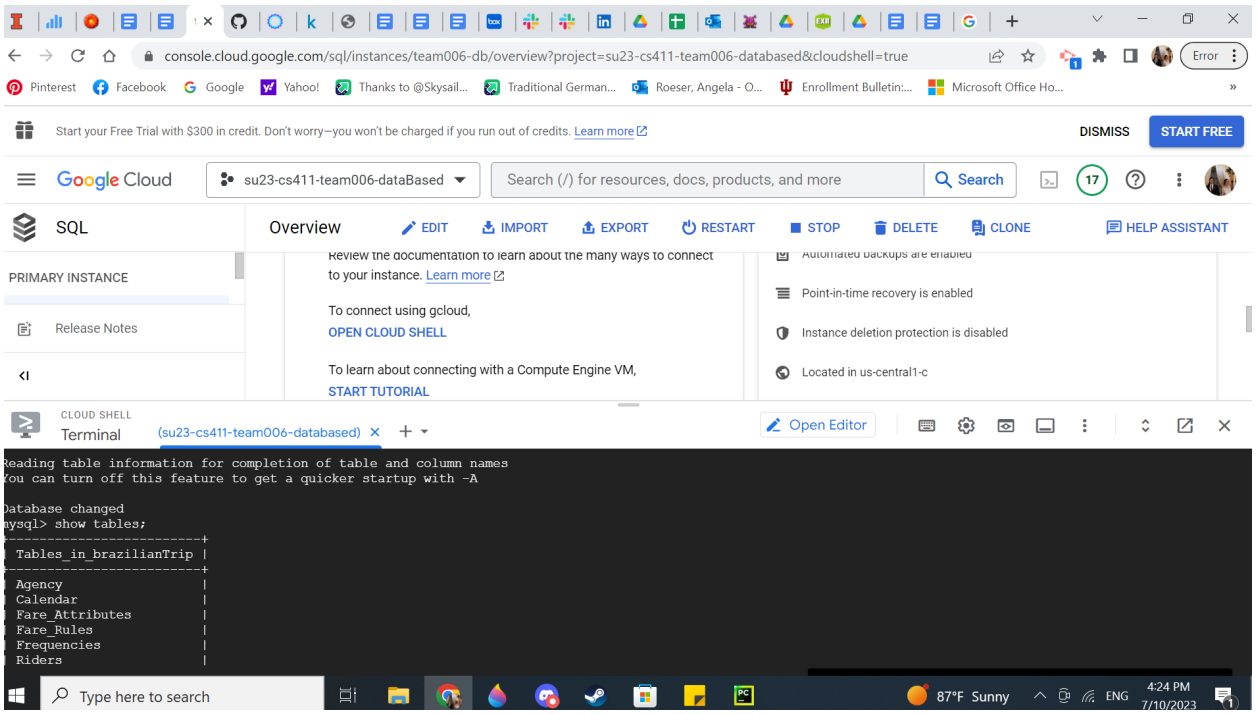
1. Database Implementation Artifacts
    1. Screenshot of the Google Cloud Connection and Terminal



    2. DDL commands for your tables

```
CREATE TABLE Agency(
  agency_id INT PRIMARY KEY,
  agency_name VARCHAR(255),
  agency_url VARCHAR(255),
  agency_timezone VARCHAR(255),
  agency_lang VARCHAR(10)
);

CREATE TABLE Calendar(
  service_id CHAR(3) PRIMARY KEY,
  monday INT,
  tuesday INT,
  wednesday INT,
```

```
  thursday INT,
  friday INT,
  saturday INT,
  sunday INT,
  start_date INT,
  end_date INT
);

CREATE TABLE Fare_Attributes(
  fare_id VARCHAR(100) PRIMARY KEY,
  price REAL,
  currency_type VARCHAR(10),
  payment_method INT,
  transfers INT,
  transfer_duration INT
);

CREATE TABLE Fare_Rules(
  fare_id VARCHAR(100) REFERENCES Fare_Attributes(fare_id) ON
DELETE CASCADE,
  route_id VARCHAR(10) REFERENCES Routes(route_id) ON DELETE
CASCADE,
  origin_id VARCHAR(255),
  destination_id VARCHAR(255),
  contains_id VARCHAR(255),
  PRIMARY KEY(fare_id, route_id)
);

CREATE TABLE Frequencies(
  trip_id VARCHAR(20) REFERENCES Trips(trip_id) ON DELETE
CASCADE,
  start_time TIME,
```

```
  end_time TIME,
  headway_secs INT,
  PRIMARY KEY(trip_id, start_time)
);

CREATE TABLE Routes(
  route_id VARCHAR(10) PRIMARY KEY,
  agency_id INT REFERENCES Agency(agency_id) ON DELETE
CASCADE,
  route_short_name VARCHAR(10),
  route_long_name VARCHAR(500),
  route_type INT,
  route_color VARCHAR(10),
  route_text_color CHAR(6)
);

CREATE TABLE Shapes(
  shape_id INT REFERENCES Trips(shape_id),
  shape_pt_lat REAL,
  shape_pt_lon REAL,
  shape_pt_sequence INT,
  shape_dist_traveled REAL,
Primary Key(shape_id, shape_pt_sequence)
);

CREATE TABLE Stop_Times(
  trip_id VARCHAR(30) REFERENCES Trips(trip_id) ON DELETE
CASCADE,
  arrival_time TIME,
  departure_time TIME,
  stop_id INT REFERENCES Stops(stop_id),
```

```
  stop_sequence INT,
  Primary Key(trip_id, stop_sequence)
);

CREATE TABLE Stops(
stop_id INT PRIMARY KEY,
stop_name VARCHAR(255),
stop_desc VARCHAR(100),
stop_lat DECIMAL,
stop_lon DECIMAL
);

CREATE TABLE Trips(
  route_id VARCHAR(10) REFERENCES Routes(route_id) ON DELETE
CASCADE,
  service_id CHAR(3) REFERENCES Calendar(service_id) ON DELETE
CASCADE,
  trip_id VARCHAR(30) PRIMARY KEY,
  trip_headsign VARCHAR(255),
  direction_id INT,
  shape_id INT
);

CREATE TABLE Riders(
  user_id INT PRIMARY KEY,
  email VARCHAR(255),
  password VARCHAR(50)
);

CREATE TABLE Travel(
  user_id INT REFERENCES Riders(user_id) ON DELETE CASCADE,
```

trip_id CHAR(9) REFERENCES Trips(trip_id) ON DELETE CASCADE,

date DATE,

PRIMARY KEY (user_id, trip_id)

);

3. Count Number of Rows Per Table
    a. 12 Tables Total
    b. Tables with over 1,000 rows:
        i. Fare_Rules,
        ii. Frequencies,
        iii. Routes,
        iv. Shapes,
        v. Stops,
        vi. Stop_Times,
        vii. Trips
    c. Agency: 1 Row

```
| Travel                   |
| Trips                    |
+--------------------------+
12 rows in set (0.01 sec)

mysql> Select Count(*) From Agency;
+----------+
| Count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.02 sec)

mysql>
```

    d. Calendar: 6 rows

```
mysql> Select COUNT(*) From Calendar;
+----------+
| COUNT(*) |
+----------+
|        6 |
+----------+
1 row in set (0.09 sec)

mysql>
```

e. Fare_Attributes: 6 rows

```
mysql> Select COUNT(*) From Fare_Attributes;
+----------+
| COUNT(*) |
+----------+
|        6 |
+----------+
1 row in set (0.02 sec)

mysql>
```

f. Fare_Rules: 5427 rows

```
mysql> Select COUNT(*) From Fare_Rules;
+----------+
| COUNT(*) |
+----------+
|     5427 |
+----------+
1 row in set (0.04 sec)

mysql>
```

g. Frequencies: 39,816 rows

```
mysql> Select COUNT(*) From Frequencies;
+----------+
| COUNT(*) |
+----------+
|    39816 |
+----------+
1 row in set (0.13 sec)

mysql>
```

h. Riders: 0 rows
    i. This will be the ‘Users’ table, and will populate later

i. Routes: 1,360 rows

```
mysql> Select COUNT(*) From Routes;
+----------+
| COUNT(*) |
+----------+
|     1360 |
+----------+
1 row in set (0.14 sec)

mysql>
```

j. Shapes: 1,048,575 rows

```
mysql> Select COUNT(*) From Shapes;
+----------+
| COUNT(*) |
+----------+
|  1048575 |
+----------+
1 row in set (8.50 sec)
```

k. Stop_Times: 95,265 rows

```
mysql> select COUNT(*) from Stop_Times;
+----------+
| COUNT(*) |
+----------+
|    95265 |
+----------+
1 row in set (0.42 sec)
```

l. Stops: 20,902 rows

```
mysql> select COUNT(*) From Stops;
+----------+
| COUNT(*) |
+----------+
|    20902 |
+----------+
1 row in set (0.17 sec)
```

m. Travel: 0 rows
    i. This will be populated as Riders (Users) save trips.
n. Riders: 0 rows
    i. This will be populated as more users use the app.
o. Trips: 2,227 rows

```
mysql> Select COUNT(*) From Trips;
+----------+
| COUNT(*) |
+----------+
|     2227 |
+----------+
1 row in set (0.09 sec)
```

3. Advanced Queries

1. Routes for a Specified Stop
    a. Description: This query will get the routes and lines of service for a specified user-inputted destination (stop_id). This query will return the route color, text color, and the name of the trips that serve that stop, as well as the average frequencies of each trip.
    b. Code

```
SET @route_type = 3;
SET @stop_id = '600012396';
SET @trip_date =
STR_TO_DATE('2023-07-10T18:30:00.000','%Y-%m-%dT%H:%i:%s.%f');

SELECT
r.route_color,
COALESCE(NULLIF(r.route_text_color, ''), '000000') AS route_text_color,
-- If color is blank use black
t.trip_headsign AS service_name,
ROUND(AVG(f.headway_secs)/60) avg_interval_minutes

FROM Stops s
INNER JOIN Stop_Times st ON s.stop_id = st.stop_id
INNER JOIN Trips t ON st.trip_id = t.trip_id
INNER JOIN Calendar c ON t.service_id = c.service_id
INNER JOIN Frequencies f ON t.trip_id = f.trip_id AND
TIME(COALESCE(@trip_date, now())) BETWEEN f.start_time AND
f.end_time
INNER JOIN Routes r ON t.route_id = r.route_id
WHERE
route_type = @route_type AND
s.stop_id = @stop_id AND
CASE     WHEN dayname(COALESCE(@trip_date, now())) = 'monday'
THEN c.monday = 1
      WHEN dayname(COALESCE(@trip_date, now())) = 'tuesday'
THEN c.tuesday = 1
      WHEN dayname(COALESCE(@trip_date, now())) = 'wednesday'
THEN c.wednesday = 1
      WHEN dayname(COALESCE(@trip_date, now())) = 'thursday'
THEN c.thursday = 1
```

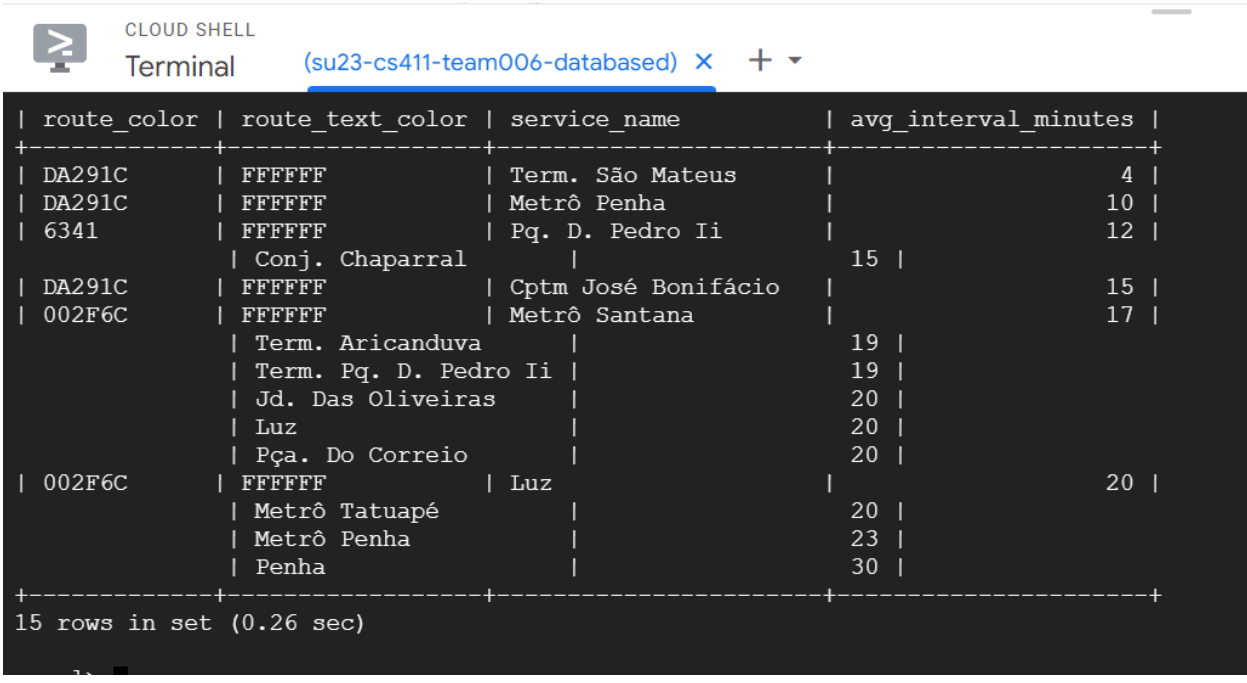
```
        WHEN dayname(COALESCE(@trip_date, now())) = 'friday'     THEN
c.friday = 1
        WHEN dayname(COALESCE(@trip_date, now())) = 'saturday'
THEN c.saturday = 1
        WHEN dayname(COALESCE(@trip_date, now())) = 'sunday'
THEN c.sunday = 1
END

GROUP BY
r.route_color,
r.route_text_color,
t.trip_headsign

ORDER BY avg_interval_minutes

limit 15;
```

c. Screenshot




```
| route_color | route_text_color | service_name          | avg_interval_minutes |
+-------------+------------------+-----------------------+----------------------+
| DA291C      | FFFFFF           | Term. São Mateus      |                    4 |
| DA291C      | FFFFFF           | Metrô Penha           |                   10 |
| 6341        | FFFFFF           | Pq. D. Pedro Ii       |                   12 |
             | Conj. Chaparral       |                   15 |
| DA291C      | FFFFFF           | Cptm José Bonifácio   |                   15 |
| 002F6C      | FFFFFF           | Metrô Santana         |                   17 |
             | Term. Aricanduva      |                   19 |
             | Term. Pq. D. Pedro Ii |                   19 |
             | Jd. Das Oliveiras     |                   20 |
             | Luz                   |                   20 |
             | Pça. Do Correio       |                   20 |
| 002F6C      | FFFFFF           | Luz                   |                   20 |
             | Metrô Tatuapé         |                   20 |
             | Metrô Penha           |                   23 |
             | Penha                 |                   30 |
+-------------+------------------+-----------------------+----------------------+
15 rows in set (0.26 sec)
```

d. Sources Cited

i. -- Source for TIME function: https://www.w3resource.com/mysql/date-and-time-functions/mysql-time-function.php
ii. -- Source for STR_TO_DATE: https://stackoverflow.com/questions/44802061/convert-string-to-datetime-in-my-sql
iii. -- Source for COALESCE w/ NULLIF: https://stackoverflow.com/questions/4101564/mysql-coalesce-equivalent-for-empty-values

2. User Stop Look-up
    a. Description: This query will look up a stop by its name, description, or stop_id and return partial matches to the User, so the User can select which trip works best for them.
    b. Code

```
SET @searchTerm = 'Dos';

SELECT
label,
stop_id
FROM
(
SELECT DISTINCT s.stop_id label, s.stop_id, 1 AS listorder

FROM Stops s INNER JOIN Stop_Times st ON s.stop_id = st.stop_id
INNER JOIN Trips t ON st.trip_id = t.trip_id
INNER JOIN Routes r ON t.route_id = r.route_id

WHERE s.stop_id like CONCAT(@searchTerm,'%')

UNION
SELECT DISTINCT CONCAT(s.stop_name, " - ", s.stop_desc) label, s.stop_id, 2 AS listorder

FROM Stops s INNER JOIN Stop_Times st ON s.stop_id = st.stop_id
```

```
INNER JOIN Trips t ON st.trip_id = t.trip_id
INNER JOIN Routes r ON t.route_id = r.route_id

WHERE s.stop_name like CONCAT(@searchTerm,'%')

UNION

SELECT DISTINCT CONCAT(s.stop_name, " - ", s.stop_desc) label,
s.stop_id, 3 AS listorder

FROM Stops s INNER JOIN Stop_Times st ON s.stop_id = st.stop_id
INNER JOIN Trips t ON st.trip_id = t.trip_id
INNER JOIN Routes r ON t.route_id = r.route_id

WHERE s.stop_desc like CONCAT(@searchTerm,'%')

UNION

SELECT DISTINCT s.stop_id label, s.stop_id, 4 AS listorder

FROM Stops s INNER JOIN Stop_Times st ON s.stop_id = st.stop_id
INNER JOIN Trips t ON st.trip_id = t.trip_id
INNER JOIN Routes r ON t.route_id = r.route_id

WHERE s.stop_id like CONCAT('%',@searchTerm,'%')

UNION

SELECT DISTINCT CONCAT(s.stop_name, " - ", s.stop_desc) label,
s.stop_id, 5 AS listorder
```

```
FROM Stops s INNER JOIN Stop_Times st ON s.stop_id = st.stop_id
INNER JOIN Trips t ON st.trip_id = t.trip_id
INNER JOIN Routes r ON t.route_id = r.route_id

WHERE s.stop_name like CONCAT('%',@searchTerm,'%')

UNION

SELECT DISTINCT CONCAT(s.stop_name, " - ", s.stop_desc) label,
s.stop_id, 6 AS listorder

FROM Stops s INNER JOIN Stop_Times st ON s.stop_id = st.stop_id
INNER JOIN Trips t ON st.trip_id = t.trip_id
INNER JOIN Routes r ON t.route_id = r.route_id

WHERE s.stop_desc like CONCAT('%',@searchTerm,'%')

ORDER BY listorder, label
) stop_labels

LIMIT 15;
```

c. Screenshot

CLOUD SHELL
Terminal (su23-cs411-team006-databased) ×

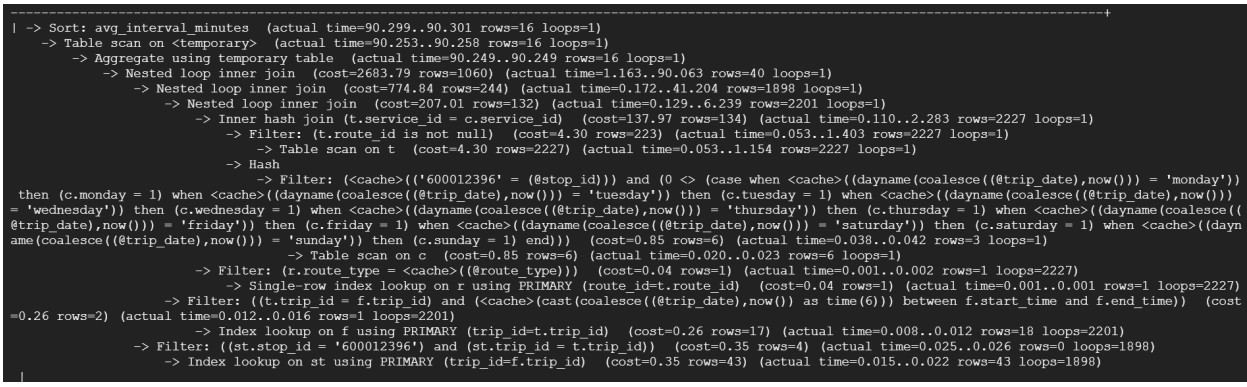
```
| label                                                                                                  | stop_id   |
+--------------------------------------------------------------------------------------------------------+-----------+
| Dos Guaicanas - Av. Miruna, 1484 Ref.: Al Dos Guaicanas/ R Alberto Willo                               | 790015511 |
| Dos Guaicanas - Av. Miruna, 1539 Ref.: Al Dos Guaicanas/ R Carmelo Damato                              | 790015512 |
| Dos Guainumbis - Av. Miruna, 1144 Ref.: Al Dos Guainumbis/ Al Dos Araes                                | 790015507 |
| Dos Inocentes B/C - Av. Atlântica Ref.: R Nossa Senhora Do Socorro/ R Dos Inocentes                    | 810009941 |
| Dos Inocentes C/B - Av. Atlântica, 748 Ref.: R Nossa Senhora Do Socorro/ R Dos Inocentes               | 810009942 |
| Dos Lagos C/B - Av. Do Rio Bonito, 890 Ref.: Av Dos Lagos/ R Guaratiba                                 | 810009909 |
| Dos Mendes B/C - Av. Sen. Teotônio Vilela, 6162 Ref.: R Luiz Rotta / R Vitor Lima Barreto              | 230009860 |
| Dos Mendes C/B - Av. Sen. Teotônio Vilela, 6000 Ref.: R Luiz Rotta/ R Vitor Lima Barreto               | 230009847 |
| Dos Otonis B/C - R. Borges Lagoa, 905 Ref.: R Dos Otonis/ R Leandro Dupre                              | 920015204 |
| Dos Otonis C/B - R. Pedro De Toledo, 848 Ref.: R Dos Otonis/ R Varpa                                   | 920015208 |
| Dos Pinheiros - Av. Brig. Faria Lima, 1215 Ref.: Proximo Ao Cruzamento Com R. Dos Pinheiros            |   6311346 |
| Dos Piratinins - Av. Miruna, 1704 Ref.: Al Dos Piratinins/ R Carmelo Damato                            | 790015513 |
| Dos Tupinas - Av. Bandeirantes, 5569 Ref.: Al. Dos Tupinás / R. Prof. Ataliba De Oliveira              |   7906153 |
| Dos Uapichanas - Av. Moaci, 1128 Ref.: Al Dos Uapixanas/ Av Moreira Guimaraes                          | 790015508 |
| Al. 2º Sg. Névio Baracho Dos Santos, 357 - Ref.: R Doutor Vidal Reis/ R Fritz Jank                     | 910000823 |
+--------------------------------------------------------------------------------------------------------+-----------+
15 rows in set (1.57 sec)
```

d. Sources Cited
   i. -- Source for CONCAT function: https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_concat
   ii. -- Source for CONCAT with wildcards https://stackoverflow.com/questions/6343133/sql-query-with-mysql-variable-in-like-statement

4. Indexing
   1. Routes for a Specified Stop
      a. Explain Analyze Runtime Without Indexes

```
| -> Sort: avg_interval_minutes  (actual time=832.980..832.983 rows=16 loops=1)
    -> Table scan on <temporary>  (actual time=832.915..832.919 rows=16 loops=1)
        -> Aggregate using temporary table  (actual time=832.910..832.910 rows=16 loops=1)
            -> Nested loop inner join  (cost=909.50 rows=107) (actual time=244.001..832.475 rows=40 loops=1)
                -> Nested loop inner join  (cost=544.10 rows=25) (actual time=106.529..392.981 rows=1898 loops=1)
                    -> Nested loop inner join  (cost=385.29 rows=13) (actual time=85.441..185.475 rows=2201 loops=1)
                        -> Inner hash join (t.service_id = c.service_id)  (cost=149.23 rows=134) (actual time=62.884..99.871 rows=2227 loops=1)
                            -> Filter: (t.route_id is not null)  (cost=6.05 rows=223) (actual time=41.376..76.626 rows=2227 loops=1)
                                -> Table scan on t  (cost=6.05 rows=2227) (actual time=41.372..76.168 rows=2227 loops=1)
                            -> Hash
                                -> Filter: (<cache>(('600012396' = (@stop_id))) and (0 <> (case when <cache>((dayname(coalesce((@trip_date),now())) = 'monday')) then (c.monday = 1) when <cache>((dayname(coalesce((@trip_date),now()))) = 'tuesday')) then (c.tuesday = 1) when <cache>((dayname(coalesce((@trip_date),now()))) = 'wednesday')) then (c.wednesday = 1) when <cache>((dayname(coalesce((@trip_date),now()))) = 'thursday')) then (c.thursday = 1) when <cache>((dayname(coalesce((@trip_date),now()))) = 'friday')) then (c.friday = 1) when <cache>((dayname(coalesce((@trip_date),now()))) = 'saturday')) then (c.saturday = 1) when <cache>((dayname(coalesce((@trip_date),now()))) = 'sunday')) then (c.sunday = 1) end)))  (cost=1.60 rows=6) (actual time=21.477..21.484 rows=3 loops=1)
                                    -> Table scan on c  (cost=1.60 rows=6) (actual time=21.444..21.449 rows=6 loops=1)
                        -> Filter: (r.route_type = <cache>((@route_type)))  (cost=0.17 rows=0.1) (actual time=0.038..0.038 rows=1 loops=2227)
                            -> Single-row index lookup on r using PRIMARY (route_id=t.route_id)  (cost=0.17 rows=1) (actual time=0.038..0.038 rows=1 loops=2227)
                    -> Filter: ((t.trip_id = f.trip_id) and (<cache>(cast(coalesce((@trip_date),now()) as time(6))) between f.start_time and f.end_time))  (cost=1.02 rows=2) (actual time=0.087..0.094 rows=1 loops=2201)
                        -> Index lookup on f using PRIMARY (trip_id=t.trip_id)  (cost=1.02 rows=17) (actual time=0.082..0.090 rows=18 loops=2201)
                -> Filter: ((st.stop_id = '600012396') and (st.trip_id = t.trip_id))  (cost=1.05 rows=4) (actual time=0.231..0.231 rows=0 loops=1898)
                    -> Index lookup on st using PRIMARY (trip_id=f.trip_id)  (cost=1.05 rows=43) (actual time=0.209..0.228 rows=43 loops=1898)
```

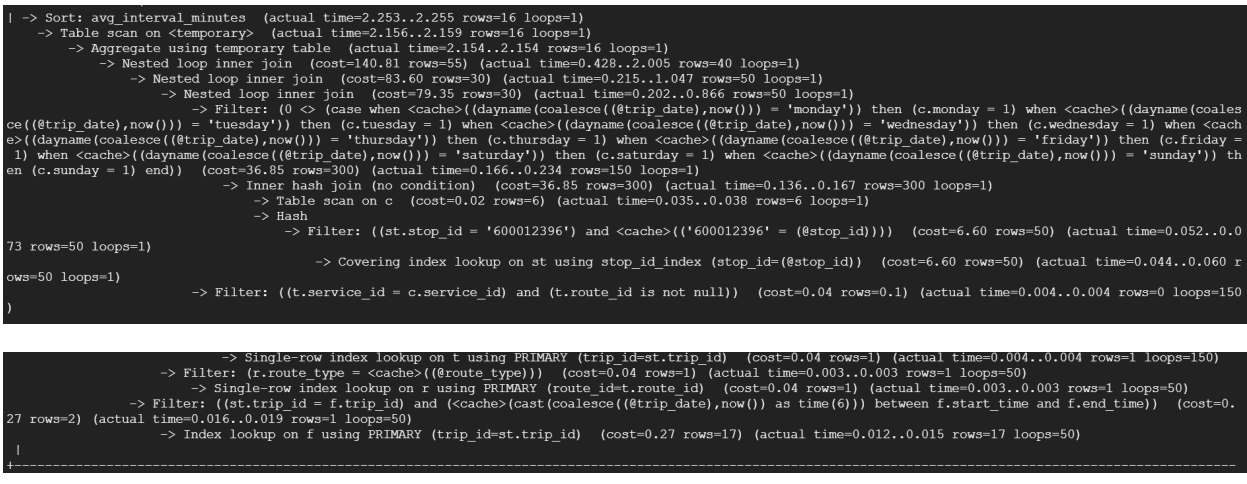
      b. Explain Analyze Runtime- Index on Routes(Route_Type)



Discussion:

To start with optimizing this query, we looked to the WHERE clause to see if we can be more efficient with our resources. The WHERE clause in this query is looking for an attribute route_type in the Routes table that matches the user inputted route_type variable. We deduced that in this statement the query is having to scan over every single record in Routes, which is one of the larger tables. Thus, we added an index on the attribute route_type. Route_type is an integer value in the relation that indicates if a trip is by bus, train, or metra. Since the user has indicated that they wish to see a '3' or 'bus' schedule, we can create an index to help reduce the amount of records that the query needs to read. This sped up our time from, approximately, 832 to 90.

c. Explain Analyze Runtime- Index on Routes(Route_Type) and Stop_Times(Stop_Id)

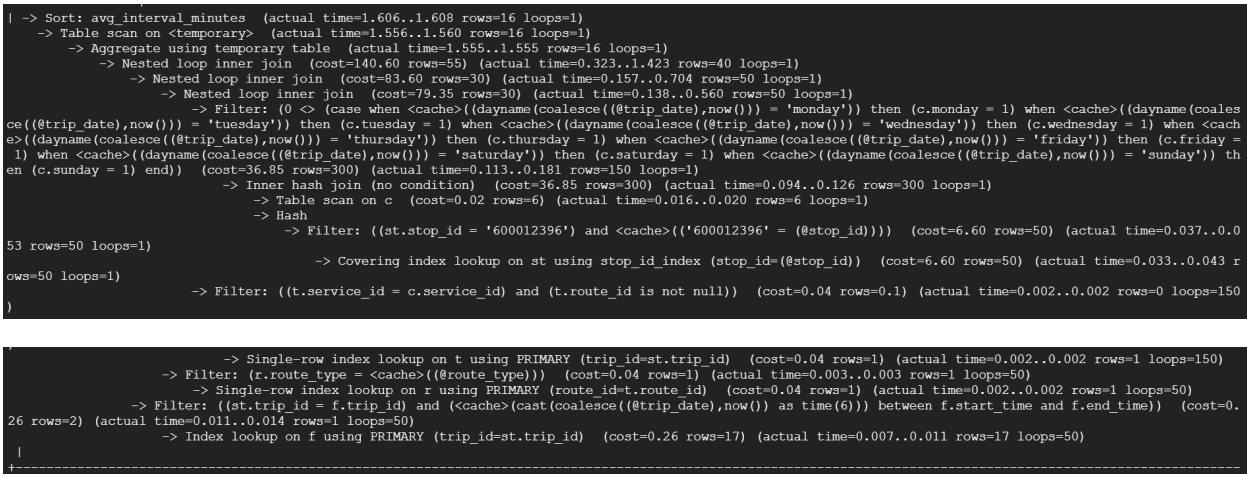


Discussion:

After creating an index on route type, we noticed that a lot of the resources utilized were dedicated to the tables joining in the From clause. We are joining Stops, Stop_Times, Trips, Calendar, Frequencies, and

Routes on their primary and foreign keys. This is adding a considerable amount of stress to the query due to the amount of joins as well as some of these tables (especially Frequencies) being voluminous, and so we decided to try to add an index to Stop_Times(stop_id). Stop_Times is a large table that has two primary keys on trip_id and stop_id. There are many duplicate stop_ids that this query has to parse, so creating an index will hopefully reduce the amount of time spent. The results of this index, with the aforementioned route_type index, brough the resources utilized from, approximately, 90 to 2.

d. Explain Analyze Runtime- Index on Routes(Route_Type) and Stop_Times(Stop_Id) and Frequencies(Trip_Id)



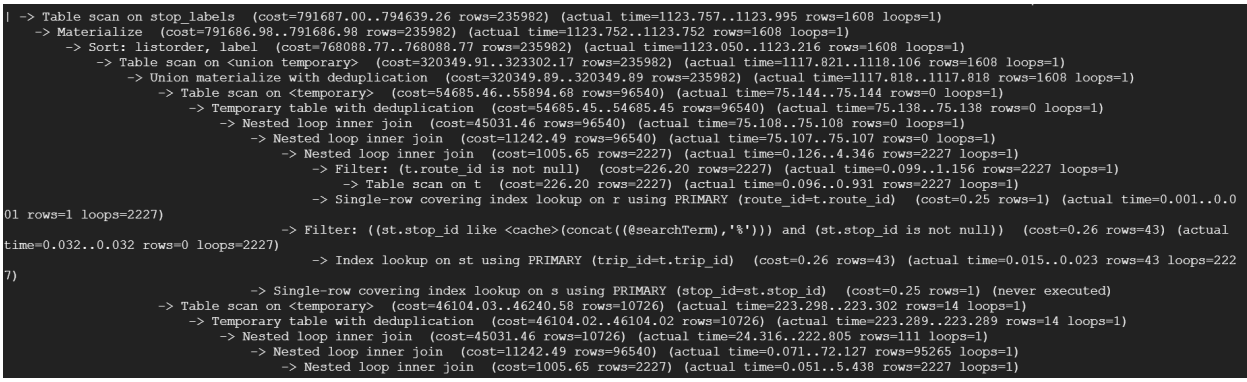
```
| -> Sort: avg_interval_minutes  (actual time=1.606..1.608 rows=16 loops=1)
    -> Table scan on <temporary>  (actual time=1.556..1.560 rows=16 loops=1)
        -> Aggregate using temporary table  (actual time=1.555..1.555 rows=16 loops=1)
            -> Nested loop inner join  (cost=140.60 rows=55) (actual time=0.323..1.423 rows=40 loops=1)
                -> Nested loop inner join  (cost=83.60 rows=30) (actual time=0.157..0.704 rows=50 loops=1)
                    -> Nested loop inner join  (cost=79.35 rows=30) (actual time=0.138..0.560 rows=50 loops=1)
                        -> Filter: (0 <> (case when <cache>((dayname(coalesce((@trip_date),now())) = 'monday')) then (c.monday = 1) when <cache>((dayname(coales
ce((@trip_date),now())) = 'tuesday')) then (c.tuesday = 1) when <cache>((dayname(coalesce((@trip_date),now())) = 'wednesday')) then (c.wednesday = 1) when <cach
e>((dayname(coalesce((@trip_date),now())) = 'thursday')) then (c.thursday = 1) when <cache>((dayname(coalesce((@trip_date),now())) = 'friday')) then (c.friday =
 1) when <cache>((dayname(coalesce((@trip_date),now())) = 'saturday')) then (c.saturday = 1) when <cache>((dayname(coalesce((@trip_date),now())) = 'sunday')) th
en (c.sunday = 1) end))  (cost=36.85 rows=300) (actual time=0.113..0.181 rows=150 loops=1)
                            -> Inner hash join (no condition)  (cost=36.85 rows=300) (actual time=0.094..0.126 rows=300 loops=1)
                                -> Table scan on c  (cost=0.02 rows=6) (actual time=0.016..0.020 rows=6 loops=1)
                                -> Hash
                                    -> Filter: ((st.stop_id = '600012396') and <cache>(('600012396' = (@stop_id))))  (cost=6.60 rows=50) (actual time=0.037..0.0
53 rows=50 loops=1)
                                        -> Covering index lookup on st using stop_id_index (stop_id=(@stop_id))  (cost=6.60 rows=50) (actual time=0.033..0.043 r
ows=50 loops=1)
                        -> Filter: ((t.service_id = c.service_id) and (t.route_id is not null))  (cost=0.04 rows=0.1) (actual time=0.002..0.002 rows=0 loops=150
)
```

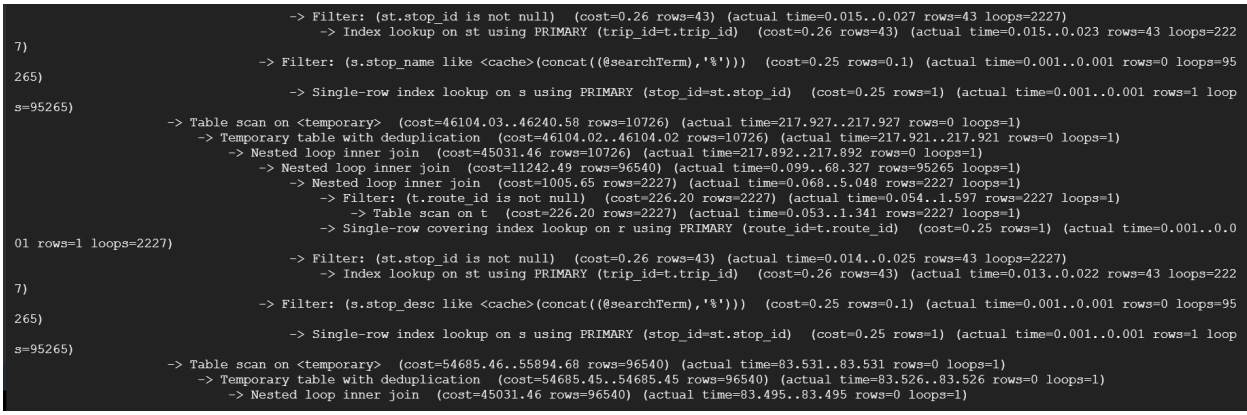
```
                            -> Single-row index lookup on t using PRIMARY (trip_id=st.trip_id)  (cost=0.04 rows=1) (actual time=0.002..0.002 rows=1 loops=150)
                    -> Filter: (r.route_type = <cache>((@route_type)))  (cost=0.04 rows=1) (actual time=0.003..0.003 rows=1 loops=50)
                        -> Single-row index lookup on r using PRIMARY (route_id=t.route_id)  (cost=0.04 rows=1) (actual time=0.002..0.002 rows=1 loops=50)
                -> Filter: ((st.trip_id = f.trip_id) and (<cache>(cast(coalesce((@trip_date),now()) as time(6))) between f.start_time and f.end_time))  (cost=0.
26 rows=2) (actual time=0.011..0.014 rows=1 loops=50)
                    -> Index lookup on f using PRIMARY (trip_id=st.trip_id)  (cost=0.26 rows=17) (actual time=0.007..0.011 rows=17 loops=50)
 |
 +-----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

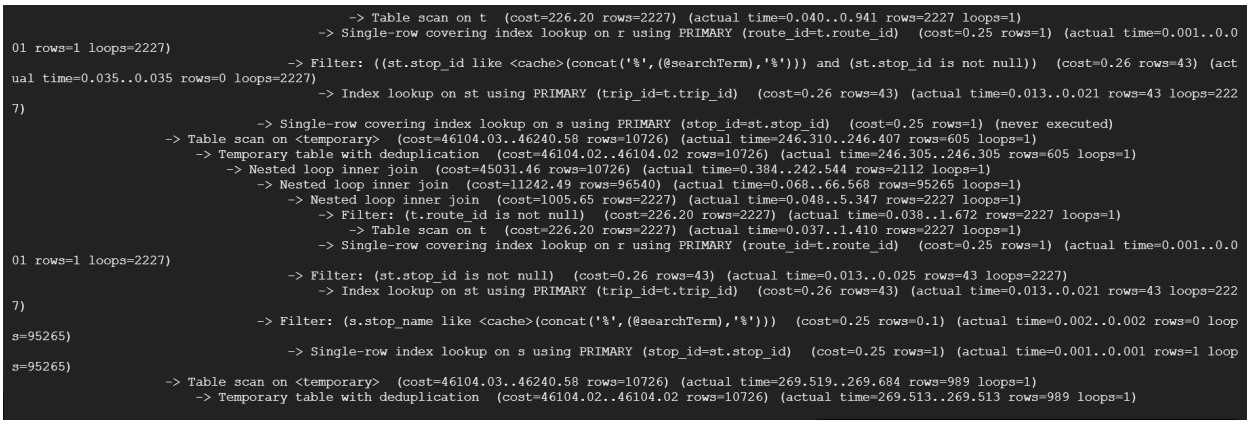
Discussion:

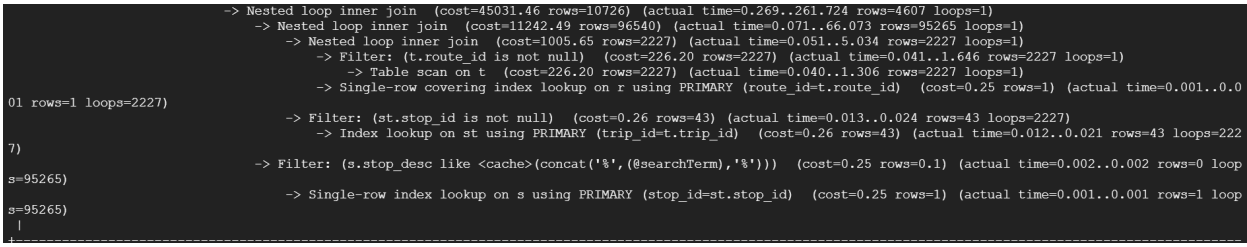
We correctly identified the bulk of the resources utilized from the FROM statement and the six joined tables, so we decided to continue to analyze this clause to see where we can improve. One of the larger tables in this JOIN is the Frequencies table which is joining the Trips table with the common key ‘trip_id’. The Frequencies table has many duplicate trip_id values because the same ‘trip’ can run multiple times a day and week. For example, the Illini bus on campus will run nearly every 15 minutes during working hours. Adding an index to the trip_id in the Frequencies table would help connect the query to the needed data in an efficient manner. The addition of this index to the route_type and stop_id brought the time from, approximately, 2 to 1.

2. User Stop Look-up
    a. Explain Analyze Runtime Without Indexes

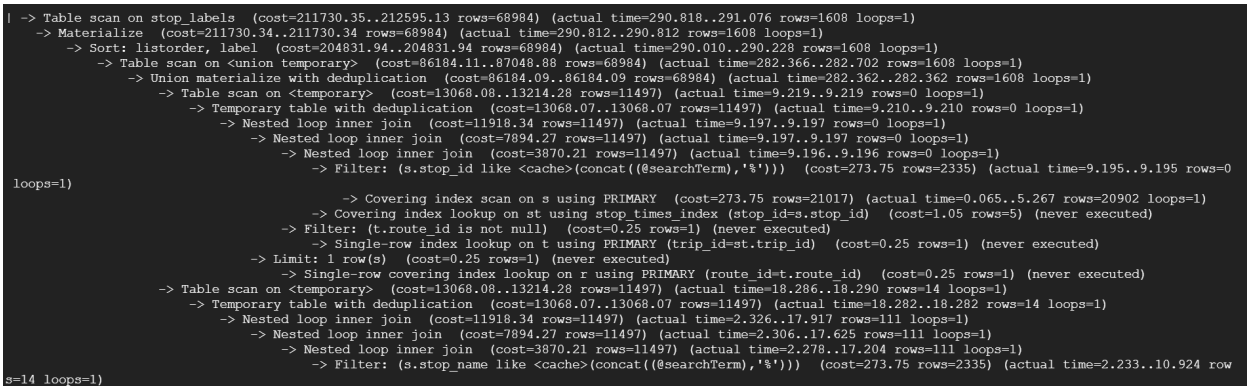


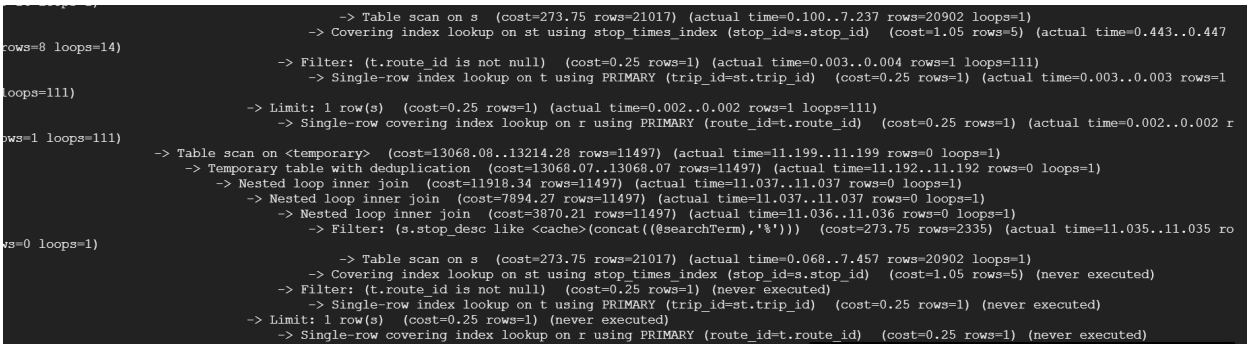


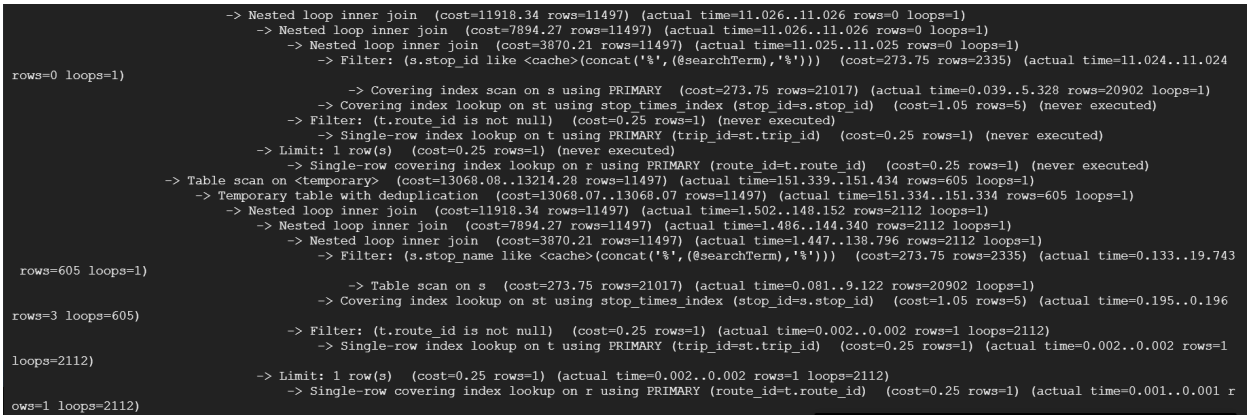


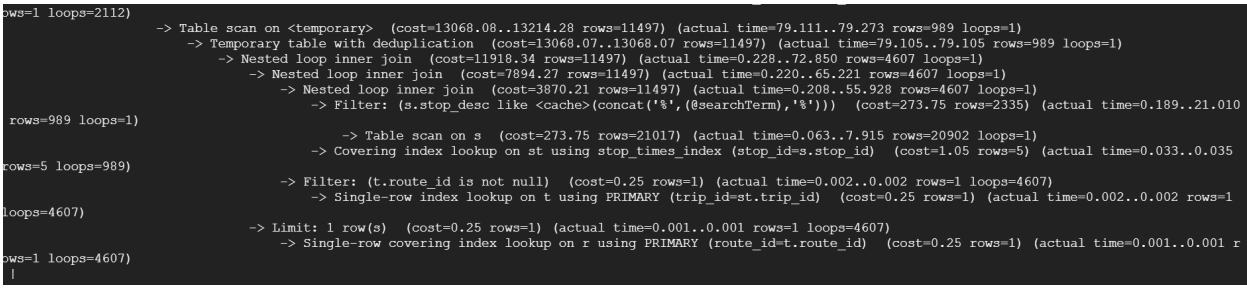


b. Explain Analyze Runtime- Index on Stop_Times(Stop_Id)







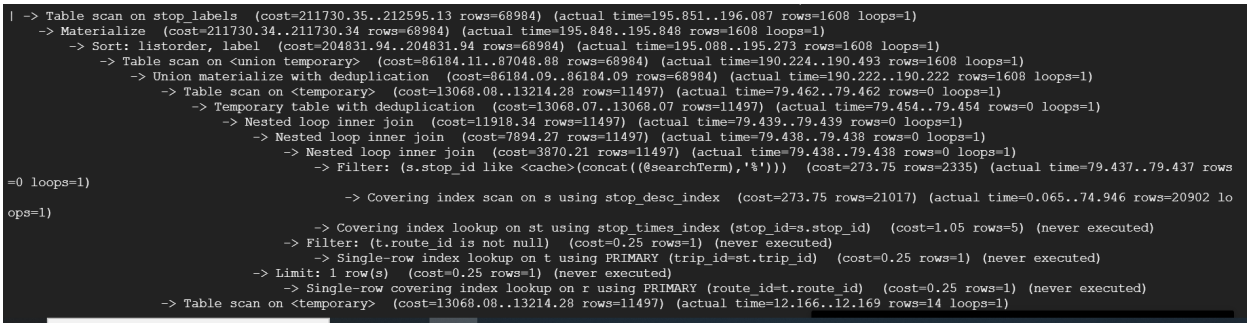


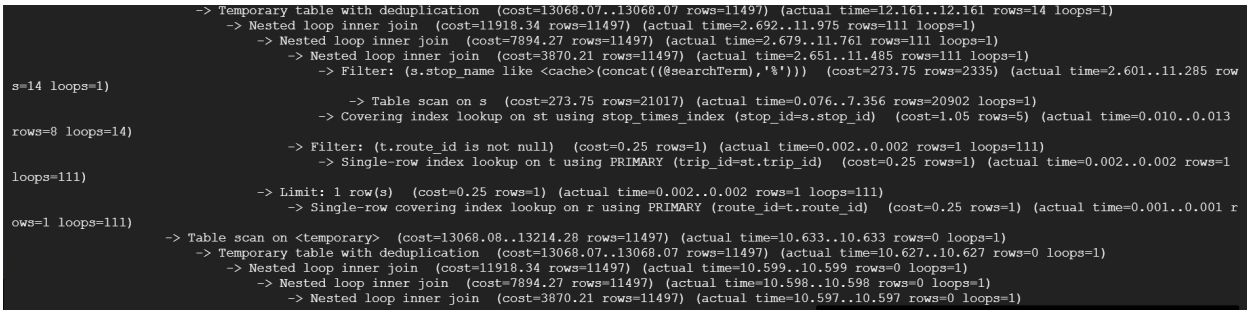
Discussion:

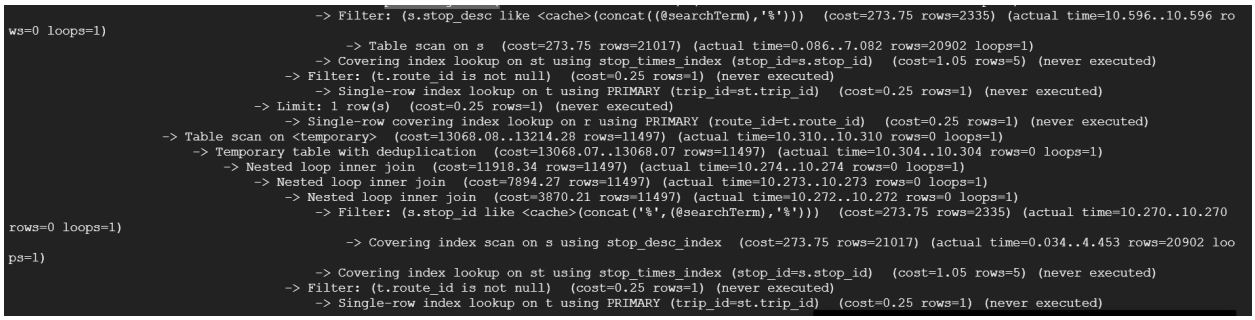
This query is unique because it does not have a lot of filters in the WHERE clause, but instead relies on five union operations to gather the

necessary information the user specified. There are only four tables being joined in this query: Stops, Stop_Times, Trips, and Routes. We deduced that a lot of the resources spent were on the five times these four tables needed to be joined together, so we decided to add an index on Stop_Times(Stop_Id). Similar to the above query, there are multiple stops on route, and the route will run multiple times a day and/or week. Adding this index reduced the time spent from, approximately, 791,000 to 211,000.

c. Explain Analyze Runtime- Index on Stop_Times(Stop_Id) and Stop(Stop_Desc)







```
                            -> Limit: 1 row(s)  (cost=0.25 rows=1) (never executed)
                                -> Single-row covering index lookup on r using PRIMARY (route_id=t.route_id)  (cost=0.25 rows=1) (never executed)
                        -> Table scan on <temporary>  (cost=13068.08..13214.28 rows=11497) (actual time=28.117..28.210 rows=605 loops=1)
                            -> Temporary table with deduplication  (cost=13068.07..13068.07 rows=11497) (actual time=28.111..28.111 rows=605 loops=1)
                                -> Nested loop inner join  (cost=11918.34 rows=11497) (actual time=0.187..25.772 rows=2112 loops=1)
                                    -> Nested loop inner join  (cost=7894.27 rows=11497) (actual time=0.175..22.783 rows=2112 loops=1)
                                        -> Nested loop inner join  (cost=3870.21 rows=11497) (actual time=0.155..19.337 rows=2112 loops=1)
                                            -> Filter: (s.stop_name like <cache>(concat('%',(@searchTerm),'%')))  (cost=273.75 rows=2335) (actual time=0.126..16.098
 rows=605 loops=1)
                                                -> Table scan on s  (cost=273.75 rows=21017) (actual time=0.077..6.844 rows=20902 loops=1)
                                            -> Covering index lookup on st using stop_times_index (stop_id=s.stop_id)  (cost=1.05 rows=5) (actual time=0.004..0.005
rows=3 loops=605)
                                        -> Filter: (t.route_id is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2112)
                                            -> Single-row index lookup on t using PRIMARY (trip_id=st.trip_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1
loops=2112)
                                -> Limit: 1 row(s)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=2112)
                                    -> Single-row covering index lookup on r using PRIMARY (route_id=t.route_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 r
ows=1 loops=2112)
                        -> Table scan on <temporary>  (cost=13068.08..13214.28 rows=11497) (actual time=47.642..47.798 rows=989 loops=1)
```

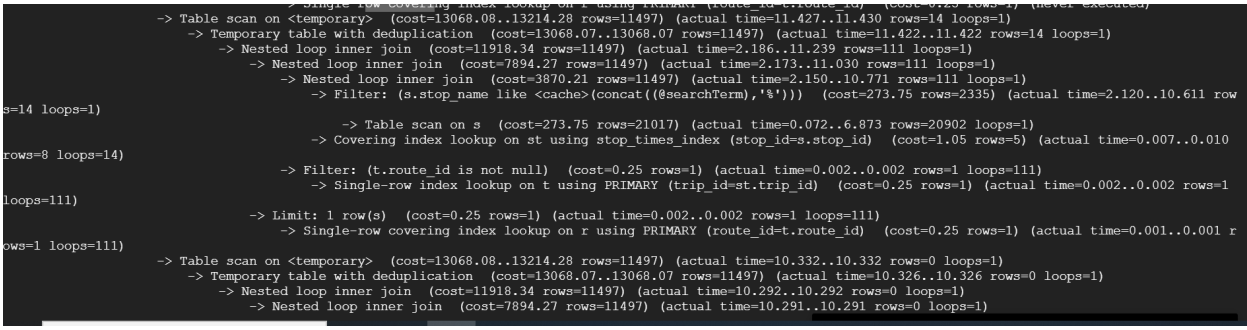
```
                    -> Table scan on <temporary>  (cost=13068.08..13214.28 rows=11497) (actual time=47.642..47.798 rows=989 loops=1)
                        -> Temporary table with deduplication  (cost=13068.07..13068.07 rows=11497) (actual time=47.637..47.637 rows=989 loops=1)
                            -> Nested loop inner join  (cost=11918.34 rows=11497) (actual time=0.194..42.068 rows=4607 loops=1)
                                -> Nested loop inner join  (cost=7894.27 rows=11497) (actual time=0.186..35.136 rows=4607 loops=1)
                                    -> Nested loop inner join  (cost=3870.21 rows=11497) (actual time=0.177..27.001 rows=4607 loops=1)
                                        -> Filter: (s.stop_desc like <cache>(concat('%',(@searchTerm),'%')))  (cost=273.75 rows=2335) (actual time=0.161..20.547
 rows=989 loops=1)
                                            -> Table scan on s  (cost=273.75 rows=21017) (actual time=0.088..7.538 rows=20902 loops=1)
                                        -> Covering index lookup on st using stop_times_index (stop_id=s.stop_id)  (cost=1.05 rows=5) (actual time=0.004..0.006
rows=5 loops=989)
                                    -> Filter: (t.route_id is not null)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=4607)
                                        -> Single-row index lookup on t using PRIMARY (trip_id=st.trip_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1
loops=4607)
                                -> Limit: 1 row(s)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=4607)
                                    -> Single-row covering index lookup on r using PRIMARY (route_id=t.route_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 r
ows=1 loops=4607)
```

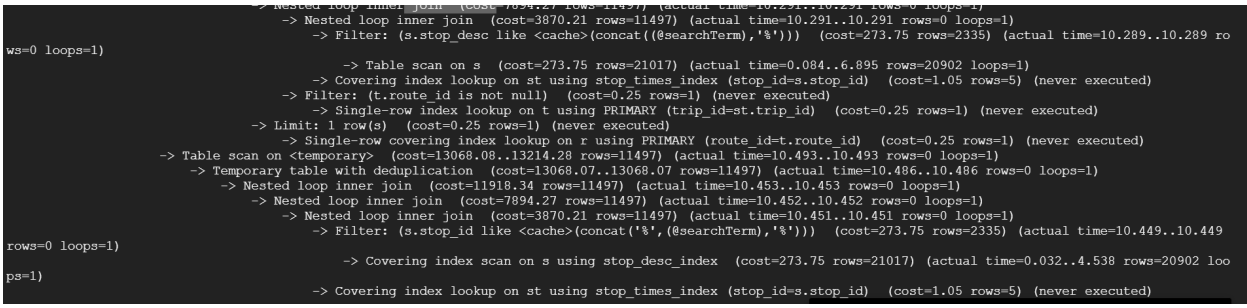
Discussion:

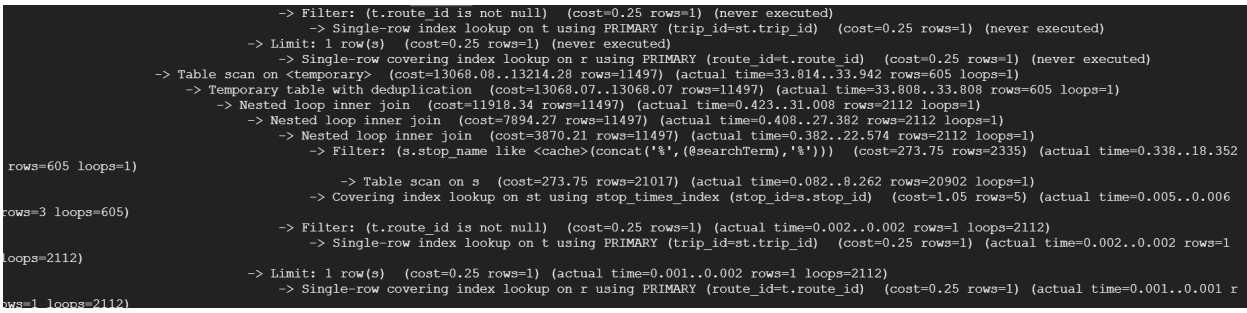
To encourage more optimization, we looked through the query to identify attributes that have longer VARCHAR lengths to see if part of the large amount of resources used was due to parsing through these attributes' values. The largest attribute values we found were in the Stops table with the stop_desc attribute. This attribute is VARCHAR(100), so we hypothesized that the query spent a lot of time sifting through these values to find unique ones to be concatenated in the SELECT clause. However, when we added the index to Stops(stop_desc) there was not a noticeable difference. When we dove into the data, we found that the stop_desc were unique and related to the stop_name, plus the query were only selecting and concatenating these values, so that is not as resource intensive as operations in FROM clause.

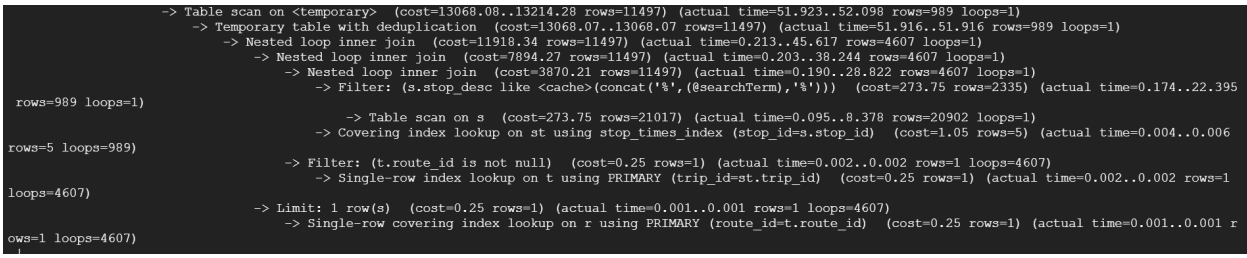
d. Explain Analyze Runtime - Index on Stop_Times(Stop_Id) and Trips(route_id)

```
| -> Table scan on stop_labels  (cost=211730.35..212595.13 rows=68984) (actual time=135.941..136.219 rows=1608 loops=1)
    -> Materialize  (cost=211730.34..211730.34 rows=68984) (actual time=135.936..135.936 rows=1608 loops=1)
        -> Sort: listorder, label  (cost=204831.94..204831.94 rows=68984) (actual time=135.132..135.329 rows=1608 loops=1)
            -> Table scan on <union temporary>  (cost=86184.11..87048.88 rows=68984) (actual time=128.725..128.997 rows=1608 loops=1)
                -> Union materialize with deduplication  (cost=86184.09..86184.09 rows=68984) (actual time=128.722..128.722 rows=1608 loops=1)
                    -> Table scan on <temporary>  (cost=13068.08..13214.28 rows=11497) (actual time=8.742..8.742 rows=0 loops=1)
                        -> Temporary table with deduplication  (cost=13068.07..13068.07 rows=11497) (actual time=8.734..8.734 rows=0 loops=1)
                            -> Nested loop inner join  (cost=11918.34 rows=11497) (actual time=8.722..8.722 rows=0 loops=1)
                                -> Nested loop inner join  (cost=7894.27 rows=11497) (actual time=8.721..8.721 rows=0 loops=1)
                                    -> Nested loop inner join  (cost=3870.21 rows=11497) (actual time=8.721..8.721 rows=0 loops=1)
                                        -> Filter: (s.stop_id like <cache>(concat((@searchTerm),'%')))  (cost=273.75 rows=2335) (actual time=8.719..8.719 rows=0
 loops=1)
                                            -> Covering index scan on s using stop_desc_index  (cost=273.75 rows=21017) (actual time=0.045..4.704 rows=20902 loo
ps=1)
                                        -> Covering index lookup on st using stop_times_index (stop_id=s.stop_id)  (cost=1.05 rows=5) (never executed)
                                    -> Filter: (t.route_id is not null)  (cost=0.25 rows=1) (never executed)
                                        -> Single-row index lookup on t using PRIMARY (trip_id=st.trip_id)  (cost=0.25 rows=1) (never executed)
                                -> Limit: 1 row(s)  (cost=0.25 rows=1) (never executed)
                                    -> Single-row covering index lookup on r using PRIMARY (route_id=t.route_id)  (cost=0.25 rows=1) (never executed)
```









Discussion:

Since we do not have a lot of filtering happening in the WHERE clause of this query, we decided to try a similar approach to the above query where we would try to reduce the amount of resources spent in the FROM clause with the joining. Since the Stop_Times(Stop_Id) was successful, we thought of experimenting with the Trips(route_id), which is one of the foreign keys of Trips. We thought that there could be trips with the same route, but since this index did not help reduce the resources, we feel that the route_id in Trips were unique.