3 Tables with >= Rows

```
mysql> SHOW TABLES;
+------------------------+
| Tables_in_flight_delay |
+------------------------+
| Airlines               |
| Cities                 |
| Delay                  |
| Favorites              |
| Flights                |
| Users                  |
+------------------------+
6 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Flights;
+----------+
| COUNT(*) |
+----------+
|  1045529 |
+----------+
1 row in set (1.52 sec)

mysql> SELECT COUNT(*) FROM Delay;
+----------+
| COUNT(*) |
+----------+
|  1045529 |
+----------+
1 row in set (1.29 sec)

mysql> SELECT COUNT(*) FROM Cities;
+----------+
| COUNT(*) |
+----------+
|    21289 |
+----------+
1 row in set (0.15 sec)
```

Data Definition Language

```
CREATE TABLE Cities (
      city VARCHAR(50),
      state VARCHAR(5),
      population INT,

    PRIMARY KEY (city, state)
);

CREATE TABLE Airlines (
      airline_code VARCHAR(5) PRIMARY KEY,
      airline VARCHAR(50),
      annual_passengers DOUBLE,
      avail_seat_miles DOUBLE
);
```

```
CREATE TABLE Flights (
      flight_date DATETIME,
      flight_number INT,
      airline_code VARCHAR(5),
      origin_city VARCHAR(50),
      origin_state VARCHAR(5),
      origin_airport VARCHAR(5),
      dest_city VARCHAR(50),
      dest_state VARCHAR(5),
      dest_airport VARCHAR(5),
      avg_price DOUBLE,

      PRIMARY KEY (flight_date, flight_number, airline_code),
      FOREIGN KEY (origin_city, origin_state) REFERENCES
            Cities(city, state),
      FOREIGN KEY (airline_code) REFERENCES Airlines(airline_code)
);

CREATE TABLE Delay (
      flight_date DATETIME,
      flight_number INT,
      airline_code VARCHAR(5),

      dep_delay FLOAT,
      arr_delay FLOAT,
      carrier_delay FLOAT,
      weather_delay FLOAT,
      nas_delay FLOAT,
      security_delay FLOAT,
      late_aircraft_delay FLOAT,

      FOREIGN KEY (flight_date, flight_number, airline_code) REFERENCES
            Flights (flight_date, flight_number, airline_code)
                  ON DELETE SET NULL
                  ON UPDATE CASCADE
);
```

Schema Changes

- Since Stage 2, we have made the following changes to our schema:
    - Replaced the Favorites entity with a many-many Favorites relationship connecting User and Flights (the previous Favorites entity was redundant)
    - Added an Airlines entity to store valuable information about each individual airline that we could use in our advanced queries (i.e. annual passengers and available seat miles)

- Changed Flights primary key from just flight_number to a set of 3 keys (flight_number, flight_date, and airline) so that flights could properly be uniquely identified (applied these changes to any entities with foreign keys referencing Flights)

Advanced Queries + Indexing Analysis

Query 1

- This query finds the number of flights between all origin and destination city pairs where the origin city has certain population threshold.

- SQL code:

```
SELECT origin_city, dest_city, COUNT(*)
      FROM (
            SELECT *
            FROM Flights f JOIN Cities c ON f.origin_city = c.city
            WHERE c.population > 1000000
      ) AS filtered_flights
GROUP BY origin_city, dest_city;
```

- Top 15 rows:

origin_city	dest_city	COUNT(*)
Chicago	Akron	202
Chicago	Baltimore	2254
Chicago	Burbank	101
Chicago	Cincinnati	1943
Chicago	Cleveland	2128
Chicago	Colorado Springs	497
Chicago	Columbia	28
Chicago	Columbus	2107
Chicago	Dallas	1402
Chicago	Dallas/Fort Worth	2792
Chicago	Dayton	431
Chicago	Denver	4065
Chicago	Des Moines	1086
Chicago	Detroit	3327
Chicago	Eagle	127
Chicago	El Paso	351

Query 1 Index Analysis

- Original (before adding indexes)

```
| -> Table scan on <temporary>  (actual time=77225.087..77225.210 rows=429 loops=1)
    -> Aggregate using temporary table  (actual time=77225.081..77225.081 rows=429 loops=1)
        -> Nested loop inner join  (cost=516414.93 rows=508295) (actual time=109.192..55390.583 rows=393577 loops=1)
            -> Filter: (f.origin_city is not null)  (cost=98706.83 rows=949152) (actual time=108.170..8824.510 rows=1045529 loops=1)
                -> Table scan on f  (cost=98706.83 rows=949152) (actual time=108.165..8063.323 rows=1045529 loops=1)
            -> Filter: (c.population > 1000000)  (cost=0.28 rows=1) (actual time=0.039..0.044 rows=0 loops=1045529)
                -> Index lookup on c using PRIMARY (city=f.origin_city)  (cost=0.28 rows=2) (actual time=0.021..0.039 rows=4 loops=1045529)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------+
1 row in set (1 min 17.32 sec)
```

- Index configuration 1: index on Flights(origin_city)

```
| -> Table scan on <temporary>  (actual time=8877.900..8878.029 rows=429 loops=1)
    -> Aggregate using temporary table  (actual time=8877.893..8877.893 rows=429 loops=1)
        -> Nested loop inner join  (cost=530724.83 rows=508295) (actual time=81.822..7038.289 rows=393577 loops=1)
            -> Filter: (f.origin_city is not null)  (cost=99058.61 rows=949152) (actual time=81.126..1612.394 rows=1045529 loops=1)
                -> Table scan on f  (cost=99058.61 rows=949152) (actual time=81.122..1526.404 rows=1045529 loops=1)
            -> Filter: (c.population > 1000000)  (cost=0.29 rows=1) (actual time=0.005..0.005 rows=0 loops=1045529)
                -> Index lookup on c using PRIMARY (city=f.origin_city)  (cost=0.29 rows=2) (actual time=0.002..0.005 rows=4 loops=1045529)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------+
1 row in set (8.95 sec)
```

- Index configuration 2: indexes on Flights(origin_city) and Cities(population)

```
| -> Table scan on <temporary>  (actual time=29051.883..29052.023 rows=429 loops=1)
    -> Aggregate using temporary table  (actual time=29050.958..29050.958 rows=429 loops=1)
        -> Nested loop inner join  (cost=66532.15 rows=66927) (actual time=488.121..27027.210 rows=393577 loops=1)
            -> Filter: (c.population > 1000000)  (cost=3.46 rows=11) (actual time=0.014..77.044 rows=11 loops=1)
                -> Covering index range scan on c using c_population_idx over (1000000 < population)  (cost=3.46 rows=11) (actual time=0.012..76.996 rows=11 loops=1)
            -> Index lookup on f using origin_city (origin_city=c.city)  (cost=5494.94 rows=6084) (actual time=117.667..2446.885 rows=35780 loops=11)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (29.06 sec)
```

- Index configuration 3: index on Cities(population)

```
| -> Table scan on <temporary>  (actual time=26657.610..26657.776 rows=429 loops=1)
    -> Aggregate using temporary table  (actual time=26656.799..26656.799 rows=429 loops=1)
        -> Nested loop inner join  (cost=60835.08 rows=66927) (actual time=489.441..24799.827 rows=393577 loops=1)
            -> Filter: (c.population > 1000000)  (cost=3.49 rows=11) (actual time=0.025..46.705 rows=11 loops=1)
                -> Covering index range scan on c using c_population_idx over (1000000 < population)  (cost=3.49 rows=11) (actual time=0.023..46.646 rows=11 loops=1)
            -> Index lookup on f using origin_city (origin_city=c.city)  (cost=4977.03 rows=6084) (actual time=108.365..2247.225 rows=35780 loops=11)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (26.75 sec)
```

- Analysis:
  - Original cost was 516414.93.
  - Index configuration 1 did not significantly impact query performance (cost of 530724.83), which I believe is due to how filtering the join on Cities and Flights is primarily based on the population column in Cities. Having quick lookup time for whether a city exists in Flights is not very useful when you still have to iterate through every row in Flights regardless.
  - Index configurations 2 and 3 greatly improved costs to 66532.15 and 60835.08. When joining Flights and Cities on the condition of cities, for the 1 million+ rows in Flights,

instead of having to scan Cities each time to check for the existence of a city, you can index it and determine existence in O(1).

Query 2

- This query finds the number of flights on airlines that have a certain number of annual passengers.

- SQL code:

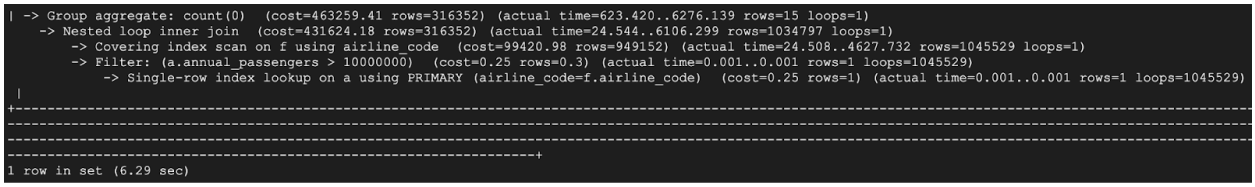
```
SELECT airline, airline_code, COUNT(*)
      FROM (
            SELECT f.airline_code, a.airline
            FROM Flights f JOIN Airlines a ON
                  f.airline_code = a.airline_code
            WHERE a.annual_passengers > 10000000
      ) AS filtered_flights
GROUP BY airline_code;
```

- Top 15 rows (exactly 15 rows in the output table):

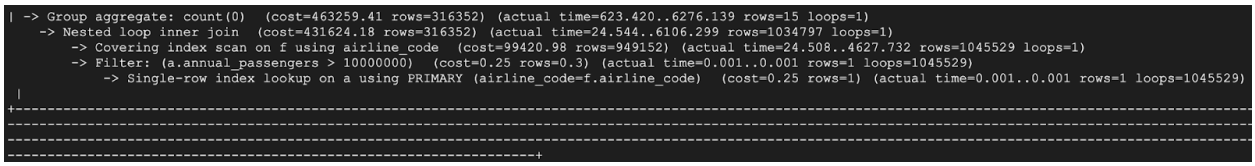
airline	airline_code	COUNT(*)
Endeavor	9E	27576
American Airlines	AA	159369
Alaska Airlines	AS	34078
JetBlue Airways	B6	50981
Delta Air Lines	DL	128088
Frontier Airlines	F9	23266
Allegiant Air	G4	11187
Envoy Air	MQ	34353
Spirit Airlines	NK	43138
PSA Airlines	OH	17317
SkyWest Airlines	OO	78439
United Airlines	UA	106042
Southwest Airlines	WN	234832
Mesa Airlines	YV	25886
Republic Airways	YX	60245

Query 2 Index Analysis

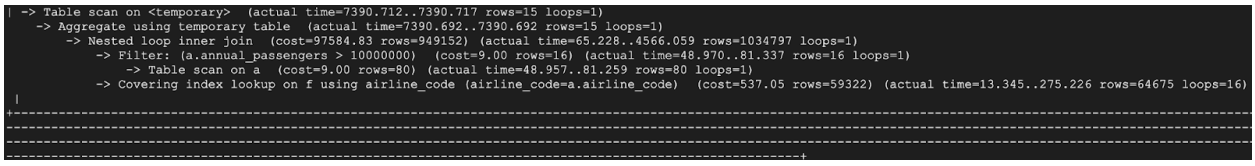
- Original (before adding indexes)



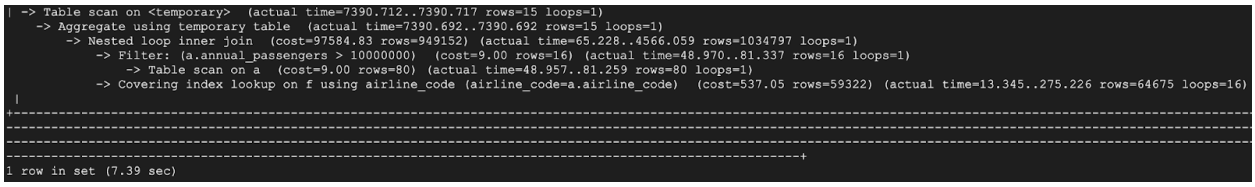
- Index configuration 2: index on Airlines(annual_passengers)



- Index configuration 2: indexes on Airlines(annual_passengers) and Airlines(airline_code)



- Index configuration 3: index on Airlines(airline_code)



- Analysis:
    - Original cost was 463259.41.
    - Index configuration 1 did not improve cost. Having a quick lookup for annual_passengers does not matter since, for each joined row of Flights and Airlines, a comparison of annual_passengers to 0 still needs to be made. Determining the existence of a specific annual_passengers value is not beneficial in this case.
    - Indexes 2 and 3 cut the cost down to 97584.83 (over 4x improvement). The reason for this is very similar to why the index on Cities(city) greatly improved the performance of query 1. In the process of joining Flights and Airlines on the condition of airline_code, for each of the 1 million+ rows in Flights, instead of having to scan Airlines each time for the existence of an airline, you can index it and determine existence in O(1).

Query 3

- Airports in terms of incoming and outgoing flights for

- SQL code:

```
SELECT origin_airport AS airport, num_outgoing, num_incoming
FROM
      (SELECT origin_airport, COUNT(origin_airport) AS num_outgoing
      FROM Flights f
      GROUP BY origin_airport) as outgoing
JOIN
      (SELECT dest_airport, COUNT(dest_airport) AS num_incoming
      FROM Flights f
      GROUP BY dest_airport) as incoming
ON origin_airport = dest_airport
ORDER BY num_outgoing DESC, num_incoming DESC;
```

- Top 15 rows:

airport	num_outgoing	num_incoming
ORD	88978	15221
DFW	82854	20475
BOS	51613	2788
LAX	45592	32865
LAS	43667	24710
DEN	43603	10777
IAH	36579	19620
DTW	32713	15458
CLT	32090	2992
MDW	29220	4871
DAL	23397	6414
PHX	19908	46256
JFK	19569	28198
MIA	19281	20068
FLL	18048	20159
MSP	17408	25439

Query 3 Index Analysis

- Original (before adding indexes)

```
| -> Sort: outgoing.num_outgoing DESC, incoming.num_incoming DESC  (actual time=32933.482..32933.488 rows=115 loops=1)
    -> Stream results  (cost=2479667.10 rows=0) (actual time=32933.174..32933.398 rows=115 loops=1)
        -> Nested loop inner join  (cost=2479667.10 rows=0) (actual time=32933.168..32933.370 rows=115 loops=1)
            -> Filter: (outgoing.origin_airport is not null)  (cost=0.11..106782.10 rows=949152) (actual time=15665.049..15665.085 rows=158 loops=1)
                -> Table scan on outgoing  (cost=2.50..2.50 rows=0) (actual time=15665.045..15665.070 rows=158 loops=1)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=15665.044..15665.044 rows=158 loops=1)
                        -> Table scan on <temporary>  (actual time=15664.863..15664.974 rows=158 loops=1)
                            -> Aggregate using temporary table  (actual time=15664.858..15664.858 rows=158 loops=1)
                                -> Table scan on f  (cost=99532.18 rows=949152) (actual time=133.460..12324.831 rows=1045529 loops=1)
            -> Index lookup on incoming using <auto_key0> (dest_airport=outgoing.origin_airport)  (actual time=109.293..109.293 rows=1 loops=158)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=17268.089..17268.089 rows=147 loops=1)
                    -> Table scan on <temporary>  (actual time=17267.811..17267.919 rows=147 loops=1)
                        -> Aggregate using temporary table  (actual time=17267.806..17267.806 rows=147 loops=1)
                            -> Table scan on f  (cost=99532.18 rows=949152) (actual time=224.985..13717.052 rows=1045529 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (33.14 sec)
```

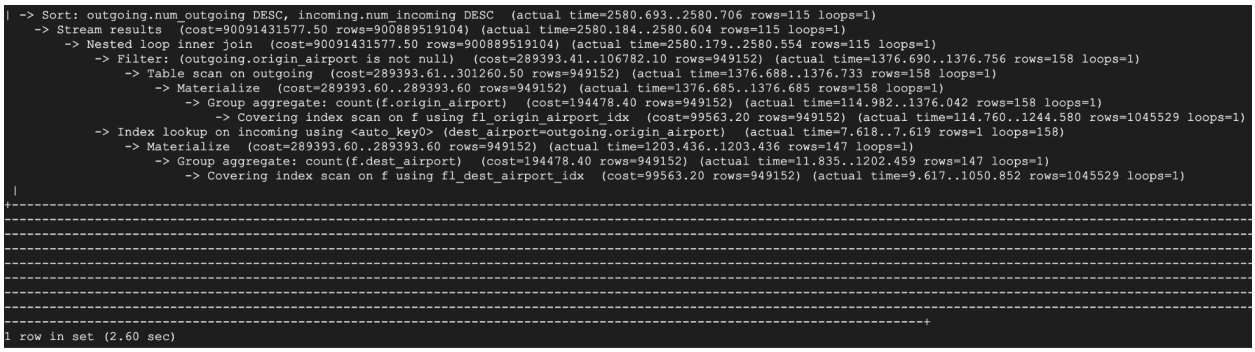
- Index configuration 1: index on Flights(origin_airport)

```
| -> Sort: outgoing.num_outgoing DESC, incoming.num_incoming DESC  (actual time=6245.183..6245.189 rows=115 loops=1)
    -> Stream results  (cost=2479667.10 rows=0) (actual time=6244.893..6245.115 rows=115 loops=1)
        -> Nested loop inner join  (cost=2479667.10 rows=0) (actual time=6244.886..6245.084 rows=115 loops=1)
            -> Filter: (outgoing.origin_airport is not null)  (cost=289393.41..106782.10 rows=949152) (actual time=1020.668..1020.705 rows=158 loops=1)
                -> Table scan on outgoing  (cost=289393.61..301260.50 rows=949152) (actual time=1020.666..1020.691 rows=158 loops=1)
                    -> Materialize  (cost=289393.60..289393.60 rows=949152) (actual time=1020.662..1020.662 rows=158 loops=1)
                        -> Group aggregate: count(f.origin_airport)  (cost=194478.40 rows=949152) (actual time=0.286..1020.038 rows=158 loops=1)
                            -> Covering index scan on f using fl_origin_airport_idx  (cost=99563.20 rows=949152) (actual time=0.083..883.647 rows=1045529 loops=1)
            -> Index lookup on incoming using <auto_key0> (dest_airport=outgoing.origin_airport)  (actual time=33.065..33.065 rows=1 loops=158)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=5224.188..5224.188 rows=147 loops=1)
                    -> Table scan on <temporary>  (actual time=5223.994..5224.041 rows=147 loops=1)
                        -> Aggregate using temporary table  (actual time=5223.989..5223.989 rows=147 loops=1)
                            -> Table scan on f  (cost=99563.20 rows=949152) (actual time=70.280..1144.917 rows=1045529 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (6.26 sec)
```

- Index configuration 2: index on Flights(dest_airport)

```
| -> Sort: outgoing.num_outgoing DESC, incoming.num_incoming DESC  (actual time=6245.183..6245.189 rows=115 loops=1)
    -> Stream results  (cost=2479667.10 rows=0) (actual time=6244.893..6245.115 rows=115 loops=1)
        -> Nested loop inner join  (cost=2479667.10 rows=0) (actual time=6244.886..6245.084 rows=115 loops=1)
            -> Filter: (outgoing.origin_airport is not null)  (cost=289393.41..106782.10 rows=949152) (actual time=1020.668..1020.705 rows=158 loops=1)
                -> Table scan on outgoing  (cost=289393.61..301260.50 rows=949152) (actual time=1020.666..1020.691 rows=158 loops=1)
                    -> Materialize  (cost=289393.60..289393.60 rows=949152) (actual time=1020.662..1020.662 rows=158 loops=1)
                        -> Group aggregate: count(f.origin_airport)  (cost=194478.40 rows=949152) (actual time=0.286..1020.038 rows=158 loops=1)
                            -> Covering index scan on f using fl_origin_airport_idx  (cost=99563.20 rows=949152) (actual time=0.083..883.647 rows=1045529 loops=1)
            -> Index lookup on incoming using <auto_key0> (dest_airport=outgoing.origin_airport)  (actual time=33.065..33.065 rows=1 loops=158)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=5224.188..5224.188 rows=147 loops=1)
                    -> Table scan on <temporary>  (actual time=5223.994..5224.041 rows=147 loops=1)
                        -> Aggregate using temporary table  (actual time=5223.989..5223.989 rows=147 loops=1)
                            -> Table scan on f  (cost=99563.20 rows=949152) (actual time=70.280..1144.917 rows=1045529 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------+
```

- Index configuration 3: indexes on Flights(origin_airport) and Flights(dest_airport)



- Analysis:
  - Original cost was 2479667.10.
  - Index configurations 1 and 2 saw no improvement in cost. The query involves two separate aggregations on the Flights table: one for counting outgoing flights and one for counting incoming flights. For each individual aggregation, having O(1) lookup for an origin_airport or dest_aiaport does not matter since you still have to go through all rows in order to determine the *count* of each airport.
  - Index configuration 3 actually worsened cost by a large factor. We are not completely sure, but we believe this might be due to low selectivity (many rows share the same origin or destination airport) or index overhead (the cost of maintaining this additional index outweighs the performance gains)

Query 4

- For each airline, this query finds the number of early/on-time flights, number of late flights, and the proportion of flights that are early/on-time.

- SQL code:

```
SELECT early.airline_code, num_early, num_late,
       num_early / (num_early + num_late) AS prop_early
FROM
       (SELECT airline_code, COUNT(*) AS num_early
       FROM Delay
       WHERE dep_delay <= 0
       GROUP BY airline_code) AS early
JOIN
       (SELECT airline_code, COUNT(*) AS num_late
       FROM Delay
       WHERE dep_delay > 0
       GROUP BY airline_code) AS late
ON early.airline_code = late.airline_code;
```

- Top 15 rows:

airline_code	num_early	num_late	prop_early
UA	68040	38002	0.6416
DL	86006	42082	0.6715
YX	47833	12412	0.7940
NK	27392	15746	0.6350
WN	125480	109352	0.5343
AA	99426	59943	0.6239
B6	30672	20309	0.6016
9E	21583	5993	0.7827
OO	58805	19634	0.7497
MQ	25247	9106	0.7349
G4	7182	4005	0.6420
OH	12740	4577	0.7357
AS	23527	10551	0.6904
EV	5001	1800	0.7353
F9	13917	9349	0.5982

Query 4 Index Analysis

- Original (before adding indexes)

```
| -> Nested loop inner join  (cost=848171.08 rows=0) (actual time=4568.614..4568.639 rows=17 loops=1)
    -> Filter: (early.airline_code is not null)  (cost=0.11..36526.08 rows=324654) (actual time=2609.700..2609.707 rows=17 loops=1)
        -> Table scan on early  (cost=2.50..2.50 rows=0) (actual time=2609.697..2609.702 rows=17 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2609.696..2609.696 rows=17 loops=1)
                -> Table scan on <temporary>  (actual time=2609.656..2609.659 rows=17 loops=1)
                    -> Aggregate using temporary table  (actual time=2609.653..2609.653 rows=17 loops=1)
                        -> Filter: (Delay.dep_delay <= 0)  (cost=101733.19 rows=324655) (actual time=48.329..2227.118 rows=674990 loops=1)
                            -> Table scan on Delay  (cost=101733.19 rows=974062) (actual time=48.324..2121.839 rows=1045529 loops=1)
    -> Index lookup on late using <auto_key0> (airline_code=early.airline_code)  (actual time=115.230..115.231 rows=1 loops=17)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1958.899..1958.899 rows=17 loops=1)
            -> Table scan on <temporary>  (actual time=1958.837..1958.839 rows=17 loops=1)
                -> Aggregate using temporary table  (actual time=1958.834..1958.834 rows=17 loops=1)
                    -> Filter: (Delay.dep_delay > 0)  (cost=101733.19 rows=324655) (actual time=110.068..1745.970 rows=370539 loops=1)
                        -> Table scan on Delay  (cost=101733.19 rows=974062) (actual time=110.058..1650.756 rows=1045529 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+
1 row in set (4.59 sec)
```

- Index configuration 1: index on Delay(dep_delay)

```
| -> Nested loop inner join  (cost=1272373.49 rows=0) (actual time=3414.366..3414.391 rows=17 loops=1)
    -> Filter: (early.airline_code is not null)  (cost=0.11..54793.49 rows=487031) (actual time=1893.931..1893.938 rows=17 loops=1)
        -> Table scan on early  (cost=2.50..2.50 rows=0) (actual time=1893.929..1893.933 rows=17 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1893.928..1893.928 rows=17 loops=1)
                -> Table scan on <temporary>  (actual time=1893.892..1893.895 rows=17 loops=1)
                    -> Aggregate using temporary table  (actual time=1893.890..1893.890 rows=17 loops=1)
                        -> Filter: (Delay.dep_delay <= 0)  (cost=101733.20 rows=487031) (actual time=60.341..1527.202 rows=674990 loops=1)
                            -> Table scan on Delay  (cost=101733.20 rows=974062) (actual time=60.335..1428.337 rows=1045529 loops=1)
    -> Index lookup on late using <auto_key0> (airline_code=early.airline_code)  (actual time=89.438..89.438 rows=1 loops=17)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1520.425..1520.425 rows=17 loops=1)
            -> Table scan on <temporary>  (actual time=1520.388..1520.390 rows=17 loops=1)
                -> Aggregate using temporary table  (actual time=1520.385..1520.385 rows=17 loops=1)
                    -> Filter: (Delay.dep_delay > 0)  (cost=101733.20 rows=487031) (actual time=2.330..1320.895 rows=370539 loops=1)
                        -> Table scan on Delay  (cost=101733.20 rows=974062) (actual time=2.322..1234.574 rows=1045529 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------+
1 row in set (3.68 sec)
```

- Analysis:
  - Original cost was 848171.08.
  - Index configuration 1 saw no improvement in the cost. Having quick indexing on dep_delay in Delay does not matter, as for each row in Delay, dep_delay still has to be compared to 0 regardless. In other words, determining the existence of a specific dep_delay in O(1) is pointless.
  - This is the only index configuration to consider. I don't believe any other indexing configurations would improve cost since all other used attributes are either foreign or primary keys which already have an index.