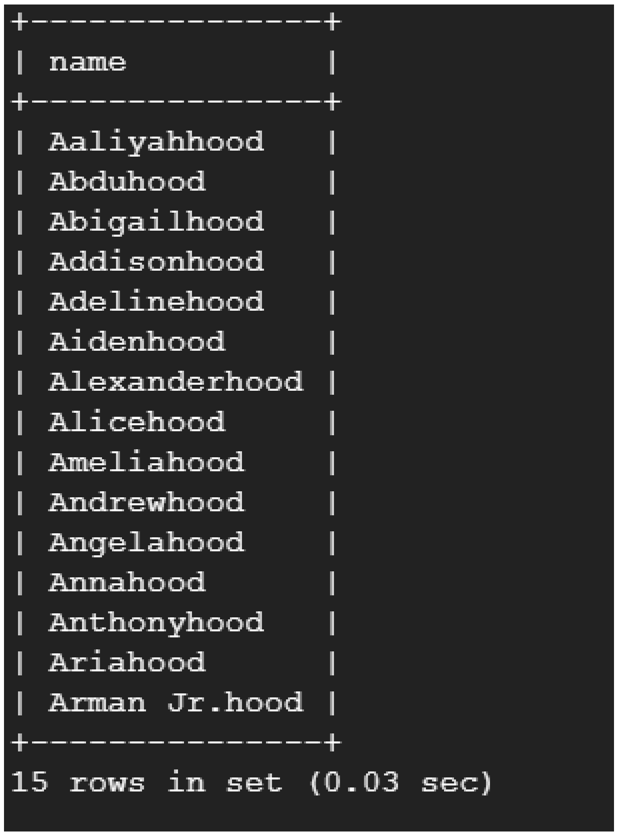
eDDL Commands

```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

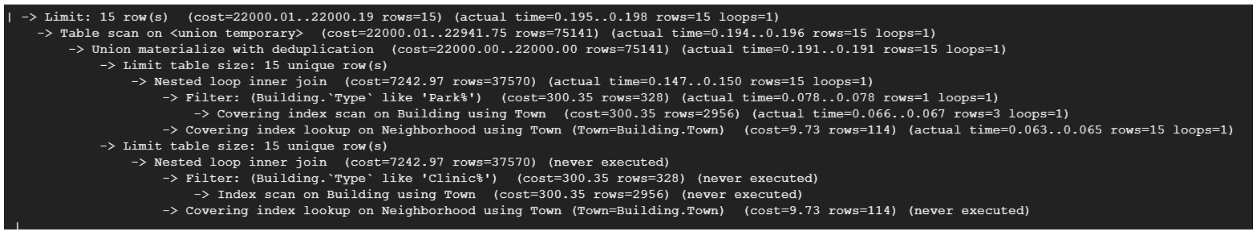
Neighborhoods in towns with certain types of buildings

```
(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;
```



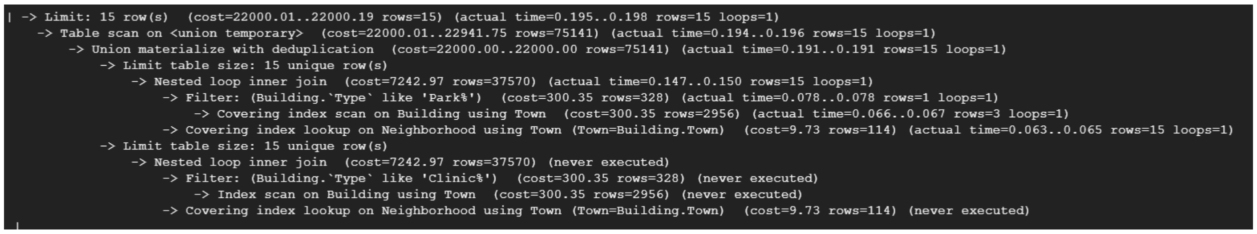
```
+---------------+
| name          |
+---------------+
| Aaliyahhood   |
| Abduhood      |
| Abigailhood   |
| Addisonhood   |
| Adelinehood   |
| Aidenhood     |
| Alexanderhood |
| Alicehood     |
| Ameliahood    |
| Andrewhood    |
| Angelahood    |
| Annahood      |
| Anthonyhood   |
| Ariahood      |
| 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)
 |
```

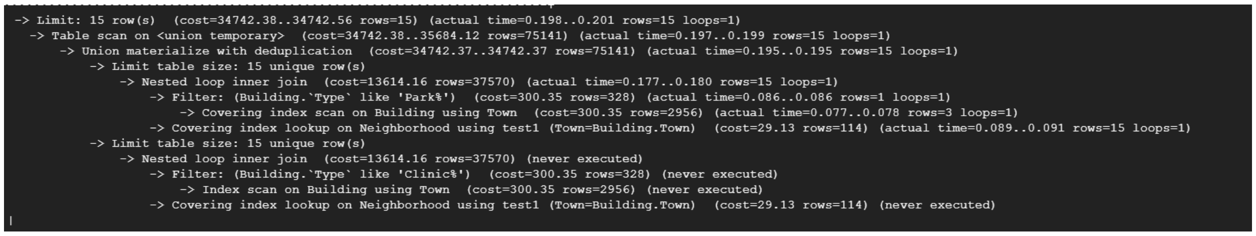
Added index on Town



```
| -> 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;

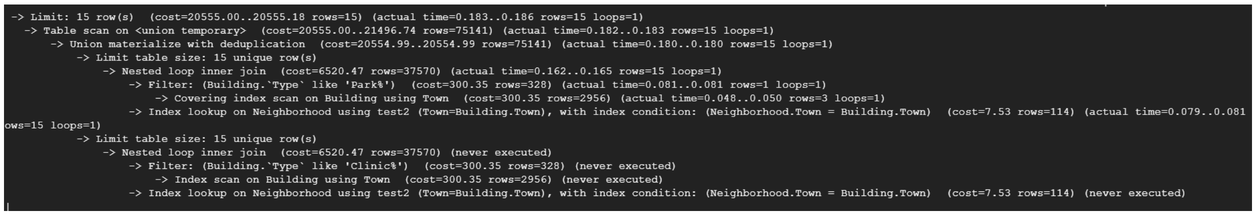
Added index to Neighborhood on Latitude and Longitude



```
 -> 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.

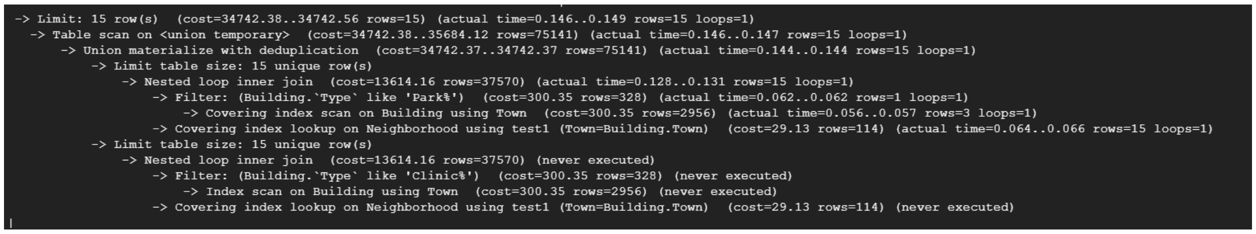
Added index to Neighborhood on Town(5)



```
 -> 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)
|
```

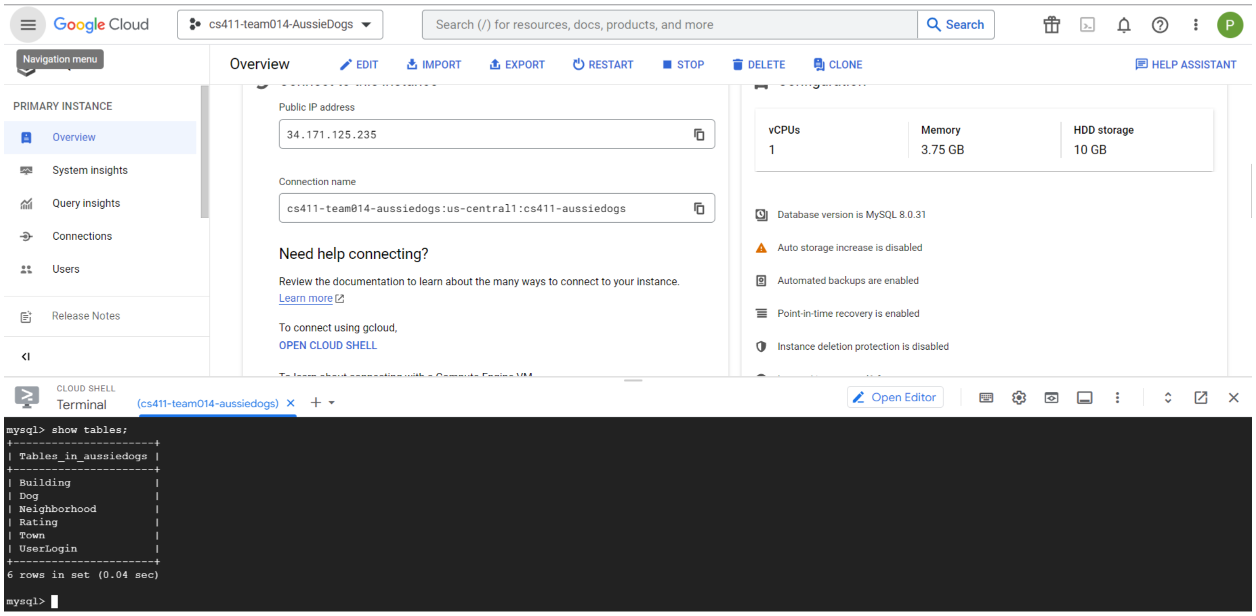
This sped up the query the most. We are down almost 25% from our initial query. This likely categorizes the data the best because we are ultimately selecting for the names of the neighborhoods for each subquery. As a result, this is the index we will be using.

Find neighborhood with highest number of specific breed (German Shepard)

Select Neighborhood.name
FROM Neighborhood JOIN Dog USING(Town)
Where SUM(Dog.Breed == “German Shepard”) >= ALL(Dog.Breed == “German Shepard”)

Find neighborhood with a population of 0 of a specific breed (German Shepard)
Select Neighborhood.name
FROM Neighborhood JOIN Dog USING(Town)
Where SUM(Dog.Breed == “German Shepard”) == 0;

SCREENSHOTS:



Public IP address
34.171.125.235

Connection name
cs411-team014-aussiedogs:us-central1:cs411-aussiedogs

Need help connecting?

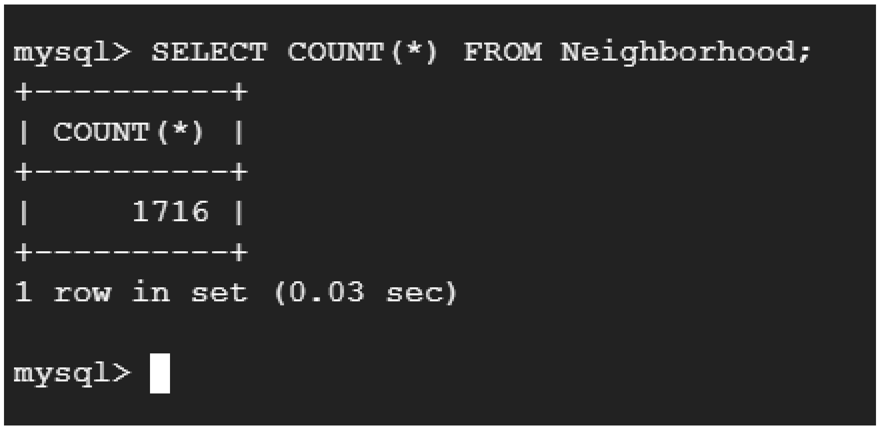
Review the documentation to learn about the many ways to connect to your instance.
Learn more

To connect using gcloud,
OPEN CLOUD SHELL

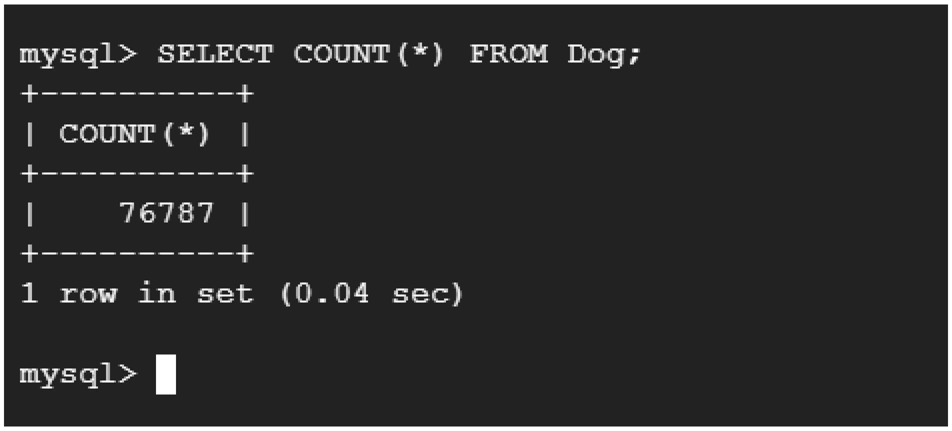
vCPUs	Memory	HDD storage
1	3.75 GB	10 GB

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

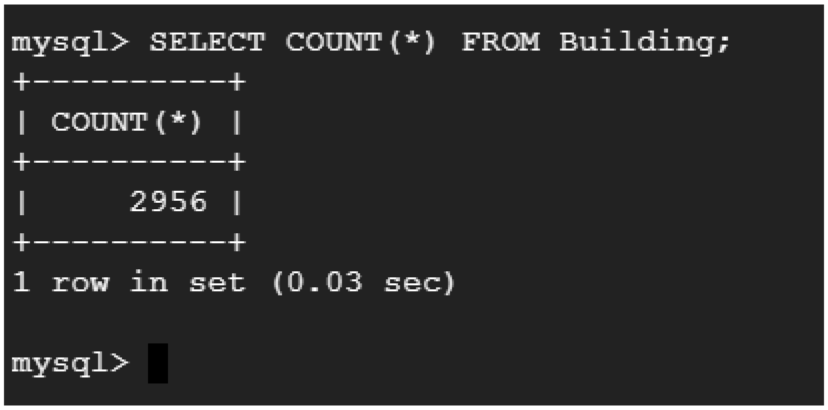
```
mysql> show tables;
+----------------------+
| Tables_in_aussiedogs |
+----------------------+
| Building             |
| Dog                  |
| Neighborhood         |
| Rating               |
| Town                 |
| UserLogin            |
+----------------------+
6 rows in set (0.04 sec)

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> 
```