Database Design

Table Creation Commands

This information is also given in sql/table_creation_queries.sql.

Note: some of the field types were changed from their initial values, as presented in the Stage 2 schema and UML diagram, to more effectively represent our data. For example, we changed:

- stop_id from VARCHAR(255) to INT in StopTimes, Stops

since this made importing the data from csv to MySQL much easier.

We also changed our representation of the departure (start) and arrival (end) stops in the *Paths* table, due to complications arising with trips containing multiple instances of the same stop. In Stage 2, our *Paths* table used *departure_stop, arrival_stop*, intended to hold the stop_id's of the departure and arrival stops. However, in our new, Stage 3 schema, the *Paths* table has replaced these with *departure_sequence*, *arrival_sequence*, respectively, to represent the sequence number of the departure and arrival stops (within their respective trip) as they are given in the *StopTimes* table.

```
CREATE TABLE Routes (
route_id VARCHAR(255) PRIMARY KEY,
route_type INT,
route_color CHAR(6),
route_long_name VARCHAR(255)
);

CREATE TABLE Stops (
stop_id INT PRIMARY KEY, -- data type differs from stage 2; see note above
stop_name VARCHAR(255),
stop_lat REAL,
stop_lon REAL
);

CREATE TABLE Calendar (
service_id VARCHAR(255) PRIMARY KEY,
monday INT,
tuesday INT,
wednesday INT,
thursday INT,
friday INT,
saturday INT,
sunday INT,
start_date DATE,
end_date DATE
```

```
);

CREATE TABLE Frequencies (
trip_id VARCHAR(255),
start_time TIME,
end_time TIME,
headway_secs INT,
PRIMARY KEY (trip_id, start_time, end_time),
FOREIGN KEY(trip_id) REFERENCES Trips(trip_id)
);

CREATE TABLE Trips (
trip_id VARCHAR(255) PRIMARY KEY,
route_id VARCHAR(255),
service_id VARCHAR(255),
trip_headsign VARCHAR(255),
direction_id INT,
FOREIGN KEY(route_id) REFERENCES Routes(route_id),
FOREIGN KEY(service_id) REFERENCES Calendar(service_id)
);

CREATE TABLE StopTimes (
trip_id VARCHAR(255),
stop_sequence INT,
stop_id INT, -- data type differs from stage 2; see note above
arrival_time TIME,
departure_time TIME,
PRIMARY KEY (trip_id, stop_sequence),
FOREIGN KEY(trip_id) REFERENCES Trips(trip_id),
FOREIGN KEY(stop_id) REFERENCES Stops(stop_id)
);

CREATE TABLE Users (
user_id VARCHAR(255) PRIMARY KEY,
email VARCHAR(255),
password VARCHAR(255),
first_name VARCHAR(255),
last_name VARCHAR(255)
);

CREATE TABLE Paths (
path_id VARCHAR(255) PRIMARY KEY,
```

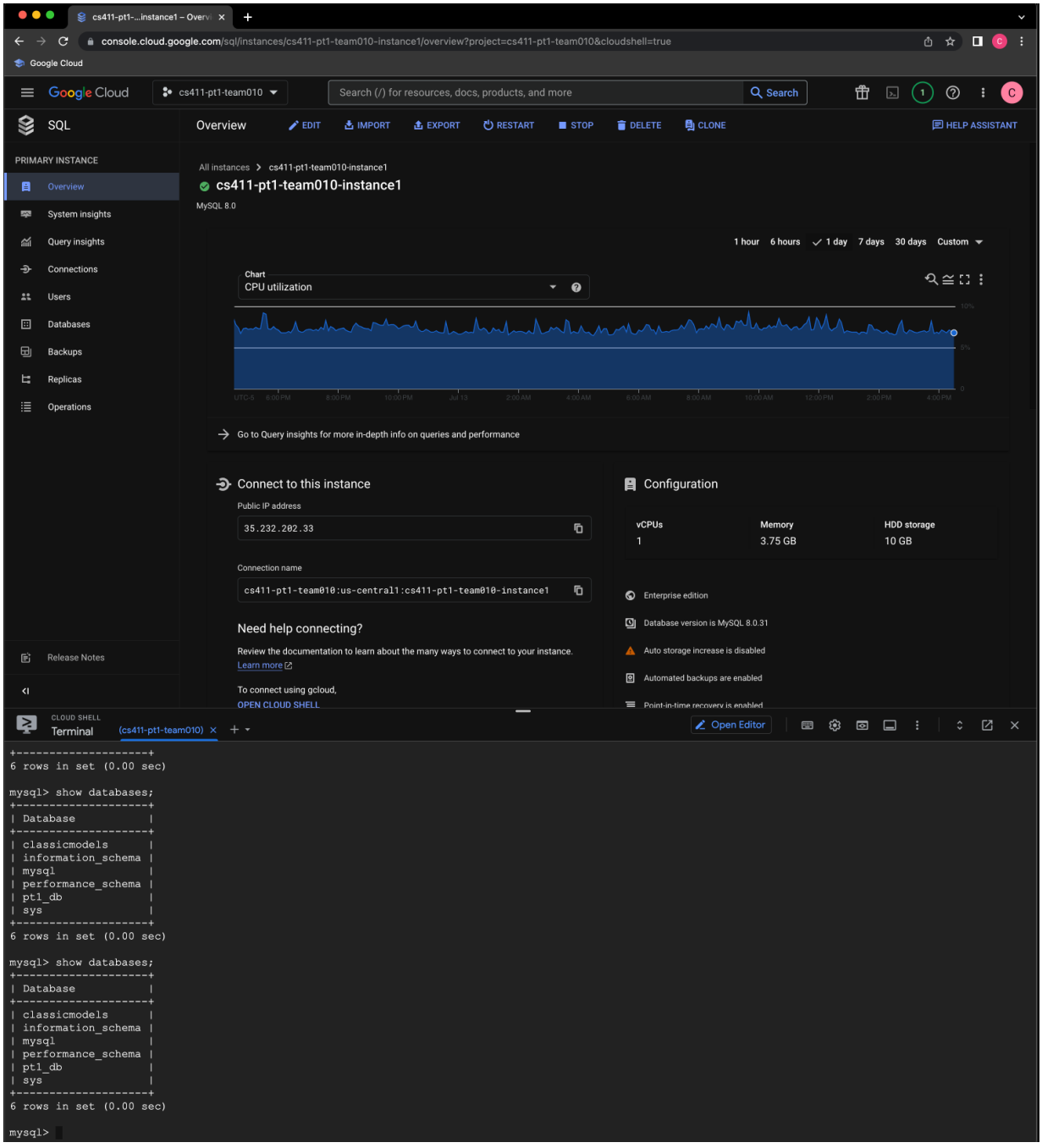
```
trip_id VARCHAR(255),
departure_sequence INT,
arrival_sequence INT,
departure_time TIME,
arrival_time TIME,

-- Foreign keys differ from stage 2; see note above
FOREIGN KEY(trip_id, departure_sequence) REFERENCES StopTimes(trip_id, stop_sequence),
FOREIGN KEY(trip_id, arrival_sequence) REFERENCES StopTimes(trip_id, stop_sequence)
);

CREATE TABLE Saved (
user_id VARCHAR(255),
path_id VARCHAR(255),
color CHAR(6),
PRIMARY KEY (user_id, path_id),
FOREIGN KEY(user_id) REFERENCES Users(user_id),
FOREIGN KEY(path_id) REFERENCES Paths(path_id)
);
```

Screenshots of database connection, Cloud Terminal

Database instance



9 tables within our pt1_db database:

```
mysql> use pt1_db;
Database changed
```

```
mysql> show tables;
+------------------+
| Tables_in_pt1_db |
+------------------+
| Calendar         |
| Frequencies      |
| Paths            |
| Routes           |
| Saved            |
| StopTimes        |
| Stops            |
| Trips            |
| Users            |
+------------------+
9 rows in set (0.01 sec)
```

```
mysql> describe Calendar;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| service_id | varchar(255) | NO   | PRI | NULL    |       |
| monday     | int          | YES  |     | NULL    |       |
| tuesday    | int          | YES  |     | NULL    |       |
| wednesday  | int          | YES  |     | NULL    |       |
| thursday   | int          | YES  |     | NULL    |       |
| friday     | int          | YES  |     | NULL    |       |
| saturday   | int          | YES  |     | NULL    |       |
| sunday     | int          | YES  |     | NULL    |       |
| start_date | date         | YES  |     | NULL    |       |
| end_date   | date         | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
10 rows in set (0.01 sec)
```

```
mysql> describe Frequencies;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| trip_id      | varchar(255) | NO   | PRI | NULL    |       |
| start_time   | time         | NO   | PRI | NULL    |       |
| end_time     | time         | NO   | PRI | NULL    |       |
| headway_secs | int          | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

```
mysql> describe Paths;
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| path_id            | varchar(255) | NO   | PRI | NULL    |       |
| trip_id            | varchar(255) | YES  | MUL | NULL    |       |
| departure_sequence | int          | YES  |     | NULL    |       |
| arrival_sequence   | int          | YES  |     | NULL    |       |
| departure_time     | time         | YES  |     | NULL    |       |
| arrival_time       | time         | YES  |     | NULL    |       |
+--------------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
mysql> describe Routes;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| route_id        | varchar(255) | NO   | PRI | NULL    |       |
| route_type      | int          | YES  |     | NULL    |       |
| route_color     | char(6)      | YES  |     | NULL    |       |
| route_long_name | varchar(255) | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> describe Saved;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| user_id | varchar(255) | NO   | PRI | NULL    |       |
| path_id | varchar(255) | NO   | PRI | NULL    |       |
| color   | char(6)      | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> describe StopTimes;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| trip_id        | varchar(255) | NO   | PRI | NULL    |       |
| stop_sequence  | int         | NO   | PRI | NULL    |       |
| stop_id        | int         | YES  | MUL | NULL    |       |
| arrival_time   | time        | YES  |     | NULL    |       |
| departure_time | time        | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> describe Stops;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| stop_id   | int         | NO   | PRI | NULL    |       |
| stop_name | varchar(255) | YES  |     | NULL    |       |
| stop_lat  | double      | YES  |     | NULL    |       |
| stop_lon  | double      | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> describe Trips;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| trip_id       | varchar(255) | NO   | PRI | NULL    |       |
| route_id      | varchar(255) | YES  | MUL | NULL    |       |
| service_id    | varchar(255) | YES  | MUL | NULL    |       |
| trip_headsign | varchar(255) | YES  |     | NULL    |       |
| direction_id  | int         | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> describe Users;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| user_id    | varchar(255) | NO   | PRI | NULL    |       |
| email      | varchar(255) | YES  |     | NULL    |       |
| password   | varchar(255) | YES  |     | NULL    |       |
| first_name | varchar(255) | YES  |     | NULL    |       |
| last_name  | varchar(255) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

3 tables with 1000+ rows:

```
mysql> SELECT COUNT(*) FROM StopTimes;
+----------+
| COUNT(*) |
+----------+
|    95265 |
+----------+
1 row in set (0.04 sec)
```

```
mysql> SELECT COUNT(*) FROM Stops;
+----------+
| COUNT(*) |
+----------+
|    20902 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM Trips;
+----------+
| COUNT(*) |
+----------+
|     2227 |
+----------+
1 row in set (0.00 sec)
```

2 Advanced Queries - result screenshots

Advanced Query 1 (source code given in sql/advanced_query_1_kaushik.sql)

```sql
/*
Given the day, day of the week, time, and two locations (lat/long), find all
the ways to get from one location to the other in one bus ride. Stops are
selected in a 0.002-degree square centered at each location.
Paths are returned by their trip_id's and the two relevant stop_sequence's.

Values are currently hardcoded, but will be replaced with inputted variables
in the final app.
*/
SELECT
    st1.trip_id AS trip_id,
    st1.stop_sequence AS departure_sequence,
    st2.stop_sequence AS arrival_sequence
FROM StopTimes st1
    JOIN StopTimes st2 USING(trip_id)
    JOIN Stops s1 ON (st1.stop_id = s1.stop_id)
    JOIN Stops s2 ON (st2.stop_id = s2.stop_id)
WHERE
    trip_id IN (
        SELECT DISTINCT trip_id
        FROM Trips NATURAL JOIN Frequencies NATURAL JOIN Calendar
        WHERE start_time <= '10:30:00'
        AND '10:30:00' <= end_time
        AND Calendar.saturday = 1
        AND start_date <= '2022-01-01'
        AND '2022-01-01' <= end_date
    )
    AND s1.stop_lat < -23.446258 + 0.001 AND s1.stop_lat > -23.446258 - 0.001
    AND s1.stop_lon < -46.712298 + 0.001 AND s1.stop_lon > -46.712298 - 0.001
    AND s2.stop_lat < -23.447209 + 0.001 AND s2.stop_lat > -23.447209 - 0.001
    AND s2.stop_lon < -46.709039 + 0.001 AND s2.stop_lon > -46.709039 - 0.001
    AND st1.stop_sequence <= st2.stop_sequence
ORDER BY st2.stop_sequence - st1.stop_sequence ASC
LIMIT 15;
```

trip_id	departure_sequence	arrival_sequence
1021-10-0	3	5
1021-10-0	2	5
1021-10-0	3	6
1021-10-1	22	25
1036-10-1	20	23
9009-10-1	32	35
971C-10-1	54	57
1021-10-0	2	6
1036-10-1	20	25
9009-10-1	32	37

10 rows in set (0.17 sec)

Advanced Query 2 (source code given in sql/AQ_2_colin/advanced_query_2_colin.sql)

```sql
-- Functionality: return the saved paths for a given user subject to a filter that the user can apply
-- which shows only the paths which are available on certain days.
-- This query returns the necessary data for our "My Trips" page of our application, but just with an
-- added bonus of complying with the aforementioned "days" filter.

-- BACKEND (NodeJS) VARIABLES
    -- * the_user_id * ==  the id of the user currently using the application.
    -- * days * == the id of the user currently using the application. This is a 7-digit numeric binary array,
    -- of 0's and 1's, where a 1 represents "true" - that a trip MUST be on this day - and where a 0
    -- represents "false" -- where it doesn't matter if a trip is on this day.


SELECT s.color, p.departure_stop, p.arrival_stop, p.departure_time, p.arrival_time
FROM Paths p NATURAL JOIN Saved s
WHERE s.user_id = the_user_id AND p.path_id IN (
    SELECT p1.path_id
    FROM Paths p1 NATURAL JOIN Trips t1 NATURAL JOIN Calendar c1
    WHERE (
        (days[0] = 0 OR c1.Monday = 1) AND
        (days[1] = 0 OR c1.Tuesday = 1) AND
        (days[2] = 0 OR c1.Wednesday = 1) AND
        (days[3] = 0 OR c1.Thursday = 1) AND
        (days[4] = 0 OR c1.Friday = 1) AND
        (days[5] = 0 OR c1.Saturday = 1) AND
        (days[6] = 0 OR c1.Sunday = 1)
    )
)
;

-- USING DUMMY VALUES:
    -- the_user_id = 1
    -- days = (0,1,1,0,0,1,0)

-- day-checking part becomes:
    -- (0 = 0 OR c1.Monday = 1) AND
    -- (1 = 0 OR c1.Tuesday = 1) AND
    -- (1 = 0 OR c1.Wednesday = 1) AND
    -- (0 = 0 OR c1.Thursday = 1) AND
    -- (0 = 0 OR c1.Friday = 1) AND
    -- (1 = 0 OR c1.Saturday = 1) AND
    -- (0 = 0 OR c1.Sunday = 1)
-- so I simplify it in the query below

SELECT s.color, p.departure_sequence, p.arrival_sequence, p.departure_time, p.arrival_time
FROM Paths p NATURAL JOIN Saved s
WHERE s.user_id = '1' AND p.path_id IN (
    SELECT p1.path_id
    FROM Paths p1 NATURAL JOIN Trips t1 NATURAL JOIN Calendar c1
    WHERE (
        (c1.Tuesday = 1) AND
        (c1.Wednesday = 1) AND
        (c1.Saturday = 1)
    )
)
LIMIT 15
;

-- This is the query that'll be run in the demo.
-- Note: we will actually have to write more SQL code for the actual applicaiton to work properly (
    -- currently this only returns departure sequence numbers and trip_id's, not the actual stop
    -- latitudes, longitudes, and names, which are what we would actually need for this
    -- functionality to work properly in "My Trips", but for now this part of the query works as
    -- A.Q. 2.)

```

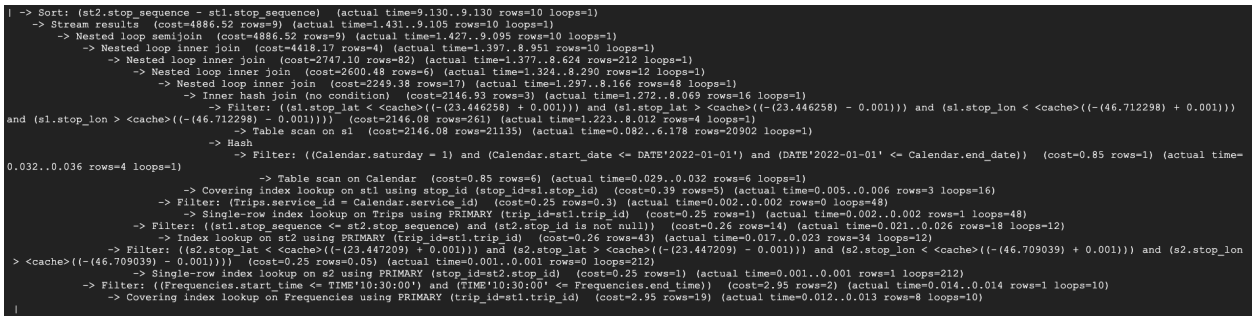
color	departure_sequence	arrival_sequence	departure_time	arrival_time
04263F	2	13	12:01:22	12:16:24
D06687	7	14	17:15:30	17:33:35
309AF6	4	10	07:05:54	07:17:42
E706F3	5	9	17:08:48	17:17:36
ECEAFE	3	14	17:04:36	17:29:54
8A405C	7	9	17:16:06	17:21:28
F45312	7	12	06:12:12	06:22:22
09FE2B	10	14	17:16:30	17:23:50
519B41	4	11	18:05:18	18:17:40
9C3107	5	10	18:07:08	18:16:03
CC3595	5	9	05:06:00	05:12:00
C435E4	4	12	07:04:45	07:17:25
9CF69A	7	10	18:11:00	18:16:30
5D6EF9	5	11	07:08:32	07:21:20
AFEEEC	5	13	16:05:00	16:15:00

15 rows in set (0.01 sec)

Index Analysis

Advanced Query 1 Index Analysis:

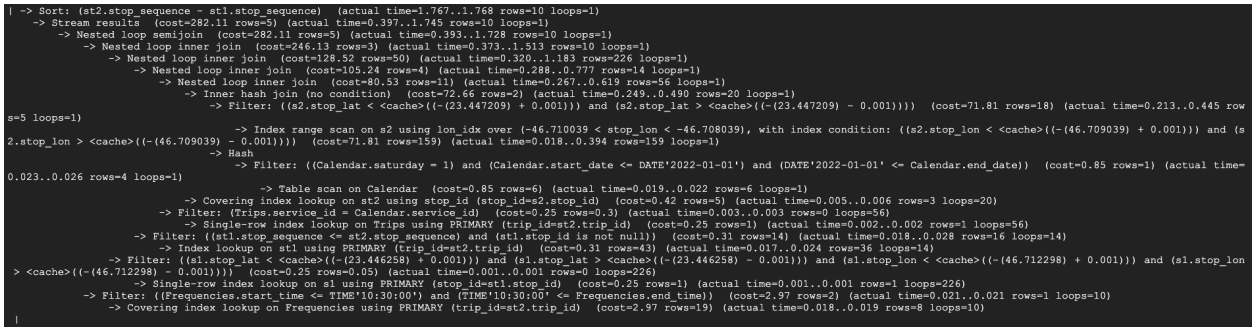
0. Results of EXPLAIN ANALYZE before adding indices



Cost: 4886.52
Actual time: 1.431 .. 9.105

1.

```
CREATE INDEX lon_idx ON Stops(stop_lon);
```

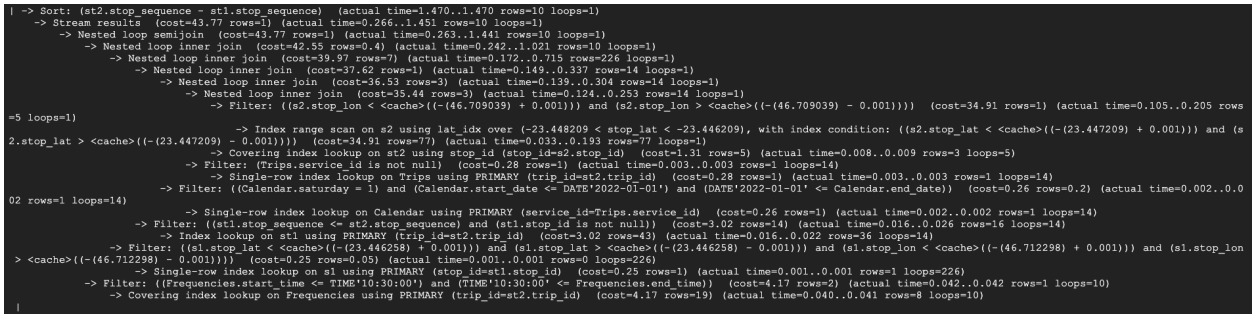


Cost: 282.11
Actual time: 0.397 .. 1.745

The original query involves two range queries, for Stops.stop_lat and Stops.stop_lon. These have the effect of creating a square around the selected coordinates where stops may be located. Neither of these attributes are primary keys, so they don't get indices by default. Since indices are very helpful for range queries, we first add an index to Stops(stop_lon), and the results are shown above. The cost is reduced dramatically, since the query can use the logarithmic-time search of a B+ tree rather than the linear-time search of going through every Stop looking for appropriate longitudes.

2.

```
CREATE INDEX lon_idx ON Stops(stop_lon); -- from previous setup
CREATE INDEX lat_idx ON Stops(stop_lat);
```

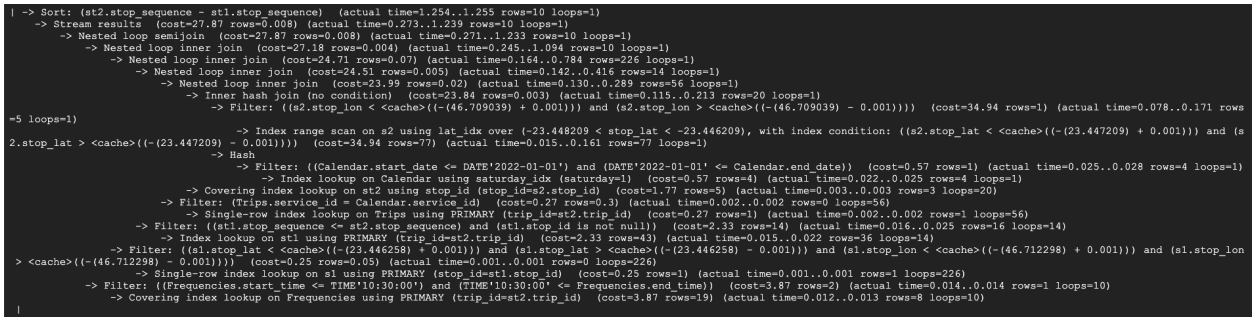


Cost: 43.77
Actual time: 0.266 .. 1.451

Now we do the same for latitudes that we did for longitudes, with the same reasoning; indices make range queries much faster because of the efficient B+ tree search and traversal algorithm. The cost once again decreases significantly. However, the difference is not as dramatic proportionally, which may be due to the specific latitude and longitude values chosen. Nevertheless, having indices for both latitude and longitude is incredibly useful for location-based queries, and will definitely be a part of the final application.

3.

```
CREATE INDEX lon_idx ON Stops(stop_lon); -- from previous setup
CREATE INDEX lat_idx ON Stops(stop_lat); -- from previous setup
CREATE INDEX saturday_idx ON Calendar(saturday);
```



Cost: 27.87
Actual time: 0.273 .. 1.239

The rest of the attributes in the query are mostly already indexed due to them being primary keys. One attribute that isn't a primary key is Calendar.saturday, which is used to check if a trip's schedule has it running on Saturdays. Since Calendar only has six rows, the index is not as immediately effective as the previous indices, but it still had a modest impact on query cost. In a query that involves multiple days of the week (like the next one), it may become useful to have indices for all days of the week, since even if the table is small, a very small performance increase becomes very useful when a subquery is run many times.

Advanced Query 2 Index Analysis:

0. Results of EXPLAIN ANALYZE on A.Q.2 *before* the addition of any indices:

```
| EXPLAIN
                                                                                                  |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
| -> Limit: 15 row(s)  (cost=48.25 rows=14) (actual time=0.097..0.466 rows=15 loops=1)
    -> Nested loop inner join  (cost=48.25 rows=14) (actual time=0.096..0.464 rows=15 loops=1)
        -> Nested loop inner join  (cost=33.90 rows=41) (actual time=0.085..0.350 rows=67 loops=1)
            -> Nested loop inner join  (cost=19.55 rows=41) (actual time=0.077..0.210 rows=67 loops=1)
                -> Inner hash join (no condition)  (cost=5.20 rows=41) (actual time=0.067..0.083 rows=67 loops=1)
                    -> Table scan on p  (cost=4.35 rows=41) (actual time=0.024..0.030 rows=34 loops=1)
                    -> Hash
                        -> Filter: ((c1.saturday = 1) and (c1.wednesday = 1) and (c1.tuesday = 1))  (cost=0.85 rows=1) (actual time=0.032..0.034 rows=2 loops=1)
                            -> Table scan on c1  (cost=0.85 rows=6) (actual time=0.027..0.031 rows=6 loops=1)
                -> Single-row index lookup on s using PRIMARY (user_id='1', path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=67)
            -> Filter: (p1.trip_id is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=67)
                -> Single-row index lookup on p1 using PRIMARY (path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
        -> Filter: (t1.service_id = c1.service_id)  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=67)
            -> Single-row index lookup on t1 using PRIMARY (trip_id=p1.trip_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
 |
```

Key metrics:

- *cost=48.25*
- *actual time=0.097…0.466*

1. Results of EXPLAIN ANALYZE on A.Q.2 after adding 1st set of indecies

```sql
CREATE INDEX mIdx ON Calendar(monday);
CREATE INDEX tIdx ON Calendar(tuesday);
CREATE INDEX wIdx ON Calendar(wednesday);
CREATE INDEX thIdx ON Calendar(thursday);
CREATE INDEX fIdx ON Calendar(friday);
CREATE INDEX sIdx ON Calendar(saturday);
CREATE INDEX suIdx ON Calendar(sunday);
```

```
| -> Limit: 15 row(s)  (cost=48.00 rows=14) (actual time=0.136..0.492 rows=15 loops=1)
    -> Nested loop inner join  (cost=48.00 rows=14) (actual time=0.135..0.490 rows=15 loops=1)
        -> Nested loop inner join  (cost=33.65 rows=41) (actual time=0.120..0.344 rows=67 loops=1)
            -> Nested loop inner join  (cost=19.30 rows=41) (actual time=0.099..0.225 rows=67 loops=1)
                -> Inner hash join (no condition)  (cost=4.95 rows=41) (actual time=0.086..0.103 rows=67 loops=1)
                    -> Table scan on p  (cost=4.35 rows=41) (actual time=0.032..0.039 rows=34 loops=1)
                    -> Hash
                        -> Filter: ((c1.saturday = 1) and (c1.wednesday = 1))  (cost=0.60 rows=1) (actual time=0.041..0.042 rows=2 loops=1)
                            -> Index lookup on c1 using tIdx (tuesday=1)  (cost=0.60 rows=3) (actual time=0.035..0.038 rows=3 loops=1)
                -> Single-row index lookup on s using PRIMARY (user_id='1', path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=67)
            -> Filter: (p1.trip_id is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=67)
                -> Single-row index lookup on p1 using PRIMARY (path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
        -> Filter: (t1.service_id = c1.service_id)  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=67)
            -> Single-row index lookup on t1 using PRIMARY (trip_id=p1.trip_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=67)
 |
```

2. Results of EXPLAIN ANALYZE on A.Q.2 after adding 2nd set of indices (in addition to 1st)

```sql
CREATE INDEX userIdIdx ON Saved(user_id);
```

```
| -> Limit: 15 row(s)  (cost=48.00 rows=14) (actual time=0.093..0.428 rows=15 loops=1)
    -> Nested loop inner join  (cost=48.00 rows=14) (actual time=0.092..0.426 rows=15 loops=1)
        -> Nested loop inner join  (cost=33.65 rows=41) (actual time=0.081..0.310 rows=67 loops=1)
            -> Nested loop inner join  (cost=19.30 rows=41) (actual time=0.075..0.198 rows=67 loops=1)
                -> Inner hash join (no condition)  (cost=4.95 rows=41) (actual time=0.066..0.082 rows=67 loops=1)
                    -> Table scan on p  (cost=4.35 rows=41) (actual time=0.025..0.031 rows=34 loops=1)
                    -> Hash
                        -> Filter: ((c1.saturday = 1) and (c1.wednesday = 1))  (cost=0.60 rows=1) (actual time=0.030..0.032 rows=2 loops=1)
                            -> Index lookup on c1 using tIdx (tuesday=1)  (cost=0.60 rows=3) (actual time=0.026..0.029 rows=3 loops=1)
                -> Single-row index lookup on s using PRIMARY (user_id='1', path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=67)
            -> Filter: (p1.trip_id is not null)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=67)
                -> Single-row index lookup on p1 using PRIMARY (path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
        -> Filter: (t1.service_id = c1.service_id)  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=67)
            -> Single-row index lookup on t1 using PRIMARY (trip_id=p1.trip_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
 |
```

3. Results of EXPLAIN ANALYZE on A.Q.2 after adding 3rd set of indices (in addition to 1st, 2nd)

```sql
CREATE INDEX pPiIdx ON Paths(path_id);
```

```
CREATE INDEX pTiIdx ON Paths(trip_id);

CREATE INDEX pDsIdx ON Paths(departure_sequence);

CREATE INDEX pAsIdx ON Paths(arrival_sequence);

CREATE INDEX pDtIdx ON Paths(departure_time);

CREATE INDEX pAtIdx ON Paths(arrival_time);
```

```
| -> Limit: 15 row(s)  (cost=48.00 rows=14) (actual time=0.090..0.409 rows=15 loops=1)
    -> Nested loop inner join  (cost=48.00 rows=14) (actual time=0.088..0.406 rows=15 loops=1)
        -> Nested loop inner join  (cost=33.65 rows=41) (actual time=0.075..0.290 rows=67 loops=1)
            -> Nested loop inner join  (cost=19.30 rows=41) (actual time=0.070..0.195 rows=67 loops=1)
                -> Inner hash join (no condition)  (cost=4.95 rows=41) (actual time=0.062..0.078 rows=67 loops=1)
                    -> Table scan on p  (cost=4.35 rows=41) (actual time=0.026..0.032 rows=34 loops=1)
                    -> Hash
                        -> Filter: ((c1.saturday = 1) and (c1.wednesday = 1))  (cost=0.60 rows=1) (actual time=0.024..0.026 rows=2 loops=1)
                            -> Index lookup on c1 using tIdx (tuesday=1)  (cost=0.60 rows=3) (actual time=0.020..0.022 rows=3 loops=1)
                -> Single-row index lookup on s using PRIMARY (user_id='1', path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=67)
            -> Filter: (p1.trip_id is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
                -> Single-row index lookup on p1 using PRIMARY (path_id=p.path_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
        -> Filter: (t1.service_id = c1.service_id)  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=67)
            -> Single-row index lookup on t1 using PRIMARY (trip_id=p1.trip_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=67)
|
```

The results of the 3 index designs clearly show that only the first added query was effective in decreasing the cost of executing advanced query 2. This makes sense for a few reasons. Firstly, the indexes added in Index Design 1 were able to replace the table scan on c1, which occur when the query filters the Calendar table in accordance with the days selected by the user in the filter provided in the My Trips part of our application, for certain “day of the week” attribute values being 1. We see a decrease in cost of 0.25, which is definitely nonzero, but is ultimately small due to the fact that Calendar is a relatively small table itself, with only 6 rows (and 10 attributes.)

The indexes added in Index Design 2 and Index Design 3 did not alter the cost of executing the query. This makes sense, as their addition did not alter any part of the executed query plan - as seen in the screenshots of the EXPLAIN ANALYZE outputs for Index Design 2 and Index Design 3. The cost of advanced query 2 is mainly comprised of:

1. The joining of multiple tables, which, unfortunately, cannot be made less costly here by the addition of indexes on specific attributes of the involved tables. Specifically, the executed query plan maintained its use of numerous nested loop inner joins, as well as a inner hash join. These joins have high cost and contribute a lot to the total cost.
2. Filters using single-row index lookups on *primary keys*. Indexes are created automatically by the MySQL database for all primary keys, and so the addition of indexes would not speed anything up here.

Thus, our team decided to keep only the indexes added in Index Design 1 in our final set of indexes for our database. Their speedup, though small, is nonzero.