

TEAM 035 STAGE 3

Provide a screenshot of the connection (i.e. showing your terminal/command-line information):

```
Welcome to Cloud Shell! Type "help" to get started.
anagha_tiwari@cloudshell:~ (cs411-415303)$ gcloud sql connect team035 --user=root --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 89582
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

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

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

mysql> █
```

DDL Commands:

In total, we have 5 tables. Their DDL command code is shown below:

```
CREATE TABLE User (
  UserID VARCHAR(255),
  UserName VARCHAR(100),
  Password VARCHAR(20),
  PRIMARY KEY (UserID)
);
```

```
CREATE TABLE Recipe (
  RecipeTitle VARCHAR(225),
  Ingredients TEXT,
  Directions TEXT,
  PRIMARY KEY (RecipeTitle)
);
```

```
CREATE TABLE Food (
  FoodName VARCHAR(225),
  Fat INT,
  Protein INT,
  Carbs INT,
  Measures INT,
  Category VARCHAR(255),
  PRIMARY KEY (FoodName)
);
```

```
CREATE TABLE Favorites (  
  FavoriteID VARCHAR(255),  
  DateAdded int,  
  UserID VARCHAR(255),  
  PRIMARY KEY (FavoriteID, UserID)  
);
```

```
CREATE TABLE MyRecipes (  
  RecipeTitle VARCHAR(255),  
  Ingredients TEXT,  
  Directions TEXT,  
  UserID VARCHAR(255),  
  PRIMARY KEY (RecipeTitle, UserID)  
);
```

(We know we should use this DDL for the Favorites and MyRecipes table but it did not work on GCP when we put the foreign key in the DDL command so we used DDL command written above.

```
CREATE TABLE Favorites (  
  FavoriteID VARCHAR(255),  
  DateAdded int,  
  UserID VARCHAR(255),  
  PRIMARY KEY (FavoriteID, UserID),  
  FOREIGN KEY (UserID) REFERENCES User(UserID)  
);
```

```
CREATE TABLE MyRecipes (  
  RecipeTitle VARCHAR(255),  
  Ingredients TEXT,  
  Directions TEXT,  
  UserID VARCHAR(255),  
  PRIMARY KEY (RecipeTitle),  
  FOREIGN KEY (UserID) REFERENCES User(UserID)  
);  
)
```

(In our diagram in Stage 2, we had ThumbsUp as an attribute in MyRecipes. But we decided to remove this since we will rank users with other factor to rank the users such as how many recipes they uploaded as shown in query 4)

```
CREATE TABLE have (  
  RecipeTitle varchar(255),  
  FoodName varchar(255),  
  PRIMARY KEY (RecipeTitle, FoodName));
```

```
CREATE TABLE Contains (  
  FavoriteID varchar(255),  
  RecipeTitle varchar(255),  
  PRIMARY KEY (FavoriteID, RecipeTitle));
```

Inserting at least 1000 rows in at least 3 tables. (You should do a count query to show this):
As shown below, our 3 tables: User, Food, and Recipe have at least 1000 rows each.

```
mysql> SELECT COUNT(*) FROM User;  
+-----+  
| COUNT(*) |  
+-----+  
|      1100 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> SELECT COUNT(*) FROM Food;  
+-----+  
| COUNT(*) |  
+-----+  
|      1188 |  
+-----+  
1 row in set (0.01 sec)  
  
mysql> SELECT COUNT(*) FROM Recipe;  
+-----+  
| COUNT(*) |  
+-----+  
|      1127 |  
+-----+  
1 row in set (0.01 sec)
```

SQL QUERIES:

1. Filter out the recipes using ingredients that contain proteins more than 15g.

Get the average value of protein per category by using Group by. Select the categories that have an average value of proteins that is greater than 15 in the subquery. Then use the selected categories to filter out the recipes that use the ingredients in those categories. Recipe table is joined with Food table by seeing the Food.FoodName is found in Recipe.Ingredients.

This query will be used for the filtering functionality that helps the user to find the recipes that include a specific amount of some nutrition; such as recipes that include carbs less than 10, and recipes that include fat less than 5.

SELECT r.RecipeTitle, r.Ingredients, r.Directions
FROM Recipe r JOIN Food f ON r.Ingredients LIKE CONCAT('%', f.FoodName, '%')
WHERE f.Category IN (
SELECT Category
FROM Food
GROUP BY Category
HAVING AVG(Protein) > 15
) LIMIT 15;

RecipeTitle	Ingredients	Directions
Apple Nut Ring	["1 1/2 c. oil", "2 c. sugar", "2 eggs", "3 c. flour", "1/4 tsp. salt", "1 tsp. baking soda", "2 tsp. cinnamon", "1 can apple pie filling", "1 c. chopped pecans"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Cream Cheese Fruit Squares	["1 1/2 c. sugar, divided in half", "1/3 c. butter", "1 1/2 c. graham cracker crumbs", "8 oz. cream cheese", "4 eggs", "1 tsp. vanilla", "1 can fruit topping mix (cherry, blueberry, apple or boysenberry)"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Creole Green Beans	["1 green pepper, chopped", "1 onion, chopped", "2 tbsp. bacon drippings", "1 tbsp. flour", "1 tsp. salt", "1 tsp. sugar", "dash of paprika", "1 (16 oz.) can tomatoes", "1 (16 oz.) can green peas"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Cornbread Salad	["2 pkg. jiffy cornbread mix, cooked, cooled and crumbled", "1 1/2 c. chopped onion", "1 medium green pepper, chopped", "3 medium firm tomatoes", "8 to 10 slices bacon, cooked, drained and crumbled", "1 1/2 c. cubed light welvesta cheese", "1 1/2 c. salad dressing or mayo", "2 tbsp. mustard", "salt and pepper to taste"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Cornbread Salad	["2 pkg. jiffy cornbread mix, cooked, cooled and crumbled", "1 1/2 c. chopped onion", "1 medium green pepper, chopped", "3 medium firm tomatoes", "8 to 10 slices bacon, cooked, drained and crumbled", "1 1/2 c. cubed light welvesta cheese", "1 1/2 c. salad dressing or mayo", "2 tbsp. mustard", "salt and pepper to taste"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Corn Okra Casserole	["1 green pepper, sliced", "1 onion, sliced", "1/2 lb. bacon, fried and crumbled", "1 can whole kernel corn, drained", "1 can creamed corn", "1 can okra and tomatoes or frozen okra or fresh smothered okra (much better)", "8 to 10 oz. grated cheddar cheese"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Apple Crisp Pie	["1 graham cracker crust", "1 large egg yolk", "3 1/2 c. sliced apples", "1 tbsp. lemon juice", "1/2 c. sugar", "1/4 c. light brown sugar", "3 tbsp. all-purpose flour", "1/4 tsp. salt", "1/2 tsp. ground cinnamon", "1/4 ts p. ground nutmeg", "3/4 c. all-purpose flour", "1/4 c. sugar", "1/4 c. light brown sugar", "1/3 c. butter"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Clam Chowder	["6 to 8 lean slices diced bacon", "2 c. diced, cooked potatoes", "1 c. diced onion", "2 cans minced clams, drained (reserve 1 c. clam juice)", "2 c. half and half", "1 c. water", "2 tbsp. butter", "salt and pepper to taste"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Bay Cranberry Relish	["2 small pkg. fresh cranberries", "1 c. sugar (or more if you like)", "1 whole orange", "walnuts"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Brown Gravy	["4 tbsp. bacon drippings", "1 tsp. salt", "3 tbsp. self-rising flour", "2 c. whole milk"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Italian Pasta And Bean Soup	["1 clove garlic, crushed", "3 tbsp. oil", "1 (16 oz.) can tomatoes or 2 c. chopped fresh tomatoes", "2 (13 3/4 oz.) cans beef broth", "1 tbsp. minced parsley", "1/2 tsp. salt", "1/4 tsp. free h ground pepper", "1/8 tsp. basil", "1/8 tsp. oregano", "1 (16 oz.) can white beans, drained", "1/2 lb. ditalini pasta or elbow pasta, cooked \al dente", "grated cheese"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Broccoli-Raisin Salad	["1 bunch broccoli heads", "2 stems green onion (single stalks) or 1/2 medium whole onion", "1/4 c. sunflower seeds", "1/2 c. raisins", "6 slices bacon, fried and cooled", "1/2 c. mayonnaise", "2 tbsp. v inegar", "1/8 c. sugar"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Low-Fat French Fries	["1 head broccoli, chopped", "1/2 head cauliflower, chopped", "red onion, chopped to taste", "shredded low-fat cheese", "bacon bits", "1 c. fat-free miracle whip", "3 pkg. sweet 'n low", "2 tbsp. vinegar"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Broccoli Cauliflower Salad	["1 head broccoli, chopped", "1/2 head cauliflower, chopped", "red onion, chopped to taste", "shredded low-fat cheese", "bacon bits", "1 c. fat-free miracle whip", "3 pkg. sweet 'n low", "2 tbsp. vinegar"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Bacon-Flavored Chicken Breasts	["8 boneless chicken breasts", "1 jar chipped beef", "8 slices bacon", "1 can cream of mushroom soup", "1 c. sour cream"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Decadent Fudge Cake	["1 c. butter or margarine, softened", "1 1/2 c. sugar", "4 eggs", "1/2 tsp. baking soda", "1 c. buttermilk", "2 1/2 c. all-purpose flour", "1 1/2 c. semi-sweet choccola te mini morsels, divided", "2 (4 oz.) bars sweet baking chocolate, melted and cooled", "1/3 c. chocolate syrup", "2 tsp. vanilla extract", "4 oz. white chocolate, chopped", "2 tbsp. plus 2 tsp. shortening, divided", "chocolate and white chocolate leaves (optional)"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Bacon Nuts	["1 can water chestnuts, drained (8 oz.)", "1/4 c. soy sauce", "1/2 lb. Bacon"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Pina Colada Cake	["1 box duncan hines butter cake mix", "4 eggs", "1/2 c. oil", "1 c. water", "1 can cream of coconut", "1 can crushed pineapple (optional)", "1 large cool whip", "flaked c oconut"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Bacon Corn Chowder	["1/2 lb. bacon, cut in 1/2 inch pieces", "4 medium potatoes, cut in 1/2 inch cubes", "1 medium onion, chopped", "2 c. cream-style corn", "2 c. whole kern el corn", "2 c. milk", "2 tsp. salt", "pepper to taste"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Quick Freezer Fudge	["1 box confectioners sugar", "1/2 c. cocoa", "1/4 tsp. salt", "1/4 c. milk", "1 tbsp. vanilla", "1/2 c. butter", "1 c. chopped nuts"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Bacon And Egg Breakfast Bake	["4 oz. fully cooked canadian bacon or ham, cut into small cubes", "1/2 c. finely chopped muenster, cheddar or monterey jack cheese", "4 eggs", "1 1/2 c. skim milk", "1/2 tsp. pepper"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]
Spiced Tea	["1 1/4 c. tang", "1/2 c. sugar", "1/3 c. instant tea", "1/2 tsp. cinnamon", "1/4 tsp. cloves", "dash of salt"]	["Beat together first three ingredients.", "Mix together in separate bowl all dry ingredients.", "Add oil mixture to dry mixture.", "Purse apple pie filling and add to mixture along with chopped pecans.", "Bake at 350\u00b0b0 in greased and floured tube pan for 1 hour and 15 minutes.", "Freezes well."]

15 rows in set (0.01 sec)

2. Filter out the recipes that use ingredients from some specific categories (e.g. Dairy products and Seafood is selected).

Filter out ingredients that use ingredients in the Dairy products category.

Join the Food table (where it was filtered out by seeing if the category of the food is Dairy products) with Recipe tables by seeing if the Food.FoodName is included in the Recipe.Ingredients. Then do the same thing for the category Seafood.

Finally, use set operator Intersect to output all of the recipes that use Dairy products and Seafood.

This query will be used to filter out the recipes according to the user's category selection when we give the recipe recommendation on our app.

```
(
SELECT r.RecipeTitle, r.Ingredients, r.Directions
FROM Recipe r
JOIN Food f ON r.Ingredients LIKE CONCAT('%', f.FoodName, '%')
WHERE f.Category = 'Dairy products'
)
INTERSECT
(
SELECT r.RecipeTitle, r.Ingredients, r.Directions
FROM Recipe r
JOIN Food f ON r.Ingredients LIKE CONCAT('%', f.FoodName, '%')
WHERE f.Category LIKE '%Seafood'
) LIMIT 15;
```


3. Show the ranking of recipes according to how many users marked those recipes as favorites.

Group by Contains.RecipeTitle and store the the count of favoriteID for each RecipeTitle into FavoriteCount. Join Favorite and Recipe tables using the Contains table. (many to many relationship, Contains is the relationship table) Then list up the RecipeTitle in descending order by the FavoriteCount.

This will be used to recommend the recipes according to which recipes are popular to the users.

```
SELECT
    c.RecipeTitle,
    COUNT(c.FavoriteID) AS FavoriteCount
FROM
    Contains c
JOIN
    Favorites f ON c.FavoriteID = f.FavoriteID
GROUP BY
    c.RecipeTitle
ORDER BY
    FavoriteCount DESC
LIMIT 15;
```

RecipeTitle	FavoriteCount
No-Bake Nut Cookies	12
Creamy Corn	3
Reeses Cups (Candy)	3
Jewell Ball'S Chicken	2
Chicken Funny	1
Cheeseburger Potato Soup	1
Rhubarb Coffee Cake	1
Scalloped Corn	1
Nolan's Pepper Steak	1
Millionaire Pie	1
Double Cherry Delight	1
Buckeye Candy	1
Quick Barbecue Wings	1
Pink Stuff(Frozen Dessert)	1
Fresh Strawberry Pie	1

15 rows in set (0.00 sec)

4. Show the ranking of users according to how many MyRecipes they uploaded.

Join User and MyRecipes using the right outer join.

Group by UserID to get count of how many RecipeTitle was uploaded by one user and store that count to RecipesUploaded. Then make a list of UserIDs in descending order of that RecipesUploaded.

Return the list of UserName according to that UserID list and show the RecipesUploaded beside the names.

This will be used to show the ranking of users according to the contribution to encourage participation.

```
SELECT
    User.UserName,
    COUNT(MyRecipes.RecipeTitle) AS RecipesUploaded
FROM
    User
RIGHT JOIN
    MyRecipes ON User.UserID = MyRecipes.UserID
GROUP BY
    MyRecipes.UserID
ORDER BY
    RecipesUploaded DESC;
```

UserName	RecipesUploaded
Matthew Allen	8
Mia White	6
Olivia Smith	5
Logan Campbell	4
Charlotte Anderson	4
Layla Rivera	2
Penelope Ramirez	1
Ethan Garcia	1
William Thomas	1
Aria Gonzalez	1
Henry Clark	1
Ryan Flores	1
Avery Young	1
Ella Rodriguez	1
Alexander Robinson	1
Addison Parker	1
Abigail Lewis	1
Victoria Evans	1
James Moore	1
Sophia Wilson	1
Zoey Carter	1
Scarlett Walker	1
John Sanchez	1
Daniel Lee	1
Evelyn Martinez	1
Nathan Collins	1
Gabriel Perez	1

27 rows in set (0.00 sec)

INDEXING ANALYSIS

Advanced Query #1

```
SELECT r.RecipeTitle, r.Ingredients, r.Directions
FROM Recipe r JOIN Food f ON r.Ingredients LIKE CONCAT('%', f.FoodName, '%')
WHERE f.Category IN (
    SELECT Category
    FROM Food
    GROUP BY Category
    HAVING AVG(Protein) > 15
) LIMIT 15;
```

1. Index Design #1: No Index

Cost: 121292.00 (using Explain Analyze command)

```
-----+-----
| -> Limit: 15 row(s) (cost=121292.00 rows=15) (actual time=8.447..11.549 rows=15 loops=1)
|   -> Filter: (<in_optimizer>(f.Category,f.Category in (select #2)) and (r.Ingredients like concat('%',f.FoodName,'%'))) (cost=12129
| 2.00 rows=1211420) (actual time=8.446..11.546 rows=15 loops=1)
|   -> Inner hash join (no condition) (cost=121292.00 rows=1211420) (actual time=1.809..6.354 rows=4570 loops=1)
|     -> Table scan on f (cost=0.14 rows=1190) (actual time=0.677..0.684 rows=13 loops=1)
|     -> Hash
|       -> Table scan on r (cost=126.05 rows=1018) (actual time=0.043..0.377 rows=356 loops=1)
|   -> Select #2 (subquery in condition; run only once)
|     -> Filter: ((f.Category = <materialized_subquery>`.Category)) (cost=0.00..0.00 rows=0) (actual time=0.108..0.108 rows=0
| loops=13)
|       -> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.107..0.107 rows=0 loops=13)
|       -> Index lookup on <materialized_subquery> using <auto_distinct_key> (Category=f.Category) (actual time=0.107..0.
| 107 rows=0 loops=13)
|         -> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=1.381..1.381 rows=5 loops=1)
|           -> Filter: (avg(Food.Protein) > 15) (actual time=1.355..1.363 rows=5 loops=1)
|             -> Table scan on <temporary> (actual time=1.349..1.352 rows=22 loops=1)
|               -> Aggregate using temporary table (actual time=1.348..1.348 rows=22 loops=1)
|                 -> Table scan on Food (cost=121.25 rows=1190) (actual time=0.101..0.513 rows=1190 loops=1)
|
|
```

Explanation: We first ran our EXPLAIN ANALYZE command when we did not have any index, so that we would have a baseline to compare our possible indexes.

2. Index Design #2: CREATE INDEX food_cat_idx on Food(Category);

Cost: 121292

Explanation: This did not improve the overall cost of the query. We believe that creating an index on the food category key does not affect the overall cost because the category attribute stores non-numerical data that cannot be “naturally ordered,” and would still require a search through every single value.

3. Index Design #3: CREATE INDEX food_protein_idx on Food(Protein);

Cost: 121292.00

```
-----+-----
| -> Limit: 15 row(s) (cost=121292.00 rows=15) (actual time=7.711..10.815 rows=15 loops=1)
|   -> Filter: (<in_optimizer>(f.Category,f.Category in (select #2)) and (r.Ingredients like concat('%',f.FoodName,'%'))) (cost=121292.00 rows=1211420) (actual time=7.709
| ..10.813 rows=15 loops=1)
|   -> Inner hash join (no condition) (cost=121292.00 rows=1211420) (actual time=0.518..5.101 rows=4570 loops=1)
|     -> Table scan on f (cost=0.14 rows=1190) (actual time=0.039..0.047 rows=13 loops=1)
|     -> Hash
|       -> Table scan on r (cost=126.05 rows=1018) (actual time=0.039..0.364 rows=356 loops=1)
|   -> Select #2 (subquery in condition; run only once)
|     -> Filter: ((f.Category = <materialized_subquery>`.Category)) (cost=0.00..0.00 rows=0) (actual time=0.148..0.148 rows=0 loops=13)
|       -> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.148..0.148 rows=0 loops=13)
|       -> Index lookup on <materialized_subquery> using <auto_distinct_key> (Category=f.Category) (actual time=0.148..0.148 rows=0 loops=13)
|         -> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=1.906..1.906 rows=5 loops=1)
|           -> Filter: (avg(Food.Protein) > 15) (actual time=1.889..1.897 rows=5 loops=1)
|             -> Table scan on <temporary> (actual time=1.881..1.884 rows=22 loops=1)
|               -> Aggregate using temporary table (actual time=1.879..1.879 rows=22 loops=1)
|                 -> Table scan on Food (cost=121.25 rows=1190) (actual time=0.510..1.019 rows=1190 loops=1)
|
|
```

Explanation: We thought to create an index on the Food Protein key since our WHERE clause includes filtering the average amount of protein > 15. This did not change the overall cost because we believe that the protein attribute wasn't used for inner hashing join operations and neither was it used for table scanning. Further, in our original non-index cost-time analysis, there was no cost for finding the average(food.protein) > 15, which is why we think indexing on protein does not affect overall cost.

4. Index Design #4: CREATE INDEX recipe_ing_idx ON Recipe(Ingredients(255)); Cost: 121292.00

```
=====
| -> Limit: 15 row(s) (cost=121292.00 rows=15) (actual time=7.642..11.157 rows=15 loops=1)
| -> Filter: (<in_optimizer>(f.Category,f.Category in (select #2)) and (r.Ingredients like concat('%',f.FoodName,'%'))) (cost=121292.00 rows=1211420) (actual time=
7.640..11.154 rows=15 loops=1)
|   -> Inner hash join (no condition) (cost=121292.00 rows=1211420) (actual time=0.539..5.507 rows=4570 loops=1)
|     -> Table scan on f (cost=0.14 rows=1190) (actual time=0.040..0.051 rows=13 loops=1)
|     -> Hash
|       -> Table scan on r (cost=126.05 rows=1018) (actual time=0.031..0.381 rows=356 loops=1)
|   -> Select #2 (subquery in condition: run only once)
|     -> Filter: ((f.Category = <materialized_subquery>`.Category)) (cost=0.00..0.00 rows=0) (actual time=0.128..0.128 rows=0 loops=13)
|       -> Limit: 1 row(s) (cost=0.00..0.00 rows=0) (actual time=0.128..0.128 rows=0 loops=13)
|         -> Index lookup on <materialized_subquery> using <auto_distinct_key> (Category=f.Category) (actual time=0.128..0.128 rows=0 loops=13)
|           -> Materialize with deduplication (cost=0.00..0.00 rows=0) (actual time=1.634..1.634 rows=5 loops=1)
|             -> Filter: (avg(Food.Protein) > 15) (actual time=1.616..1.624 rows=5 loops=1)
|               -> Table scan on <temporary> (actual time=1.610..1.613 rows=22 loops=1)
|                 -> Aggregate using temporary table (actual time=1.609..1.609 rows=22 loops=1)
|                   -> Table scan on Food (cost=121.25 rows=1190) (actual time=0.231..0.688 rows=1190 loops=1)
```

Explanation: We thought to create an index on the Recipe Ingredients because that is one of the non-primary keys where we do a JOIN operation. However, this does not change the overall cost because Ingredients is of type "TEXT," and doing indexing on such large key types can be inefficient and not change the cost at all. Further, when looking at the no-index original cost analysis, there was actually no cost associated with recipe ingredients lookups, which is why it makes sense that indexing on it would not change anything.

Final Index Design Used: No-Index Configuration (Index Design #1)

We believe that this is the best indexing design since none of the other index designs improved our overall cost, we believe that choosing no index is the best since it avoids the extra memory and time needed to create an unnecessary index that does not change the overall cost of the SQL query. We believe that there are a lot of factors that affect the Cost of the SQL queries, and simply indexing on variables that don't impact the cost outcome would not be very valuable; instead, it would be better to not have any indexing involved as to save computing and memory resources.

Advanced Query #2

```
(
SELECT r.RecipeTitle, r.Ingredients, r.Directions
FROM Recipe r
JOIN Food f ON r.Ingredients LIKE CONCAT('%', f.FoodName, '%')
WHERE f.Category = 'Dairy products'
```

```

)
INTERSECT
(
SELECT r.RecipeTitle, r.Ingredients, r.Directions
FROM Recipe r
JOIN Food f ON r.Ingredients LIKE CONCAT('%', f.FoodName, '%')
WHERE f.Category LIKE '%Seafood'
) LIMIT 15;

```

1. Index Design #1: No Index
Cost: 27254.71 (using Explain-Analyze command)

```

| -> Limit: 15 row(s) (cost=27254.71..27254.89 rows=15) (actual time=362.698..362.797 rows=15 loops=1)
    -> Table scan on <intersect temporary> (cost=27254.71..27425.42 rows=13459) (actual time=362.696..362.795 rows=15 loops=1)
        -> Intersect materialize with deduplication (cost=27254.70..27254.70 rows=13459) (actual time=362.685..362.685 rows=776 loops=1)
            -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=12280.89 rows=13459) (actual time=1.163..1.172 rows=1537 loops=1)
                -> Inner hash join (no condition) (cost=12280.89 rows=13459) (actual time=1.048..1.048 rows=69874 loops=1)
                    -> Table scan on r (cost=0.48 rows=1018) (actual time=0.012..1.640 rows=1127 loops=1)
                    -> Hash
                        -> Filter: (f.Category = 'Dairy products') (cost=121.25 rows=119) (actual time=0.470..1.013 rows=62 loops=1)
                            -> Table scan on f (cost=121.25 rows=1190) (actual time=0.463..0.881 rows=1190 loops=1)
            -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=13627.92 rows=14953) (actual time=13.068..177.177 rows=39 loops=1)
                -> Inner hash join (no condition) (cost=13627.92 rows=14953) (actual time=0.875..8.108 rows=72128 loops=1)
                    -> Table scan on r (cost=0.45 rows=1018) (actual time=0.012..1.300 rows=1127 loops=1)
                    -> Hash
                        -> Filter: (f.Category like '%Seafood') (cost=121.25 rows=132) (actual time=0.063..0.809 rows=64 loops=1)
                            -> Table scan on f (cost=121.25 rows=1190) (actual time=0.048..0.436 rows=1190 loops=1)

```

Explanation: We first ran our EXPLAIN ANALYZE command when we did not have any index, so that we would have a baseline to compare our possible indexes.

2. Index Design #2: CREATE INDEX recipe_ing_idx ON Recipe(Ingredients(255));
Cost: 27254.71

```

| -> Limit: 15 row(s) (cost=27254.71..27254.89 rows=15) (actual time=366.820..366.931 rows=15 loops=1)
    -> Table scan on <intersect temporary> (cost=27254.71..27425.42 rows=13459) (actual time=366.820..366.929 rows=15 loops=1)
        -> Intersect materialize with deduplication (cost=27254.70..27254.70 rows=13459) (actual time=366.807..366.807 rows=776 loops=1)
            -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=12280.89 rows=13459) (actual time=0.763..1.179 rows=1537 loops=1)
                -> Inner hash join (no condition) (cost=12280.89 rows=13459) (actual time=0.650..8.823 rows=69874 loops=1)
                    -> Table scan on r (cost=0.48 rows=1018) (actual time=0.010..1.834 rows=1127 loops=1)
                    -> Hash
                        -> Filter: (f.Category = 'Dairy products') (cost=121.25 rows=119) (actual time=0.111..0.621 rows=62 loops=1)
                            -> Table scan on f (cost=121.25 rows=1190) (actual time=0.104..0.495 rows=1190 loops=1)
            -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=13627.92 rows=14953) (actual time=12.493..175.582 rows=39 loops=1)
                -> Inner hash join (no condition) (cost=13627.92 rows=14953) (actual time=0.845..8.117 rows=72128 loops=1)
                    -> Table scan on r (cost=0.45 rows=1018) (actual time=0.010..1.258 rows=1127 loops=1)
                    -> Hash
                        -> Filter: (f.Category like '%Seafood') (cost=121.25 rows=132) (actual time=0.063..0.807 rows=64 loops=1)
                            -> Table scan on f (cost=121.25 rows=1190) (actual time=0.047..0.439 rows=1190 loops=1)

```

Explanation: We thought that creating an index on the Recipe ingredients would decrease the overall runtime since the ingredients attribute was used in the “JOIN” clause in this SQL query. However, this does not improve the overall cost because we believe that recipe ingredients don’t have a natural order that would improve the search time and cost.

3. Index Design #3: CREATE INDEX food_cat_idx ON Food(Category);
Cost: 20692.97

```

-----+
| -> Limit: 15 row(s) (cost=20692.97..20693.15 rows=15) (actual time=374.282..374.383 rows=15 loops=1)
|   -> Table scan on <intersect temporary> (cost=20692.97..20783.10 rows=7012) (actual time=374.281..374.381 rows=15 loops=1)
|     -> Intersect materialize with deduplication (cost=20692.95..20692.95 rows=7012) (actual time=374.268..374.268 rows=776 loops=1)
|       -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=6363.82 rows=7012) (actual time=0.254..178.838 rows=1537 loops=1)
|         -> Inner hash join (no condition) (cost=6363.82 rows=7012) (actual time=0.145..8.051 rows=69874 loops=1)
|           -> Table scan on r (cost=0.75 rows=1018) (actual time=0.014..1.488 rows=1127 loops=1)
|             -> Hash
|               -> Covering index lookup on f using food_cat_idx (Category='Dairy products') (cost=16.93 rows=62) (actual time=0.101..0.113 rows=62 loops=1)
|                 -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=13627.92 rows=14953) (actual time=12.364..183.406 rows=39 loops=1)
|                   -> Inner hash join (no condition) (cost=13627.92 rows=14953) (actual time=0.721..8.642 rows=72128 loops=1)
|                     -> Table scan on r (cost=0.45 rows=1018) (actual time=0.012..1.529 rows=1127 loops=1)
|                       -> Hash
|                         -> Filter: (f.Category like '%Seafood') (cost=121.25 rows=132) (actual time=0.297..0.682 rows=64 loops=1)
|                           -> Covering index scan on f using food_cat_idx (cost=121.25 rows=1190) (actual time=0.043..0.346 rows=1190 loops=1)
|
|

```

Explanation: We thought of creating an index on the food “Category” key as it is used in the “WHERE” clause of the SQL query. This in fact decreased our overall cost from 27254.71 to 20692.97 because the overall cost to scan the table and hash the food category decreased after adding the index. We expected this to happen, as it would make searching for certain categories in the WHERE clause easier.

4. Index Design #4: CREATE INDEX recipe_dir_idx ON Recipe(Directions(255)); Cost: 27254.71

```

-----+
| -> Limit: 15 row(s) (cost=27254.71..27254.89 rows=15) (actual time=367.984..368.085 rows=15 loops=1)
|   -> Table scan on <intersect temporary> (cost=27254.71..27425.42 rows=13459) (actual time=367.983..368.083 rows=15 loops=1)
|     -> Intersect materialize with deduplication (cost=27254.70..27254.70 rows=13459) (actual time=367.970..367.970 rows=776 loops=1)
|       -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=12280.89 rows=13459) (actual time=0.737..177.373 rows=1537 loops=1)
|         -> Inner hash join (no condition) (cost=12280.89 rows=13459) (actual time=0.626..8.380 rows=69874 loops=1)
|           -> Table scan on r (cost=0.48 rows=1018) (actual time=0.009..1.589 rows=1127 loops=1)
|             -> Hash
|               -> Filter: (f.Category = 'Dairy products') (cost=121.25 rows=119) (actual time=0.066..0.598 rows=62 loops=1)
|                 -> Table scan on f (cost=121.25 rows=1190) (actual time=0.060..0.435 rows=1190 loops=1)
|                   -> Filter: (r.Ingredients like concat('%',f.FoodName,'%')) (cost=13627.92 rows=14953) (actual time=12.484..179.034 rows=39 loops=1)
|                     -> Inner hash join (no condition) (cost=13627.92 rows=14953) (actual time=0.815..8.343 rows=72128 loops=1)
|                       -> Table scan on r (cost=0.45 rows=1018) (actual time=0.011..1.350 rows=1127 loops=1)
|                         -> Hash
|                           -> Filter: (f.Category like '%Seafood') (cost=121.25 rows=132) (actual time=0.059..0.778 rows=64 loops=1)
|                             -> Table scan on f (cost=121.25 rows=1190) (actual time=0.044..0.416 rows=1190 loops=1)
|
|

```

Explanation: We thought of creating an index on the Recipe “Direction” key as it is used to filter and select the rows in our SQL queries including with the “INTERSECT” command. This did not decrease the cost of our SQL query, and after further inspection, we saw that the recipe directions were not used in an inner hashing join operation or table scanning functions or loops. It made sense then that the overall cost stayed the same as the indexing did not impact any of these operations.

Final Index Design Used: Index Design #3

Out of all 4 configurations, we believe that index design #3 involving an index for the Food Category key is the best configuration, as it decreased our cost from 2725.41 to 20692.97. This makes sense as the cost was originally increasing due to the hash function when searching for a specific category that met a certain condition (as stated in the SQL query). By indexing our category, we could sort the category values and ensure that they were searched in less time and cost than before.

Advanced Query #3

```
SELECT
  c.RecipeTitle,
  COUNT(c.FavoriteID) AS FavoriteCount
FROM
  Contains c
JOIN
  Favorites f ON c.FavoriteID = f.FavoriteID
GROUP BY
  c.RecipeTitle
ORDER BY
  FavoriteCount DESC
LIMIT 15;
```

1. Index Design #1: No Index
Cost: 45.25

```
| -> Limit: 15 row(s) (actual time=1.184..1.186 rows=15 loops=1)
|   -> Sort: FavoriteCount DESC, limit input to 15 row(s) per chunk (actual time=1.183..1.184 rows=15 loops=1)
|     -> Table scan on <temporary> (actual time=1.059..1.072 rows=84 loops=1)
|       -> Aggregate using temporary table (actual time=1.057..1.057 rows=84 loops=1)
|         -> Nested loop inner join (cost=45.25 rows=100) (actual time=0.633..0.916 rows=100 loops=1)
|           -> Covering index scan on c using PRIMARY (cost=10.25 rows=100) (actual time=0.605..0.628 rows=100 loops=1)
|             -> Covering index lookup on f using PRIMARY (FavoriteID=c.FavoriteID) (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=100)
|
```

Explanation: We first ran our EXPLAIN ANALYZE command when we did not have any index, so that we would have a baseline to compare our possible indexes.

2. Index Design #2: CREATE INDEX dateadded_idx on Favorites(DateAdded);
Cost: 45.25

```
| -> Limit: 15 row(s) (actual time=1.252..1.254 rows=15 loops=1)
|   -> Sort: FavoriteCount DESC, limit input to 15 row(s) per chunk (actual time=1.251..1.252 rows=15 loops=1)
|     -> Table scan on <temporary> (actual time=1.150..1.161 rows=84 loops=1)
|       -> Aggregate using temporary table (actual time=1.148..1.148 rows=84 loops=1)
|         -> Nested loop inner join (cost=45.25 rows=100) (actual time=0.707..1.049 rows=100 loops=1)
|           -> Covering index scan on c using PRIMARY (cost=10.25 rows=100) (actual time=0.673..0.696 rows=100 loops=1)
|             -> Covering index lookup on f using PRIMARY (FavoriteID=c.FavoriteID) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=100)
|
```

Explanation: It makes sense why we would not see an improvement in cost with this index. This is because when we perform our inner join, we still need to check each value in the Favorites table with the Contains table, so indexing by DateAdded doesn't change the amount of times we do this. Furthermore, we realize that DateAdded only gives us information about the date of Favorites table, so this does not really help us because even by indexing by information like this, our query still has to go through Favorites and count them as well as order by our favorite count.

3. Index Design #3: CREATE INDEX food_category_idx on Food(Category); Cost: 45.25

```
| -> Limit: 15 row(s) (actual time=0.573..0.575 rows=15 loops=1)
|   -> Sort: FavoriteCount DESC, limit input to 15 row(s) per chunk (actual time=0.572..0.573 rows=15 loops=1)
|     -> Table scan on <temporary> (actual time=0.523..0.539 rows=84 loops=1)
|       -> Aggregate using temporary table (actual time=0.521..0.521 rows=84 loops=1)
|         -> Nested loop inner join (cost=45.25 rows=100) (actual time=0.073..0.421 rows=100 loops=1)
|           -> Covering index scan on c using PRIMARY (cost=10.25 rows=100) (actual time=0.050..0.073 rows=100 loops=1)
|             -> Covering index lookup on f using PRIMARY (FavoriteID=c.FavoriteID) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=100)
```

Explanation: I thought that adding this index could help us, since although we do not directly grab from the Food table, we do use the Contains table, which contains the key to the Food table. However, it does make sense why this key would not work, since we still have to go through all the values for our inner join, so indexing by a different attribute does not really change or improve our cost in any way. Since Food table is not even used in our query, it shows us that it most likely would not have any impact on our cost.

4. Index Design #4: CREATE INDEX food_measures_idx on Food(Measures); Cost: 45.25

```
| -> Limit: 15 row(s) (actual time=0.472..0.474 rows=15 loops=1)
|   -> Sort: FavoriteCount DESC, limit input to 15 row(s) per chunk (actual time=0.472..0.473 rows=15 loops=1)
|     -> Table scan on <temporary> (actual time=0.439..0.450 rows=84 loops=1)
|       -> Aggregate using temporary table (actual time=0.438..0.438 rows=84 loops=1)
|         -> Nested loop inner join (cost=45.25 rows=100) (actual time=0.074..0.347 rows=100 loops=1)
|           -> Covering index scan on c using PRIMARY (cost=10.25 rows=100) (actual time=0.052..0.074 rows=100 loops=1)
|             -> Covering index lookup on f using PRIMARY (FavoriteID=c.FavoriteID) (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=100)
```

Explanation: As seen above, we realize that when we use an attribute to index that does not appear in our query, it will not help our cost at all. This makes sense, because we would ideally want to index by some relevant attribute, so that we can improve our costs, searches, and queries. Unfortunately, we are not left with many index options from our query, so using an irrelevant attribute does not improve our cost. Since Food table is not even used in our query, it shows us that it most likely would not have any impact on our cost.

Final Index Design Used:

Overall, it is realized that we want to select a good index value that is not a primary key and that appears in our table. However, in our original query, we only use two tables, and the attributes from these tables are already primary keys, so we cannot use them in our index. This leads us to use other attributes from the JOIN tables, however we also see that these do not improve our cost since they are not even part of the original query. Overall, it is deemed that finding an index that improves this query is very difficult, since this query utilizes primary keys only. If we wanted to improve this cost from our query, we may want to try utilizing other attributes of the table that are not primary keys, so we can use them as an index value. Because of this, we ultimately find that using no index gives us the same cost as trying to create an index.

Advanced Query #4

```
SELECT
    User.UserName,
    COUNT(MyRecipes.RecipeTitle) AS RecipesUploaded
FROM
    User
RIGHT JOIN
    MyRecipes ON User.UserID = MyRecipes.UserID
GROUP BY
    MyRecipes.UserID
ORDER BY
    RecipesUploaded DESC;
```

1. Index Design #1: No Index Cost: 23.25

```
| -> Sort: RecipesUploaded DESC (actual time=3.721..3.723 rows=27 loops=1)
| -> Table scan on <temporary> (actual time=3.694..3.698 rows=27 loops=1)
| -> Aggregate using temporary table (actual time=3.691..3.691 rows=27 loops=1)
| -> Nested loop left join (cost=23.25 rows=50) (actual time=0.721..3.606 rows=50 loops=1)
| ->   Covering index scan on MyRecipes using PRIMARY (cost=5.75 rows=50) (actual time=0.687..0.710 rows=50 loops=1)
| ->   Single-row index lookup on User using PRIMARY (UserID=MyRecipes.UserID) (cost=0.25 rows=1) (actual time=0.058..0.058 rows=1 loops=50)
|
|
```

Explanation: We first ran our EXPLAIN ANALYZE command when we did not have any index, so that we would have a baseline to compare our possible indexes.

2. Index Design #2: CREATE INDEX user_name_idx on User(UserName); Cost: 23.25

```
| -> Sort: RecipesUploaded DESC (actual time=0.318..0.320 rows=27 loops=1)
| -> Table scan on <temporary> (actual time=0.297..0.301 rows=27 loops=1)
| -> Aggregate using temporary table (actual time=0.296..0.296 rows=27 loops=1)
| -> Nested loop left join (cost=23.25 rows=50) (actual time=0.118..0.212 rows=50 loops=1)
| ->   Covering index scan on MyRecipes using PRIMARY (cost=5.75 rows=50) (actual time=0.097..0.125 rows=50 loops=1)
| ->   Single-row index lookup on User using PRIMARY (UserID=MyRecipes.UserID) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=50)
|
|
```

Explanation: When we tried indexing on UserName, we found that we did not get a cost improvement. Although this is an attribute in our query, it makes sense why we would not see any improvement, because this is just a select attribute, and indexing it does not really change our actual results, since we still have joins and groupbys on other attributes. We end up RIGHT JOINING our MyRecipes table on users, but we compare with userid, so we still have to make this comparison regardless of whether we index on an attribute in the Users table or not, therefore it makes sense that we do not see a cost improvement.

3. Index Design #3: CREATE INDEX password_idx on User(Password); Cost: 23.25

```
| -> Sort: RecipesUploaded DESC (actual time=0.263..0.265 rows=27 loops=1)
| -> Table scan on <temporary> (actual time=0.243..0.247 rows=27 loops=1)
|   -> Aggregate using temporary table (actual time=0.241..0.241 rows=27 loops=1)
|     -> Nested loop left join (cost=23.25 rows=50) (actual time=0.108..0.184 rows=50 loops=1)
|       -> Covering index scan on MyRecipes using PRIMARY (cost=5.75 rows=50) (actual time=0.089..0.097 rows=50 loops=1)
|       -> Single-row index lookup on User using PRIMARY (UserID=MyRecipes.UserID) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=50)
|
```

Explanation: We now try indexing on a different attribute of the User table, since we have limited options with this query. We assume we would get similar results to the UserName table because we are again wanting to go through our Users, however indexing by UserName or Password does not help optimize our query at all. We end up RIGHT JOINING our MyRecipes table on users, but we compare with userid, so we still have to make this comparison regardless of whether we index on an attribute in the Users table or not, therefore it makes sense that we do not see a cost improvement.

4. Index Design #4: CREATE INDEX ingredients_idx on Recipe(Ingredients(255)); Cost: 23.25

```
| -> Sort: RecipesUploaded DESC (actual time=0.256..0.257 rows=27 loops=1)
| -> Table scan on <temporary> (actual time=0.235..0.238 rows=27 loops=1)
|   -> Aggregate using temporary table (actual time=0.234..0.234 rows=27 loops=1)
|     -> Nested loop left join (cost=23.25 rows=50) (actual time=0.080..0.156 rows=50 loops=1)
|       -> Covering index scan on MyRecipes using PRIMARY (cost=5.75 rows=50) (actual time=0.059..0.068 rows=50 loops=1)
|       -> Single-row index lookup on User using PRIMARY (UserID=MyRecipes.UserID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=50)
|
```

Explanation: We again do not see a cost improvement here. On our RightJOIN, we still rely on the UserIds, and therefore still end up going through all of them, so even though indexing on Ingredients could seem helpful, it does not actually allow us any improvement because we still rely on UserIds to go through, and we cannot index on UserIds because that is a primary key.

Final Index Design Used:

Overall, we find that not much can be done to improve our costs for this query. One downside of the query is that we utilize a lot of primary keys, and we do not have many tables included in this query, so it is hard to find an index that is part of a table that we use in the query, but also is not a primary key. One way we could improve this, is by joining in more tables and expanding our query a little bit, so that we could introduce some different attributes to index on. Otherwise, as we can see above, all the indexes that we tried did not improve our cost, so this query performs the same with or without an index. This would make sense, because we would probably want to rewrite our query in such a way that we could maximize from an index by using joins. However, in our query, it is quite hard to have an index that is actually beneficial since we require a count and right join that are hard to optimize.