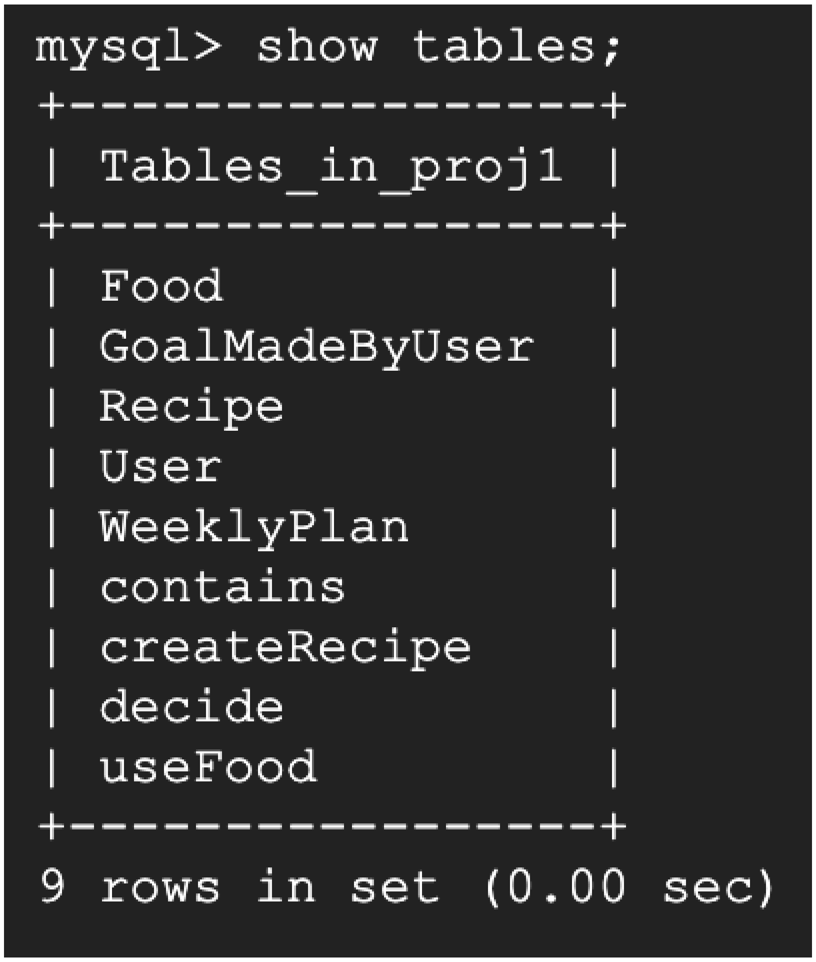
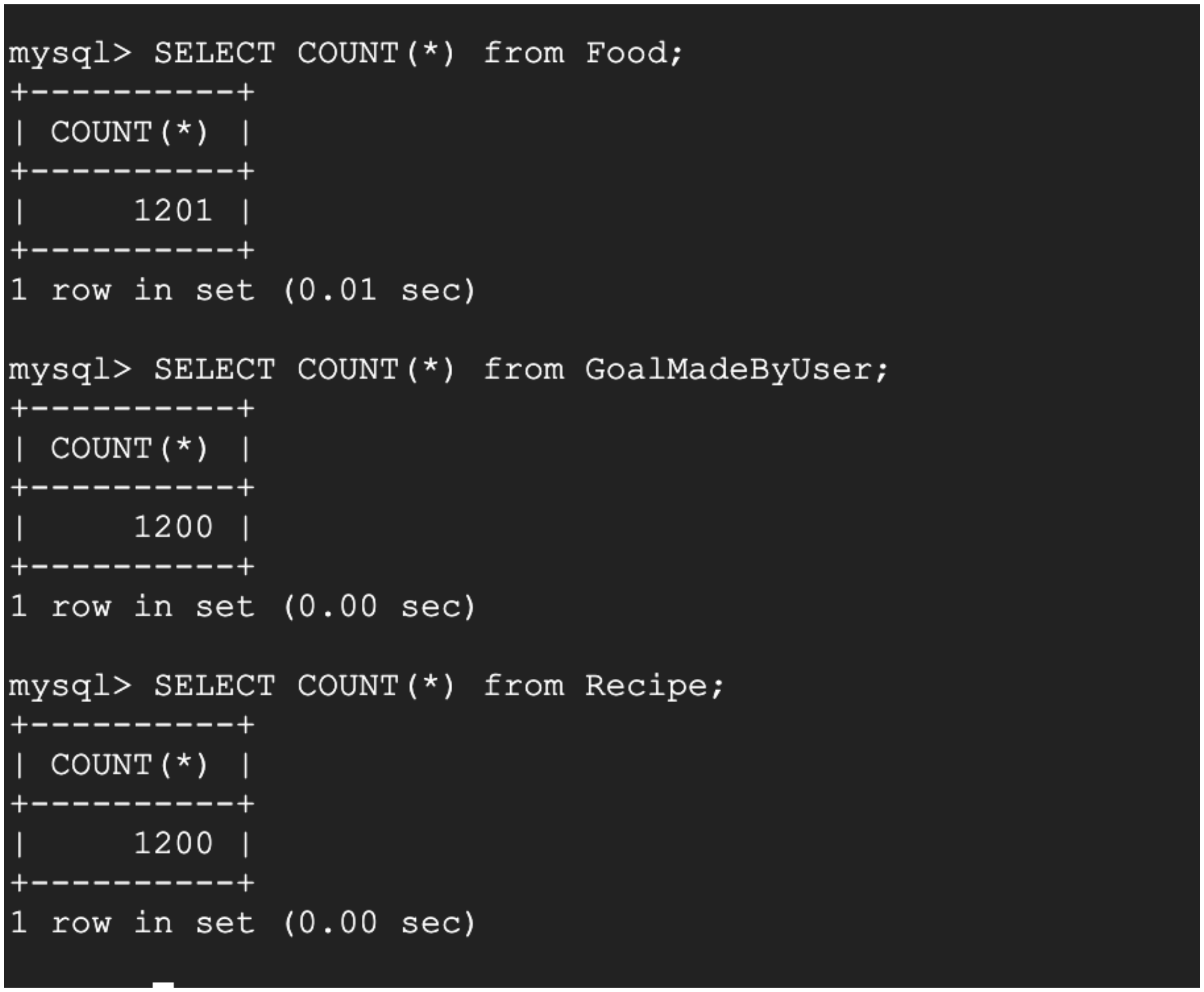
Database Implementation and Indexing

Zhaokuan Chen zc56, Siwei Zhang siwei4, Jiachen Kou jkou4, Haoran Jiang hj28



```
mysql> show tables;
+-----------------+
| Tables_in_proj1 |
+-----------------+
| Food            |
| GoalMadeByUser  |
| Recipe          |
| User            |
| WeeklyPlan      |
| contains        |
| createRecipe    |
| decide          |
| useFood         |
+-----------------+
9 rows in set (0.00 sec)
```

Fig.1 Table View


```
mysql> SELECT COUNT(*) from Food;
+----------+
| COUNT(*) |
+----------+
|     1201 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) from GoalMadeByUser;
+----------+
| COUNT(*) |
+----------+
|     1200 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) from Recipe;
+----------+
| COUNT(*) |
+----------+
|     1200 |
+----------+
1 row in set (0.00 sec)
```
Fig.2 Numbers Of Records In Tables having over 1000 rows

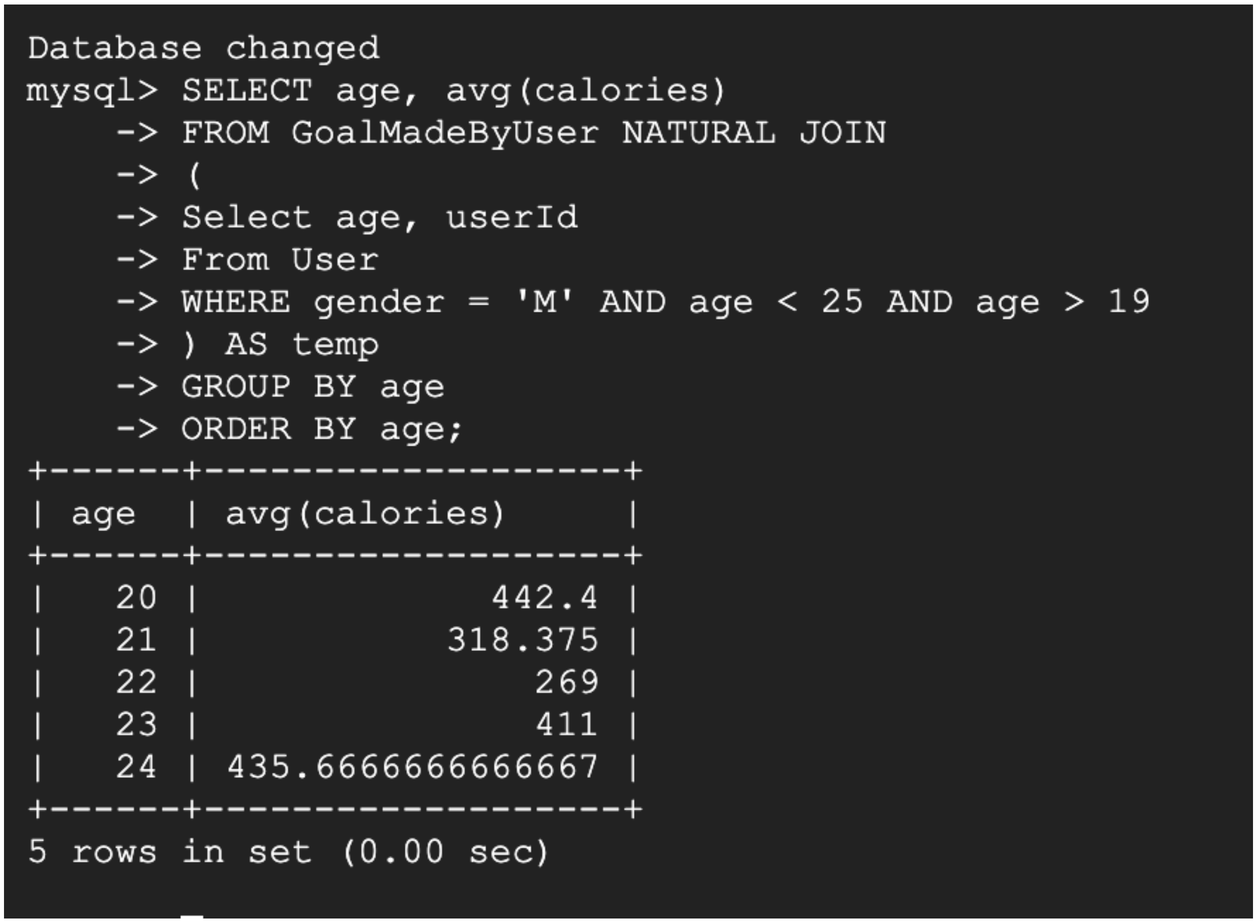
DDL commands for Tables:

```
CREATE TABLE User(
userId VARCHAR(30),
email VARCHAR(255),
age INTEGER,
gender VARCHAR(10)
password VARCHAR(30),
PRIMARY KEY(userId)
);

CREATE TABLE GoalMadeByUser(
goalId VARCHAR(30),
userId VARCHAR(30),
fat REAL,
protein REAL,
carb REAL,
calories REAL,
startDate DATE,
endDate DATE,
PRIMARY KEY(goalId),
FOREIGN KEY(userId) REFERENCES User(userId) ON DELETE CASCADE
);

CREATE TABLE WeeklyPlan(
planID VARCHAR(30),
createUserId VARCHAR(30),
PRIMARY KEY(planId)
);

CREATE TABLE Recipe(
recipeId VARCHAR(30),
recipeName VARCHAR(30),
userId VARCHAR(30),
PRIMARY KEY(recipeId)
);

CREATE TABLE Food(
foodId VARCHAR(30),
foodName VARCHAR(30),
fat REAL,
protein REAL,
carb REAL,
PRIMARY KEY(foodId)
);

CREATE TABLE UseFood(
recipeId VARCHAR(30),
foodId VARCHAR(30) NOT NULL,
weight REAL,
PRIMARY KEY(recipeId, foodId),
FOREIGN KEY(recipeId) REFERENCES Recipe(recipeId) ON DELETE CASCADE,
FOREIGN KEY(foodId) REFERENCES Food(foodId) ON DELETE CASCADE
);

CREATE TABLE decide(
userId VARCHAR(30),
planId VARCHAR(30),
PRIMARY KEY(userId, planId),
FOREIGN KEY(userId) REFERENCES User(userId) ON DELETE CASCADE,
FOREIGN KEY(planId) REFERENCES WeeklyPlan(planId) ON DELETE CASCADE
);

CREATE TABLE contains(
planId VARCHAR(30),
recipeId VARCHAR(30) NOT NULL,
PRIMARY KEY(planId, recipeId),
FOREIGN KEY(recipeId) REFERENCES Recipe(recipeId) ON DELETE CASCADE,
FOREIGN KEY(planId) REFERENCES WeeklyPlan(planId) ON DELETE CASCADE
);

CREATE TABLE createRecipe(
userId VARCHAR(30),
recipeId VARCHAR(30),
PRIMARY KEY(userId, recipeId),
FOREIGN KEY(userId) REFERENCES User(userId) ON DELETE CASCADE,
FOREIGN KEY(recipeId) REFERENCES Recipe(recipeId) ON DELETE CASCADE
);
```

Advanced Query and its index analysis:

Return the average goal calories made by same gender and similar age range with the current user(male, age22 for example) in the same year

```
SELECT age, avg(calories)
FROM GoalMadeByUser NATURAL JOIN
(
Select age, userId
From User
WHERE gender = 'M' AND age < 25 AND age > 19
) AS temp
GROUP BY age
ORDER BY age;
```



```
Database changed
mysql> SELECT age, avg(calories)
    -> FROM GoalMadeByUser NATURAL JOIN
    -> (
    -> Select age, userId
    -> From User
    -> WHERE gender = 'M' AND age < 25 AND age > 19
    -> ) AS temp
    -> GROUP BY age
    -> ORDER BY age;
+------+-------------------+
| age  | avg(calories)     |
+------+-------------------+
|   20 |             442.4 |
|   21 |           318.375 |
|   22 |               269 |
|   23 |               411 |
|   24 | 435.666666666667 |
+------+-------------------+
5 rows in set (0.00 sec)
```

Fig. 3 Advanced Query #1 Result

RESULT OF EXPLAIN ANALYZE:

RESULT of EXPLAIN ANALYZE without index:

```
| -> Sort: temp.age  (actual time=0.763..0.764 rows=5 loops=1)
    -> Table scan on <temporary>  (actual time=0.000..0.001 rows=5 loops=1)
        -> Aggregate using temporary table  (actual time=0.750..0.751 rows=5 loops=1)
            -> Nested loop inner join  (cost=130.15 rows=23) (actual time=0.079..0.721 rows=34 loops=1)
                -> Filter: ((`User`.gender = 'M') and (`User`.age < 25) and (`User`.age > 19))  (cost=122.10 rows=13) (actual time=0.056..0.532 rows=35 loops=1)
                    -> Table scan on User  (cost=122.10 rows=1201) (actual time=0.048..0.369 rows=1201 loops=1)
                -> Index lookup on GoalMadeByUser using userId (userId=`User`.userId)  (cost=0.44 rows=2) (actual time=0.005..0.005 rows=1 loops=35)
 |
```

Index on age:

```
| -> Group aggregate: avg(GoalMadeByUser.calories)  (cost=40.36 rows=13) (actual
time=0.379..0.593 rows=5 loops=1)
    -> Nested loop inner join  (cost=39.05 rows=13) (actual time=0.347..0.583 rows=34 loops=1)
        -> Filter: (`User`.gender = 'M')  (cost=34.46 rows=8) (actual time=0.315..0.337 rows=35
loops=1)
            -> Index range scan on User using age_idx, with index condition: ((`User`.age < 25) and
(`User`.age > 19))  (cost=34.46 rows=76) (actual time=0.311..0.325 rows=76 loops=1)
        -> Index lookup on GoalMadeByUser using userId (userId=`User`.userId)  (cost=0.45
rows=2) (actual time=0.006..0.007 rows=1 loops=35)
 |
```

Index on gender & age:

```
| -> Group aggregate: avg(GoalMadeByUser.calories)  (cost=63.92 rows=65) (actual
time=0.291..0.445 rows=5 loops=1)
    -> Nested loop inner join  (cost=57.37 rows=65) (actual time=0.258..0.435 rows=34 loops=1)
        -> Filter: (`User`.gender = 'M')  (cost=34.46 rows=38) (actual time=0.206..0.228 rows=35
loops=1)
            -> Index range scan on User using age_idx, with index condition: ((`User`.age < 25) and
(`User`.age > 19))  (cost=34.46 rows=76) (actual time=0.203..0.215 rows=76 loops=1)
        -> Index lookup on GoalMadeByUser using userId (userId=`User`.userId)  (cost=0.44
rows=2) (actual time=0.005..0.006 rows=1 loops=35)
 |
```

Index on gender only:

```
| -> Sort: temp.age  (actual time=1.446..1.446 rows=5 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.001 rows=5 loops=1)
        -> Aggregate using temporary table  (actual time=1.428..1.428 rows=5 loops=1)
            -> Nested loop inner join  (cost=52.89 rows=115) (actual time=0.238..1.390 rows=34
loops=1)
                -> Filter: ((`User`.age < 25) and (`User`.age > 19))  (cost=12.67 rows=67) (actual
time=0.212..1.200 rows=35 loops=1)
                    -> Index lookup on User using gender_idx (gender='M')  (cost=12.67 rows=600)
(actual time=0.207..1.149 rows=600 loops=1)
                -> Index lookup on GoalMadeByUser using userId (userId=`User`.userId)  (cost=0.43
rows=2) (actual time=0.005..0.005 rows=1 loops=35)
 |
```

Index Analysis report:

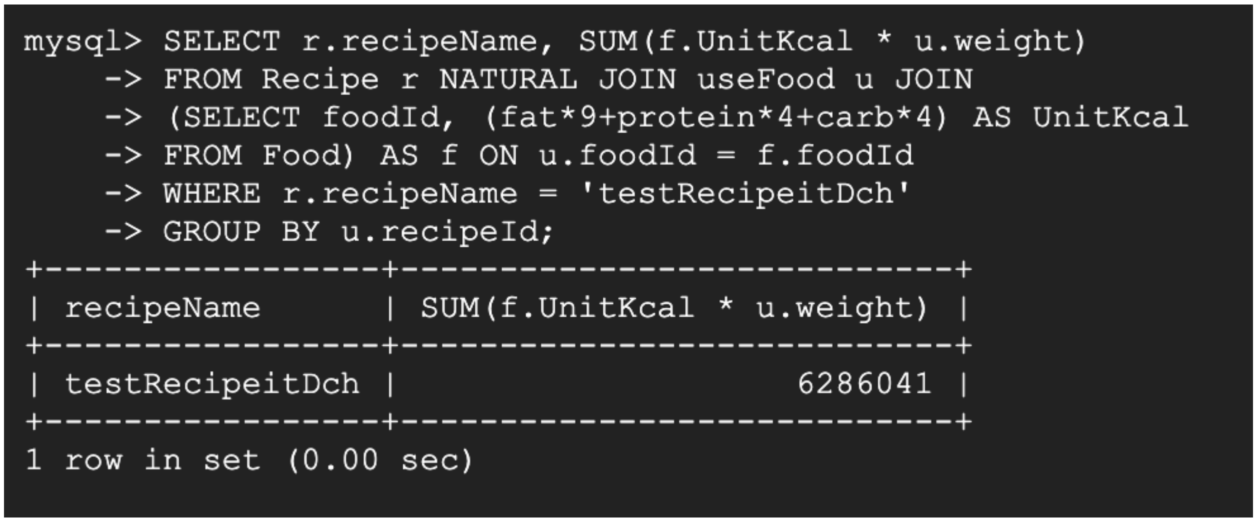
Here we tried three different indexes:

1. On User.age
2. On User.gender
3. On User.age & User.gender

The reason we choose these indexes is that we are using those attributes in WHERE and GROUP.
The result shows that indexing on User.age significantly reduces the cost but indexing on User.gender will have negative effects which in fact increase the cost. The reason behind these beliefs is that age is in range(1,80) while for gender we only have two different types 'M' and 'F'. So adding index file will help when we index on age for ages in the five years range but will increase the total cost when we index on gender since this won't save less on filter rows compared to the increased cost for going through index file.

Second Advanced Query:

Return the total calories of specific recipes:
SELECT r.recipeName, SUM(f.UnitKcal * u.weight)
FROM Recipe r NATURAL JOIN useFood u JOIN
(SELECT foodId, (fat*9+protein*4+carb*4) AS UnitKcal
FROM Food) AS f ON u.foodId = f.foodId
WHERE r.recipeName = 'testRecipeitDch'
GROUP BY u.recipeId;



```
mysql> SELECT r.recipeName, SUM(f.UnitKcal * u.weight)
    -> FROM Recipe r NATURAL JOIN useFood u JOIN
    -> (SELECT foodId, (fat*9+protein*4+carb*4) AS UnitKcal
    -> FROM Food) AS f ON u.foodId = f.foodId
    -> WHERE r.recipeName = 'testRecipeitDch'
    -> GROUP BY u.recipeId;
+-----------------+----------------------------+
| recipeName      | SUM(f.UnitKcal * u.weight) |
+-----------------+----------------------------+
| testRecipeitDch |                    6286041 |
+-----------------+----------------------------+
1 row in set (0.00 sec)
```

Fig. 4 Advanced Query #2 Result

RESULT OF EXPLAIN ANALYZE:

RESULT of EXPLAIN ANALYZE without index:

```
| -> Group aggregate: sum(((((Food.fat * 9) + (Food.protein * 4)) + (Food.carb * 4)) * u.weight))
(cost=49.75 rows=10) (actual time=0.166..0.166 rows=1 loops=1)
   -> Nested loop inner join  (cost=48.75 rows=10) (actual time=0.049..0.159 rows=5 loops=1)
      -> Nested loop inner join  (cost=45.25 rows=10) (actual time=0.043..0.141 rows=5
loops=1)
         -> Index scan on u using PRIMARY  (cost=10.25 rows=100) (actual time=0.029..0.042
rows=100 loops=1)
```

```
            -> Filter: (r.recipeName = 'testRecipeitDch')  (cost=0.25 rows=0) (actual
time=0.001..0.001 rows=0 loops=100)
                -> Single-row index lookup on r using PRIMARY (recipeId=u.recipeId)  (cost=0.25
rows=1) (actual time=0.000..0.001 rows=1 loops=100)
        -> Single-row index lookup on Food using PRIMARY (foodId=u.foodId)  (cost=0.26 rows=1)
(actual time=0.003..0.003 rows=1 loops=5)
 |
```

```
CREATE INDEX name_idx ON Recipe(recipeName);

| -> Table scan on <temporary>  (actual time=0.001..0.001 rows=1 loops=1)
    -> Aggregate using temporary table  (actual time=0.090..0.090 rows=1 loops=1)
        -> Nested loop inner join  (cost=3.35 rows=5) (actual time=0.027..0.064 rows=5 loops=1)
            -> Nested loop inner join  (cost=1.60 rows=5) (actual time=0.019..0.022 rows=5
loops=1)
                -> Index lookup on r using name_idx (recipeName='testRecipeitDch')  (cost=0.85
rows=1) (actual time=0.010..0.011 rows=1 loops=1)
                -> Index lookup on u using PRIMARY (recipeId=r.recipeId)  (cost=0.75 rows=5)
(actual time=0.008..0.011 rows=5 loops=1)
            -> Single-row index lookup on Food using PRIMARY (foodId=u.foodId)  (cost=0.27
rows=1) (actual time=0.008..0.008 rows=1 loops=5)
 |
```

```
CREATE INDEX fat_idx ON Food(fat)
| -> Group aggregate: sum(((((Food.fat * 9) + (Food.protein * 4)) + (Food.carb * 4)) * u.weight))
(cost=49.75 rows=10) (actual time=0.202..0.202 rows=1 loops=1)
    -> Nested loop inner join  (cost=48.75 rows=10) (actual time=0.067..0.194 rows=5 loops=1)
        -> Nested loop inner join  (cost=45.25 rows=10) (actual time=0.052..0.167 rows=5
loops=1)
            -> Index scan on u using PRIMARY  (cost=10.25 rows=100) (actual time=0.034..0.059
rows=100 loops=1)
            -> Filter: (r.recipeName = 'testRecipeitDch')  (cost=0.25 rows=0) (actual
time=0.001..0.001 rows=0 loops=100)
                -> Single-row index lookup on r using PRIMARY (recipeId=u.recipeId)  (cost=0.25
rows=1) (actual time=0.001..0.001 rows=1 loops=100)
        -> Single-row index lookup on Food using PRIMARY (foodId=u.foodId)  (cost=0.26 rows=1)
(actual time=0.005..0.005 rows=1 loops=5)
 |
```

```
CREATE INDEX weight_idx on useFood(weight);
| -> Group aggregate: sum(((((Food.fat * 9) + (Food.protein * 4)) + (Food.carb * 4)) * u.weight))
(cost=49.75 rows=10) (actual time=0.231..0.231 rows=1 loops=1)
    -> Nested loop inner join  (cost=48.75 rows=10) (actual time=0.069..0.224 rows=5 loops=1)
    -> Nested loop inner join  (cost=45.25 rows=10) (actual time=0.062..0.204 rows=5
loops=1)
        -> Index scan on u using PRIMARY  (cost=10.25 rows=100) (actual time=0.042..0.057
rows=100 loops=1)
        -> Filter: (r.recipeName = 'testRecipeitDch')  (cost=0.25 rows=0) (actual
time=0.001..0.001 rows=0 loops=100)
            -> Single-row index lookup on r using PRIMARY (recipeId=u.recipeId)  (cost=0.25
rows=1) (actual time=0.001..0.001 rows=1 loops=100)
    -> Single-row index lookup on Food using PRIMARY (foodId=u.foodId)  (cost=0.26 rows=1)
(actual time=0.004..0.004 rows=1 loops=5)
```

INDEX ANALYSIS:
Here we tried three different indexes:

1. Index on Food.fat
2. Index on useFood.weight
3. Index on Recipe.recipeName

We choose these attributes to index on since we are filtering rows using those attributes.
Index1 and index2 does not change the cost because the system prefers to use the PRIMARY key for those tables.
Index3 significantly reduces the cost because we are searching for a specific recipe name thus using index on recipeName will help the system quickly locate the right row to use.