we think there are many different airports in our dataset. However, after indexing, the cost remains at 4803.37 and the time decreases slightly from 0.696-40.369 to 0.661-39.229. We think the reason for these slight changes is the uneven distribution of latitude in Airport data. We find that most of the data are concentrated in the same small interval, so indexing on latitude is similar to using the original index.

2. **Created index on *arrivalDelayTime* in *FlightActual*.**

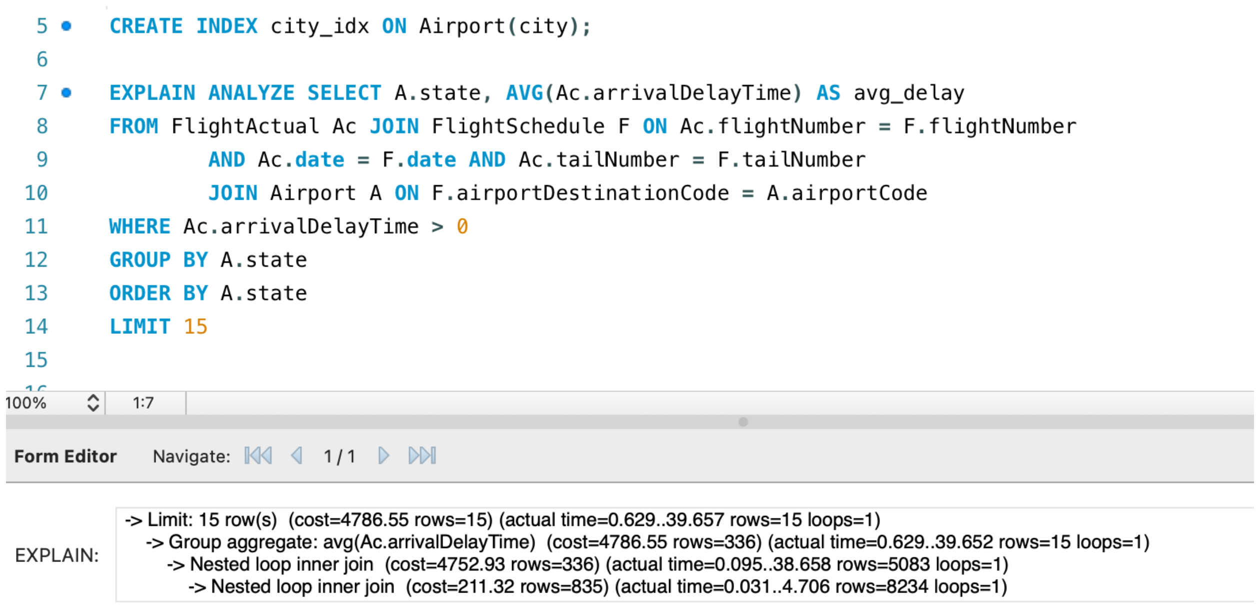


```sql
CREATE INDEX arrivalDelayTime_idx ON FlightActual(arrivalDelayTime);

EXPLAIN ANALYZE SELECT A.state, AVG(Ac.arrivalDelayTime) AS avg_delay
FROM FlightActual Ac JOIN FlightSchedule F ON Ac.flightNumber = F.flightNumber
        AND Ac.date = F.date AND Ac.tailNumber = F.tailNumber
        JOIN Airport A ON F.airportDestinationCode = A.airportCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.state
ORDER BY A.state
LIMIT 15
```

```
-> Limit: 15 row(s)  (cost=4803.37 rows=15) (actual time=0.639..39.314 rows=15 loops=1)
    -> Group aggregate: avg(Ac.arrivalDelayTime)  (cost=4803.37 rows=504) (actual time=0.639..39.310 rows=15 loops=1)
        -> Nested loop inner join  (cost=4752.93 rows=504) (actual time=0.118..38.271 rows=5083 loops=1)
            -> Nested loop inner join  (cost=211.32 rows=835) (actual time=0.049..4.612 rows=8234 loops=1)
```

The second index that we apply for our query is arrivalDelayTime in FlightActual, because we think every flight has a different arrival delay time. However, we still get the same cost 4803.37 and very small changes in time, from 0.696-40.369 to 0.639-39.314. We think the reason for the slight changes is still uneven distribution of data. After we check our dataset, we find out that most of our data in arrivalDelayTime concentrate around 0, and only a small amount of data stay far away from 0. As a result, indexing on arrivalDelayTime is similar to our default index.

3. **Created index on *city* in *Airport*.**



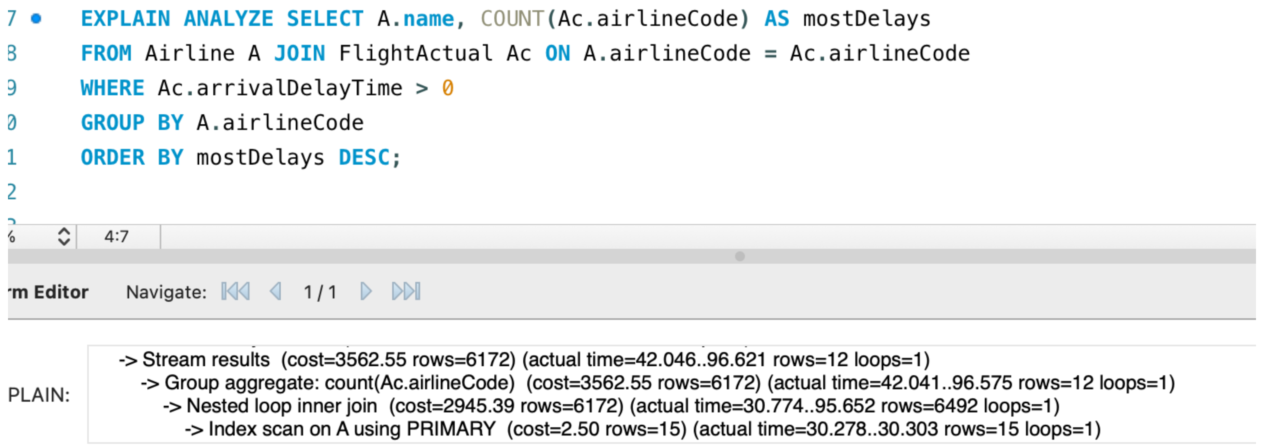
```sql
CREATE INDEX city_idx ON Airport(city);

EXPLAIN ANALYZE SELECT A.state, AVG(Ac.arrivalDelayTime) AS avg_delay
FROM FlightActual Ac JOIN FlightSchedule F ON Ac.flightNumber = F.flightNumber
        AND Ac.date = F.date AND Ac.tailNumber = F.tailNumber
        JOIN Airport A ON F.airportDestinationCode = A.airportCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.state
ORDER BY A.state
LIMIT 15
```

```
-> Limit: 15 row(s)  (cost=4786.55 rows=15) (actual time=0.629..39.657 rows=15 loops=1)
  -> Group aggregate: avg(Ac.arrivalDelayTime)  (cost=4786.55 rows=336) (actual time=0.629..39.652 rows=15 loops=1)
    -> Nested loop inner join  (cost=4752.93 rows=336) (actual time=0.095..38.658 rows=5083 loops=1)
      -> Nested loop inner join  (cost=211.32 rows=835) (actual time=0.031..4.706 rows=8234 loops=1)
```

The third index we try is city in Airport, because we find out that there are 308 different cities in the city column which involves more unique data than other columns, so we think indexing on city can improve overall performance. Fortunately, we improved our cost from 4803.37 to 4786.55, and we also improved time from 0.696-40.369 to 0.629-39.675. After we index on the city, we create pointers to the data which will decrease the cost.

Conclusion:
After indexing on three different variables, we find out that indexing on city results in the best performance, decreasing cost from 4803.37 to 4786.55 and decreasing time from 0.696-40.369 to 0.629-39.675. Consequently, we will choose to index the city.

Query 2

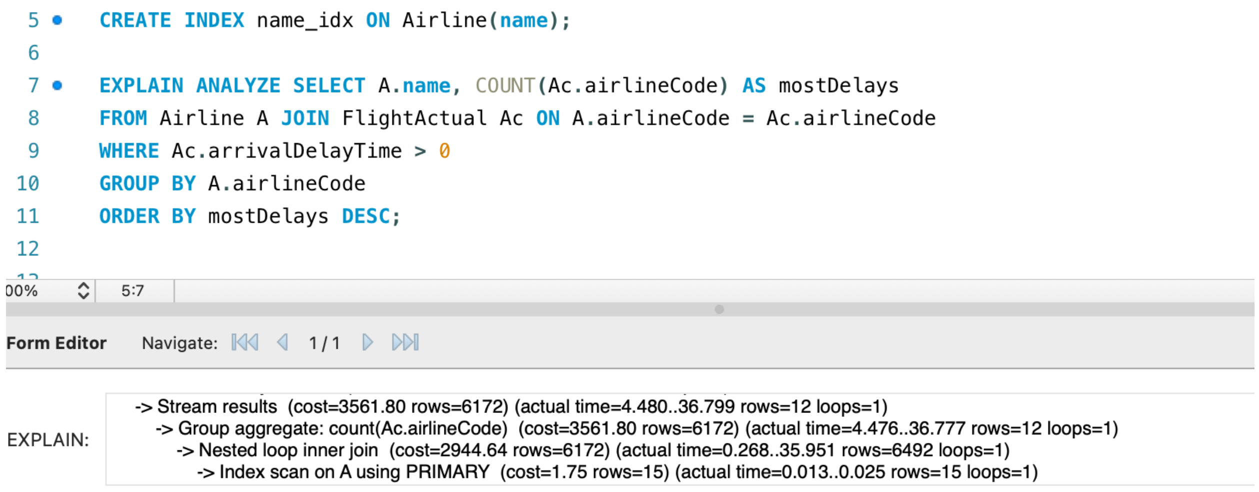


```sql
EXPLAIN ANALYZE SELECT A.name, COUNT(Ac.airlineCode) AS mostDelays
FROM Airline A JOIN FlightActual Ac ON A.airlineCode = Ac.airlineCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.airlineCode
ORDER BY mostDelays DESC;
```

```
-> Stream results  (cost=3562.55 rows=6172) (actual time=42.046..96.621 rows=12 loops=1)
  -> Group aggregate: count(Ac.airlineCode)  (cost=3562.55 rows=6172) (actual time=42.041..96.575 rows=12 loops=1)
    -> Nested loop inner join  (cost=2945.39 rows=6172) (actual time=30.774..95.652 rows=6492 loops=1)
      -> Index scan on A using PRIMARY  (cost=2.50 rows=15) (actual time=30.278..30.303 rows=15 loops=1)
```

These are the results from our second advanced query without any index. The aggregate cost is 3562.55 and the time is around 42.046 to 96.621. We consider these results as the foundation for our index performance analysis.

1. **Created index on *name* in *Airline*.**

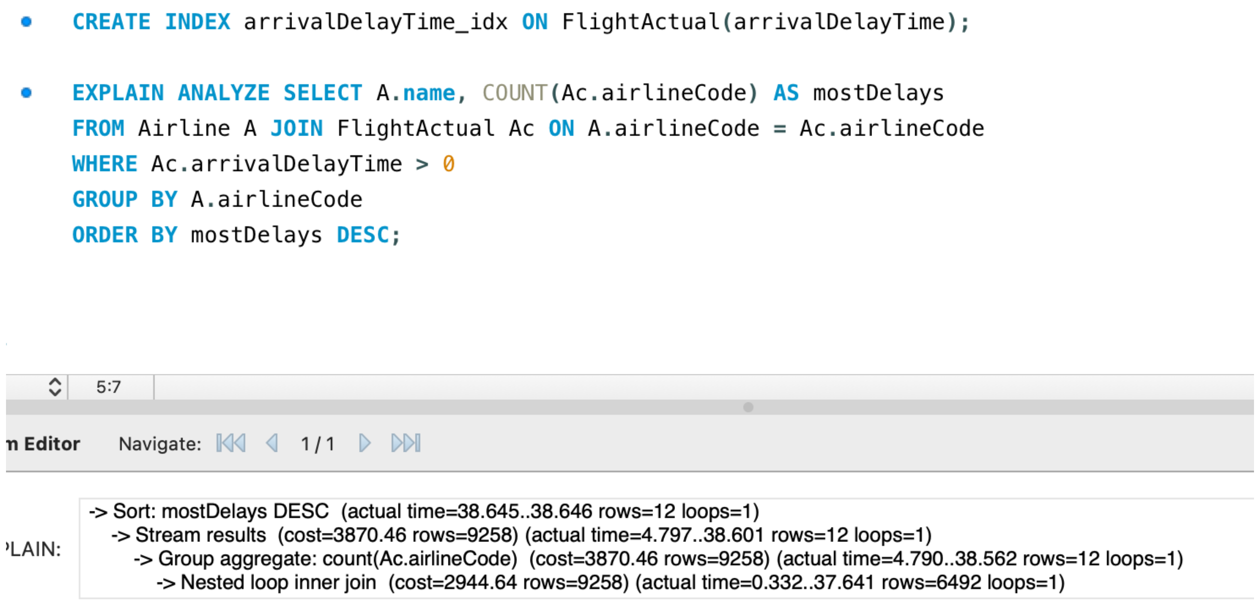


```
CREATE INDEX name_idx ON Airline(name);

EXPLAIN ANALYZE SELECT A.name, COUNT(Ac.airlineCode) AS mostDelays
FROM Airline A JOIN FlightActual Ac ON A.airlineCode = Ac.airlineCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.airlineCode
ORDER BY mostDelays DESC;
```

```
EXPLAIN:
-> Stream results  (cost=3561.80 rows=6172) (actual time=4.480..36.799 rows=12 loops=1)
    -> Group aggregate: count(Ac.airlineCode)  (cost=3561.80 rows=6172) (actual time=4.476..36.777 rows=12 loops=1)
        -> Nested loop inner join  (cost=2944.64 rows=6172) (actual time=0.268..35.951 rows=6492 loops=1)
            -> Index scan on A using PRIMARY  (cost=1.75 rows=15) (actual time=0.013..0.025 rows=15 loops=1)
```

First of all, we choose to index on name in Airline because the airline name varies from flight to flight. The results show that we decrease our cost from 3562.55 to 3561.8 and also decrease time from 42.046-96.621 to 4.48-36.799. Although the cost decreases slightly, we decrease the time a lot. After we checked our data, we found 14 unique airline names in our data. We think 14 is not enough for making huge improvements in performance.

2. **Created index on *arrivalDelayTime* in *FlightActual*.**



```
CREATE INDEX arrivalDelayTime_idx ON FlightActual(arrivalDelayTime);

EXPLAIN ANALYZE SELECT A.name, COUNT(Ac.airlineCode) AS mostDelays
FROM Airline A JOIN FlightActual Ac ON A.airlineCode = Ac.airlineCode
WHERE Ac.arrivalDelayTime > 0
GROUP BY A.airlineCode
ORDER BY mostDelays DESC;
```

```
PLAIN:
-> Sort: mostDelays DESC  (actual time=38.645..38.646 rows=12 loops=1)
    -> Stream results  (cost=3870.46 rows=9258) (actual time=4.797..38.601 rows=12 loops=1)
        -> Group aggregate: count(Ac.airlineCode)  (cost=3870.46 rows=9258) (actual time=4.790..38.562 rows=12 loops=1)
            -> Nested loop inner join  (cost=2944.64 rows=9258) (actual time=0.332..37.641 rows=6492 loops=1)
```

Then, we apply another indexing on arrival delay time in FlightActual. After doing the same indexing for our first query, we think this indexing may not improve our performance. However, the result shows that the cost increases from 3562.55 to 3870.46, but the time decreases from 42.046-96.621 to 38.645-38.646. At first, we thought we made a mistake when we were doing the third indexing. However, we dropped all possible indexing and did the analysis again, we still get the same result. We wonder why this was the case.

3. Fail to apply more indexing

When we wanted to do a third indexing, we found that we had no more variables to use, because the remaining variables in Airline or FlightActual are primary keys which are already indexed in our query.

Conclusion:

For this query, we only performed two indexing analyses. Only the first indexing decreased our cost and time, so we chose to index on name in Airline.