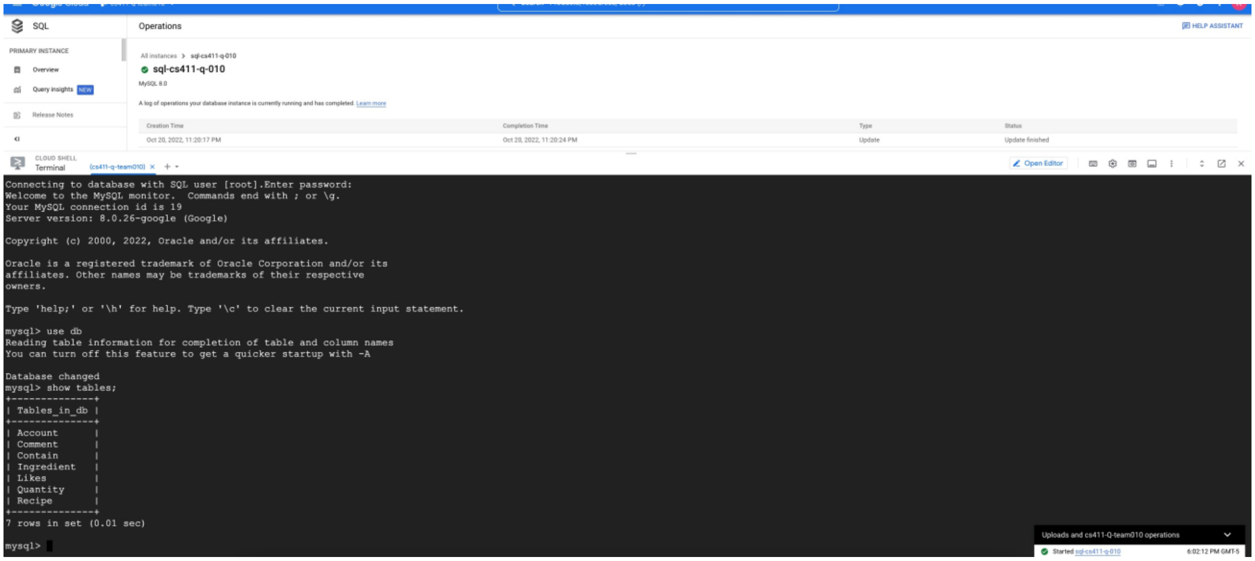
Google Cloud SQL Screenshot:



sql-cs411-q-010

MySQL 8.0

A log of operations your database instance is currently running and has completed. Learn more

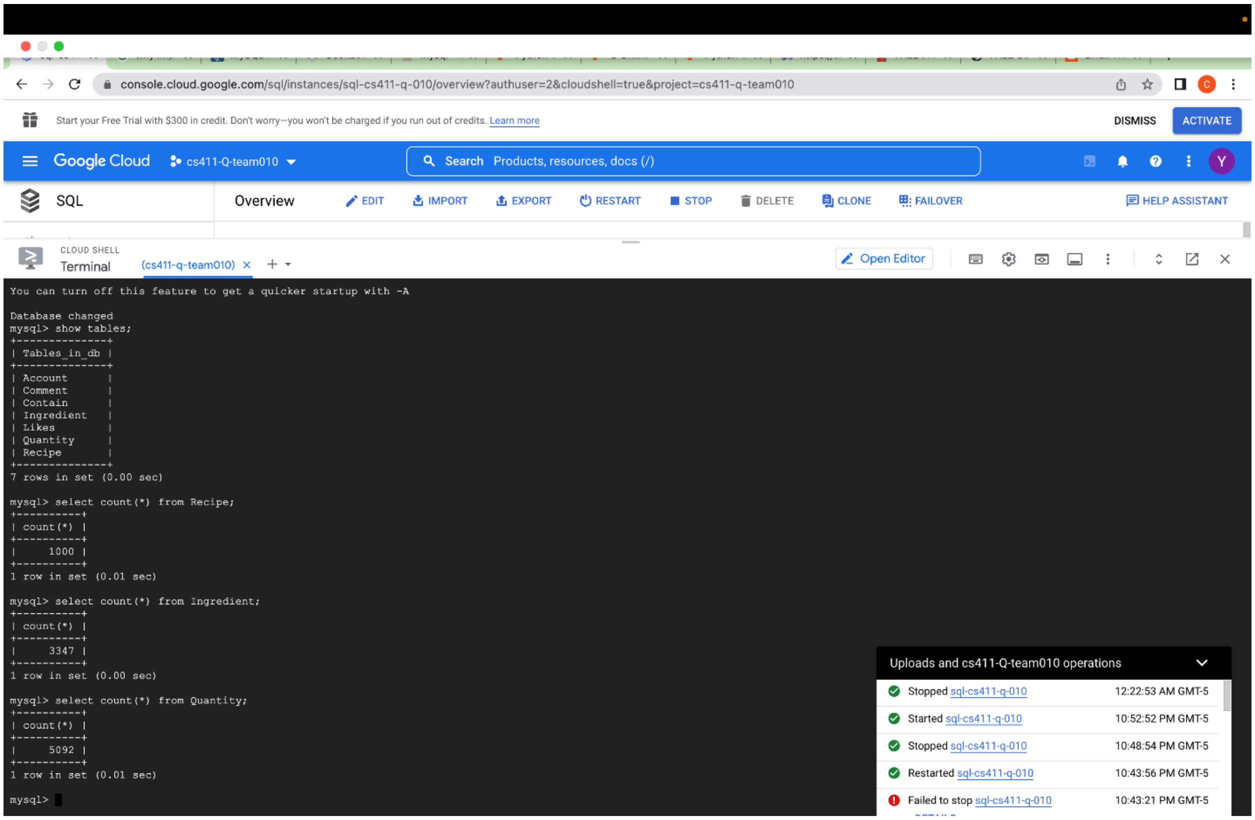
Creation Time	Completion Time	Type	Status
Oct 20, 2022, 11:20:17 PM	Oct 20, 2022, 11:20:24 PM	Update	Update finished

```
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use db
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+--------------+
| Tables_in_db |
+--------------+
| Account      |
| Comment      |
| Contain      |
| Ingredient   |
| Likes        |
| Quantity     |
| Recipe       |
+--------------+
7 rows in set (0.01 sec)

mysql> 
```

Uploads and cs411-Q-team010 operations

Started sql-cs411-q-010 6:02:12 PM GMT-5

count query :


console.cloud.google.com/sql/instances/sql-cs411-q-010/overview?authuser=2&cloudshell=true&project=cs411-q-team010
Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more
DISMISS
ACTIVATE
Google Cloud
cs411-Q-team010
Search Products, resources, docs (/)
SQL
Overview
EDIT
IMPORT
EXPORT
RESTART
STOP
DELETE
CLONE
FAILOVER
HELP ASSISTANT
CLOUD SHELL
Terminal
(cs411-q-team010)
Open Editor

```
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+--------------+
| Tables_in_db |
+--------------+
| Account      |
| Comment      |
| Contain      |
| Ingredient   |
| Likes        |
| Quantity     |
| Recipe       |
+--------------+
7 rows in set (0.00 sec)

mysql> select count(*) from Recipe;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from Ingredient;
+----------+
| count(*) |
+----------+
|     3347 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from Quantity;
+----------+
| count(*) |
+----------+
|     5092 |
+----------+
1 row in set (0.01 sec)

mysql> 
```

Uploads and cs411-Q-team010 operations

✓ Stopped sql-cs411-q-010	12:22:53 AM GMT-5
✓ Started sql-cs411-q-010	10:52:52 PM GMT-5
✓ Stopped sql-cs411-q-010	10:48:54 PM GMT-5
✓ Restarted sql-cs411-q-010	10:43:56 PM GMT-5
! Failed to stop sql-cs411-q-010	10:43:21 PM GMT-5

DDL:

create table Account(

```
account_id int primary key,
username char(255),
password char(255),
Email char(255),
address char(255),
IsManager bool);

create table Recipe(
recipe_id INT,
title char(255),
subtitle char(255),
servings INT,
yield_unit char(255),
prep_min INT,
cook_min INT,
source char(255),
intro char(255),
directions char(255),
account_id INT,
primary key(recipe_id),
foreign key(account_id) references Account(account_id)
        on delete cascade
        on update cascade);

create table Comment(
comment_id int,
recipe_id int,
account_id int,
content char(255),
primary key(comment_id),
foreign key(recipe_id) references Recipe(recipe_id)
        on delete cascade
        on update cascade,
foreign key(account_id) references Account(account_id)
        on delete cascade
        on update cascade);



create table Likes(
account_id INT,
recipe_id INT,
primary key(Account_id, recipe_id),
foreign key(recipe_id) references Recipe(recipe_id)
        on delete cascade
        on update cascade,
foreign key(account_id) references Account(account_id)
        on delete cascade
        on update cascade);

create table Ingredient(
ingredient_id INT primary key,
category char(255),
name char(255),
protein Decimal,
carbo Decimal,
alcohol Decimal,
total_fat Decimal,
sat_fat Decimal,
cholestrl Decimal,
sodium Decimal,
iron Decimal,
vitamin_c Decimal,
vitamin_a Decimal,
fiber Decimal,
pcnt_cal_carb Decimal,
pcnt_cal_fat Decimal,
pcnt_cal_prot Decimal,
calories Decimal);



create table Quantity(
quantity_id INT,
recipe_id INT,
ingredient_id INT,
max_qty Decimal,
min_qty Decimal,
unit char(255),
preparation char(255),
optional bool,
primary key(quantity_id, recipe_id, ingredient_id),
foreign key(recipe_id) references Recipe(recipe_id)
        on delete cascade
        on update cascade,
foreign key(ingredient_id) references Ingredient(ingredient_id)
        on delete cascade
        on update cascade);

create table Contain(
recipe_id INT,
ingredient_id INT,
primary key(recipe_id, ingredient_id),
foreign key(recipe_id) references Recipe(recipe_id)
        on delete cascade
        on update cascade,
foreign key(ingredient_id) references Ingredient(ingredient_id)
        on delete cascade
        on update cascade);
```

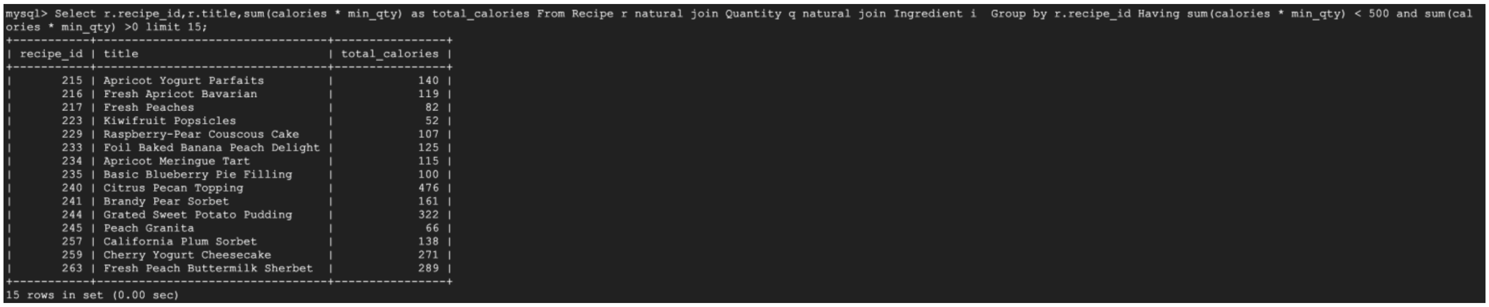
Query: compute recipe with calories less than 500.

```
Select r.recipe_id,r.title,sum(calories * min_qty) as total_calories
```

From Recipe r natural join Quantity q natural join Ingredient i
Group by r.recipe_id
Having sum(calories * min_qty) < 500 and sum(calories * min_qty) >0
Limit 15;

Used:

- Join of multiple relations
- Aggregation via GROUP BY



```
mysql> Select r.recipe_id,r.title,sum(calories * min_qty) as total_calories From Recipe r natural join Quantity q natural join Ingredient i  Group by r.recipe_id Having sum(calories * min_qty) < 500 and sum(cal
ories * min_qty) >0 limit 15;
+-----------+---------------------------------+----------------+
| recipe_id | title                           | total_calories |
+-----------+---------------------------------+----------------+
|       215 | Apricot Yogurt Parfaits         |            140 |
|       216 | Fresh Apricot Bavarian          |            119 |
|       217 | Fresh Peaches                   |             82 |
|       223 | Kiwifruit Popsicles             |             52 |
|       229 | Raspberry-Pear Couscous Cake    |            107 |
|       233 | Foil Baked Banana Peach Delight |            125 |
|       234 | Apricot Meringue Tart           |            115 |
|       235 | Basic Blueberry Pie Filling     |            100 |
|       240 | Citrus Pecan Topping            |            476 |
|       241 | Brandy Pear Sorbet              |            161 |
|       244 | Grated Sweet Potato Pudding     |            322 |
|       245 | Peach Granita                   |             66 |
|       257 | California Plum Sorbet          |            138 |
|       259 | Cherry Yogurt Cheesecake        |            271 |
|       263 | Fresh Peach Buttermilk Sherbet  |            289 |
+-----------+---------------------------------+----------------+
15 rows in set (0.00 sec)
```

Index analysis:

1. Without index:

```
| -> Filter: ((sum((i.calories * q.min_qty)) < 500) and (sum((i.calories * q.min_qty)) > 0))  (actual time=20.981..21.509 rows=357 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.290 rows=843 loops=1)
        -> Aggregate using temporary table  (actual time=20.973..21.309 rows=843 loops=1)
            -> Nested loop inner join  (cost=4008.25 rows=4940) (actual time=0.070..13.289 rows=5092 loops=1)
                -> Nested loop inner join  (cost=2279.25 rows=4940) (actual time=0.063..5.291 rows=5092 loops=1)
                    -> Table scan on q  (cost=550.25 rows=4940) (actual time=0.049..2.079 rows=5092 loops=1)
                    -> Single-row index lookup on r using PRIMARY (recipe_id=q.recipe_id)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=5092)
                -> Single-row index lookup on i using PRIMARY (ingredient_id=q.ingredient_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5092)
 |
```

2. With index on Quantity.ingredient_id

```
| -> Filter: ((sum((i.calories * q.min_qty)) < 500) and (sum((i.calories * q.min_qty)) > 0))  (actual time=19.314..19.667 rows=357 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.154 rows=843 loops=1)
        -> Aggregate using temporary table  (actual time=19.295..19.494 rows=843 loops=1)
            -> Nested loop inner join  (cost=4008.25 rows=4940) (actual time=0.076..12.291 rows=5092 loops=1)
                -> Nested loop inner join  (cost=2279.25 rows=4940) (actual time=0.069..5.054 rows=5092 loops=1)
                    -> Table scan on q  (cost=550.25 rows=4940) (actual time=0.055..2.009 rows=5092 loops=1)
                    -> Single-row index lookup on r using PRIMARY (recipe_id=q.recipe_id)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=5092)
                -> Single-row index lookup on i using PRIMARY (ingredient_id=q.ingredient_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5092)
```

We choose it because in the last line there is a single row index lookup using it.

There is no improvement. I think first of all it is a single-row index look up so adding index won’t provide a huge improvement. Secondly, the cost is already small enough on this step. So the improvement might be invisible.

3. With index on Quantity.recipe_id and Recipe.recipe_id

```
| -> Filter: ((sum((i.calories * q.min_qty)) < 500) and (sum((i.calories * q.min_qty)) > 0))  (actual time=20.378..20.738 rows=357
        loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.159 rows=843 loops=1)
        -> Aggregate using temporary table  (actual time=20.369..20.574 rows=843 loops=1)
            -> Nested loop inner join  (cost=4008.25 rows=4940) (actual time=0.053..13.134 rows=5092 loops=1)
                -> Nested loop inner join  (cost=2279.25 rows=4940) (actual time=0.047..5.281 rows=5092 loops=1)
                    -> Table scan on q  (cost=550.25 rows=4940) (actual time=0.037..2.051 rows=5092 loops=1)
                    -> Single-row index lookup on r using PRIMARY (recipe_id=q.recipe_id)  (cost=0.25 rows=1) (actual
                time=0.000..0.000 rows=1 loops=5092)
                -> Single-row index lookup on i using PRIMARY (ingredient_id=q.ingredient_id)  (cost=0.25 rows=1) (actual
                time=0.001..0.001 rows=1 loops=5092)
 |
```

We choose it because it is used when we join the table.
The actual time in nested loop inner join part all become shorter. Because that is where the join happens. With index on the attribute we are trying to join it can compute in a faster way.

4. With index on ingredient.calories

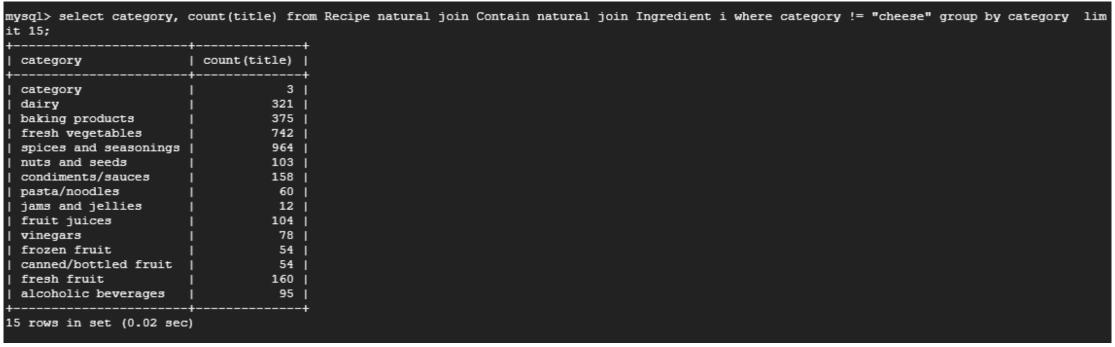
```
| -> Filter: ((sum((i.calories * q.min_qty)) < 500) and (sum((i.calories * q.min_qty)) > 0))  (actual time=19.247..19.624 rows=357
        loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.173 rows=843 loops=1)
        -> Aggregate using temporary table  (actual time=19.236..19.455 rows=843 loops=1)
            -> Nested loop inner join  (cost=4008.25 rows=4940) (actual time=0.058..12.298 rows=5092 loops=1)
                -> Nested loop inner join  (cost=2279.25 rows=4940) (actual time=0.051..5.068 rows=5092 loops=1)
                    -> Table scan on q  (cost=550.25 rows=4940) (actual time=0.038..1.987 rows=5092 loops=1)
                    -> Single-row index lookup on r using PRIMARY (recipe_id=q.recipe_id)  (cost=0.25 rows=1) (actual
                time=0.000..0.000 rows=1 loops=5092)
                -> Single-row index lookup on i using PRIMARY (ingredient_id=q.ingredient_id)  (cost=0.25 rows=1) (actual
                time=0.001..0.001 rows=1 loops=5092)
 |
```

We choose it because we are using sum(.calories * q.min_qty) in the filter.
The time when we perform filter became shorter. Because we need to use the ingredient.calories when we filter. With index on calories, the procedure become faster.

Query: Count the number of recipes that contain a category, and the category is not cheese.

```
select category, count(title)
from Recipe natural join Contain natural join Ingredient i
```

```
where category != "cheese"
group by category
limit 15;
```



```
mysql> select category, count(title) from Recipe natural join Contain natural join Ingredient i where category != "cheese" group by category  lim
it 15;
+-----------------------+--------------+
| category              | count(title) |
+-----------------------+--------------+
| category              |            3 |
| dairy                 |          321 |
| baking products       |          375 |
| fresh vegetables      |          742 |
| spices and seasonings |          964 |
| nuts and seeds        |          103 |
| condiments/sauces     |          158 |
| pasta/noodles         |           60 |
| jams and jellies      |           12 |
| fruit juices          |          104 |
| vinegars              |           78 |
| frozen fruit          |           54 |
| canned/bottled fruit  |           54 |
| fresh fruit           |          160 |
| alcoholic beverages   |           95 |
+-----------------------+--------------+
15 rows in set (0.02 sec)
```

Used:

- Join of multiple relations
- Aggregation via GROUP BY

Index analysis:

1. Without index:

```
-> Limit: 15 row(s)  (actual time=22.099..22.102 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.003 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=22.098..22.100 rows=15 loops=1)
            -> Nested loop inner join  (cost=7058.90 rows=13263) (actual time=0.100..17.932 rows=5048 loops=1)
                -> Nested loop inner join  (cost=2416.85 rows=13263) (actual time=0.092..10.631 rows=5048 loops=1)
                    -> Filter: (i.category <> 'chesse')  (cost=362.25 rows=2898) (actual time=0.069..2.855 rows=3347 loops=1)
                        -> Table scan on i  (cost=362.25 rows=3220) (actual time=0.064..1.745 rows=3347 loops=1)
                    -> Index lookup on Contain using ingredient_id (ingredient_id=i.ingredient_id)  (cost=0.25 rows=5) (actual
time=0.002..0.002 rows=2 loops=3347)
                -> Single-row index lookup on Recipe using PRIMARY (recipe_id=Contain.recipe_id)  (cost=0.25 rows=1) (actual
time=0.001..0.001 rows=1 loops=5048)
```

2. With index on Ingredient.category

```
-> Limit: 15 row(s)  (cost=4048.91 rows=15) (actual time=0.591..3.426 rows=15 loops=1)
    -> Group aggregate: count(Recipe.title)  (cost=4048.91 rows=62) (actual time=0.590..3.424 rows=15 loops=1)
        -> Nested loop inner join  (cost=4042.76 rows=62) (actual time=0.082..3.036 rows=861 loops=1)
            -> Nested loop inner join  (cost=732.29 rows=62) (actual time=0.072..1.843 rows=861 loops=1)
                -> Filter: (i.category <> 'chesse')  (cost=0.35 rows=13) (actual time=0.047..0.447 rows=619 loops=1)
                    -> Index scan on i using ecategory  (cost=0.35 rows=15) (actual time=0.043..0.263 rows=619 loops=1)
                -> Index lookup on Contain using ingredient_id (ingredient_id=i.ingredient_id)  (cost=0.25 rows=5) (actual
time=0.002..0.002 rows=1 loops=619)
            -> Single-row index lookup on Recipe using PRIMARY (recipe_id=Contain.recipe_id)  (cost=0.25 rows=1) (actual
time=0.001..0.001 rows=1 loops=861)
```

We choose it because it helps us to remove the category which name is “chesse”.
This index improves the query performance. Because the index on category helper filter find "cheese" quickly.

3. With index on Contain.recipe_id

```
-> Limit: 15 row(s)  (cost=3810.68 rows=15) (actual time=0.523..3.465 rows=15 loops=1)
    -> Group aggregate: count(Recipe.title)  (cost=3810.68 rows=58) (actual time=0.522..3.463 rows=15 loops=1)
        -> Nested loop inner join  (cost=3804.88 rows=58) (actual time=0.047..3.071 rows=861 loops=1)
            -> Nested loop inner join  (cost=689.28 rows=58) (actual time=0.037..1.830 rows=861 loops=1)
                -> Filter: (i.category <> 'cheese')  (cost=0.43 rows=13) (actual time=0.021..0.466 rows=619 loops=1)
                    -> Index scan on i using ecategory  (cost=0.43 rows=15) (actual time=0.019..0.272 rows=702 loops=1)
                -> Index lookup on Contain using ingredient_id (ingredient_id=i.ingredient_id)  (cost=0.25 rows=5) (actual
time=0.002..0.002 rows=1 loops=619)
            -> Single-row index lookup on Recipe using PRIMARY (recipe_id=Contain.recipe_id)  (cost=0.25 rows=1) (actual
time=0.001..0.001 rows=1 loops=861)
```

The reason why we choose it is that we need to join table using recipe_id to join Recipe and Contain, this helps query spend less time to find Recipe_id in Contain.

4. With index on Contain.ingredient_id

```
-> Limit: 15 row(s)  (actual time=17.669..17.673 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.003 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=17.669..17.671 rows=15 loops=1)
            -> Nested loop inner join  (cost=2937.27 rows=4832) (actual time=0.080..13.832 rows=4944 loops=1)
                -> Nested loop inner join  (cost=933.64 rows=5725) (actual time=0.068..4.148 rows=5048 loops=1)
                    -> Table scan on Recipe  (cost=119.85 rows=956) (actual time=0.050..0.439 rows=1000 loops=1)
                    -> Index lookup on Contain using PRIMARY (recipe_id=Recipe.recipe_id)  (cost=0.25 rows=6) (actual
time=0.002..0.003 rows=5 loops=1000)
                -> Filter: (i.category <> 'cheese')  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=5048)
                    -> Single-row index lookup on i using PRIMARY (ingredient_id=Contain.ingredient_id)  (cost=0.25 rows=1) (actual
time=0.001..0.001 rows=1 loops=5048)
```

This index does not improve query efficiency, the reason is ingredient_id is of little help in finding the category as "cheese" and aggregation, and it also does not improve the efficiency of aggregation. We can see that it takes almost the same time at aggregation as when index is not used.