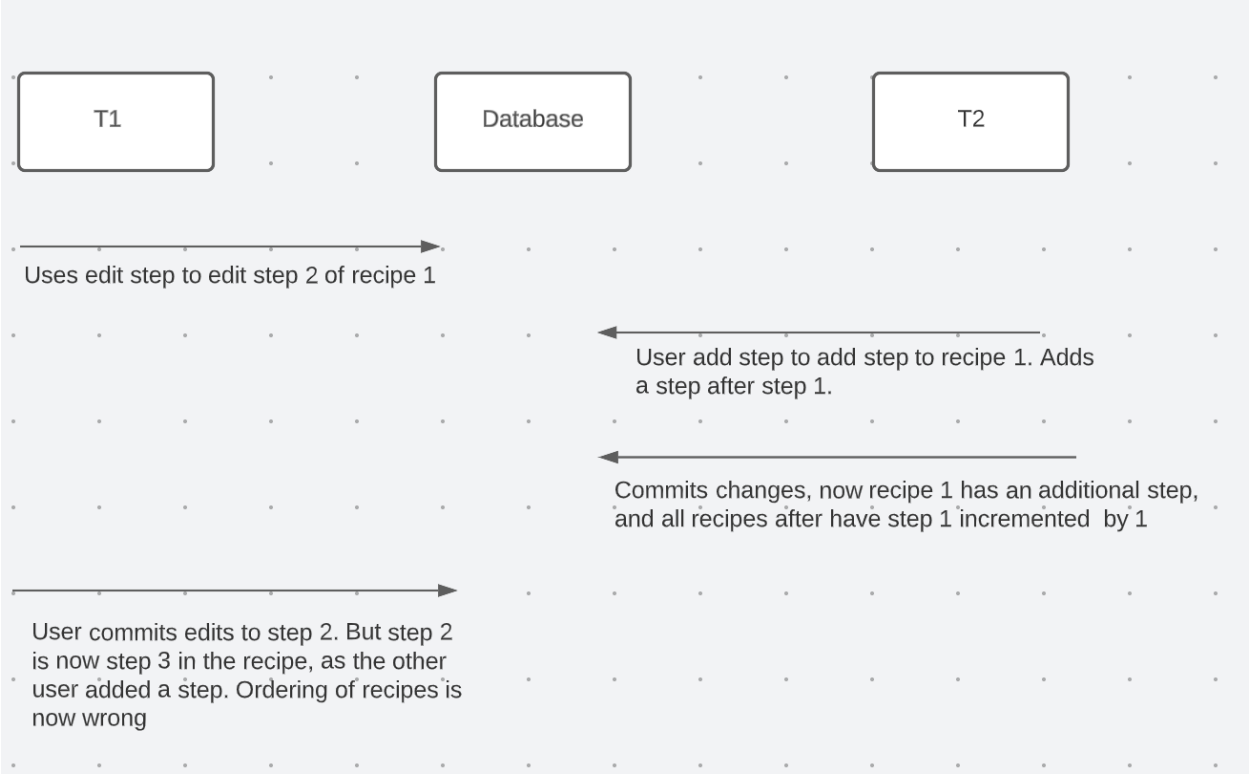
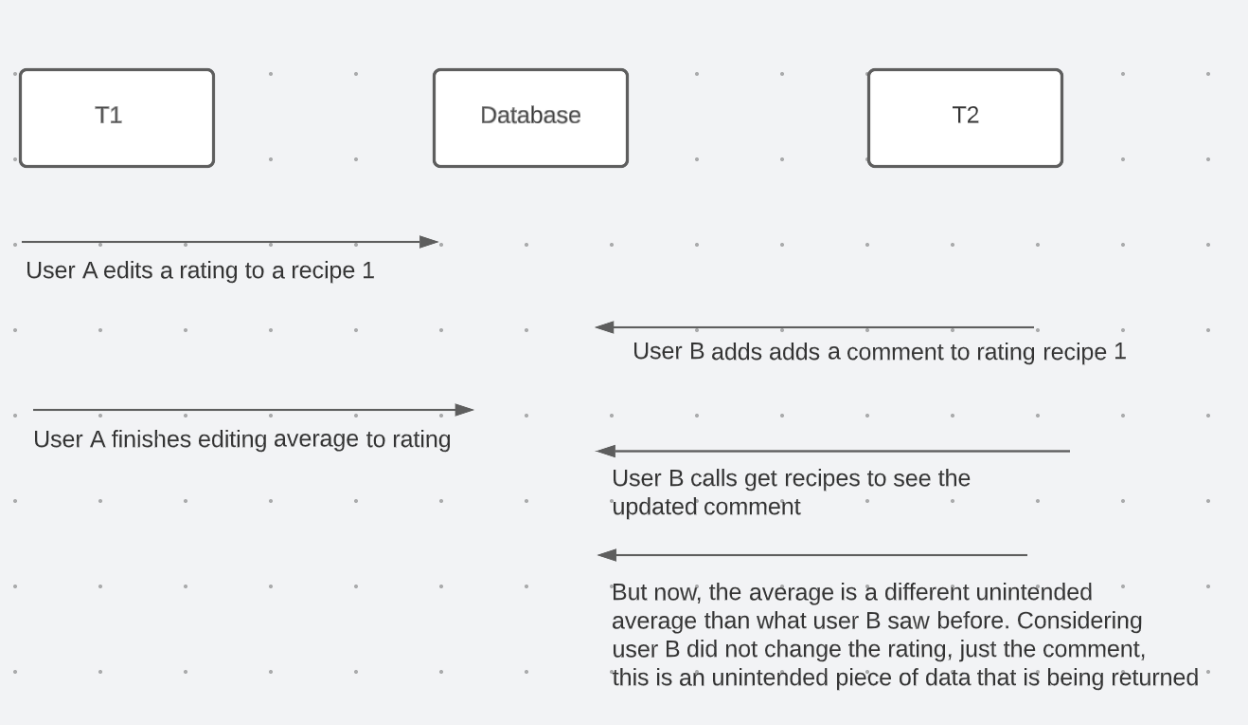
Edit / Add Steps:

- This is an example of a phantom read. Phantom reads occur when a transaction retrieves a set of records based on a certain condition, and another concurrent transaction inserts or deletes records that meet the same condition, causing the first transaction to see different results in subsequent reads. To prevent phantom reads, you can use the "Serializable" isolation level.
- In our database, if one client uses the edit_steps endpoint to edit step 2, but another client uses the add_steps endpoint to add a step to the same recipe, say add a step after step 1, the subsequent step numbers would be incremented by 1, and if the first client is still editing their step after the second client has added, they are now editing the next step, not the step they were intending to edit.
- By using the Serializable isolation level, the database guarantees that a transaction sees a consistent snapshot of the database regardless of other concurrent transactions. It achieves this by acquiring locks and placing strict constraints on the execution order of transactions.
- Sequence Diagram:



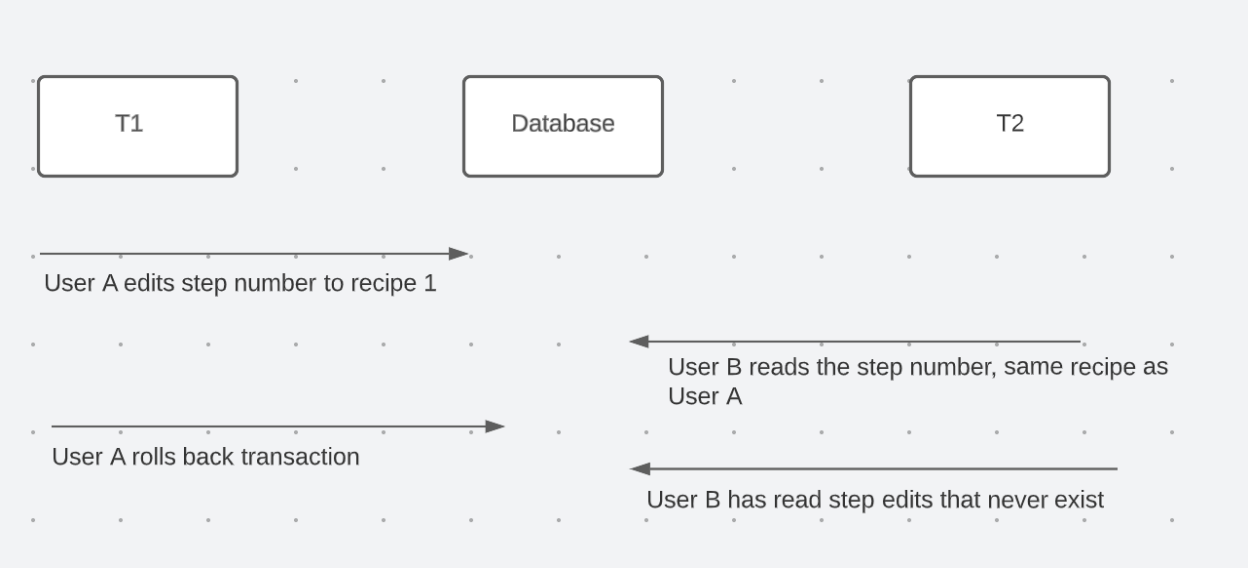
Recipe Rating:

- This is an example of a phantom read. Phantom reads occur when a transaction retrieves a set of records based on a certain condition, and another concurrent transaction inserts or deletes records that meet the same condition, causing the first transaction to see different results in subsequent reads. To prevent phantom reads, you can use the "Serializable" isolation level.
- In our database, a user can update a recipe as well as add a rating to a recipe. If a user A is updated the rating for a recipe, and user B is updating a comment to their rating for that recipe, and user B commits after user A, user B is changing just a comment, but the data that is being returned has changed the average rating as well.
- By using the Serializable isolation level, the database guarantees that a transaction sees a consistent snapshot of the database regardless of other concurrent transactions. It achieves this by acquiring locks and placing strict constraints on the execution order of transactions.
- Sequence Diagram



Edit/Add Steps:

- This is an example of a dirty read. A Dirty read is a situation when a transaction reads data that has not yet been committed. Let's say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.
- In our database, our edit recipes allows users to edit a recipe. If user A edits a recipe, and it gots rolled back during the edit, but user B reads get recipe before user A commits, this results in changes that would have never existed.
- Using appropriate isolation levels will help with this kind of concurrency problem. By setting a higher isolation level in our database, we can reduce the number of dirty reads in the database. By using a read committed setting to ensure that every time something is edited, written to make sure it is committed.



Get recipes profile:

```
| QUERY PLAN                                                                                          |
| -----------------------------------------------------------------------------------------------------------
|
| Nested Loop Left Join  (cost=1001.70..62917.52 rows=2 width=169)
        |
|   Join Filter: (recipe_tags.recipe_id = recipe.recipe_id)
|
|   ->  Nested Loop  (cost=1.27..50330.47 rows=1 width=137)
        |
|         ->  Nested Loop Left Join  (cost=0.42..50325.51 rows=1 width=105)
        |
|               Join Filter: (recipe_ratings.recipe_id = recipe.recipe_id)
        |
|               ->  Nested Loop  (cost=0.42..24861.47 rows=1 width=73)
        |
|               ->  Index Scan using recipes_pkey on recipes recipe  (cost=0.42..8.44 rows=1
width=41)               |
|                     Index Cond: (recipe_id = 0)
|
|               ->  GroupAggregate  (cost=0.00..24853.01 rows=1 width=36)
        |
|                     Group Key: instructions.recipe_id
|
|                     ->  Seq Scan on instructions  (cost=0.00..19853.00 rows=1000000
width=22)                     |
|                     Filter: (recipe_id = 0)                                                   |
|               ->  GroupAggregate  (cost=0.00..25464.01 rows=1 width=36)
        |
|               Group Key: recipe_ratings.recipe_id
|
|               ->  Seq Scan on recipe_ratings  (cost=0.00..20464.00 rows=1000000 width=16)
        |
|                     Filter: (recipe_id = 0)
|
|         ->  GroupAggregate  (cost=0.85..4.94 rows=1 width=36)
        |
|               Group Key: recipe_ingredients.recipe_id
        |
|               ->  Merge Join  (cost=0.85..4.93 rows=1 width=24)
        |
|               Merge Cond: (recipe_ingredients.ingredient_id = ingredients.ingredient_id)
        |
```

```
|                 ->  Index Only Scan using recipe_ingredients_pkey on recipe_ingredients
(cost=0.42..4.44 rows=1 width=8) |
|                       Index Cond: (recipe_id = 0)
|
|                 ->  Index Scan using ingredients_pkey on ingredients  (cost=0.42..44548.43
rows=1000000 width=24)      |
|   ->  GroupAggregate  (cost=1000.42..12587.01 rows=2 width=36)
          |
|         Group Key: recipe_tags.recipe_id
|
|         ->  Gather  (cost=1000.42..12586.98 rows=2 width=16)
          |
|                 Workers Planned: 2
|
|                 ->  Nested Loop  (cost=0.42..11586.78 rows=1 width=16)
          |
|                 ->  Parallel Seq Scan on recipe_tags  (cost=0.00..11578.33 rows=1 width=8)
          |
|                       Filter: (recipe_id = 0)
|
|                 ->  Index Scan using tags_pkey on tags  (cost=0.42..8.44 rows=1 width=16)
          |
|                       Index Cond: (tag_id = recipe_tags.tag_id)
          |
```

This endpoint looks like it can only be improved marginally through the use of indexing, particularly the sequential scan at the end can be improved by adding an index on recipe_id for the table recipe_tags. That would look like the following:

```
CREATE INDEX recipe_id ON recipe_tags (recipe_id)
```

Giving a profile of this:

```
| QUERY PLAN                                                                                                                  |
| ----------------------------------------------------------------------------------------------------------------------------
|
| Nested Loop Left Join  (cost=2.12..50359.89 rows=2 width=169)
          |
|   Join Filter: (recipe_tags.recipe_id = recipe.recipe_id)
|
|   ->  Nested Loop  (cost=1.27..50330.47 rows=1 width=137)
          |
|         ->  Nested Loop Left Join  (cost=0.42..50325.51 rows=1 width=105)
          |
|                 Join Filter: (recipe_ratings.recipe_id = recipe.recipe_id)
          |
```

```
|                 ->  Nested Loop  (cost=0.42..24861.47 rows=1 width=73)
        |
|                 ->  Index Scan using recipes_pkey on recipes recipe  (cost=0.42..8.44 rows=1
width=41)               |
|                         Index Cond: (recipe_id = 0)
|
|                 ->  GroupAggregate  (cost=0.00..24853.01 rows=1 width=36)
        |
|                         Group Key: instructions.recipe_id
|
|                         ->  Seq Scan on instructions  (cost=0.00..19853.00 rows=1000000
width=22)                       |
|                         Filter: (recipe_id = 0)                                         |
|                 ->  GroupAggregate  (cost=0.00..25464.01 rows=1 width=36)
        |
|                 Group Key: recipe_ratings.recipe_id
|
|                 ->  Seq Scan on recipe_ratings  (cost=0.00..20464.00 rows=1000000 width=16)
        |
|                         Filter: (recipe_id = 0)
|
|         ->  GroupAggregate  (cost=0.85..4.94 rows=1 width=36)
        |
|                 Group Key: recipe_ingredients.recipe_id
        |
|                 ->  Merge Join  (cost=0.85..4.93 rows=1 width=24)
        |
|                 Merge Cond: (recipe_ingredients.ingredient_id = ingredients.ingredient_id)
        |
|                 ->  Index Only Scan using recipe_ingredients_pkey on recipe_ingredients
(cost=0.42..4.44 rows=1 width=8) |
|                         Index Cond: (recipe_id = 0)
|
|                 ->  Index Scan using ingredients_pkey on ingredients  (cost=0.42..44548.43
rows=1000000 width=24)      |
|   ->  GroupAggregate  (cost=0.85..29.38 rows=2 width=36)
        |
|         Group Key: recipe_tags.recipe_id
|
|         ->  Nested Loop  (cost=0.85..29.34 rows=2 width=16)
        |
|                 ->  Index Scan using recipe_id on recipe_tags  (cost=0.42..12.46 rows=2
width=8)                        |
```

```
|                   Index Cond: (recipe_id = 0)
|
|                   ->  Index Scan using tags_pkey on tags  (cost=0.42..8.44 rows=1 width=16)
          |
|                   Index Cond: (tag_id = recipe_tags.tag_id)
|
```

Find recipes endpoint Profile:

```
| QUERY PLAN
          |
|
---------------------------------------------------------------------------------------------------------------------------------
---------------------------------------- |
| Sort  (cost=12688555.80..12688761.91 rows=82444 width=88)
          |
|   Sort Key: (count(DISTINCT CASE WHEN ((i.core_ingredient)::text = 'Flour'::text) THEN
i.name ELSE NULL::character varying END)) DESC, ri_2.recipe_id                        |
|   ->  Merge Join  (cost=188827.12..12677875.79 rows=82444 width=88)
          |
|         Merge Cond: (r.recipe_id = ri_2.recipe_id)
          |
|         ->  GroupAggregate  (cost=88852.44..91119.65 rows=82444 width=56)
          |
|                   Group Key: r.recipe_id
          |
|                   ->  Sort  (cost=88852.44..89058.55 rows=82444 width=42)
          |
|                   Sort Key: r.recipe_id
          |
|                   ->  Nested Loop  (cost=53628.03..79582.92 rows=82444 width=42)
          |
|                             ->  Hash Join  (cost=53627.59..77514.03 rows=82444 width=20)
          |
|                             Hash Cond: (r.recipe_id = ri.recipe_id)
          |
|                             ->  Seq Scan on recipes r  (cost=0.00..19312.00 rows=1000000
width=16)                                                 |
|                             ->  Hash  (cost=52597.04..52597.04 rows=82444 width=12)
          |
```

```
|                                 ->  Hash Join  (cost=33602.03..52597.04 rows=82444 width=12)
        |
|                                 Hash Cond: (ri.recipe_id = ri_1.recipe_id)
        |
|                                 ->  Seq Scan on recipe_ingredients ri  (cost=0.00..16370.00
rows=1000000 width=8)                                   |
|                                 ->  Hash  (cost=32571.48..32571.48 rows=82444 width=4)
        |
|                                       ->  HashAggregate  (cost=30922.60..31747.04
rows=82444 width=4)                                         |
|                                       Group Key: ri_1.recipe_id
        |
|                                       ->  Gather  (cost=22059.87..30716.49 rows=82444
width=4)                                        |
|                                             Workers Planned: 2
        |
|                                             ->  HashAggregate  (cost=21059.87..21472.09
rows=41222 width=4)                                     |
|                                             Group Key: ri_1.recipe_id
        |
|                                             ->  Nested Loop  (cost=0.43..20956.82 rows=41222
width=4)                                  |
|                                                   ->  Parallel Seq Scan on recipe_ingredients
ri_1  (cost=0.00..10536.67 rows=416667 width=8)   |
|                                                   ->  Memoize  (cost=0.43..1.75 rows=1
width=4)                                  |
|                                                   Cache Key: ri_1.ingredient_id
        |
|                                                   Cache Mode: logical
        |
|                                                   ->  Index Scan using ingredients_pkey on
ingredients i_1  (cost=0.42..1.74 rows=1 width=4) |
|                                                         Index Cond: (ingredient_id =
ri_1.ingredient_id)                               |
|                                                         Filter: ((core_ingredient)::text =
'Flour'::text)                                  |
|                       ->  Memoize  (cost=0.43..3.91 rows=1 width=30)
        |
|                       Cache Key: ri.ingredient_id
        |
|                       Cache Mode: logical
        |
|                       ->  Index Scan using ingredients_pkey on ingredients i  (cost=0.42..3.90
rows=1 width=30)                                        |
```

```
|                                 Index Cond: (ingredient_id = ri.ingredient_id)
        |
|       ->  GroupAggregate  (cost=99974.68..12584898.65 rows=200 width=36)
        |
|             Group Key: ri_2.recipe_id
        |
|             ->  Merge Join  (cost=99974.68..6401596.15 rows=412220000 width=48)
        |
|             Merge Cond: (ri_2.recipe_id = ins.recipe_id)
        |
|             ->  Unique  (cost=40015.17..40839.61 rows=82444 width=34)
        |
|                   ->  Sort  (cost=40015.17..40221.28 rows=82444 width=34)
        |
|                   Sort Key: ri_2.recipe_id, ri_2.ingredient_id, i_2.name
        |
|                   ->  Gather  (cost=22369.04..31025.66 rows=82444 width=34)
        |
|                         Workers Planned: 2
        |
|                         ->  HashAggregate  (cost=21369.04..21781.26 rows=41222
width=34)                                       |
|                         Group Key: ri_2.recipe_id, ri_2.ingredient_id, i_2.name,
i_2.core_ingredient                             |
|                         ->  Nested Loop  (cost=0.43..20956.82 rows=41222 width=34)
        |
|                               ->  Parallel Seq Scan on recipe_ingredients ri_2
(cost=0.00..10536.67 rows=416667 width=8)                               |
|                               ->  Memoize  (cost=0.43..1.75 rows=1 width=30)
        |
|                               Cache Key: ri_2.ingredient_id
        |
|                               Cache Mode: logical
        |
|                               ->  Index Scan using ingredients_pkey on ingredients i_2
(cost=0.42..1.74 rows=1 width=30)                       |
|                                     Index Cond: (ingredient_id = ri_2.ingredient_id)
        |
|                                     Filter: ((core_ingredient)::text = 'Flour'::text)
        |
|             ->  Materialize  (cost=59959.50..178925.98 rows=1000000 width=22)
        |
|                   ->  Gather Merge  (cost=59959.50..176425.98 rows=1000000 width=22)
        |
```

```
|                         Workers Planned: 2
        |
|                         ->  Sort  (cost=58959.48..60001.15 rows=416667 width=22)
        |
|                               Sort Key: ins.recipe_id
        |
|                               ->  Parallel Seq Scan on instructions ins  (cost=0.00..11519.67
rows=416667 width=22)                                 |
| JIT:
        |
|   Functions: 77
        |
|   Options: Inlining true, Optimization true, Expressions true, Deforming true
        |
```

This can be improved by adding the following indexes:

```
CREATE INDEX recipe_id1 ON recipe_ingredients (recipe_id);
CREATE INDEX recipe_id2 ON instructions (recipe_id);
```

These are the correct indexes because it prevents 2 total sequential scans which would slow the endpoint down further.

After adding the indexes:

```
| QUERY PLAN
        |
|
----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------- |
| Sort  (cost=12537882.25..12538088.36 rows=82444 width=88)
        |
|   Sort Key: (count(DISTINCT CASE WHEN ((i.core_ingredient)::text = 'Flour'::text) THEN
i.name ELSE NULL::character varying END)) DESC, ri_2.recipe_id                    |
|   ->  Merge Join  (cost=128868.04..12527202.23 rows=82444 width=88)
        |
|         Merge Cond: (r.recipe_id = ri_2.recipe_id)
        |
|         ->  GroupAggregate  (cost=88852.44..91119.65 rows=82444 width=56)
        |
|               Group Key: r.recipe_id
        |
|               ->  Sort  (cost=88852.44..89058.55 rows=82444 width=42)
        |
|               Sort Key: r.recipe_id
        |
```

```
|                 ->  Nested Loop  (cost=53628.03..79582.92 rows=82444 width=42)
        |
|                         ->  Hash Join  (cost=53627.59..77514.03 rows=82444 width=20)
        |
|                         Hash Cond: (r.recipe_id = ri.recipe_id)
        |
|                         ->  Seq Scan on recipes r  (cost=0.00..19312.00 rows=1000000
width=16)                                       |
|                         ->  Hash  (cost=52597.04..52597.04 rows=82444 width=12)
        |
|                                 ->  Hash Join  (cost=33602.03..52597.04 rows=82444 width=12)
        |
|                                 Hash Cond: (ri.recipe_id = ri_1.recipe_id)
        |
|                                 ->  Seq Scan on recipe_ingredients ri  (cost=0.00..16370.00
rows=1000000 width=8)                           |
|                                 ->  Hash  (cost=32571.48..32571.48 rows=82444 width=4)
        |
|                                         ->  HashAggregate  (cost=30922.60..31747.04
rows=82444 width=4)                                     |
|                                         Group Key: ri_1.recipe_id
        |
|                                         ->  Gather  (cost=22059.87..30716.49 rows=82444
width=4)                                        |
|                                                 Workers Planned: 2
        |
|                                                 ->  HashAggregate  (cost=21059.87..21472.09
rows=41222 width=4)                                     |
|                                                 Group Key: ri_1.recipe_id
        |
|                                                 ->  Nested Loop  (cost=0.43..20956.82 rows=41222
width=4)                                |
|                                                         ->  Parallel Seq Scan on recipe_ingredients
ri_1  (cost=0.00..10536.67 rows=416667 width=8)  |
|                                                         ->  Memoize  (cost=0.43..1.75 rows=1
width=4)                                        |
|                                                         Cache Key: ri_1.ingredient_id
        |
|                                                         Cache Mode: logical
        |
|                                                         ->  Index Scan using ingredients_pkey on
ingredients i_1  (cost=0.42..1.74 rows=1 width=4) |
|                                                                 Index Cond: (ingredient_id =
ri_1.ingredient_id)                             |
```

| Filter: ((core_ingredient)::text = 'Flour'::text) |
| -> Memoize (cost=0.43..3.91 rows=1 width=30) |
| Cache Key: ri.ingredient_id |
| Cache Mode: logical |
| -> Index Scan using ingredients_pkey on ingredients i (cost=0.42..3.90 rows=1 width=30) |
| Index Cond: (ingredient_id = ri.ingredient_id) |
| -> GroupAggregate (cost=40015.60..12434225.09 rows=200 width=36) |
| Group Key: ri_2.recipe_id |
| -> Merge Join (cost=40015.60..6250922.59 rows=412220000 width=48) |
| Merge Cond: (ri_2.recipe_id = ins.recipe_id) |
| -> Unique (cost=40015.17..40839.61 rows=82444 width=34) |
| -> Sort (cost=40015.17..40221.28 rows=82444 width=34) |
| Sort Key: ri_2.recipe_id, ri_2.ingredient_id, i_2.name |
| -> Gather (cost=22369.04..31025.66 rows=82444 width=34) |
| Workers Planned: 2 |
| -> HashAggregate (cost=21369.04..21781.26 rows=41222 width=34) |
| Group Key: ri_2.recipe_id, ri_2.ingredient_id, i_2.name, i_2.core_ingredient |
| -> Nested Loop (cost=0.43..20956.82 rows=41222 width=34) |
| -> Parallel Seq Scan on recipe_ingredients ri_2 (cost=0.00..10536.67 rows=416667 width=8) |
| -> Memoize (cost=0.43..1.75 rows=1 width=30) |
| Cache Key: ri_2.ingredient_id |
| Cache Mode: logical |

```
|                                          ->  Index Scan using ingredients_pkey on ingredients i_2
(cost=0.42..1.74 rows=1 width=30)                        |
|                                                Index Cond: (ingredient_id = ri_2.ingredient_id)
        |
|                                                Filter: ((core_ingredient)::text = 'Flour'::text)
        |
|                  ->  Materialize  (cost=0.42..28252.42 rows=1000000 width=22)
        |
|                           ->  Index Scan using recipe_id2 on instructions ins  (cost=0.42..25752.42
rows=1000000 width=22)                                    |
| JIT:
        |
|   Functions: 75
        |
|   Options: Inlining true, Optimization true, Expressions true, Deforming true
        |
```

Recipes endpoint profile:

```
| QUERY PLAN                                                                                  |
| ------------------------------------------------------------------------------------------- |
| Limit  (cost=28320.07..28325.90 rows=50 width=31)                                           |
|   ->  Gather Merge  (cost=28320.07..125549.16 rows=833334 width=31)                         |
|         Workers Planned: 2                                                                  |
|         ->  Sort  (cost=27320.04..28361.71 rows=416667 width=31)                            |
|               Sort Key: total_time                                                          |
|               ->  Parallel Seq Scan on recipes  (cost=0.00..13478.67 rows=416667 width=31) |
```

The following are the added indexes:

```
create index total_time on recipes (total_time);
create index spice_level on recipes (spice_level);
create index cooking_level on recipes (cooking_level);
create index servings on recipes (servings);
```

Since find list needs to be flexible and allow for fast service adding indexes for all of these will allow it to cut down the work of gathering, merging and scanning to just an index scan for all of these possible sort options.

```
| QUERY PLAN                                                                                  |
| ------------------------------------------------------------------------------------------- |
| Limit  (cost=0.42..3.21 rows=50 width=31)                                                   |
|   ->  Index Scan using total_time on recipes  (cost=0.42..55662.59 rows=1000000 width=31) |
```

Get pairings:

```
| QUERY PLAN                                                                                  |
| ------------------------------------------------------------------------------------------- |
```

```
| Nested Loop Left Join  (cost=1000.85..12591.49 rows=2 width=32)                                  |
|   Join Filter: (recipe_tags.recipe_id = recipe.recipe_id)                                        |
|   ->  Index Only Scan using recipes_pkey on recipes recipe  (cost=0.42..4.44 rows=1 width=4)
|
|         Index Cond: (recipe_id = 10)                                                            |
|   ->  GroupAggregate  (cost=1000.42..12587.01 rows=2 width=36)                                   |
|         Group Key: recipe_tags.recipe_id                                                        |
|         ->  Gather  (cost=1000.42..12586.98 rows=2 width=16)                                     |
|                 Workers Planned: 2                                                              |
|                 ->  Nested Loop  (cost=0.42..11586.78 rows=1 width=16)                           |
|                 ->  Parallel Seq Scan on recipe_tags  (cost=0.00..11578.33 rows=1 width=8) |
|                         Filter: (recipe_id = 10)                                                |
|                 ->  Index Scan using tags_pkey on tags  (cost=0.42..8.44 rows=1 width=16)  |
|                         Index Cond: (tag_id = recipe_tags.tag_id)                               |
```

Can benefit from the following index:

```
create index rt_recipe_id on recipe_tags (recipe_id)
```

This is the only index that will improve performance since the only pages that are being touched are inside of recipe_tags and we are only using the ids. So indexing by recipe id will minimize the amount of pages searched during the left join statement

```
| QUERY PLAN                                                                                      |
| ----------------------------------------------------------------------------------------------- |
| Nested Loop Left Join  (cost=1.27..33.86 rows=2 width=32)                                        |
|   Join Filter: (recipe_tags.recipe_id = recipe.recipe_id)                                        |
|   ->  Index Only Scan using recipes_pkey on recipes recipe  (cost=0.42..4.44 rows=1 width=4)
|
|         Index Cond: (recipe_id = 10)                                                            |
|   ->  GroupAggregate  (cost=0.85..29.38 rows=2 width=36)                                         |
|         Group Key: recipe_tags.recipe_id                                                        |
|         ->  Nested Loop  (cost=0.85..29.34 rows=2 width=16)                                      |
|                 ->  Index Scan using rt_recipe_id on recipe_tags  (cost=0.42..12.46 rows=2
width=8) |
|                 Index Cond: (recipe_id = 10)                                                    |
|                 ->  Index Scan using tags_pkey on tags  (cost=0.42..8.44 rows=1 width=16)
|
|                 Index Cond: (tag_id = recipe_tags.tag_id)                                       |
```

Post recipe, post rating, post user, put rating, edit steps and add steps
All use a series of selects, inserts or updates based on indexes which are assigned by default in postgress sql. They all use a very similar query to check if the user is valid before inserting or updating their respective table. Since they all rely on these default indexes their before and after would look identical and no additional indexes would speed them up.