

# **DATABASE DESIGN**

## **DDL TABLE COMMANDS -**

```
CREATE TABLE UserLogin(  
Email VARCHAR(255) PRIMARY KEY,  
Password VARCHAR(255),  
Authority INT  
);
```

```
CREATE TABLE Airports(  
AirportId VARCHAR(10) PRIMARY KEY,  
AirportName VARCHAR(255),  
City VARCHAR(255),  
State VARCHAR(255),  
Latitude REAL,  
Longitude REAL  
);
```

```
CREATE TABLE Airline(  
AirlineId VARCHAR(5) PRIMARY KEY,  
AirlineName VARCHAR(255)  
);
```

```
CREATE TABLE Services(  
AirportId VARCHAR(10),  
AirlineId VARCHAR(5),  
PRIMARY KEY(AirportId, AirlineId),  
FOREIGN KEY (AirportId) REFERENCES Airports(AirportId),  
FOREIGN KEY (AirlineId) REFERENCES Airline(AirlineId)  
);
```

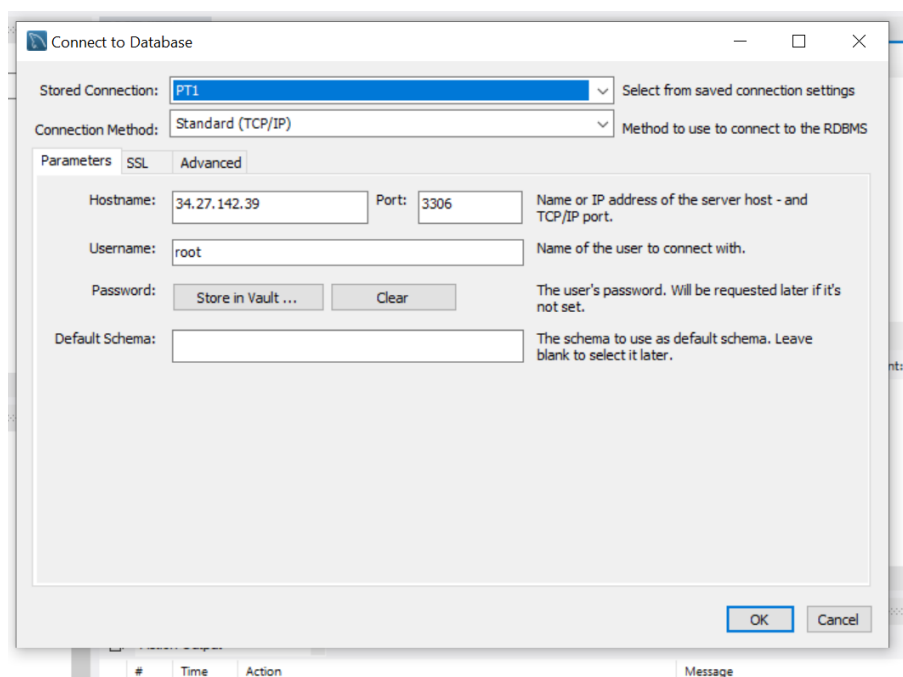
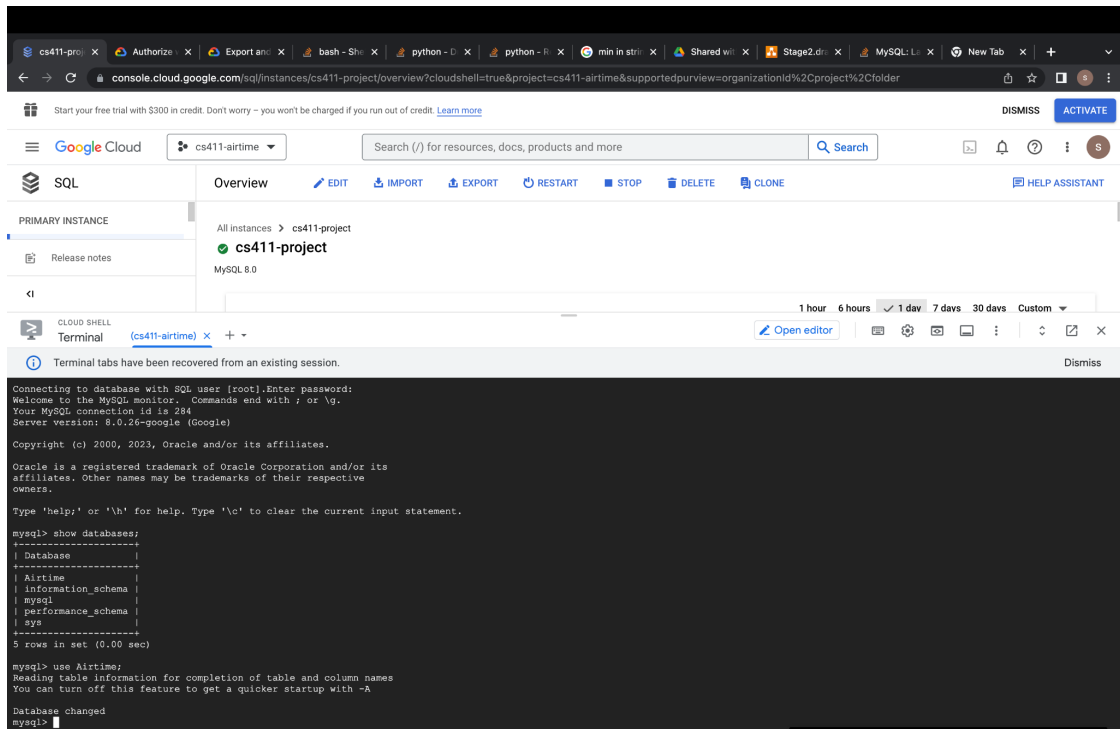
**CREATE TABLE FlightReviews(**

ReviewId INT PRIMARY KEY,  
AirlineId VARCHAR(5),  
ReviewerEmail VARCHAR(255),  
ReviewDate DATE,  
CustomerReview TEXT,  
TravellerType VARCHAR(50),  
Cabin VARCHAR(50),  
Route VARCHAR(255),  
DateFlown VARCHAR(255),  
SeatComfort REAL,  
CabinService REAL,  
FoodBev REAL,  
Entertainment REAL,  
GroundService REAL,  
ValueForMoney REAL,  
Recommended VARCHAR(10),  
FOREIGN KEY(AirlineId) REFERENCES Airline (AirlineId),  
FOREIGN KEY (ReviewerEmail) REFERENCES UserLogin(Email)  
);

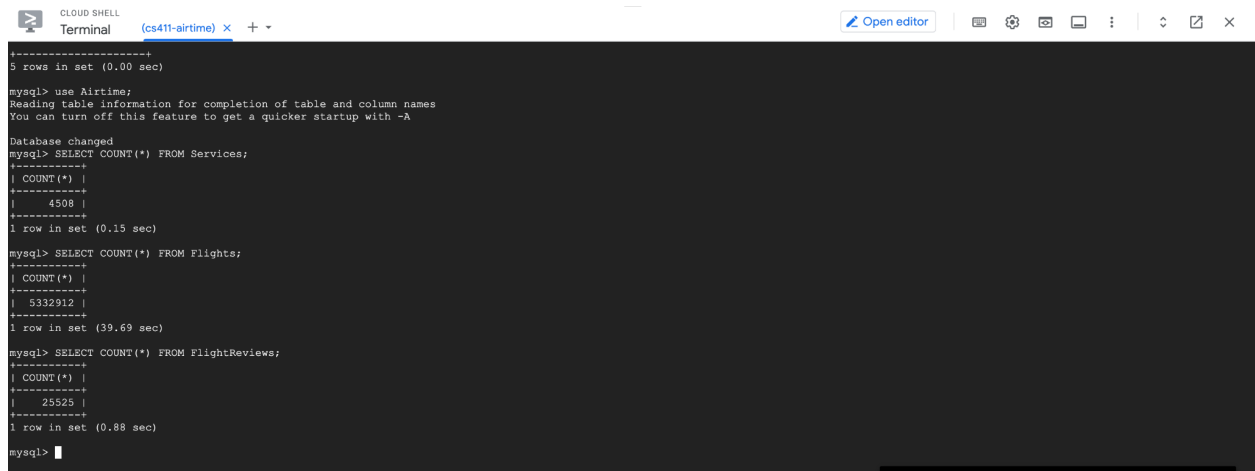
**CREATE TABLE Flights(**

Date DATE,  
DayOfWeek INT,  
AirlineId VARCHAR(5),  
FlightNumber INT,  
OriginAirport VARCHAR(10) ,  
DestinationAirport VARCHAR(10) ,  
ScheduledDeparture VARCHAR(5),  
DepartureDelay INT,  
Distance REAL,  
ScheduledArrival VARCHAR(5) ,  
ArrivalDelay INT ,  
AirlineName VARCHAR(255),  
PRIMARY KEY (Date, FlightNumber, AirlineId ),  
FOREIGN KEY (OriginAirport) REFERENCES Airport(AirportId),  
FOREIGN KEY (DestinationAirport) REFERENCES Airport (AirportId),  
FOREIGN KEY (AirlineId) REFERENCES Airline(AirlineId) ON DELETE CASCADE  
);

## CONNECTION SCREENSHOTS-



## TABLE ROW COUNT-



```
Cloud SHELL
Terminal (cs411-airtime) x + -
Open editor

+-----+
5 rows in set (0.00 sec)

mysql> use Airtime;
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> SELECT COUNT(*) FROM Services;
+-----+
| COUNT(*) |
+-----+
|      4508 |
+-----+
1 row in set (0.15 sec)

mysql> SELECT COUNT(*) FROM Flights;
+-----+
| COUNT(*) |
+-----+
| 5332912 |
+-----+
1 row in set (39.69 sec)

mysql> SELECT COUNT(*) FROM FlightReviews;
+-----+
| COUNT(*) |
+-----+
|    25525 |
+-----+
1 row in set (0.88 sec)

mysql>
```

We counted the number of rows in the following tables:

1. Flights: 5332912 rows
2. Services: 4508 rows
3. FlightReviews: 25525 rows

## ADVANCED QUERIES-

1.)

```
SELECT f.AirlineName, temp.delayCount / COUNT(*) as delayProbability
FROM Flights f JOIN (SELECT f1.AirlineName, COUNT(*) as delayCount
FROM Flights f1
WHERE f1.DepartureDelay > 0 OR f1.ArrivalDelay > 0
GROUP BY f1.AirlineName) as temp
USING (AirlineName)
GROUP BY f.AirlineName
ORDER BY delayProbability;
```

	AirlineName	delayProbability
▶	Alaska Airlines Inc.	0.4022
	Delta Air Lines Inc.	0.4161
	American Eagle Airlines Inc.	0.4281
	Hawaiian Airlines Inc.	0.4313
	Atlantic Southeast Airlines	0.4323
	Skywest Airlines Inc.	0.4360
	American Airlines Inc.	0.4554
	US Airways Inc.	0.4646
	JetBlue Airways	0.4795
	Virgin America	0.5037
	Southwest Airlines Co.	0.5186
	Frontier Airlines Inc.	0.5306
	United Air Lines Inc.	0.5667
	Spirit Air Lines	0.5688

There are only 14 Airlines in the database, so we get only 14 rows, one for each Airline as shown above. The above query tells us the delay probability of each Airline.

2.)

```
SELECT A.AirlineName, (SUM(o.performance)/COUNT(A.AirlineName)) as avgPerformance
FROM (SELECT fr.AirlineName, (fr.CabinService + fr.ValueForMoney + fr.Entertainment + fr.FoodBev) AS performance
FROM FlightReviews fr
) AS o JOIN Airline A USING (AirlineName)
GROUP BY A.AirlineName
ORDER BY avgPerformance DESC
LIMIT 5;
```

Result Grid			Filter Rows:
	AirlineName	avgPerformance	
▶	Virgin America	10.633440514469454	
	Delta Air Lines Inc.	9.293550778354337	
	Hawaiian Airlines Inc.	8.360655737704919	
	Alaska Airlines Inc.	8.30967741935484	
	JetBlue Airways	8.282015395381386	

We wanted to know only the top 5 best performing airlines.

## FIRST QUERY EXPLAIN ANALYZE-

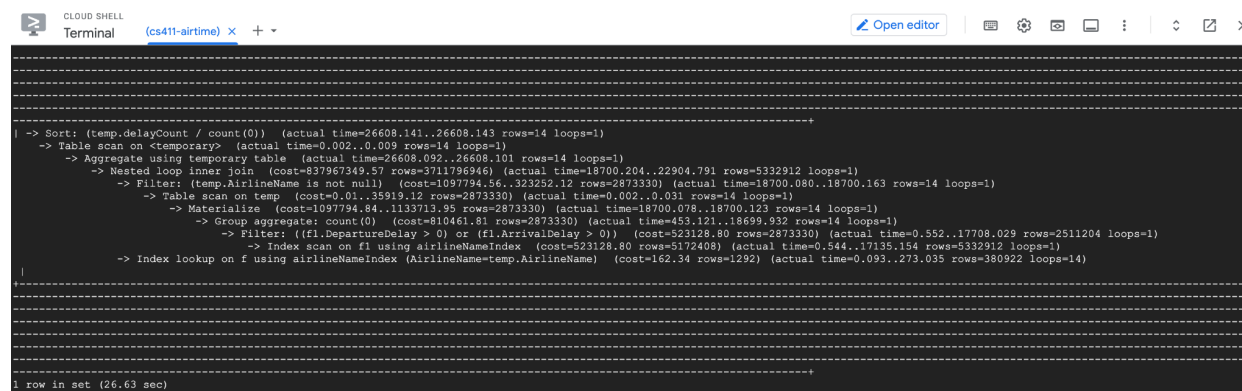
```
SELECT f.AirlineName, temp.delayCount / COUNT(*) as delayProbability
FROM Flights f JOIN (SELECT f1.AirlineName, COUNT(*) as delayCount
FROM Flights f1
WHERE f1.DepartureDelay > 0 OR f1.ArrivalDelay > 0
GROUP BY f1.AirlineName) as temp
USING (AirlineName)
GROUP BY f.AirlineName
ORDER BY delayProbability;
```

### 1. NO INDEXING-

```
1  -> Sort: (temp.delayCount / count(0)) (actual time=13961.012..13961.013 rows=14 loops=1)
2  -> Table scan on <temporary> (actual time=0.002..0.009 rows=14 loops=1)
3  -> Aggregate using temporary table (actual time=13960.862..13960.871 rows=14 loops=1)
4  -> Filter: (f.AirlineName = temp.AirlineName) (cost=66544244.30 rows=0) (actual time=4439.800..10454.536 rows=5332912 loops=1)
5  -> Inner hash join (<hash>(f.AirlineName)=<hash>(temp.AirlineName)) (cost=66544244.30 rows=0) (actual time=4439.795..8164.171 rows=5332912 loops=1)
6  -> Table scan on f (cost=23.18 rows=5172408) (actual time=0.079..2137.815 rows=5332912 loops=1)
7  -> Hash
8  -> Table scan on temp (cost=2.50..2.50 rows=0) (actual time=0.002..0.003 rows=14 loops=1)
9  -> Materialize (cost=2.50..2.50 rows=0) (actual time=4439.620..4439.623 rows=14 loops=1)
10 -> Table scan on <temporary> (actual time=0.002..0.008 rows=14 loops=1)
11 -> Aggregate using temporary table (actual time=4439.480..4439.487 rows=14 loops=1)
12 -> Filter: ((f1.DepartureDelay > 0) or (f1.ArrivalDelay > 0)) (cost=523128.80 rows=2873330) (actual time=0.199..2924.179 rows=2511204 loops=1)
13 -> Table scan on f1 (cost=523128.80 rows=5172408) (actual time=0.166..2350.023 rows=5332912 loops=1)
14
```

### 2. WITH INDEXING-

#### a. ON AIRLINE NAME-



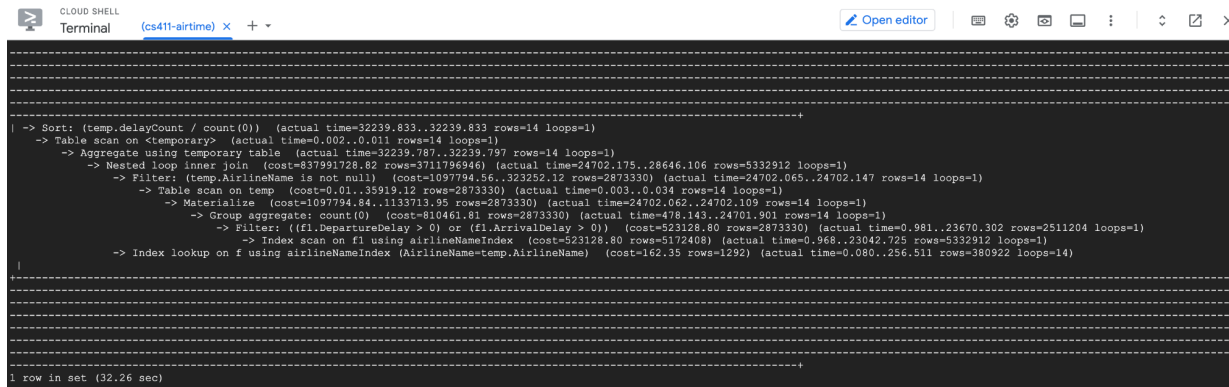
```
CLOUD SHELL
Terminal (cs411-airtime) x + - Open editor

1  -> Sort: (temp.delayCount / count(0)) (actual time=26608.141..26608.143 rows=14 loops=1)
2  -> Table scan on <temporary> (actual time=0.002..0.009 rows=14 loops=1)
3  -> Aggregate using temporary table (actual time=26608.092..26608.101 rows=14 loops=1)
4  -> Filter: (temp.AirlineName is not null) (cost=1097794.56..323252.12 rows=2873330) (actual time=18700.080..18700.163 rows=14 loops=1)
5  -> Table scan on temp (cost=0.01..35919.12 rows=2873330) (actual time=0.002..0.031 rows=14 loops=1)
6  -> Materialize (cost=1097794.84..1133713.95 rows=2873330) (actual time=18700.078..18700.123 rows=14 loops=1)
7  -> Group aggregate: count(0) (cost=810461.81 rows=2873330) (actual time=453.121..18699.932 rows=14 loops=1)
8  -> Filter: ((f1.DepartureDelay > 0) or (f1.ArrivalDelay > 0)) (cost=523128.80 rows=2873330) (actual time=0.552..17708.029 rows=2511204 loops=1)
9  -> Index scan on f1 using airlineNameIndex (cost=523128.80 rows=5172408) (actual time=0.544..17135.154 rows=5332912 loops=1)
10 -> Index lookup on f using airlineNameIndex (AirlineName=temp.AirlineName) (cost=162.34 rows=1292) (actual time=0.093..273.035 rows=380922 loops=14)

1 row in set (26.63 sec)
```

We tried indexing on the airline name, but the problem was that indexing on the airline name did not improve the performance because there are only 14 airlines and the number of flights is large. This means that the index has very low selectivity. The low selectivity makes it very difficult for the index to reduce the number of rows that need to be scanned. Instead, the engine would still have to perform a lot of lookups to get all matching rows.

## b. ON AIRLINE NAME, DEPARTURE DELAY AND ARRIVAL DELAY-

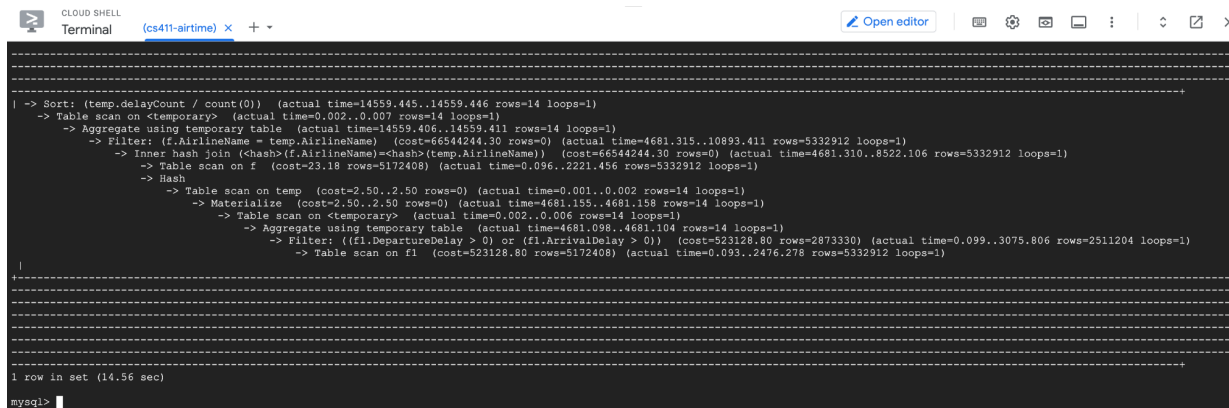


```
CLOUD SHELL
Terminal (cs411-airtime) x +
Open editor

1 -> Sort: (temp.delayCount / count(0)) (actual time=32239.833..32239.833 rows=14 loops=1)
    -> Table scan on <temporary> (actual time=0.002..0.011 rows=14 loops=1)
        -> Aggregate using temporary table (actual time=32239.797..32239.797 rows=14 loops=1)
            -> Nested loop inner join (cost=837991728.82 rows=3711796946) (actual time=24702.175..28646.106 rows=5332912 loops=1)
                -> Filter: (temp.AirlineName is not null) (cost=1097794.56..323252.12 rows=2873330) (actual time=24702.065..24702.147 rows=14 loops=1)
                    -> Table scan on temp (cost=0.01..35919.12 rows=2873330) (actual time=0.003..0.034 rows=14 loops=1)
                        -> Materialize (cost=1097794.84..113713.95 rows=2873330) (actual time=24702.062..24702.109 rows=14 loops=1)
                            -> Group aggregate: count(0) (cost=410461.81 rows=2873330) (actual time=478.143..24701.901 rows=14 loops=1)
                                -> Filter: ((f1.DepartureDelay > 0) or (f1.ArrivalDelay > 0)) (cost=523128.80 rows=2873330) (actual time=0.981..23670.302 rows=2511204 loops=1)
                                    -> Index scan on f1 using airlineNameIndex (cost=523128.80 rows=5172408) (actual time=0.968..23042.725 rows=5332912 loops=1)
                                        -> Index lookup on f using airlineNameIndex (AirlineName=temp.AirlineName) (cost=162.35 rows=1292) (actual time=0.080..256.511 rows=380922 loops=14)
1 row in set (32.26 sec)
```

We tried indexing on airline name, departure delay, and arrival delay, but the problem was that indexing on the airline name did not improve the performance because there are only 14 airlines. This means that the index has very low selectivity. The low selectivity makes it very difficult for the index to reduce the number of rows that need to be scanned. Instead, the engine would still have to perform a lot of lookups to get all matching rows. For the arrival and departure delay, they were not affecting the lookups as we were only using airline names to look up rows so adding them as indexes did not change anything.

## c. ON DEPARTURE DELAY AND ARRIVAL DELAY-



```
CLOUD SHELL
Terminal (cs411-airtime) x +
Open editor

1 -> Sort: (temp.delayCount / count(0)) (actual time=14559.445..14559.446 rows=14 loops=1)
    -> Table scan on <temporary> (actual time=0.002..0.007 rows=14 loops=1)
        -> Aggregate using temporary table (actual time=14559.406..14559.411 rows=14 loops=1)
            -> Filter: (f.AirlineName = temp.AirlineName) (cost=66544244.30 rows=0) (actual time=4681.315..10893.411 rows=5332912 loops=1)
                -> Inner hash join (<hash>(f.AirlineName)=<hash>(temp.AirlineName)) (cost=66544244.30 rows=0) (actual time=4681.310..8522.106 rows=5332912 loops=1)
                    -> Table scan on f (cost=23.18 rows=5172408) (actual time=0.096..2221.456 rows=5332912 loops=1)
                        -> Hash
                            -> Table scan on temp (cost=2.50..2.50 rows=0) (actual time=0.001..0.002 rows=14 loops=1)
                                -> Materialize (cost=2.50..2.50 rows=0) (actual time=4681.155..4681.158 rows=14 loops=1)
                                    -> Table scan on <temporary> (actual time=0.002..0.006 rows=14 loops=1)
                                        -> Aggregate using temporary table (actual time=4681.098..4681.104 rows=14 loops=1)
                                            -> Filter: ((f1.DepartureDelay > 0) or (f1.ArrivalDelay > 0)) (cost=523128.80 rows=2873330) (actual time=0.099..3075.806 rows=2511204 loops=1)
                                                -> Table scan on f1 (cost=523128.80 rows=5172408) (actual time=0.093..2476.278 rows=5332912 loops=1)
1 row in set (14.56 sec)

mysql>
```

For the arrival and departure delay, they were not affecting the lookups as we were only using airline names to look up rows so adding them as indexes did not change anything. This means that adding indexes for these attributes is not important as they are only used for filtering. Basically, the indexes that were added did not change anything when compared to the no index time/cost.

We will be going with the no index query (except the default indices) for our first advanced query. This is because when we added an index for the airline name, it did not



decrease the time or cost, in fact it increased both of them. We had discussed and came to the conclusion that adding an index was not helping the query as there were only 14 airlines, meaning that the index was not very selective. When we tried to add indexes on the departure delay or the arrival delay, we noticed no improvement in the performance. Using default indices gives the best query performance and hence we stick to that for this query.

```
SELECT A.AirlineName, (SUM(o.performance)/COUNT(A.AirlineName)) as avgPerformance
FROM (SELECT fr.AirlineName, (fr.CabinService + fr.ValueForMoney + fr.Entertainment + fr.FoodBev) AS performance
FROM FlightReviews fr
) AS o JOIN Airline A USING (AirlineName)
GROUP BY A.AirlineName
ORDER BY avgPerformance DESC
LIMIT 5;
```

```
CLOUD SHELL
Terminal [cs411-airtime] x + v Open editor       

|
-----+-----
|
-----+-----
| -> Limit: 5 row(s) (actual time=66.799..66.800 rows=5 loops=1)
|   -> Sort: (sum((((fr.CabinService + fr.ValueForMoney) + fr.Entertainment) + fr.FoodBev)) / count(A.AirlineName)) DESC, limit input to 5 row(s) per chunk (actual time=66.798..66.799 rows=5 loops=1)
|     -> Table scan on <temporary> (actual time=0.002..0.008 rows=12 loops=1)
|       -> Aggregate using temporary table (actual time=66.762..66.768 rows=12 loops=1)
|         -> Filter: (fr.AirlineName = A.AirlineName) (cost=36794.06 rows=36326) (actual time=0.136..42.522 rows=25525 loops=1)
|           -> Inner hash join (<hash>(fr.AirlineName)=<hash>(A.AirlineName)) (cost=36794.06 rows=36326) (actual time=0.132..29.213 rows=25525 loops=1)
|             -> Table scan on fr (cost=47.27 rows=24217) (actual time=0.057..20.404 rows=25525 loops=1)
|               -> Hash
|                 -> Table scan on A (cost=1.75 rows=15) (actual time=0.042..0.047 rows=14 loops=1)
|
|
-----+-----
1 row in set (0.07 sec)

mysql>
```

**a. ON AIRLINE NAME-**

[illegible]

So looking at the difference between the query without indexing vs the one with airline name as an index shows that the cost has decreased, but the time has increased. We believe the decrease in cost is a good sign that the airline name works as an index. This index works because the flight review table is not as big as the flights table, so we did not have the same problem that the first query had.

After looking at all the different index designs, we have decided to go with the airline name, cabin service, and value for money indexes. This is because we believe that it uses the least cost and the least time available with the given options. Like we have specified above, the airline name index is used to find the rows faster, and the cabin service/value for money indexes decrease the time it takes to finish the query.