

Database Implementation and Indexing

Screenshot of connection to GCP:

```
Your Cloud Platform project in this session is set to travel-planner-440113.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
akilkarthikeyan90@cloudshell:~ (travel-planner-440113)$ gcloud sql connect travel-planner-mysql --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1489
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use travel_planner;
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_travel_planner |
+-----+
| airbnb                    |
| airline                   |
| flight                    |
| host                      |
| plan                      |
| plan_airbnb               |
| plan_flight               |
| user                      |
+-----+
8 rows in set (0.01 sec)

mysql> █
```

Data Definition Language (DDL):

```
DROP TABLE IF EXISTS plan_flight;
DROP TABLE IF EXISTS plan_airbnb;
DROP TABLE IF EXISTS plan;
DROP TABLE IF EXISTS airbnb;
DROP TABLE IF EXISTS flight;
DROP TABLE IF EXISTS user;
DROP TABLE IF EXISTS airline;
DROP TABLE IF EXISTS host;
```

```
CREATE TABLE host
(
    host_id          INT,
```

```
    host_url          VARCHAR(300),
    host_name          VARCHAR(50),
    host_response_rate NUMERIC,
    host_acceptance_rate NUMERIC,
    host_is_superhost  BOOL,
    host_identity_verified BOOL,
    PRIMARY KEY (host_id)
);
```

```
CREATE TABLE airline
(
    airline_id VARCHAR(2),
    airline_name VARCHAR(50),
    PRIMARY KEY (airline_id)
);
```

```
CREATE TABLE user
(
    user_id INT AUTO_INCREMENT,
    user_name VARCHAR(50) NOT NULL,
    phone NUMERIC,
    email VARCHAR(50),
    PRIMARY KEY (user_id)
);
```

```
CREATE TABLE flight
(
    flight_id          VARCHAR(50),
    flight_date         DATE          NOT NULL,
    starting_airport    VARCHAR(3)    NOT NULL,
    destination_airport VARCHAR(3)    NOT NULL,
    travel_duration     NUMERIC,
    is_basic_economy    BOOL,
    total_fare          DECIMAL(10, 2) NOT NULL,
    departure_time      TIME          NOT NULL,
    arrival_time        TIME          NOT NULL,
    equipment_description VARCHAR(50),
    airline_id          VARCHAR(2)    NOT NULL,
    FOREIGN KEY (airline_id) REFERENCES airline (airline_id),
    PRIMARY KEY (flight_id)
);
```

```
CREATE TABLE airbnb
(
```

```

airbnb_id      INT,
listing_url    VARCHAR(300),
name           VARCHAR(150),
description    VARCHAR(300),
neighborhood_overview VARCHAR(300),
picture_url    VARCHAR(300),
latitude       DECIMAL(8, 6),
longitude      DECIMAL(9, 6),
property_type  VARCHAR(150),
accommodates   NUMERIC,
bathrooms      NUMERIC,
bedrooms       NUMERIC,
beds           NUMERIC,
amenities      JSON,
price          NUMERIC NOT NULL,
number_of_reviews NUMERIC,
review_scores_rating NUMERIC NOT NULL,
close_to_airport VARCHAR(3) NOT NULL,
host_id        INT NOT NULL,
FOREIGN KEY (host_id) REFERENCES host (host_id),
PRIMARY KEY (airbnb_id)
);

```

```

CREATE TABLE plan
(
    plan_id      INT AUTO_INCREMENT,
    plan_name     VARCHAR(50) NOT NULL,
    plan_description VARCHAR(150),
    user_id      INT,
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON DELETE CASCADE,
    PRIMARY KEY (plan_id)
);

```

```

CREATE TABLE plan_airbnb
(
    plan_id  INT,
    airbnb_id INT,
    start_date DATE,
    end_date  DATE,
    ordinal  INT,
    FOREIGN KEY (plan_id) REFERENCES plan (plan_id) ON DELETE CASCADE,
    FOREIGN KEY (airbnb_id) REFERENCES airbnb (airbnb_id),
    PRIMARY KEY (plan_id, airbnb_id, ordinal)
);

```

);

CREATE TABLE plan_flight

(

plan_id INT,

flight_id VARCHAR(50),

ordinal INT,

FOREIGN KEY (plan_id) REFERENCES plan (plan_id) ON DELETE CASCADE,

FOREIGN KEY (flight_id) REFERENCES flight (flight_id),

PRIMARY KEY (plan_id, flight_id)

);

Count query on 3 tables:

```
mysql> select count(*) from airbnb;
```

```
+-----+
```

```
| count(*) |
```

```
+-----+
```

```
|    16000 |
```

```
+-----+
```

```
1 row in set (7.76 sec)
```

```
mysql> select count(*) from host;
```

```
+-----+
```

```
| count(*) |
```

```
+-----+
```

```
|    10987 |
```

```
+-----+
```

```
1 row in set (0.02 sec)
```

```
mysql> select count(*) from flight;
```

```
+-----+
```

```
| count(*) |
```

```
+-----+
```

```
|    70221 |
```

```
+-----+
```

```
1 row in set (0.82 sec)
```

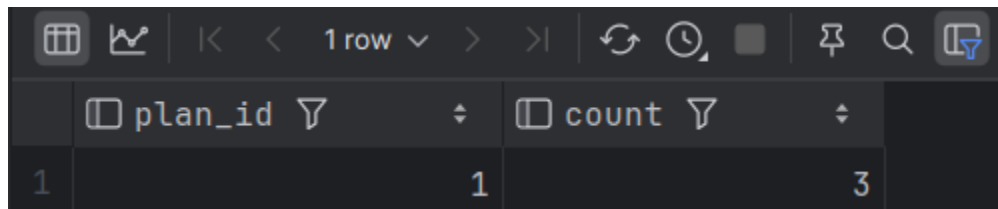
```
mysql> █
```

Advanced Queries:

1. Query to find count of flights/airbnbs for every plan for a given user

```
select p.plan_id, (select count(*) from plan_airbnb pa where pa.plan_id = p.plan_id) +  
                (select count(*) from plan_flight pf where pf.plan_id = p.plan_id) count  
from user u natural join plan p where  
u.user_id = 1;
```

Output:



	plan_id	count
1	1	3

Explanation:

Each time a user creates a new plan, a unique plan_id is generated. Currently, only one plan has been inserted for user_id=1, which is why the query outputs only one record.

2. Query to find airbnbs close to a given airport, whose host owns at least 3 airbnbs

```
select a.airbnb_id  
from airbnb a  
      natural join host h  
where a.close_to_airport = 'JFK'  
and a.host_id in  
  (select h.host_id  
   from host h  
     natural join airbnb a  
   group by h.host_id  
   having count(*) >= 3)  
LIMIT 15;
```

Output:

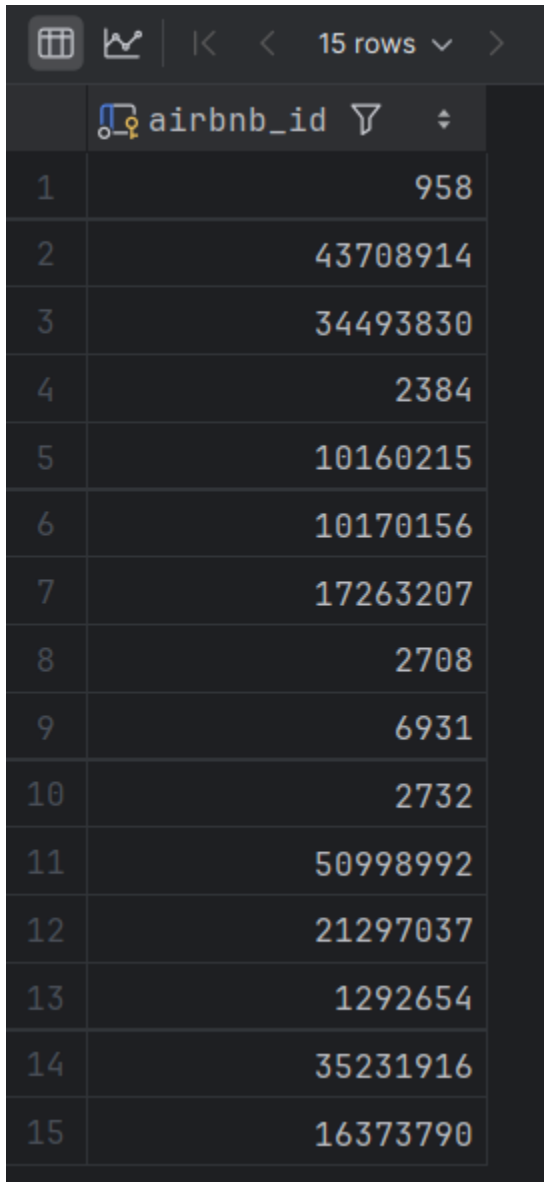
Output		# listing airbnb
		15 rows
	airbnb_id	
1	10160215	
2	10170156	
3	17263207	
4	3779230	
5	9603803	
6	9944808	
7	47803519	
8	51455566	
9	51455950	
10	4916340	
11	13559208	
12	38454938	
13	42476362	
14	31130	
15	20009122	

3. Query to find airbnbs close to a given airport, whose host response rate is greater than or equal to the average host response rate of all airbnbs close to the given airport

```
select a.airbnb_id
from airbnb a
     natural join host h
where a.close_to_airport = 'JFK'
     and h.host_response_rate >=
       (select AVG(h1.host_response_rate)
```

```
FROM airbnb a1
      NATURAL JOIN host h1
      where a1.close_to_airport = 'JFK')
LIMIT 15;
```

Output:



The image shows a screenshot of a database interface displaying the results of a query. The interface includes a toolbar at the top with icons for a table, a line graph, and navigation arrows. A dropdown menu indicates '15 rows'. The query results are shown in a table with two columns: a row number and the 'airbnb_id' value. The 'airbnb_id' column has a filter icon and a dropdown arrow. The table contains 15 rows of data.

	airbnb_id
1	958
2	43708914
3	34493830
4	2384
5	10160215
6	10170156
7	17263207
8	2708
9	6931
10	2732
11	50998992
12	21297037
13	1292654
14	35231916
15	16373790

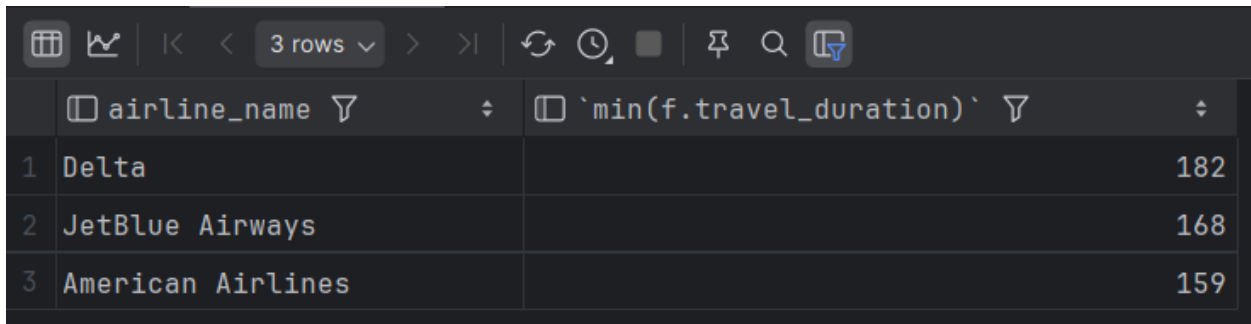
4. Query to find the quickest flight's duration for each airline from a specific starting airport to a specific destination airport on a specific date

```

select a.airline_name, min(f.travel_duration)
from flight f
      natural join airline a
where f.flight_date = '2022-07-03'
      and f.starting_airport = 'JFK'
      and f.destination_airport = 'ORD'
group by a.airline_id;

```

Output:



	airline_name	`min(f.travel_duration)`
1	Delta	182
2	JetBlue Airways	168
3	American Airlines	159

Explanation:

The query returns only 3 records because it applies specific filters that narrow down the data to flights on a particular route and date, and it groups the results by airline to show only the minimum travel duration per airline. Since only 3 airlines operate flights on July 3 2022, from JFK to ORD, the query returns only 3 records.

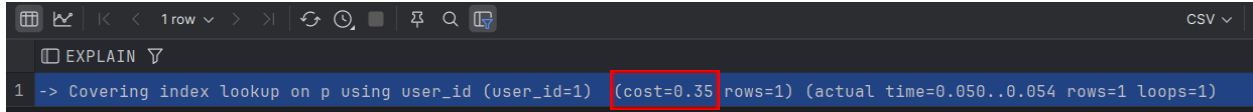
Indexing Advanced Queries:

1.

```

explain analyze select p.plan_id, (select count(*) from plan_airbnb pa where pa.plan_id =
p.plan_id) +
      (select count(*) from plan_flight pf where pf.plan_id = p.plan_id) count
from user u natural join plan p where
u.user_id = 1;

```

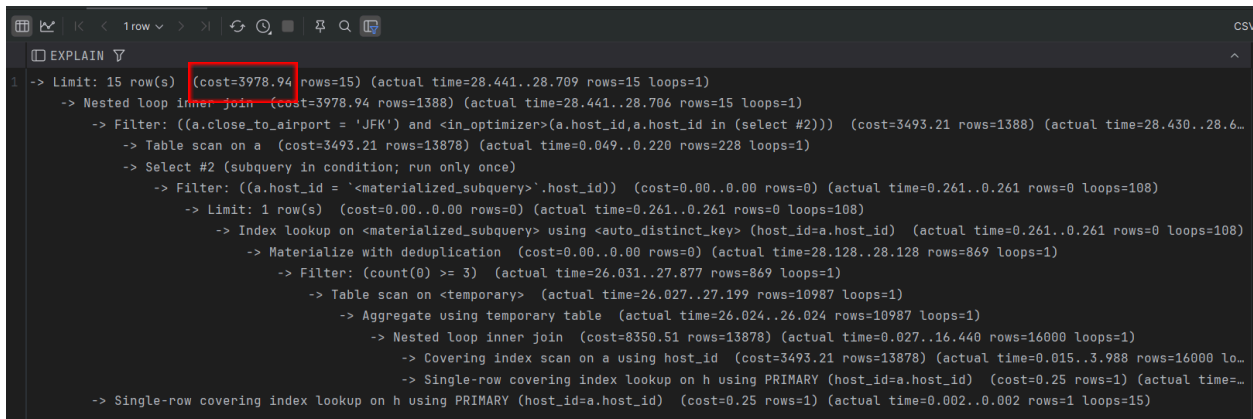
```
1 -> Covering index lookup on p using user_id (user_id=1) (cost=0.35 rows=1) (actual time=0.050..0.054 rows=1 loops=1)
```

We did not do any indexing for this advanced query, because join attributes, where clause attributes and group by attributes are all primary keys that are already indexed.

2.

```
select a.airbnb_id
from airbnb a
      natural join host h
where a.close_to_airport = 'JFK'
and a.host_id in
  (select h.host_id
   from host h
      natural join airbnb a
   group by h.host_id
   having count(*) >= 3)
LIMIT 15;
```

Before indexing:



```
-> Limit: 15 row(s) (cost=3978.94 rows=15) (actual time=28.441..28.709 rows=15 loops=1)
-> Nested loop inner join (cost=3978.94 rows=1388) (actual time=28.441..28.706 rows=15 loops=1)
-> Filter: ((a.close_to_airport = 'JFK') and <in_optimizer>(a.host_id,a.host_id in (select #2))) (cost=3493.21 rows=1388) (actual time=28.430..28.6...
-> Table scan on a (cost=3493.21 rows=13878) (actual time=0.049..0.220 rows=228 loops=1)
-> Select #2 (subquery in condition; run only once)
-> Filter: ((a.host_id = '<materialized_subquery>'.host_id)) (cost=0.00..0.00 rows=0) (actual time=0.261..0.261 rows=0 loops=108)
-> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.261..0.261 rows=0 loops=108)
-> Index lookup on <materialized_subquery> using <auto_distinct_key> (host_id=a.host_id) (actual time=0.261..0.261 rows=0 loops=108)
-> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=28.128..28.128 rows=869 loops=1)
-> Filter: (count(0) >= 3) (actual time=26.031..27.877 rows=869 loops=1)
-> Table scan on <temporary> (actual time=26.027..27.199 rows=10987 loops=1)
-> Aggregate using temporary table (actual time=26.024..26.024 rows=10987 loops=1)
-> Nested loop inner join (cost=8350.51 rows=13878) (actual time=0.027..16.440 rows=16000 loops=1)
-> Covering index scan on a using host_id (cost=3493.21 rows=13878) (actual time=0.015..3.988 rows=16000 lo...
-> Single-row covering index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.25 rows=1) (actual time=...
-> Single-row covering index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
```

After indexing only airbnb(close_to_airport):

```

EXPLAIN
--> Limit: 15 row(s) (cost=3201.85 rows=15) (actual time=52.586..57.506 rows=15 loops=1)
--> Nested loop inner join (cost=3201.85 rows=4246) (actual time=52.585..57.503 rows=15 loops=1)
--> Filter: <in_optimizer>(a.host_id,a.host_id in (select #2)) (cost=1672.12 rows=4246) (actual time=52.569..57.459 rows=15 loops=1)
--> Index lookup on a using airbnb_cta_idx (close_to_airport='JFK') (cost=1672.12 rows=4246) (actual time=0.222..5.006 rows=111 loops=1)
--> Select #2 (subquery in condition; run only once)
--> Filter: ((a.host_id = <materialized_subquery>'.host_id)) (cost=0.00..0.00 rows=0) (actual time=0.484..0.484 rows=0 loops=108)
--> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.483..0.483 rows=0 loops=108)
--> Index lookup on <materialized_subquery> using <auto_distinct_key> (host_id=a.host_id) (actual time=0.483..0.483 rows=0 loops=108)
--> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=52.145..52.145 rows=869 loops=1)
--> Filter: (count(0) >= 3) (actual time=50.174..51.987 rows=869 loops=1)
--> Table scan on <temporary> (actual time=50.170..51.330 rows=10987 loops=1)
--> Aggregate using temporary table (actual time=50.167..50.167 rows=10987 loops=1)
--> Nested loop inner join (cost=8833.65 rows=13878) (actual time=0.039..44.241 rows=16000 loops=1)
--> Covering index scan on a using host_id (cost=3833.77 rows=13878) (actual time=0.028..4.142 rows=16000 loops=1)
--> Single-row covering index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=1)
--> Single-row covering index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=15)

```

After indexing on airbnb(close_to_airport, host_id):

```

EXPLAIN
--> Limit: 15 row(s) (cost=1966.19 rows=15) (actual time=35.862..35.962 rows=15 loops=1)
--> Nested loop inner join (cost=1966.19 rows=4246) (actual time=35.861..35.959 rows=15 loops=1)
--> Filter: <in_optimizer>(a.host_id,a.host_id in (select #2)) (cost=436.46 rows=4246) (actual time=35.842..35.931 rows=15 loops=1)
--> Covering index lookup on a using airbnb_combined_idx (close_to_airport='JFK') (cost=436.46 rows=4246) (actual time=0.035..0.077 rows=73 loops=1)
--> Select #2 (subquery in condition; run only once)
--> Filter: ((a.host_id = <materialized_subquery>'.host_id)) (cost=0.00..0.00 rows=0) (actual time=0.605..0.605 rows=0 loops=59)
--> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.604..0.604 rows=0 loops=59)
--> Index lookup on <materialized_subquery> using <auto_distinct_key> (host_id=a.host_id) (actual time=0.604..0.604 rows=0 loops=59)
--> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=35.622..35.622 rows=869 loops=1)
--> Filter: (count(0) >= 3) (actual time=33.633..35.457 rows=869 loops=1)
--> Table scan on <temporary> (actual time=33.628..34.838 rows=10987 loops=1)
--> Aggregate using temporary table (actual time=33.576..33.576 rows=10987 loops=1)
--> Nested loop inner join (cost=8724.16 rows=13878) (actual time=0.035..28.101 rows=16000 loops=1)
--> Covering index scan on a using host_id (cost=3724.27 rows=13878) (actual time=0.024..3.995 rows=16000 loops=1)
--> Single-row covering index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=1)
--> Single-row covering index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=15)

```

Explanation:

We examined the where clause attributes, join attributes and group by attributes for this query. The attributes being used are airbnb.close_to_airport and airbnb.host_id. We tried indexing close_to_airport and (close_to_airport, host_id). We omitted trying to index host_id alone, as this is already a foreign key and is indexed. Finally, we chose (close_to_airport, host_id) as the index, because this has lesser cost compared to the former as indicated by the boxes.

3.

```

explain analyze
select a.airbnb_id
from airbnb a
      natural join host h
where a.close_to_airport = 'JFK'
and h.host_response_rate >=
      (select AVG(h1.host_response_rate)
       FROM airbnb a1

```

NATURAL JOIN host h1
where a1.close_to_airport = 'JFK')
LIMIT 15;

Before indexing:

```
EXPLAIN
--> Limit: 15 row(s) (cost=4291.91 rows=15) (actual time=7173.700..7173.816 rows=15 loops=1)
--> Nested loop inner join (cost=4291.91 rows=463) (actual time=7173.695..7173.810 rows=15 loops=1)
--> Filter: (a.close_to_airport = 'JFK') (cost=3806.18 rows=1388) (actual time=0.106..0.176 rows=20 loops=1)
--> Table scan on a (cost=3806.18 rows=13878) (actual time=0.097..0.150 rows=48 loops=1)
--> Filter: (h.host_response_rate >= (select #2)) (cost=0.25 rows=0.3) (actual time=358.655..358.655 rows=1 loops=20)
--> Single-row index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.25 rows=1) (actual time=0.005..0.006 rows=1 loops=20)
--> Select #2 (subquery in condition; run only once)
--> Aggregate: avg(h1.host_response_rate) (cost=4430.69 rows=1) (actual time=7172.280..7172.282 rows=1 loops=1)
--> Nested loop inner join (cost=4291.91 rows=1388) (actual time=0.062..7167.504 rows=4246 loops=1)
--> Filter: (a1.close_to_airport = 'JFK') (cost=3806.18 rows=1388) (actual time=0.057..6969.058 rows=4246 loops=1)
--> Table scan on a1 (cost=3806.18 rows=13878) (actual time=0.051..6960.563 rows=16000 loops=1)
--> Single-row index lookup on h1 using PRIMARY (host_id=a1.host_id) (cost=0.25 rows=1) (actual time=0.046..0.046 rows=1 loops=4246)
```

After indexing only airbnb(close_to_airport):

```
EXPLAIN
--> Limit: 15 row(s) (cost=3651.45 rows=15) (actual time=6244.069..6244.101 rows=15 loops=1)
--> Nested loop inner join (cost=3651.45 rows=1415) (actual time=6244.068..6244.099 rows=15 loops=1)
--> Index lookup on a using temp (close_to_airport='JFK') (cost=1598.25 rows=4246) (actual time=0.114..0.122 rows=20 loops=1)
--> Filter: (h.host_response_rate >= (select #2)) (cost=0.38 rows=0.3) (actual time=312.199..312.199 rows=1 loops=20)
--> Single-row index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.38 rows=1) (actual time=0.002..0.002 rows=1 loops=20)
--> Select #2 (subquery in condition; run only once)
--> Aggregate: avg(h1.host_response_rate) (cost=4076.05 rows=1) (actual time=6243.898..6243.898 rows=1 loops=1)
--> Nested loop inner join (cost=3651.45 rows=4246) (actual time=0.050..6241.945 rows=4246 loops=1)
--> Index lookup on a1 using temp (close_to_airport='JFK') (cost=1598.25 rows=4246) (actual time=0.046..6147.373 rows=4246 loops=1)
--> Single-row index lookup on h1 using PRIMARY (host_id=a1.host_id) (cost=0.38 rows=1) (actual time=0.022..0.022 rows=1 loops=4246)
```

After indexing only host(host_response_rate):

```
EXPLAIN
--> Limit: 15 row(s) (cost=4237.59 rows=15) (actual time=0.269..0.339 rows=15 loops=1)
--> Nested loop inner join (cost=4237.59 rows=1055) (actual time=0.267..0.337 rows=15 loops=1)
--> Filter: (a.close_to_airport = 'JFK') (cost=3751.86 rows=1388) (actual time=0.232..0.275 rows=20 loops=1)
--> Table scan on a (cost=3751.86 rows=13878) (actual time=0.221..0.261 rows=48 loops=1)
--> Filter: (h.host_response_rate >= (select #2)) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=20)
--> Single-row index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=20)
--> Select #2 (subquery in condition; run only once)
--> Aggregate: avg(h1.host_response_rate) (cost=4376.37 rows=1) (actual time=7382.726..7382.726 rows=1 loops=1)
--> Nested loop inner join (cost=4237.59 rows=1388) (actual time=0.059..7378.620 rows=4246 loops=1)
--> Filter: (a1.close_to_airport = 'JFK') (cost=3751.86 rows=1388) (actual time=0.047..7355.800 rows=4246 loops=1)
--> Table scan on a1 (cost=3751.86 rows=13878) (actual time=0.041..7346.904 rows=16000 loops=1)
--> Single-row index lookup on h1 using PRIMARY (host_id=a1.host_id) (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=4246)
```

After indexing airbnb(close_to_airport, host_id):

```

EXPLAIN
--> Limit: 15 row(s) (cost=1922.56 rows=15) (actual time=6.734..6.786 rows=15 loops=1)
--> Nested loop inner join (cost=1922.56 rows=1415) (actual time=6.732..6.784 rows=15 loops=1)
-->   Covering index lookup on a using airbnb_combined_idx (close_to_airport='JFK') (cost=436.46 rows=4246) (actual time=0.202..0.219 rows=28 loops=1)
-->   Filter: (h.host_response_rate >= (select #2)) (cost=0.25 rows=0.3) (actual time=0.234..0.234 rows=1 loops=28)
-->     Single-row index lookup on h using PRIMARY (host_id=a.host_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=28)
-->     Select #2 (subquery in condition; run only once)
-->       Aggregate: avg(h1.host_response_rate) (cost=2347.16 rows=1) (actual time=6.492..6.492 rows=1 loops=1)
-->       Nested loop inner join (cost=1922.56 rows=4246) (actual time=0.066..6.000 rows=4246 loops=1)
-->         Covering index lookup on a1 using airbnb_combined_idx (close_to_airport='JFK') (cost=436.46 rows=4246) (actual time=0.062..1.545 rows=28 loops=1)
-->         Single-row index lookup on h1 using PRIMARY (host_id=a1.host_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4246)

```

Explanation:

We examined the where clause attributes, join attributes and group by attributes for this query. The attributes being used are airbnb.close_to_airport, airbnb.host_id and host.host_response_rate. After trying to index various combinations of these attributes, we settled for airbnb(close_to_airport, host_id) because this results in the lowest cost (also this works best for the previous query too).

4.

```

explain analyze
select a.airline_name, min(f.travel_duration)
from flight f
      natural join airline a
where f.flight_date = '2022-07-03'
      and f.starting_airport = 'JFK'
      and f.destination_airport = 'ORD'
group by a.airline_id;

```

Before indexing:

```

EXPLAIN
--> Table scan on <temporary> (actual time=1155.159..1155.159 rows=3 loops=1)
--> Aggregate using temporary table (actual time=1155.157..1155.157 rows=3 loops=1)
-->   Inner hash join (a.airline_id = f.airline_id) (cost=7674.65 rows=69) (actual time=1154.920..1154.929 rows=4 loops=1)
-->     Table scan on a (cost=0.02 rows=8) (actual time=31.562..31.568 rows=7 loops=1)
-->     Hash
-->       Filter: ((f.destination_airport = 'ORD') and (f.starting_airport = 'JFK') and (f.flight_date = DATE'2022-07-03')) (cost=7618.60 rows=69) (actual time=58.461..1109.991 rows=70221 loops=1)
-->         Table scan on f (cost=7618.60 rows=68796) (actual time=58.461..1109.991 rows=70221 loops=1)

```

After indexing only flight(starting_airport, destination_airport):

```

EXPLAIN
-> Table scan on <temporary> (actual time=564.469..564.470 rows=3 loops=1)
-> Aggregate using temporary table (actual time=564.467..564.467 rows=3 loops=1)
-> Inner hash join (a.airline_id = f.airline_id) (cost=680.60 rows=84) (actual time=563.509..563.542 rows=4 loops=1)
-> Table scan on a (cost=0.01 rows=8) (actual time=40.357..40.373 rows=7 loops=1)
-> Hash
-> Filter: (f.flight_date = DATE'2022-07-03') (cost=612.54 rows=84) (actual time=249.626..523.020 rows=4 loops=1)
-> Index lookup on f using temp (starting_airport='JFK', destination_airport='ORD') (cost=612.54 rows=838) (actual time=79.982..522.918 rows=838)

```

After indexing only flight(flight_date):

```

Output # listing flight with filters
EXPLAIN
1 -> Table scan on <temporary> (actual time=17.293..17.293 rows=3 loops=1)
-> Aggregate using temporary table (actual time=17.290..17.290 rows=3 loops=1)
-> Nested loop inner join (cost=151.21 rows=4) (actual time=12.433..17.225 rows=4 loops=1)
-> Filter: ((f.destination_airport = 'ORD') and (f.starting_airport = 'JFK')) (cost=149.91 rows=4) (actual time=12.411..17.162 rows=4 loops=1)
-> Index lookup on f using temp (flight_date=DATE'2022-07-03') (cost=149.91 rows=371) (actual time=12.376..17.097 rows=371 loops=1)
-> Single-row index lookup on a using PRIMARY (airline_id=f.airline_id) (cost=0.28 rows=1) (actual time=0.014..0.014 rows=1 loops=4)

```

After indexing flight(starting_airport, destination_airport, flight_date):

```

EXPLAIN
1 -> Table (actual time=0.190..0.191 rows=3 loops=1)
-> Aggregate using temporary table (actual time=0.189..0.189 rows=3 loops=1)
-> Nested loop inner join (cost=3.90 rows=4) (actual time=0.149..0.162 rows=4 loops=1)
-> Index lookup on f using flight_combined_idx (starting_airport='JFK', destination_airport='ORD', flight_date=DATE'2022-07-03') (cost=2.50 rows=4) (actual time=0.149..0.162 rows=4 loops=1)
-> Single-row index lookup on a using PRIMARY (airline_id=f.airline_id) (cost=0.28 rows=1) (actual time=0.003..0.003 rows=1 loops=4)

```

Explanation:

The attributes being used in the where clause are flight.starting_airport, flight.destination_airport and flight.flight_date. We tried indexing various combinations of these and indexing flight(starting_airport, destination_airport, flight_date) gave us the best result by far. Also, this works well with our use case because other flight filters that we plan to implement for our application will also contain flight.starting_airport, flight.destination_airport and flight.flight_date in the where clause.