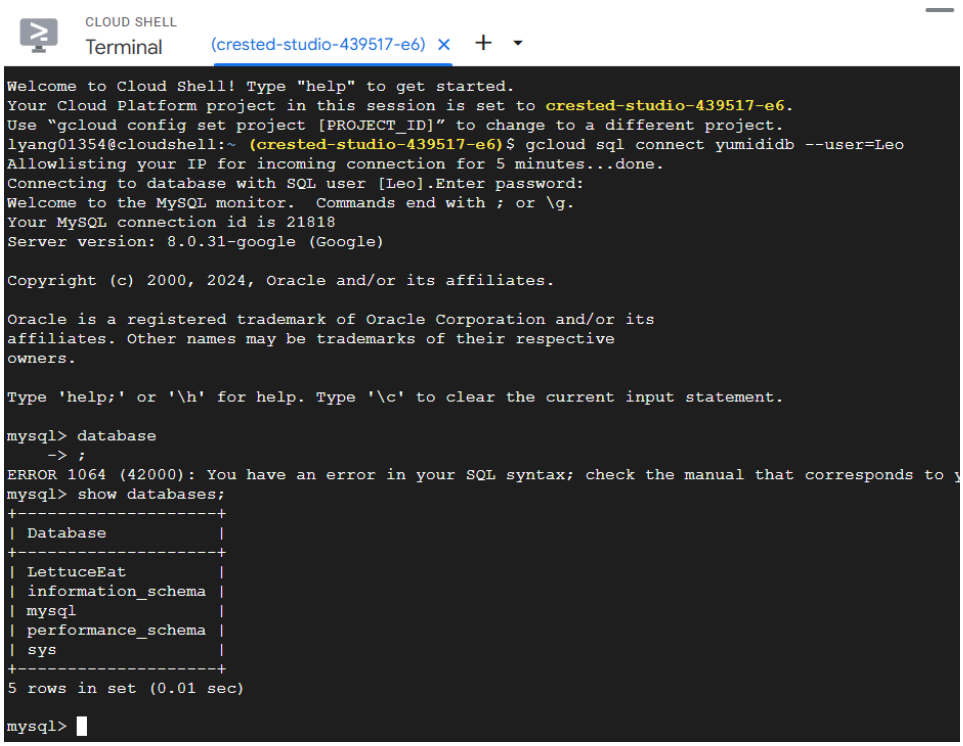
Stage 3: Database Implementation and Indexing Report

Part 1: Database Implementation

Database connection:



Data Insertion:

```
mysql> SELECT COUNT(*) FROM Ingredients;
+----------+
| COUNT(*) |
+----------+
|     7993 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Recipes;
+----------+
| COUNT(*) |
+----------+
|   231637 |
+----------+
1 row in set (0.08 sec)

mysql> SELECT COUNT(*) FROM Reviews;
+----------+
| COUNT(*) |
+----------+
|  1132367 |
+----------+
1 row in set (0.19 sec)
```

Advanced Query:

Query 1:

Calculate the average rating of the recipe based on the Reviews table.

```
SELECT
 recipe_id, name, AVG(rating)
FROM
 Reviews NATURAL JOIN Recipes
GROUP BY
 recipe_id
LIMIT 15;
```

recipe_id	name	AVG(rating)
38	low fat berry blue frozen dessert	4.2500
39	biryani	3.0000
40	best lemonade	4.3333
41	carina s tofu vegetable kebabs	4.5000
43	best blackbottom pie	1.0000
45	buttermilk pie with gingersnap crumb crust	2.6667
46	a jad cucumber pickle	5.0000
48	boston cream pie	1.0000
49	chicken breasts lombardi	4.3500
50	biscotti di prato	4.5000
52	cafe cappuccino	5.0000
53	jimmy g s carrot cake	3.0000
55	betty crocker s southwestern guacamole dip	4.7500
58	low fat burgundy beef vegetable stew	4.4286
59	lou s fabulous bruschetta	5.0000

Query 2:

Retrieve and rank the recipes based on their ratings in 2003

```
SELECT
 DISTINCT recipe_id, name, rating
FROM
 Recipes NATURAL JOIN Reviews
WHERE
 recipe_id in (
  SELECT DISTINCT
    recipe_id
  FROM
    Reviews NATURAL JOIN Recipes
  WHERE
    date >= '2003-1-1' and date <= '2003-12-31'
 )
ORDER BY
 rating DESC
LIMIT 15;
```

recipe_id	name	rating
14813	pork chop and new potato skillet	5
40893	white bean green chile pepper soup	5
20930	2 tomato pasta salad	5
41090	black and white bean salad	5
45732	cheesy fried eggplant aubergine	5
33096	world s easiest lemonade ice cream pie	5
45119	roasted sweet potato side or main	5
77001	caramelized nuts	5
58758	frozen coffee cooler	5
56494	sweet chilli prawn cakes	5
30968	summer tomato sauce	5
50181	savory tofu sauce	5
52968	extra cheesy macaroni cheese	5
77021	chicken raviolini soup	5
56916	ooey gooey butter cake	5

Query 3:

Get the recipe that uses beef as an ingredient and has calories less than 600 and number of reviews larger than 10, order the result based review count from high to low.

```
SELECT Recipes.recipe_id, name, description, calories, COUNT(review_id) AS review_count
FROM Recipes
JOIN Reviews ON Recipes.recipe_id = Reviews.recipe_id
JOIN recipe_ingredient ON Recipes.recipe_id = recipe_ingredient.recipe_id
JOIN Ingredients ON Ingredients.ingredient_id = recipe_ingredient.ingredient_id
WHERE calories < 600 and ingredient_name like '%beef%'
GROUP BY recipe_id, name, calories, description
HAVING COUNT(review_id) > 10
ORDER BY review_count DESC;
```

recipe_id	name	description	calories	review_cour
54257	yes virginia there is a great meatloaf	absolutely delicious meatloaf and sauce! those who claim they don't believe there can be ...	43	1305
69173	kittencal s italian melt in your mouth meatballs	cooking the meatballs in simmering pasta sauce will not only add so much extra flavor to ...	129	997
33919	creamy burrito casserole	satisfy your craving for something different tonight. it's good and fairly easy. this is stuff i ...	32	877
63689	my family s favorite sloppy joes pizza joes	i've tried many recipes, this is the best. i once 'dec-tupled' (10x) this recipe to sell sandwic...	26	720
27520	poverty meal	when i was a child, my family used to eat this at least once a week due to the fact that it is...	33	516
92095	authentic italian meatballs	meatball recipes are often challenged by chefs who claim, "mine are the best!" even thoug...	12	489
26217	bev s spaghetti sauce	this is a recipe i have developed over a number of years. this is the only spaghetti sauce m...	51	344
48760	szechuan noodles with spicy beef sauce	tired of using ground beef the same old way? try this spicy dish! feel free to double the sa...	39	315
87085	southwestern baked spaghetti	this is an easy, yet very tasty spaghetti casserole, which reminds me of the spaghetti they ...	30	302
37413	beef patties in onion gravy	good	29	272
155186	fantastic taco casserole	i originally found this taco casserole recipe in a taste of home magazine. since then, i hav...	32	271
75302	mrs geraldine s ground beef casserole	this recipe came from a local church fund raising cook book. mrs. geraldine is a good frie...	32	253
302120	traditional irish shepherd s pie	posting this per a request. i've said it once and i'll say it again there is nothing irish about c...	39	248
29884	easy enchiladas beef or chicken	these easy enchiladas are my husband's favorite recipe of mine. because of the many diff...	61	242

Query 4:

Retrieve the recipes that require more than 10 ingredients.

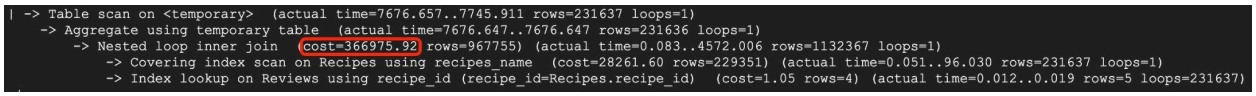
```
select recipe_id, name, count(*)
from Recipes NATURAL JOIN recipe_ingredient
GROUP BY recipe_id, name
HAVING COUNT(*) > 10
LIMIT 15;
```

recipe_id	name	count(*)
82	brazilian empadinhas	11
517	greek stuffed meatloaf	11
539	gingerbread yule log	11
2501	chocolate pumpkin spice cake	12
2974	cinnamon peach coffee cake	11
3667	apple cheese crisp	11
5366	pumpkin roll ii	11
5416	stroganoff laibchen small stroganoff loaves	11
6894	chocolate munchies	12
7074	spicy pineapple zucchini cake	11
8348	chocolate sauerkraut cake	12
8400	whole wheat chocolate cake	11
8519	pat s pumpkin bread	11
8846	lazy day oatmeal cake	11
11009	cream cheese apple muffins	11

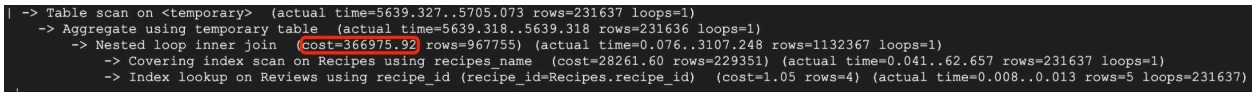
Part 2: Indexing

Query 1:

Original query:

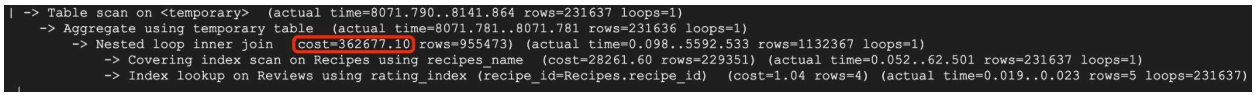


Create an index on rating:



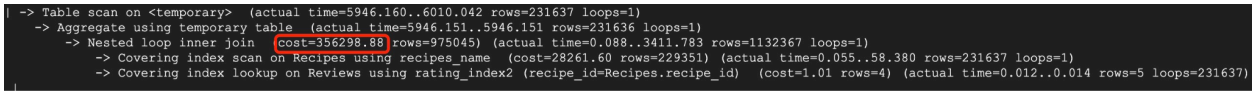
There is no difference after using the query. The reason is the attribute ‘rating’ doesn’t participate in the join process. Although the attribute ‘rating’ appears in the query, but it doesn’t participate in the join process.

Set index on (review_id, rating)



There is no obvious difference after setting the index. Although ‘review_id’ is the attribute for join process, it is already the primary key that determines the attribute ‘rating’.

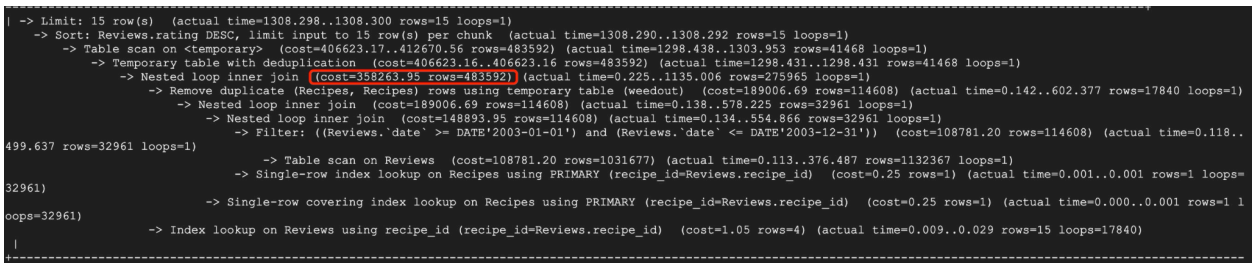
Set index on (recipe_id, rating)



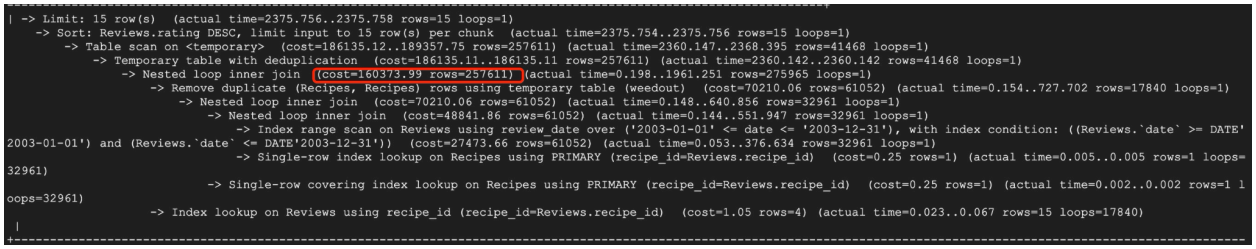
There is no obvious difference after implementing the index. Since ‘recipe_id’ is a foreign key that already got indexed, and rating doesn’t participate in the join process. Thus, there is no obvious decrease on the cost. Therefore, based on the experiment of this query, we will not add any index on the Reviews table.

Query 2

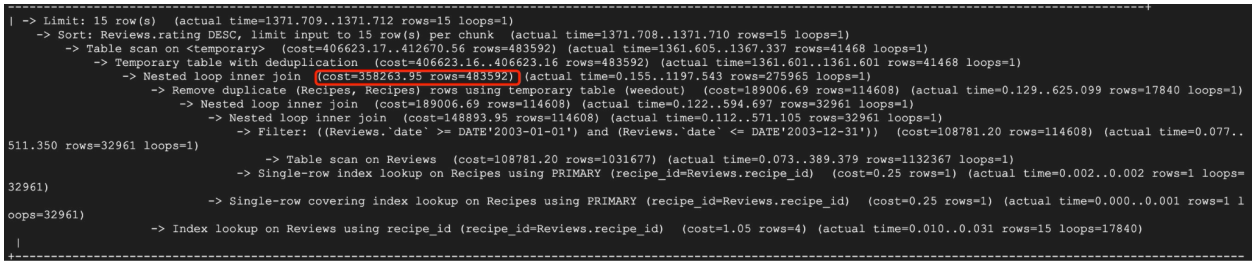
Only when creating an index on Reviews(date) will it affect the cost because in the query it has a comparison. The other two indexes don't participate in the join process.
Default:



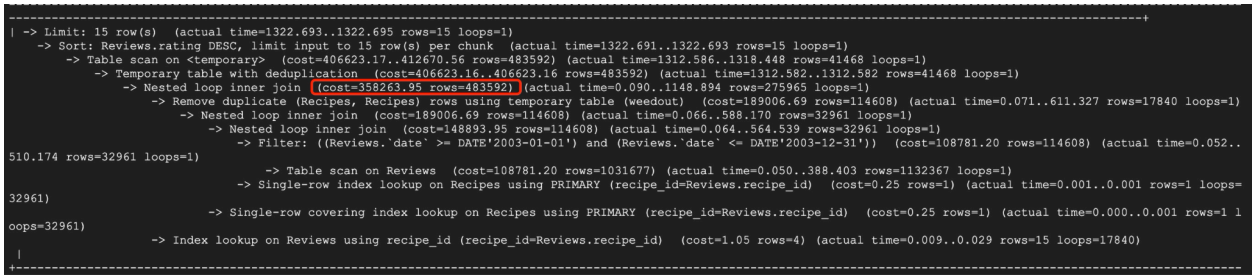
Create index on Reviews(date);



Create index on rating;



Create index on name;



Query 3

We created the indexes for calories, ingredient_name, and description of the recipe respectively and then ran the explain analyze command for the query after dropping the last index created (if it exists). Since the table JOIN on the primary key, we chose the attributes in the WHERE and GROUP_BY clause.
The index calories increases the cost from 55787 to 58922. It could be because the calories have a high cardinality, and it takes longer to retrieve the row with an additional index. The index ingredient_name reduces the cost from 55787 to 51760, for ingredient_name is a frequent attribute used for filtering. The index recipe_description does not change the cost at all. Since recipe_description is used in GROUP_BY after recipe_id, the optimizer may utilize recipe_id rather than recipe_description.
Another interesting observation is that the cost is permanently reduced by the ingredient_name even after dropping the index. It's possible that the optimizer learned a way to reduce the cost from adding the index.

Thus, the final index choice is adding index ingredient_name, because it reduces the cost effectively.

Default:

```
mysql> explain analyze SELECT Recipes.recipe_id, name, description, calories, COUNT(review_id) AS review_count
    -> FROM Recipes
    -> JOIN Reviews ON Recipes.recipe_id = Reviews.recipe_id
    -> JOIN recipe_ingredient ON Recipes.recipe_id = recipe_ingredient.recipe_id
    -> JOIN Ingredients on Ingredients.ingredient_id = recipe_ingredient.ingredient_id
    -> WHERE calories < 600 and ingredient_name like '%beef%'
    -> GROUP BY recipe_id, name, calories, description
    -> HAVING COUNT(review_id) > 10
    -> ORDER BY review_count DESC;
+----------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                            |
+----------------------------------------------------------------------------------------------------+
| -> Sort: review_count DESC  (actual time=256.635..256.948 rows=489 loops=1)
    -> Filter: (count(Reviews.review_id) > 10)  (actual time=254.624..256.352 rows=489 loops=1)
        -> Table scan on <temporary>  (actual time=254.613..256.048 rows=4730 loops=1)
            -> Aggregate using temporary table  (actual time=254.610..254.610 rows=4730 loops=1)
                -> Nested loop inner join  (cost=55787.43 rows=102960) (actual time=0.457..64.811 rows=32674 loops=1)
                    -> Nested loop inner join  (cost=39726.21 rows=22983) (actual time=0.423..30.626 rows=4730 loops=1)
                        -> Nested loop inner join  (cost=15591.94 rows=68955) (actual time=0.388..12.586 rows=4732 loops=1)
                            -> Filter: (Ingredients.ingredient_name like '%beef%')  (cost=9.55 rows=10) (actual time=0.079..0.118 rows=1 loops=1)
                                -> Table scan on Ingredients  (cost=9.55 rows=93) (actual time=0.058..0.079 rows=93 loops=1)
                            -> Index lookup on recipe_ingredient using ingredient_id (ingredient_id=Ingredients.ingredient_id)  (cost=905.34 rows=6674) (actual time=0.308..12.156 rows=4732 loops=1)
                        -> Filter: (Recipes.calories < 600)  (cost=0.25 rows=0.3) (actual time=0.004..0.004 rows=1 loops=4732)
                            -> Single-row index lookup on Recipes using PRIMARY (recipe_id=recipe_ingredient.recipe_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=4732)
                    -> Covering index lookup on Reviews using recipe_id (recipe_id=recipe_ingredient.recipe_id)  (cost=0.25 rows=4) (actual time=0.004..0.007 rows=7 loops=4730)
 |
```

create index recipe_calories on Recipes(calories);

```
mysql> create index recipe_calories on Recipes(calories);
Query OK, 0 rows affected (1.58 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT Recipes.recipe_id, name, description, calories, COUNT(review_id) AS review_count
    -> FROM Recipes
    -> JOIN Reviews ON Recipes.recipe_id = Reviews.recipe_id
    -> JOIN recipe_ingredient ON Recipes.recipe_id = recipe_ingredient.recipe_id
    -> JOIN Ingredients on Ingredients.ingredient_id = recipe_ingredient.ingredient_id
    -> WHERE calories < 600 and ingredient_name like '%beef%'
    -> GROUP BY recipe_id, name, calories, description
    -> HAVING COUNT(review_id) > 10
    -> ORDER BY review_count DESC;
+----------------------------------------------------------------------------------------------------
| EXPLAIN                                                                                            |
+----------------------------------------------------------------------------------------------------
| -> Sort: review_count DESC  (actual time=266.647..266.911 rows=489 loops=1)
    -> Filter: (count(Reviews.review_id) > 10)  (actual time=264.302..266.172 rows=489 loops=1)
        -> Table scan on <temporary>  (actual time=264.295..265.885 rows=4730 loops=1)
            -> Aggregate using temporary table  (actual time=264.291..264.291 rows=4730 loops=1)
                -> Nested loop inner join  (cost=58922.73 rows=134753) (actual time=0.478..74.500 rows=32674 loops=1)
                    -> Nested loop inner join  (cost=37438.49 rows=31935) (actual time=0.463..35.308 rows=4730 loops=1)
                        -> Nested loop inner join  (cost=15083.55 rows=63871) (actual time=0.445..14.196 rows=4732 loops=1)
                            -> Filter: (Ingredients.ingredient_name like '%beef%')  (cost=9.55 rows=10) (actual time=0.120..0.163 rows=1 loops=1)
                                -> Table scan on Ingredients  (cost=9.55 rows=93) (actual time=0.087..0.112 rows=93 loops=1)
                            -> Index lookup on recipe_ingredient using ingredient_id (ingredient_id=Ingredients.ingredient_id)  (cost=900.58 rows=6182) (actual time=0.324..13.727 rows=4732 loops=1)
                        -> Filter: (Recipes.calories < 600)  (cost=0.25 rows=0.5) (actual time=0.004..0.004 rows=1 loops=4732)
                            -> Single-row index lookup on Recipes using PRIMARY (recipe_id=recipe_ingredient.recipe_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=4732)
                    -> Covering index lookup on Reviews using recipe_id (recipe_id=recipe_ingredient.recipe_id)  (cost=0.25 rows=4) (actual time=0.005..0.008 rows=7 loops=4730)
 |
```

drop index recipe calories on Recipes;

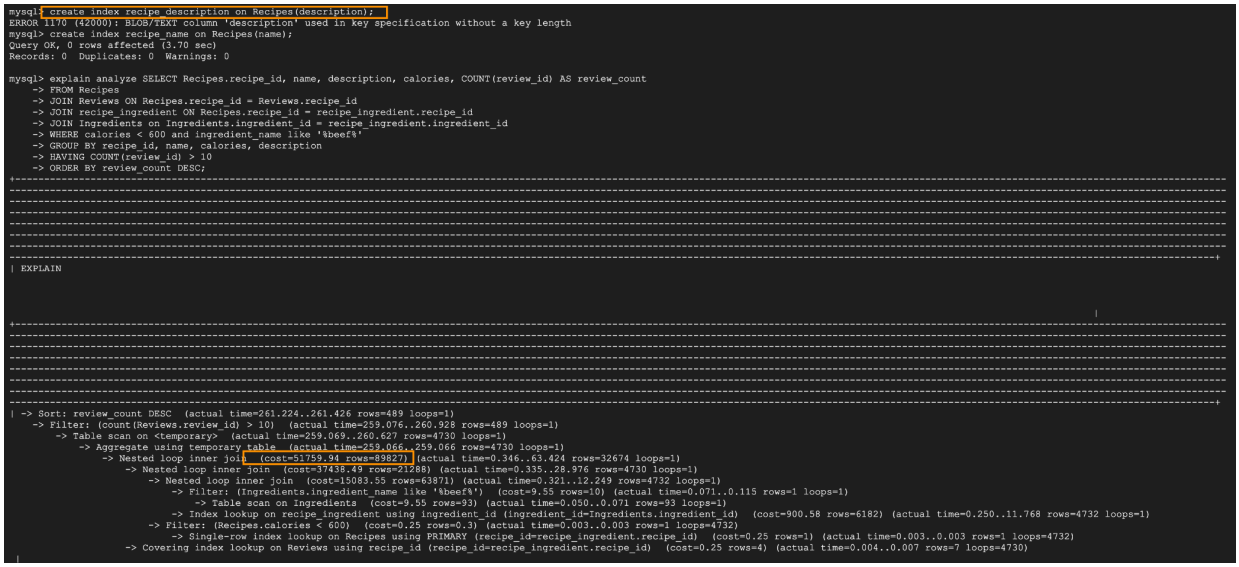
create index ingredient_name on Ingredients

```
mysql> drop index recipe_calories on Recipes;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> create index ingredient_name_index on Ingredients(ingredient_name);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT Recipes.recipe_id, name, description, calories, COUNT(review_id) AS review_count
    -> FROM Recipes
    -> JOIN Reviews ON Recipes.recipe_id = Reviews.recipe_id
    -> JOIN recipe_ingredient ON Recipes.recipe_id = recipe_ingredient.recipe_id
    -> JOIN Ingredients on Ingredients.ingredient_id = recipe_ingredient.ingredient_id
    -> WHERE calories < 600 and ingredient_name like '%beef%'
    -> GROUP BY recipe_id, name, calories, description
    -> HAVING COUNT(review_id) > 10
    -> ORDER BY review_count DESC;
+----------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                            |
+----------------------------------------------------------------------------------------------------+
| -> Sort: review_count DESC  (actual time=263.967..264.163 rows=489 loops=1)
    -> Filter: (count(Reviews.review_id) > 10)  (actual time=262.017..263.702 rows=489 loops=1)
        -> Table scan on <temporary>  (actual time=262.011..263.412 rows=4730 loops=1)
            -> Aggregate using temporary table  (actual time=262.008..262.008 rows=4730 loops=1)
                -> Nested loop inner join  (cost=51759.94 rows=89827) (actual time=0.451..71.515 rows=32674 loops=1)
                    -> Nested loop inner join  (cost=37438.49 rows=21288) (actual time=0.439..35.140 rows=4730 loops=1)
                        -> Nested loop inner join  (cost=15083.55 rows=63871) (actual time=0.427..14.821 rows=4732 loops=1)
                            -> Filter: (Ingredients.ingredient_name like '%beef%')  (cost=9.55 rows=10) (actual time=0.140..0.174 rows=1 loops=1)
                                -> Covering index scan on Ingredients using ingredient_name_index  (cost=9.55 rows=93) (actual time=0.041..0.060 rows=93 loops=1)
                            -> Index lookup on recipe_ingredient using ingredient_id (ingredient_id=Ingredients.ingredient_id)  (cost=900.58 rows=6182) (actual time=0.286..14.287 rows=4732 loops=1)
                        -> Filter: (Recipes.calories < 600)  (cost=0.25 rows=0.3) (actual time=0.004..0.004 rows=1 loops=4732)
                            -> Single-row index lookup on Recipes using PRIMARY (recipe_id=recipe_ingredient.recipe_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=4732)
                    -> Covering index lookup on Reviews using recipe_id (recipe_id=recipe_ingredient.recipe_id)  (cost=0.25 rows=4) (actual time=0.005..0.007 rows=7 loops=4730)
 |
```

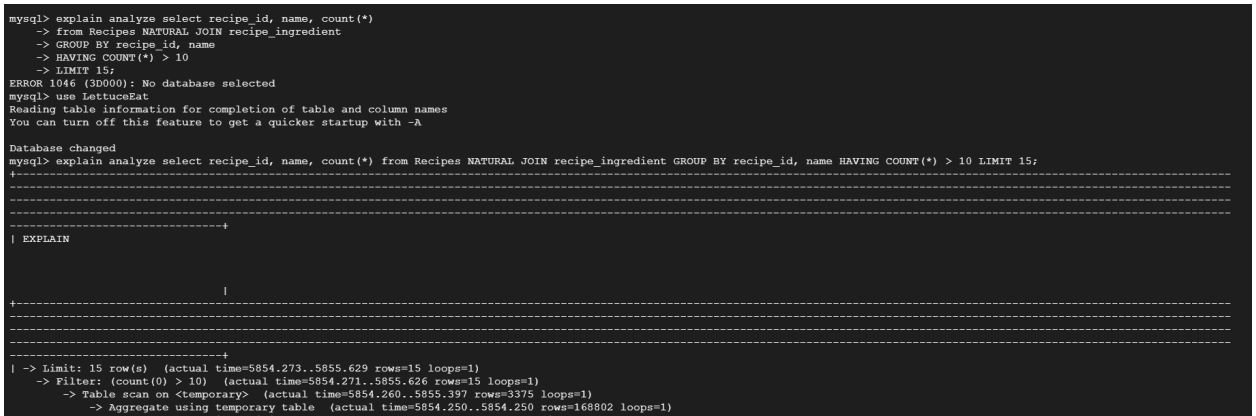
drop index ingredient_name on Ingredients
create index recipe_description on Recipes



Query 4

For the fourth query, we tried to add an index on name, minutes, and fat attributes of the Recipes table. However, creating index on each of these attributes showed no improvements. The EXPLAIN ANALYZE results are as follows.

This is the original cost of Query 4, cost=172502.33.



After creating an index on Recipes(name), the EXPLAIN ANALYZE result is as follows. The cost remains at 172502.33.

```
mysql> create index rec_name on Recipes(name);
Query OK, 0 rows affected (3.92 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze select recipe_id, name, count(*) from Recipes NATURAL JOIN recipe_ingredient GROUP BY recipe_id, name HAVING COUNT(*) > 10 LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| EXPLAIN



                                                        |
+----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=6170.273..6171.658 rows=15 loops=1)
    -> Filter: (count(0) > 10)  (actual time=6170.272..6171.656 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=6170.251..6171.427 rows=3375 loops=1)
            -> Aggregate using temporary table  (actual time=6170.242..6170.242 rows=168802 loops=1)
                -> Nested loop inner join  (cost=172502.33 rows=867473) (actual time=1.672..1792.951 rows=634859 loops=1)
                    -> Covering index scan on Recipes using rec_name  (cost=28261.60 rows=229351) (actual time=1.627..529.678 rows=231637 loops=1)
                    -> Covering index lookup on recipe_ingredient using recipe_id (recipe_id=Recipes.recipe_id)  (cost=0.25 rows=4) (actual time=0.004..0.005 rows=3 loops=231637)
 |
```

After removing the above index, we tried to create an index on Recipes(minutes), the EXPLAIN ANALYZE result is as follows. The cost remains at 172502.33.

```
mysql> explain analyze select recipe_id, name, count(*)
    -> from Recipes NATURAL JOIN recipe_ingredient
    -> GROUP BY recipe_id, name
    -> HAVING COUNT(*) > 10
    -> LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------+
| EXPLAIN



                                |
+----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------+
| -> Limit: 15 row(s)  (actual time=4524.244..4525.580 rows=15 loops=1)
    -> Filter: (count(0) > 10)  (actual time=4524.243..4525.577 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=4524.232..4525.370 rows=3375 loops=1)
            -> Aggregate using temporary table  (actual time=4524.221..4524.221 rows=168802 loops=1)
                -> Nested loop inner join  (cost=172502.33 rows=867473) (actual time=0.173..942.756 rows=634859 loops=1)
                    -> Table scan on Recipes  (cost=28261.60 rows=229351) (actual time=0.153..149.187 rows=231637 loops=1)
                    -> Covering index lookup on recipe_ingredient using recipe_id (recipe_id=Recipes.recipe_id)  (cost=0.25 rows=4) (actual time=0.002..0.003 rows=3 loops=231637)
 |
```

Similarly, removing the above index, we tried to create an index on Recipes(fat), the EXPLAIN ANALYZE result is as follows. The cost remains at 172502.33.

```
mysql> create index rec_fat on Recipes(fat);
Query OK, 0 rows affected (1.54 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze select recipe_id, name, count(*) from Recipes NATURAL JOIN recipe_ingredient GROUP BY recipe_id, name HAVING COUNT(*) > 10 LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------+
| EXPLAIN



                                |
+----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------
-------------------------------+
| -> Limit: 15 row(s)  (actual time=4535.653..4536.961 rows=15 loops=1)
    -> Filter: (count(0) > 10)  (actual time=4535.651..4536.959 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=4535.641..4536.748 rows=3375 loops=1)
            -> Aggregate using temporary table  (actual time=4535.632..4535.632 rows=168802 loops=1)
                -> Nested loop inner join  (cost=172502.33 rows=867473) (actual time=0.091..946.798 rows=634859 loops=1)
                    -> Table scan on Recipes  (cost=28261.60 rows=229351) (actual time=0.076..141.267 rows=231637 loops=1)
                    -> Covering index lookup on recipe_ingredient using recipe_id (recipe_id=Recipes.recipe_id)  (cost=0.25 rows=4) (actual time=0.002..0.003 rows=3 loops=231637)
 |
```

Since adding indexes on either name, minutes, and fat did not cause any change to the query time, we decided to not add any index based on the experiment with this query. In terms of the ineffectiveness of these indexing setups, we believe it has to do with the simplicity of this query. This query only joins two tables (Recipes, and recipe_ingredient), groups the rows with the same recipe_id, and outputs the number of rows in each group. The joining action is done via foreign key, which is indexed by its nature. And, we do not have other actions in this query that need to scan tables to access a certain row. Therefore, adding index on any attributes of Recipes and recipe_ingredient that are not primary key or foreign key will not have impacts on the query.