Project Track 1 Stage 3:

Database tables on GCP:

```
mysql> show tables;
+----------------------------+
| Tables_in_411_Smaller      |
+----------------------------+
| Ingredients                |
| Markets                    |
| MarketsSellsIngredients    |
| Nutrition                  |
| RecipeIncludesIngredients  |
| Recipes                    |
| UserLikeRecipe             |
| UserRatesRecipe            |
| Users                      |
+----------------------------+
9 rows in set (0.00 sec)

mysql>
```

Database imported via .sql file.

DDL commands for our tables:

```
CREATE TABLE Users (
    user_id         INT NOT NULL,
    user_email      VARCHAR(50) NOT NULL UNIQUE,
    user_name       VARCHAR(20) NOT NULL,
    password        VARCHAR(100) NOT NULL,

    PRIMARY KEY (user_id)
);

CREATE TABLE Recipes (
    recipe_id       INT NOT NULL,
    recipe_name     VARCHAR(255) NOT NULL,
    recipe_type     VARCHAR(10),
    user_email      VARCHAR(50) NOT NULL,

    PRIMARY KEY (recipe_id),
    FOREIGN KEY (user_email) REFERENCES Users(user_email) ON DELETE CASCADE
);

CREATE TABLE Ingredients (
    ingredient_id   INT NOT NULL,
    ingredient_name VARCHAR(255) NOT NULL UNIQUE,
    ingredient_type VARCHAR(255),

    PRIMARY KEY (ingredient_id)
);

CREATE TABLE Nutrition (
    recipe_id       INT NOT NULL,
    calorie         DECIMAL(10, 2),
    carbon_hydrate  INT,
    fat             DECIMAL(5, 2),

    PRIMARY KEY (recipe_id),
    FOREIGN KEY (recipe_id) REFERENCES Recipes(recipe_id) ON DELETE CASCADE
);

CREATE TABLE Markets (
    market_id          INT NOT NULL,
    market_name        VARCHAR(30) NOT NULL,
    market_location    VARCHAR(50) NOT NULL,

    PRIMARY KEY (market_id)
);

#--------------- many-to-many tables ---------------#
CREATE TABLE UserLikeRecipe (
    user_id         INT NOT NULL,
    recipe_id       INT NOT NULL,

    PRIMARY KEY (user_id, recipe_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (recipe_id) REFERENCES Recipes(recipe_id)
);

CREATE TABLE UserRatesRecipe (
     user_id        INT NOT NULL,
     recipe_id      INT NOT NULL,
     rating         INT NOT NULL,
     comment        VARCHAR(200),

     PRIMARY KEY (user_id, recipe_id),
     FOREIGN KEY (user_id) REFERENCES Users(user_id),
     FOREIGN KEY (recipe_id) REFERENCES Recipes(recipe_id)
);

CREATE TABLE RecipeIncludesIngredients (
    recipe_id          INT NOT NULL,
    ingredient_id      INT NOT NULL,

    PRIMARY KEY (recipe_id, ingredient_id),
    FOREIGN KEY (recipe_id) REFERENCES Recipes(recipe_id) ON DELETE CASCADE,
    FOREIGN KEY (ingredient_id) REFERENCES Ingredients(ingredient_id) ON DELETE CASCADE
);

CREATE TABLE MarketsSellsIngredients (
    market_id          INT NOT NULL,
    ingredient_id      INT NOT NULL,

    PRIMARY KEY (market_id, ingredient_id),
    FOREIGN KEY (market_id) REFERENCES Markets(market_id) ON DELETE CASCADE,
    FOREIGN KEY (ingredient_id) REFERENCES Ingredients(ingredient_id) ON DELETE CASCADE
);
```

(May also check the table.sql file under the dataset branch)

Counts of rows:

```
mysql> SELECT COUNT(recipe_id) FROM Recipes;
+------------------+
| COUNT(recipe_id) |
+------------------+
|             1485 |
+------------------+
1 row in set (0.10 sec)

mysql> SELECT COUNT(ingredient_id) FROM Ingredients;
+----------------------+
| COUNT(ingredient_id) |
+----------------------+
|                 8023 |
+----------------------+
1 row in set (0.04 sec)

mysql> SELECT COUNT(recipe_id) FROM Nutrition;
+------------------+
| COUNT(recipe_id) |
+------------------+
|             1485 |
+------------------+
1 row in set (0.05 sec)

mysql>
```

```
mysql> SELECT COUNT(recipe_id) FROM RecipeIncludesIngredients;
+------------------+
| COUNT(recipe_id) |
+------------------+
|             9153 |
+------------------+
1 row in set (0.21 sec)

mysql>
```

1485 rows in the Recipe and Nutrition table;
8023 rows in the Ingredient table;
9153 rows in the many-to-many Recipe Include relation table;

Four advanced queries and corresponding result:

Query1:

```sql
SELECT
    Ingredients.ingredient_name,
    COUNT(Ingredients.ingredient_name) AS ingredient_count
FROM
    Ingredients
JOIN
    RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
JOIN
    Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
JOIN
    UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
WHERE
    UserRatesRecipe.rating >= 2
    AND UserRatesRecipe.comment LIKE '%good%'
GROUP BY
    Ingredients.ingredient_name
HAVING
    COUNT(DISTINCT Recipes.recipe_id) > 1 -- Ensures the ingredient is common across multiple recipes
ORDER BY
    ingredient_count DESC;
```

Result of query 1:

ingredient_name	ingredient_count
salt	11
onion	5
water	5
sugar	4
baking powder	3
butter	3
celery	3
baking soda	2
brown sugar	2
canola oil	2
carrot	2
chili powder	2
cinnamon	2
dried thyme	2
garlic	2

Query2:

```sql
SELECT
    Ingredients.ingredient_name,
    COUNT(*) AS popularity
FROM
    Ingredients
JOIN
    RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
JOIN
    Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
JOIN
    UserLikeRecipe ON Recipes.recipe_id = UserLikeRecipe.recipe_id
WHERE
    UserLikeRecipe.user_id > 2
    AND Recipes.recipe_name NOT LIKE '%shit%'
GROUP BY
    Ingredients.ingredient_name
ORDER BY
    popularity DESC
LIMIT 15;
```

Result of query 2(we limit the outputs in 15):

```
+-------------------+------------+
| ingredient_name   | popularity |
+-------------------+------------+
| salt              |         22 |
| onion             |         15 |
| butter            |          9 |
| sugar             |          9 |
| water             |          8 |
| milk              |          7 |
| honey             |          6 |
| garlic            |          6 |
| baking powder     |          6 |
| celery            |          5 |
| oil               |          5 |
| garlic powder     |          5 |
| baking soda       |          5 |
| egg               |          5 |
| vanilla ice cream |          4 |
+-------------------+------------+
15 rows in set (0.00 sec)
```

Query3:

```sql
SELECT
    Ingredients.ingredient_name,
    AVG(UserRatesRecipe.rating) AS average_rating
FROM
    Ingredients
JOIN
    RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
JOIN
    Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
JOIN
    UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
JOIN
    Nutrition ON Recipes.recipe_id = Nutrition.recipe_id
WHERE
    Recipes.recipe_id IN (
        SELECT
            RecipeIncludesIngredients.recipe_id
        FROM
            RecipeIncludesIngredients
        JOIN
            UserLikeRecipe ON RecipeIncludesIngredients.recipe_id = UserLikeRecipe.recipe_id
        GROUP BY
            RecipeIncludesIngredients.recipe_id
        HAVING
            COUNT(UserLikeRecipe.user_id) > 10
    )
    AND Nutrition.calorie < 200
    AND Nutrition.fat < 20
GROUP BY
    Ingredients.ingredient_name
ORDER BY
    average_rating DESC;
```

Result of query 3:

ingredient_name	average_rating
broccoli floret	4.0000
honey	4.0000
cinnamon	4.0000
butter	4.0000
brown sugar	4.0000
water	4.0000
salt and pepper	4.0000
red bell pepper	4.0000
onion	4.0000
ground coriander	4.0000
great northern bean	4.0000
garlic	4.0000
fresh herb	4.0000
escarole	4.0000
cauliflower	4.0000

Query4:

```
SELECT DISTINCT
    Ingredients.ingredient_name
FROM
    Ingredients
JOIN (
    SELECT
        RecipeIncludesIngredients.ingredient_id
    FROM
        RecipeIncludesIngredients
    JOIN
        UserRatesRecipe ON RecipeIncludesIngredients.recipe_id = UserRatesRecipe.recipe_id
    WHERE
        UserRatesRecipe.comment NOT LIKE '%bad%'
    GROUP BY
        RecipeIncludesIngredients.ingredient_id
    HAVING
        AVG(UserRatesRecipe.rating) > 1
) AS HighRatedIngredients ON Ingredients.ingredient_id = HighRatedIngredients.ingredient_id
JOIN (
    SELECT
        MarketsSellsIngredients.ingredient_id
    FROM
        MarketsSellsIngredients
    JOIN
        Markets ON MarketsSellsIngredients.market_id = Markets.market_id
    WHERE
        Markets.market_location LIKE '%GreenVille%'
) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id;
```

Result of query 4:

ingredient_name
chili powder
cornmeal
louisiana hot sauce
milk
buttermilk
cracked black pepper
dry mustard
orange, juice of
butter
cooking oil
red bell pepper
salt and pepper
cornstarch
lemon, juice of
black cumin

—--------------------Part 2—---

1. Our first advanced query:

This is the first advanced query

```
SELECT
    Ingredients.ingredient_name,
    COUNT(Ingredients.ingredient_name) AS ingredient_count
FROM
    Ingredients
JOIN
    RecipeIncludesIngredients ON Ingredients.ingredient_id =
RecipeIncludesIngredients.ingredient_id
JOIN
    Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
JOIN
    UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
WHERE
    UserRatesRecipe.rating >= 2
    AND UserRatesRecipe.comment LIKE '%good%'
GROUP BY
    Ingredients.ingredient_name
HAVING
    COUNT(DISTINCT Recipes.recipe_id) > 1 -- Ensures the ingredient is common across
multiple recipes
ORDER BY
    ingredient_count DESC;
```

1a. Create index for rating:

Before, the cost is:

23.89+13.50+6.82 = 44.21

```
mysql> use 411_Smaller;
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> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     COUNT(Ingredients.ingredient_name) AS ingredient_count
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
    -> WHERE
    ->     UserRatesRecipe.rating >= 2
    ->     AND UserRatesRecipe.comment LIKE '%good%'
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> HAVING
    ->     COUNT(DISTINCT Recipes.recipe_id) > 1 -- Ensures the ingredient is common across multiple recipes
    -> ORDER BY
    ->     ingredient_count DESC;
+--------------------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------+
| -> Sort: ingredient_count DESC  (actual time=1.195..1.197 rows=28 loops=1)
    -> Filter: (count(distinct Recipes.recipe_id) > 1)  (actual time=1.017..1.133 rows=28 loops=1)
        -> Stream results  (actual time=0.942..1.119 rows=108 loops=1)
            -> Group aggregate: count(distinct Recipes.recipe_id), count(Ingredients.ingredient_name)  (actual time=0.939..1.023 rows=108 loops=1)
                -> Sort: Ingredients.ingredient_name  (actual time=0.929..0.956 rows=156 loops=1)
                    -> Stream results  (cost=23.89 rows=30) (actual time=0.095..0.632 rows=156 loops=1)
                        -> Nested loop inner join  (cost=23.89 rows=30) (actual time=0.092..0.572 rows=156 loops=1)
                            -> Nested loop inner join  (cost=13.50 rows=30) (actual time=0.081..0.306 rows=156 loops=1)
                                -> Nested loop inner join  (cost=6.82 rows=5) (actual time=0.064..0.137 rows=19 loops=1)
                                    -> Filter: ((UserRatesRecipe.rating >= 2) and (UserRatesRecipe.`comment` like '%good%'))  (cost=5.15 rows=5) (actual time=0.050.
.0.085 rows=19 loops=1)
                                        -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.041..0.064 rows=53 loops=1)
                                    -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.27 rows=1) (actual t
ime=0.002..0.002 rows=1 loops=19)
                                -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.91 rows=6) (actua
l time=0.006..0.008 rows=8 loops=19)
                            -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (act
ual time=0.001..0.002 rows=1 loops=156)
 |
+-----------------------------------------------------------------------------
```

After create index for rating, the cost is:

23.89+13.5+6.82=44.21

```
------------------------------------------------------------------------------
----------------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> CREATE INDEX rt_idx on UserRatesRecipe(rating);
ERROR 1061 (42000): Duplicate key name 'rt_idx'
mysql> CREATE INDEX rate_idx on UserRatesRecipe(rating);
Query OK, 0 rows affected, 1 warning (0.18 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     COUNT(Ingredients.ingredient_name) AS ingredient_count FROM      Ingredients JOIN      RecipeInc
ludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
 JOIN      UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id WHERE      UserRatesRecipe.rating >= 2     AND UserRatesRecipe.comment LIKE '%good%' GRO
UP BY      Ingredients.ingredient_name HAVING      COUNT(DISTINCT Recipes.recipe_id) > 1  ORDER BY      ingredient_count DESC;
+-----------------------------------------------------------------------------
------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----------------------------------------------------------------------------+
| -> Sort: ingredient_count DESC  (actual time=1.035..1.037 rows=28 loops=1)
    -> Filter: (count(distinct Recipes.recipe_id) > 1)  (actual time=0.907..1.004 rows=28 loops=1)
        -> Stream results  (actual time=0.903..0.995 rows=108 loops=1)
            -> Group aggregate: count(distinct Recipes.recipe_id), count(Ingredients.ingredient_name)  (actual time=0.901..0.967 rows=108 loops=1)
                -> Sort: Ingredients.ingredient_name  (actual time=0.895..0.908 rows=156 loops=1)
                    -> Stream results  (cost=23.89 rows=30) (actual time=0.112..0.593 rows=156 loops=1)
                        -> Nested loop inner join  (cost=23.89 rows=30) (actual time=0.109..0.538 rows=156 loops=1)
                            -> Nested loop inner join  (cost=13.50 rows=30) (actual time=0.102..0.301 rows=156 loops=1)
                                -> Nested loop inner join  (cost=6.82 rows=5) (actual time=0.087..0.141 rows=19 loops=1)
                                    -> Filter: ((UserRatesRecipe.rating >= 2) and (UserRatesRecipe.`comment` like '%good%'))  (cost=5.15 rows=5) (actual time=0.074.
.0.097 rows=19 loops=1)
                                        -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.067..0.079 rows=53 loops=1)
                                    -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.27 rows=1) (actual t
ime=0.002..0.002 rows=1 loops=19)
                                -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.91 rows=6) (actua
l time=0.005..0.008 rows=8 loops=19)
                            -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (act
ual time=0.001..0.001 rows=1 loops=156)
 |
+-----------------------------------------------------------------------------
```

We can see that when we create index for rating, the cost does not change.

1b Create index for comment:

Before create index, the cost is: 25.85+15.45+8.78 = 50.08

```
mysql> use 411_Smaller;
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> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     COUNT(Ingredients.ingredient_name) AS ingredient_count FROM      Ingredients JOIN      RecipeInc
ludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
 JOIN      UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id WHERE      UserRatesRecipe.rating >= 2     AND UserRatesRecipe.comment LIKE '%good%' GRO
UP BY      Ingredients.ingredient_name HAVING      COUNT(DISTINCT Recipes.recipe_id) > 1  ORDER BY      ingredient_count DESC;
+-----------------------------------------------------------------------------
------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----------------------------------------------------------------------------+
| -> Sort: ingredient_count DESC  (actual time=1.413..1.416 rows=28 loops=1)
    -> Filter: (count(distinct Recipes.recipe_id) > 1)  (actual time=1.308..1.399 rows=28 loops=1)
        -> Stream results  (actual time=1.304..1.390 rows=108 loops=1)
            -> Group aggregate: count(distinct Recipes.recipe_id), count(Ingredients.ingredient_name)  (actual time=1.302..1.362 rows=108 loops=1)
                -> Sort: Ingredients.ingredient_name  (actual time=1.282..1.294 rows=156 loops=1)
                    -> Stream results  (cost=25.85 rows=30) (actual time=0.100..0.732 rows=156 loops=1)
                        -> Nested loop inner join  (cost=25.85 rows=30) (actual time=0.097..0.667 rows=156 loops=1)
                            -> Nested loop inner join  (cost=15.45 rows=30) (actual time=0.088..0.331 rows=156 loops=1)
                                -> Nested loop inner join  (cost=8.78 rows=5) (actual time=0.067..0.133 rows=19 loops=1)
                                    -> Filter: ((UserRatesRecipe.rating >= 2) and (UserRatesRecipe.`comment` like '%good%'))  (cost=5.15 rows=5) (actual time=0.052.
.0.080 rows=19 loops=1)
                                        -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.042..0.057 rows=53 loops=1)
                                    -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.68 rows=1) (actual t
ime=0.002..0.002 rows=1 loops=19)
                                -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.91 rows=6) (actua
l time=0.007..0.009 rows=8 loops=19)
                            -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (act
ual time=0.002..0.002 rows=1 loops=156)
 |
```

After creating index for comment, the cost is:25.85+15.45+8.78 = 50.08

```
1 row in set (0.00 sec)

mysql> CREATE INDEX cmt_idx on UserRatesRecipe(comment);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     COUNT(Ingredients.ingredient_name) AS ingredient_count
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
    -> WHERE
    ->     UserRatesRecipe.rating >= 2
    ->     AND UserRatesRecipe.comment LIKE '%good%'
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> HAVING
    ->     COUNT(DISTINCT Recipes.recipe_id) > 1 -- Ensures the ingredient is common across multiple recipes
    -> ORDER BY
    ->     ingredient_count DESC;
```

```
| -> Sort: ingredient_count DESC  (actual time=1.210..1.212 rows=28 loops=1)
    -> Filter: (count(distinct Recipes.recipe_id) > 1)  (actual time=1.036..1.196 rows=28 loops=1)
        -> Stream results  (actual time=1.031..1.185 rows=108 loops=1)
            -> Group aggregate: count(distinct Recipes.recipe_id), count(Ingredients.ingredient_name)  (actual time=1.028..1.152 rows=108 loops=1)
                -> Sort: Ingredients.ingredient_name  (actual time=1.021..1.035 rows=156 loops=1)
                    -> Stream results  (cost=25.85 rows=30) (actual time=0.085..0.754 rows=156 loops=1)
                        -> Nested loop inner join  (cost=25.85 rows=30) (actual time=0.082..0.680 rows=156 loops=1)
                            -> Nested loop inner join  (cost=15.45 rows=30) (actual time=0.074..0.296 rows=156 loops=1)
                                -> Nested loop inner join  (cost=8.78 rows=5) (actual time=0.058..0.133 rows=19 loops=1)
                                    -> Filter: ((UserRatesRecipe.rating >= 2) and (UserRatesRecipe.`comment` like '%good%'))  (cost=5.15 rows=5) (actual time=0.046
.0.080 rows=19 loops=1)
                                        -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.039..0.056 rows=53 loops=1)
                                -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.68 rows=1) (actual t
ime=0.002..0.002 rows=1 loops=19)
                            -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.91 rows=6) (actua
l time=0.006..0.008 rows=8 loops=19)
                        -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (act
ual time=0.002..0.002 rows=1 loops=156)
 |
```

We can see that the time does not change.

1c. Create index for ingredient_name:

Before creating index, the cost is: 49.03+16.35+9.43=74.81

```
Database changed
mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     COUNT(Ingredients.ingredient_name) AS ingredient_count
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
    -> WHERE
    ->     UserRatesRecipe.rating >= 2
    ->     AND UserRatesRecipe.comment LIKE '%good%'
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> HAVING
    ->     COUNT(DISTINCT Recipes.recipe_id) > 1 -- Ensures the ingredient is common across multiple recipes
    -> ORDER BY
    ->     ingredient_count DESC;
```

```
| -> Sort: ingredient_count DESC  (actual time=221.967..221.969 rows=28 loops=1)
    -> Filter: (count(distinct Recipes.recipe_id) > 1)  (actual time=221.769..221.953 rows=28 loops=1)
        -> Stream results  (actual time=221.764..221.943 rows=108 loops=1)
            -> Group aggregate: count(distinct Recipes.recipe_id), count(Ingredients.ingredient_name)  (actual time=221.762..221.913 rows=108 loops=1)
                -> Sort: Ingredients.ingredient_name  (actual time=221.752..221.766 rows=156 loops=1)
                    -> Stream results  (cost=49.03 rows=30) (actual time=37.908..221.415 rows=156 loops=1)
                        -> Nested loop inner join  (cost=49.03 rows=30) (actual time=37.897..221.156 rows=156 loops=1)
                            -> Nested loop inner join  (cost=16.35 rows=30) (actual time=0.076..0.741 rows=156 loops=1)
                                -> Nested loop inner join  (cost=9.43 rows=5) (actual time=0.063..0.361 rows=19 loops=1)
                                    -> Filter: ((UserRatesRecipe.rating >= 2) and (UserRatesRecipe.`comment` like '%good%'))  (cost=5.15 rows=5) (actual time=0.045.
.0.208 rows=19 loops=1)
                                        -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.037..0.083 rows=53 loops=1)
                                -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.82 rows=1) (actual t
ime=0.008..0.008 rows=1 loops=19)
                            -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.96 rows=6) (actua
l time=0.008..0.019 rows=8 loops=19)
                        -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=1.00 rows=1) (act
ual time=1.412..1.412 rows=1 loops=156)
 |
```

After creating index for ingredient_name, the cost is: 26.75+16.35+9.43=52.53

```
mysql> CREATE INDEX igd_name3_idx on Ingredients(ingredient_name);
Query OK, 0 rows affected, 1 warning (0.29 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     COUNT(Ingredients.ingredient_name) AS ingredient_count FROM      Ingredients JOIN      RecipeInc
ludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
 JOIN      UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id WHERE      UserRatesRecipe.rating >= 2     AND UserRatesRecipe.comment LIKE '%good%' GRO
UP BY      Ingredients.ingredient_name HAVING      COUNT(DISTINCT Recipes.recipe_id) > 1  ORDER BY      ingredient_count DESC;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
| -> Sort: ingredient_count DESC  (actual time=1.174..1.176 rows=28 loops=1)
    -> Filter: (count(distinct Recipes.recipe_id) > 1)  (actual time=1.070..1.159 rows=28 loops=1)
        -> Stream results  (actual time=1.066..1.150 rows=108 loops=1)
            -> Group aggregate: count(distinct Recipes.recipe_id), count(Ingredients.ingredient_name)  (actual time=1.064..1.122 rows=108 loops=1)
                -> Sort: Ingredients.ingredient_name  (actual time=1.056..1.068 rows=156 loops=1)
                    -> Stream results  (cost=26.75 rows=30) (actual time=0.067..0.974 rows=156 loops=1)
                        -> Nested loop inner join  (cost=26.75 rows=30) (actual time=0.065..0.913 rows=156 loops=1)
                            -> Nested loop inner join  (cost=16.35 rows=30) (actual time=0.056..0.290 rows=156 loops=1)
                                -> Nested loop inner join  (cost=9.43 rows=5) (actual time=0.044..0.140 rows=19 loops=1)
                                    -> Filter: ((UserRatesRecipe.rating >= 2) and (UserRatesRecipe.`comment` like '%good%'))  (cost=5.15 rows=5) (actual time=0.032.
.0.075 rows=19 loops=1)
                                        -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.025..0.056 rows=53 loops=1)
                                    -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.82 rows=1) (actual t
ime=0.003..0.003 rows=1 loops=19)
                                -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.96 rows=6) (actua
l time=0.005..0.007 rows=8 loops=19)
                            -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (act
ual time=0.004..0.004 rows=1 loops=156)
 |
+----------------------------------------------------------------------------+
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

It is obvious that the cost has decreased.

---

2. Our second advanced query:

```
SELECT
    Ingredients.ingredient_name,
    COUNT(*) AS popularity
FROM
    Ingredients
JOIN
    RecipeIncludesIngredients ON Ingredients.ingredient_id =
RecipeIncludesIngredients.ingredient_id
JOIN
    Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
JOIN
    UserLikeRecipe ON Recipes.recipe_id = UserLikeRecipe.recipe_id
WHERE
    UserLikeRecipe.user_id > 2
    AND Recipes.recipe_name NOT LIKE '%shit%'
GROUP BY
    Ingredients.ingredient_name
ORDER BY
    popularity DESC
LIMIT 15;
```

2a: Create index for user_id:

Before creating index, the cost is: 276.55+132.15+61.30 = 470

```
Database changed
mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     COUNT(*) AS popularity
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserLikeRecipe ON Recipes.recipe_id = UserLikeRecipe.recipe_id
    -> WHERE
    ->     UserLikeRecipe.user_id > 2
    ->     AND Recipes.recipe_name NOT LIKE '%shit%'
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> ORDER BY
    ->     popularity DESC
    -> LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=36.358..36.360 rows=15 loops=1)
    -> Sort: popularity DESC, limit input to 15 row(s) per chunk  (actual time=36.357..36.358 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=36.275..36.305 rows=185 loops=1)
            -> Aggregate using temporary table  (actual time=36.272..36.272 rows=185 loops=1)
                -> Nested loop inner join  (cost=276.55 rows=304) (actual time=0.147..35.537 rows=368 loops=1)
                    -> Nested loop inner join  (cost=132.15 rows=304) (actual time=0.116..15.217 rows=368 loops=1)
                        -> Nested loop inner join  (cost=61.30 rows=49) (actual time=0.090..0.652 rows=55 loops=1)
                            -> Filter: (UserLikeRecipe.user_id > 2)  (cost=12.05 rows=55) (actual time=0.051..0.162 rows=55 loops=1)
                                -> Covering index range scan on UserLikeRecipe using PRIMARY over (2 < user_id)  (cost=12.05 rows=55) (actual time=0.046..0.112 rows
=55 loops=1)
                            -> Filter: (not((Recipes.recipe_name like '%shit%')))  (cost=0.80 rows=1) (actual time=0.008..0.009 rows=1 loops=55)
                                -> Single-row index lookup on Recipes using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.006..0.0
06 rows=1 loops=55)
                        -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.84 rows=6) (actual time=0.
261..0.264 rows=7 loops=55)
                    -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.38 rows=1) (actual time
=0.055..0.055 rows=1 loops=368)
 |
+----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
```

- -

After creating index for user_id, the cost is: 242.82+130.09+60.52=433.43

```
-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
1 row in set (0.10 sec)

mysql> CREATE INDEX usid_idx on UserLikeRecipe(user_id);
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     COUNT(*) AS popularity FROM      Ingredients JOIN      RecipeIncludesIngredients ON Ingredients.
ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id JOIN      UserLikeRecipe ON Rec
ipes.recipe_id = UserLikeRecipe.recipe_id WHERE      UserLikeRecipe.user_id > 2     AND Recipes.recipe_name NOT LIKE '%shit%' GROUP BY      Ingredients.ingredient_n
ame ORDER BY      popularity DESC LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=2.359..2.361 rows=15 loops=1)
    -> Sort: popularity DESC, limit input to 15 row(s) per chunk  (actual time=2.358..2.359 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.292..2.318 rows=185 loops=1)
            -> Aggregate using temporary table  (actual time=2.289..2.289 rows=185 loops=1)
                -> Nested loop inner join  (cost=242.82 rows=304) (actual time=0.087..1.893 rows=368 loops=1)
                    -> Nested loop inner join  (cost=130.09 rows=304) (actual time=0.073..0.601 rows=368 loops=1)
                        -> Nested loop inner join  (cost=60.52 rows=49) (actual time=0.056..0.245 rows=55 loops=1)
                            -> Filter: (UserLikeRecipe.user_id > 2)  (cost=11.27 rows=55) (actual time=0.030..0.059 rows=55 loops=1)
                                -> Covering index range scan on UserLikeRecipe using PRIMARY over (2 < user_id)  (cost=11.27 rows=55) (actual time=0.028..0.050 rows
=55 loops=1)
                            -> Filter: (not((Recipes.recipe_name like '%shit%')))  (cost=0.80 rows=1) (actual time=0.003..0.003 rows=1 loops=55)
                                -> Single-row index lookup on Recipes using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.002..0.0
02 rows=1 loops=55)
                        -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.81 rows=6) (actual time=0.
004..0.006 rows=7 loops=55)
                    -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.27 rows=1) (actual time
=0.003..0.003 rows=1 loops=368)
 |
+----------------------------------------------------------------------------------------------------------------------------
```

We can see that the time does decrease.

2b: Create index for recipe_name:

Before creating index, the cost is: 244.10+131.37+60.52 = 435.99

```
Database changed
mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     COUNT(*) AS popularity
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserLikeRecipe ON Recipes.recipe_id = UserLikeRecipe.recipe_id
    -> WHERE
    ->     UserLikeRecipe.user_id > 2
    ->     AND Recipes.recipe_name NOT LIKE '%shit%'
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> ORDER BY
    ->     popularity DESC
    -> LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=3.767..3.771 rows=15 loops=1)
    -> Sort: popularity DESC, limit input to 15 row(s) per chunk  (actual time=3.766..3.768 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=3.617..3.677 rows=185 loops=1)
            -> Aggregate using temporary table  (actual time=3.614..3.614 rows=185 loops=1)
                -> Nested loop inner join  (cost=244.10 rows=304) (actual time=0.064..3.080 rows=368 loops=1)
                    -> Nested loop inner join  (cost=131.37 rows=304) (actual time=0.052..0.842 rows=368 loops=1)
                        -> Nested loop inner join  (cost=60.52 rows=49) (actual time=0.039..0.365 rows=55 loops=1)
                            -> Filter: (UserLikeRecipe.user_id > 2)  (cost=11.27 rows=55) (actual time=0.020..0.078 rows=55 loops=1)
                                -> Covering index range scan on UserLikeRecipe using PRIMARY over (2 < user_id)  (cost=11.27 rows=55) (actual time=0.018..0.061 rows
=55 loops=1)
                            -> Filter: (not((Recipes.recipe_name like '%shit%')))  (cost=0.80 rows=1) (actual time=0.005..0.005 rows=1 loops=55)
                                -> Single-row index lookup on Recipes using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.004..0.0
04 rows=1 loops=55)
                        -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.84 rows=6) (actual time=0.
005..0.008 rows=7 loops=55)
                    -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.27 rows=1) (actual time
=0.006..0.006 rows=1 loops=368)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

After creating index for recipe_name, the cost is: 214.10+101.37+30.52=345.99

```
-------------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> CREATE INDEX rcp_nm2_idx on Recipes(recipe_name);
Query OK, 0 rows affected, 1 warning (0.46 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     COUNT(*) AS popularity FROM      Ingredients JOIN      RecipeIncludesIngredients ON Ingredients.
ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id JOIN      UserLikeRecipe ON Rec
ipes.recipe_id = UserLikeRecipe.recipe_id WHERE      UserLikeRecipe.user_id > 2     AND Recipes.recipe_name NOT LIKE '%shit%' GROUP BY      Ingredients.ingredient_n
ame ORDER BY      popularity DESC LIMIT 15;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=2.239..2.241 rows=15 loops=1)
    -> Sort: popularity DESC, limit input to 15 row(s) per chunk  (actual time=2.238..2.238 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.174..2.199 rows=185 loops=1)
            -> Aggregate using temporary table  (actual time=2.172..2.172 rows=185 loops=1)
                -> Nested loop inner join  (cost=214.10 rows=304) (actual time=0.063..1.855 rows=368 loops=1)
                    -> Nested loop inner join  (cost=101.37 rows=304) (actual time=0.051..0.584 rows=368 loops=1)
                        -> Nested loop inner join  (cost=30.52 rows=49) (actual time=0.028..0.230 rows=55 loops=1)
                            -> Filter: (UserLikeRecipe.user_id > 2)  (cost=11.27 rows=55) (actual time=0.011..0.042 rows=55 loops=1)
                                -> Covering index range scan on UserLikeRecipe using PRIMARY over (2 < user_id)  (cost=11.27 rows=55) (actual time=0.009..0.033 rows
=55 loops=1)
                            -> Filter: (not((Recipes.recipe_name like '%shit%')))  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=55)
                                -> Single-row index lookup on Recipes using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.25 rows=1) (actual time=0.002..0.0
03 rows=1 loops=55)
                        -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.84 rows=6) (actual time=0.
004..0.006 rows=7 loops=55)
                    -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.27 rows=1) (actual time
=0.003..0.003 rows=1 loops=368)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

We can see that the time does decrease.

2c: Create index for ingredient_name:

Before creating index, the cost is: 244.10+131.37+60.52=435.99

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

Database changed
mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     COUNT(*) AS popularity
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserLikeRecipe ON Recipes.recipe_id = UserLikeRecipe.recipe_id
    -> WHERE
    ->     UserLikeRecipe.user_id > 2
    ->     AND Recipes.recipe_name NOT LIKE '%shit%'
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> ORDER BY
    ->     popularity DESC
    -> LIMIT 15;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=2.362..2.363 rows=15 loops=1)
    -> Sort: popularity DESC, limit input to 15 row(s) per chunk  (actual time=2.361..2.361 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.288..2.317 rows=185 loops=1)
            -> Aggregate using temporary table  (actual time=2.286..2.286 rows=185 loops=1)
                -> Nested loop inner join  (cost=244.10 rows=304) (actual time=0.126..1.962 rows=368 loops=1)
                    -> Nested loop inner join  (cost=131.37 rows=304) (actual time=0.107..0.659 rows=368 loops=1)
                        -> Nested loop inner join  (cost=60.52 rows=49) (actual time=0.089..0.319 rows=55 loops=1)
                            -> Filter: (UserLikeRecipe.user_id > 2)  (cost=11.27 rows=55) (actual time=0.058..0.089 rows=55 loops=1)
                                -> Covering index range scan on UserLikeRecipe using PRIMARY over (2 < user_id)  (cost=11.27 rows=55) (actual time=0.052..0.076 rows
=55 loops=1)
                            -> Filter: (not((Recipes.recipe_name like '%shit%')))  (cost=0.80 rows=1) (actual time=0.004..0.004 rows=1 loops=55)
                                -> Single-row index lookup on Recipes using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.003..0.0
03 rows=1 loops=55)
                        -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.84 rows=6) (actual time=0.
004..0.006 rows=7 loops=55)
                    -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.27 rows=1) (actual time
=0.003..0.003 rows=1 loops=368)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

After creating index for ingredient_name, the cost is: 237.77+131.37+60.52=429.66

```
-----------------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> CREATE INDEX igdnm_3_idx on Ingredients(ingredient_name);
Query OK, 0 rows affected, 1 warning (0.49 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     COUNT(*) AS popularity FROM      Ingredients JOIN      RecipeIncludesIngredients ON Ingredients.
ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id JOIN      UserLikeRecipe ON Rec
ipes.recipe_id = UserLikeRecipe.recipe_id WHERE      UserLikeRecipe.user_id > 2      AND Recipes.recipe_name NOT LIKE '%shit%' GROUP BY      Ingredients.ingredient_n
ame ORDER BY      popularity DESC LIMIT 15;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=2.205..2.207 rows=15 loops=1)
    -> Sort: popularity DESC, limit input to 15 row(s) per chunk  (actual time=2.204..2.205 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.124..2.149 rows=185 loops=1)
            -> Aggregate using temporary table  (actual time=2.121..2.121 rows=185 loops=1)
                -> Nested loop inner join  (cost=237.77 rows=304) (actual time=0.049..1.827 rows=368 loops=1)
                    -> Nested loop inner join  (cost=131.37 rows=304) (actual time=0.039..0.568 rows=368 loops=1)
                        -> Nested loop inner join  (cost=60.52 rows=49) (actual time=0.028..0.233 rows=55 loops=1)
                            -> Filter: (UserLikeRecipe.user_id > 2)  (cost=11.27 rows=55) (actual time=0.010..0.040 rows=55 loops=1)
                                -> Covering index range scan on UserLikeRecipe using PRIMARY over (2 < user_id)  (cost=11.27 rows=55) (actual time=0.009..0.031 rows
=55 loops=1)
                            -> Filter: (not((Recipes.recipe_name like '%shit%')))  (cost=0.80 rows=1) (actual time=0.003..0.003 rows=1 loops=55)
                                -> Single-row index lookup on Recipes using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.003..0.0
03 rows=1 loops=55)
                        -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0.84 rows=6) (actual time=0.
004..0.006 rows=7 loops=55)
                    -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (actual time
=0.003..0.003 rows=1 loops=368)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

We can see that the time does decrease.

---

3. Our third advanced query:

```
SELECT
    Ingredients.ingredient_name,
    AVG(UserRatesRecipe.rating) AS average_rating
FROM
    Ingredients
JOIN
    RecipeIncludesIngredients ON Ingredients.ingredient_id =
RecipeIncludesIngredients.ingredient_id
JOIN
    Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
JOIN
    UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
JOIN
    Nutrition ON Recipes.recipe_id = Nutrition.recipe_id
WHERE
    Recipes.recipe_id IN (
        SELECT
            RecipeIncludesIngredients.recipe_id
        FROM
            RecipeIncludesIngredients
        JOIN
            UserLikeRecipe ON RecipeIncludesIngredients.recipe_id =
UserLikeRecipe.recipe_id
        GROUP BY
            RecipeIncludesIngredients.recipe_id
        HAVING
            COUNT(UserLikeRecipe.user_id) > 10
    )
    AND Nutrition.calorie < 200
    AND Nutrition.fat < 20
```

GROUP BY
    Ingredients.ingredient_name
ORDER BY
    average_rating DESC;

3a: create index for calorie:

Before creating index, the cost is: 59.93+48.09+40.20+23.05+87.00=258.27

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

Database changed
mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     AVG(UserRatesRecipe.rating) AS average_rating
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
    -> JOIN
    ->     Nutrition ON Recipes.recipe_id = Nutrition.recipe_id
    -> WHERE
    ->     Recipes.recipe_id IN (
    ->         SELECT
    ->             RecipeIncludesIngredients.recipe_id
    ->         FROM
    ->             RecipeIncludesIngredients
    ->         JOIN
    ->             UserLikeRecipe ON RecipeIncludesIngredients.recipe_id = UserLikeRecipe.recipe_id
    ->         GROUP BY
    ->             RecipeIncludesIngredients.recipe_id
    ->         HAVING
    ->             COUNT(UserLikeRecipe.user_id) > 10
    ->     )
    ->     AND Nutrition.calorie < 200
    ->     AND Nutrition.fat < 20
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> ORDER BY
    ->     average_rating DESC;
+---------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------+
| -> Sort: average_rating DESC  (actual time=120.759..120.761 rows=32 loops=1)
    -> Table scan on <temporary>  (actual time=120.712..120.717 rows=32 loops=1)
        -> Aggregate using temporary table  (actual time=120.710..120.710 rows=32 loops=1)
            -> Nested loop inner join  (cost=59.93 rows=34) (actual time=120.083..120.605 rows=42 loops=1)
                -> Nested loop inner join  (cost=48.09 rows=34) (actual time=119.956..120.185 rows=42 loops=1)
                    -> Nested loop inner join  (cost=40.20 rows=5) (actual time=119.940..120.116 rows=4 loops=1)
                        -> Nested loop inner join  (cost=23.05 rows=49) (actual time=119.879..119.977 rows=14 loops=1)
                            -> Filter: <in_optimizer>(UserRatesRecipe.recipe_id,UserRatesRecipe.recipe_id in (select #2))  (cost=5.90 rows=49) (actual time=119.838..119.900 rows=14
loops=1)
                                -> Covering index scan on UserRatesRecipe using rt_idx  (cost=5.90 rows=49) (actual time=34.952..34.985 rows=53 loops=1)
                                -> Select #2 (subquery in condition; run only once)
                                    -> Filter: ((UserRatesRecipe.recipe_id = `<materialized_subquery>`.recipe_id))  (cost=0.00..0.00 rows=0) (actual time=1.632..1.632 rows=0 loops=5
2)
                                        -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=1.632..1.632 rows=0 loops=52)
                                            -> Index lookup on <materialized_subquery> using <auto_distinct_key> (recipe_id=UserRatesRecipe.recipe_id)  (actual time=1.631..1.631 row
s=0 loops=52)
                                                -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=84.795..84.795 rows=12 loops=1)
                                                    -> Filter: (count(UserLikeRecipe.user_id) > 10)  (actual time=84.748..84.773 rows=12 loops=1)
                                                        -> Table scan on <temporary>  (actual time=84.741..84.750 rows=52 loops=1)
                                                            -> Aggregate using temporary table  (actual time=84.736..84.736 rows=52 loops=1)
                                                                -> Nested loop inner join  (cost=87.00 rows=348) (actual time=48.610..84.492 rows=411 loops=1)
                                                                    -> Covering index scan on UserLikeRecipe using recipe_id  (cost=5.85 rows=56) (actual time=48.566..48.582 rows=60
 loops=1)
                                                                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0
.84 rows=6) (actual time=0.596..0.598 rows=7 loops=60)
                            -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 l
oops=14)
                        -> Filter: ((Nutrition.calorie < 200.00) and (Nutrition.fat < 20.00))  (cost=0.25 rows=0.1) (actual time=0.010..0.010 rows=0 loops=14)
                            -> Single-row index lookup on Nutrition using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=14
)
                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.94 rows=6) (actual time=0.013..0.016 rows=10 l
oops=4)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (actual time=0.010..0.010 rows=1
loops=42)
 |
+---------------------------------------------------------------------------------------------------------------------------
```

- -

After creating index for calorie, the cost is: 59.68+47.90+40.20+23.05+85.54=256.37

```
----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
1 row in set (0.13 sec)

mysql> CREATE INDEX clr_idx on Nutrition(calorie);
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     AVG(UserRatesRecipe.rating) AS average_rating FROM      Ingredients JOIN      RecipeIncludesIngredients ON Ingred
ients.ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id JOIN      UserRatesRecipe ON Recipes.recip
e_id = UserRatesRecipe.recipe_id JOIN      Nutrition ON Recipes.recipe_id = Nutrition.recipe_id WHERE      Recipes.recipe_id IN (         SELECT             RecipeIncludesIngredien
ts.recipe_id         FROM             RecipeIncludesIngredients         JOIN             UserLikeRecipe ON RecipeIncludesIngredients.recipe_id = UserLikeRecipe.recipe_id         G
ROUP BY             RecipeIncludesIngredients.recipe_id         HAVING             COUNT(UserLikeRecipe.user_id) > 10     )     AND Nutrition.calorie < 200     AND Nutrition.fat <
 20 GROUP BY      Ingredients.ingredient_name ORDER BY      average_rating DESC;
+---------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------+
| -> Sort: average_rating DESC  (actual time=1.182..1.184 rows=32 loops=1)
    -> Table scan on <temporary>  (actual time=1.068..1.073 rows=32 loops=1)
        -> Aggregate using temporary table  (actual time=1.068..1.068 rows=32 loops=1)
            -> Nested loop inner join  (cost=59.68 rows=34) (actual time=0.709..1.012 rows=42 loops=1)
                -> Nested loop inner join  (cost=47.90 rows=34) (actual time=0.693..0.837 rows=42 loops=1)
                    -> Nested loop inner join  (cost=40.20 rows=5) (actual time=0.686..0.806 rows=4 loops=1)
                        -> Nested loop inner join  (cost=23.05 rows=49) (actual time=0.666..0.753 rows=14 loops=1)
                            -> Filter: <in_optimizer>(UserRatesRecipe.recipe_id,UserRatesRecipe.recipe_id in (select #2))  (cost=5.90 rows=49) (actual time=0.648..0.701 rows=14 loop
s=1)
                                -> Covering index scan on UserRatesRecipe using rt_idx  (cost=5.90 rows=49) (actual time=0.032..0.041 rows=53 loops=1)
                                -> Select #2 (subquery in condition; run only once)
                                    -> Filter: ((UserRatesRecipe.recipe_id = `<materialized_subquery>`.recipe_id))  (cost=0.00..0.00 rows=0) (actual time=0.012..0.012 rows=0 loops=5
2)
                                        -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.012..0.012 rows=0 loops=52)
                                            -> Index lookup on <materialized_subquery> using <auto_distinct_key> (recipe_id=UserRatesRecipe.recipe_id)  (actual time=0.012..0.012 row
s=0 loops=52)
                                                -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=0.590..0.590 rows=12 loops=1)
                                                    -> Filter: (count(UserLikeRecipe.user_id) > 10)  (actual time=0.570..0.580 rows=12 loops=1)
                                                        -> Table scan on <temporary>  (actual time=0.568..0.575 rows=52 loops=1)
                                                            -> Aggregate using temporary table  (actual time=0.566..0.566 rows=52 loops=1)
                                                                -> Nested loop inner join  (cost=85.54 rows=348) (actual time=0.033..0.454 rows=411 loops=1)
                                                                    -> Covering index scan on UserLikeRecipe using recipe_id  (cost=5.85 rows=56) (actual time=0.015..0.025 rows=60 l
oops=1)
                                                                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0
.81 rows=6) (actual time=0.005..0.007 rows=7 loops=60)
                            -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 l
oops=14)
                        -> Filter: ((Nutrition.calorie < 200.00) and (Nutrition.fat < 20.00))  (cost=0.25 rows=0.1) (actual time=0.004..0.004 rows=0 loops=14)
                            -> Single-row index lookup on Nutrition using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=14
)
                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.92 rows=6) (actual time=0.004..0.007 rows=10 l
oops=4)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1
loops=42)
 |
+-----------------------------------------+
```

We can see that the time decreases a little bit.

3b: create index for fat:

Before creating index, the cost is: 118.97+96.68+88.98+49.78+86.29=440.7

```
Database changed
mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     AVG(UserRatesRecipe.rating) AS average_rating
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
    -> JOIN
    ->     Nutrition ON Recipes.recipe_id = Nutrition.recipe_id
    -> WHERE
    ->     Recipes.recipe_id IN (
    ->         SELECT
    ->             RecipeIncludesIngredients.recipe_id
    ->         FROM
    ->             RecipeIncludesIngredients
    ->         JOIN
    ->             UserLikeRecipe ON RecipeIncludesIngredients.recipe_id = UserLikeRecipe.recipe_id
    ->         GROUP BY
    ->             RecipeIncludesIngredients.recipe_id
    ->         HAVING
    ->             COUNT(UserLikeRecipe.user_id) > 10
    ->     )
    ->     AND Nutrition.calorie < 200
    ->     AND Nutrition.fat < 20
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> ORDER BY
    ->     average_rating DESC;
+-----------------------------------------+
```

```
-----------------------------------------+
| -> Sort: average_rating DESC  (actual time=0.954..0.957 rows=32 loops=1)
    -> Table scan on <temporary>  (actual time=0.924..0.929 rows=32 loops=1)
        -> Aggregate using temporary table  (actual time=0.923..0.923 rows=32 loops=1)
            -> Nested loop inner join  (cost=118.97 rows=34) (actual time=0.582..0.873 rows=42 loops=1)
                -> Nested loop inner join  (cost=96.68 rows=34) (actual time=0.571..0.722 rows=42 loops=1)
                    -> Nested loop inner join  (cost=88.98 rows=5) (actual time=0.566..0.692 rows=4 loops=1)
                        -> Nested loop inner join  (cost=49.78 rows=49) (actual time=0.555..0.649 rows=14 loops=1)
                            -> Filter: <in_optimizer>(UserRatesRecipe.recipe_id,UserRatesRecipe.recipe_id in (select #2))  (cost=5.90 rows=49) (actual time=0.543..0.606 rows=14 loop
s=1)
                                -> Covering index scan on UserRatesRecipe using rt_idx  (cost=5.90 rows=49) (actual time=0.026..0.069 rows=53 loops=1)
                                -> Select #2 (subquery in condition; run only once)
                                    -> Filter: ((UserRatesRecipe.recipe_id = `<materialized_subquery>`.recipe_id))  (cost=0.00..0.00 rows=0) (actual time=0.010..0.010 rows=0 loops=5
2)
                                        -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.010..0.010 rows=0 loops=52)
                                            -> Index lookup on <materialized_subquery> using <auto_distinct_key> (recipe_id=UserRatesRecipe.recipe_id)  (actual time=0.010..0.010 row
s=0 loops=52)
                                                -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=0.479..0.479 rows=12 loops=1)
                                                    -> Filter: (count(UserLikeRecipe.user_id) > 10)  (actual time=0.449..0.458 rows=12 loops=1)
                                                        -> Table scan on <temporary>  (actual time=0.447..0.453 rows=52 loops=1)
                                                            -> Aggregate using temporary table  (actual time=0.446..0.446 rows=52 loops=1)
                                                                -> Nested loop inner join  (cost=86.29 rows=348) (actual time=0.027..0.340 rows=411 loops=1)
                                                                    -> Covering index scan on UserLikeRecipe using recipe_id  (cost=6.60 rows=56) (actual time=0.013..0.022 rows=60 l
oops=1)
                                                                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0
.81 rows=6) (actual time=0.003..0.005 rows=7 loops=60)
                            -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.003..0.003 rows=1 l
oops=14)
                        -> Filter: ((Nutrition.calorie < 200.00) and (Nutrition.fat < 20.00))  (cost=0.70 rows=0.1) (actual time=0.003..0.003 rows=0 loops=14)
                            -> Single-row index lookup on Nutrition using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.70 rows=1) (actual time=0.002..0.002 rows=1 loops=14
)
                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.92 rows=6) (actual time=0.004..0.007 rows=10 l
oops=4)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.57 rows=1) (actual time=0.003..0.003 rows=1
loops=42)
 |
```

After creating index for fat, the cost is: 118.07+80.06+66.93+49.78+86.29=401.13

```
mysql> CREATE INDEX fat_idx on Nutrition(fat);
Query OK, 0 rows affected (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     AVG(UserRatesRecipe.rating) AS average_rating FROM      Ingredients JOIN      RecipeIncludesIngredients ON Ingred
ients.ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id JOIN      UserRatesRecipe ON Recipes.recip
e_id = UserRatesRecipe.recipe_id JOIN      Nutrition ON Recipes.recipe_id = Nutrition.recipe_id WHERE      Recipes.recipe_id IN (         SELECT             RecipeIncludesIngredien
ts.recipe_id         FROM             RecipeIncludesIngredients         JOIN             UserLikeRecipe ON RecipeIncludesIngredients.recipe_id = UserLikeRecipe.recipe_id         G
ROUP BY             RecipeIncludesIngredients.recipe_id         HAVING             COUNT(UserLikeRecipe.user_id) > 10     )     AND Nutrition.calorie < 200     AND Nutrition.fat <
 20 GROUP BY      Ingredients.ingredient_name ORDER BY      average_rating DESC;
+-----------------------------------------+
```

```
-----------------------------------------+
| -> Sort: average_rating DESC  (actual time=0.933..0.935 rows=32 loops=1)
    -> Table scan on <temporary>  (actual time=0.903..0.907 rows=32 loops=1)
        -> Aggregate using temporary table  (actual time=0.902..0.902 rows=32 loops=1)
            -> Nested loop inner join  (cost=118.07 rows=57) (actual time=0.527..0.853 rows=42 loops=1)
                -> Nested loop inner join  (cost=80.06 rows=57) (actual time=0.517..0.698 rows=42 loops=1)
                    -> Nested loop inner join  (cost=66.93 rows=9) (actual time=0.513..0.667 rows=4 loops=1)
                        -> Nested loop inner join  (cost=49.78 rows=49) (actual time=0.503..0.616 rows=14 loops=1)
                            -> Filter: <in_optimizer>(UserRatesRecipe.recipe_id,UserRatesRecipe.recipe_id in (select #2))  (cost=5.90 rows=49) (actual time=0.492..0.549 rows=14 loop
s=1)

                                -> Covering index scan on UserRatesRecipe using rt_idx  (cost=5.90 rows=49) (actual time=0.026..0.036 rows=53 loops=1)
                                -> Select #2 (subquery in condition; run only once)
                                    -> Filter: ((UserRatesRecipe.recipe_id = `<materialized_subquery>`.recipe_id))  (cost=0.00..0.00 rows=0) (actual time=0.009..0.009 rows=0 loops=5
2)

                                        -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.009..0.009 rows=0 loops=52)
                                            -> Index lookup on <materialized_subquery> using <auto_distinct_key> (recipe_id=UserRatesRecipe.recipe_id)  (actual time=0.009..0.009 row
s=0 loops=52)

                                                -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=0.447..0.447 rows=12 loops=1)
                                                    -> Filter: (count(UserLikeRecipe.user_id) > 10)  (actual time=0.432..0.442 rows=12 loops=1)
                                                        -> Table scan on <temporary>  (actual time=0.431..0.437 rows=52 loops=1)
                                                            -> Aggregate using temporary table  (actual time=0.430..0.430 rows=52 loops=1)
                                                                -> Nested loop inner join  (cost=86.29 rows=348) (actual time=0.026..0.327 rows=411 loops=1)
                                                                    -> Covering index scan on UserLikeRecipe using recipe_id  (cost=6.60 rows=56) (actual time=0.011..0.020 rows=60 l
oops=1)

                                                                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0
.81 rows=6) (actual time=0.003..0.004 rows=7 loops=60)
                            -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.005..0.005 rows=1 l
oops=14)
                        -> Filter: ((Nutrition.calorie < 200.00) and (Nutrition.fat < 20.00))  (cost=0.25 rows=0.2) (actual time=0.003..0.003 rows=0 loops=14)
                            -> Single-row index lookup on Nutrition using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=14
)
                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.87 rows=6) (actual time=0.004..0.007 rows=10 l
oops=4)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.56 rows=1) (actual time=0.003..0.004 rows=1
loops=42)
 |
+-----------------------------------------+
```

We can see the time does decrease.

3c: create index for ingredient_name:

Before creating index, the cost is: 140.12+102.11+88.98+49.78+86.29=467.28

```
Database changed
mysql> EXPLAIN ANALYZE SELECT
    ->     Ingredients.ingredient_name,
    ->     AVG(UserRatesRecipe.rating) AS average_rating
    -> FROM
    ->     Ingredients
    -> JOIN
    ->     RecipeIncludesIngredients ON Ingredients.ingredient_id = RecipeIncludesIngredients.ingredient_id
    -> JOIN
    ->     Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id
    -> JOIN
    ->     UserRatesRecipe ON Recipes.recipe_id = UserRatesRecipe.recipe_id
    -> JOIN
    ->     Nutrition ON Recipes.recipe_id = Nutrition.recipe_id
    -> WHERE
    ->     Recipes.recipe_id IN (
    ->         SELECT
    ->             RecipeIncludesIngredients.recipe_id
    ->         FROM
    ->             RecipeIncludesIngredients
    ->         JOIN
    ->             UserLikeRecipe ON RecipeIncludesIngredients.recipe_id = UserLikeRecipe.recipe_id
    ->         GROUP BY
    ->             RecipeIncludesIngredients.recipe_id
    ->         HAVING
    ->             COUNT(UserLikeRecipe.user_id) > 10
    ->     )
    ->     AND Nutrition.calorie < 200
    ->     AND Nutrition.fat < 20
    -> GROUP BY
    ->     Ingredients.ingredient_name
    -> ORDER BY
    ->     average_rating DESC;
+-----------------------------------------+
-----------------------------------------+
| -> Sort: average_rating DESC  (actual time=1.345..1.347 rows=32 loops=1)
    -> Table scan on <temporary>  (actual time=1.303..1.307 rows=32 loops=1)
        -> Aggregate using temporary table  (actual time=1.301..1.301 rows=32 loops=1)
            -> Nested loop inner join  (cost=140.12 rows=57) (actual time=0.785..1.225 rows=42 loops=1)
                -> Nested loop inner join  (cost=102.11 rows=57) (actual time=0.697..0.851 rows=42 loops=1)
                    -> Nested loop inner join  (cost=88.98 rows=9) (actual time=0.689..0.816 rows=4 loops=1)
                        -> Nested loop inner join  (cost=49.78 rows=49) (actual time=0.668..0.750 rows=14 loops=1)
                            -> Filter: <in_optimizer>(UserRatesRecipe.recipe_id,UserRatesRecipe.recipe_id in (select #2))  (cost=5.90 rows=49) (actual time=0.642..0.692 rows=14 loop
s=1)

                                -> Covering index scan on UserRatesRecipe using rt_idx  (cost=5.90 rows=49) (actual time=0.052..0.062 rows=53 loops=1)
                                -> Select #2 (subquery in condition; run only once)
                                    -> Filter: ((UserRatesRecipe.recipe_id = `<materialized_subquery>`.recipe_id))  (cost=0.00..0.00 rows=0) (actual time=0.011..0.011 rows=0 loops=5
2)

                                        -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.011..0.011 rows=0 loops=52)
                                            -> Index lookup on <materialized_subquery> using <auto_distinct_key> (recipe_id=UserRatesRecipe.recipe_id)  (actual time=0.011..0.011 row
s=0 loops=52)

                                                -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=0.555..0.555 rows=12 loops=1)
                                                    -> Filter: (count(UserLikeRecipe.user_id) > 10)  (actual time=0.536..0.546 rows=12 loops=1)
                                                        -> Table scan on <temporary>  (actual time=0.532..0.538 rows=52 loops=1)
                                                            -> Aggregate using temporary table  (actual time=0.531..0.531 rows=52 loops=1)
                                                                -> Nested loop inner join  (cost=86.29 rows=348) (actual time=0.045..0.410 rows=411 loops=1)
                                                                    -> Covering index scan on UserLikeRecipe using recipe_id  (cost=6.60 rows=56) (actual time=0.018..0.027 rows=60 l
oops=1)

                                                                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0
.81 rows=6) (actual time=0.004..0.006 rows=7 loops=60)
                            -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.004..0.004 rows=1 l
oops=14)
                        -> Filter: ((Nutrition.calorie < 200.00) and (Nutrition.fat < 20.00))  (cost=0.70 rows=0.2) (actual time=0.004..0.004 rows=0 loops=14)
                            -> Single-row index lookup on Nutrition using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.70 rows=1) (actual time=0.003..0.003 rows=1 loops=14
)
                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.87 rows=6) (actual time=0.005..0.007 rows=10 l
oops=4)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.56 rows=1) (actual time=0.009..0.009 rows=1
loops=42)
 |
+-----------------------------------------+
```

After creating index for ingredient_name, the cost is:
121.44+101.36+88.23+49.03+86.29=446.35

```
-----------------------------------------+
1 row in set (0.01 sec)

mysql> CREATE INDEX igd_nm5_idx on Ingredients(ingredient_name);
Query OK, 0 rows affected, 1 warning (0.39 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT      Ingredients.ingredient_name,     AVG(UserRatesRecipe.rating) AS average_rating FROM      Ingredients JOIN      RecipeIncludesIngredients ON Ingred
ients.ingredient_id = RecipeIncludesIngredients.ingredient_id JOIN      Recipes ON RecipeIncludesIngredients.recipe_id = Recipes.recipe_id JOIN      UserRatesRecipe ON Recipes.recip
e_id = UserRatesRecipe.recipe_id JOIN      Nutrition ON Recipes.recipe_id = Nutrition.recipe_id WHERE      Recipes.recipe_id IN (          SELECT              RecipeIncludesIngredien
ts.recipe_id          FROM              RecipeIncludesIngredients          JOIN              UserLikeRecipe ON RecipeIncludesIngredients.recipe_id = UserLikeRecipe.recipe_id          G
ROUP BY              RecipeIncludesIngredients.recipe_id          HAVING              COUNT(UserLikeRecipe.user_id) > 10      )      AND Nutrition.calorie < 200      AND Nutrition.fat <
 20 GROUP BY      Ingredients.ingredient_name ORDER BY      average_rating DESC;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------+
| -> Sort: average_rating DESC  (actual time=0.917..0.919 rows=32 loops=1)
    -> Table scan on <temporary>  (actual time=0.883..0.888 rows=32 loops=1)
        -> Aggregate using temporary table  (actual time=0.882..0.882 rows=32 loops=1)
            -> Nested loop inner join  (cost=121.44 rows=57) (actual time=0.542..0.833 rows=42 loops=1)
                -> Nested loop inner join  (cost=101.36 rows=57) (actual time=0.531..0.660 rows=42 loops=1)
                    -> Nested loop inner join  (cost=88.23 rows=9) (actual time=0.525..0.632 rows=4 loops=1)
                        -> Nested loop inner join  (cost=49.03 rows=49) (actual time=0.514..0.587 rows=14 loops=1)
                            -> Filter: <in_optimizer>(UserRatesRecipe.recipe_id,UserRatesRecipe.recipe_id in (select #2))  (cost=5.15 rows=49) (actual time=0.503..0.546 rows=14 loop
s=1)

                                -> Covering index scan on UserRatesRecipe using rt_idx  (cost=5.15 rows=49) (actual time=0.022..0.029 rows=53 loops=1)
                                -> Select #2 (subquery in condition; run only once)
                                    -> Filter: ((UserRatesRecipe.recipe_id = `<materialized_subquery>`.recipe_id))  (cost=0.00..0.00 rows=0) (actual time=0.010..0.010 rows=0 loops=5
2)

                                        -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.009..0.009 rows=0 loops=52)
                                            -> Index lookup on <materialized_subquery> using <auto_distinct_key> (recipe_id=UserRatesRecipe.recipe_id)  (actual time=0.009..0.009 row
s=0 loops=52)

                                                -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=0.461..0.461 rows=12 loops=1)
                                                    -> Filter: (count(UserLikeRecipe.user_id) > 10)  (actual time=0.444..0.453 rows=12 loops=1)
                                                        -> Table scan on <temporary>  (actual time=0.443..0.448 rows=52 loops=1)
                                                            -> Aggregate using temporary table  (actual time=0.442..0.442 rows=52 loops=1)
                                                                -> Nested loop inner join  (cost=86.29 rows=348) (actual time=0.028..0.338 rows=411 loops=1)
                                                                    -> Covering index scan on UserLikeRecipe using recipe_id  (cost=6.60 rows=56) (actual time=0.014..0.023 rows=60 l
oops=1)

                                                                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserLikeRecipe.recipe_id)  (cost=0
.81 rows=6) (actual time=0.003..0.005 rows=7 loops=60)
                            -> Single-row covering index lookup on Recipes using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.80 rows=1) (actual time=0.003..0.003 rows=1 l
oops=14)
                        -> Filter: ((Nutrition.calorie < 200.00) and (Nutrition.fat < 20.00))  (cost=0.70 rows=0.2) (actual time=0.003..0.003 rows=0 loops=14)
                            -> Single-row index lookup on Nutrition using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.70 rows=1) (actual time=0.003..0.003 rows=1 loops=14
)
                    -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.87 rows=6) (actual time=0.004..0.006 rows=10 l
oops=4)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=RecipeIncludesIngredients.ingredient_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1
loops=42)
 |
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

We can see that the time does decrease.

---

4. Our fourth advanced query:

```
SELECT DISTINCT
    Ingredients.ingredient_name
FROM
    Ingredients
JOIN (
    SELECT
        RecipeIncludesIngredients.ingredient_id
    FROM
        RecipeIncludesIngredients
    JOIN
        UserRatesRecipe ON RecipeIncludesIngredients.recipe_id =
UserRatesRecipe.recipe_id
    WHERE
        UserRatesRecipe.comment NOT LIKE '%bad%'
    GROUP BY
        RecipeIncludesIngredients.ingredient_id
    HAVING
        AVG(UserRatesRecipe.rating) > 1
) AS HighRatedIngredients ON Ingredients.ingredient_id =
HighRatedIngredients.ingredient_id
JOIN (
    SELECT
        MarketsSellsIngredients.ingredient_id
    FROM
        MarketsSellsIngredients
```

JOIN
    Markets ON MarketsSellsIngredients.market_id = Markets.market_id
WHERE
    Markets.market_location LIKE '%GreenVille%'
) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id;

4a: create index for rating:

Before creating index, the cost is: 279.22+79.65+67.13 = 426

```
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> EXPLAIN ANALYZE SELECT DISTINCT
    ->     Ingredients.ingredient_name
    -> FROM
    ->     Ingredients
    -> JOIN (
    ->     SELECT
    ->         RecipeIncludesIngredients.ingredient_id
    ->     FROM
    ->         RecipeIncludesIngredients
    ->     JOIN
    ->         UserRatesRecipe ON RecipeIncludesIngredients.recipe_id = UserRatesRecipe.recipe_id
    ->     WHERE
    ->         UserRatesRecipe.comment NOT LIKE '%bad%'
    ->     GROUP BY
    ->         RecipeIncludesIngredients.ingredient_id
    ->     HAVING
    ->         AVG(UserRatesRecipe.rating) > 1
    -> ) AS HighRatedIngredients ON Ingredients.ingredient_id = HighRatedIngredients.ingredient_id
    -> JOIN (
    ->     SELECT
    ->         MarketsSellsIngredients.ingredient_id
    ->     FROM
    ->         MarketsSellsIngredients
    ->     JOIN
    ->         Markets ON MarketsSellsIngredients.market_id = Markets.market_id
    ->     WHERE
    ->         Markets.market_location LIKE '%GreenVille%'
    -> ) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id;
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------------------
-----------------------------------------+
| -> Table scan on <temporary>  (cost=281.72..281.72 rows=0) (actual time=1.686..1.691 rows=30 loops=1)
    -> Temporary table with deduplication  (cost=279.22..279.22 rows=0) (actual time=1.685..1.685 rows=30 loops=1)
        -> Nested loop inner join  (cost=279.22 rows=0) (actual time=0.748..1.637 rows=30 loops=1)
            -> Nested loop inner join  (cost=79.65 rows=0) (actual time=0.666..1.172 rows=145 loops=1)
                -> Inner hash join (no condition)  (cost=6.53 rows=0) (actual time=0.653..0.702 rows=145 loops=1)
                    -> Table scan on HighRatedIngredients  (cost=2.50..2.50 rows=0) (actual time=0.518..0.539 rows=145 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=0.517..0.517 rows=145 loops=1)
                            -> Filter: (avg(UserRatesRecipe.rating) > 1)  (actual time=0.448..0.498 rows=145 loops=1)
                                -> Table scan on <temporary>  (actual time=0.441..0.458 rows=159 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.440..0.440 rows=159 loops=1)
                                        -> Nested loop inner join  (cost=67.13 rows=271) (actual time=0.055..0.326 rows=299 loops=1)
                                            -> Filter: (not((UserRatesRecipe.`comment` like '%bad%')))  (cost=5.15 rows=44) (actual time=0.032..0.051 rows=38 loops=1)
                                                -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.028..0.038 rows=53 loops=1)
                                            -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.82 rows=6) (actual tim
e=0.005..0.007 rows=8 loops=38)
                    -> Hash
                        -> Filter: (Markets.market_location like '%GreenVille%')  (cost=0.65 rows=1) (actual time=0.052..0.109 rows=1 loops=1)
                            -> Table scan on Markets  (cost=0.65 rows=4) (actual time=0.043..0.100 rows=4 loops=1)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0.27 rows=1) (actual time=0.003..0.003 rows=1 loops
=145)
            -> Limit: 1 row(s)  (cost=0.74 rows=1) (actual time=0.003..0.003 rows=0 loops=145)
                -> Single-row covering index lookup on MarketsSellsIngredients using PRIMARY (market_id=Markets.market_id, ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0
.74 rows=1) (actual time=0.003..0.003 rows=0 loops=145)
 |
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
```

After creating index for rating, the cost is: 279.22+79.65+67.13 = 426

```
1 row in set (0.01 sec)

mysql> CREATE INDEX rt_66_idx on UserRatesRecipe(rating);
Query OK, 0 rows affected, 1 warning (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT DISTINCT     Ingredients.ingredient_name FROM     Ingredients JOIN (     SELECT         RecipeIncludesIngredients.ingredient_id     FROM         Rec
ipeIncludesIngredients     JOIN         UserRatesRecipe ON RecipeIncludesIngredients.recipe_id = UserRatesRecipe.recipe_id     WHERE         UserRatesRecipe.comment NOT LIKE '%bad
%'     GROUP BY         RecipeIncludesIngredients.ingredient_id     HAVING         AVG(UserRatesRecipe.rating) > 1 ) AS HighRatedIngredients ON Ingredients.ingredient_id = HighRat
edIngredients.ingredient_id JOIN (     SELECT         MarketsSellsIngredients.ingredient_id     FROM         MarketsSellsIngredients     JOIN         Markets ON MarketsSellsIngre
dients.market_id = Markets.market_id     WHERE         Markets.market_location LIKE '%GreenVille%' ) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id
;
+------------------------------------------------------------------------------------------------
```

```
-----------------------------------------+
| -> Table scan on <temporary>  (cost=281.72..281.72 rows=0) (actual time=1.858..1.862 rows=30 loops=1)
    -> Temporary table with deduplication  (cost=279.22..279.22 rows=0) (actual time=1.856..1.856 rows=30 loops=1)
        -> Nested loop inner join  (cost=279.22 rows=0) (actual time=0.637..1.809 rows=30 loops=1)
            -> Nested loop inner join  (cost=79.65 rows=0) (actual time=0.596..1.212 rows=145 loops=1)
                -> Inner hash join (no condition)  (cost=6.53 rows=0) (actual time=0.581..0.640 rows=145 loops=1)
                    -> Table scan on HighRatedIngredients  (cost=2.50..2.50 rows=0) (actual time=0.528..0.553 rows=145 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=0.527..0.527 rows=145 loops=1)
                            -> Filter: (avg(UserRatesRecipe.rating) > 1)  (actual time=0.419..0.501 rows=145 loops=1)
                                -> Table scan on <temporary>  (actual time=0.413..0.438 rows=159 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.412..0.412 rows=159 loops=1)
                                        -> Nested loop inner join  (cost=67.13 rows=271) (actual time=0.045..0.293 rows=299 loops=1)
                                            -> Filter: (not((UserRatesRecipe.`comment` like '%bad%')))  (cost=5.15 rows=44) (actual time=0.026..0.046 rows=38 loops=1)
                                                -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.024..0.034 rows=53 loops=1)
                                            -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.82 rows=6) (actual tim
e=0.004..0.006 rows=8 loops=38)
                    -> Hash
                        -> Filter: (Markets.market_location like '%GreenVille%')  (cost=0.65 rows=1) (actual time=0.035..0.038 rows=1 loops=1)
                            -> Table scan on Markets  (cost=0.65 rows=4) (actual time=0.029..0.032 rows=4 loops=1)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0.27 rows=1) (actual time=0.004..0.004 rows=1 loops
=145)
            -> Limit: 1 row(s)  (cost=0.74 rows=1) (actual time=0.004..0.004 rows=0 loops=145)
                -> Single-row covering index lookup on MarketsSellsIngredients using PRIMARY (market_id=Markets.market_id, ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0
.74 rows=1) (actual time=0.004..0.004 rows=0 loops=145)
 |
+------------------------------------------------------------------------------------------------
```

4b: create index for comment:

Before creating index, the cost is: 547.28+277.28+76.98=901.54

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

Database changed
mysql> EXPLAIN ANALYZE SELECT DISTINCT
    ->     Ingredients.ingredient_name
    -> FROM
    ->     Ingredients
    -> JOIN (
    ->     SELECT
    ->         RecipeIncludesIngredients.ingredient_id
    ->     FROM
    ->         RecipeIncludesIngredients
    ->     JOIN
    ->         UserRatesRecipe ON RecipeIncludesIngredients.recipe_id = UserRatesRecipe.recipe_id
    ->     WHERE
    ->         UserRatesRecipe.comment NOT LIKE '%bad%'
    ->     GROUP BY
    ->         RecipeIncludesIngredients.ingredient_id
    ->     HAVING
    ->         AVG(UserRatesRecipe.rating) > 1
    -> ) AS HighRatedIngredients ON Ingredients.ingredient_id = HighRatedIngredients.ingredient_id
    -> JOIN (
    ->     SELECT
    ->         MarketsSellsIngredients.ingredient_id
    ->     FROM
    ->         MarketsSellsIngredients
    ->     JOIN
    ->         Markets ON MarketsSellsIngredients.market_id = Markets.market_id
    ->     WHERE
    ->         Markets.market_location LIKE '%GreenVille%'
    -> ) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id;
+----------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (cost=549.78..549.78 rows=0) (actual time=321.006..321.011 rows=30 loops=1)
    -> Temporary table with deduplication  (cost=547.28..547.28 rows=0) (actual time=321.004..321.004 rows=30 loops=1)
        -> Nested loop inner join  (cost=547.28 rows=0) (actual time=271.270..320.811 rows=30 loops=1)
            -> Nested loop inner join  (cost=277.28 rows=0) (actual time=233.364..279.077 rows=145 loops=1)
                -> Inner hash join (no condition)  (cost=7.28 rows=0) (actual time=183.695..183.877 rows=145 loops=1)
                    -> Table scan on HighRatedIngredients  (cost=2.50..2.50 rows=0) (actual time=160.356..160.441 rows=145 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=160.355..160.355 rows=145 loops=1)
                            -> Filter: (avg(UserRatesRecipe.rating) > 1)  (actual time=160.273..160.325 rows=145 loops=1)
                                -> Table scan on <temporary>  (actual time=160.265..160.282 rows=159 loops=1)
                                    -> Aggregate using temporary table  (actual time=160.263..160.263 rows=159 loops=1)
                                        -> Nested loop inner join  (cost=76.98 rows=271) (actual time=101.018..160.063 rows=299 loops=1)
                                            -> Filter: (not((UserRatesRecipe.`comment` like '%bad%')))  (cost=5.90 rows=44) (actual time=41.604..41.681 rows=38 loops=1)
                                                -> Table scan on UserRatesRecipe  (cost=5.90 rows=49) (actual time=41.594..41.622 rows=53 loops=1)
                                            -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=1.02 rows=6) (actual tim
e=3.112..3.114 rows=8 loops=38)
                    -> Hash
                        -> Filter: (Markets.market_location like '%GreenVille%')  (cost=1.40 rows=1) (actual time=23.290..23.313 rows=1 loops=1)
                            -> Table scan on Markets  (cost=1.40 rows=4) (actual time=23.276..23.300 rows=4 loops=1)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=HighRatedIngredients.ingredient_id)  (cost=1.00 rows=1) (actual time=0.656..0.656 rows=1 loops
=145)
            -> Limit: 1 row(s)  (cost=1.00 rows=1) (actual time=0.287..0.288 rows=0 loops=145)
                -> Single-row covering index lookup on MarketsSellsIngredients using PRIMARY (market_id=Markets.market_id, ingredient_id=HighRatedIngredients.ingredient_id)  (cost=1
.00 rows=1) (actual time=0.287..0.287 rows=0 loops=145)
 |
+----------------------------------------------------------------------------------------------------------------------------------
```

- -

After creating index for comment, the cost is: 296.09+96.53+43.23=435.85

```
mysql> CREATE INDEX cmt2_idx on UserRatesRecipe(comment);
Query OK, 0 rows affected, 1 warning (0.20 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> EXPLAIN ANALYZE SELECT DISTINCT     Ingredients.ingredient_name FROM     Ingredients JOIN (     SELECT         RecipeIncludesIngredients.ingredient_id     FROM         Rec
ipeIncludesIngredients     JOIN         UserRatesRecipe ON RecipeIncludesIngredients.recipe_id = UserRatesRecipe.recipe_id     WHERE         UserRatesRecipe.comment NOT LIKE '%bad
%'     GROUP BY         RecipeIncludesIngredients.ingredient_id     HAVING         AVG(UserRatesRecipe.rating) > 1 ) AS HighRatedIngredients ON Ingredients.ingredient_id = HighRat
edIngredients.ingredient_id JOIN (     SELECT         MarketsSellsIngredients.ingredient_id     FROM         MarketsSellsIngredients     JOIN         Markets ON MarketsSellsIngre
dients.market_id = Markets.market_id     WHERE         Markets.market_location LIKE '%GreenVille%' ) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id
;
+----------------------------------------------------------------------------------------------------------------------------------
```

œ

We can see that the time decreases

4c: create index for market_location:

Before creating index, the cost is: 296.09+96.53+68.27=460.89

```
9 rows in set (0.01 sec)

mysql> use 411_Smaller;
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> EXPLAIN ANALYZE SELECT DISTINCT
    ->     Ingredients.ingredient_name
    -> FROM
    ->     Ingredients
    -> JOIN (
    ->     SELECT
    ->         RecipeIncludesIngredients.ingredient_id
    ->     FROM
    ->         RecipeIncludesIngredients
    ->     JOIN
    ->         UserRatesRecipe ON RecipeIncludesIngredients.recipe_id = UserRatesRecipe.recipe_id
    ->     WHERE
    ->         UserRatesRecipe.comment NOT LIKE '%bad%'
    ->     GROUP BY
    ->         RecipeIncludesIngredients.ingredient_id
    ->     HAVING
    ->         AVG(UserRatesRecipe.rating) > 1
    -> ) AS HighRatedIngredients ON Ingredients.ingredient_id = HighRatedIngredients.ingredient_id
    -> JOIN (
    ->     SELECT
    ->         MarketsSellsIngredients.ingredient_id
    ->     FROM
    ->         MarketsSellsIngredients
    ->     JOIN
    ->         Markets ON MarketsSellsIngredients.market_id = Markets.market_id
    ->     WHERE
    ->         Markets.market_location LIKE '%GreenVille%'
    -> ) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id;
+----------------------------------------------------------------------------------------------------------------------------------
```

```
---------------------------------------+
| -> Table scan on <temporary>  (cost=298.59..298.59 rows=0) (actual time=1.715..1.719 rows=30 loops=1)
    -> Temporary table with deduplication  (cost=296.09..296.09 rows=0) (actual time=1.714..1.714 rows=30 loops=1)
        -> Nested loop inner join  (cost=296.09 rows=0) (actual time=0.751..1.650 rows=30 loops=1)
            -> Nested loop inner join  (cost=96.53 rows=0) (actual time=0.717..1.238 rows=145 loops=1)
                -> Inner hash join (no condition)  (cost=6.53 rows=0) (actual time=0.686..0.728 rows=145 loops=1)
                    -> Table scan on HighRatedIngredients  (cost=2.50..2.50 rows=0) (actual time=0.619..0.638 rows=145 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=0.619..0.619 rows=145 loops=1)
                            -> Filter: (avg(UserRatesRecipe.rating) > 1)  (actual time=0.544..0.602 rows=145 loops=1)
                                -> Table scan on <temporary>  (actual time=0.536..0.553 rows=159 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.535..0.535 rows=159 loops=1)
                                        -> Nested loop inner join  (cost=68.27 rows=271) (actual time=0.050..0.394 rows=299 loops=1)
                                            -> Filter: (not((UserRatesRecipe.`comment` like '%bad%')))  (cost=5.15 rows=44) (actual time=0.027..0.052 rows=38 loops=1)
                                                -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.025..0.037 rows=53 loops=1)
                                            -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.84 rows=6) (actual tim
e=0.006..0.008 rows=8 loops=38)
                    -> Hash
                        -> Filter: (Markets.market_location like '%GreenVille%')  (cost=0.65 rows=1) (actual time=0.049..0.053 rows=1 loops=1)
                            -> Table scan on Markets  (cost=0.65 rows=4) (actual time=0.042..0.046 rows=4 loops=1)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0.33 rows=1) (actual time=0.003..0.003 rows=1 loops
=145)
            -> Limit: 1 row(s)  (cost=0.74 rows=1) (actual time=0.003..0.003 rows=0 loops=145)
                -> Single-row covering index lookup on MarketsSellsIngredients using PRIMARY (market_id=Markets.market_id, ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0
.74 rows=1) (actual time=0.003..0.003 rows=0 loops=145)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

After creating index for market_location, the cost is: 296.09+96.53+68.27=460.89

```
mysql> CREATE INDEX mk_lc_idx on Markets(market_location);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT DISTINCT     Ingredients.ingredient_name FROM      Ingredients JOIN (     SELECT          RecipeIncludesIngredients.ingredient_id     FROM          Rec
ipeIncludesIngredients     JOIN          UserRatesRecipe ON RecipeIncludesIngredients.recipe_id = UserRatesRecipe.recipe_id     WHERE          UserRatesRecipe.comment NOT LIKE '%bad
%'     GROUP BY          RecipeIncludesIngredients.ingredient_id     HAVING          AVG(UserRatesRecipe.rating) > 1 ) AS HighRatedIngredients ON Ingredients.ingredient_id = HighRat
edIngredients.ingredient_id JOIN (     SELECT          MarketsSellsIngredients.ingredient_id     FROM          MarketsSellsIngredients     JOIN          Markets ON MarketsSellsIngre
dients.market_id = Markets.market_id     WHERE          Markets.market_location LIKE '%GreenVille%' ) AS DowntownMarkets ON Ingredients.ingredient_id = DowntownMarkets.ingredient_id
;
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------+
| -> Table scan on <temporary>  (cost=298.59..298.59 rows=0) (actual time=1.941..1.945 rows=30 loops=1)
    -> Temporary table with deduplication  (cost=296.09..296.09 rows=0) (actual time=1.939..1.939 rows=30 loops=1)
        -> Nested loop inner join  (cost=296.09 rows=0) (actual time=0.620..1.901 rows=30 loops=1)
            -> Nested loop inner join  (cost=96.53 rows=0) (actual time=0.587..1.483 rows=145 loops=1)
                -> Inner hash join (no condition)  (cost=6.53 rows=0) (actual time=0.572..0.620 rows=145 loops=1)
                    -> Table scan on HighRatedIngredients  (cost=2.50..2.50 rows=0) (actual time=0.541..0.563 rows=145 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=0.540..0.540 rows=145 loops=1)
                            -> Filter: (avg(UserRatesRecipe.rating) > 1)  (actual time=0.476..0.525 rows=145 loops=1)
                                -> Table scan on <temporary>  (actual time=0.470..0.486 rows=159 loops=1)
                                    -> Aggregate using temporary table  (actual time=0.470..0.470 rows=159 loops=1)
                                        -> Nested loop inner join  (cost=68.27 rows=271) (actual time=0.044..0.338 rows=299 loops=1)
                                            -> Filter: (not((UserRatesRecipe.`comment` like '%bad%')))  (cost=5.15 rows=44) (actual time=0.028..0.051 rows=38 loops=1)
                                                -> Table scan on UserRatesRecipe  (cost=5.15 rows=49) (actual time=0.026..0.038 rows=53 loops=1)
                                            -> Covering index lookup on RecipeIncludesIngredients using PRIMARY (recipe_id=UserRatesRecipe.recipe_id)  (cost=0.84 rows=6) (actual tim
e=0.005..0.007 rows=8 loops=38)
                    -> Hash
                        -> Filter: (Markets.market_location like '%GreenVille%')  (cost=0.65 rows=1) (actual time=0.019..0.021 rows=1 loops=1)
                            -> Covering index scan on Markets using mk_lc_idx  (cost=0.65 rows=4) (actual time=0.013..0.015 rows=4 loops=1)
                -> Single-row index lookup on Ingredients using PRIMARY (ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0.33 rows=1) (actual time=0.006..0.006 rows=1 loops
=145)
            -> Limit: 1 row(s)  (cost=0.74 rows=1) (actual time=0.003..0.003 rows=0 loops=145)
                -> Single-row covering index lookup on MarketsSellsIngredients using PRIMARY (market_id=Markets.market_id, ingredient_id=HighRatedIngredients.ingredient_id)  (cost=0
.74 rows=1) (actual time=0.003..0.003 rows=0 loops=145)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

We can see that the cost does not decrease.

Explanation:

As it is shown above, creating an index in most cases is able to lower the cost. In very few cases, the cost remains the same.

Without the index, the database selection needs to scan through all the information inside the table, which is very slow. With the index, the database is able to just look for the corresponding information directly based on the search key values, which does not need to scan through the entire table.