DDL Commands

```
CREATE TABLE `Users` (
  `User_id` int NOT NULL,
  `Username` varchar(50) NOT NULL,
  `Dietary_restrictions` varchar(255) DEFAULT NULL,
  `Budget_goal` decimal(6,2) DEFAULT NULL,
  `Nutrition_goals` int DEFAULT NULL,
  PRIMARY KEY (`User_id`)
)

CREATE TABLE Meal_Plan (
  Meal_plan_id int NOT NULL AUTO_INCREMENT,
  User_id int DEFAULT NULL,
  Date date DEFAULT NULL,
  Time enum('Breakfast','Lunch','Dinner') DEFAULT NULL,
  Recipe_id int DEFAULT NULL,
  PRIMARY KEY (Meal_plan_id),
  KEY Recipe_id (Recipe_id),
  KEY User_id (User_id),
  FOREIGN KEY (User_id) REFERENCES Users (User_id),
  FOREIGN KEY (Recipe_id) REFERENCES Recipe (Recipe_id)
)

CREATE TABLE Recipe (
  Recipe_id int NOT NULL,
  Title varchar(127) DEFAULT NULL,
  Instructions varchar(1023) DEFAULT NULL,
  Image_name varchar(255) DEFAULT NULL,
  PRIMARY KEY (Recipe_id)
)

CREATE TABLE Ingredient_Nutrition (
  Ingredient_Name varchar(255) DEFAULT NULL,
  Nutrition_id int NOT NULL,
  Carbohydrates float DEFAULT NULL,
  Kilocalories float DEFAULT NULL,
  Protein float DEFAULT NULL,
  Sugar float DEFAULT NULL,
  Total_fat float DEFAULT NULL,
```

```
  Household_weight int DEFAULT NULL,
  PRIMARY KEY (Nutrition_id),
  FOREIGN KEY (Ingredient_Name) REFERENCES Ingredient_Price (Ingredient_Name)
)

CREATE TABLE Recipe_Ingredient (
  Quantity float DEFAULT NULL,
  Unit varchar(255) DEFAULT NULL,
  Recipe_id int DEFAULT NULL,
  Ingredient_id int NOT NULL,
  Ingredient_Name varchar(255) DEFAULT NULL,
  PRIMARY KEY (Ingredient_id),
  FOREIGN KEY (Ingredient_Name) REFERENCES Ingredient_Price (Ingredient_Name),
  FOREIGN KEY (Recipe_id) REFERENCES Recipe (Recipe_id)
)

CREATE TABLE Ingredient_Price (
  Ingredient_Name varchar(255) NOT NULL,
  Price_per_100_g float DEFAULT NULL,
  PRIMARY KEY (Ingredient_Name)
)
```

Advanced Queries

Query for total price of recipe:

```
SELECT Recipe_id, Title, SUM(Quantity * Price_per_100_g) as total_price
FROM Recipe NATURAL JOIN Recipe_Ingredient NATURAL JOIN Ingredient_Price
GROUP BY Recipe_Id
```

Output rows:

```
mysql> SELECT Recipe_id, Title, SUM(Quantity * Price_per_100_g) as total_price FROM Recipe NATURAL JOIN Recipe_Ingredient NATURAL JOIN Ingredient_Pric
e GROUP BY Recipe_Id LIMIT 15;
+-----------+-----------------------------------------------------+--------------------+
| Recipe_id | Title                                               | total_price        |
+-----------+-----------------------------------------------------+--------------------+
|         0 | Miso-Butter Roast Chicken With Acorn Squash Panzanella | 187.46750229597092 |
|         1 | Crispy Salt and Pepper Potatoes                     | 21.945000052452087 |
|         2 | Thanksgiving Mac and Cheese                         | 43.320000410079956 |
|         3 | Italian Sausage and Bread Stuffing                  |  84.71750104427338 |
|         4 | Newton's Law                                        | 16.980000019073486 |
|         5 | Warm Comfort                                        |  59.56000208854675 |
|         6 | Apples and Oranges                                  |  53.07999759912491 |
|         7 | Turmeric Hot Toddy                                  | 14.717499792575836 |
|         8 | Instant Pot Lamb Haleem                             | 121.53249993920326 |
|         9 | Spiced Lentil and Caramelized Onion Baked Eggs      |  38.57000070810318 |
|        10 | Hot Pimento Cheese Dip                              |  79.41749899089336 |
|        11 | Spiral Ham in the Slow Cooker                       |  42.81000030040741 |
|        12 | Butternut Squash and Apple Soup                     |  33.21250003576279 |
|        13 | Caesar Salad Roast Chicken                          | 141.81000012159348 |
|        14 | Chicken and Rice With Leeks and Salsa Verde         | 149.53499799966812 |
+-----------+-----------------------------------------------------+--------------------+
15 rows in set (1.64 sec)
```

Query for recipe with total amount of nutrient of choice:

SELECT Recipe_id, Title, SUM(Quantity * {nutrient_of_choice})
FROM Recipe NATURAL JOIN Recipe_Ingredients NATURAL JOIN Ingredient_Nutrition
GROUP BY Recipe_Id

Output rows:

```
Database changed
mysql>  SELECT Recipe_id, Title, SUM(Quantity * Protein) FROM Recipe NATURAL JOIN Recipe_Ingredient NATURAL JOIN Ingredient_Nutrition GROUP BY Recipe_
id LIMIT 15;
+-----------+---------------------------------------------------------+-------------------------+
| Recipe_id | Title                                                   | SUM(Quantity * Protein) |
+-----------+---------------------------------------------------------+-------------------------+
|         0 | Miso-Butter Roast Chicken With Acorn Squash Panzanella |       171.78249698877335 |
|         1 | Crispy Salt and Pepper Potatoes                        |        66.40249919891357 |
|         2 | Thanksgiving Mac and Cheese                            |       184.97499656677246 |
|         3 | Italian Sausage and Bread Stuffing                     |        323.1199996471405 |
|         4 | Newton's Law                                           |       26.649999894201756 |
|         5 | Warm Comfort                                           |        4.495000064373016 |
|         6 | Apples and Oranges                                     |        9.790000021457672 |
|         7 | Turmeric Hot Toddy                                     |       13.992499865591526 |
|         8 | Instant Pot Lamb Haleem                                |       268.29249542951584 |
|         9 | Spiced Lentil and Caramelized Onion Baked Eggs         |        61.92249947786331 |
|        10 | Hot Pimento Cheese Dip                                 |        335.9674924015999 |
|        11 | Spiral Ham in the Slow Cooker                          |       27.870000064373016 |
|        12 | Butternut Squash and Apple Soup                        |       132.31749898195267 |
|        13 | Caesar Salad Roast Chicken                             |       320.28499591350555 |
|        14 | Chicken and Rice With Leeks and Salsa Verde            |        661.9799800515175 |
+-----------+---------------------------------------------------------+-------------------------+
15 rows in set (1.76 sec)
```

Query for total budget of meal plan:
SELECT User_id, SUM(Quantity * Price_per_100_g) as plan_cost
FROM Meal_Plan NATURAL JOIN Recipe NATURAL JOIN Recipe_Ingredient NATURAL JOIN Ingredient_Price
GROUP BY User_id

Output rows:
We didn't have to limit since we only have 10 meal plans

```
Database changed
mysql> SELECT User_id, SUM(Quantity * Price_per_100_g) as plan_cost
    -> FROM Meal_Plan NATURAL JOIN Recipe NATURAL JOIN Recipe_Ingredient NATURAL JOIN Ingredient_Price
    -> GROUP BY User_id
    -> ;
+---------+--------------------+
| User_id | plan_cost          |
+---------+--------------------+
|      10 | 2004.8599823713303 |
|       1 |   93.2700003683567 |
|       3 | 170.80000060796738 |
|       2 | 130.11999744176865 |
|       5 | 113.85000067949295 |
|       7 |  56.30000060796738 |
|       8 |  12.62999975681305 |
|       4 |  41.36999934911728 |
|       6 |  69.44000029563904 |
|       9 |  93.24000149965286 |
+---------+--------------------+
10 rows in set (0.34 sec)
```

Query for restricted ingredients in recipe:
SELECT Username, user_id, Dietary_restrictions, Recipe_id, Title
FROM User, Recipe
WHERE Recipe.Recipe_id NOT IN (SELECT Recipe_id FROM Recipe NATURAL JOIN Recipe_Ingredient WHERE Ingredient_name LIKE "%{forbidden_food}%")

Output rows:

```
mysql> SELECT Username, user_id, Dietary_restrictions, Recipe_id, Title FROM Users, Recipe WHERE Recipe.Recipe_id NOT IN (SELECT Recipe_id FROM Recipe
 NATURAL JOIN Recipe_Ingredient WHERE Ingredient_name LIKE "%oil%" ) LIMIT 15;
+-------------------+---------+----------------------+-----------+---------------------------------+
| Username          | user_id | Dietary_restrictions | Recipe_id | Title                           |
+-------------------+---------+----------------------+-----------+---------------------------------+
| cuisine_explorer  |      10 | nuts, nutmeg         |         1 | Crispy Salt and Pepper Potatoes |
| meal_prepper      |       9 | NULL                 |         1 | Crispy Salt and Pepper Potatoes |
| healthy_choices   |       8 | beef, milk           |         1 | Crispy Salt and Pepper Potatoes |
| gourmet_chef      |       7 | nutmeg               |         1 | Crispy Salt and Pepper Potatoes |
| simple_meals      |       6 | NULL                 |         1 | Crispy Salt and Pepper Potatoes |
| wellness_seeker   |       5 | milk, nuts           |         1 | Crispy Salt and Pepper Potatoes |
| spice_lover       |       4 | mustard seed yellow  |         1 | Crispy Salt and Pepper Potatoes |
| foodie_expert     |       3 | beef                 |         1 | Crispy Salt and Pepper Potatoes |
| budget_cook       |       2 | NULL                 |         1 | Crispy Salt and Pepper Potatoes |
| health_enthusiast |       1 | nuts, milk           |         1 | Crispy Salt and Pepper Potatoes |
| cuisine_explorer  |      10 | nuts, nutmeg         |         2 | Thanksgiving Mac and Cheese     |
| meal_prepper      |       9 | NULL                 |         2 | Thanksgiving Mac and Cheese     |
| healthy_choices   |       8 | beef, milk           |         2 | Thanksgiving Mac and Cheese     |
| gourmet_chef      |       7 | nutmeg               |         2 | Thanksgiving Mac and Cheese     |
| simple_meals      |       6 | NULL                 |         2 | Thanksgiving Mac and Cheese     |
+-------------------+---------+----------------------+-----------+---------------------------------+
15 rows in set (0.83 sec)
```

Indexing

Query for total price of recipe:

Default index

```
| EXPLAIN
      |
+-----------------------------------------------------------------------------------------------------
| -> Table scan on <temporary>  (actual time=1253.865..1258.017 rows=13484 loops=1)
    -> Aggregate using temporary table  (actual time=1253.859..1253.859 rows=13484 loops=1)
        -> Nested loop inner join  (cost=225903.61 rows=148823) (actual time=5.415..1101.529 rows=142092 loops=1)
            -> Nested loop inner join  (cost=173815.56 rows=148823) (actual time=5.382..731.228 rows=142092 loops=1)
                -> Filter: ((Recipe_Ingredient.Ingredient_Name is not null) and (Recipe_Ingredient.Recipe_id is not null))  (cost=15034.55 rows=148823) (actual time=0.072
..125.854 rows=148327 loops=1)
                    -> Table scan on Recipe_Ingredient  (cost=15034.55 rows=148823) (actual time=0.067..107.447 rows=148327 loops=1)
                -> Single-row index lookup on Ingredient_Price using PRIMARY (Ingredient_Name=Recipe_Ingredient.Ingredient_Name)  (cost=0.97 rows=1) (actual time=0.004..0
.004 rows=1 loops=148327)
            -> Single-row index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=142092)
 |
```

225k 174k 15k

Index by: Recipe_Id

```
-----------------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                                     |
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| -> Group aggregate: sum((Recipe_Ingredient.Quantity * Ingredient_Price.Price_per_100_g))  (cost=233547.56 rows=138248) (actual time=134.051..1923.908 rows=13484 loops=1
)
    -> Nested loop inner join  (cost=219722.76 rows=138248) (actual time=115.752..1891.984 rows=142092 loops=1)
        -> Nested loop inner join  (cost=73749.12 rows=138248) (actual time=76.641..1310.792 rows=148327 loops=1)
            -> Index scan on Recipe using PRIMARY  (cost=1983.88 rows=12694) (actual time=74.937..298.618 rows=13501 loops=1)
            -> Filter: (Recipe_Ingredient.Ingredient_Name is not null)  (cost=4.56 rows=11) (actual time=0.063..0.074 rows=11 loops=13501)
                -> Index lookup on Recipe_Ingredient using r_id (Recipe_id=Recipe.Recipe_id)  (cost=4.56 rows=11) (actual time=0.062..0.073 rows=11 loops=13501)
        -> Single-row index lookup on Ingredient_Price using PRIMARY (Ingredient_Name=Recipe_Ingredient.Ingredient_Name)  (cost=0.96 rows=1) (actual time=0.004..0.004 row
s=1 loops=148327)
 |
```

219k 74k 2k

Index by: Ingredient_id

```
| EXPLAIN





      |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=1795.089..1799.204 rows=13484 loops=1)
    -> Aggregate using temporary table  (actual time=1795.082..1795.082 rows=13484 loops=1)
        -> Nested loop inner join  (cost=318736.73 rows=148823) (actual time=164.509..1587.240 rows=142092 loops=1)
            -> Nested loop inner join  (cost=177327.19 rows=148823) (actual time=129.038..1222.541 rows=142092 loops=1)
                -> Filter: ((Recipe_Ingredient.Ingredient_Name is not null) and (Recipe_Ingredient.Recipe_id is not null))  (cost=15263.32 rows=148823) (actual time=23.36
0..528.955 rows=148327 loops=1)
                    -> Table scan on Recipe_Ingredient  (cost=15263.32 rows=148823) (actual time=23.350..509.500 rows=148327 loops=1)
                -> Single-row index lookup on Ingredient_Price using PRIMARY (Ingredient_Name=Recipe_Ingredient.Ingredient_Name)  (cost=0.99 rows=1) (actual time=0.004..0
.004 rows=1 loops=148327)
            -> Single-row index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.85 rows=1) (actual time=0.002..0.002 rows=1 loops=142092)
 |
```

318k 177k 15k

Index by Price_per_100_g

```
| EXPLAIN





      |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=1734.768..1739.138 rows=13484 loops=1)
    -> Aggregate using temporary table  (actual time=1734.761..1734.761 rows=13484 loops=1)
        -> Nested loop inner join  (cost=187624.91 rows=148823) (actual time=116.501..1531.425 rows=142092 loops=1)
            -> Nested loop inner join  (cost=67403.25 rows=148823) (actual time=79.164..1070.475 rows=142092 loops=1)
                -> Filter: ((Recipe_Ingredient.Ingredient_Name is not null) and (Recipe_Ingredient.Recipe_id is not null))  (cost=15315.20 rows=148823) (actual time=79.12
2..592.744 rows=148327 loops=1)
                    -> Table scan on Recipe_Ingredient  (cost=15315.20 rows=148823) (actual time=79.116..571.633 rows=148327 loops=1)
                -> Single-row index lookup on Ingredient_Price using PRIMARY (Ingredient_Name=Recipe_Ingredient.Ingredient_Name)  (cost=0.25 rows=1) (actual time=0.003..0
.003 rows=1 loops=148327)
            -> Single-row index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.71 rows=1) (actual time=0.003..0.003 rows=1 loops=142092)
 |
```

187k 67k 15k

Analysis

Indexing was done on Recipe_id, Ingredient_id, and Price_per_100_g. While Recipe_id and Ingredient_id are primary keys in certain tables, Recipe_id isn't a primary key in the Recipe_Ingredient table. Both id attributes were chosen for indexing despite their primary key status because joins were done on these attributes.

Because the queries were quite complex, there were multiple steps with their own associated costs. By far the largest was the outermost loop inner join. The next nested loop inner join was the next most costly, but was much less than the outermost. A table scan and filter had substantial costs, but were much less than either nested loop inner join. Single row index lookups cost so little they can be ignored here.

For the default configuration, the lion's share of the cost of the index went to the two joins and the table scan. The two joins cost 225k and 174k, and the filter cost 15k. Indexing on recipe_id did not change the largest join cost much, reducing it to 219k, but the next join and the filter saw their costs reduced substantially to 74k and 2k respectively. While the improvement was noticeable, it did not reduce the largest cost.

Indexing by Ingredient_id kept the lesser two costs the same and massively raised the outermost cost. The query accesses each recipe very often as it iterates through each ingredient in the recipe, meaning each recipe can expect to be accessed more than each ingredient. An improvement in the efficiency of an attribute accessed more often would logically make a larger improvement to efficiency of the query as a whole. Additionally, Ingredient_id, while an attribute that joins are done on, is a primary key of the relation it was indexed for, which would also explain poor performance.

Indexing by Price_per_100g made massive improvements to cost, reducing the cost of the largest join to 187k, the second join to 67k, and maintaining the table scan at 15k. This was the largest improvement, which makes sense given that this allowed easier access to an attribute that wasn't already a primary key that was being accessed quite repeatedly as the same ingredients were used in multiple recipes. Overall, the large improvements made by the Price_per_100g index made this index the clear optimal choice for later implementation.

<u>Query for Total Nutrient in a Recipe</u>

Default index

```
| EXPLAIN




       |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=1773.532..1777.612 rows=13484 loops=1)
    -> Aggregate using temporary table  (actual time=1773.523..1773.523 rows=13484 loops=1)
        -> Nested loop inner join  (cost=11958229.99 rows=10852173) (actual time=296.571..1569.494 rows=142095 loops=1)
            -> Filter: (Recipe_Ingredient.Ingredient_Name = Ingredient_Nutrition.Ingredient_Name)  (cost=10865774.60 rows=10852173) (actual time=296.540..980.623 rows=142
095 loops=1)
                -> Inner hash join (<hash>(Recipe_Ingredient.Ingredient_Name)=<hash>(Ingredient_Nutrition.Ingredient_Name))  (cost=10865774.60 rows=10852173) (actual time
=296.536..909.503 rows=142095 loops=1)
                    -> Filter: (Recipe_Ingredient.Recipe_id is not null)  (cost=1.96 rows=14882) (actual time=26.813..489.500 rows=148327 loops=1)
                        -> Table scan on Recipe_Ingredient  (cost=1.96 rows=148823) (actual time=26.808..475.818 rows=148327 loops=1)
                    -> Hash
                        -> Table scan on Ingredient_Nutrition  (cost=824.51 rows=7292) (actual time=54.585..265.525 rows=7413 loops=1)
            -> Single-row index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.00 rows=1) (actual time=0.004..0.004 rows=1 loops=142095)
 |
+--------------------------------------------------------------------------------------------------------------------------------------
```

1196k, 1086k,14k

Index by: Recipe_Id

```
| EXPLAIN




       |
+--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=1603.304..1607.595 rows=13484 loops=1)
    -> Aggregate using temporary table  (actual time=1603.295..1603.295 rows=13484 loops=1)
        -> Nested loop inner join  (cost=11953045.49 rows=10852173) (actual time=179.631..1376.127 rows=142095 loops=1)
            -> Filter: (Recipe_Ingredient.Ingredient_Name = Ingredient_Nutrition.Ingredient_Name)  (cost=10859800.75 rows=10852173) (actual time=131.342..617.014 rows=142
095 loops=1)
                -> Inner hash join (<hash>(Recipe_Ingredient.Ingredient_Name)=<hash>(Ingredient_Nutrition.Ingredient_Name))  (cost=10859800.75 rows=10852173) (actual time
=131.335..543.127 rows=142095 loops=1)
                    -> Filter: (Recipe_Ingredient.Recipe_id is not null)  (cost=1.14 rows=14882) (actual time=13.849..276.269 rows=148327 loops=1)
                        -> Table scan on Recipe_Ingredient  (cost=1.14 rows=148823) (actual time=13.844..262.130 rows=148327 loops=1)
                    -> Hash
                        -> Table scan on Ingredient_Nutrition  (cost=824.51 rows=7292) (actual time=33.882..111.767 rows=7413 loops=1)
            -> Single-row index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.00 rows=1) (actual time=0.005..0.005 rows=1 loops=142095)
 |
+--------------------------------------------------------------------------------------------------------------------------------------
```

1195k, 1086k, 14.8k

Index by: Ingredient_id

```
----------------------------------------------------------------------
---+
| EXPLAIN

      |
+---------------------------------------------------------------------
----------------------------------------------------------------------
---+
| -> Table scan on <temporary>  (actual time=3898.004..3902.969 rows=13484 loops=1)
    -> Aggregate using temporary table  (actual time=3897.997..3897.997 rows=13484 loops=1)
        -> Nested loop inner join  (cost=829547.71 rows=487957) (actual time=182.878..3533.161 rows=142095 loops=1)
            -> Nested loop inner join  (cost=384513.58 rows=487957) (actual time=182.841..2227.130 rows=142095 loops=1)
                -> Filter: (Ingredient_Nutrition.Ingredient_Name is not null)  (cost=824.51 rows=7292) (actual time=34.461..159.071 rows=7413 loops=1)
                    -> Table scan on Ingredient_Nutrition  (cost=824.51 rows=7292) (actual time=34.455..157.503 rows=7413 loops=1)
                -> Filter: (Recipe_Ingredient.Recipe_id is not null)  (cost=45.93 rows=67) (actual time=0.143..0.278 rows=19 loops=7413)
                    -> Index lookup on Recipe_Ingredient using r_id (Ingredient_Name=Ingredient_Nutrition.Ingredient_Name)  (cost=45.93 rows=67) (actual time=0.143..0.275
 rows=19 loops=7413)
            -> Single-row index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.81 rows=1) (actual time=0.009..0.009 rows=1 loops=142095)
 |
+---------------------------------------------------------------------
```

829k, 384k, 824,

Index by Protein

```
| EXPLAIN

      |
+---------------------------------------------------------------------
----------------------------------------------------------------------
---------------------------------------------------+
| -> Table scan on <temporary>  (actual time=1882.506..1887.122 rows=13484 loops=1)
    -> Aggregate using temporary table  (actual time=1882.496..1882.496 rows=13484 loops=1)
        -> Nested loop inner join  (cost=11958386.48 rows=10852173) (actual time=67.005..1619.273 rows=142095 loops=1)
            -> Filter: (Recipe_Ingredient.Ingredient_Name = Ingredient_Nutrition.Ingredient_Name)  (cost=10864587.81 rows=10852173) (actual time=43.737..941.827 rows=1420
95 loops=1)
                -> Inner hash join (<hash>(Recipe_Ingredient.Ingredient_Name)=<hash>(Ingredient_Nutrition.Ingredient_Name))  (cost=10864587.81 rows=10852173) (actual time
=43.731..853.518 rows=142095 loops=1)
                    -> Filter: (Recipe_Ingredient.Recipe_id is not null)  (cost=1.80 rows=14882) (actual time=38.028..649.574 rows=148327 loops=1)
                        -> Table scan on Recipe_Ingredient  (cost=1.80 rows=148823) (actual time=38.022..634.900 rows=148327 loops=1)
                    -> Hash
                        -> Table scan on Ingredient_Nutrition  (cost=753.45 rows=7292) (actual time=0.090..2.767 rows=7413 loops=1)
            -> Single-row index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.00 rows=1) (actual time=0.005..0.005 rows=1 loops=142095)
 |
+---------------------------------------------------------------------
```

1196k, 1086k,14,8k

Analysis

While this query was similar to the previous one, the costs associated were much different. Indeed, differences in the tables used for this relation specifically seem to have motivated a change in how the queries were executed. The steps of this query varied from those of the previous query and from each other. Most of the queries used a nested loop inner join as the outermost step, but performed a filter and inner hash join before (next in the output image). The innermost step remained a filter and table scan. Costs were much higher for even the best performing indexed queries, on the order of 1 million for the largest cost steps.

The default configuration saw the two joins cost about 1 million each with a filter that cost 15k. Indexing on Recipe_id did not make a noticeable difference in cost. Indexing on Ingredient_id this time made the best improvement, reducing the largest cost to 829k, the next largest to 384k, and the table scan to less than 1000. Indexing by Protein barely changed from the default.

It appears that while similar in concept, this query was fundamentally different from the past query to produce such different behavior. Considering that the only difference was that queries were done on the nutrition table, it is likely that the difference between this table and the price table is responsible. The larger number of attributes in the nutrient table increased the size of the tuples that had to be stored and evaluated over the course of the query. While indexing on the summed value worked very well for the previous query, here it did not, likely because of the aforementioned larger tuples. The clear optimal choice here was indexing on Ingredient_id, strangely enough, though it may be worth looking into improving the query itself considering the massive costs.

Query for Price of Meal Plan

```
------------------------------------------------------------------------------------------------------------------+
| -> Table scan on <temporary>  (actual time=600.941..600.943 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=600.938..600.938 rows=10 loops=1)
        -> Nested loop inner join  (cost=37693.82 rows=22323) (actual time=291.575..600.383 rows=162 loops=1)
            -> Inner hash join (Recipe_Ingredient.Recipe_id = Meal_Plan.Recipe_id)  (cost=22548.89 rows=22323) (actual time=256.404..439.743 rows=179 loops=1)
                -> Filter: (Recipe_Ingredient.Ingredient_Name is not null)  (cost=113.04 rows=14882) (actual time=17.817..343.417 rows=148327 loops=1)
                    -> Table scan on Recipe_Ingredient  (cost=113.04 rows=148823) (actual time=17.814..328.712 rows=148327 loops=1)
                -> Hash
                    -> Nested loop inner join  (cost=18.01 rows=15) (actual time=26.768..73.681 rows=15 loops=1)
                        -> Filter: (Meal_Plan.Recipe_id is not null)  (cost=2.50 rows=15) (actual time=26.737..26.770 rows=15 loops=1)
                            -> Table scan on Meal_Plan  (cost=2.50 rows=15) (actual time=26.733..26.757 rows=15 loops=1)
                        -> Single-row covering index lookup on Recipe using PRIMARY (Recipe_id=Meal_Plan.Recipe_id)  (cost=0.94 rows=1) (actual time=3.126..3.127 rows=1 loops=15)
            -> Single-row index lookup on Ingredient_Price using PRIMARY (Ingredient_Name=Recipe_Ingredient.Ingredient_Name)  (cost=0.06 rows=1) (actual time=0.897..0.897 rows=1 loops=179)
 |
```

37693,22548,113,18,2,0.94,0.06

Index by: Recipe_Ingredient.Recipe_Id

```
| -> Table scan on <temporary>  (actual time=290.810..290.812 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=290.808..290.808 rows=10 loops=1)
        -> Nested loop inner join  (cost=231.76 rows=171) (actual time=125.068..290.313 rows=162 loops=1)
            -> Nested loop inner join  (cost=85.22 rows=171) (actual time=19.884..111.384 rows=179 loops=1)
                -> Nested loop inner join  (cost=18.11 rows=15) (actual time=0.050..68.440 rows=15 loops=1)
                    -> Filter: (Meal_Plan.Recipe_id is not null)  (cost=1.75 rows=15) (actual time=0.032..0.080 rows=15 loops=1)
                        -> Table scan on Meal_Plan  (cost=1.75 rows=15) (actual time=0.032..0.066 rows=15 loops=1)
                    -> Single-row covering index lookup on Recipe using PRIMARY (Recipe_id=Meal_Plan.Recipe_id)  (cost=1.00 rows=1) (actual time=4.556..4.556 rows=1 loops=15)
                -> Filter: (Recipe_Ingredient.Ingredient_Name is not null)  (cost=3.41 rows=11) (actual time=2.814..2.860 rows=12 loops=15)
                    -> Index lookup on Recipe_Ingredient using meal_recipe (Recipe_id=Meal_Plan.Recipe_id)  (cost=3.41 rows=11) (actual time=2.811..2.854 rows=12 loops=15)
            -> Single-row index lookup on Ingredient_Price using PRIMARY (Ingredient_Name=Recipe_Ingredient.Ingredient_Name)  (cost=0.76 rows=1) (actual time=0.999..0.999 rows=1 loops=179)
 |
```

231.85,18,1.75,3.4,3.41,0.76

Index by: Ingredient_Name

```
| -> Table scan on <temporary>  (actual time=635.991..635.994 rows=10 loops=1)
    -> Aggregate using temporary table  (actual time=635.987..635.987 rows=10 loops=1)
        -> Nested loop inner join  (cost=35682.01 rows=22323) (actual time=121.778..635.477 rows=162 loops=1)
            -> Inner hash join (Recipe_Ingredient.Recipe_id = Meal_Plan.Recipe_id)  (cost=22670.94 rows=22323) (actual time=98.508..552.891 rows=179 [illegible]oops=1)
                -> Filter: (Recipe_Ingredient.Ingredient_Name is not null)  (cost=121.17 rows=14882) (actual time=0.116..435.804 rows=148327 loops=1)
                    -> Table scan on Recipe_Ingredient  (cost=121.17 rows=148823) (actual time=0.115..413.953 rows=148327 loops=1)
                -> Hash
                    -> Nested loop inner join  (cost=18.11 rows=15) (actual time=0.085..86.388 rows=15 loops=1)
                        -> Filter: (Meal_Plan.Recipe_id is not null)  (cost=1.75 rows=15) (actual time=0.055..0.100 rows=15 loops=1)
                            -> Table scan on Meal_Plan  (cost=1.75 rows=15) [illegible]ctual time=0.052..0.086 rows=15 loops=1)
                        -> Single-row covering index lookup on Recipe us[illegible]ARY (Recipe_id=Meal_Plan.Recipe_id)  (cost=1.00 rows=1) (actual time=[illegible].751..5.751 rows=1 loops=15)
            -> Single-row index lookup on Ingredient_Price using PRIMARY [illegible]dient_Name=Recipe_Ingredient.Ingredient_Name)  (cost=0.05 rows=1) (act[illegible]al time=0.461..0.461 rows=1 loops=179)
 |
```

35682,22670,121,18,1.75,1.0,0.05

I indexed by default, the Recipe_Ingredient.Recipe_id, and Ingredient_Name. The recipe id decreased the costs of the joins substantially. This makes sense since we reference the recipe_id constantly as it is the primary key of both Recipe_Ingredient and Meal_Plan. The last indexing with the ingredient_name doesn't really improve performance since it was a single-row index lookup that had no cost anyways. As such I chose to keep the index for Recipe_Ingredient.Recipe_id as this had the best cost and performance

Query for Forbidden Ingredient

```
| -> Limit: 15 row(s)  (cost=1754946354.48 rows=15) (actual time=1174.830..1174.839 rows=15 loops=1)
    -> Nested loop antijoin  (cost=1754946354.48 rows=17549203016) (actual time=1174.828..1174.836 rows=15 loops=1)
        -> Inner hash join (no condition)  (cost=13358.85 rows=126940) (actual time=63.280..63.291 rows=25 loops=1)
            -> Table scan on Recipe  (cost=193.22 rows=12694) (actual time=31.762..31.765 rows=3 loops=1)
            -> Hash
                -> Table scan on Users  (cost=2.00 rows=10) (actual time=31.486..31.492 rows=10 loops=1)
        -> Single-row index lookup on <subquery2> using <auto_distinct_key> (Recipe_id=Recipe.Recipe_id)  (actual time=44.461..44.461 rows=0 loops=25)
            -> Materialize with deduplication  (cost=84815.71..84815.71 rows=138248) (actual time=1111.525..1111.525 rows=7763 loops=1)
                -> Filter: (Recipe.Recipe_id is not null)  (cost=70990.90 rows=138248) (actual time=0.296..1102.777 rows=10426 loops=1)
                    -> Nested loop inner join  (cost=70990.90 rows=138248) (actual time=0.294..1101.303 rows=10426 loops=1)
                        -> Covering index scan on Recipe using PRIMARY  (cost=1902.59 rows=12694) (actual time=0.062..311.841 rows=13501 loops=1)
                        -> Filter: (Recipe_Ingredient.Ingredient_Name like '%oil%')  (cost=4.35 rows=11) (actual time=0.051..0.058 rows=1 loops=13501)
                            -> Index lookup on Recipe_Ingredient using r_id (Recipe_id=Recipe.Recipe_id)  (cost=4.35 rows=11) (actual time=0.044..0.05
4 rows=11 loops=13501)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
```

11754946354,13358,193,2,84815,70990,1902,4.35,4.35

Index by: Recipe_Id

```
---------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=1889184996.62 rows=15) (actual time=1115.066..1115.075 rows=15 loops=1)
    -> Nested loop antijoin  (cost=1889184996.62 rows=18891591620) (actual time=1115.063..1115.072 rows=15 loops=1)
        -> Inner hash join (no condition)  (cost=13140.62 rows=126940) (actual time=74.931..74.940 rows=25 loops=1)
            -> Table scan on Recipe  (cost=171.40 rows=12694) (actual time=30.096..30.097 rows=3 loops=1)
            -> Hash
                -> Table scan on Users  (cost=2.00 rows=10) (actual time=44.539..44.549 rows=10 loops=1)
        -> Single-row index lookup on <subquery2> using <auto_distinct_key> (Recipe_id=Recipe.Recipe_id)  (actual time=41.605..41.605 rows=0 loops=25)
            -> Materialize with deduplication  (cost=118064.59..118064.59 rows=148823) (actual time=1040.109..1040.109 rows=7763 loops=1)
                -> Filter: (Recipe.Recipe_id is not null)  (cost=103182.29 rows=148823) (actual time=8.489..1009.068 rows=10426 loops=1)
                    -> Nested loop inner join  (cost=103182.29 rows=148823) (actual time=8.486..1006.992 rows=10426 loops=1)
                        -> Filter: (Recipe_Ingredient.Ingredient_Name like '%oil%')  (cost=15227.16 rows=148823) (actual time=8.417..555.489 rows=1042
6 loops=1)
                            -> Table scan on Recipe_Ingredient  (cost=15227.16 rows=148823) (actual time=8.324..484.365 rows=148327 loops=1)
                        -> Single-row covering index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.49 rows=1) (actua
l time=0.043..0.043 rows=1 loops=10426)
 |
```

1889184996,13140,171,118064.59,1031782,15227,0.49

Index by: Ingredient_Name

```
| -> Limit: 15 row(s)  (cost=1889185383.37 rows=15) (actual time=1463.987..1463.996 rows=15 loops=1)
    -> Nested loop antijoin  (cost=1889185383.37 rows=18891591620) (actual time=1463.984..1463.992 rows=15 loops=1)
        -> Inner hash join (no condition)  (cost=13527.37 rows=126940) (actual time=101.262..101.271 rows=25 loops=1)
            -> Table scan on Recipe  (cost=210.08 rows=12694) (actual time=82.753..82.755 rows=3 loops=1)
            -> Hash
                -> Table scan on Users  (cost=2.00 rows=10) (actual time=18.458..18.464 rows=10 loops=1)
        -> Single-row index lookup on <subquery2> using <auto_distinct_key> (Recipe_id=Recipe.Recipe_id)  (actual time=54.509..54.509 rows=0 loops=25)
            -> Materialize with deduplication  (cost=181755.61..181755.61 rows=148823) (actual time=1362.699..1362.699 rows=7763 loops=1)
                -> Filter: (Recipe.Recipe_id is not null)  (cost=166873.31 rows=148823) (actual time=17.195..1353.324 rows=10426 loops=1)
                    -> Nested loop inner join  (cost=166873.31 rows=148823) (actual time=17.193..1351.579 rows=10426 loops=1)
                        -> Filter: (Recipe_Ingredient.Ingredient_Name like '%oil%')  (cost=15354.51 rows=148823) (actual time=17.150..546.522 rows=104
26 loops=1)
                            -> Table scan on Recipe_Ingredient  (cost=15354.51 rows=148823) (actual time=17.127..482.791 rows=148327 loops=1)
                        -> Single-row covering index lookup on Recipe using PRIMARY (Recipe_id=Recipe_Ingredient.Recipe_id)  (cost=0.92 rows=1) (actua
l time=0.077..0.077 rows=1 loops=10426)
 |
```

1881985383,13527,181755,166873,15353,0.92

I ran indexes on nothing, recipe_id, and ingredient_name. Indexing by recipe_id decreased the cost of the lookup very slightly but significantly increased the cost of the filter for ingredient_name. Because of this this Index is not optimal for performance. The same can be said for indexing by ingredient_name since it just increased the costs of both operations.The highest cost was the nester inner join which looks over the whole table so indexing here is negligible. Therefore the default indexing is optimal since we don’t increase costs. This is because we are just doing some default joins so since the default indexes the primary keys its more optimal to do it this way.