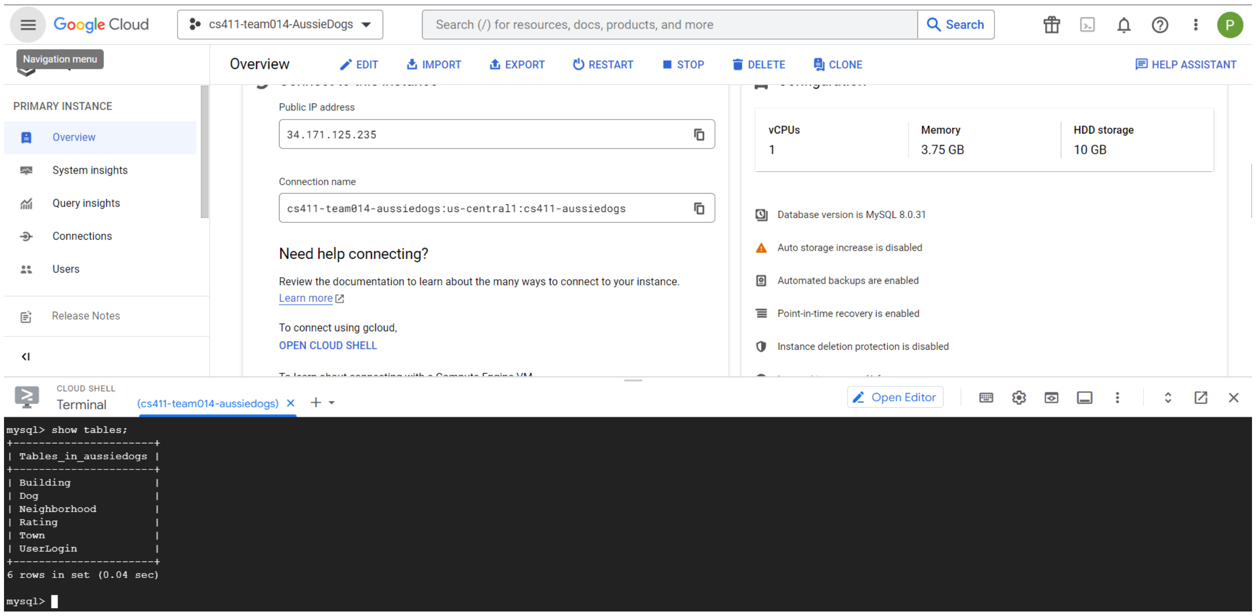
Database Screenshots:



Public IP address

34.171.125.235

Connection name

cs411-team014-aussiedogs:us-central1:cs411-aussiedogs

Need help connecting?

Review the documentation to learn about the many ways to connect to your instance. Learn more

To connect using gcloud, OPEN CLOUD SHELL

vCPUs	Memory	HDD storage
1	3.75 GB	10 GB

Database version is MySQL 8.0.31

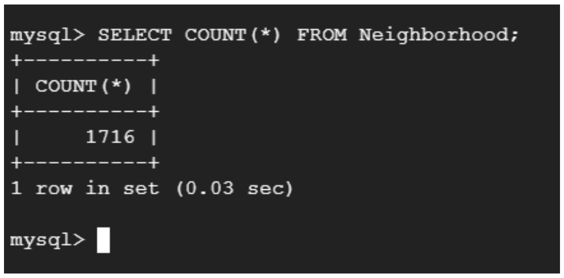
Auto storage increase is disabled

Automated backups are enabled

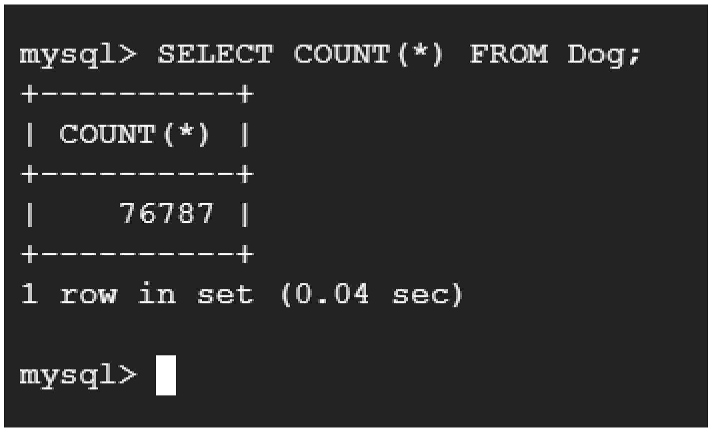
Point-in-time recovery is enabled

Instance deletion protection is disabled

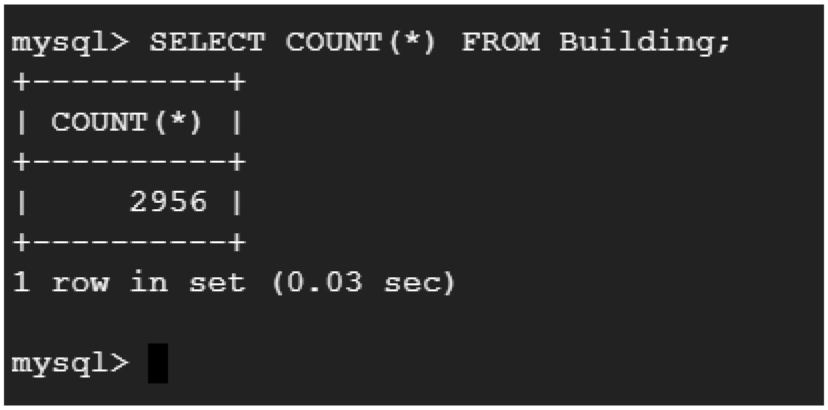
```
mysql> show tables;
+----------------------+
| Tables_in_aussiedogs |
+----------------------+
| Building             |
| Dog                  |
| Neighborhood         |
| Rating               |
| Town                 |
| UserLogin            |
+----------------------+
6 rows in set (0.04 sec)

mysql> 
```



```
mysql> SELECT COUNT(*) FROM Neighborhood;
+----------+
| COUNT(*) |
+----------+
|     1716 |
+----------+
1 row in set (0.03 sec)

mysql> 
```



```
mysql> SELECT COUNT(*) FROM Dog;
+----------+
| COUNT(*) |
+----------+
|    76787 |
+----------+
1 row in set (0.04 sec)

mysql> 
```



```
mysql> SELECT COUNT(*) FROM Building;
+----------+
| COUNT(*) |
+----------+
|     2956 |
+----------+
1 row in set (0.03 sec)

mysql> 
```

DDL Commands

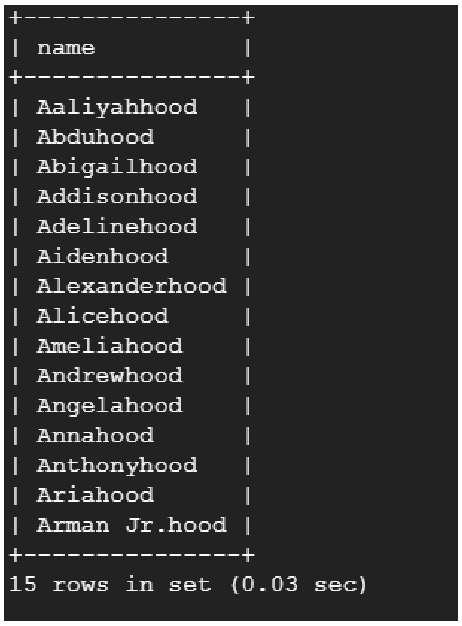
```sql
CREATE TABLE Town(TownName VARCHAR(255) NOT NULL PRIMARY KEY, State
VARCHAR(255), Country VARCHAR(255));

CREATE TABLE Rating(TownName VARCHAR(255) NOT NULL, Username VARCHAR(255)
NOT NULL, Type VARCHAR(255), Value INT, PRIMARY KEY(TownName, Username),
FOREIGN KEY(TownName) REFERENCES Town(TownName), FOREIGN KEY(Username)
REFERENCES UserLogin(Username));

CREATE TABLE UserLogin(Username VARCHAR(255) NOT NULL PRIMARY KEY, Email
VARCHAR(255), Phone INT(15), Password VARCHAR(128));

CREATE TABLE Dog(ReferenceID INT NOT NULL, Animal_Name VARCHAR(255), Breed
VARCHAR(255), Suburb VARCHAR(255), BirthYear INT, Gender VARCHAR(6), Latitude REAL,
Longitude REAL, PRIMARY KEY(ReferenceID));

CREATE TABLE Dog(ReferenceID INT NOT NULL, Owner VARCHAR(255), Neighborhood
VARCHAR(255), Suburb VARCHAR(255), Name VARCHAR(255), Breed VARCHAR(255),
BirthYear INT, Desexed_Microchip BOOLEAN, Gender VARCHAR(6), PRIMARY
KEY(ReferenceID), FOREIGN KEY(Owner) REFERENCES UserLogin(Username));

CREATE TABLE Neighborhood(Name VARCHAR(255) NOT NULL, Town VARCHAR(255) NOT
NULL, Latitude REAL, Longitude REAL, PRIMARY KEY(Name, Town), FOREIGN KEY(Town)
REFERENCES Town(TownName) ON DELETE CASCADE);

CREATE TABLE Building(Name VARCHAR(255) NOT NULL, Town VARCHAR(255) NOT
NULL, Latitude REAL, Longitude REAL, Type VARCHAR(255), PRIMARY KEY(Name, Town),
FOREIGN KEY(Town) REFERENCES Town(TownName) ON DELETE CASCADE);
```

Advance Queries
Neighborhoods in towns with certain types of buildings

(SELECT Neighborhood.name
FROM Neighborhood JOIN Building ON Neighborhood.Town = Building.Town
WHERE Building.Type LIKE ‘Park%’;)

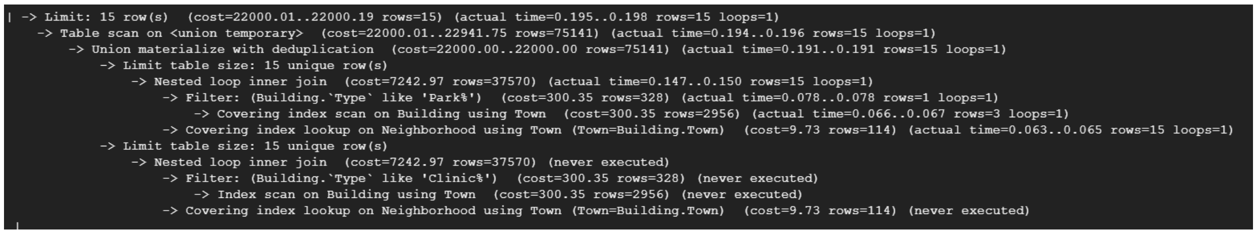
UNION

(SELECT Neighborhood.name
FROM Neighborhood JOIN Building Neighborhood.Town = Building.Town
WHERE Building.Type LIKE ‘Clinic%’)
LIMIT 15;



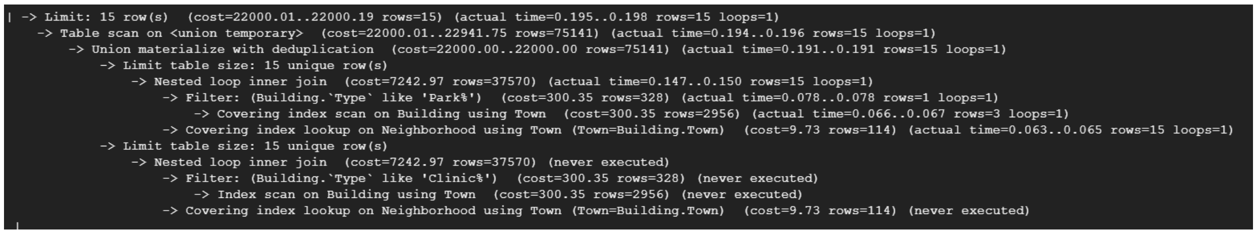
```
+---------------+
| name          |
+---------------+
| Aaliyahhood   |
| Abduhood      |
| Abigailhood   |
| Addisonhood   |
| Adelinehood   |
| Aidenhood     |
| Alexanderhood |
| Alicehood     |
| Ameliahood    |
| Andrewhood    |
| Angelahood    |
| Annahood      |
| Anthonyhood   |
| Ariahood      |
| Arman Jr.hood |
+---------------+
15 rows in set (0.03 sec)
```

No index



```
| -> Limit: 15 row(s)  (cost=22000.01..22000.19 rows=15) (actual time=0.195..0.198 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=22000.01..22941.75 rows=75141) (actual time=0.194..0.196 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=22000.00..22000.00 rows=75141) (actual time=0.191..0.191 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=7242.97 rows=37570) (actual time=0.147..0.150 rows=15 loops=1)
                    -> Filter: (Building.`Type` like 'Park%')  (cost=300.35 rows=328) (actual time=0.078..0.078 rows=1 loops=1)
                        -> Covering index scan on Building using Town  (cost=300.35 rows=2956) (actual time=0.066..0.067 rows=3 loops=1)
                    -> Covering index lookup on Neighborhood using Town (Town=Building.Town)  (cost=9.73 rows=114) (actual time=0.063..0.065 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=7242.97 rows=37570) (never executed)
                    -> Filter: (Building.`Type` like 'Clinic%')  (cost=300.35 rows=328) (never executed)
                        -> Index scan on Building using Town  (cost=300.35 rows=2956) (never executed)
                    -> Covering index lookup on Neighborhood using Town (Town=Building.Town)  (cost=9.73 rows=114) (never executed)
 |
```

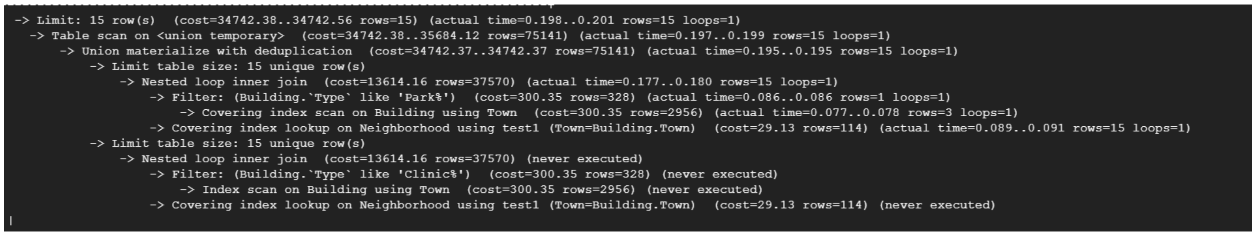
Added index on Town



```
| -> Limit: 15 row(s)  (cost=22000.01..22000.19 rows=15) (actual time=0.195..0.198 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=22000.01..22941.75 rows=75141) (actual time=0.194..0.196 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=22000.00..22000.00 rows=75141) (actual time=0.191..0.191 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=7242.97 rows=37570) (actual time=0.147..0.150 rows=15 loops=1)
                    -> Filter: (Building.`Type` like 'Park%')  (cost=300.35 rows=328) (actual time=0.078..0.078 rows=1 loops=1)
                        -> Covering index scan on Building using Town  (cost=300.35 rows=2956) (actual time=0.066..0.067 rows=3 loops=1)
                    -> Covering index lookup on Neighborhood using Town (Town=Building.Town)  (cost=9.73 rows=114) (actual time=0.063..0.065 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Nested loop inner join  (cost=7242.97 rows=37570) (never executed)
                    -> Filter: (Building.`Type` like 'Clinic%')  (cost=300.35 rows=328) (never executed)
                        -> Index scan on Building using Town  (cost=300.35 rows=2956) (never executed)
                    -> Covering index lookup on Neighborhood using Town (Town=Building.Town)  (cost=9.73 rows=114) (never executed)
 |
```

No change on the actual data speed. This is because Town is already part of the primary key constraint on the Neighborhood table;

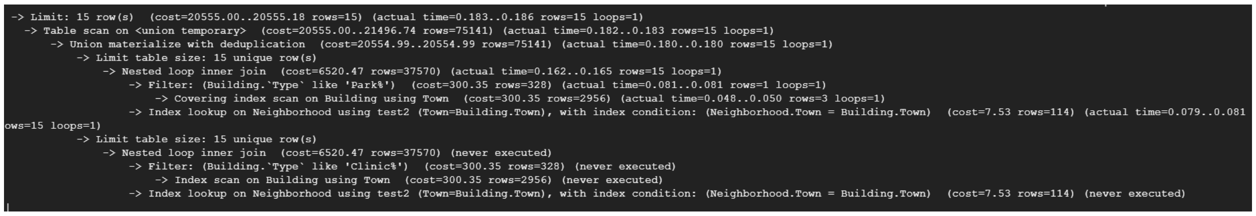
Added index to Neighborhood on Latitude and Longitude



```
-> Limit: 15 row(s)  (cost=34742.38..34742.56 rows=15) (actual time=0.198..0.201 rows=15 loops=1)
  -> Table scan on <union temporary>  (cost=34742.38..35684.12 rows=75141) (actual time=0.197..0.199 rows=15 loops=1)
      -> Union materialize with deduplication  (cost=34742.37..34742.37 rows=75141) (actual time=0.195..0.195 rows=15 loops=1)
          -> Limit table size: 15 unique row(s)
              -> Nested loop inner join  (cost=13614.16 rows=37570) (actual time=0.177..0.180 rows=15 loops=1)
                  -> Filter: (Building.`Type` like 'Park%')  (cost=300.35 rows=328) (actual time=0.086..0.086 rows=1 loops=1)
                      -> Covering index scan on Building using Town  (cost=300.35 rows=2956) (actual time=0.077..0.078 rows=3 loops=1)
                  -> Covering index lookup on Neighborhood using test1 (Town=Building.Town)  (cost=29.13 rows=114) (actual time=0.089..0.091 rows=15 loops=1)
          -> Limit table size: 15 unique row(s)
              -> Nested loop inner join  (cost=13614.16 rows=37570) (never executed)
                  -> Filter: (Building.`Type` like 'Clinic%')  (cost=300.35 rows=328) (never executed)
                      -> Index scan on Building using Town  (cost=300.35 rows=2956) (never executed)
                  -> Covering index lookup on Neighborhood using test1 (Town=Building.Town)  (cost=29.13 rows=114) (never executed)
|
```

This index actually added time to the overall search, probably because the latitude and longitude data do not help separate the data based on the parameters we are searching for.

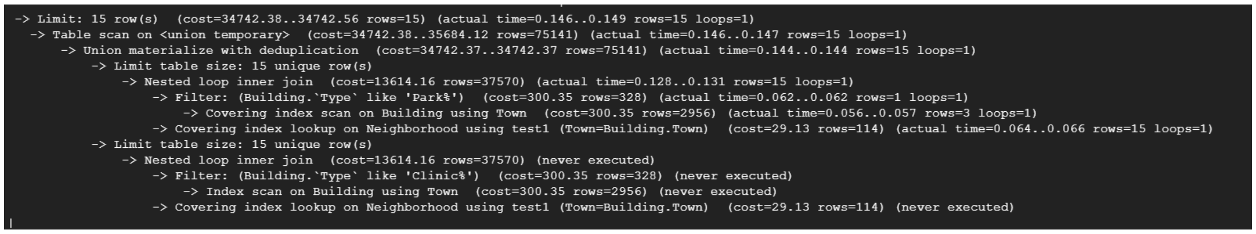
Added index to Neighborhood on Town(5)



```
-> Limit: 15 row(s)  (cost=20555.00..20555.18 rows=15) (actual time=0.183..0.186 rows=15 loops=1)
  -> Table scan on <union temporary>  (cost=20555.00..21496.74 rows=75141) (actual time=0.182..0.183 rows=15 loops=1)
      -> Union materialize with deduplication  (cost=20554.99..20554.99 rows=75141) (actual time=0.180..0.180 rows=15 loops=1)
          -> Limit table size: 15 unique row(s)
              -> Nested loop inner join  (cost=6520.47 rows=37570) (actual time=0.162..0.165 rows=15 loops=1)
                  -> Filter: (Building.`Type` like 'Park%')  (cost=300.35 rows=328) (actual time=0.081..0.081 rows=1 loops=1)
                      -> Covering index scan on Building using Town  (cost=300.35 rows=2956) (actual time=0.048..0.050 rows=3 loops=1)
                  -> Index lookup on Neighborhood using test2 (Town=Building.Town), with index condition: (Neighborhood.Town = Building.Town)  (cost=7.53 rows=114) (actual time=0.079..0.081
ows=15 loops=1)
          -> Limit table size: 15 unique row(s)
              -> Nested loop inner join  (cost=6520.47 rows=37570) (never executed)
                  -> Filter: (Building.`Type` like 'Clinic%')  (cost=300.35 rows=328) (never executed)
                      -> Index scan on Building using Town  (cost=300.35 rows=2956) (never executed)
                  -> Index lookup on Neighborhood using test2 (Town=Building.Town), with index condition: (Neighborhood.Town = Building.Town)  (cost=7.53 rows=114) (never executed)
|
```

This sped up the query very slightly. This probably helps group the data because we are not using the entire town now, only the first 5 characters.

Added index to Neighborhood on Name(8)



```
-> Limit: 15 row(s)  (cost=34742.38..34742.56 rows=15) (actual time=0.146..0.149 rows=15 loops=1)
  -> Table scan on <union temporary>  (cost=34742.38..35684.12 rows=75141) (actual time=0.146..0.147 rows=15 loops=1)
      -> Union materialize with deduplication  (cost=34742.37..34742.37 rows=75141) (actual time=0.144..0.144 rows=15 loops=1)
          -> Limit table size: 15 unique row(s)
              -> Nested loop inner join  (cost=13614.16 rows=37570) (actual time=0.128..0.131 rows=15 loops=1)
                  -> Filter: (Building.`Type` like 'Park%')  (cost=300.35 rows=328) (actual time=0.062..0.062 rows=1 loops=1)
                      -> Covering index scan on Building using Town  (cost=300.35 rows=2956) (actual time=0.056..0.057 rows=3 loops=1)
                  -> Covering index lookup on Neighborhood using test1 (Town=Building.Town)  (cost=29.13 rows=114) (actual time=0.064..0.066 rows=15 loops=1)
          -> Limit table size: 15 unique row(s)
              -> Nested loop inner join  (cost=13614.16 rows=37570) (never executed)
                  -> Filter: (Building.`Type` like 'Clinic%')  (cost=300.35 rows=328) (never executed)
                      -> Index scan on Building using Town  (cost=300.35 rows=2956) (never executed)
                  -> Covering index lookup on Neighborhood using test1 (Town=Building.Town)  (cost=29.13 rows=114) (never executed)
|
```

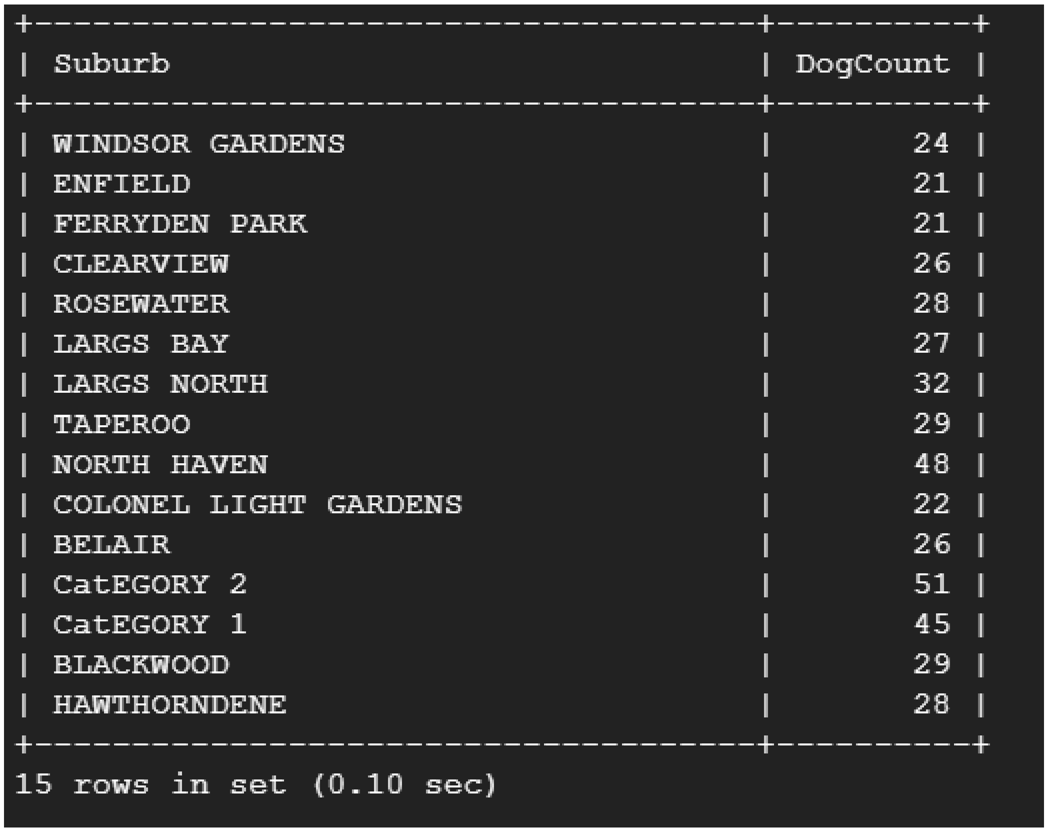
This sped up the query the most. We are down almost 25% from our initial query. This likely categorizes the data the best because we are selecting the names of the neighborhoods for each subquery.

Ultimately, we will end up using the index based on Name(8). This is for two main reasons:

1) We will often be working with the Town information of a neighborhood, but since that is already a primary key for the table, we will not have to specify any other indices related to this column/information.
2) We will likely be returning neighborhood name information quite frequently. If adding this index significantly improves the overall performance of these queries, we can end up saving a lot of time and resources.

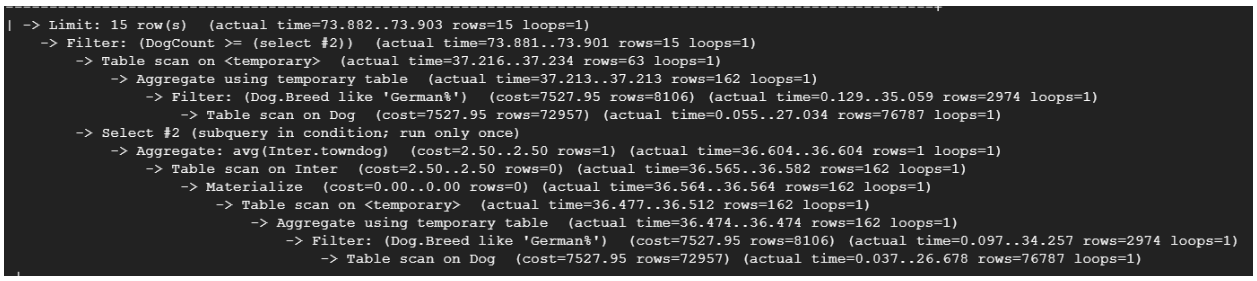
Find number of dogs in towns with above average German Shepherd

```
Select Suburb, Count(*) as DogCount
FROM Dog
Where Breed LIKE 'German%'
GROUP BY Suburb
HAVING DogCount >= (SELECT AVG(towndog)
                                        FROM (SELECT Count(*) AS towndog
                                                 FROM Dog WHERE Breed LIKE 'German%'
                                                GROUP BY Suburb) AS Inter)
LIMIT 15;
```



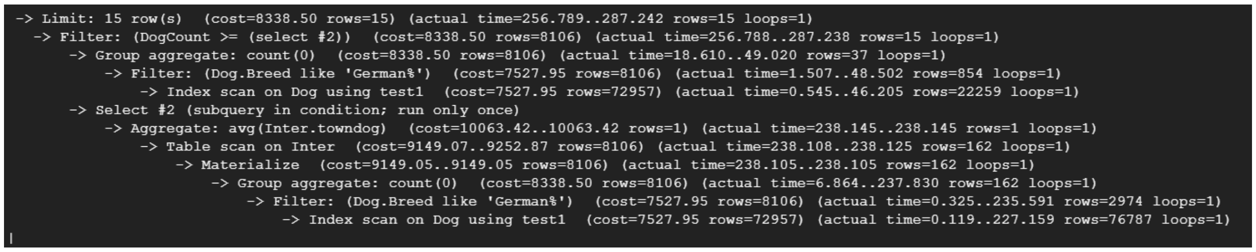
```
+--------------------------------------+----------+
| Suburb                               | DogCount |
+--------------------------------------+----------+
| WINDSOR GARDENS                      |       24 |
| ENFIELD                              |       21 |
| FERRYDEN PARK                        |       21 |
| CLEARVIEW                            |       26 |
| ROSEWATER                            |       28 |
| LARGS BAY                            |       27 |
| LARGS NORTH                          |       32 |
| TAPEROO                              |       29 |
| NORTH HAVEN                          |       48 |
| COLONEL LIGHT GARDENS                |       22 |
| BELAIR                               |       26 |
| CatEGORY 2                           |       51 |
| CatEGORY 1                           |       45 |
| BLACKWOOD                            |       29 |
| HAWTHORNDENE                         |       28 |
+--------------------------------------+----------+
15 rows in set (0.10 sec)
```

No index



```
| -> Limit: 15 row(s)  (actual time=73.882..73.903 rows=15 loops=1)
    -> Filter: (DogCount >= (select #2))  (actual time=73.881..73.901 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=37.216..37.234 rows=63 loops=1)
            -> Aggregate using temporary table  (actual time=37.213..37.213 rows=162 loops=1)
                -> Filter: (Dog.Breed like 'German%')  (cost=7527.95 rows=8106) (actual time=0.129..35.059 rows=2974 loops=1)
                    -> Table scan on Dog  (cost=7527.95 rows=72957) (actual time=0.055..27.034 rows=76787 loops=1)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(Inter.towndog)  (cost=2.50..2.50 rows=1) (actual time=36.604..36.604 rows=1 loops=1)
                -> Table scan on Inter  (cost=2.50..2.50 rows=0) (actual time=36.565..36.582 rows=162 loops=1)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=36.564..36.564 rows=162 loops=1)
                        -> Table scan on <temporary>  (actual time=36.477..36.512 rows=162 loops=1)
                            -> Aggregate using temporary table  (actual time=36.474..36.474 rows=162 loops=1)
                                -> Filter: (Dog.Breed like 'German%')  (cost=7527.95 rows=8106) (actual time=0.097..34.257 rows=2974 loops=1)
                                    -> Table scan on Dog  (cost=7527.95 rows=72957) (actual time=0.037..26.678 rows=76787 loops=1)
```

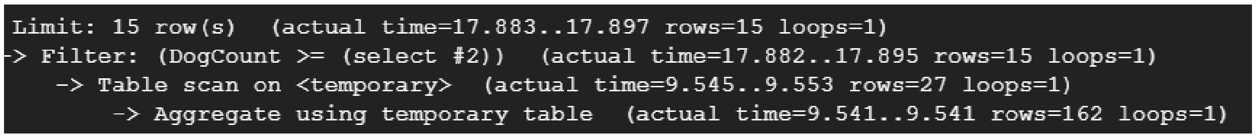
Added index to Dog on Suburb



```
-> Limit: 15 row(s)  (cost=8338.50 rows=15) (actual time=256.789..287.242 rows=15 loops=1)
  -> Filter: (DogCount >= (select #2))  (cost=8338.50 rows=8106) (actual time=256.788..287.238 rows=15 loops=1)
      -> Group aggregate: count(0)  (cost=8338.50 rows=8106) (actual time=18.610..49.020 rows=37 loops=1)
          -> Filter: (Dog.Breed like 'German%')  (cost=7527.95 rows=8106) (actual time=1.507..48.502 rows=854 loops=1)
              -> Index scan on Dog using test1  (cost=7527.95 rows=72957) (actual time=0.545..46.205 rows=22259 loops=1)
      -> Select #2 (subquery in condition; run only once)
          -> Aggregate: avg(Inter.towndog)  (cost=10063.42..10063.42 rows=1) (actual time=238.145..238.145 rows=1 loops=1)
              -> Table scan on Inter  (cost=9149.07..9252.87 rows=8106) (actual time=238.108..238.125 rows=162 loops=1)
                  -> Materialize  (cost=9149.05..9149.05 rows=8106) (actual time=238.105..238.105 rows=162 loops=1)
                      -> Group aggregate: count(0)  (cost=8338.50 rows=8106) (actual time=6.864..237.830 rows=162 loops=1)
                          -> Filter: (Dog.Breed like 'German%')  (cost=7527.95 rows=8106) (actual time=0.325..235.591 rows=2974 loops=1)
                              -> Index scan on Dog using test1  (cost=7527.95 rows=72957) (actual time=0.119..227.159 rows=76787 loops=1)
|
```

Learning from our first advanced query, we thought indexing by suburb would help reduce the overall time because we are selecting for suburb information at the end. However, because we are running subqueries without using suburb names (simply using aggregated values), this ended up increasing the time by a factor of 3x.

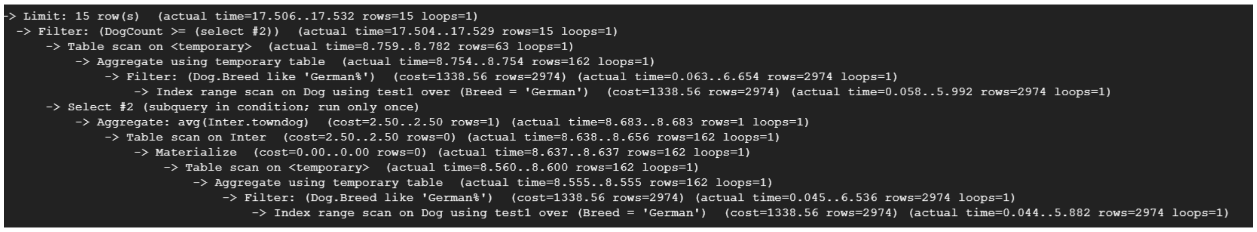
Added index to Dog on Breed



```
Limit: 15 row(s)  (actual time=17.883..17.897 rows=15 loops=1)
-> Filter: (DogCount >= (select #2))  (actual time=17.882..17.895 rows=15 loops=1)
   -> Table scan on <temporary>  (actual time=9.545..9.553 rows=27 loops=1)
       -> Aggregate using temporary table  (actual time=9.541..9.541 rows=162 loops=1)
```

This index resulted in HUGE performance gains, bringing the actual time down from ~70 to ~17. This is likely due to the fact that we are repeatedly using aggregate information based on breeds, so adding this index streamlines the execution of both the subquery and the outer query.

Added index to Dog on Breed(6)



```
-> Limit: 15 row(s)  (actual time=17.506..17.532 rows=15 loops=1)
 -> Filter: (DogCount >= (select #2))  (actual time=17.504..17.529 rows=15 loops=1)
     -> Table scan on <temporary>  (actual time=8.759..8.782 rows=63 loops=1)
         -> Aggregate using temporary table  (actual time=8.754..8.754 rows=162 loops=1)
             -> Filter: (Dog.Breed like 'German%')  (cost=1338.56 rows=2974) (actual time=0.063..6.654 rows=2974 loops=1)
                 -> Index range scan on Dog using test1 over (Breed = 'German')  (cost=1338.56 rows=2974) (actual time=0.058..5.992 rows=2974 loops=1)
     -> Select #2 (subquery in condition; run only once)
         -> Aggregate: avg(Inter.towndog)  (cost=2.50..2.50 rows=1) (actual time=8.683..8.683 rows=1 loops=1)
             -> Table scan on Inter  (cost=2.50..2.50 rows=0) (actual time=8.638..8.656 rows=162 loops=1)
                 -> Materialize  (cost=0.00..0.00 rows=0) (actual time=8.637..8.637 rows=162 loops=1)
                     -> Table scan on <temporary>  (actual time=8.560..8.600 rows=162 loops=1)
                         -> Aggregate using temporary table  (actual time=8.555..8.555 rows=162 loops=1)
                             -> Filter: (Dog.Breed like 'German%')  (cost=1338.56 rows=2974) (actual time=0.045..6.536 rows=2974 loops=1)
                                 -> Index range scan on Dog using test1 over (Breed = 'German')  (cost=1338.56 rows=2974) (actual time=0.044..5.882 rows=2974 loops=1)
```

Building off of the previous index choice, we thought about limiting the Breed index to just the first 6 characters (since we are searching for breeds starting with the word “German”). However, instead of improving the performance like we thought, there really was no noticeable difference between the two index options.

As a result, we will simply be using the index on Breed. There was no noticeable performance boost when limiting the index to just the first 6 characters (and upon further testing the same is true for a variety of values instead of just 6). Furthermore, it is not guaranteed that other queries we perform on this table will rely on only a portion of the name. Instead, we might want to query for a very specific breed of dog (e.g. Labrador Retriever Cross), though being able to efficiently aggregate information over the entire breed will nonetheless be very useful.