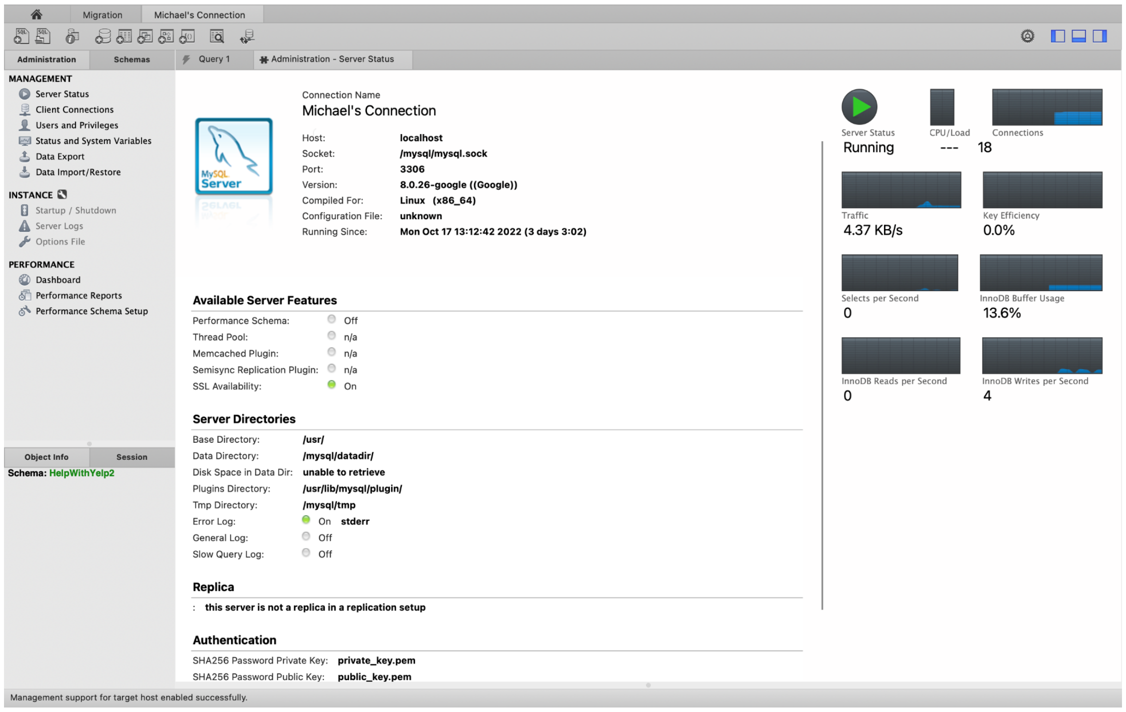
Database Implementation

Implement the database tables locally or on GCP. Provide a screenshot of the connection (i.e. showing your terminal information).



Schema: HelpWithYelp2

Connection Name
Michael's Connection

```
Host:                 localhost
Socket:               /mysql/mysql.sock
Port:                 3306
Version:              8.0.26-google ((Google))
Compiled For:         Linux   (x86_64)
Configuration File:   unknown
Running Since:        Mon Oct 17 13:12:42 2022 (3 days 3:02)
```

```
Server Status   CPU/Load   Connections
Running           ---      18

Traffic                    Key Efficiency
4.37 KB/s                  0.0%

Selects per Second         InnoDB Buffer Usage
0                          13.6%

InnoDB Reads per Second    InnoDB Writes per Second
0                          4
```

Available Server Features

```
Performance Schema:            Off
Thread Pool:                   n/a
Memcached Plugin:              n/a
Semisync Replication Plugin:   n/a
SSL Availability:              On
```

Server Directories

```
Base Directory:          /usr/
Data Directory:          /mysql/datadir/
Disk Space in Data Dir:  unable to retrieve
Plugins Directory:       /usr/lib/mysql/plugin/
Tmp Directory:           /mysql/tmp
Error Log:               On   stderr
General Log:             Off
Slow Query Log:          Off
```

Replica

: this server is not a replica in a replication setup

Authentication

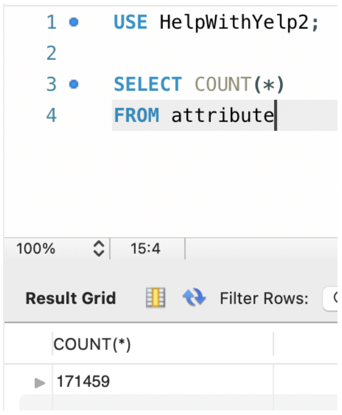
```
SHA256 Password Private Key:   private_key.pem
SHA256 Password Public Key:    public_key.pem
```

Management support for target host enabled successfully.

Provide the DDL commands for your tables.

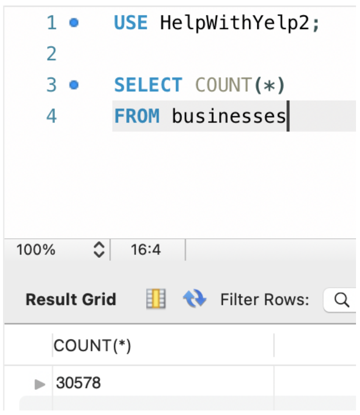
```
CREATE TABLE weather_by_zip ( weather_id int Primary Key, postal int,
month varchar(9), avgTemp float(10), highTemp float(10), lowTemp
float(10), total_rain_inches float(10) );

CREATE TABLE businesses ( buisness_id varchar(25) PRIMARY KEY, name
varchar(30), address varchar(65), city varchar(20), state
varchar(15), postal int, longitude float(10), latitude float(10),
stars float(10), review_conut int, attributes varchar(1000),
categories varchar(1000), monday varchar(9), tuesday varchar(9),
wednesday varchar(9), thursday varchar(9), friday varchar(9),
saturday varchar(9), sunday varchar(9) --FOREIGN KEY
(longitude,latitude) REFERENCES locations (longitude,latitude));

CREATE TABLE reviews ( review_id varchar(25) PRIMARY KEY, user_id
varchar(25), business_id varchar(100), stars int, useful int, funny
int, cool int, text varchar(6000), date varchar(24) --FOREIGN KEY
(user_id) REFERENCES users(user_id), FOREIGN KEY (business_id)
REFERENCES businesses(businesses));

CREATE TABLE users ( user_id varchar(25) PRIMARY KEY, name
varchar(30), review_count int, yelping_since varchar(24), useful int,
funny int, cool int );

CREATE TABLE tips ( tip_id int PRIMARY KEY, user_id varchar(25),
business_id varchar(30), text varchar(3000), date varchar(24),
compliment_count int FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE attributes (attribute varchar(15), business_id
varchar(25), PRIMARY KEY (attribute, business_id))

CREATE TABLE categories (category varchar(15), business_id
varchar(25), PRIMARY KEY (category, business_id))
```

Insert at least 1000 rows in your tables. Do a count query to show this.



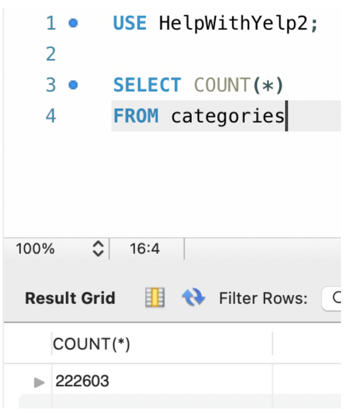
```
USE HelpWithYelp2;

SELECT COUNT(*)
FROM attribute
```

```
COUNT(*)
171459
```



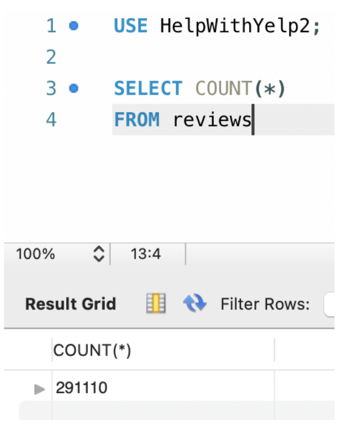
```
USE HelpWithYelp2;

SELECT COUNT(*)
FROM businesses
```

```
COUNT(*)
30578
```



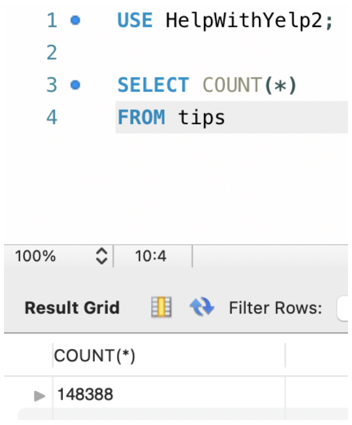
```
USE HelpWithYelp2;

SELECT COUNT(*)
FROM categories
```

```
COUNT(*)
222603
```



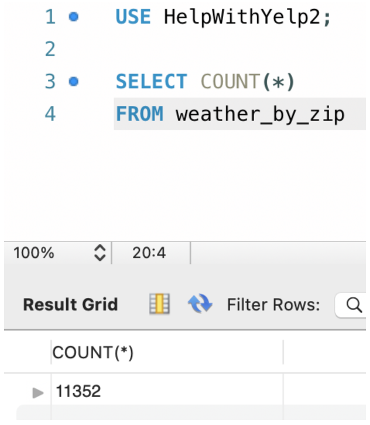
```
USE HelpWithYelp2;

SELECT COUNT(*)
FROM reviews
```

```
COUNT(*)
291110
```



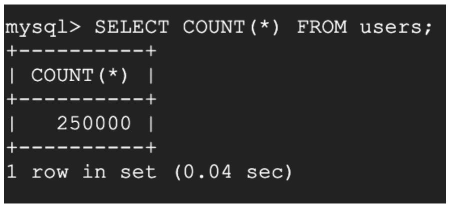
```
USE HelpWithYelp2;

SELECT COUNT(*)
FROM tips
```

```
COUNT(*)
148388
```



```
USE HelpWithYelp2;

SELECT COUNT(*)
FROM weather_by_zip
```

```
COUNT(*)
11352
```



```
mysql> SELECT COUNT(*) FROM users;
+----------+
| COUNT(*) |
+----------+
|   250000 |
+----------+
1 row in set (0.04 sec)
```

Advanced Queries

Advanced Query 1

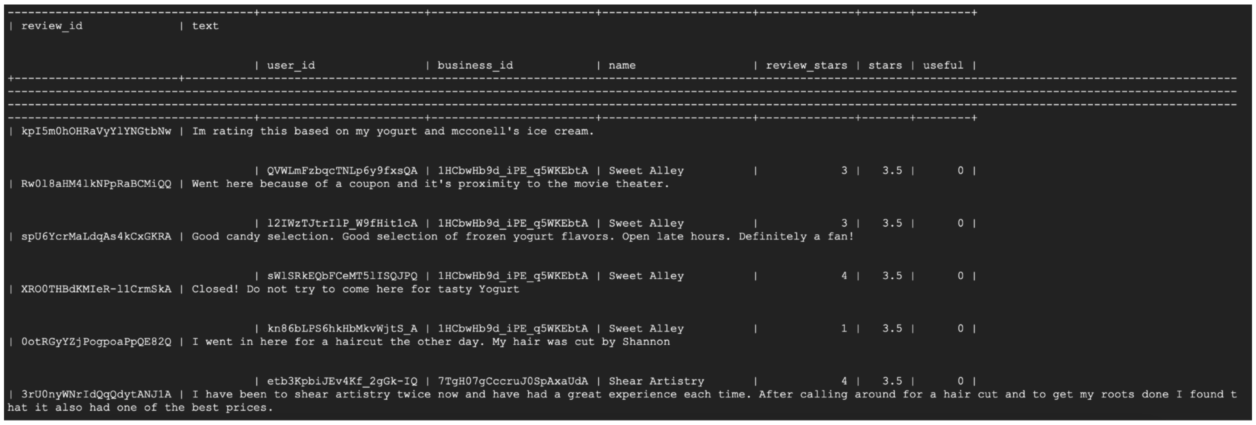
Involves the SQL concepts “Join of multiple relations” and “Subqueries”.

```
SELECT review_id, text, user_id, business_id, name, review_stars,stars,
useful FROM reviews INNER JOIN (SELECT * FROM businesses where postal =
93117 and stars >= 3.5 ) as b using(business_id)
```

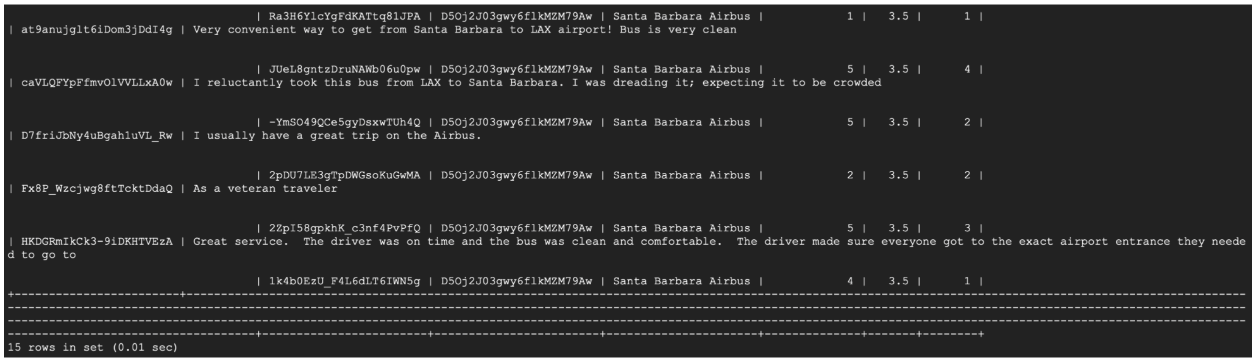
Purpose: To get the reviews of businesses that perform well (defined as businesses which have an average Yelp rating of at least 3.5 stars) in the desired selected area (defined by postal code). This allows a business owner using our website to better understand what customers value in existing businesses from anecdotes “from the horse’s mouth”; they can then tailor their business plan accordingly.

Execute your advanced SQL query. Screenshot the top 15 rows of the query result:

The screenshot for this query is jumbled because the text column being of wide variable length since review lengths vary a lot.

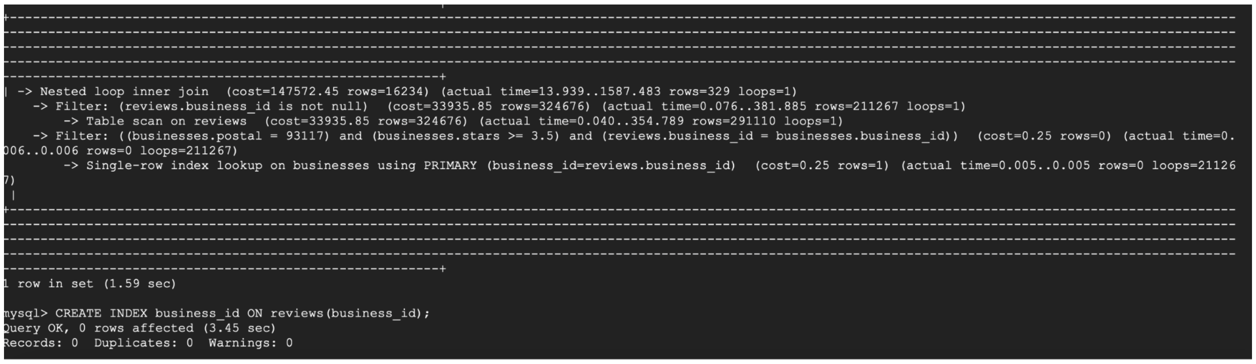


```
------------------------------------+-----------------------+-----------------------+---------------------+-------------+-------+--------+
| review_id              | text


                                    | user_id                | business_id            | name                 | review_stars | stars | useful |
+------------------------+-----------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------+-----------------------+-----------------------+---------------------+-------------+-------+--------+
| kpI5m0hOHRaVyYlYNGtbNw | Im rating this based on my yogurt and mcconell's ice cream.


                                    | QVWLmFzbqcTNLp6y9fxsQA | 1HCbwHb9d_iPE_q5WKEbtA | Sweet Alley          |            3 |   3.5 |      0 |
| Rw0l8aHM4lkNPpRaBCMiQQ | Went here because of a coupon and it's proximity to the movie theater.


                                    | l2IWzTJtrIlP_W9fHit1cA | 1HCbwHb9d_iPE_q5WKEbtA | Sweet Alley          |            3 |   3.5 |      0 |
| spU6YcrMaLdqAs4kCxGKRA | Good candy selection. Good selection of frozen yogurt flavors. Open late hours. Definitely a fan!


                                    | sWlSRkEQbFCeMT5lISQJPQ | 1HCbwHb9d_iPE_q5WKEbtA | Sweet Alley          |            4 |   3.5 |      0 |
| XRO0THBdKMIeR-l1CrmSkA | Closed! Do not try to come here for tasty Yogurt


                                    | kn86bLPS6hkHbMkvWjtS_A | 1HCbwHb9d_iPE_q5WKEbtA | Sweet Alley          |            1 |   3.5 |      0 |
| 0otRGyYZjPogpoaPpQE82Q | I went in here for a haircut the other day. My hair was cut by Shannon


                                    | etb3KpbiJEv4Kf_2gGk-IQ | 7TgH07gCccruJ0SpAxaUdA | Shear Artistry       |            4 |   3.5 |      0 |
| 3rU0nyWNrIdQqQdytANJ1A | I have been to shear artistry twice now and have had a great experience each time. After calling around for a hair cut and to get my roots done I found t
hat it also had one of the best prices.
```



```
                                    | Ra3H6YlcYgFdKATtq81JPA | D5Oj2J03gwy6flkMZM79Aw | Santa Barbara Airbus |            1 |   3.5 |      1 |
| at9anujglt6iDom3jDdI4g | Very convenient way to get from Santa Barbara to LAX airport! Bus is very clean


                                    | JUeL8gntzDruNAWb06u0pw | D5Oj2J03gwy6flkMZM79Aw | Santa Barbara Airbus |            5 |   3.5 |      4 |
| caVLQFYpFfmvOlVVLLxA0w | I reluctantly took this bus from LAX to Santa Barbara. I was dreading it; expecting it to be crowded


                                    | -YmSO49QCe5gyDsxwTUh4Q | D5Oj2J03gwy6flkMZM79Aw | Santa Barbara Airbus |            5 |   3.5 |      2 |
| D7friJbNy4uBgah1uVL_Rw | I usually have a great trip on the Airbus.


                                    | 2pDU7LE3gTpDWGsoKuGwMA | D5Oj2J03gwy6flkMZM79Aw | Santa Barbara Airbus |            2 |   3.5 |      2 |
| Fx8P_Wzcjwg8ftTcktDdaQ | As a veteran traveler


                                    | 2ZpI58gpkhK_c3nf4PvPfQ | D5Oj2J03gwy6flkMZM79Aw | Santa Barbara Airbus |            5 |   3.5 |      3 |
| HKDGRmIkCk3-9iDKHTVEzA | Great service.  The driver was on time and the bus was clean and comfortable.  The driver made sure everyone got to the exact airport entrance they neede
d to go to

                                    | 1k4b0EzU_F4L6dLT6IWN5g | D5Oj2J03gwy6flkMZM79Aw | Santa Barbara Airbus |            4 |   3.5 |      1 |
+------------------------+------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------+-----------------------+-----------------------+---------------------+-------------+-------+--------+
15 rows in set (0.01 sec)
```

Pre-index analysis:

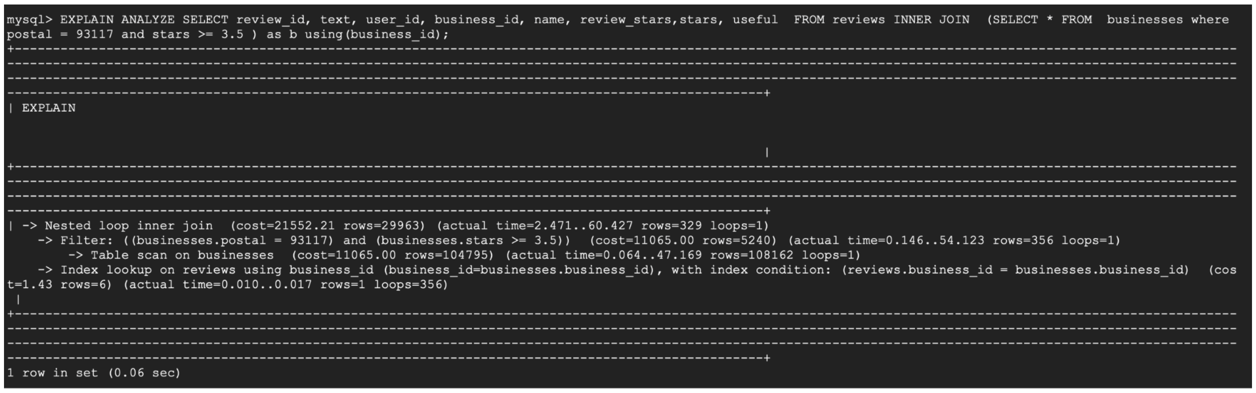


```
+----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| -> Nested loop inner join  (cost=147572.45 rows=16234) (actual time=13.939..1587.483 rows=329 loops=1)
    -> Filter: (reviews.business_id is not null)  (cost=33935.85 rows=324676) (actual time=0.076..381.885 rows=211267 loops=1)
        -> Table scan on reviews  (cost=33935.85 rows=324676) (actual time=0.040..354.789 rows=291110 loops=1)
    -> Filter: ((businesses.postal = 93117) and (businesses.stars >= 3.5) and (reviews.business_id = businesses.business_id))  (cost=0.25 rows=0) (actual time=0.
006..0.006 rows=0 loops=211267)
        -> Single-row index lookup on businesses using PRIMARY (business_id=reviews.business_id)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=0 loops=21126
7)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
1 row in set (1.59 sec)

mysql> CREATE INDEX business_id ON reviews(business_id);
Query OK, 0 rows affected (3.45 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Post index analysis 1/3:

Index on business_id in reviews:

```
CREATE INDEX business_id_idx ON reviews(business_id)
```

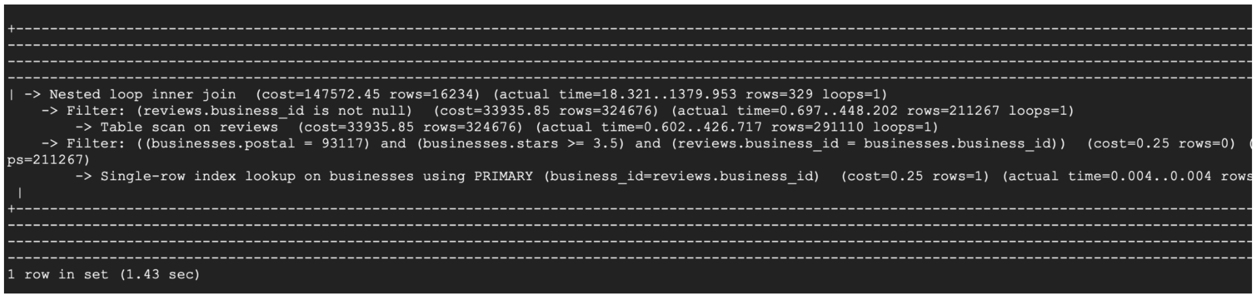


```
mysql> EXPLAIN ANALYZE SELECT review_id, text, user_id, business_id, name, review_stars,stars, useful  FROM reviews INNER JOIN  (SELECT * FROM  businesses where
postal = 93117 and stars >= 3.5 ) as b using(business_id);
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                   |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=21552.21 rows=29963) (actual time=2.471..60.427 rows=329 loops=1)
    -> Filter: ((businesses.postal = 93117) and (businesses.stars >= 3.5))  (cost=11065.00 rows=5240) (actual time=0.146..54.123 rows=356 loops=1)
        -> Table scan on businesses  (cost=11065.00 rows=104795) (actual time=0.064..47.169 rows=108162 loops=1)
    -> Index lookup on reviews using business_id (business_id=businesses.business_id), with index condition: (reviews.business_id = businesses.business_id)  (cos
t=1.43 rows=6) (actual time=0.010..0.017 rows=1 loops=356)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------+
1 row in set (0.06 sec)
```

JUSTIFICATION:
We explored a couple possible areas we can try to index and speed up the query time. For example in our query we join two tables on `business_id` (`INNER JOIN ... using(business_id)`). Since the first table is already sorted by `business_id` (it is a primary key), we can index the `reviews` table by `business_id` to give the inner join of `reviews` and `businesses` two sorted `business_id` columns to work with. We can see for the inner join the actual time goes from 13.939 to 2.417 after indexing which leads to a significant speed up.

Post index analysis 2/3:

Index on stars in businesses
`CREATE INDEX stars_idx ON business(stars)`



```
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=147572.45 rows=16234) (actual time=18.321..1379.953 rows=329 loops=1)
    -> Filter: (reviews.business_id is not null)  (cost=33935.85 rows=324676) (actual time=0.697..448.202 rows=211267 loops=1)
        -> Table scan on reviews  (cost=33935.85 rows=324676) (actual time=0.602..426.717 rows=291110 loops=1)
    -> Filter: ((businesses.postal = 93117) and (businesses.stars >= 3.5) and (reviews.business_id = businesses.business_id))  (cost=0.25 rows=0) (
ps=211267)
        -> Single-row index lookup on businesses using PRIMARY (business_id=reviews.business_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (1.43 sec)
```

JUSTIFICATION:
In the subqueries we see that we are sorting businesses by their star value and by their postal code. We explored indexing by the number of stars since searching through an indexed stars column could be done in $\log_2(n)$ time (binary search).

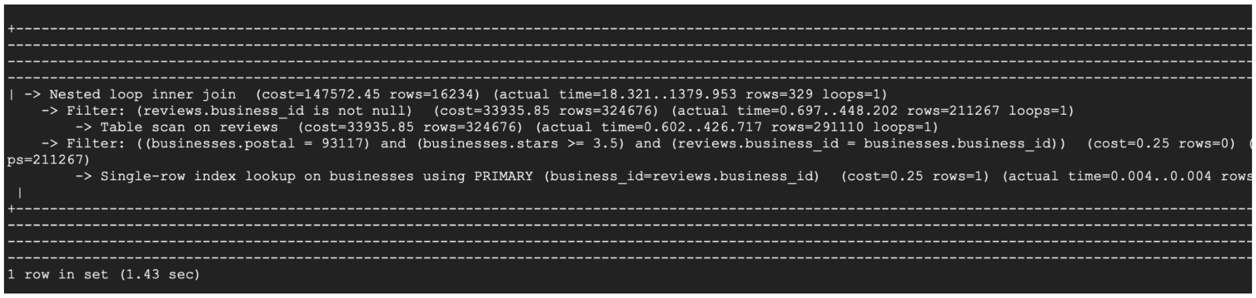
```
(SELECT * FROM businesses where postal = 93117 and stars >= 3.5 )
```

When we actually added the index, and ran `EXPLAIN ANALYZE` we see that there is only a minor speedup from actual time .006 to .004 in the subquery. We think this is because the relative cost of this subquery was very low and therefore this speedup is quite minor compared to the last one.

Post index analysis 3/3:

Index on postal in businesses

`CREATE INDEX postal_idx ON business(postal)`



```
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=147572.45 rows=16234) (actual time=18.321..1379.953 rows=329 loops=1)
    -> Filter: (reviews.business_id is not null)  (cost=33935.85 rows=324676) (actual time=0.697..448.202 rows=211267 loops=1)
        -> Table scan on reviews  (cost=33935.85 rows=324676) (actual time=0.602..426.717 rows=291110 loops=1)
    -> Filter: ((businesses.postal = 93117) and (businesses.stars >= 3.5) and (reviews.business_id = businesses.business_id))  (cost=0.25 rows=0) (
ps=211267)
        -> Single-row index lookup on businesses using PRIMARY (business_id=reviews.business_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows
 |
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
1 row in set (1.43 sec)
```

JUSTIFICATION:

We explored indexing by `postal` (postal code value) and we saw only minor speedup times. This speedup was also in the subquery and was about as effective as indexing by `stars` (which suggests that each had about the same impact on speedup).

Again because the subquery was a lower cost then the inner join this led to a small optimization. If we added more rows to businesses, we anticipate that we would see greater utilization of this index.

Selected Index

We selected the index design `CREATE INDEX business_id_idx ON reviews(business_id)` based on the above analysis. This is because of the three index designs we explored above, only this index design resulted in a considerable speedup.

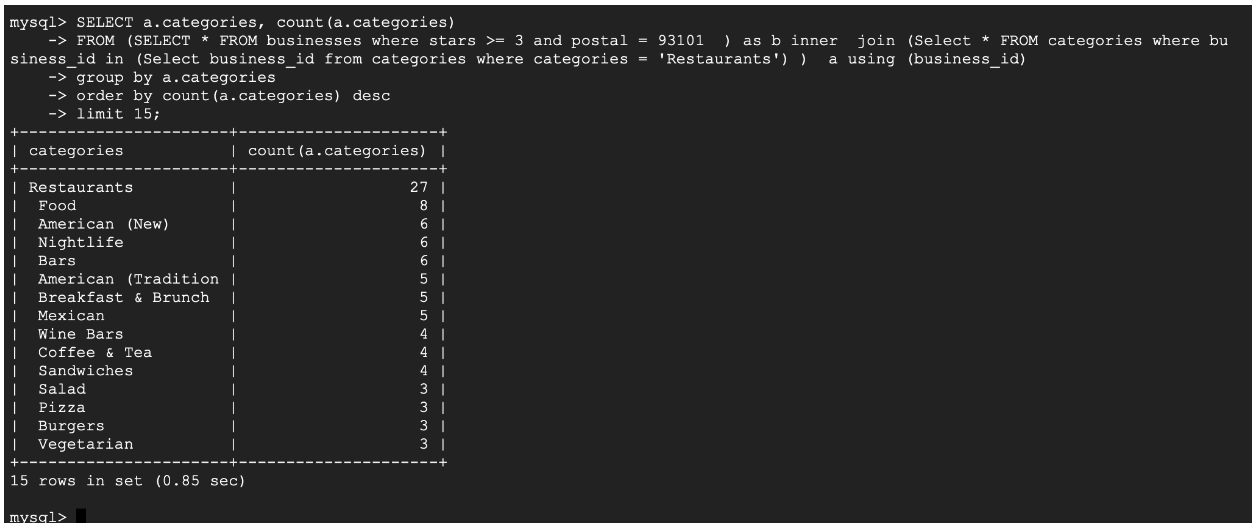
Advanced Query 2

Involves the SQL concepts “Join of multiple relations,” “Set operations,” “Aggregation via GROUP BY,” and “Subqueries”

```
SELECT a.categories, count(a.categories) FROM (SELECT * FROM businesses
where stars >= 3 and postal = 93101 ) as b inner join (Select * FROM
categories where business_id in (Select business_id from categories where
categories = 'Restaurants') ) a using (business_id) group by a.categories
order by count(a.categories) desc ;
```

Purpose: This query will allow us to show restaurant owners the most common types of restaurants which were successful (defined as a restaurant which has a rating of greater than three stars) in a given area (defined by postal code). From this, users would be able to get a sense of the tastes of customers in the area. For example, as a consequence of this query, a restaurant owner may see that Sushi restaurants were the most commonly successful business in the Chicago 60007 zip code. Knowing this, the owner may decide to open a sushi restaurant.

Execute your advanced SQL query. Screenshot the top 15 rows of the query result:



```
mysql> SELECT a.categories, count(a.categories)
    -> FROM (SELECT * FROM businesses where stars >= 3 and postal = 93101  ) as b inner  join (Select * FROM categories where bu
siness_id in (Select business_id from categories where categories = 'Restaurants') )  a using (business_id)
    -> group by a.categories
    -> order by count(a.categories) desc
    -> limit 15;
+----------------------+---------------------+
| categories           | count(a.categories) |
+----------------------+---------------------+
| Restaurants          |                  27 |
|  Food                |                   8 |
|  American (New)      |                   6 |
|  Nightlife           |                   6 |
|  Bars                |                   6 |
|  American (Tradition |                   5 |
|  Breakfast & Brunch  |                   5 |
|  Mexican             |                   5 |
|  Wine Bars           |                   4 |
|  Coffee & Tea        |                   4 |
|  Sandwiches          |                   4 |
|  Salad               |                   3 |
|  Pizza               |                   3 |
|  Burgers             |                   3 |
|  Vegetarian          |                   3 |
+----------------------+---------------------+
15 rows in set (0.85 sec)

mysql>
```

Pre-index analysis:

Pre-index Analysis

```
-> Sort: `count(categories.categories)` DESC  (actual time=879.049..879.052 rows=44 loops=1)
    -> Stream results  (cost=170183.45 rows=10550) (actual time=21.068..878.864 rows=44 loops=1)
        -> Group aggregate: count(categories.categories), count(categories.categories)  (cost=170183.45 rows=10550)
(actual time=21.062..878.635 rows=44 loops=1)
            -> Nested loop inner join  (cost=169128.45 rows=10550) (actual time=12.787..878.199 rows=131 loops=1)
                -> Nested loop inner join  (cost=95278.80 rows=210999) (actual time=0.122..786.831 rows=18517 loops=1)
```

```
                    -> Index scan on categories using PRIMARY  (cost=21429.15 rows=210999) (actual time=0.038..61.821
rows=222603 loops=1)
                    -> Single-row index lookup on categories using PRIMARY (categories='Restaurants',
business_id=categories.business_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=222603)
                -> Filter: ((businesses.postal = 93101) and (businesses.stars >= 3))  (cost=0.25 rows=0) (actual
time=0.005..0.005 rows=0 loops=18517)
                    -> Single-row index lookup on businesses using PRIMARY (business_id=categories.business_id)
(cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=18517)
```

Execution time: 0:00:0.98996592

Post index analysis 1/3:

```
CREATE INDEX stars_postal_idx ON businesses(stars, postal)
```

```
-> Sort: `count(categories.categories)` DESC  (actual time=894.157..894.160 rows=44 loops=1)
    -> Stream results  (cost=170183.45 rows=10550) (actual time=22.298..893.947 rows=44 loops=1)
        -> Group aggregate: count(categories.categories), count(categories.categories)  (cost=170183.45 rows=10550)
(actual time=22.290..893.647 rows=44 loops=1)
            -> Nested loop inner join  (cost=169128.45 rows=10550) (actual time=13.177..893.121 rows=131 loops=1)
                -> Nested loop inner join  (cost=95278.80 rows=210999) (actual time=0.118..789.205 rows=18517 loops=1)
                    -> Index scan on categories using PRIMARY  (cost=21429.15 rows=210999) (actual time=0.044..63.066
rows=222603 loops=1)
                    -> Single-row index lookup on categories using PRIMARY (categories='Restaurants',
business_id=categories.business_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=222603)
                -> Filter: ((businesses.postal = 93101) and (businesses.stars >= 3))  (cost=0.25 rows=0) (actual
time=0.005..0.005 rows=0 loops=18517)
                    -> Single-row index lookup on businesses using PRIMARY (business_id=categories.business_id)
(cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=18517)
```

Execution time: 0:00:0.93399191

JUSTIFICATION:

We explored an indexing design with stars and postal code as our indices based on the predicates of one of the `WHERE` clauses in our query (they are both being used in the subquery we have in the `FROM` statement [`SELECT * FROM businesses where stars >= 3 and postal = 93101`]).

By indexing these two columns, we hoped that we could speed these subqueries. In the filter line, the time for execution of this subquery remained .005; there was little to no overall speedup. We realized that the cause for this was that our index was never utilized in the query (it did not, for example, allow us to avoid a full table scan in favor of an index scan). Both the pre-index and post-index query do `Filter: ((businesses.postal = 93101) and (businesses.stars >= 3))`. This choice of index did not considerably improve the execution time.

Post index analysis 2/3:

```
CREATE INDEX business_idx ON categories(business_id)
```

```
-> Sort: `count(categories.categories)` DESC  (actual time=81.696..81.699 rows=44 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.012 rows=44 loops=1)
        -> Aggregate using temporary table  (actual time=81.640..81.653 rows=44 loops=1)
            -> Nested loop inner join  (cost=5102.46 rows=2020) (actual time=3.528..80.820 rows=131 loops=1)
                -> Nested loop inner join  (cost=4381.91 rows=478) (actual time=1.151..28.971 rows=27 loops=1)
                    -> Index lookup on categories using PRIMARY (categories='Restaurants')  (cost=1035.91 rows=9560)
(actual time=0.059..2.400 rows=4949 loops=1)
                    -> Filter: ((businesses.postal = 93101) and (businesses.stars >= 3))  (cost=0.25 rows=0) (actual
time=0.005..0.005 rows=0 loops=4949)
                        -> Single-row index lookup on businesses using PRIMARY (business_id=categories.business_id)
(cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=4949)
                -> Index lookup on categories using business_idx (business_id=categories.business_id)  (cost=1.09
rows=4) (actual time=1.913..1.917 rows=5 loops=27)
```

Execution time: 0:00:0.05104899

JUSTIFICATION:

We explored this indexing design based on the predicates of one of the `WHERE` clauses in our query (a subquery in which we do `Select business_id from categories where categories = 'Restaurants'`). Our pre-indexing design did a single-row lookup on businesses. By creating an index on `categories(business_id)`, we can instead do an index lookup on categories using business_idx. This choice in index significantly improved the execution time (we see an improvement of about 0.939).

Post index analysis 3/3:

Indexing design #3:

`CREATE INDEX cat_idx ON categories(categories)`

```
-> Sort: `count(categories.categories)` DESC  (actual time=1242.251..1242.254 rows=44 loops=1)
    -> Stream results  (cost=170183.45 rows=10550) (actual time=33.937..1242.092 rows=44 loops=1)
        -> Group aggregate: count(categories.categories), count(categories.categories)  (cost=170183.45 rows=10550)
(actual time=33.930..1241.818 rows=44 loops=1)
            -> Nested loop inner join  (cost=169128.45 rows=10550) (actual time=19.570..1241.283 rows=131 loops=1)
                -> Nested loop inner join  (cost=95278.80 rows=210999) (actual time=0.747..1140.480 rows=18517
loops=1)
                    -> Index scan on categories using cat_idx  (cost=21429.15 rows=210999) (actual time=0.679..393.743
rows=222603 loops=1)
                    -> Single-row index lookup on categories using PRIMARY (categories='Restaurants',
business_id=categories.business_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=222603)
                -> Filter: ((businesses.postal = 93101) and (businesses.stars >= 3))  (cost=0.25 rows=0) (actual
time=0.005..0.005 rows=0 loops=18517)
                    -> Single-row index lookup on businesses using PRIMARY (business_id=categories.business_id)
(cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=18517)
```

Execution time: 0:00:1.36580896

JUSTIFICATION:

We also explored this indexing design based on the predicates of one of the `WHERE` clauses in our query (a subquery in which we do `Select business_id from categories where categories = 'Restaurants'`). Our pre-indexing design did an index scan on categories using `PRIMARY`, while this design instead does an index scan on categories using the `categories(categories)` index. This had no effect on the cost of this step (it remained about 21429.15). We later realized that the cause of this modest impact is that our primary key for this table is already `(category, business_id)`, indexing on `category` does little. Consequently, this choice of index marginally worsened the execution time (an increase of about 0.375).

Selected index design:

We selected the index design `CREATE INDEX business_idx ON categories(business_id)` based on the above analysis. This is because of the three index designs we explored above, only this index design resulted in any speedup.