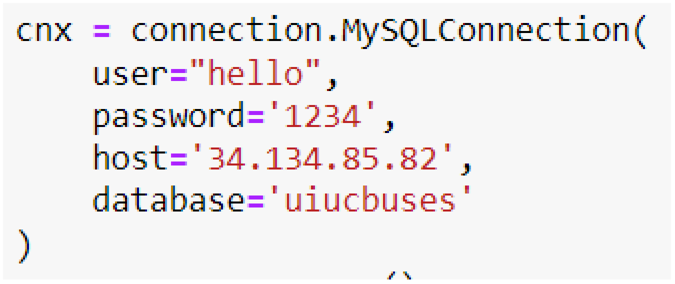
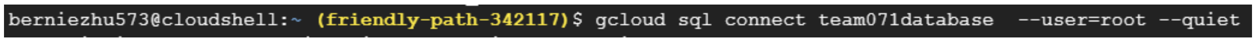
Connection Information:
This is the information we used when connecting to the database through python



```
cnx = connection.MySQLConnection(
    user="hello",
    password='1234',
    host='34.134.85.82',
    database='uiucbuses'
)
```

team071database is the name of the instance

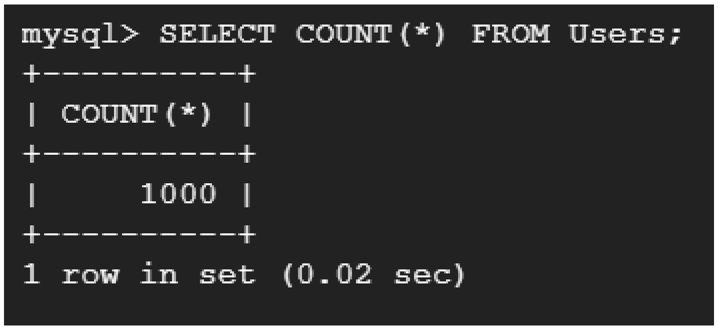


```
berniezhu573@cloudshell:~ (friendly-path-342117)$ gcloud sql connect team071database  --user=root --quiet
```

Data Definition Language (DDL) Commands:

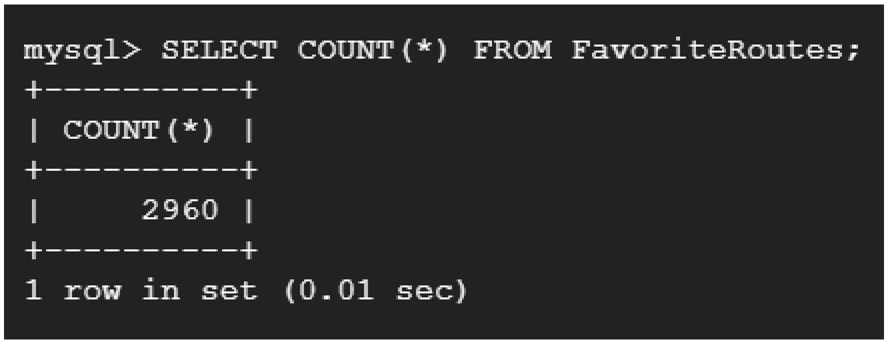
```
CREATE TABLE BusLine (
  LineID VARCHAR(40) NOT NULL,
  Color VARCHAR(10),
  Number INTEGER,
  PRIMARY KEY(LineID)
);

CREATE TABLE BusStop (
  StopID VARCHAR(35) NOT NULL,
  StopLatitude REAL,
  StopLongitude REAL,
  PRIMARY KEY(StopID)
);

CREATE TABLE Buses (
  BusID VARCHAR(5) NOT NULL,
  LineID VARCHAR(40),
  StopOrigin VARCHAR(35),
  StopDestination VARCHAR(35),
  Direction VARCHAR(10),
  ShapeID VARCHAR(40),
  PRIMARY KEY(BusID),
  FOREIGN KEY(LineID) REFERENCES BusLine(LineID) ON DELETE SET NULL
);

CREATE TABLE Users (
  Username VARCHAR(20) NOT NULL,
  Password VARCHAR(20) NOT NULL,
  PRIMARY KEY(Username)
);

CREATE TABLE FavoriteRoutes (
  LineID VARCHAR(40) NOT NULL,
  Username VARCHAR(20) NOT NULL,
  DateCreated VARCHAR(10),
  PRIMARY KEY(LineID, Username),
  FOREIGN KEY(LineID) REFERENCES BusLine(LineID) ON DELETE CASCADE,
  FOREIGN KEY(Username) REFERENCES Users(Username) ON DELETE CASCADE
);

CREATE TABLE FavoriteStops (
  StopID VARCHAR(35) NOT NULL,
  Username VARCHAR(20) NOT NULL,
  DateCreated VARCHAR(10),
  PRIMARY KEY(StopID, Username),
  FOREIGN KEY(StopID) REFERENCES BusStop(StopID) ON DELETE CASCADE,
  FOREIGN KEY(Username) REFERENCES Users(Username) ON DELETE CASCADE
);
```

Users - 1000 entries:



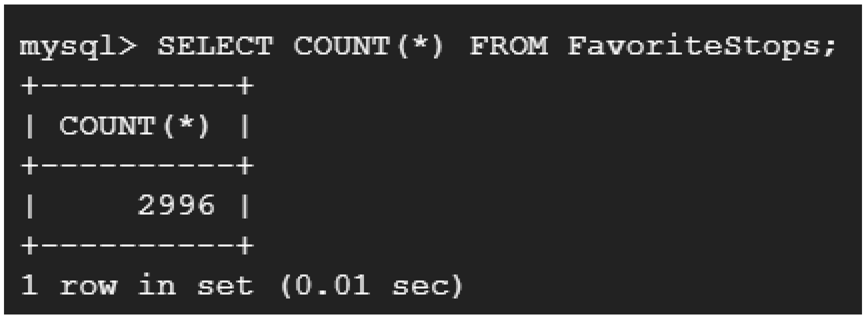
```
mysql> SELECT COUNT(*) FROM Users;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.02 sec)
```

FavoriteRoutes - 2960 entries:



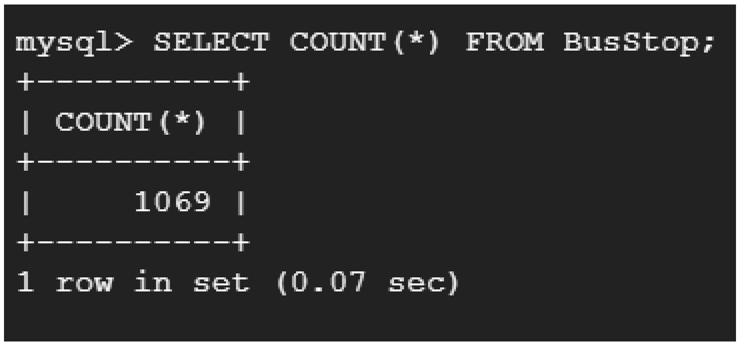
```
mysql> SELECT COUNT(*) FROM FavoriteRoutes;
+----------+
| COUNT(*) |
+----------+
|     2960 |
+----------+
1 row in set (0.01 sec)
```

FavoriteStops - 2996 entries:



```
mysql> SELECT COUNT(*) FROM FavoriteStops;
+----------+
| COUNT(*) |
+----------+
|     2996 |
+----------+
1 row in set (0.01 sec)
```

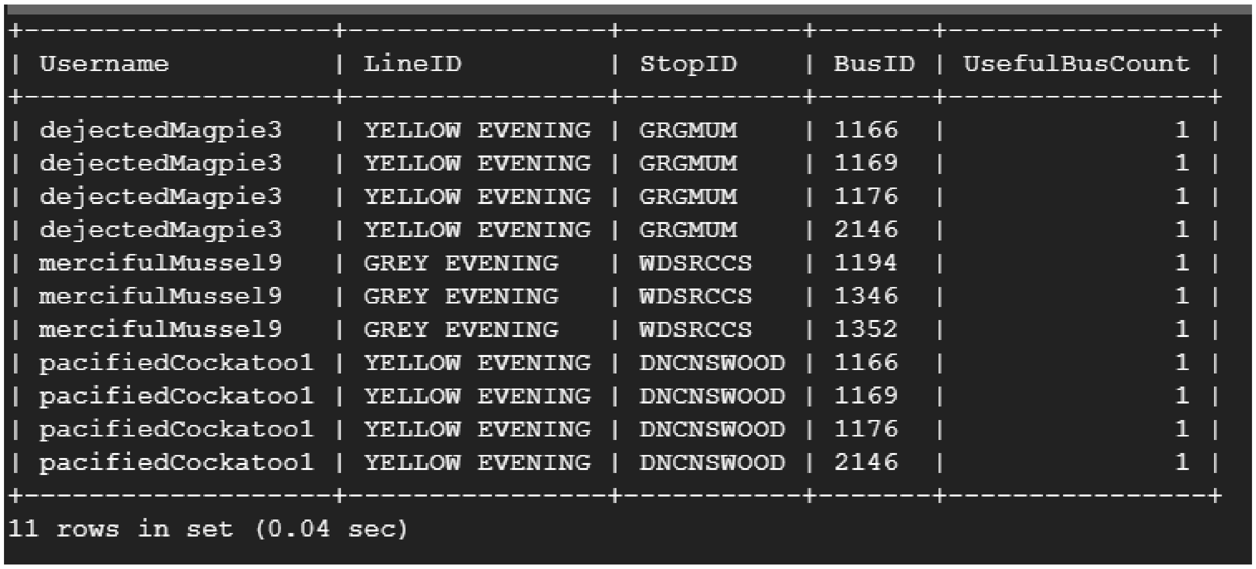
BusStop - 1069 entries:



```
mysql> SELECT COUNT(*) FROM BusStop;
+----------+
| COUNT(*) |
+----------+
|     1069 |
+----------+
1 row in set (0.07 sec)
```

Advanced Query 1:
Retrieve all buses that are part of a user’s favorite routes, as well as starting at one of their favorite stops

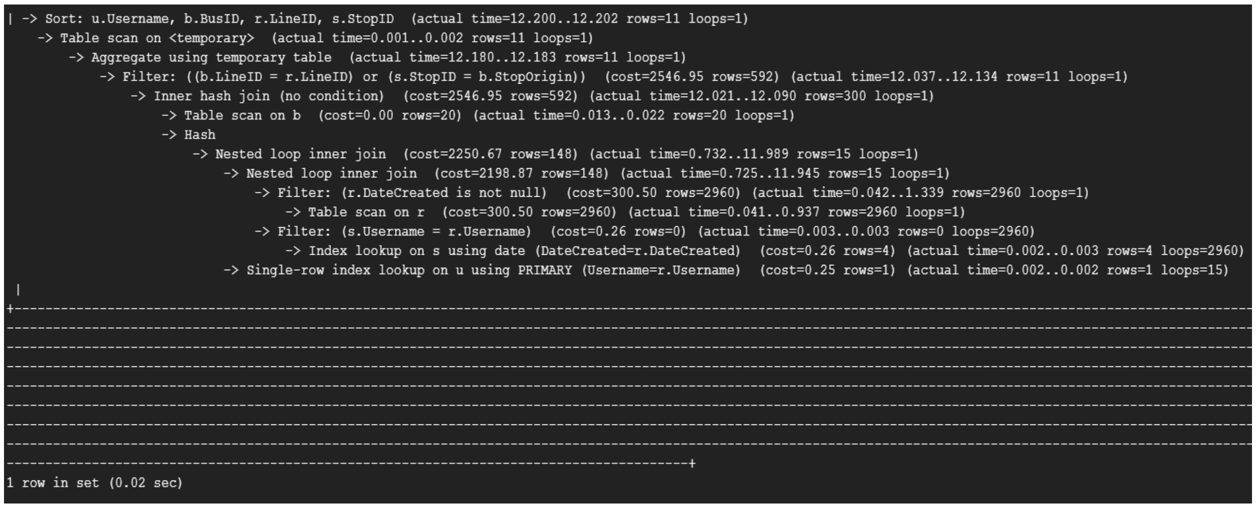
SELECT u.Username, r.LineID, s.StopID, b.BusID, COUNT(b.BusID) AS UsefulBusCount
FROM Users u NATURAL JOIN FavoriteRoutes r NATURAL JOIN FavoriteStops s JOIN Buses b ON r.LineID = b.LineID OR s.StopID = b.StopOrigin
GROUP BY u.Username, b.BusID, r.LineID, s.StopID
ORDER BY u.Username, b.BusID;



```
+-------------------+----------------+-----------+-------+----------------+
| Username          | LineID         | StopID    | BusID | UsefulBusCount |
+-------------------+----------------+-----------+-------+----------------+
| dejectedMagpie3   | YELLOW EVENING | GRGMUM    | 1166  |              1 |
| dejectedMagpie3   | YELLOW EVENING | GRGMUM    | 1169  |              1 |
| dejectedMagpie3   | YELLOW EVENING | GRGMUM    | 1176  |              1 |
| dejectedMagpie3   | YELLOW EVENING | GRGMUM    | 2146  |              1 |
| mercifulMussel9   | GREY EVENING   | WDSRCCS   | 1194  |              1 |
| mercifulMussel9   | GREY EVENING   | WDSRCCS   | 1346  |              1 |
| mercifulMussel9   | GREY EVENING   | WDSRCCS   | 1352  |              1 |
| pacifiedCockatoo1 | YELLOW EVENING | DNCNSWOOD | 1166  |              1 |
| pacifiedCockatoo1 | YELLOW EVENING | DNCNSWOOD | 1169  |              1 |
| pacifiedCockatoo1 | YELLOW EVENING | DNCNSWOOD | 1176  |              1 |
| pacifiedCockatoo1 | YELLOW EVENING | DNCNSWOOD | 2146  |              1 |
+-------------------+----------------+-----------+-------+----------------+
11 rows in set (0.04 sec)
```

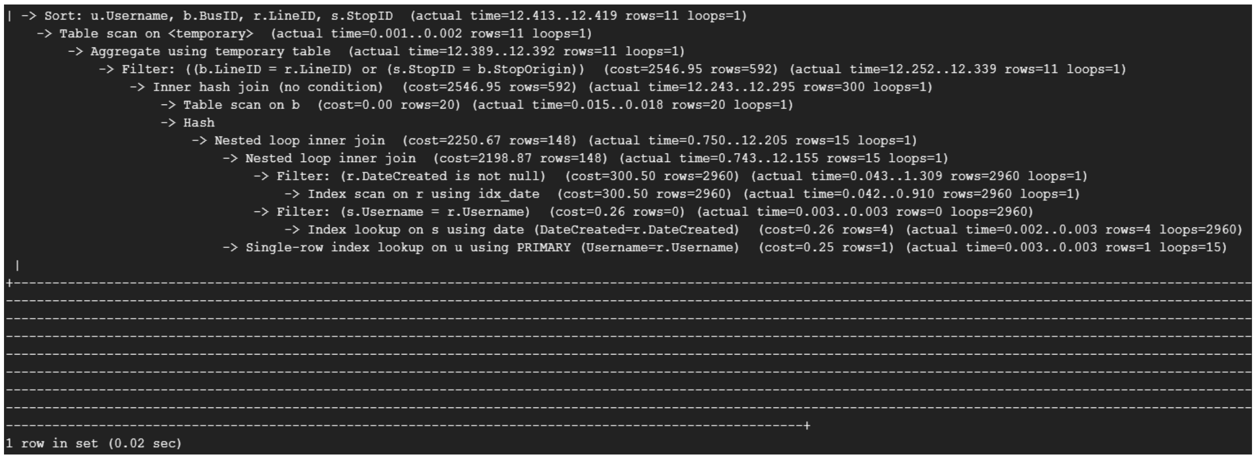
****NOTE****: Number of rows in set may change depending on what buses are currently being used as well as the randomized user's favorites. In this case, we found all buses which were being used at night (only 20), which may have caused a lower number of rows in set.

Original Explain Analysis:



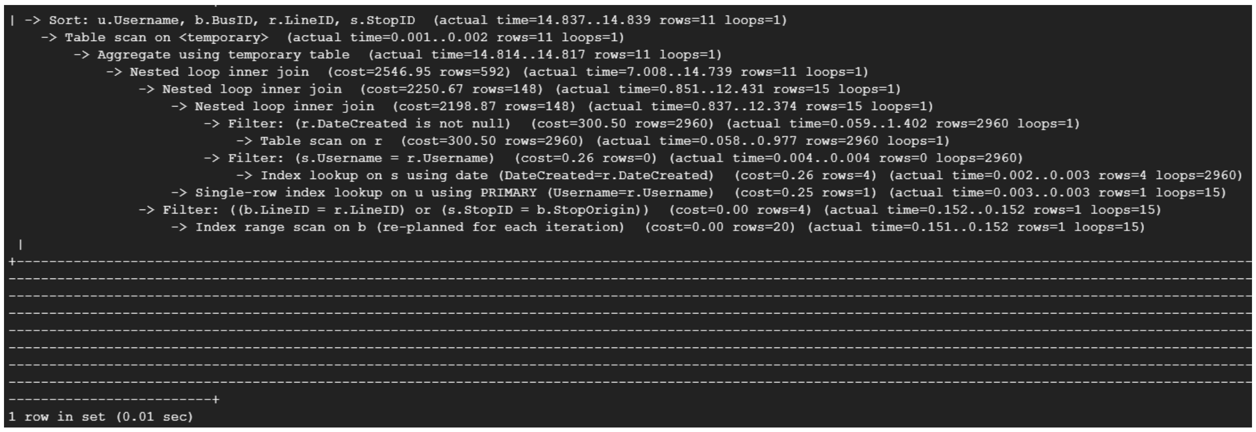
```
| -> Sort: u.Username, b.BusID, r.LineID, s.StopID  (actual time=12.200..12.202 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.002 rows=11 loops=1)
        -> Aggregate using temporary table  (actual time=12.180..12.183 rows=11 loops=1)
            -> Filter: ((b.LineID = r.LineID) or (s.StopID = b.StopOrigin))  (cost=2546.95 rows=592) (actual time=12.037..12.134 rows=11 loops=1)
                -> Inner hash join (no condition)  (cost=2546.95 rows=592) (actual time=12.021..12.090 rows=300 loops=1)
                    -> Table scan on b  (cost=0.00 rows=20) (actual time=0.013..0.022 rows=20 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=2250.67 rows=148) (actual time=0.732..11.989 rows=15 loops=1)
                            -> Nested loop inner join  (cost=2198.87 rows=148) (actual time=0.725..11.945 rows=15 loops=1)
                                -> Filter: (r.DateCreated is not null)  (cost=300.50 rows=2960) (actual time=0.042..1.339 rows=2960 loops=1)
                                    -> Table scan on r  (cost=300.50 rows=2960) (actual time=0.041..0.937 rows=2960 loops=1)
                                -> Filter: (s.Username = r.Username)  (cost=0.26 rows=0) (actual time=0.003..0.003 rows=0 loops=2960)
                                    -> Index lookup on s using date (DateCreated=r.DateCreated)  (cost=0.26 rows=4) (actual time=0.002..0.003 rows=4 loops=2960)
                            -> Single-row index lookup on u using PRIMARY (Username=r.Username)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

First Index added onto FavoriteRoutes-DateCreated
CREATE INDEX idx_date on FavoriteRoutes (DateCreated);



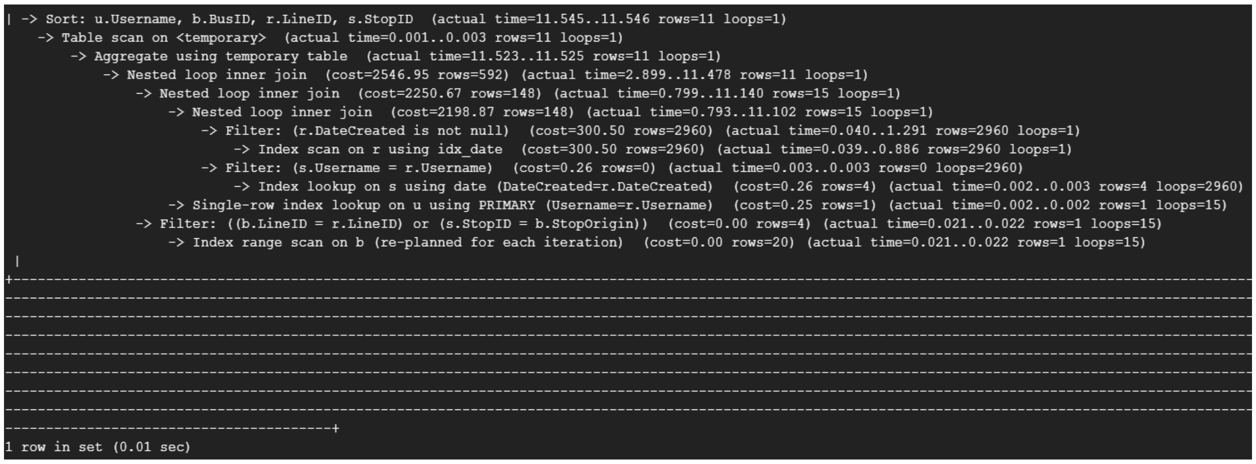
```
| -> Sort: u.Username, b.BusID, r.LineID, s.StopID  (actual time=12.413..12.419 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.002 rows=11 loops=1)
        -> Aggregate using temporary table  (actual time=12.389..12.392 rows=11 loops=1)
            -> Filter: ((b.LineID = r.LineID) or (s.StopID = b.StopOrigin))  (cost=2546.95 rows=592) (actual time=12.252..12.339 rows=11 loops=1)
                -> Inner hash join (no condition)  (cost=2546.95 rows=592) (actual time=12.243..12.295 rows=300 loops=1)
                    -> Table scan on b  (cost=0.00 rows=20) (actual time=0.015..0.018 rows=20 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=2250.67 rows=148) (actual time=0.750..12.205 rows=15 loops=1)
                            -> Nested loop inner join  (cost=2198.87 rows=148) (actual time=0.743..12.155 rows=15 loops=1)
                                -> Filter: (r.DateCreated is not null)  (cost=300.50 rows=2960) (actual time=0.043..1.309 rows=2960 loops=1)
                                    -> Index scan on r using idx_date  (cost=300.50 rows=2960) (actual time=0.042..0.910 rows=2960 loops=1)
                                -> Filter: (s.Username = r.Username)  (cost=0.26 rows=0) (actual time=0.003..0.003 rows=0 loops=2960)
                                    -> Index lookup on s using date (DateCreated=r.DateCreated)  (cost=0.26 rows=4) (actual time=0.002..0.003 rows=4 loops=2960)
                            -> Single-row index lookup on u using PRIMARY (Username=r.Username)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=15)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

Second Index added onto Buses-StopOrigin and StopDestination
CREATE INDEX idx_stop on Buses (StopOrign, StopDestination);



```
| -> Sort: u.Username, b.BusID, r.LineID, s.StopID  (actual time=14.837..14.839 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.002 rows=11 loops=1)
        -> Aggregate using temporary table  (actual time=14.814..14.817 rows=11 loops=1)
            -> Nested loop inner join  (cost=2546.95 rows=592) (actual time=7.008..14.739 rows=11 loops=1)
                -> Nested loop inner join  (cost=2250.67 rows=148) (actual time=0.851..12.431 rows=15 loops=1)
                    -> Nested loop inner join  (cost=2198.87 rows=148) (actual time=0.837..12.374 rows=15 loops=1)
                        -> Filter: (r.DateCreated is not null)  (cost=300.50 rows=2960) (actual time=0.059..1.402 rows=2960 loops=1)
                            -> Table scan on r  (cost=300.50 rows=2960) (actual time=0.058..0.977 rows=2960 loops=1)
                        -> Filter: (s.Username = r.Username)  (cost=0.26 rows=0) (actual time=0.004..0.004 rows=0 loops=2960)
                            -> Index lookup on s using date (DateCreated=r.DateCreated)  (cost=0.26 rows=4) (actual time=0.002..0.003 rows=4 loops=2960)
                    -> Single-row index lookup on u using PRIMARY (Username=r.Username)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=15)
                -> Filter: ((b.LineID = r.LineID) or (s.StopID = b.StopOrigin))  (cost=0.00 rows=4) (actual time=0.152..0.152 rows=1 loops=15)
                    -> Index range scan on b (re-planned for each iteration)  (cost=0.00 rows=20) (actual time=0.151..0.152 rows=1 loops=15)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------+
1 row in set (0.01 sec)
```

Third Index added onto Buses-StopOrigin and StopDestination as well as FavoriteRoutes-DateCreated and FavoriteStops-DateCreated

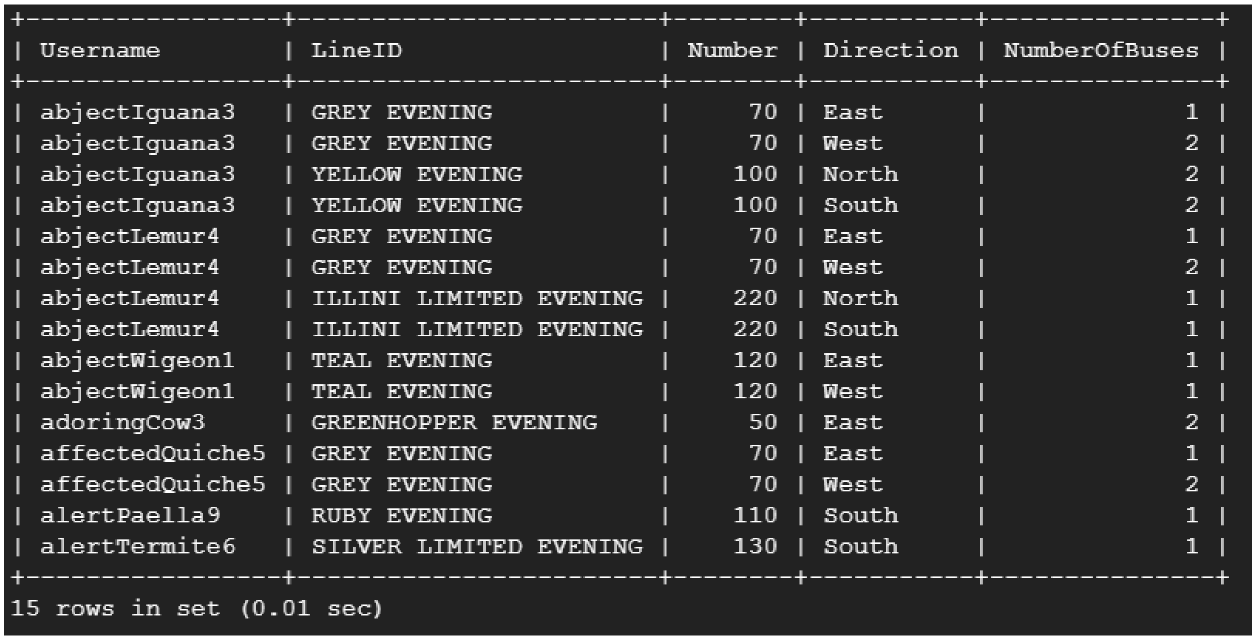


```
| -> Sort: u.Username, b.BusID, r.LineID, s.StopID  (actual time=11.545..11.546 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.003 rows=11 loops=1)
        -> Aggregate using temporary table  (actual time=11.523..11.525 rows=11 loops=1)
            -> Nested loop inner join  (cost=2546.95 rows=592) (actual time=2.899..11.478 rows=11 loops=1)
                -> Nested loop inner join  (cost=2250.67 rows=148) (actual time=0.799..11.140 rows=15 loops=1)
                    -> Nested loop inner join  (cost=2198.87 rows=148) (actual time=0.793..11.102 rows=15 loops=1)
                        -> Filter: (r.DateCreated is not null)  (cost=300.50 rows=2960) (actual time=0.040..1.291 rows=2960 loops=1)
                            -> Index scan on r using idx_date  (cost=300.50 rows=2960) (actual time=0.039..0.886 rows=2960 loops=1)
                        -> Filter: (s.Username = r.Username)  (cost=0.26 rows=0) (actual time=0.003..0.003 rows=0 loops=2960)
                            -> Index lookup on s using date (DateCreated=r.DateCreated)  (cost=0.26 rows=4) (actual time=0.002..0.003 rows=4 loops=2960)
                    -> Single-row index lookup on u using PRIMARY (Username=r.Username)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
                -> Filter: ((b.LineID = r.LineID) or (s.StopID = b.StopOrigin))  (cost=0.00 rows=4) (actual time=0.021..0.022 rows=1 loops=15)
                    -> Index range scan on b (re-planned for each iteration)  (cost=0.00 rows=20) (actual time=0.021..0.022 rows=1 loops=15)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------+
1 row in set (0.01 sec)
```

We will choose the third indexing design, as it provides the same speed up as the second indexing design, where the time it takes to execute the query goes from 0.02 seconds to 0.01 seconds. It combines this speed up along with the benefit of the first indexing design, where it changes the table scan on our FavoriteRoutes table into an index scan, reducing the time of the scan. The index on the stop origin and stop destination greatly help, as it causes the cost of the filter, one of the greatest costing attributes at 2546.95 to be reduced down to a cost of 0, with the time of it being reduced from 12.2 to 0.021. Furthermore, the first inner join's time is also reduced from 12.02 to 12.13, to 2.89 to 11.47, causing it to be much more efficient.

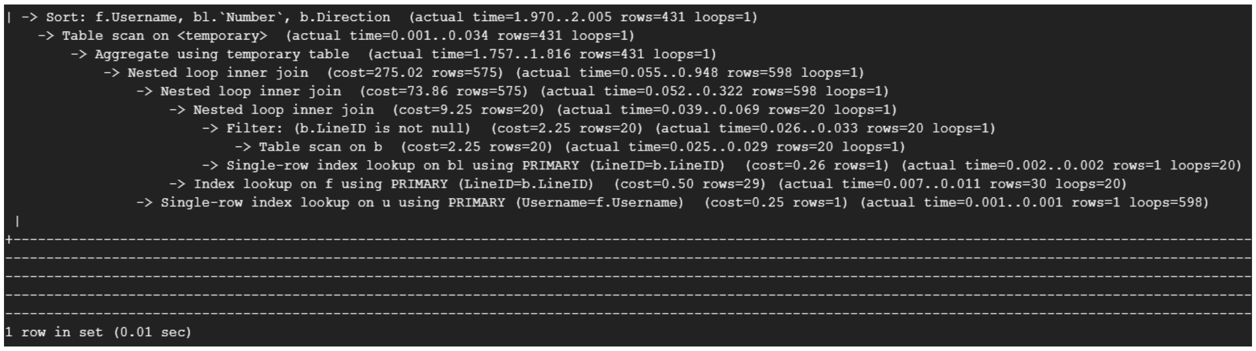
Advanced Query 2:
Retrieve all routes that are currently being utilized for each user, the information corresponding to that route, and the total number of buses that are following that route.

SELECT Username, LineID, Number, Direction, COUNT(LineID) AS NumberOfBuses
FROM Buses b NATURAL JOIN BusLine bl NATURAL JOIN FavoriteRoutes f NATURAL JOIN Users u
GROUP BY Username, LineID, Direction
ORDER BY Username, Number, Direction;



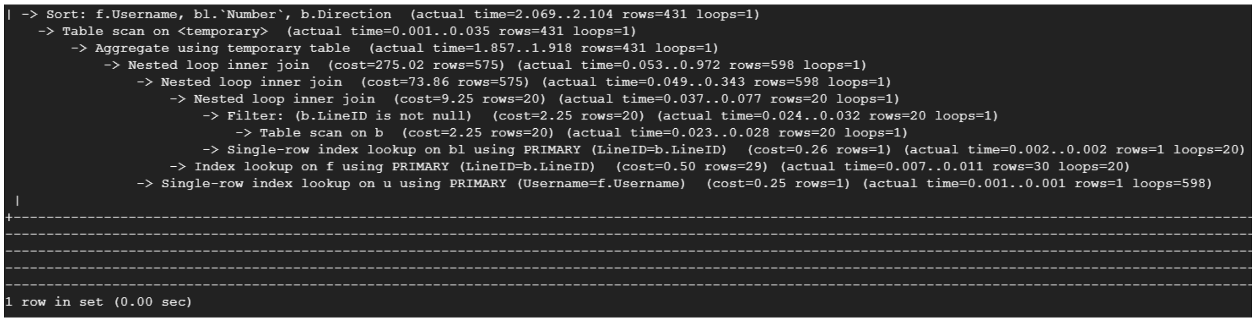
```
+-----------------+-----------------------+--------+-----------+---------------+
| Username        | LineID                | Number | Direction | NumberOfBuses |
+-----------------+-----------------------+--------+-----------+---------------+
| abjectIguana3   | GREY EVENING          |     70 | East      |             1 |
| abjectIguana3   | GREY EVENING          |     70 | West      |             2 |
| abjectIguana3   | YELLOW EVENING        |    100 | North     |             2 |
| abjectIguana3   | YELLOW EVENING        |    100 | South     |             2 |
| abjectLemur4    | GREY EVENING          |     70 | East      |             1 |
| abjectLemur4    | GREY EVENING          |     70 | West      |             2 |
| abjectLemur4    | ILLINI LIMITED EVENING |   220 | North     |             1 |
| abjectLemur4    | ILLINI LIMITED EVENING |   220 | South     |             1 |
| abjectWigeon1   | TEAL EVENING          |    120 | East      |             1 |
| abjectWigeon1   | TEAL EVENING          |    120 | West      |             1 |
| adoringCow3     | GREENHOPPER EVENING   |     50 | East      |             2 |
| affectedQuiche5 | GREY EVENING          |     70 | East      |             1 |
| affectedQuiche5 | GREY EVENING          |     70 | West      |             2 |
| alertPaella9    | RUBY EVENING          |    110 | South     |             1 |
| alertTermite6   | SILVER LIMITED EVENING |   130 | South     |             1 |
+-----------------+-----------------------+--------+-----------+---------------+
15 rows in set (0.01 sec)
```

Original Explain Analyze:



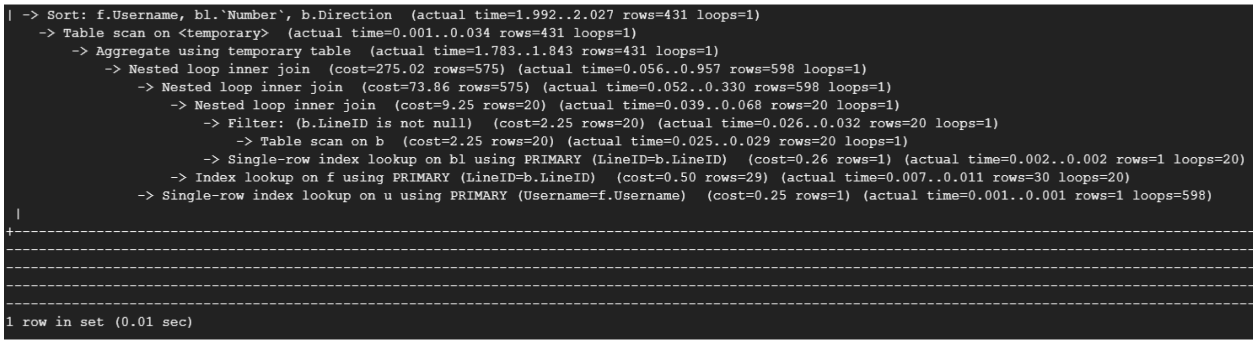
```
| -> Sort: f.Username, bl.`Number`, b.Direction  (actual time=1.970..2.005 rows=431 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.034 rows=431 loops=1)
        -> Aggregate using temporary table  (actual time=1.757..1.816 rows=431 loops=1)
            -> Nested loop inner join  (cost=275.02 rows=575) (actual time=0.055..0.948 rows=598 loops=1)
                -> Nested loop inner join  (cost=73.86 rows=575) (actual time=0.052..0.322 rows=598 loops=1)
                    -> Nested loop inner join  (cost=9.25 rows=20) (actual time=0.039..0.069 rows=20 loops=1)
                        -> Filter: (b.LineID is not null)  (cost=2.25 rows=20) (actual time=0.026..0.033 rows=20 loops=1)
                            -> Table scan on b  (cost=2.25 rows=20) (actual time=0.025..0.029 rows=20 loops=1)
                        -> Single-row index lookup on bl using PRIMARY (LineID=b.LineID)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=20)
                    -> Index lookup on f using PRIMARY (LineID=b.LineID)  (cost=0.50 rows=29) (actual time=0.007..0.011 rows=30 loops=20)
                -> Single-row index lookup on u using PRIMARY (Username=f.Username)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=598)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

First index was added on Buses-Shape
CREATE INDEX idx_shape on Buses(ShapeID);



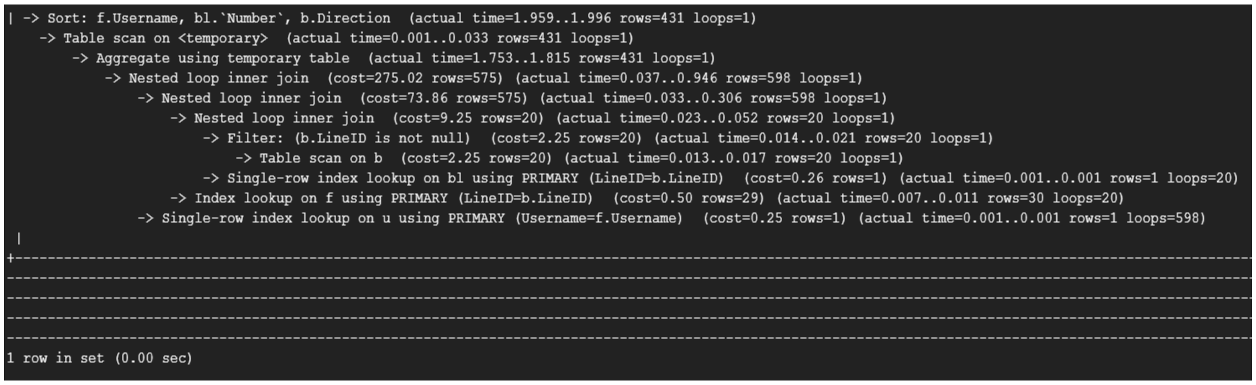
```
| -> Sort: f.Username, bl.`Number`, b.Direction  (actual time=2.069..2.104 rows=431 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.035 rows=431 loops=1)
        -> Aggregate using temporary table  (actual time=1.857..1.918 rows=431 loops=1)
            -> Nested loop inner join  (cost=275.02 rows=575) (actual time=0.053..0.972 rows=598 loops=1)
                -> Nested loop inner join  (cost=73.86 rows=575) (actual time=0.049..0.343 rows=598 loops=1)
                    -> Nested loop inner join  (cost=9.25 rows=20) (actual time=0.037..0.077 rows=20 loops=1)
                        -> Filter: (b.LineID is not null)  (cost=2.25 rows=20) (actual time=0.024..0.032 rows=20 loops=1)
                            -> Table scan on b  (cost=2.25 rows=20) (actual time=0.023..0.028 rows=20 loops=1)
                        -> Single-row index lookup on bl using PRIMARY (LineID=b.LineID)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=20)
                    -> Index lookup on f using PRIMARY (LineID=b.LineID)  (cost=0.50 rows=29) (actual time=0.007..0.011 rows=30 loops=20)
                -> Single-row index lookup on u using PRIMARY (Username=f.Username)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=598)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

Second index was added on Buses-Direction:
CREATE INDEX idx_dir on Buses (Direction);



```
| -> Sort: f.Username, bl.`Number`, b.Direction  (actual time=1.992..2.027 rows=431 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.034 rows=431 loops=1)
        -> Aggregate using temporary table  (actual time=1.783..1.843 rows=431 loops=1)
            -> Nested loop inner join  (cost=275.02 rows=575) (actual time=0.056..0.957 rows=598 loops=1)
                -> Nested loop inner join  (cost=73.86 rows=575) (actual time=0.052..0.330 rows=598 loops=1)
                    -> Nested loop inner join  (cost=9.25 rows=20) (actual time=0.039..0.068 rows=20 loops=1)
                        -> Filter: (b.LineID is not null)  (cost=2.25 rows=20) (actual time=0.026..0.032 rows=20 loops=1)
                            -> Table scan on b  (cost=2.25 rows=20) (actual time=0.025..0.029 rows=20 loops=1)
                        -> Single-row index lookup on bl using PRIMARY (LineID=b.LineID)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=20)
                    -> Index lookup on f using PRIMARY (LineID=b.LineID)  (cost=0.50 rows=29) (actual time=0.007..0.011 rows=30 loops=20)
                -> Single-row index lookup on u using PRIMARY (Username=f.Username)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=598)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

Third index was added on BusLine-Number and Color
CREATE INDEX idx_num_color on BusLine (Number, Color);



```
| -> Sort: f.Username, bl.`Number`, b.Direction  (actual time=1.959..1.996 rows=431 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.033 rows=431 loops=1)
        -> Aggregate using temporary table  (actual time=1.753..1.815 rows=431 loops=1)
            -> Nested loop inner join  (cost=275.02 rows=575) (actual time=0.037..0.946 rows=598 loops=1)
                -> Nested loop inner join  (cost=73.86 rows=575) (actual time=0.033..0.306 rows=598 loops=1)
                    -> Nested loop inner join  (cost=9.25 rows=20) (actual time=0.023..0.052 rows=20 loops=1)
                        -> Filter: (b.LineID is not null)  (cost=2.25 rows=20) (actual time=0.014..0.021 rows=20 loops=1)
                            -> Table scan on b  (cost=2.25 rows=20) (actual time=0.013..0.017 rows=20 loops=1)
                        -> Single-row index lookup on bl using PRIMARY (LineID=b.LineID)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=20)
                    -> Index lookup on f using PRIMARY (LineID=b.LineID)  (cost=0.50 rows=29) (actual time=0.007..0.011 rows=30 loops=20)
                -> Single-row index lookup on u using PRIMARY (Username=f.Username)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=598)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

We chose the third indexing design, where we add an index to both the BusLine's number and color. This is because it creates an increase in speed from 0.01 to 0.00 seconds from the original. Although minimal, any increase in speed is good. Because both the first and third indexing design have this increase in speed, we look further into the analysis. The highest cost in both designs is the first nested loop inner join. However, in the first design, we see that the time it takes is between 0.053 and 0.972, however, in the third design, we see that the actual time is 0.037 and 0.946. As such, the third design could be a bit faster than the first. A very apparent change is also seen in the time of the table scan on b, where the time in the third design is 0.013 to 0.017, but in the first, it is 0.023 to 0.028. This is a very drastic difference, nearly double the speed. As a result, we choose the third indexing design due to these increases in speed. It is helps that a bus line's number and color is very correlated to its Line ID, and thus is efficient to find.