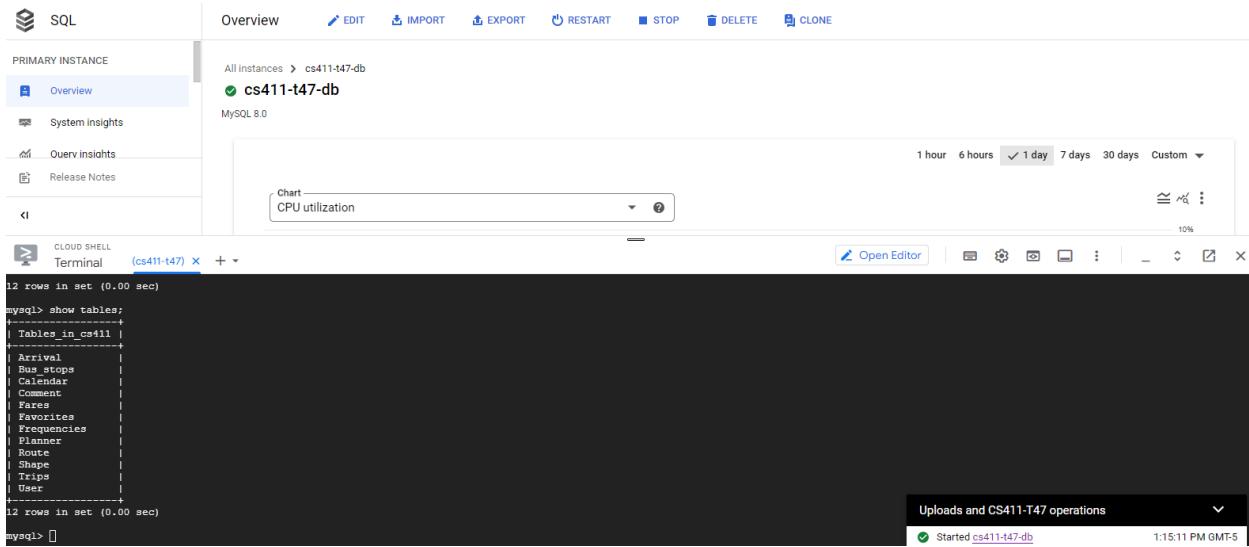


Stage 3 - Database Design

PART 1

Implementation of the database tables on GCP



```
mysql> show tables;
+-----+
| Tables_in_cs411 |
+-----+
| Arrival          |
| Bus_stops        |
| Calendar         |
| Comment          |
| Fares            |
| Favorites         |
| Frequencies      |
| Planner          |
| Route             |
| Shape             |
| Ticket            |
| User              |
+-----+
12 rows in set (0.00 sec)

mysql> 
```

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:

```

1 •  SELECT COUNT(*)
2   FROM cs411.Trips;

```

Result Grid
COUNT(*) 2227

Arrival x Frequencies - Table

```
1 •  SELECT COUNT(*) FROM cs411.Bus_stops;
2
```

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

COUNT(*)	20902
----------	-------

Arrival x Frequencies - Table

```
1 •  SELECT COUNT(*) FROM cs411.Arrival;
2
```

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

COUNT(*)	94911
----------	-------

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. Britânia - 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 VI. Iório	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1018-10	VI. 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 VI. Iório	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1021-10	Cohab Brasilândia - Term. Pirituba	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1023-10	VI. Pirituba - Cptm Pirituba	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1024-10	Jd. Caromba - Conexão Petrólio Portela	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1025-10	Jd. Carombé - Conexão Petrólio Portela	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1026-10	VI. Iara - Conexão Petrólio Portela	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1034-10	VI. Iara - Conexão VI. Iório	monday,tuesday,wednesday,thursday,friday,saturday,sunday
1036-10	Cohab Brasilândia - Conexão VI. 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:

The screenshot shows the MySQL Workbench interface. In the top-left, a query window displays the following EXPLAIN ANALYZE output:

```

1 • EXPLAIN ANALYZE SELECT
2     bs.stop_id,
3     bs.stop_name,
4     r.route_id,
5     r.route_long_name,
6     COUNT(t.trip_id) AS trip_count,
7     (
8         6371 * ACOS(
9             COS(RADIANS(-33.61261311) * COS(RADIANS(bs.stop_lat)) *

```

The EXPLAIN output below the query shows the execution plan:

- > Sort: distance_km (actual time=798.974..799.441 rows=4323 loops=1)
- > Filter: (distance_km <= 5) (actual time=761.520..797.008 rows=4323 loops=1)
- > Table scan on <temporary> (actual time=761.517..791.768 rows=92214 loops=1)
- > Aggregate using temporary table (actual time=761.509..761.509 rows=92213 loops=1)

In the bottom-right, a results grid shows the execution history for three statements:

#	Time	Action	Message	Duration / Fetch
19	14:41:44	EXPLAIN ANALYZE SELECT	bs.stop_id, bs.stop_name, r.route_id, r.route_long_name, COUNT(t.trip_id) AS trip_count	1 row(s) returned 0.875 sec / 0.000 sec
20	14:42:04	SHOW INDEX FROM cs411.Bus_stops		2 row(s) returned 0.032 sec / 0.000 sec
21	14:42:07	EXPLAIN ANALYZE SELECT	bs.stop_id, bs.stop_name, r.route_id, r.route_long_name, COUNT(t.trip_id) AS trip_count	1 row(s) returned 0.875 sec / 0.000 sec

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.

```

1 • CREATE INDEX lon ON cs411.Bus_stops(stop_lon);
2 • SHOW INDEX FROM cs411.Bus_stops;
3

```

The screenshot shows the MySQL Workbench Result Grid displaying the indexes for the Bus_stops table:

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	NULL	BTREE
Bus_stops	0	stop_id_UNIQUE	1	stop_id	A	20796	NULL	NULL	NULL	BTREE
Bus_stops	1	lon	1	stop_lon	A	20312	NULL	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.

The screenshot shows the MySQL Workbench interface with the 'Form Editor' tab selected. In the main pane, the query is:

```

1 • EXPLAIN ANALYZE SELECT
2     bs.stop_id,
3     bs.stop_name,
4     r.route_id,
5     r.route_long_name,
6     COUNT(t.trip_id) AS trip_count,
7     (
8         6371 * ACOS(
9             COS(RADTAN(-23.61261311 * r.stop_lat)) * COS(RADTAN(bs.stop_lat))

```

The 'EXPLAIN' output shows the execution plan:

- > Sort: distance_km (actual time=809.943..810.461 rows=4323 loops=1)
- > Filter: (distance_km <= 5) (actual time=772.563..807.903 rows=4323 loops=1)
 -> Table scan on <temporary> (actual time=772.559..802.671 rows=92214 loops=1)
 -> Aggregate using temporary table (actual time=772.552..772.552 rows=92213 loops=1)

On the right, there is a vertical toolbar with icons for Result Grid, Form Editor (selected), Field Types, and Context Help.

Below the main pane, the 'Result 15' tab is open, showing the execution history:

#	Time	Action	Message	Duration / Fetch
37	14:44:21	EXPLAIN ANALYZE SELECT bs.stop_id, bs.stop_name, r.route_id, r.route_long_name, COUNT(t.trip_id) AS trip_count, (6371 * ACOS(COS(RADTAN(-23.61261311 * r.stop_lat)) * COS(RADTAN(bs.stop_lat)))	1 row(s) returned	0.906 sec / 0.000 sec
38	14:45:03	SHOW INDEX FROM cs411.Bus_stops	3 row(s) returned	0.031 sec / 0.000 sec
39	14:45:07	EXPLAIN ANALYZE SELECT bs.stop_id, bs.stop_name, r.route_id, r.route_long_name, COUNT(t.trip_id) AS trip_count, (6371 * ACOS(COS(RADTAN(-23.61261311 * r.stop_lat)) * COS(RADTAN(bs.stop_lat)))	1 row(s) returned	0.890 sec / 0.000 sec

At the bottom, there are tabs for Read Only, Context Help, and Snippets.

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.

```

1 • CREATE INDEX lat ON cs411.Bus_stops(stop_lat);
2 • SHOW INDEX FROM cs411.Bus_stops;
3

```

The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. The output of the SHOW INDEX command is:

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	lat	1	stop_lat	A	20161	NULL	NULL		BTREE

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.

```

1 • EXPLAIN ANALYZE SELECT
2     bs.stop_id,
3     bs.stop_name,
4     r.route_id,
5     r.route_long_name,
6     COUNT(t.trip_id) AS trip_count,
7     (
8         6371 * ACOS(
9             COS(RADIANS(-23.612613)) * COS(RADIANS(bs.stop_lat)) *

```

EXPLAIN:

- > Sort: distance_km (actual time=948.447..949.019 rows=4323 loops=1)
- > Filter: (distance_km <= 5) (actual time=906.715..946.097 rows=4323 loops=1)
 -> Table scan on <temporary> (actual time=906.711..940.495 rows=92214 loops=1)
 -> Aggregate using temporary table (actual time=906.703..906.703 rows=92213 loops=1)

Result 13 x

Action Output

#	Time	Action	Message	Duration / Fetch
75	23:53:04	CREATE INDEX lat ON cs411.Bus_stops(stop_lat)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.219 sec
76	23:53:18	SHOW INDEX FROM cs411.Bus_stops	3 rows	0.032 sec / 0.000 sec
77	23:53:40	EXPLAIN ANALYZE SEL 192.17.127.92:6443 is sharing a window.	Stop sharing Hide	0.984 sec / 0.000 sec

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

```

1 • CREATE INDEX stopName ON cs411.Bus_stops(stop_name);
2 • SHOW INDEX FROM cs411.Bus_stops;
3

```

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	NULL	BTREE
Bus_stops	0	stop_id_UNIQUE	1	stop_id	A	20796	NULL	NULL	NULL	BTREE
Bus_stops	1	stopName	1	stop_name	A	18780	NULL	NULL	NULL	BTREE

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.

```

1 • EXPLAIN ANALYZE SELECT
2     bs.stop_id,
3     bs.stop_name,
4     r.route_id,
5     r.route_long_name,
6     COUNT(t.trip_id) AS trip_count,
7     (
8         6371 * ACOS(
9             COS(RANTANS - 23.61261311 * COS(RANTANS/bs.stop_lat)) *

```

EXPLAIN:

- > Sort: distance_km (actual time=776.885..777.395 rows=4323 loops=1)
- > Filter: (distance_km <= 5) (actual time=739.435..774.937 rows=4323 loops=1)
- > Table scan on <temporary> (actual time=739.431..769.734 rows=92214 loops=1)
- > Aggregate using temporary table (actual time=739.423..739.423 rows=92213 loops=1)

Action Output:

#	Time	Action	Message	Duration / Fetch	
102	00:03:29	EXPLAIN ANALYZE SELECT	bs.stop_id, bs.stop_name, r.route_id, r.route_long_name, COUNT(t.trip_id) AS trip_count	1.219 sec / 0.000 sec	
103	00:03:35	SHOW INDEX FROM cs411.Route		0.032 sec / 0.000 sec	
104	00:03:40	EXPLAIN ANALYZE SEL	192.17.127.92:6443 is sharing a window.	Stop sharing Hide	0.813 sec / 0.000 sec

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.

- 1 • CREATE INDEX routeName ON cs411.Route(route_long_name);
- 2 • SHOW INDEX FROM cs411.Route;
- 3

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
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. VI. 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.

The screenshot shows the pgAdmin interface with the following details:

- EXPLAIN ANALYZE Output:**

```

1 • EXPLAIN ANALYZE SELECT
2     bs.stop_id,
3     bs.stop_name,
4     r.route_id,
5     r.route_long_name,
6     COUNT(t.trip_id) AS trip_count,
7     (
8         6371 * ACOS(
9             COS(RADIANS(-23.61261311 * COS(RADIANS(bs.stop_lat)))) *

```
- EXPLAIN Plan:**

```

-> Sort: distance_km (actual time=788.843..789.349 rows=4323 loops=1)
-> Filter: (distance_km <= 5) (actual time=749.980..786.804 rows=4323 loops=1)
-> Table scan on <temporary> (actual time=749.975..781.550 rows=92214 loops=1)
-> Aggregate using temporary table (actual time=749.966..749.966 rows=92213 loops=1)

```
- Action Output:**

#	Time	Action	Message	Duration / Fetch
112	00:05:58	SHOW INDEX FROM cs411.Route	4 row(s) returned	0.031 sec / 0.000 sec
113	00:06:11	EXPLAIN ANALYZE SELECT ...	1 row(s) returned	0.828 sec / 0.000 sec
114	00:06:21	EXPLAIN ANALYZE SELECT ... 192.17.127.92:6443 is sharing a window.	Stop sharing Hide	0.813 sec / 0.000 sec
- Result Grid:** Shows the results of the query execution.

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.

The screenshot shows the pgAdmin interface with the following details:

- SQL History:**

```

1 • CREATE INDEX stopName ON cs411.Bus_stops(stop_name);
2 • CREATE INDEX routeName ON cs411.Route(route_long_name);
3 • SHOW INDEX FROM cs411.Bus_stops;
4

```
- Result Grid:**

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	stopName	1	stop_name	A	19125	NULL	NULL		BTREE

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.

The screenshot shows a database interface with the following components:

- Query Editor:** Displays the SQL query:


```

1 • EXPLAIN ANALYZE SELECT
2     bs.stop_id,
3     bs.stop_name,
4     r.route_id,
5     r.route_long_name,
6     COUNT(t.trip_id) AS trip_count,
7     (
8         6371 * ACOS(
9             COS(RADTAN($-23.612613)) * COS(RADTAN(bs.stop_lat)) *
      
```
- EXPLAIN Output:** Shows the execution plan:


```

-> Sort: distance_km (actual time=787.903..788.398 rows=4323 loops=1)
-> Filter: (distance_km <= 5) (actual time=750.303..785.909 rows=4323 loops=1)
-> Table scan on <temporary> (actual time=750.299..780.644 rows=92214 loops=1)
-> Aggregate using temporary table (actual time=750.292..750.292 rows=92213 loops=1)
      
```
- Result Grid:** Shows the results of the query, which are empty (0 rows).
- Toolbar:** Includes buttons for Form Editor, Result Grid, Field Types, Read Only, Context Help, and Snippets.
- Help Message:** A tooltip on the right says: "Automatic context help disabled. Use the toolbar manually get help for the current caret position or toggle automatic help."

Index for Advanced Query 2:

For default index, we got an analysis like this:

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** cs411
- Query Editor:** EXPLAIN ANALYZE SELECT r.route_id, r.route_long_name, ...
Detailed description: The query uses COALESCE to select days of the week based on MAX values from the calendar table. It then joins the route table and calendar table to get route names and service IDs. Finally, it groups by route ID and name, and orders by route ID.
- Result Grid:** Shows the execution plan:
 - Sort: cs411.r.route_id, cs411.r.route_long_name (actual time=14.266, 14.667 rows=1347 loops=1)
 - Table scan on <temporar> (actual time=12.835, 13.234 rows=1347 loops=1)
 - Aggregate using temporary table (actual time=12.833, 12.833 rows=1347 loops=1)
 - Nested loop left join (cost=1768.19 rows=2330) (actual time=0.078, 8.892 rows=2272 loops=1)
- Action Output:** Shows the command: EXPLAIN ANALYZE SELECT r.route_id, r.route_long_name, ... 1 row(s) returned
- Timing:** 0.059 sec / 0.000005s

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

The screenshot shows the MySQL Workbench interface with the 'Schemas' tree on the left expanded to show the 'cs411' database, which contains the 'classicmodels' schema and several tables like 'Arrival', 'Bus_stops', 'Calendar', etc. The 'Tables' section for 'cs411' is selected. In the main pane, a SQL query is being analyzed:

```
EXPLAIN ANALYZE
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;
```

The 'EXPLAIN' output shows the execution plan:

```
> Sort: cs411.r.route_id, cs411.r.route_long_name (actual time=34.492..34.950 rows=1347 loops=1)
-> Table scan on <temporary> (actual time=30.865..31.506 rows=1347 loops=1)
-> Aggregate using temporary table (actual time=30.862..30.862 rows=1347 loops=1)
-> Nested loop left join (cost=1768.19 rows=2330) (actual time=0.245..21.053 rows=2272 loops=1)
```

On the right side of the interface, there is a note: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the main pane, a status bar shows "Query Completed".

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.

The screenshot shows the MySQL Workbench interface with the schema 'cs411' selected. In the central pane, a query is being analyzed:

```
1 • EXPLAIN ANALYZE
2   SELECT r.route_id, r.route_long_name,
3       COALESCE((SELECT GROUP_CONCAT(day) FROM (
4           SELECT 'monday' as day WHERE MAX(c.monday) = 1
5           UNION SELECT 'tuesday' WHERE MAX(c.tuesday) = 1
6           UNION SELECT 'wednesday' WHERE MAX(c.wednesday) = 1
7           UNION SELECT 'thursday' WHERE MAX(c.thursday) = 1
8           UNION SELECT 'friday' WHERE MAX(c.friday) = 1
9           UNION SELECT 'saturday' WHERE MAX(c.saturday) = 1
10          UNION SELECT 'sunday' WHERE MAX(c.sunday) = 1
11      ) days), 'No Service') as service_days
12     FROM cs411.Route r
13    LEFT JOIN cs411.Trips t ON r.route_id = t.route_id
14    LEFT JOIN cs411.Calendar c ON t.service_id = c.service_id
15   GROUP BY r.route_id, r.route_long_name
16  ORDER BY r.route_id;
17
```

The 'EXPLAIN' output shows the execution plan:

```
> Sort: cs411.r.route_id, cs411.r.route_long_name (actual time=15.457..15.912 rows=1347 loops=1)
-> Table scan on <temporary> (actual time=13.762..14.187 rows=1347 loops=1)
-> Aggregate using temporary table (actual time=13.758..13.759 rows=1347 loops=1)
-> Nested loop left join (cost=1768.19 rows=2330) (actual time=0.050..9.593 rows=2272 loops=1)
```

The bottom pane shows the results of the EXPLAIN ANALYZE command:

Action Output	c		
Time	Action	Response	Duration / Fetch Time
9	14:19:03	EXPLAIN ANALYZE SELECT r.route_id, r.route_long_name,... 1 row(s) returned	0.064 sec / 0.000012...

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.

The screenshot shows the MySQL Workbench interface with the schema 'cs411' selected. In the left sidebar, under the 'Tables' section of the 'Route' table, the 'route_id' and 'route_long_name' columns are highlighted. The main pane displays the SQL query:

```
1 EXPLAIN ANALYZE
2 SELECT r.route_id, r.route_long_name,
3       COALESCE((SELECT GROUP_CONCAT(day) FROM (
4           SELECT 'monday' as day WHERE MAX(c.monday) = 1
5           UNION SELECT 'tuesday' WHERE MAX(c.tuesday) = 1
6           UNION SELECT 'wednesday' WHERE MAX(c.wednesday) = 1
7           UNION SELECT 'thursday' WHERE MAX(c.thursday) = 1
8           UNION SELECT 'friday' WHERE MAX(c.friday) = 1
9           UNION SELECT 'saturday' WHERE MAX(c.saturday) = 1
10          UNION SELECT 'sunday' WHERE MAX(c.sunday) = 1
11      ) days), 'No Service') as service_days
12 FROM cs411.Route r
13 LEFT JOIN cs411.Trips t ON r.route_id = t.route_id
14 LEFT JOIN cs411.Calendar c ON t.service_id = c.service_id
15 GROUP BY r.route_id, r.route_long_name
16 ORDER BY r.route_id;
```

The 'EXPLAIN' output shows the execution plan, indicating a group aggregate and nested loop joins. The results pane shows the query took 0.057 seconds for 1 row.

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.