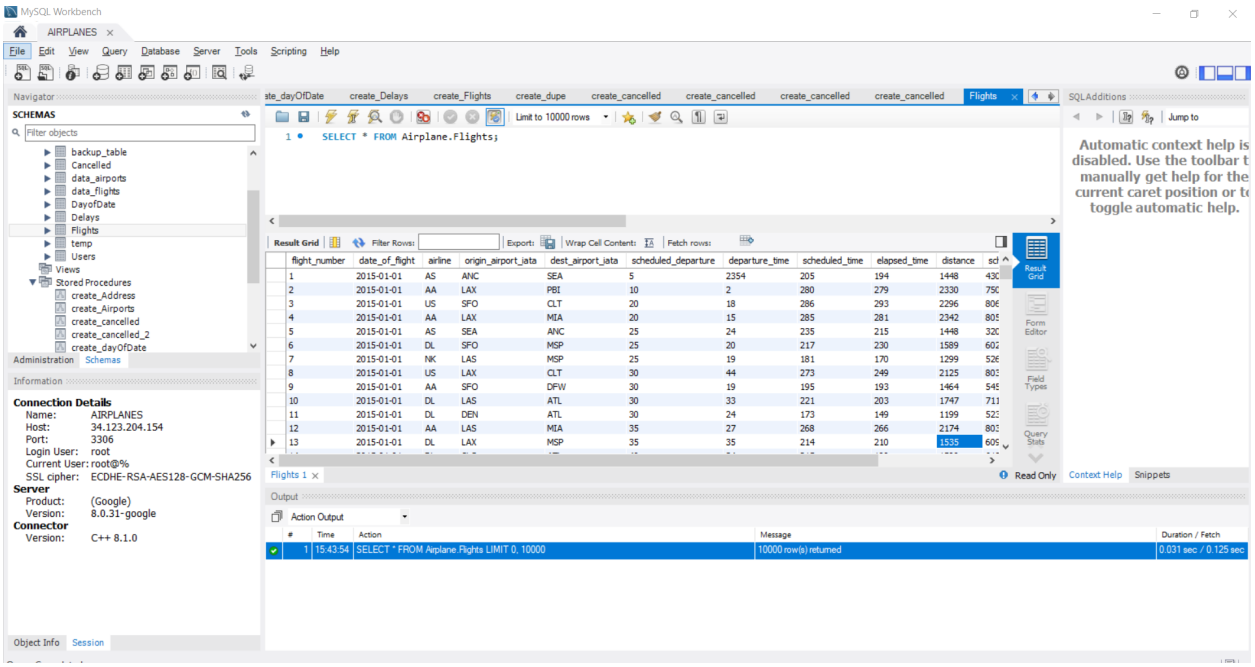
Group 112: Orange
NetIDs: zxchoo2, pnp4, hansenp2

Part 1: Database Implementation

1. Implemented the database tables locally or on GCP



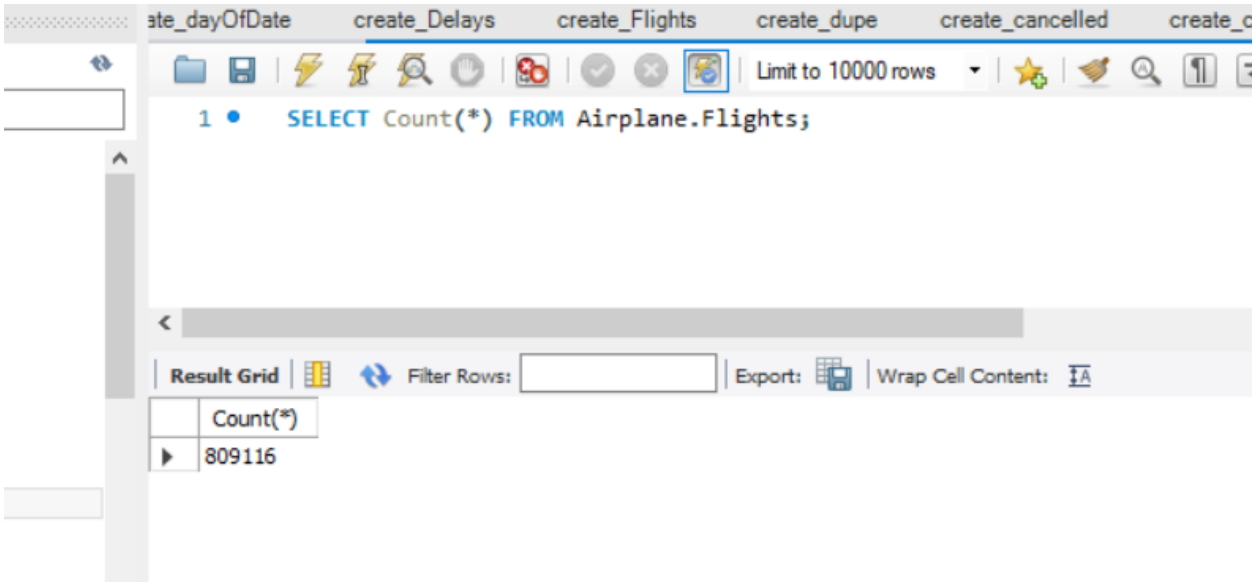
2. Provide the Data Definition Language (DDL) commands you all used to create each of these tables in the database.

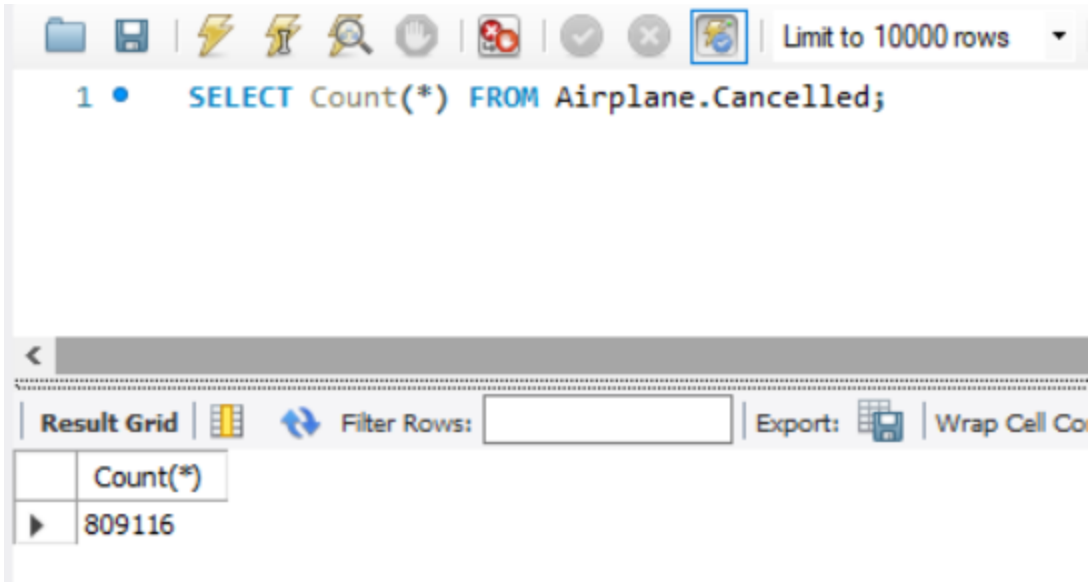
```
CREATE TABLE Users (
    user_id INT PRIMARY KEY,
    username VARCHAR(255),
    password VARCHAR(255),
    email VARCHAR(255)
);


CREATE TABLE Searches (
    search_id INT PRIMARY KEY,
    date_searched DATETIME,
```

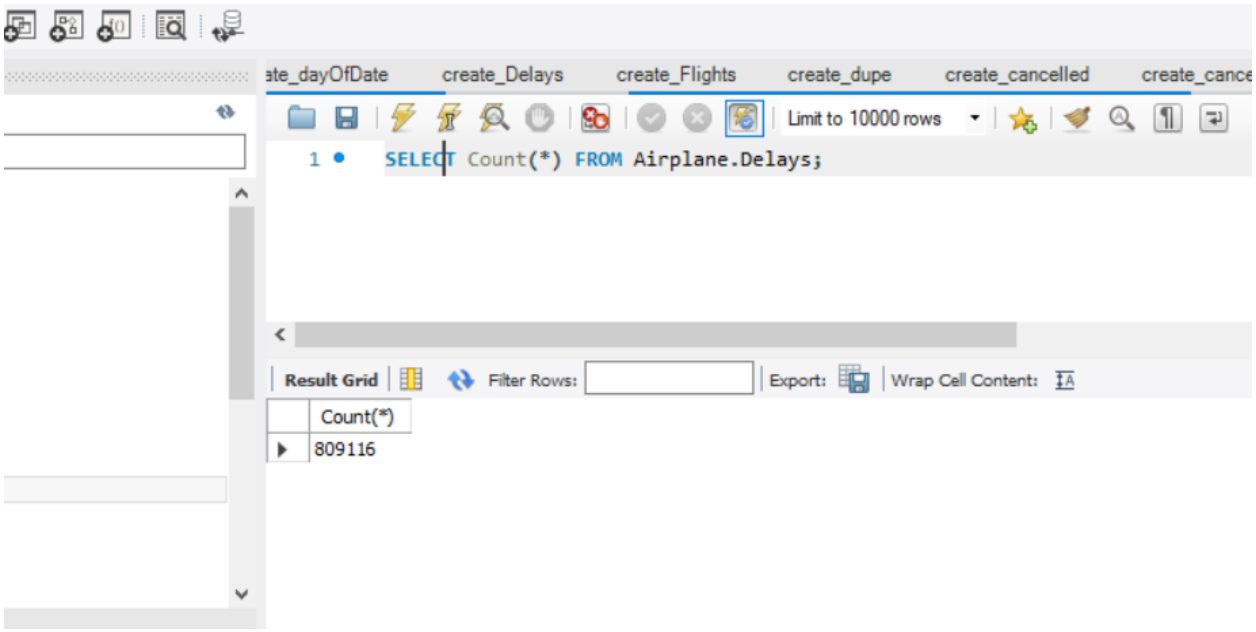
```
    search_start_date DATE,
    search_end_date DATE,
    search_airline VARCHAR(255),
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (search_airline) REFERENCES Airlines(airline_iata_code)
);

CREATE TABLE Flights (
    flight_number INT PRIMARY KEY,
    date_of_flight DATE,
    airline VARCHAR(255),
    origin_airport_iata VARCHAR(255),
    dest_airport_iata VARCHAR(255),
    scheduled_departure INT,
    departure_time INT,
    scheduled_time INT,
    elapsed_time REAL,
    distance REAL,
    scheduled_arrival INT,
    arrival_time INT,
    FOREIGN KEY (origin_airport_iata) REFERENCES Airports(airport_iata_code),
    FOREIGN KEY (dest_airport_iata) REFERENCES Airports(airport_iata_code),
    FOREIGN KEY (date_of_flight) REFERENCES DayofDate(date_of_flight)
);

CREATE TABLE DayofDate (
    date_of_flight DATE PRIMARY KEY,
    day_of_week VARCHAR(255)
);

CREATE TABLE Cancelled (
    flight_number INT PRIMARY KEY,
    canceled BOOLEAN,
    cancellation_reason VARCHAR(255),
    FOREIGN KEY (flight_number) REFERENCES Flights(flight_number)
);

CREATE TABLE Delays (
    flight_number INT PRIMARY KEY,
```

```
    departure_delay INT,
    arrival_delay INT,
    FOREIGN KEY (flight_number) REFERENCES Flights(flight_number)
);

CREATE TABLE Airports (
    airport_iata_code VARCHAR(255) PRIMARY KEY,
    airport VARCHAR(255),
);


CREATE TABLE Address (
    airport_code VARCHAR(255) PRIMARY KEY,
    city VARCHAR(255),
    state VARCHAR(255),
    country VARCHAR(255),
    latitude REAL,
    longitude REAL,
    FOREIGN KEY (airport_code) REFERENCES Airports(airport_iata_code)
);


CREATE TABLE Airlines (
    airline_iata_code VARCHAR(255) PRIMARY KEY,
    airline_name VARCHAR(255)
);
```

3. Insert data into these tables. You should insert at least 1000 rows each in three of the tables. Try to use real data, but if you cannot find a good dataset for a particular table, you may use auto-generated data


ate_dayOfDate
create_Delays
create_Flights
create_dupe
create_cancelled
Limit to 10000 rows
1 SELECT Count(*) FROM Airplane.Flights;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Count(*)
809116





To note for the other tables:

- Airlines matches IATA_Code to airline name, which has count < 1000
- Airports matches airport_IATA_Code to airport, which has count < 1000
- DayofDate matches date_of_flight to day of week, which has count < 1000

Advanced Queries (show top 15 results)

Advanced Query 1

This query gives aggregate information about the airlines between the dates the user selects. These dates are represented below as start_date and end_date, and will be replaced by the frontend of the website when it gets the actual dates from the user. When generating the screenshot of the top 15 rows, we used 2015-01-27 as the start_date and 2015-02-01 as the end_date. The information will be displayed from highest to lowest delay number by default. Right now the information shown to the user is the number of delays, average severity of a delay, and number of cancellations. More information could be added in the future. There are only 14 airlines, so the query returns less than 15 entries.

```
SELECT *
FROM (
        SELECT airline, AVG(arrival_delay) AS delay_severity, COUNT(flight_number) AS
        delay_number
        FROM Delays NATURAL JOIN Flights
        WHERE date_of_flight >= start_date AND date_of_flight <= end_date
        GROUP BY airline
) AS DelayData NATURAL JOIN (
        SELECT airline, SUM(Cancelled) AS cancellation_number
        FROM Cancelled NATURAL JOIN Flights
        WHERE date_of_flight >= start_date AND date_of_flight <= end_date
        GROUP BY airline
) AS CancelData
ORDER BY delay_number DESC
LIMIT 15;
```

airline	delay_severity	delay_number	cancellation_number
WN	-1.6697	18927	909
DL	-4.6599	12060	458
EV	-0.3635	9068	735
OO	3.6939	8911	286
AA	-1.0662	[illegible]	597
UA	-1.1961	[illegible]	726
US	4.9669	6350	618
MQ	2.7969	5707	686
B6	3.3314	4083	710
AS	1.6416	2536	31
NK	4.4483	1684	60
F9	12.6900	1258	40
HA	2.5529	1239	10
VX	-3.2769	892	66

Advanced Query 2

This second query shows the aggregate information about airlines on a specific day of the week. This is represented by week_day in the query, which will be replaced by what the user selects on the website. To test the query and generate the top 15 rows for the query, we used the value 'Friday' for week_day. The query currently sorts things by the number of delays by default, but could potentially be changed by the user. Right now we show the number of delays, average time of each delay, and the number of cancellations. More information could be added. There are only 14 airlines, so the query returns less than 15 entries.

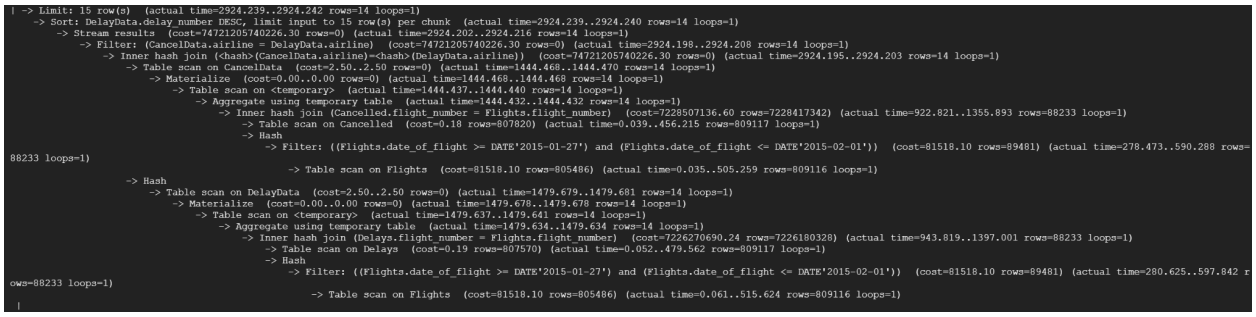
```
SELECT *
FROM (
        SELECT airline, AVG(arrival_delay) AS delay_severity, COUNT(flight_number) AS
        delay_number
        FROM Delays NATURAL JOIN Flights NATURAL JOIN DayofDate
        WHERE day_of_week = week_day
        GROUP BY airline
) AS DelayData NATURAL JOIN (
        SELECT airline, SUM(Cancelled) AS cancellation_number
        FROM Cancelled NATURAL JOIN Flights NATURAL JOIN DayofDate
        WHERE day_of_week = week_day
        GROUP BY airline
) AS CancelData
ORDER BY delay_number DESC
LIMIT 15;
```

airline	delay_severity	delay_number	cancellation_number
WN	2.3473	27201	311
DL	0.5489	18485	33
EV	5.7957	14102	170
OO	8.0546	13253	383
AA	4.8002	11707	86
UA	7.6203	11039	68
US	3.6486	9339	97
MQ	13.7975	8108	363
B6	6.5977	5807	22
AS	2.8898	3603	29
NK	9.1865	2311	13
F9	13.8379	1844	3
HA	9.2829	1778	1
VX	13.6102	1347	12

Part 2: Indexing Analysis

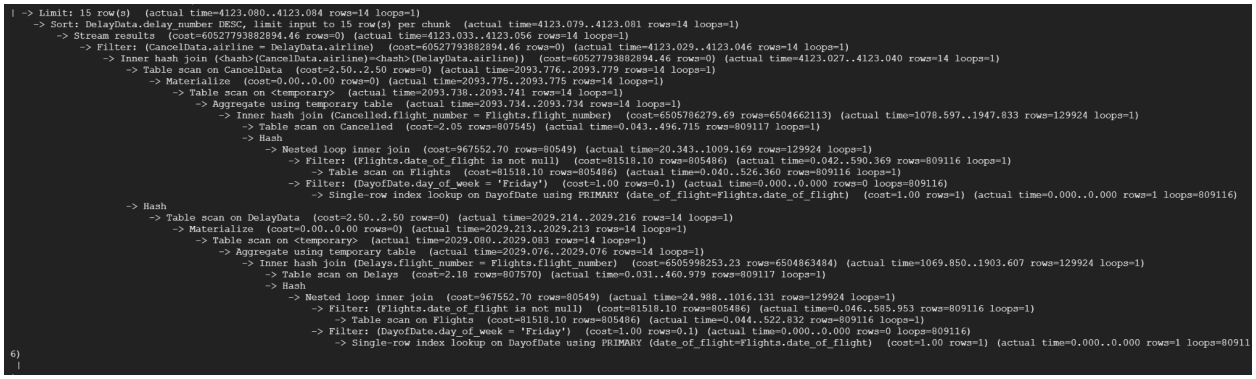
Default Index Design

First we needed to use EXPLAIN ANALYZE on both of our advanced queries. This is what the results were for advanced query 1:



The results come back as a tree with two main branches. This is because the query joins Flights with Delays and Flights with Cancelled separately, collects aggregate data about them, and then joins the results. The highest cost in both amount and placement up the tree is for Stream Results, with a cost 74721205740226.30.

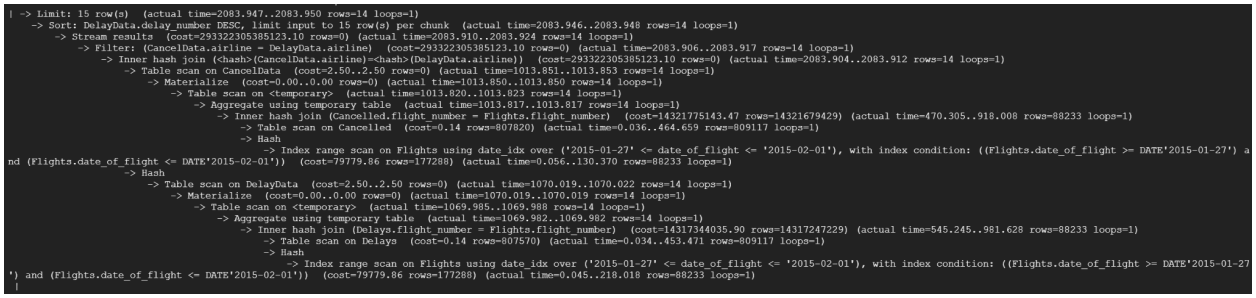
Here are the results for query 2:



The query is still a tree with two branches, but the overall result is bigger, probably due to the extra join with the DayofDate table required to get the Day of the week for each flight. The cost for Stream Result is actually slightly lower, at 60527793882894.46.

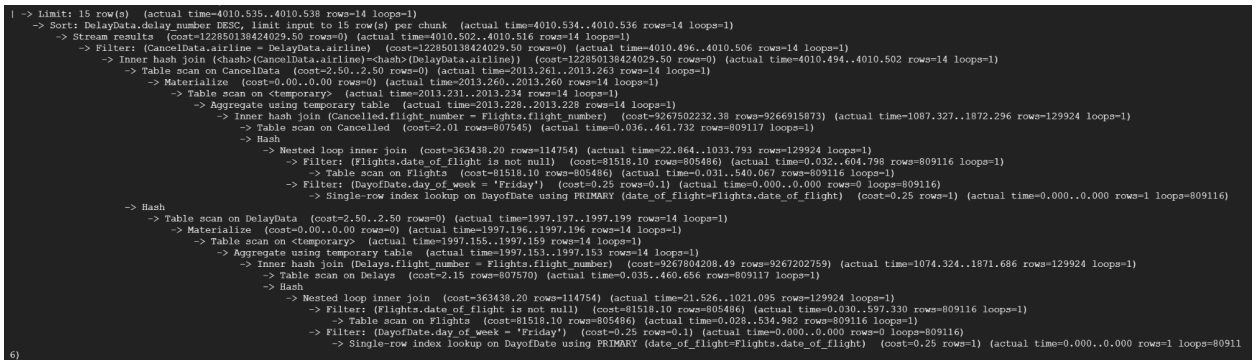
Index Design 1

The idea behind our first design was that we would create an index over Flights.date_of_flight called date_idx. The hope was that since we were only looking at flights on certain dates, an index on the Flights.date_of_flight would allow the system to skip just to the tuples that were within the date range. After implementing the index, the EXPLAIN ANALYZE results for the first advanced query were:



Looking at this result, we can see that date_idx was used at the bottom of the two legs of the tree, replacing two Table Scan on Flights with Index range scans on Flights. The table scans cost 81510.1 each, while the new index scans cost 79779.86 each. This actually leads to a higher cost further up the tree. The highest up cost in the tree is the Stream Results. With the default design, the cost of Stream Results with the original database design was 74721205740226.30. The cost of Stream Results using date_idx is 293322305385123.10.

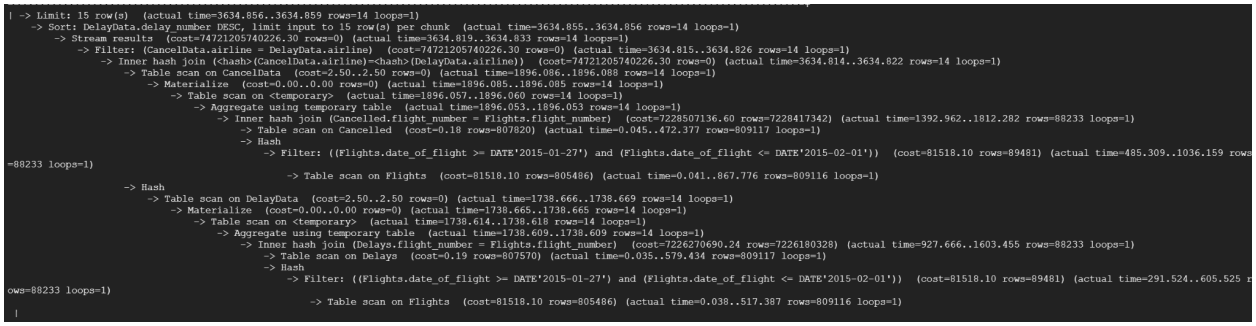
The second advanced query looks at flights on certain days of the week. Because of that we switched out date_idx for day_idx, an index on DayofDate.day_of_week. After implementing this new index, the EXPLAIN ANALYZE results for the second advanced query were:



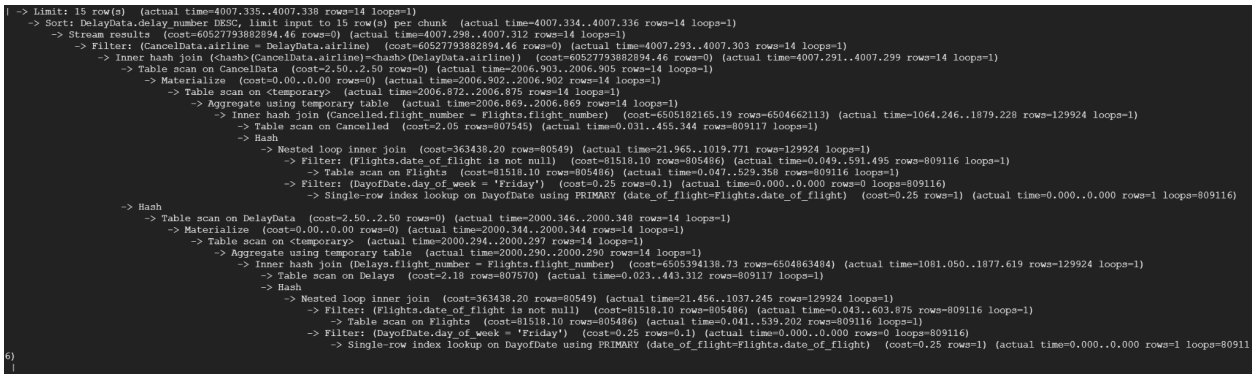
Interestingly the results never reference day_idx, which would make one think that day_idx was never used. However, it had a noticeable effect on performance, with the cost of Stream Results now at 122850138424029.50, which was worse than the original cost. We are unsure why this design makes performance worse.

Index Design 2

The idea behind our second design was that we would create an index over Flights.airline called airline_idx. The hope was that it would improve the efficiency of grouping and aggregating the information. Here is the EXPLAIN ANALYZE results of the first query with airline_idx:

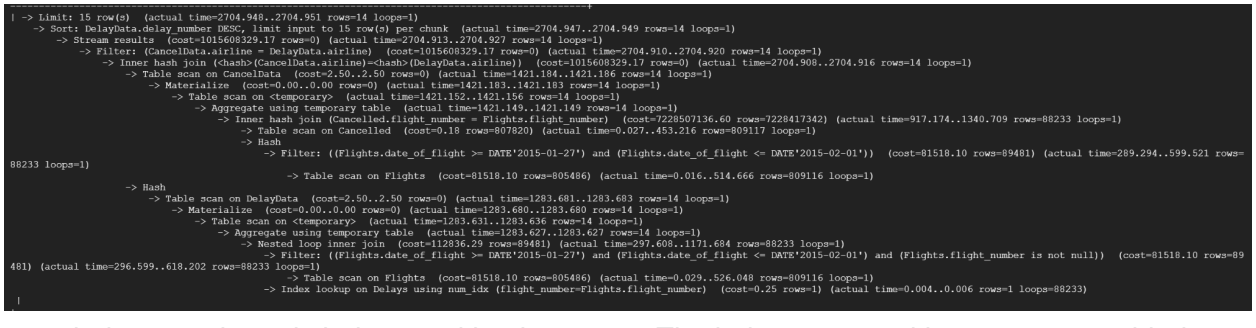


Unfortunately, it seems like the database did not use airline_idx for anything, and the costs are the exact same as with the default design. This also held true for the other advanced query:



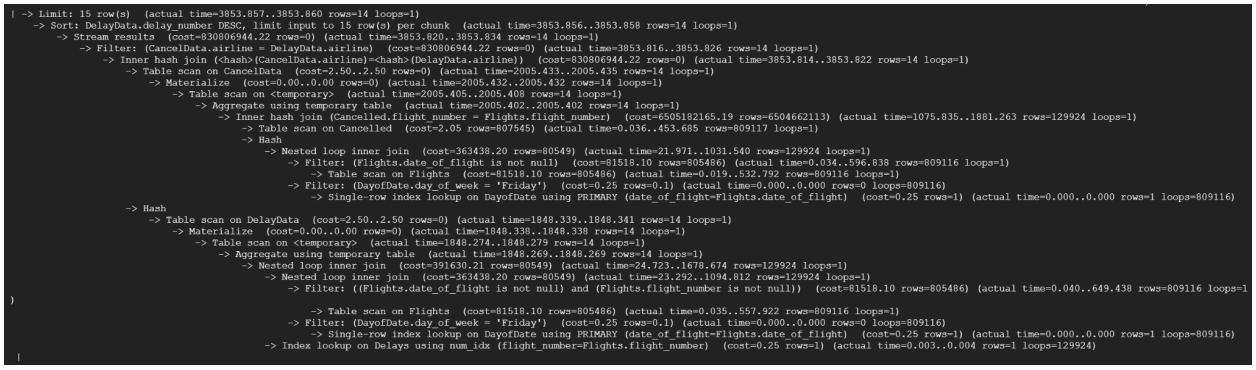
Index Design 3

For our final design, we decided to create indexes on the attributes that we were aggregating. These attributes are Delays.flight_number, Delays.arrival_delay, and Cancelled.cancelled, with indexes num_idx, delay_idx, and cancelled_idx respectively. The hope was that indexing these functions would help speed up the aggregation process in some way. Here is the EXPLAIN ANALYZE results of using these indexes on advanced query 1:



num_index was the only index used by the query. The index was used in a new step added at the end of one of the tree branches, titled Index Lookup on Delays. The cost of this new step was 0.25. Rather than help with the aggregation process as expected, we believe the index helped with the process of joining Flights with Delays, as flight_number was what they were joining on. This apparently resulted in a net time save, as the highest cost up the tree, Stream Results, dropped from 74695768970595.86 to 1015608329.17.

The results for the second advanced query were:

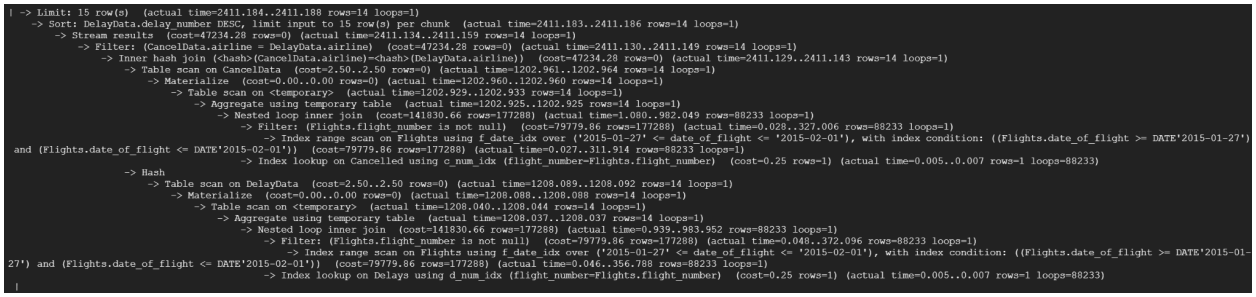


```
| -> Limit: 15 row(s)  (actual time=3853.857..3853.860 rows=14 loops=1)
    -> Sort: DelayData.delay_number DESC, limit input to 15 row(s) per chunk  (actual time=3853.856..3853.858 rows=14 loops=1)
        -> Stream results  (cost=830806944.22 rows=0) (actual time=3853.820..3853.834 rows=14 loops=1)
            -> Filter: (CancelData.airline = DelayData.airline)  (cost=830806944.22 rows=0) (actual time=3853.816..3853.826 rows=14 loops=1)
                -> Inner hash join (<hash>(CancelData.airline)=<hash>(DelayData.airline))  (cost=830806944.22 rows=0) (actual time=3853.814..3853.822 rows=14 loops=1)
                    -> Table scan on CancelData  (cost=2.50..2.50 rows=0) (actual time=2005.433..2005.435 rows=14 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2005.432..2005.432 rows=14 loops=1)
                            -> Table scan on <temporary>  (actual time=2005.405..2005.408 rows=14 loops=1)
                                -> Aggregate using temporary table  (actual time=2005.402..2005.402 rows=14 loops=1)
                                    -> Inner hash join (Cancelled.flight_number = Flights.flight_number)  (cost=6505182165.19 rows=6504662113) (actual time=1075.835..1881.263 rows=129924 loops=1)
                                        -> Table scan on Cancelled  (cost=2.05 rows=807545) (actual time=0.036..453.685 rows=809117 loops=1)
                                        -> Hash
                                            -> Nested loop inner join  (cost=363438.20 rows=80549) (actual time=21.971..1031.540 rows=129924 loops=1)
                                                -> Filter: (Flights.date_of_flight is not null)  (cost=81518.10 rows=805486) (actual time=0.034..596.838 rows=809116 loops=1)
                                                    -> Table scan on Flights  (cost=81518.10 rows=805486) (actual time=0.019..532.792 rows=809116 loops=1)
                                                -> Filter: (DayofDate.day_of_week = 'Friday')  (cost=0.25 rows=0.1) (actual time=0.000..0.000 rows=0 loops=809116)
                                                    -> Single-row index lookup on DayofDate using PRIMARY (date_of_flight=Flights.date_of_flight)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=809116)
                    -> Hash
                        -> Table scan on DelayData  (cost=2.50..2.50 rows=0) (actual time=1848.339..1848.341 rows=14 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1848.338..1848.338 rows=14 loops=1)
                                -> Table scan on <temporary>  (actual time=1848.274..1848.279 rows=14 loops=1)
                                    -> Aggregate using temporary table  (actual time=1848.269..1848.269 rows=14 loops=1)
                                        -> Nested loop inner join  (cost=391630.21 rows=80549) (actual time=24.723..1678.674 rows=129924 loops=1)
                                            -> Nested loop inner join  (cost=363438.20 rows=80549) (actual time=23.292..1094.812 rows=129924 loops=1)
                                                -> Filter: ((Flights.date_of_flight is not null) and (Flights.flight_number is not null))  (cost=81518.10 rows=805486) (actual time=0.040..649.438 rows=809116 loops=1
)
                                                    -> Table scan on Flights  (cost=81518.10 rows=805486) (actual time=0.035..557.922 rows=809116 loops=1)
                                                -> Filter: (DayofDate.day_of_week = 'Friday')  (cost=0.25 rows=0.1) (actual time=0.000..0.000 rows=0 loops=809116)
                                                    -> Single-row index lookup on DayofDate using PRIMARY (date_of_flight=Flights.date_of_flight)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=809116)
                                            -> Index lookup on Delays using num_idx (flight_number=Flights.flight_number)  (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=129924)
 |
```

A very similar result to what happened with the first query also happened here. A new step was added to the end of one of the tree branches, with the name Index Lookup on Delays and with a cost of 0.25. The Stream Results cost dropped to 830806944.22.

Final Index Analysis and Design:

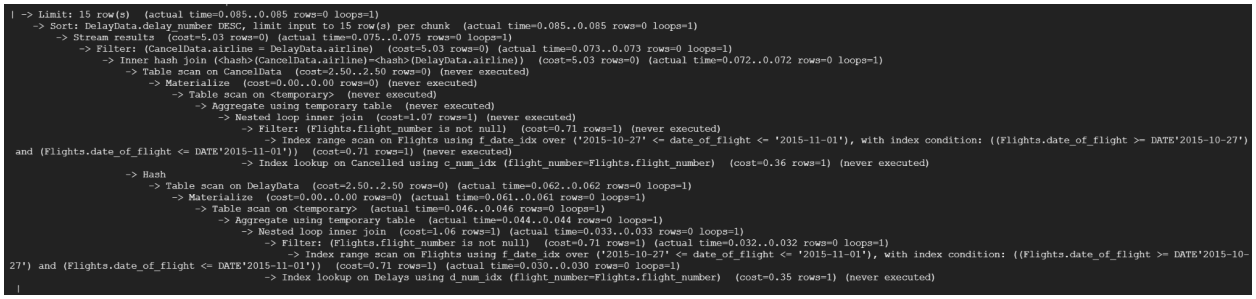
After looking at all 3 indexes, we came to some conclusions. First off, the second design did not do anything, so we did not consider it. It also looked like the indexes were never used to speed up aggregation, so we decided not to try to do that in the final design. We decided to add indexes for all of the points that tables were joined with, to hopefully capitalize on the speed up from design 3. In the end, this created the following indexes: f_date_idx on Flights(date_of_week), d_date_idx on DayofDate(date_of_week), f_num_idx on Flights(flight_number), d_num_idx on Delays(flight_number), and c_num_idx on Cancelled(flights_number). This is the result for out first advanced query:



```
| -> Limit: 15 row(s)  (actual time=2411.184..2411.188 rows=14 loops=1)
    -> Sort: DelayData.delay_number DESC, limit input to 15 row(s) per chunk  (actual time=2411.183..2411.186 rows=14 loops=1)
        -> Stream results  (cost=47234.28 rows=0) (actual time=2411.134..2411.159 rows=14 loops=1)
            -> Filter: (CancelData.airline = DelayData.airline)  (cost=47234.28 rows=0) (actual time=2411.130..2411.149 rows=14 loops=1)
                -> Inner hash join (<hash>(CancelData.airline)=<hash>(DelayData.airline))  (cost=47234.28 rows=0) (actual time=2411.129..2411.143 rows=14 loops=1)
                    -> Table scan on CancelData  (cost=2.50..2.50 rows=0) (actual time=1202.961..1202.964 rows=14 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1202.960..1202.960 rows=14 loops=1)
                            -> Table scan on <temporary>  (actual time=1202.929..1202.933 rows=14 loops=1)
                                -> Aggregate using temporary table  (actual time=1202.925..1202.925 rows=14 loops=1)
                                    -> Nested loop inner join  (cost=141830.66 rows=177288) (actual time=1.080..982.049 rows=88233 loops=1)
                                        -> Filter: (Flights.flight_number is not null)  (cost=79779.86 rows=177288) (actual time=0.028..327.006 rows=88233 loops=1)
                                            -> Index range scan on Flights using f_date_idx over ('2015-01-27' <= date_of_flight <= '2015-02-01'), with index condition: ((Flights.date_of_flight >= DATE'2015-01-27')
 and (Flights.date_of_flight <= DATE'2015-02-01'))  (cost=79779.86 rows=177288) (actual time=0.027..311.914 rows=88233 loops=1)
                                        -> Index lookup on Cancelled using c_num_idx (flight_number=Flights.flight_number)  (cost=0.25 rows=1) (actual time=0.005..0.007 rows=1 loops=88233)
                    -> Hash
                        -> Table scan on DelayData  (cost=2.50..2.50 rows=0) (actual time=1208.089..1208.092 rows=14 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=1208.088..1208.088 rows=14 loops=1)
                                -> Table scan on <temporary>  (actual time=1208.040..1208.044 rows=14 loops=1)
                                    -> Aggregate using temporary table  (actual time=1208.037..1208.037 rows=14 loops=1)
                                        -> Nested loop inner join  (cost=141830.66 rows=177288) (actual time=0.939..983.952 rows=88233 loops=1)
                                            -> Filter: (Flights.flight_number is not null)  (cost=79779.86 rows=177288) (actual time=0.048..372.096 rows=88233 loops=1)
                                                -> Index range scan on Flights using f_date_idx over ('2015-01-27' <= date_of_flight <= '2015-02-01'), with index condition: ((Flights.date_of_flight >= DATE'2015-01-
27') and (Flights.date_of_flight <= DATE'2015-02-01'))  (cost=79779.86 rows=177288) (actual time=0.046..356.788 rows=88233 loops=1)
                                            -> Index lookup on Delays using d_num_idx (flight_number=Flights.flight_number)  (cost=0.25 rows=1) (actual time=0.005..0.007 rows=1 loops=88233)
 |
```

It creates the extra steps Index Lookup on Delays using d_num_idx with a cost of 0.25 and Index Lookup on Cancelled using c_num_idx with a cost of 0.25. It also replaces the table scans on Flights in each branch of the tree with Index Range Scans on Flights using f_date_idx with a cost of 79779.86 each. The cost of Stream Results went down to 47234.28.

The EXPLAIN ANALYZE results of the second advanced query were:



```
| -> Limit: 15 row(s)  (actual time=0.085..0.085 rows=0 loops=1)
    -> Sort: DelayData.delay_number DESC, limit input to 15 row(s) per chunk  (actual time=0.085..0.085 rows=0 loops=1)
        -> Stream results  (cost=5.03 rows=0) (actual time=0.075..0.075 rows=0 loops=1)
            -> Filter: (CancelData.airline = DelayData.airline)  (cost=5.03 rows=0) (actual time=0.073..0.073 rows=0 loops=1)
                -> Inner hash join (<hash>(CancelData.airline)=<hash>(DelayData.airline))  (cost=5.03 rows=0) (actual time=0.072..0.072 rows=0 loops=1)
                    -> Table scan on CancelData  (cost=2.50..2.50 rows=0) (never executed)
                        -> Materialize  (cost=0.00..0.00 rows=0) (never executed)
                            -> Table scan on <temporary>  (never executed)
                                -> Aggregate using temporary table  (never executed)
                                    -> Nested loop inner join  (cost=1.07 rows=1) (never executed)
                                        -> Filter: (Flights.flight_number is not null)  (cost=0.71 rows=1) (never executed)
                                            -> Index range scan on Flights using f_date_idx over ('2015-10-27' <= date_of_flight <= '2015-11-01'), with index condition: ((Flights.date_of_flight >= DATE'2015-10-27')
 and (Flights.date_of_flight <= DATE'2015-11-01'))  (cost=0.71 rows=1) (never executed)
                                        -> Index lookup on Cancelled using c_num_idx (flight_number=Flights.flight_number)  (cost=0.36 rows=1) (never executed)
                    -> Hash
                        -> Table scan on DelayData  (cost=2.50..2.50 rows=0) (actual time=0.062..0.062 rows=0 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=0.061..0.061 rows=0 loops=1)
                                -> Table scan on <temporary>  (actual time=0.046..0.046 rows=0 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.044..0.044 rows=0 loops=1)
                                        -> Nested loop inner join  (cost=1.06 rows=1) (actual time=0.033..0.033 rows=0 loops=1)
                                            -> Filter: (Flights.flight_number is not null)  (cost=0.71 rows=1) (actual time=0.032..0.032 rows=0 loops=1)
                                                -> Index range scan on Flights using f_date_idx over ('2015-10-27' <= date_of_flight <= '2015-11-01'), with index condition: ((Flights.date_of_flight >= DATE'2015-10-
27') and (Flights.date_of_flight <= DATE'2015-11-01'))  (cost=0.71 rows=1) (actual time=0.030..0.030 rows=0 loops=1)
                                            -> Index lookup on Delays using d_num_idx (flight_number=Flights.flight_number)  (cost=0.35 rows=1) (never executed)
 |
```

We once again created the extra Index Lookup steps, but this time the cost for the Index Lookup on Cancelled was only 0.26. The cost of Stream Results was 10685.02. Considering that both queries had smaller costs under this final design than any of the previous designs, this is the design that we will be using in our final database.