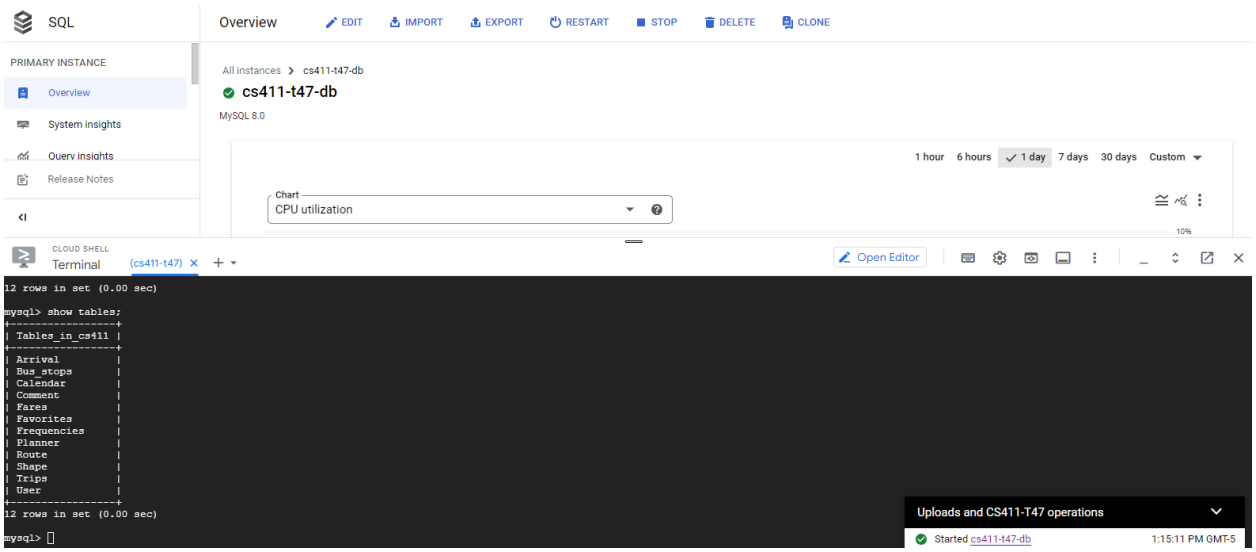
Stage 3 - Database Design

PART 1

Implementation of the database tables on GCP



Data Definition Language (DDL) commands:

```
CREATE TABLE `cs411`.`Calendar` (
  `service_id` VARCHAR(3) NOT NULL,
  `monday` INT NOT NULL,
  `tuesday` INT NOT NULL,
  `wednesday` INT NOT NULL,
  `friday` INT NOT NULL,
  `sunday` INT NOT NULL,
  `start_date` INT NULL,
  `end_date` INT NULL,
  PRIMARY KEY (`service_id`)
);

CREATE TABLE `cs411`.`Bus_stops` (
  `stop_id` INT NOT NULL,
```

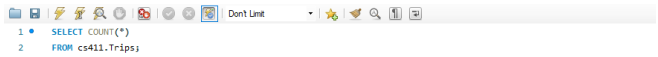
```
  `stop_name` VARCHAR(256) NOT NULL,
  `stop_desc` VARCHAR(256) NULL,
  `stop_lat` REAL NOT NULL,
  `stop_lon` REAL NOT NULL,
  PRIMARY KEY (`stop_id`));

CREATE TABLE `cs411`.`Route` (
  `route_id` VARCHAR(9) NOT NULL,
  `route_long_name` VARCHAR(256) NULL,
  `route_color` VARCHAR(30) NULL,
  `route_text_color` VARCHAR(30) NULL,
  `fare_id` VARCHAR(21) NOT NULL,
   PRIMARY KEY (`route_id`))
   FOREIGN KEY (`fare_id`)  REFERENCES `cs411`.`Fares` (`fare_id`)
     ON DELETE CASCADE
     ON UPDATE CASCADE);

CREATE TABLE `cs411`.`Shape` (
  `shape_id` INT NOT NULL,
  `shape_pt_lat` REAL NULL,
  `shape_pt_lon` REAL NULL,
  `shape_pt_sequence` INT NOT NULL,
  `shape_dist_traveled` REAL NULL,
  PRIMARY KEY (`shape_id`, `shape_pt_sequence`)
 FOREIGN KEY (`shape_id`)  REFERENCES `cs411`.`Trips` (`shape_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE);

CREATE TABLE `cs411`.`Fares` (
  `fare_id` VARCHAR(23) NOT NULL,
  `price` REAL NULL,
  `currency_type` VARCHAR(5) NULL,
  `payment_method` INT NULL,
  `transfers` VARCHAR(3) NULL,
  `transfer_duration` INT NULL,
  PRIMARY KEY (`fare_id`));
```

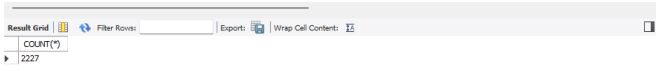
```
CREATE TABLE `cs411`.`Trips` (
  `trip_id` VARCHAR(256) NOT NULL,
  `route_id` VARCHAR(256) NOT NULL,
  `service_id` VARCHAR(5) NOT NULL,
  `trip_headsign` VARCHAR(256) DEFAULT NULL,
  `direction_id` INT NULL,
  `shape_id` INT NOT NULL,
  PRIMARY KEY (`trip_id`),
  FOREIGN KEY (`route_id`) REFERENCES `cs411`.`Route`(`route_id`)
  ON DELETE NO ACTION
  ON UPDATE CASCADE,
  FOREIGN KEY (`service_id`) REFERENCES `cs411`.`Calendar` (`service_id`)
  ON DELETE RESTRICT
  ON UPDATE CASCADE
  FOREIGN KEY (`shape_id`) REFERENCES `cs411`.`Shape` (`shape_id`)
  ON DELETE RESTRICT
  ON UPDATE CASCADE

);

CREATE TABLE `cs411`.`User` (
  `username` VARCHAR(17) NOT NULL,
  `password` VARCHAR(17) NOT NULL,
  `favorites_id` INT DEFAULT NULL,
  `email` VARCHAR(42) NOT NULL,
  PRIMARY KEY (`email`),
  FOREIGN KEY (`favorites_id`)
    REFERENCES `cs411`.`Favorites` (`favorites_id`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
);


CREATE TABLE `cs411`.`Frequencies` (
  `trip_id` VARCHAR(13) NOT NULL,
```

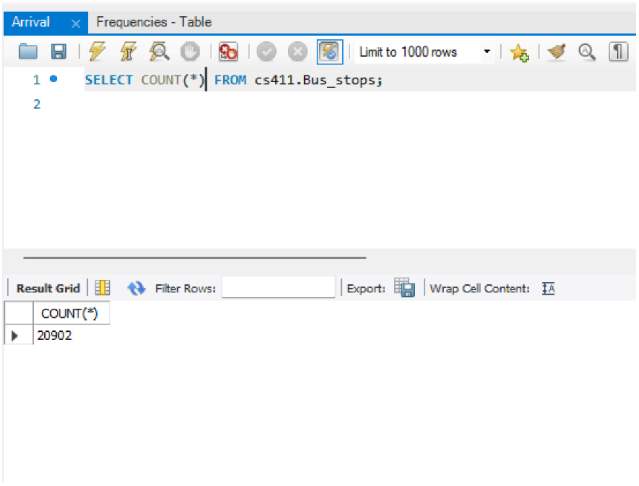
```
  `headway_secs` INT DEFAULT NULL,
  `start_time` VARCHAR(10) NOT NULL,
  `stop_time` VARCHAR(10) DEFAULT NULL,
  PRIMARY KEY (`trip_id`, `start_time`),
  FOREIGN KEY (`trip_id`)
    REFERENCES `cs411`.`Trips` (`trip_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE);


CREATE TABLE `cs411`.`Arrival` (
  `trip_id` VARCHAR(13) NOT NULL,
  `stop_id` INT NOT NULL,
  `arrival_time` VARCHAR(10) DEFAULT NULL,
  `stop_sequence` INT NULL,
  PRIMARY KEY (`trip_id`, `stop_id`),
  FOREIGN KEY (`trip_id`) REFERENCES `cs411`.`Trips`(`trip_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (`stop_id`) REFERENCES `cs411`.`Bus_stops`(`stop_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);


CREATE TABLE `cs411`. `Planner` (
  `planner_id` INT NOT NULL,
  `plan_stops` MEDIUMTEXT,
  `email` VARCHAR(42) DEFAULT NULL,
  PRIMARY KEY (`planner_id`),
  FOREIGN KEY (`email`) REFERENCES`cs411`. `User` (`email`)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE `cs411`.`Comment` (
```

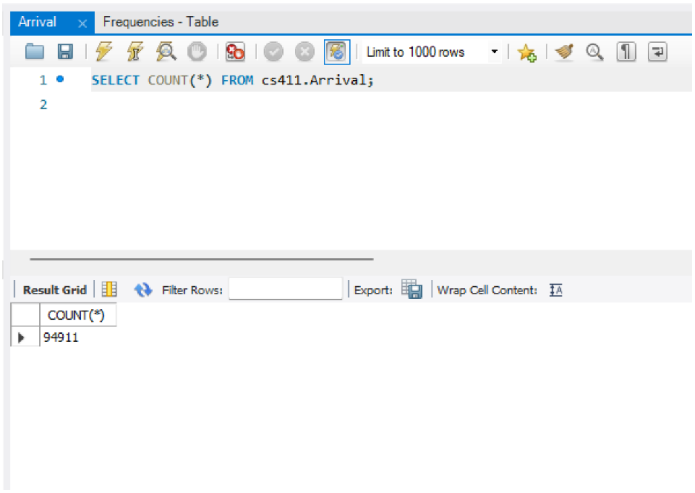
```
 `email` VARCHAR(42) NOT NULL,
 `route_id` VARCHAR(9) NOT NULL,
 `crowdedness` TEXT NULL,
 `safety` TEXT NULL,
 `temperature` TEXT NULL,
 `accessibility` TEXT NULL,
 PRIMARY KEY (`email`, `route_id`),
 FOREIGN KEY (`email`) REFERENCES`cs411`. `User` (`email`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
 FOREIGN KEY (`route_id`) REFERENCES `cs411`.`Route`(`route_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);

CREATE TABLE `cs411`.`Favorites` (
 `favorite_id` INT NOT NULL,
 `favorite_stops` MEDIUMTEXT,
 `favorite_routes` MEDIUMTEXT,
 PRIMARY KEY (`favorite_id`)
);
```

3 Tables with at least 1000 rows:









Advanced SQL Queries:

Query 1: With a given location(latitude and longitude), we will return the stops within 5km radius.

```
SELECT
  bs.stop_id,
  bs.stop_name,
  r.route_id,
  r.route_long_name,
  COUNT(t.trip_id) AS trip_count,
  (
    6371 * ACOS(
      COS(RADIANS(-23.612613)) * COS(RADIANS(bs.stop_lat)) *
      COS(RADIANS(bs.stop_lon) - RADIANS(-46.476124)) +
      SIN(RADIANS(-23.612613)) * SIN(RADIANS(bs.stop_lat))
    )
```

```
    ) AS distance_km
FROM
    cs411.Bus_stops bs
JOIN
    cs411.Arrival a ON bs.stop_id = a.stop_id
JOIN
    cs411.Trips t ON a.trip_id = t.trip_id
JOIN
    cs411.Route r ON t.route_id = r.route_id
GROUP BY
    bs.stop_id,
    bs.stop_name,
    r.route_id,
    r.route_long_name
HAVING
    distance_km <= 5  -- Filter rows based on the calculated distance
ORDER BY
    distance_km ASC
LIMIT
    15;
```

stop_id	stop_name	route_id	route_long_name	trip_count	distance_km
18847	Metropolitano São Mateus - Tma	4735-10	Jd. Vera Cruz - São Mateus	1	0
750006773	Terminal S. Mateus - Plat. B	4013-10	Jd. Sto. André - Term. São Mateus	1	0.06359240774837156
750006773	Terminal S. Mateus - Plat. B	N431-11	Term. São Mateus - Jd. Limoeiro	1	0.06359240774837156
750006774	Terminal S. Mateus - Plat. B	N431-11	Term. São Mateus - Jd. Limoeiro	1	0.07229227879558178
750006774	Terminal S. Mateus - Plat. B	3069-10	Jd. Recanto Verde Sol - Term. São Mateus	1	0.07229227879558178
750006774	Terminal S. Mateus - Plat. B	4013-10	Jd. Sto. André - Term. São Mateus	2	0.07229227879558178
750006772	Terminal S. Mateus - Plat. A	4735-10	Jd. Vera Cruz - São Mateus	1	0.07481883296907274
750006772	Terminal S. Mateus - Plat. A	407F-10	Term. São Mateus - Metrô Belém	2	0.07481883296907274
750006792	Terminal S. Mateus - Plat. C	N503-11	Term. São Mateus - Term. Pq. D. Pedro Ii	2	0.08589278625370152
750006771	Terminal S. Mateus - Plat. A	407K-10	Term. São Mateus - Metrô Carrão	1	0.08632051677205038
750006771	Terminal S. Mateus - Plat. A	4056-10	Pq. Boa Esperança - Term. São Mateus	2	0.08632051677205038
750006771	Terminal S. Mateus - Plat. A	4735-10	Jd. Vera Cruz - São Mateus	1	0.08632051677205038
750006788	Terminal São Mateus	4735-10	Jd. Vera Cruz - São Mateus	1	0.09166744133021713
750006788	Terminal São Mateus	2290-10	Term. São Mateus - Term. Pq. D. Pedro Ii	2	0.09166744133021713
770006791	Terminal S. Mateus - Plat. C	N404-11	Term. São Mateus - Term. Penha	1	0.09417835162297465

Query 2:

This query gets all routes along with their stops and also shows on which days of the week each route has service.

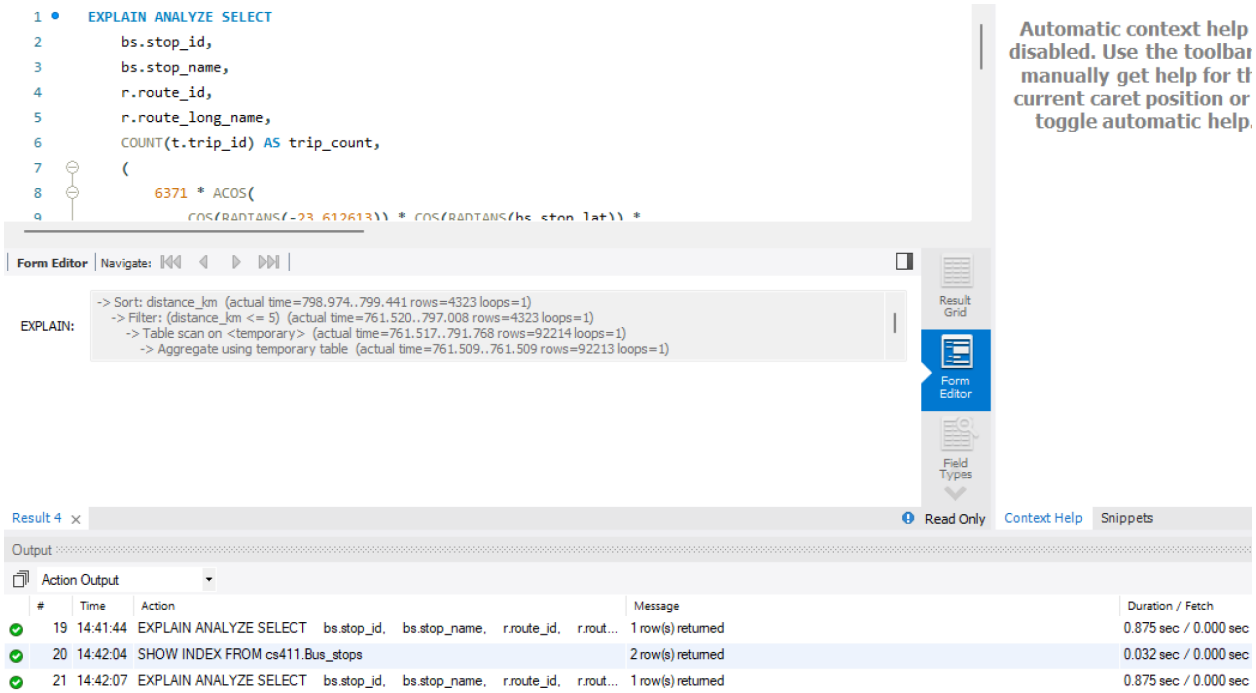
```
SELECT r.route_id, r.route_long_name,
  COALESCE((SELECT GROUP_CONCAT(day) FROM (
    SELECT 'monday' as day WHERE MAX(c.monday) = 1
    UNION SELECT 'tuesday' WHERE MAX(c.tuesday) = 1
    UNION SELECT 'wednesday' WHERE MAX(c.wednesday) = 1
    UNION SELECT 'thursday' WHERE MAX(c.thursday) = 1
    UNION SELECT 'friday' WHERE MAX(c.friday) = 1
    UNION SELECT 'saturday' WHERE MAX(c.saturday) = 1
    UNION SELECT 'sunday' WHERE MAX(c.sunday) = 1
  ) days), 'No Service') as service_days
FROM cs411.Route r
LEFT JOIN cs411.Trips t ON r.route_id = t.route_id
LEFT JOIN cs411.Calendar c ON t.service_id = c.service_id
GROUP BY r.route_id, r.route_long_name
ORDER BY r.route_id
LIMIT 15;
```

route_id	route_long_name	service_days
1012-10	Term. Jd. Britania - Jd. Monte Belo	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1015-10	Term. Jd. Britânia - Chác. Maria Trindade	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1016-10	Cem. Do Horto - Shop. Center Norte	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1017-10	Perus - Conexão Vl. Iório	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1018-10	Vl. Rosa - Metrô Santana	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1019-10	Sol Nascente - Term. Pirituba	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1020-10	Perus - Conexão Vl. Iório	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1021-10	Cohab Brasilândia - Term. Pirituba	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1023-10	Vl. Pirituba - Cptm Pirituba	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1024-10	Jd. Carombé - Conexão Petrônio Portela	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1025-10	Jd. Carombé - Conexão Petrônio Portela	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1026-10	Vl. Iara - Conexão Petrônio Portela	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1034-10	Vl. Iara - Conexão Vl. Iório	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1036-10	Cohab Brasilândia - Conexão Vl. Iório	monday,tuesday,wednesday,thursday,friday,saturday,sunday
106A-10	Metrô Santana - Itaim Bibi	monday,tuesday,wednesday,thursday,friday,saturday,sunday

PART 2

Index for Advanced Query 1:

For the default index, we got an analysis like this:



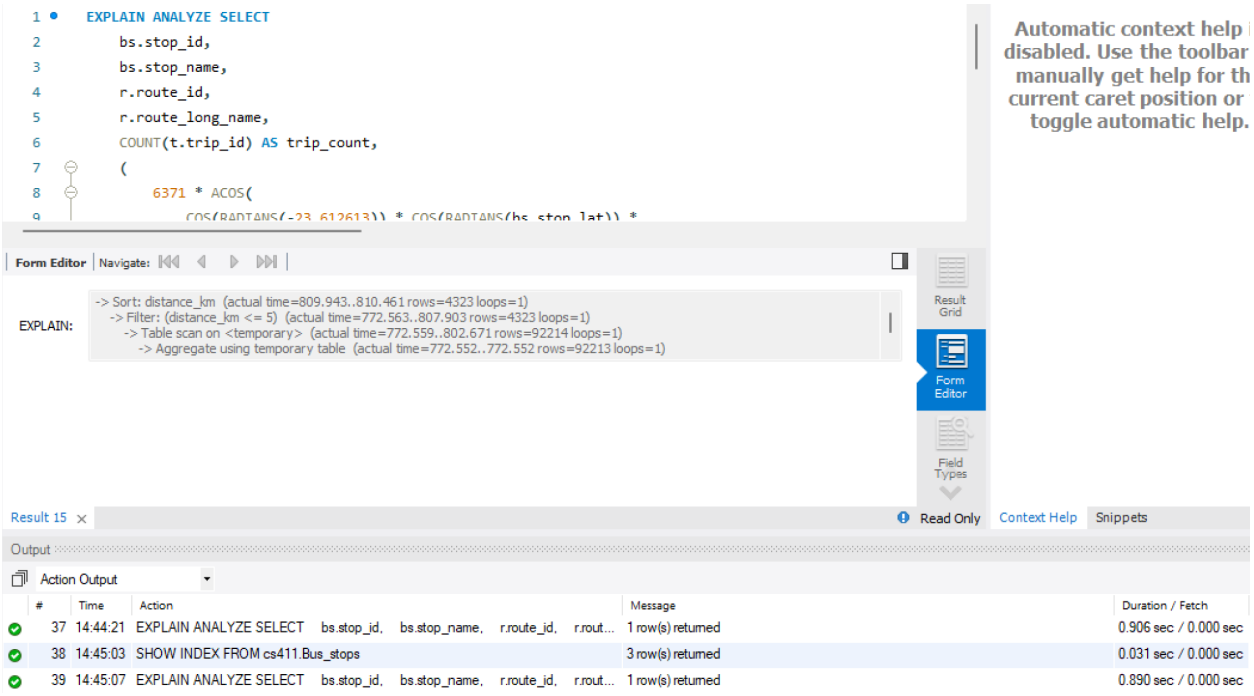
Design 1: This query uses longitude and latitude to calculate the distance, and this query is grouped by distance and has a HAVING filter with the distance. As such, we tried to add an index in Bus_stops.stop_lon first.

```
CREATE INDEX lon ON cs411.Bus_stops(stop_lon);
SHOW INDEX FROM cs411.Bus_stops;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

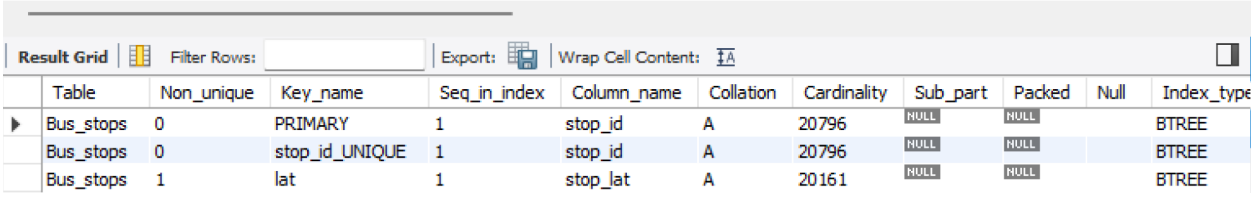
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
Bus_stops	0	PRIMARY	1	stop_id	A	20796	NULL	NULL		BTREE
Bus_stops	0	stop_id_UNIQUE	1	stop_id	A	20796	NULL	NULL		BTREE
Bus_stops	1	lon	1	stop_lon	A	20312	NULL	NULL		BTREE

Below is the analysis, and adding the index of longitude gives a very similar runtime as the original. It might have increased the runtime of the query a bit, thus decreasing the efficiency. This decrease in efficiency can be due to a potential lower selectivity in Bus_stops.stop_lon. The values in stop_lon are very similar, ranging from -46.3 to -46.8, so there might be a longer scan through the column. Furthermore, although the query is filtering based on distance, the distance calculation is not proportionally related to stop_lon, so the query can't filter based on the longitude index; therefore, this index design can't optimize the query significantly.

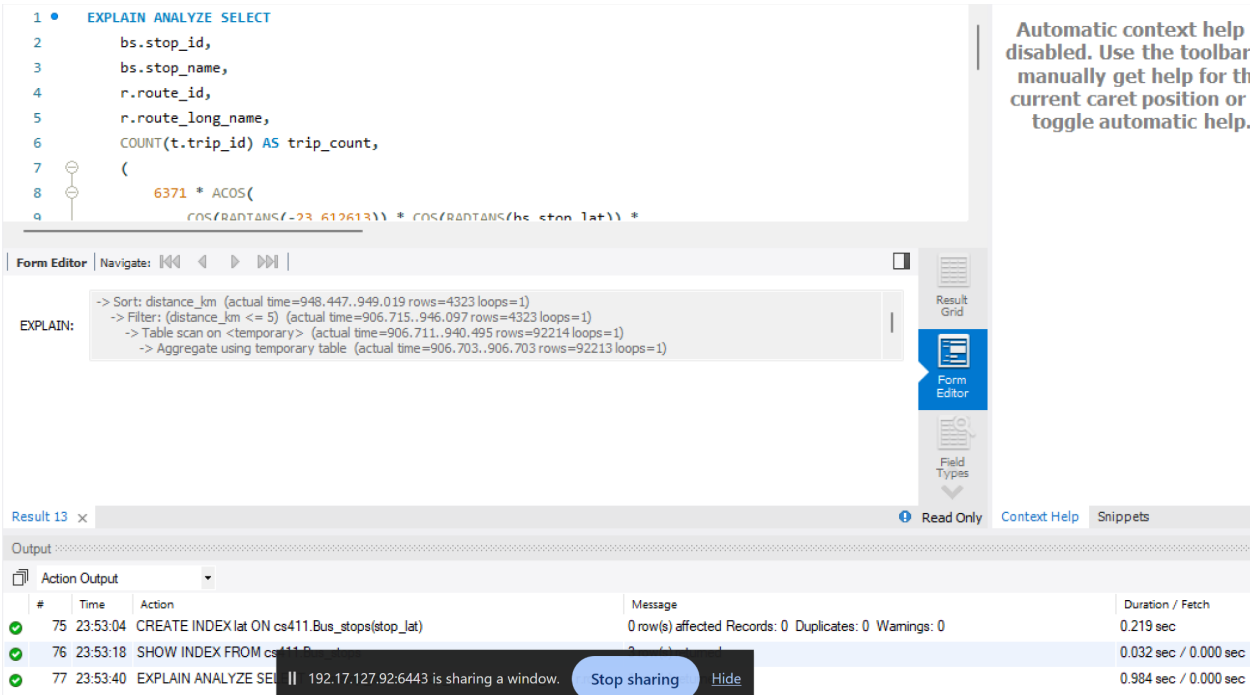


Design 2: This query uses longitude and latitude to calculate the distance, and this query is grouped by distance and has a HAVING filter with the distance. As such, we tried to add an index in Bus_stops.lat instead.

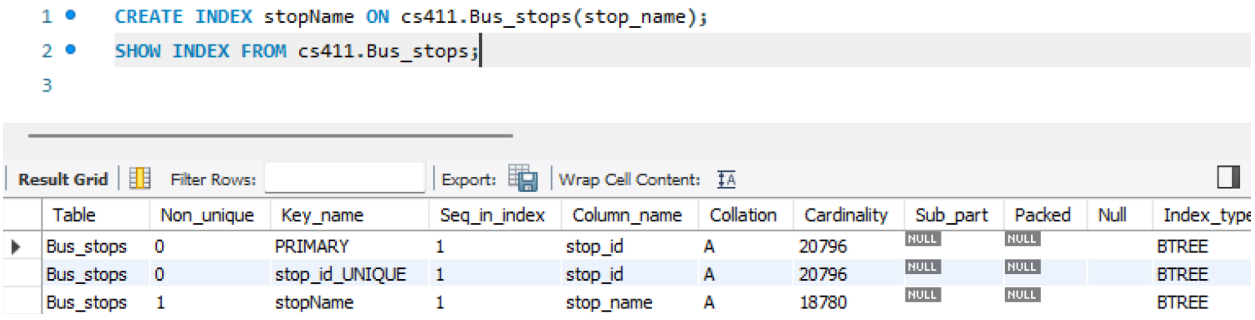
```
CREATE INDEX lat ON cs411.Bus_stops(stop_lat);
SHOW INDEX FROM cs411.Bus_stops;
```



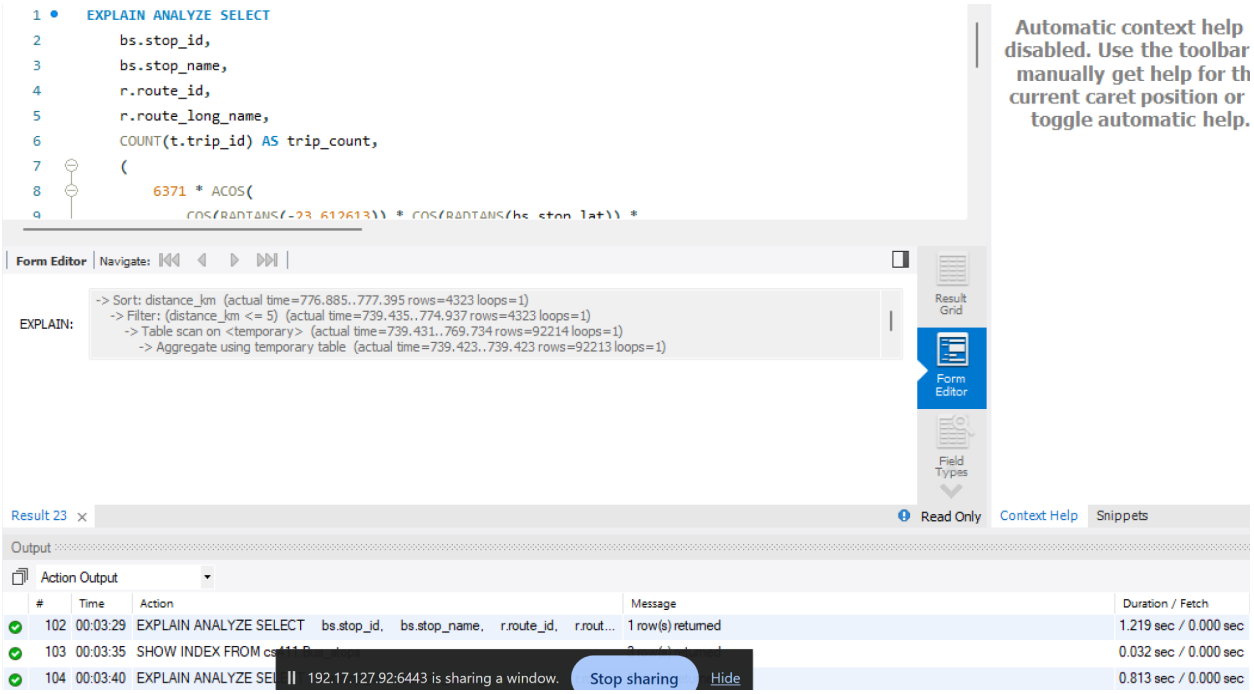
Below is the analysis, and adding the index of latitude decreases the efficiency quite significantly. This decrease in efficiency can be due to a potential lower selectivity in Bus_stops.stop_lat. The values in stop_lat are very similar, ranging from -23.4 to -23.9, and the column has more duplicate values than stop_lon. As such, there might be an even longer scan through the column than the previous index design, increasing the running time. Furthermore, similar to the previous design, although the query is filtering based on distance, the distance calculation is not proportionally related to stop_lat, so the query can't filter based on the latitude index; therefore, this index design can't optimize the query significantly.



Design 3: Since we wanted to analyze the performance change in indexing on GROUP BY attributes, we tried to add an index on Bus_stops.stop_name



Below is the analysis, and adding an index on the bus name might make the query slightly more efficient. One possible reason is that the database engine might utilize the index to more efficiently organize and access data during the grouping process as Bus_stops.stop_name is one of the GROUP BY factors used in the query, which possibly accelerates the speed of data retrieval. Additionally, while grouping the results, the indexed stop_name could help sort out the records quickly since there is a significant number of unique stop names. Therefore, it shows that indexing on GROUP BY keywords can enhance query efficiency.



Design 4: With the performance improvement from indexing on GROUP BY attribute from Design 3, Bus_stops.stop_name, we wanted to further investigate performance changes on other GROUP BY attributes, and therefore we tried to add an index on Route.route_long_name.

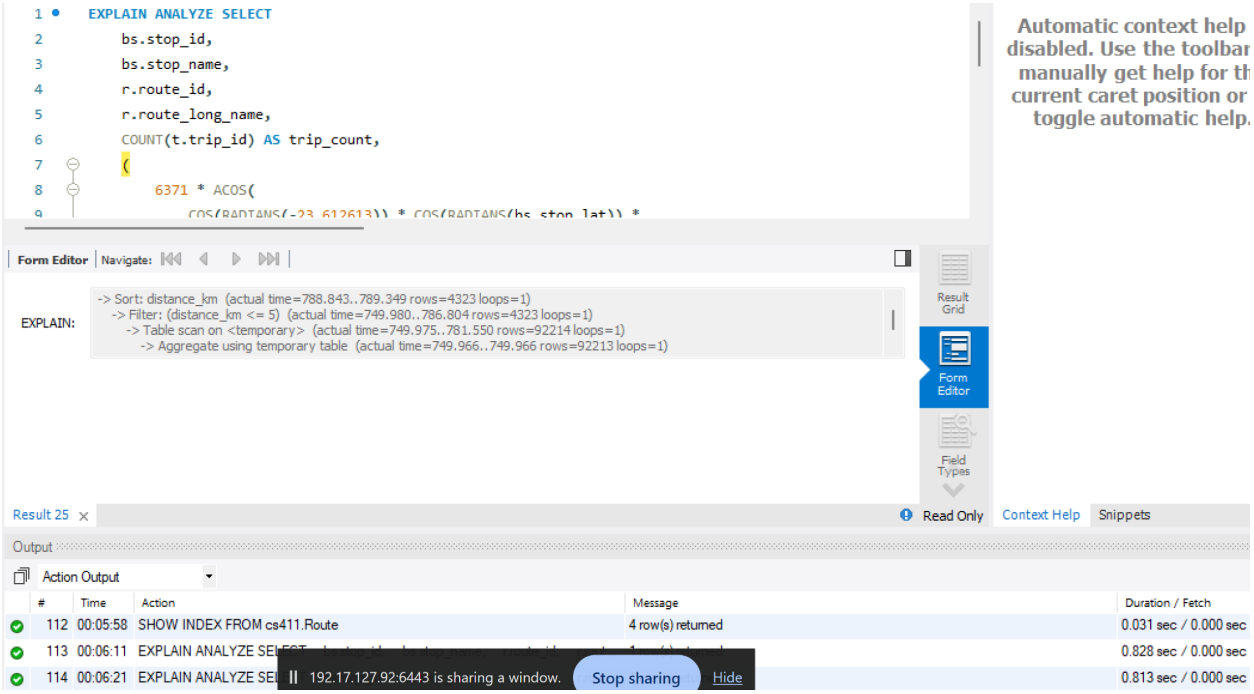
```
CREATE INDEX routeName ON cs411.Route(route_long_name);
SHOW INDEX FROM cs411.Route;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

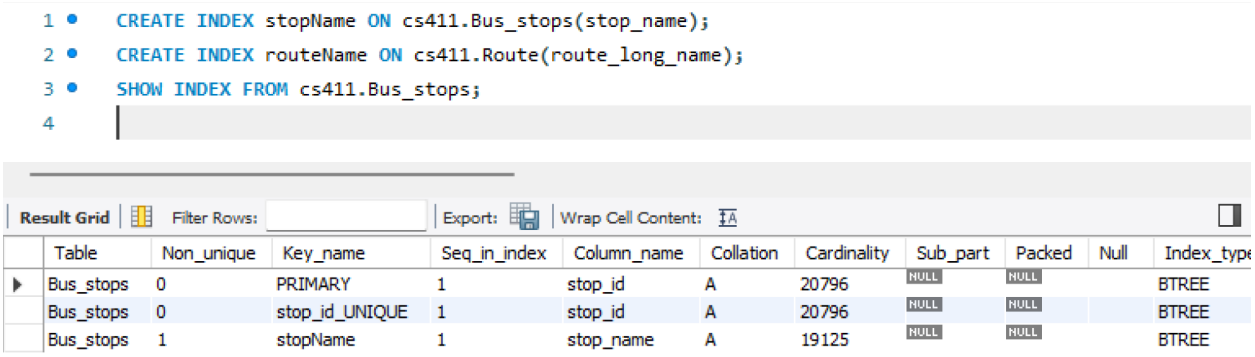
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_typ
Route	0	PRIMARY	1	route_id	A	1347	NULL	NULL		BTREE
Route	0	route_id_UNIQUE	1	route_id	A	1347	NULL	NULL		BTREE
Route	1	fare_id_idx	1	fare_id	A	1	NULL	NULL	YES	BTREE
Route	1	routeName	1	route_long_name	A	1200	NULL	NULL	YES	BTREE

Below is the analysis, and adding an index on the route name might make the query marginally more efficient than before. Before adding an index on Route.route_long_name, each Route.route_long_name was distinctive and lengthy, such as "Conj. Hab. Vl. Sílvia - Pça. Do Correio" and "Conexão Petrônio Portela - Metrô Barra Funda." Therefore, with the applied indexing on the attribute, the database is possibly able to increase the data retrieval efficiently during grouping operations. Also, it is worth noting that the differences in cardinality between the two attributes, Bus_stops.stop_name from Design 3 and Route.route_long_name from the current Design could play an important role here: indexing on Bus_stops.stop_name with 20,000+ rows shows its more

impactful result, whereas indexing on Route.route_long_name with 1900 + rows that yields comparatively milder optimization/result. Therefore, these two designs show the importance of indexing on GROUP BY attributes.

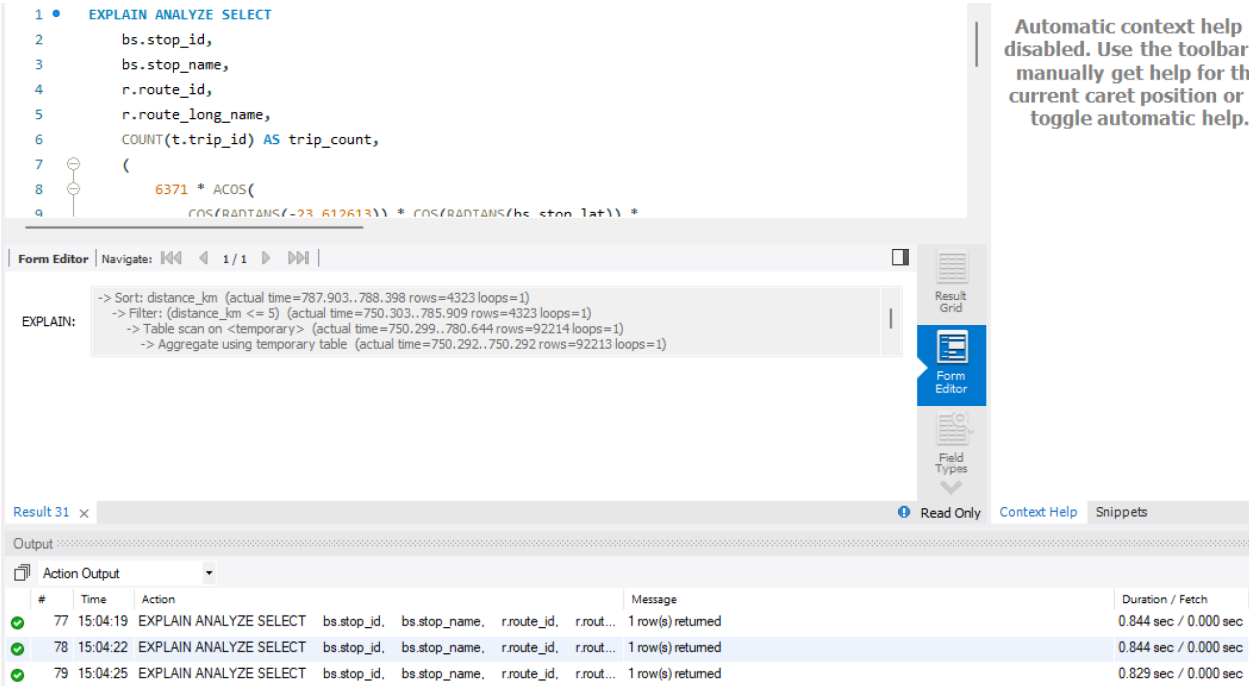


Design 5: Since the designs above showed the efficiency increase, we decided to keep both indices on each Bus_stop.stop_name and Route.route_long_name.



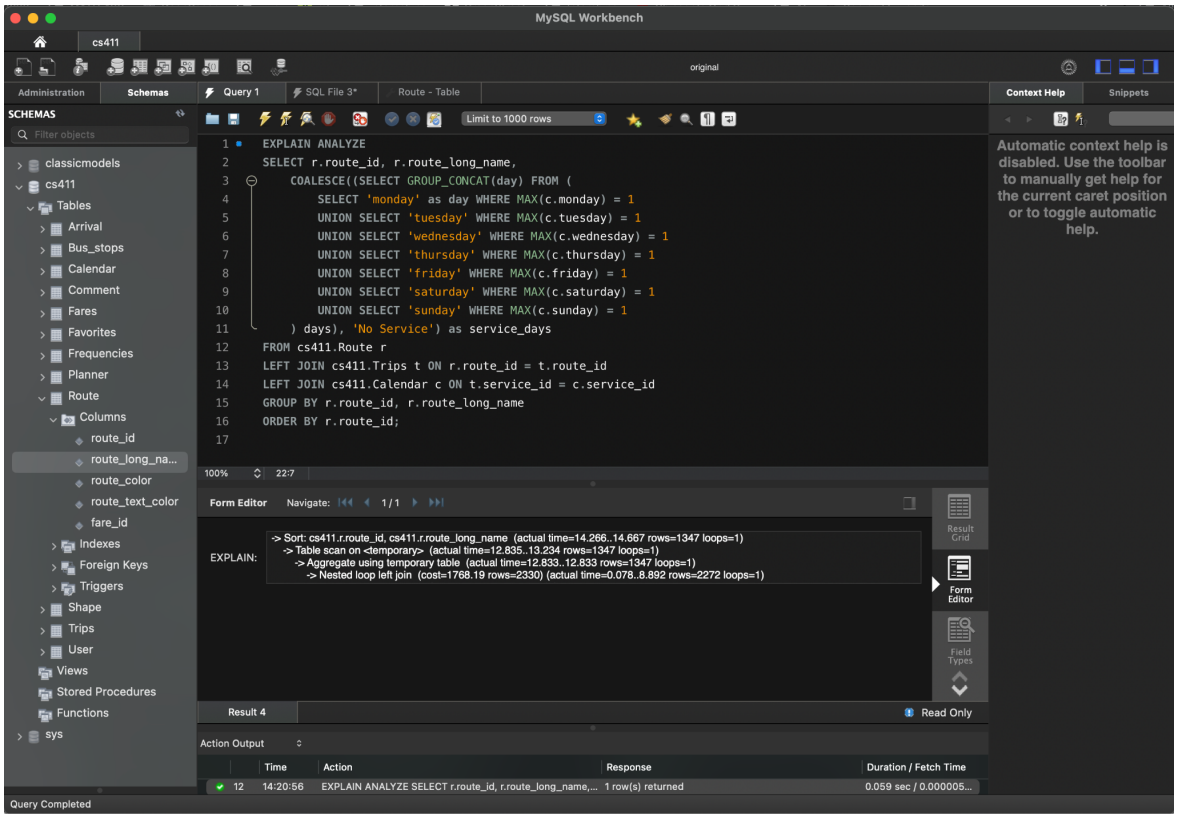
Below is the analysis, and adding an index on the Bus_stop.stop_name and Route.route_long_name might make the query slightly more efficient. Since Bus_stop.stop_name and Route.route_long_name are likely to contain a broader range of values with a higher degree of uniqueness, the search within these columns could be more efficient. However, both Bus_stop.stop_name and Route.route_long_name are in text, and text-based indices can sometimes add overhead compared to

the number-based indices. Thus, while these indices may theoretically streamline certain lookups, their actual impact appears to be limited.

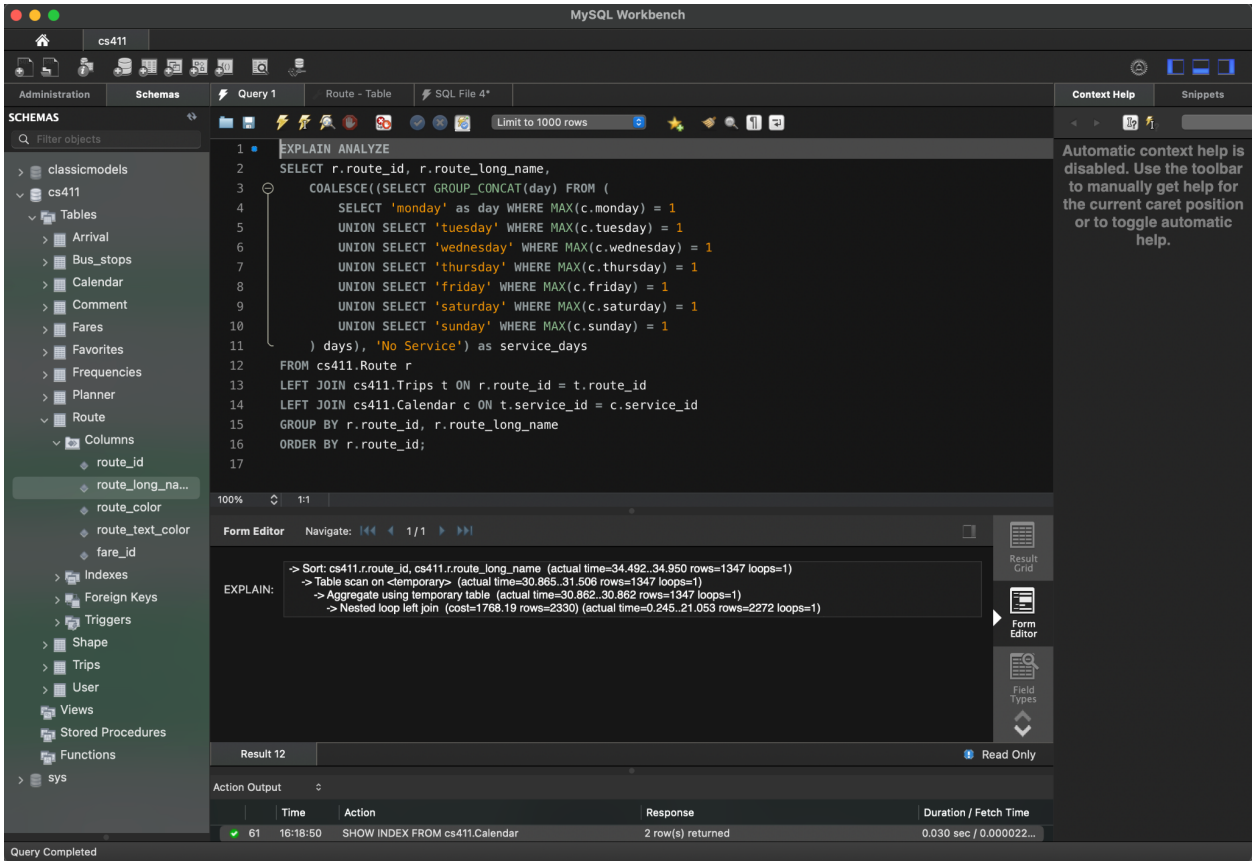


Index for Advanced Query 2:

For default index, we got an analysis like this:

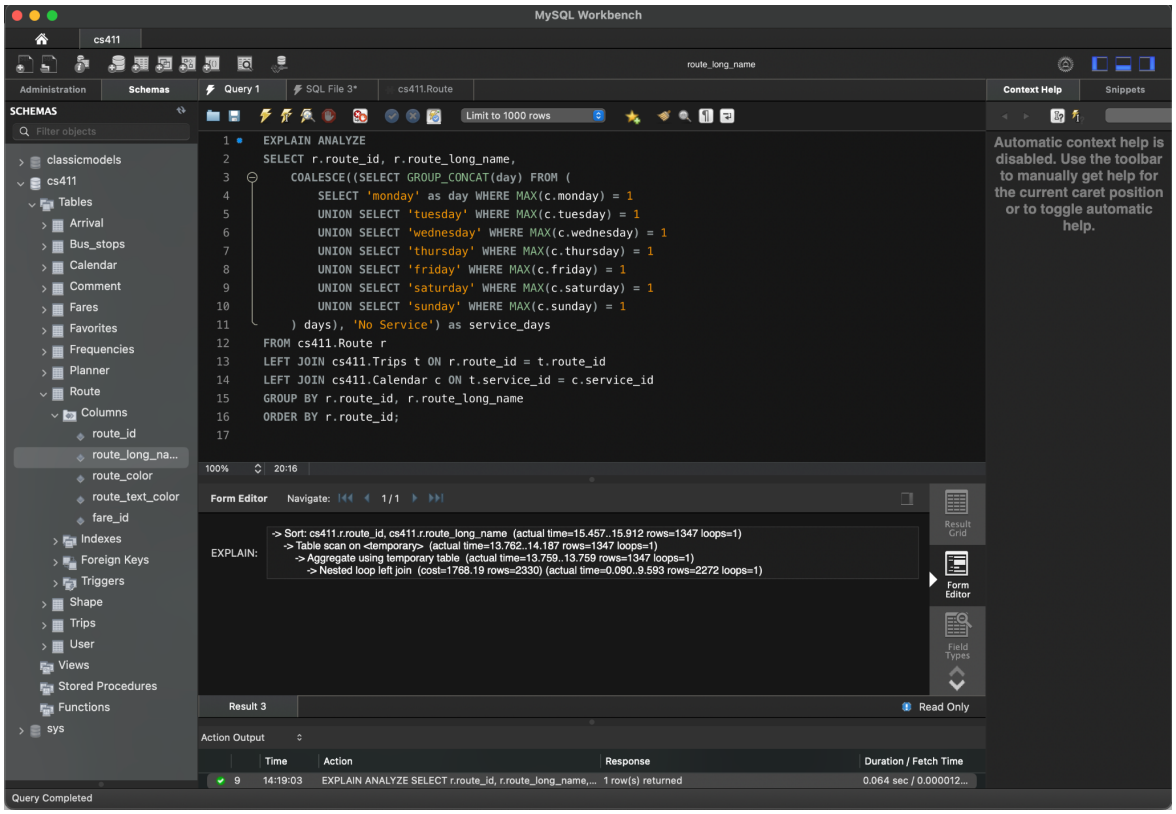


Design 1: We tried to add an index in Calendar.monday.



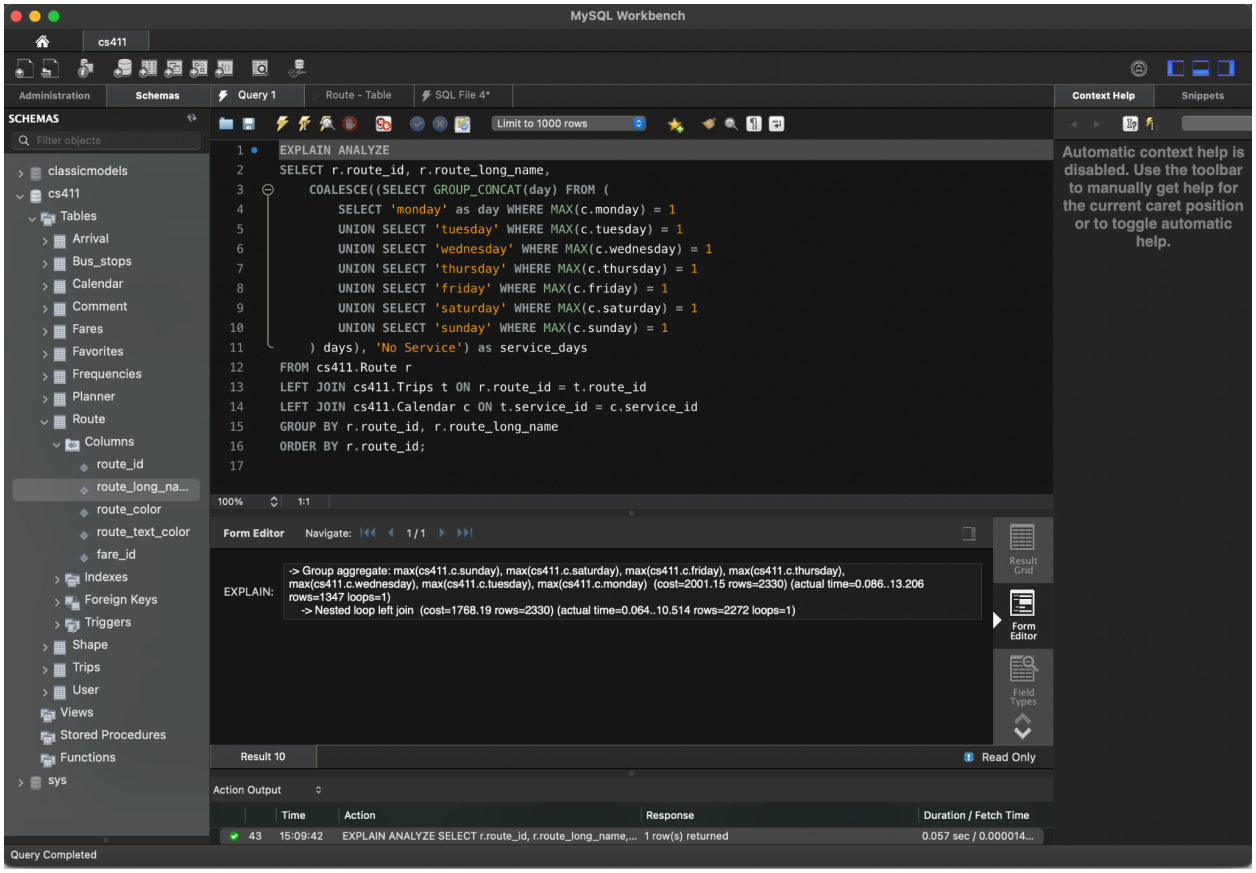
This is the analysis, and adding an index on the Calendar.monday decreases the efficiency significantly. In this advanced query, it is aggregating over the “monday” column using the MAX function. The index may not be as effective because the query processor must still perform a full scan of all records to determine the maximum value, negating the benefits of a targeted index seek. Furthermore, the values in “monday” are either 1 or 0, meaning, many of its values are duplicated in the table. This makes the indexing become less useful and inefficient.

Design 2: We tried to add an index in Route.route_long_name.



Adding an index on Route.route_long_name did not result in a performance improvement for this specific query. In fact, the sort and table scan operations took slightly longer after the index was added. This might be due to the overhead of maintaining the index during the query execution. The sort operation's time slightly increased to between 15.457 and 15.912 milliseconds, still processing 1347 rows. Table Scan on «temporary»: This operation’s time increased to between 13.762 and 14.187 milliseconds, processing 1347 rows.

Design 3: We tried to add an index combining Route.route_id and Route.route_long_name.



Adding the index of (route_id, route_long_name) might increase the efficiency of the query. The index sorted the rows on these columns, eliminating the need for an explicit sort operation when executing the query. This change, along with the shift from using a temporary table to a direct group aggregate, reflects a more efficient data handling and a positive impact on the query's performance, even though other parts like nested loop joins and subquery projections remained largely unaffected.