Pantry Pal Database Definition

Team 101: nehasv2, alizain2, pawudu2, manasiv2

Tables

```
mysql> show tables;
+------------------+
| Tables_in_Pantry |
+------------------+
| Fridge           |
| Ingredients      |
| Interactions     |
| Likes            |
| Recipes          |
| Reviews          |
| Used_Ingredients |
| Users            |
+------------------+
8 rows in set (0.00 sec)
```

Data Definition Language Commands

1. Recipes

```
CREATE TABLE Recipes (
       Name VARCHAR(150),
       Id INT PRIMARY KEY,
       Instructions VARCHAR(5000),
       Cal_Level INT
);
```

```
mysql> SELECT Count(*) FROM Recipes;
+----------+
| Count(*) |
+----------+
|    11943 |
+----------+
1 row in set (0.28 sec)
```

---

2. Ingredients

```
CREATE TABLE Ingredients (
       Name VARCHAR(100),
       Ing_Id INT PRIMARY KEY,
       Cal_Count INT
);
```

```
mysql> SELECT Count(*) FROM Ingredients;
+----------+
| Count(*) |
+----------+
|     8023 |
+----------+
1 row in set (0.11 sec)
```

---

3. Users

```
CREATE TABLE Users (
       Name VARCHAR(150),
       User_Id INT PRIMARY KEY
);
```

```
mysql> SELECT Count(*) FROM Users;
+----------+
| Count(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.10 sec)
```

---

4. Reviews

```
CREATE TABLE Reviews (
        Review_Id INT PRIMARY KEY,
        Rating INT,
        Review VARCHAR(5000)
);
```

```
mysql> SELECT Count(*) FROM Reviews;
+----------+
| Count(*) |
+----------+
|    20001 |
+----------+
1 row in set (0.58 sec)
```

---

5. Interactions

```
CREATE TABLE Interactions (
        Recipe_Id INT,
        Review_Id INT,
        FOREIGN KEY (Recipe_Id) REFERENCES
Recipes(Id) ON UPDATE CASCADE ON DELETE
CASCADE,
        FOREIGN KEY (Review_Id) REFERENCES
Reviews(Review_Id) ON UPDATE CASCADE ON
DELETE CASCADE,
        PRIMARY KEY (Recipe_Id, Review_Id)
);
```

```
mysql> SELECT Count(*) FROM Interactions;
+----------+
| Count(*) |
+----------+
|     1528 |
+----------+
1 row in set (0.04 sec)
```

---

6. Used_Ingredients

```
CREATE TABLE Used_Ingredients (
        Recipe_Id INT,
        Ing_Id INT,
        FOREIGN KEY (Recipe_Id) REFERENCES
Recipes(Id) ON UPDATE CASCADE ON DELETE
CASCADE,
        FOREIGN KEY (Ing_Id) REFERENCES
Ingredients(Ing_Id) ON UPDATE CASCADE ON
DELETE CASCADE,
        PRIMARY KEY (Recipe_Id, Ing_Id)
);
```

```
mysql> SELECT Count(*) FROM Used_Ingredients;
+----------+
| Count(*) |
+----------+
|   105668 |
+----------+
1 row in set (0.49 sec)
```

---

7. Fridge

```
CREATE TABLE Fridge (
	User_Id INT,
	Ing_Id INT,
	FOREIGN KEY (User_Id) REFERENCES
Users(User_Id) ON UPDATE CASCADE ON
DELETE CASCADE,
	FOREIGN KEY (Ing_Id) REFERENCES
Ingredients(Ing_Id) ON UPDATE CASCADE ON
DELETE CASCADE,
	PRIMARY KEY (User_Id, Ing_Id)
);
```

```
mysql> SELECT Count(*) FROM Fridge;
+----------+
| Count(*) |
+----------+
|     7996 |
+----------+
1 row in set (0.03 sec)
```

---

8. Likes

```
CREATE TABLE Likes (
	User_Id INT,
	Recipe_Id INT,
FOREIGN KEY (User_Id) REFERENCES
Users(User_Id) ON UPDATE CASCADE ON
DELETE CASCADE,
	FOREIGN KEY (Recipe_Id) REFERENCES
Recipes(Id) ON UPDATE CASCADE ON DELETE
CASCADE,
	PRIMARY KEY (User_Id, Recipe_Id)
);
```

```
mysql> SELECT Count(*) FROM Likes;
+----------+
| Count(*) |
+----------+
|     7997 |
+----------+
1 row in set (0.01 sec)
```

Advanced Queries:

1. **Popular Recipes**: Filter Recipes to show the Id, Name and Instructions for recipes that have an average rating of at least 4.

```
SELECT r.Id, r.Name, r.Instruction
FROM Recipes r NATURAL JOIN
	(SELECT Recipe_Id as Id, AVG(Rating) AS AvgRating
	FROM Interactions NATURAL JOIN Reviews
	GROUP BY Recipe_Id) as Inter
WHERE AvgRating >= 4;
```

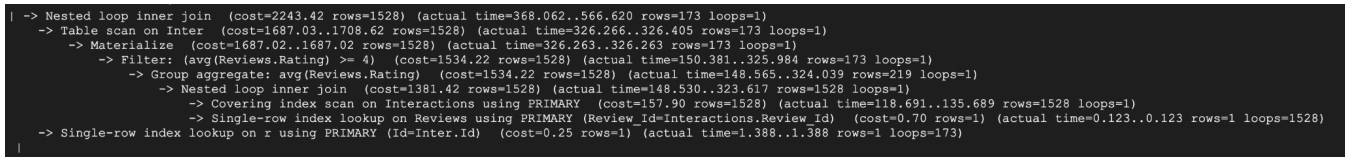
```
mysql> SELECT r.Id, r.Name, r.Instruction
    -> FROM Recipes r NATURAL JOIN
    -> (SELECT Recipe_Id as Id, AVG(Rating) AS AvgRating
    -> FROM Interactions NATURAL JOIN Reviews
    -> GROUP BY Recipe_Id) as Inter
    -> WHERE AvgRating >= 4
    -> LIMIT 15;
+-------+-----------------------------------------------+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Id    | Name                                          | Instruction

                                                                                                                                                                      |
+-------+-----------------------------------------------+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|   355 | apple crisp                                   | ['fill a 6 x 10 baking pan with sliced apples', 'mix the flour , salt , sugar , egg , baking powder together with two knives or pastry blender until crumbly', 'spread over the apples', 'pour 1 / 2
 cup melted butter over top and sprinkle with cinnamon', 'bake at 350 degrees fahrenheit for 30 minutes or until apples are done and topping is crisp']
                                                                                                                                                                      |
|   360 | baked zucchini frittatas                      | ['heat oven to 400f', 'grease two 10-oz', 'custard cups or individual baking dishes', 'in 8-inch skillet over medium low heat , saut zucchini , onion , red pepper and garlic in margarine until ten
der', 'let cool slightly', 'in medium bowl , beat eggs , half-and-half and seasonings until combined', 'stir in zucchini mixture', 'pour into prepared custard cups or baking dishes', 'sprinkle with cheeses', 'bake , uncovered , at 400f for 14 to 17 minu
tes or until top is puffy and light golden brown', 'let stand 5 minutes before serving']                                                                              |
|  2561 | ambrosia                                      | ['drain fruit well', 'mix all ingredients together', 'refrigerate for 24 hours', 'be sure fruit is well drained or ambrosia will be mushy']

                                                                                                                                                                      |
|  3689 | amish pot roast                               | ['preheat oven to 300 degrees', 'do not pound or flour the meat', 'heat oil in a heavy skillet over high heat , then sear meat on both sides', 'meanwhile , in a large roasting pan , combine soy sa
uce , coffee , bay leaves , garlic , oregano and one of the sliced onions', 'transfer the browned meat to the roasting pan', 'top with the second sliced onion', 'cover and bake for 3 1 / 2 to 4 hours , basting every hour with pan juices', 'if the liquid
 begins to boil away , add another cup of coffee and a splash of soy sauce', 'you may need to repeat this procedure', 'there should be quite a bit of liquid', 'cut the meat in thin slices and serve with pan juices']    |
|  4527 | asian grilled chicken                         | ['combine all ingredients and mix well', 'cover and chill for 2-4 hours', 'remove chicken from marinade , drain well', 'grill 5 minutes per side or until cooked or , bake at 375 for 30 minutes', '
remove to a serving platter', 'heat marinade until boiling', 'simmer 3 minutes', 'serve over grilled chicken', 'garnish with cilantro leaves , if desired']
                                                                                                                                                                      |
|  4779 | apple scones                                  | ['preheat oven to 425 degrees', 'measure flour , sugar , b', 'powder , soda , and salt into large bowl', 'cut in butter until crumbly', 'add apple and milk', 'stir to form soft dough', 'turn out o
nto lightly floured board', 'knead gently 8 to 10 times', 'pat into two 6-inch circles', 'place on greased baking sheet', 'brush tops with milk', 'sprinkle with sugar and cinnamon', 'score each top into six pie-shaped wedges', 'bake for 15 minutes or un
til browned and risen', 'serve with warm butter and marmalade', 'optional: add 1 / 2 cup currants to batter']                                                         |
|  6809 | asparagus soup                                | ['wash and cut up the celery and put it in a large pot with the water and the canned beans , including the packing liquid from the beans', 'heat to a simmer', 'wash the asparagus , and snap off bo
th the tough ends and the', 'grind the spices and add them to the soup', 'add the asparagus stalks and ends to the soup and simmer until quite tender', 'about 10 minutes', 'reserve the', 'remove the tough stem ends from the soup and discard', 'puree the
 soup thoroughly in a blender', 'the soup may be made ahead up to this point', 'it should keep in the fridge for 24 hours', 'to serve , add the cream and reheat the soup gently', 'steam the reserved', 'serve the soup garnished with the steamed'] |
|  7253 | apple almond pie                              | ['prepare pie crust for filled one-crust pie using 9-inch pan', 'flute edge', 'heat oven to 375f', 'reserve 2 t sugar', 'in medium bowl , combine remaining sugar and corn starch', 'add eggs , beat
ing well to combine', 'stir in butter , corn syrup and almond extract', 'mix in apples and almonds', 'pour into pie crust-lined pan', 'if desired , garnish with apple slices overlapped in a circle around edge of pie', 'sprinkle center with almonds', 'sp
rinkle reserved sugar over top', 'bake at 375f for 50 minutes or until center of pie is set', 'cool completely on wire rack', 'store in refrigerator']               |
|  7397 | banana bread 3                                | ['mash bananas and mix with other ingredients', 'pour batter into prepared 9 x 5 inch loaf pan', 'bake at 350f for 50 minutes']

                                                                                                                                                                      |
|  7414 | beef  n  beer                                 | ['salt & pepper meat and brown lightly in heavy skillet', 'add onions , pepper and mushrooms and brown lightly', 'pour beer over all , stir in catsup and mustard', 'cover tightly and simmer slowly
 for 2 to 2 1 / 2 hours or until meat is tender']
                                                                                                                                                                      |
|  7608 | baked beans                                   | ['drain pork and beans and place into oven-proof bowl', 'chop onion and apples and add to beans', 'also add bar-b-que sauce , brown sugar and rasins', 'mix together and lay bacon strips on top', '
place in covered grill and bake at 350 degrees for 1 to 1 1 / 2 hours']
                                                                                                                                                                      |
| 11040 | asparagus  oven roasted                       | ['heat an oven proof pan in a hot oven for 5 minutes', 'pour in the oil& italian seasoning', 'coat the whole pan', 'roll the asparagus in the oil to coat all sides', 'bake in 400f degrees oven for
 5-7 min do not over cook !', 'the asparagus should just be heated through but still very crisp', 'remove from oven , drizzle with the vinegar', 'add salt& pepper', 'sprinkle with parmesan', 'serve immediately or serve at room temperature']
                                                                                                                                                                      |
| 11761 | bamia                                         | ['heat oil in a medium sized heavy saucepan over medium heat', 'saut onions for 2 minutes', 'add the garlic and saut for 1 minute', 'stir in all the remaining ingredients except the okra', 'bring
the mixture to a hard simmer and cook for 5 minutes-stirring often', 'trim the okra and slice into 1 / 2 inch rounds and stir in', 'simmer for 10 minutes-serve hot over rice']
                                                                                                                                                                      |
| 12100 | amazing stuffed  shrooms                      | ['wash mushrooms and remove stems', 'chop stems', 'mix the bread crumbs , cream cheese , and enough milk to just moisten the mixture', 'stirring constantly , place over medium low heat on the stov
e until cheese has melted', 'if needed , add more milk', 'stuff mixture into mushroom caps- sprinkle with parmesan cheese', 'place mushrooms on foil lined pan and cook at 300 until browned , approximately 10 minutes', 'if you wish , you may add two crum
bled strips of bacon to the mixture before you stuff it into the mushrooms', 'serve hot']                                                                              |
| 13041 | balah isham  deep fried  cream puffs  pastry | ['in a large saucepan , heat water and butter until butter melts', 'bring to a boil', 'remove from the heat source , add flour and baking powder , mix well , then return to the heat and mix', 'for
m dough into a ball in the center', 'remove from heat and add eggs , one at a time , mixing well between each addition', 'add vanilla with 5th egg', 'spoon dough into a pastry bag with a medium star', 'pour sugar syrup overbalah-isham while still warm',
 'sugar syrup: put water , sugar and lemon in a small saucepan , bring to a boil , simmer over medium heat for 20-30 minutes', 'should thicken when it cools']       |
+-------+-----------------------------------------------+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------+
15 rows in set (0.01 sec)
```

Here is the query without displaying the Instructions (for better formatting):

```
mysql> SELECT r.Id, r.Name
    -> FROM Recipes r NATURAL JOIN
    -> (SELECT Recipe_Id as Id, AVG(Rating) AS AvgRating
    -> FROM Interactions NATURAL JOIN Reviews
    -> GROUP BY Recipe_Id) as Inter
    -> WHERE AvgRating >= 4
    -> LIMIT 15;
+-------+-----------------------------------------------+
| Id    | Name                                          |
+-------+-----------------------------------------------+
|   355 | apple crisp                                   |
|   360 | baked zucchini frittatas                      |
|  2561 | ambrosia                                      |
|  3689 | amish pot roast                               |
|  4527 | asian grilled chicken                         |
|  4779 | apple scones                                  |
|  6809 | asparagus soup                                |
|  7253 | apple almond pie                              |
|  7397 | banana bread 3                                |
|  7414 | beef  n  beer                                 |
|  7608 | baked beans                                   |
| 11040 | asparagus  oven roasted                       |
| 11761 | bamia                                         |
| 12100 | amazing stuffed  shrooms                      |
| 13041 | balah isham  deep fried  cream puffs  pastry |
+-------+-----------------------------------------------+
15 rows in set (0.00 sec)
```

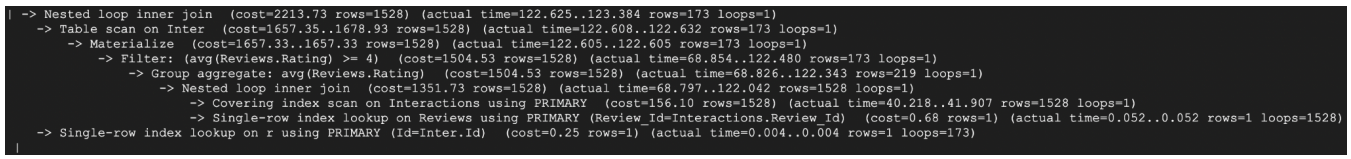
Indexing Analysis:

Default: Cost = 2243.42



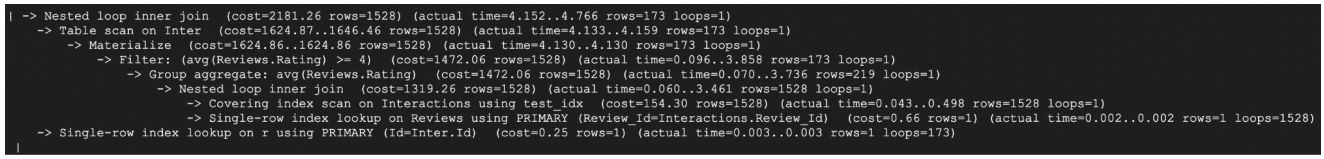
I. Indexed **Recipes(Name)**:

- Cost: 2213.73
- Percent Decrease: 1.3
- Explanation: Indexing the Recipe names did not help too much because they are almost unique. Each recipe will have a variation of a name, so it is similar to indexing a primary key.



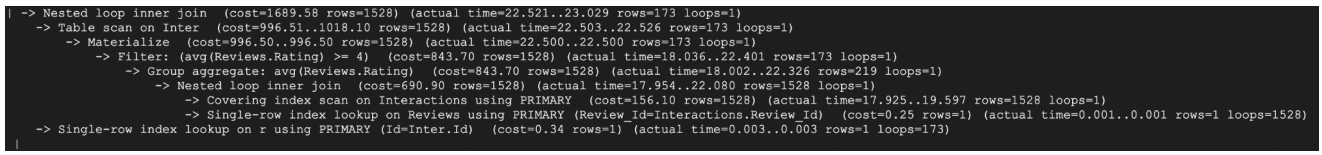
II. Indexed **Interactions(Recipe_Id)**:

- Cost: 2181.26
- Percent Decrease: 2.7
- Explanation: Recipe_Id is a foreign key in Interactions. Along with Review_Id, it acts as a primary key. Indexing the Recipe_Id in Interactions lowered the cost slightly because there may be multiple reviews that review the same Recipe, so the hash improved performance slightly by grouping these reviews together.



III. Indexed **Reviews(Rating)**:

- Cost: 1689. 58
- Percent Decrease: 24.6
- Explanation: Rating is not unique and is not strongly associated with any primary keys. There is also a smaller range of Ratings, so indexing this attribute decreased the cost significantly.



Indexing the Review(Rating) attribute gave us the best cost performance, so this is the indexing design we chose for this query.

2. Create a User's grocery list: Select the ingredient name and the number of liked recipes it occurs in for a given user. (For example, create a grocery list for the user with User_Id = 6).

```
SELECT Name, Count(Recipe_Id)
FROM Ingredients NATURAL JOIN Used_Ingredients NATURAL JOIN Likes
WHERE User_Id = 6
GROUP BY Name;
```

```
mysql> SELECT Name, Count(Recipe_Id)
    -> FROM Ingredients NATURAL JOIN Used_Ingredients NATURAL JOIN Likes
    -> WHERE User_Id = 6
    -> GROUP BY Name;
+---------------------------+------------------+
| Name                      | Count(Recipe_Id) |
+---------------------------+------------------+
| butter                    |                1 |
| honey                     |                1 |
| liqueur                   |                1 |
| orange rind               |                1 |
| apricot nectar            |                1 |
| frozen limeade concentrate |                1 |
| ginger ale                |                1 |
| pineapple juice           |                1 |
| apple                     |                1 |
| bread flmy                |                1 |
| cinnamon                  |                2 |
| milk                      |                3 |
| salt                      |                2 |
| sugar                     |                4 |
| vegetable oil             |                1 |
| yeast                     |                1 |
| almond extract            |                1 |
| apricot half              |                1 |
| cardamom                  |                1 |
| egg                       |                4 |
| vanilla                   |                2 |
| blackberry jam            |                1 |
| creme fraiche             |                1 |
| golden delicious apple    |                1 |
| unsalted butter           |                1 |
| anise oil                 |                1 |
| aniseing                  |                1 |
| baking powder             |                1 |
| flmy                      |                2 |
| margarine                 |                1 |
| bread                     |                1 |
| frozen corn               |                1 |
| pepper                    |                1 |
| shortening                |                1 |
| bacon                     |                1 |
| cream cheese              |                1 |
| croissant                 |                1 |
| salt and pepper           |                1 |
+---------------------------+------------------+
38 rows in set (0.00 sec)
```

Indexing Analysis:

Default: Cost = 83.57

```
| -> Table scan on <temporary>  (actual time=132.766..132.773 rows=38 loops=1)
    -> Aggregate using temporary table  (actual time=132.764..132.764 rows=38 loops=1)
        -> Nested loop inner join  (cost=83.57 rows=63) (actual time=71.347..132.399 rows=50 loops=1)
            -> Nested loop inner join  (cost=16.11 rows=63) (actual time=20.159..51.753 rows=50 loops=1)
                -> Covering index lookup on Likes using PRIMARY (User_Id=6)  (cost=1.77 rows=8) (actual time=0.046..0.062 rows=8 loops=1)
                -> Covering index lookup on Used_Ingredients using PRIMARY (Recipe_Id=Likes.Recipe_Id)  (cost=1.11 rows=8) (actual time=6.448..6.459 rows=6 loops=8)
            -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=Used_Ingredients.Ing_Id)  (cost=0.98 rows=1) (actual time=1.612..1.612 rows=1 loops=50)
|
```

I. Indexed **Users(User_Id)**:

- Cost: 54.38
- Percent Decrease: 34.9
- Explanation: Indexing the User Id helped because we use the User Id in the WHERE clause. It was easier to retrieve the correct ingredient names because hashing the user id made it faster to find ids that were equal to 6 and directly fetch the desired data.

```
| -> Table scan on <temporary>  (actual time=0.483..0.490 rows=38 loops=1)
    -> Aggregate using temporary table  (actual time=0.480..0.480 rows=38 loops=1)
        -> Nested loop inner join  (cost=54.38 rows=63) (actual time=0.046..0.368 rows=50 loops=1)
            -> Nested loop inner join  (cost=15.91 rows=63) (actual time=0.033..0.114 rows=50 loops=1)
                -> Covering index lookup on Likes using PRIMARY (User_Id=6)  (cost=1.72 rows=8) (actual time=0.016..0.019 rows=8 loops=1)
                -> Covering index lookup on Used_Ingredients using PRIMARY (Recipe_Id=Likes.Recipe_Id)  (cost=1.09 rows=8) (actual time=0.009..0.011 rows=6 loops=8)
            -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=Used_Ingredients.Ing_Id)  (cost=0.52 rows=1) (actual time=0.005..0.005 rows=1 loops=50)
|
```

II. Indexed **Likes(Recipe_Id)**

- Cost: 53.72
- Percent Decrease: 35.7
- Explanation: Indexing the Recipe Id decreased the cost because it directly hashed the id and made it faster to search for ingredients related to those ids in the Likes table without searching every time.

```
| -> Table scan on <temporary>  (actual time=0.394..0.399 rows=38 loops=1)
    -> Aggregate using temporary table  (actual time=0.392..0.392 rows=38 loops=1)
        -> Nested loop inner join  (cost=53.72 rows=63) (actual time=0.050..0.325 rows=50 loops=1)
            -> Nested loop inner join  (cost=15.25 rows=63) (actual time=0.037..0.095 rows=50 loops=1)
                -> Covering index lookup on Likes using PRIMARY (User_Id=6)  (cost=1.05 rows=8) (actual time=0.023..0.024 rows=8 loops=1)
                -> Covering index lookup on Used_Ingredients using PRIMARY (Recipe_Id=Likes.Recipe_Id)  (cost=1.09 rows=8) (actual time=0.006..0.008 rows=6 loops=8)
            -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=Used_Ingredients.Ing_Id)  (cost=0.52 rows=1) (actual time=0.004..0.004 rows=1 loops=50)
|
```

III. Indexed **Ingredients(Name)**

- Cost: 37.15
- Percent Decrease: 55.5
- Explanation: Indexing the ingredients name will lower the cost because when doing the natural join it will have to re-find the name every time but when hashing it creates more pointers for the columns in the ingredient name from ids making the query faster.

```
| -> Table scan on <temporary>  (actual time=0.536..0.541 rows=38 loops=1)
    -> Aggregate using temporary table  (actual time=0.535..0.535 rows=38 loops=1)
        -> Nested loop inner join  (cost=37.15 rows=63) (actual time=0.033..0.472 rows=50 loops=1)
            -> Nested loop inner join  (cost=15.25 rows=63) (actual time=0.022..0.083 rows=50 loops=1)
                -> Covering index lookup on Likes using PRIMARY (User_Id=6)  (cost=1.05 rows=8) (actual time=0.011..0.013 rows=8 loops=1)
                -> Covering index lookup on Used_Ingredients using PRIMARY (Recipe_Id=Likes.Recipe_Id)  (cost=1.09 rows=8) (actual time=0.006..0.008 rows=6 loops=8)
            -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=Used_Ingredients.Ing_Id)  (cost=0.25 rows=1) (actual time=0.008..0.008 rows=1 loops=50)
|
```

3. Quick Meals: Filter Recipes to display only recipes that use less than 7 ingredients, and don't include peanuts (in case of allergies).

```
SELECT DISTINCT r.Id, r.Name
```

```
FROM Recipes r
JOIN Used_Ingredients u ON (r.Id = u.Recipe_Id)
JOIN Ingredients i ON (i.Ing_Id = u.Ing_Id)
WHERE i.Name NOT LIKE '%peanut%'
GROUP BY r.Id HAVING COUNT(u.Ing_Id) < 7
LIMIT 15;
```

```
+------+-----------------------------------------------+
| Id   | Name                                          |
+------+-----------------------------------------------+
|    0 | Name                                          |
|   46 | a jad   cucumber pickle                       |
|  243 | amber and zach s kisses                       |
|  401 | banana marshmallow ice cream  still freeze    |
|  458 | basic crepes ii                               |
|  459 | basic crepes                                  |
|  614 | augie oliver s italian sausage                |
|  653 | b c  cherry and raspberry preserves           |
|  824 | 3 berry shakes                                |
|  876 | bananas flambe                                |
| 1883 | apple cheese casserole                        |
| 2056 | baked acorn squash with spicy maple syrup     |
| 2121 | basic tea biscuits                            |
| 2155 | african ginger cake                           |
| 2561 | ambrosia                                      |
+------+-----------------------------------------------+
15 rows in set (0.01 sec)
```

Indexing Analysis:

Default: Cost = 60792.44

```
| -> Limit: 15 row(s)  (cost=60792.44 rows=15) (actual time=43.765..71.359 rows=15 loops=1)
    -> Filter: (count(u.Ing_Id) < 7)  (cost=60792.44 rows=79432) (actual time=43.763..71.354 rows=15 loops=1)
        -> Group aggregate: count(u.Ing_Id)  (cost=60792.44 rows=79432) (actual time=43.733..71.300 rows=59 loops=1)
            -> Nested loop inner join  (cost=52849.26 rows=79432) (actual time=43.707..71.190 rows=490 loops=1)
                -> Nested loop inner join  (cost=21573.39 rows=89360) (actual time=43.669..68.546 rows=490 loops=1)
                    -> Index scan on r using PRIMARY  (cost=1305.58 rows=11423) (actual time=40.388..40.414 rows=60 loops=1)
                    -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.99 rows=8) (actual time=0.060..0.468 rows=8 loops=60)
                -> Filter: (not((i.`Name` like '%peanut%')))  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=490)
                    -> Single-row index lookup on i using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=490)
|
```

I. Indexed **Recipes(Name)**:

- Cost: 60679.96
- Percent Decrease: 0.1
- Explanation: The recipe name is already unique already and creating more pointers can help point to more names but will not affect the query significantly but still decrease the cost.

```
| -> Limit: 15 row(s)  (cost=60679.96 rows=15) (actual time=0.116..1.339 rows=15 loops=1)
    -> Filter: (count(u.Ing_Id) < 7)  (cost=60679.96 rows=79432) (actual time=0.115..1.337 rows=15 loops=1)
        -> Group aggregate: count(u.Ing_Id)  (cost=60679.96 rows=79432) (actual time=0.114..1.330 rows=59 loops=1)
            -> Nested loop inner join  (cost=52736.79 rows=79432) (actual time=0.097..1.271 rows=490 loops=1)
                -> Nested loop inner join  (cost=21460.91 rows=89360) (actual time=0.087..0.389 rows=490 loops=1)
                    -> Index scan on r using PRIMARY  (cost=1262.55 rows=11423) (actual time=0.054..0.065 rows=60 loops=1)
                    -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.99 rows=8) (actual time=0.003..0.004 rows=8 loops=60)
                -> Filter: (not((i.`Name` like '%peanut%')))  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=490)
                    -> Single-row index lookup on i using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=490)
|
```

II. Indexed **Ingredients(Name)**:

- Cost: 60679.96
- Percent Decrease: 0.1

- Explanation: The ingredient name is already unique already and creating more pointers can help point to more names but will not affect the query significantly but still decrease the cost.

```
| -> Limit: 15 row(s)  (cost=60679.96 rows=15) (actual time=0.103..1.485 rows=15 loops=1)
    -> Filter: (count(u.Ing_Id) < 7)  (cost=60679.96 rows=79432) (actual time=0.102..1.483 rows=15 loops=1)
        -> Group aggregate: count(u.Ing_Id)  (cost=60679.96 rows=79432) (actual time=0.101..1.473 rows=59 loops=1)
            -> Nested loop inner join  (cost=52736.79 rows=79432) (actual time=0.084..1.392 rows=490 loops=1)
                -> Nested loop inner join  (cost=21460.91 rows=89360) (actual time=0.072..0.370 rows=490 loops=1)
                    -> Index scan on r using PRIMARY  (cost=1262.55 rows=11423) (actual time=0.060..0.072 rows=60 loops=1)
                    -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.99 rows=8) (actual time=0.003..0.004 rows=8 loops=60)
                -> Filter: (not((i.`Name` like '%peanut%')))  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=490)
                    -> Single-row index lookup on i using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=490)
|
```

III. Indexed **Ingredients(Ing_Id)**:

- Cost: 60679.96
- Percent Decrease: 0.1
- Explanation: The ingredient ids are already unique and creating more pointers can help point to more columns in the query but will not change the cost significantly but still did decrease cost slightly.

```
| -> Limit: 15 row(s)  (cost=60679.96 rows=15) (actual time=0.097..1.861 rows=15 loops=1)
    -> Filter: (count(u.Ing_Id) < 7)  (cost=60679.96 rows=79432) (actual time=0.096..1.859 rows=15 loops=1)
        -> Group aggregate: count(u.Ing_Id)  (cost=60679.96 rows=79432) (actual time=0.095..1.850 rows=59 loops=1)
            -> Nested loop inner join  (cost=52736.79 rows=79432) (actual time=0.078..1.790 rows=490 loops=1)
                -> Nested loop inner join  (cost=21460.91 rows=89360) (actual time=0.066..0.495 rows=490 loops=1)
                    -> Index scan on r using PRIMARY  (cost=1262.55 rows=11423) (actual time=0.053..0.066 rows=60 loops=1)
                    -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.99 rows=8) (actual time=0.005..0.007 rows=8 loops=60)
                -> Filter: (not((i.`Name` like '%peanut%')))  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=490)
                    -> Single-row index lookup on i using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=490)
|
```

4. Popular Ingredients: Make a list of ingredients that users should consider keeping in their fridge because they are popularly used in highly rated recipes (ingredient occurs in >= 50 recipes and those recipes have at least one 4 star review).

```
(SELECT DISTINCT Ing_Id, Ingredients.Name
FROM Ingredients NATURAL JOIN
(SELECT Ing_Id, COUNT(Recipe_Id) as Count
FROM  Used_Ingredients u
JOIN Recipes r ON (u.Recipe_Id = r.Id)
GROUP BY Ing_Id) AS Inter
WHERE Count >= 50)
INTERSECT
(SELECT DISTINCT Ing_Id, Ingredients.Name
FROM Ingredients
NATURAL JOIN Used_Ingredients u
JOIN Recipes r ON (u.Recipe_Id = r.Id)
JOIN Interactions i ON (i.Recipe_Id = r.Id)
NATURAL JOIN Reviews
WHERE Rating > 3);
```

```
mysql> (SELECT DISTINCT Ing_Id, Ingredients.Name
    -> FROM Ingredients NATURAL JOIN
    -> (SELECT Ing_Id, COUNT(Recipe_Id) as Count
    -> FROM  Used_Ingredients u
    -> JOIN Recipes r ON (u.Recipe_Id = r.Id)
    -> GROUP BY Ing_Id) AS Inter
    -> WHERE Count >= 50)
    -> INTERSECT
    -> (SELECT DISTINCT Ing_Id, Ingredients.Name
    -> FROM Ingredients
    -> NATURAL JOIN Used_Ingredients u
    -> JOIN Recipes r ON (u.Recipe_Id = r.Id)
    -> JOIN Interactions i ON (i.Recipe_Id = r.Id)
    -> NATURAL JOIN Reviews
    -> WHERE Rating > 3)
    -> LIMIT 15;
+--------+----------------------+
| Ing_Id | Name                 |
+--------+----------------------+
|     26 | acorn squash         |
|     63 | all-purpose flmy     |
|     68 | allspice             |
|     70 | almond               |
|     77 | almond extract       |
|    132 | angel hair pastum    |
|    150 | apple                |
|    154 | apple cider          |
|    155 | apple cider vinegar  |
|    159 | apple juice          |
|    170 | applesauce           |
|    184 | apricot preserve     |
|    204 | artichoke heart      |
|    208 | artificial sweetener |
|    221 | asparagu             |
+--------+----------------------+
15 rows in set (0.36 sec)
```

Indexing Analysis:

Default: Cost = 28306.96

```
| -> Table scan on <intersect temporary>  (cost=28306.96..28306.96 rows=0) (actual time=1318.196..1318.239 rows=268 loops=1)
    -> Intersect materialize with deduplication  (cost=28304.46..28304.46 rows=0) (actual time=1318.195..1318.195 rows=325 loops=1)
        -> Table scan on <temporary>  (cost=23193.70..23193.70 rows=0) (actual time=826.607..826.661 rows=325 loops=1)
            -> Temporary table with deduplication  (cost=23191.20..23191.20 rows=0) (actual time=826.605..826.605 rows=325 loops=1)
                -> Nested loop inner join  (cost=23191.20 rows=0) (actual time=816.959..826.273 rows=325 loops=1)
                    -> Table scan on Ingredients  (cost=851.45 rows=8272) (actual time=0.078..2.479 rows=8023 loops=1)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.103..0.103 rows=0 loops=8023)
                        -> Index lookup on Inter using <auto_key0> (Ing_Id=Ingredients.Ing_Id)  (actual time=0.102..0.102 rows=0 loops=8023)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=816.841..816.841 rows=325 loops=1)
                                -> Filter: (count(u.Recipe_Id) >= 50)  (actual time=815.867..816.564 rows=325 loops=1)
                                    -> Table scan on <temporary>  (actual time=815.860..816.294 rows=4154 loops=1)
                                        -> Aggregate using temporary table  (actual time=815.857..815.857 rows=4154 loops=1)
                                            -> Nested loop inner join  (cost=21460.91 rows=89360) (actual time=0.152..763.569 rows=105668 loops=1)
                                                -> Covering index scan on r using PRIMARY  (cost=1262.55 rows=11423) (actual time=0.135..330.440 rows=11943 loops=1)
                                                -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.99 rows=8) (actual time=0.026..0.035 rows=9 loops=11943)
        -> Table scan on <temporary>  (cost=5058.47..5110.76 rows=3984) (actual time=489.720..489.813 rows=586 loops=1)
            -> Temporary table with deduplication  (cost=5058.46..5058.46 rows=3984) (actual time=489.716..489.716 rows=586 loops=1)
                -> Nested loop inner join  (cost=4660.06 rows=3984) (actual time=58.098..479.109 rows=11672 loops=1)
                    -> Nested loop inner join  (cost=3265.66 rows=3984) (actual time=58.086..461.678 rows=11672 loops=1)
                        -> Nested loop inner join  (cost=2365.14 rows=509) (actual time=58.058..450.523 rows=1333 loops=1)
                            -> Nested loop inner join  (cost=692.70 rows=1528) (actual time=35.242..44.942 rows=1528 loops=1)
                                -> Covering index scan on i using Review_Id  (cost=157.90 rows=1528) (actual time=35.206..37.037 rows=1528 loops=1)
                                -> Single-row covering index lookup on r using PRIMARY (Id=i.Recipe_Id)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=1528)
                            -> Filter: (Reviews.Rating > 3)  (cost=0.99 rows=0.3) (actual time=0.265..0.265 rows=1 loops=1528)
                                -> Single-row index lookup on Reviews using PRIMARY (Review_Id=i.Review_Id)  (cost=0.99 rows=1) (actual time=0.264..0.264 rows=1 loops=1528)
                        -> Covering index lookup on u using PRIMARY (Recipe_Id=i.Recipe_Id)  (cost=0.99 rows=8) (actual time=0.004..0.007 rows=9 loops=1333)
                    -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11672)
|
```

I. Indexed **Review(Ratings)**:

- Cost:30135.86

- Percent Decrease: - 6.4 (increased)
- Explanation: Creating the Ratings index increased the query because now it will create more overhead when reading the pages for the Review table and since the ratings are already unique for each Review the index would just cause more pointers to be added increasing the cost and slowing down the query.

```
| -> Table scan on <intersect temporary>  (cost=30135.86..30135.86 rows=0) (actual time=1107.452..1107.624 rows=268 loops=1)
    -> Intersect materialize with deduplication  (cost=30133.36..30133.36 rows=0) (actual time=1107.450..1107.450 rows=325 loops=1)
        -> Table scan on <temporary>  (cost=23217.24..23217.24 rows=0) (actual time=834.718..834.768 rows=325 loops=1)
            -> Temporary table with deduplication  (cost=23214.74..23214.74 rows=0) (actual time=834.715..834.715 rows=325 loops=1)
                -> Nested loop inner join  (cost=23214.74 rows=0) (actual time=757.478..834.330 rows=325 loops=1)
                    -> Table scan on Ingredients  (cost=874.99 rows=8272) (actual time=0.063..69.526 rows=8023 loops=1)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.095..0.095 rows=0 loops=8023)
                        -> Index lookup on Inter using <auto_key0> (Ing_Id=Ingredients.Ing_Id)  (actual time=0.095..0.095 rows=0 loops=8023)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=757.376..757.376 rows=325 loops=1)
                                -> Filter: (count(u.Recipe_Id) >= 50)  (actual time=756.467..757.225 rows=325 loops=1)
                                    -> Table scan on <temporary>  (actual time=756.460..756.962 rows=4154 loops=1)
                                        -> Aggregate using temporary table  (actual time=756.456..756.456 rows=4154 loops=1)
                                            -> Nested loop inner join  (cost=21183.12 rows=89360) (actual time=79.141..700.871 rows=105668 loops=1)
                                                -> Covering index scan on r using PRIMARY  (cost=1262.55 rows=11423) (actual time=79.073..279.510 rows=11943 loops=1)
                                                -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.96 rows=8) (actual time=0.026..0.034 rows=9 loops=11943)
        -> Table scan on <temporary>  (cost=6838.93..6916.12 rows=5976) (actual time=271.852..271.947 rows=586 loops=1)
            -> Temporary table with deduplication  (cost=6838.92..6838.92 rows=5976) (actual time=271.848..271.848 rows=586 loops=1)
                -> Nested loop inner join  (cost=6241.29 rows=5976) (actual time=36.663..262.102 rows=11672 loops=1)
                    -> Nested loop inner join  (cost=2699.45 rows=5976) (actual time=36.653..244.921 rows=11672 loops=1)
                        -> Nested loop inner join  (cost=1367.18 rows=764) (actual time=36.562..234.944 rows=1333 loops=1)
                            -> Nested loop inner join  (cost=1099.79 rows=764) (actual time=36.530..229.916 rows=1333 loops=1)
                                -> Covering index scan on i using Review_Id  (cost=157.90 rows=1528) (actual time=36.470..39.028 rows=1528 loops=1)
                                -> Filter: (Reviews.Rating > 3)  (cost=0.52 rows=0.5) (actual time=0.124..0.125 rows=1 loops=1528)
                                    -> Single-row index lookup on Reviews using PRIMARY (Review_Id=i.Review_Id)  (cost=0.52 rows=1) (actual time=0.124..0.124 rows=1 loops=1528)
                            -> Single-row covering index lookup on r using PRIMARY (Id=i.Recipe_Id)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1333)
                        -> Covering index lookup on u using PRIMARY (Recipe_Id=i.Recipe_Id)  (cost=0.96 rows=8) (actual time=0.004..0.007 rows=9 loops=1333)
                    -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.49 rows=1) (actual time=0.001..0.001 rows=1 loops=11672)
|
```

II. Indexed **Recipes(Id)**:

- Cost: 28822.06
- Percent Decrease: -1.8 (increased)
- Explanation: Recipe Id is a Primary Key which is already unique so creating an index would just add more pointers to the column increasing the space but would not help to index the table any faster since every entry points to the unique value increasing the cost when adding an index.

```
| -> Table scan on <intersect temporary>  (cost=28822.06..28822.06 rows=0) (actual time=788.017..788.058 rows=268 loops=1)
    -> Intersect materialize with deduplication  (cost=28819.56..28819.56 rows=0) (actual time=788.016..788.016 rows=325 loops=1)
        -> Table scan on <temporary>  (cost=23217.24..23217.24 rows=0) (actual time=447.133..447.182 rows=325 loops=1)
            -> Temporary table with deduplication  (cost=23214.74..23214.74 rows=0) (actual time=447.130..447.130 rows=325 loops=1)
                -> Nested loop inner join  (cost=23214.74 rows=0) (actual time=366.196..446.610 rows=325 loops=1)
                    -> Table scan on Ingredients  (cost=874.99 rows=8272) (actual time=0.053..71.647 rows=8023 loops=1)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.047..0.047 rows=0 loops=8023)
                        -> Index lookup on Inter using <auto_key0> (Ing_Id=Ingredients.Ing_Id)  (actual time=0.046..0.046 rows=0 loops=8023)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=366.107..366.107 rows=325 loops=1)
                                -> Filter: (count(u.Recipe_Id) >= 50)  (actual time=365.281..365.966 rows=325 loops=1)
                                    -> Table scan on <temporary>  (actual time=365.274..365.677 rows=4154 loops=1)
                                        -> Aggregate using temporary table  (actual time=365.271..365.271 rows=4154 loops=1)
                                            -> Nested loop inner join  (cost=17120.32 rows=89360) (actual time=0.045..313.232 rows=105668 loops=1)
                                                -> Covering index scan on r using test_idx  (cost=1262.55 rows=11423) (actual time=0.032..5.379 rows=11943 loops=1)
                                                -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.61 rows=8) (actual time=0.019..0.025 rows=9 loops=11943)
        -> Table scan on <temporary>  (cost=5550.04..5602.33 rows=3984) (actual time=340.186..340.270 rows=586 loops=1)
            -> Temporary table with deduplication  (cost=5550.03..5550.03 rows=3984) (actual time=340.182..340.182 rows=586 loops=1)
                -> Nested loop inner join  (cost=5151.63 rows=3984) (actual time=24.288..331.607 rows=11672 loops=1)
                    -> Nested loop inner join  (cost=2790.52 rows=3984) (actual time=24.280..315.729 rows=11672 loops=1)
                        -> Nested loop inner join  (cost=2083.51 rows=509) (actual time=24.267..305.926 rows=1333 loops=1)
                            -> Nested loop inner join  (cost=692.70 rows=1528) (actual time=24.219..30.476 rows=1528 loops=1)
                                -> Covering index scan on i using Review_Id  (cost=157.90 rows=1528) (actual time=24.175..26.250 rows=1528 loops=1)
                                -> Single-row covering index lookup on r using PRIMARY (Id=i.Recipe_Id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1528)
                            -> Filter: (Reviews.Rating > 3)  (cost=0.81 rows=0.3) (actual time=0.180..0.180 rows=1 loops=1528)
                                -> Single-row index lookup on Reviews using PRIMARY (Review_Id=i.Review_Id)  (cost=0.81 rows=1) (actual time=0.179..0.179 rows=1 loops=1528)
                        -> Covering index lookup on u using PRIMARY (Recipe_Id=i.Recipe_Id)  (cost=0.61 rows=8) (actual time=0.004..0.007 rows=9 loops=1333)
                    -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.49 rows=1) (actual time=0.001..0.001 rows=1 loops=11672)
|
```

III. Indexed **Ingredients(Name)**:

- Cost: 27548.68
- Percent Decrease: 2.6 (decrease)
- Explanation: The Ingredients name attribute is not unique and many can be seen in many entries so adding an index will help reduce the cost and make it faster to find names that are seen many times without checking every entry per query.

```
| -> Table scan on <intersect temporary>  (cost=27548.68..27548.68 rows=0) (actual time=573.088..573.126 rows=268 loops=1)
    -> Intersect materialize with deduplication  (cost=27546.18..27546.18 rows=0) (actual time=573.087..573.087 rows=325 loops=1)
        -> Table scan on <temporary>  (cost=23193.70..23193.70 rows=0) (actual time=541.020..541.065 rows=325 loops=1)
            -> Temporary table with deduplication  (cost=23191.20..23191.20 rows=0) (actual time=541.017..541.017 rows=325 loops=1)
                -> Nested loop inner join  (cost=23191.20 rows=0) (actual time=531.337..540.690 rows=325 loops=1)
                    -> Covering index scan on Ingredients using test_idx  (cost=851.45 rows=8272) (actual time=0.041..2.359 rows=8023 loops=1)
                    -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.067..0.067 rows=0 loops=8023)
                        -> Index lookup on Inter using <auto_key0> (Ing_Id=Ingredients.Ing_Id)  (actual time=0.067..0.067 rows=0 loops=8023)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=531.262..531.262 rows=325 loops=1)
                                -> Filter: (count(u.Recipe_Id) >= 50)  (actual time=530.176..531.117 rows=325 loops=1)
                                    -> Table scan on <temporary>  (actual time=530.169..530.858 rows=4154 loops=1)
                                        -> Aggregate using temporary table  (actual time=530.165..530.165 rows=4154 loops=1)
                                            -> Nested loop inner join  (cost=15835.50 rows=89360) (actual time=18.634..486.645 rows=105668 loops=1)
                                                -> Covering index scan on r using PRIMARY  (cost=1262.55 rows=11423) (actual time=18.603..249.845 rows=11943 loops=1)
                                                -> Covering index lookup on u using PRIMARY (Recipe_Id=r.Id)  (cost=0.49 rows=8) (actual time=0.015..0.019 rows=9 loops=11943
        -> Table scan on <temporary>  (cost=4300.20..4352.48 rows=3984) (actual time=31.397..31.482 rows=586 loops=1)
            -> Temporary table with deduplication  (cost=4300.18..4300.18 rows=3984) (actual time=31.393..31.393 rows=586 loops=1)
                -> Nested loop inner join  (cost=3901.78 rows=3984) (actual time=0.106..25.342 rows=11672 loops=1)
                    -> Nested loop inner join  (cost=2507.38 rows=3984) (actual time=0.097..13.617 rows=11672 loops=1)
                        -> Nested loop inner join  (cost=1857.66 rows=509) (actual time=0.086..6.232 rows=1333 loops=1)
                            -> Nested loop inner join  (cost=692.70 rows=1528) (actual time=0.055..2.516 rows=1528 loops=1)
                                -> Covering index scan on i using Review_Id  (cost=157.90 rows=1528) (actual time=0.036..0.457 rows=1528 loops=1)
                                -> Single-row covering index lookup on r using PRIMARY (Id=i.Recipe_Id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1528)
                            -> Filter: (Reviews.Rating > 3)  (cost=0.66 rows=0.3) (actual time=0.002..0.002 rows=1 loops=1528)
                                -> Single-row index lookup on Reviews using PRIMARY (Review_Id=i.Review_Id)  (cost=0.66 rows=1) (actual time=0.002..0.002 rows=1 loops=1528)
                        -> Covering index lookup on u using PRIMARY (Recipe_Id=i.Recipe_Id)  (cost=0.50 rows=8) (actual time=0.003..0.005 rows=9 loops=1333)
                    -> Single-row index lookup on Ingredients using PRIMARY (Ing_Id=u.Ing_Id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11672)
```