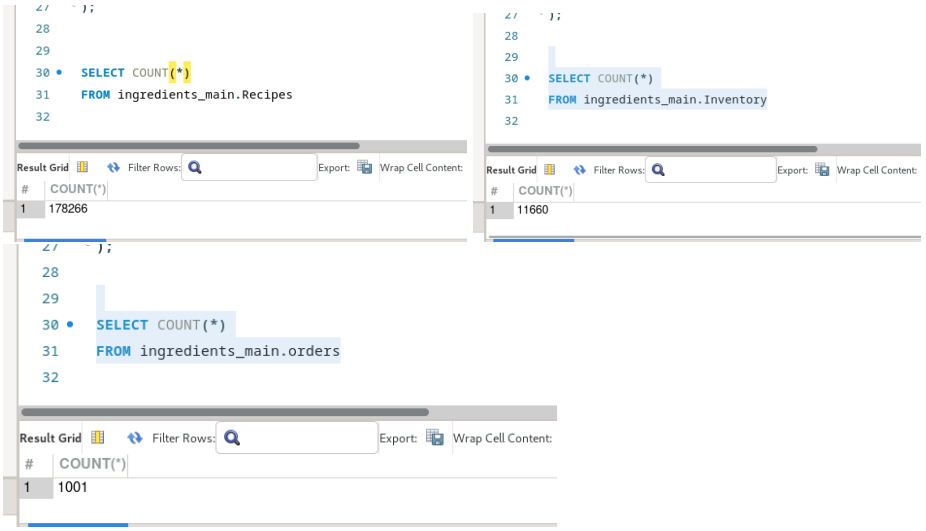
```
CREATE TABLE Recipes (
id  INT NOT NULL, name VARCHAR(160), steps VARCHAR(1000),
description VARCHAR(1000), serves TINYINT,
PRIMARY KEY(id));

CREATE TABLE Inventory (
id INT NOT NULL, name VARCHAR(160), simple_name VARCHAR(50),
isVeg BOOLEAN, isLactoseFree BOOLEAN, isGlutenFree BOOLEAN, isJain BOOLEAN,
MOQ INT, QtyAvailable INT, Price REAL, isVegan BOOLEAN, isHalal BOOLEAN,
PRIMARY KEY(id));

CREATE TABLE discount_coupons (
CouponID TINYINT NOT NULL, discpercent REAL,
PRIMARY KEY(id));

CREATE TABLE Users (
UserID INT NOT NULL, name VARCHAR(30), address VARCHAR(200),
CardNum CHAR(16), LoginPassword VARCHAR(64), isAdmin BOOLEAN,
PRIMARY KEY (UserID));

CREATE TABLE orders (
OrderID INT NOT NULL, TotalPrice REAL, CouponID TINYINT,
PRIMARY KEY (OrderID),
FOREIGN KEY (CouponID) REFERENCES discount_coupons(CouponID));

CREATE TABLE Rating (
UserID INT NOT NULL, RecipeID INT NOT NULL, Rating REAL, Comment VARCHAR(300),
PRIMARY KEY (UserID, RecipeID),
FOREIGN KEY (UserID) REFERENCES Users(UserID),
FOREIGN KEY (RecipeID) REFERENCES Recipes(id));

CREATE TABLE Ingredient (
id INT NOT NULL, ingredient_id INT NOT NULL, quantity INT,
PRIMARY KEY (id, ingredient_id),
FOREIGN KEY (id) REFERENCES Recipes(id) ON DELETE CASCADE,
FOREIGN KEY (ingredient_id) REFERENCES Inventory(id));
```



Query 1: *Returns the average rating of all recipes that serve a certain number of people (4).*

```
SELECT name, AVG(rt.Rating)
FROM Recipes rc JOIN Rating rt ON rc.id = rt.RecipeID
GROUP BY rc.name, rc.serves
HAVING rc.serves =4;
```

```
mysql> SELECT name, AVG(rt.Rating) FROM Recipes rc JOIN Rating rt ON rc.id = rt.RecipeID GROUP BY rc.name, rc.serves  HAVING rc.serves =4 LI
MIT 15;
+----------------------------------------+----------------+
| name                                   | AVG(rt.Rating) |
+----------------------------------------+----------------+
| linguine with wild mushroom alfredo    |              1 |
| chewy fudge drop cookies  healthy      |              1 |
| rice pilaf with glazed tomatoes        |              1 |
| cranberry cinnamon bread  bread machine|              1 |
| cheddar crumble apple pie              |              5 |
| island chicken packet                  |              1 |
| caramelized crust cinnalicious loaf cake |            5 |
| octoberfest german potato salad        |              5 |
| cherry crisp   from scratch            |              1 |
| faroe island   coffee with cardamom cream |           5 |
| rosemary s hanky panky s               |              3 |
| open faced vegetable sandwich          |            2.5 |
| flank steak with cilantro almond pesto |            2.5 |
| orange  yogurt                         |              3 |
| crockery bbq beef for sandwiches       |              3 |
+----------------------------------------+----------------+
15 rows in set (0.22 sec)
```

(a) EXLPAIN ANALYZE without indexing

```
mysql> EXPLAIN ANALYZE SELECT name, AVG(rt.Rating) FROM Recipes rc JOIN Rating rt ON rc.id = rt.RecipeID GROUP BY rc.name, rc.serves  HAVING
 rc.serves =4;
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| EXPLAIN                              |
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| -> Filter: (rc.serves = 4)  (actual time=0.736..0.754 rows=29 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.009 rows=77 loops=1)
        -> Aggregate using temporary table  (actual time=0.731..0.743 rows=77 loops=1)
            -> Nested loop inner join  (cost=92.64 rows=77) (actual time=0.070..0.588 rows=77 loops=1)
                -> Table scan on rt  (cost=7.95 rows=77) (actual time=0.049..0.068 rows=77 loops=1)
                -> Single-row index lookup on rc using PRIMARY (id=rt.RecipeID)  (cost=1.00 rows=1) (actual time=0.006..0.006 rows=1 loops=7
7)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------
```

(b) EXPLAIN ANALYZE after creating index on Rating(RecipeID) since it is a column used to JOIN => Result : virtually no change in time or costs since index lookup happens on another column and we still have to scan the table on Rating to get the results of JOIN.

```
mysql> CREATE INDEX RecipeID_idx ON Rating(RecipeID);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT name, AVG(rt.Rating) FROM Recipes rc JOIN Rating rt ON rc.id = rt.RecipeID GROUP BY rc.name, rc.serves  HAVING
 rc.serves =4;
+-----------------------------------------------------------------------------------------------------------------------------------------

mysql> EXPLAIN ANALYZE SELECT name, AVG(rt.Rating) FROM Recipes rc JOIN Rating rt ON rc.id = rt.RecipeID GROUP BY rc.name, rc.serves  HAVING
 rc.serves =4;
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| EXPLAIN                              |
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| -> Filter: (rc.serves = 4)  (actual time=0.440..0.455 rows=29 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.007 rows=77 loops=1)
        -> Aggregate using temporary table  (actual time=0.438..0.448 rows=77 loops=1)
            -> Nested loop inner join  (cost=92.64 rows=77) (actual time=0.053..0.327 rows=77 loops=1)
                -> Table scan on rt  (cost=7.95 rows=77) (actual time=0.038..0.049 rows=77 loops=1)
                -> Single-row index lookup on rc using PRIMARY (id=rt.RecipeID)  (cost=1.00 rows=1) (actual time=0.003..0.003 rows=1 loops=7
7)
 |
```

(c) Indexing Recipes(id) produces no change in costs as it is already indexed being a Primary Key. primary keys are already indexed by default during table creation. So, there is no need to read the table in drive & create a hash in the memory for joining.

```
mysql> CREATE INDEX id_idx ON Recipes(id);
Query OK, 0 rows affected (0.63 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE SELECT name, AVG(rt.Rating) FROM Recipes rc JOIN Rating rt ON rc.id = rt.RecipeID GROUP BY rc.name, rc.serves  HAVING
 rc.serves =4;
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| EXPLAIN



                       |
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------+
| -> Filter: (rc.serves = 4)  (actual time=0.421..0.436 rows=29 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.007 rows=77 loops=1)
        -> Aggregate using temporary table  (actual time=0.418..0.429 rows=77 loops=1)
            -> Nested loop inner join  (cost=92.64 rows=77) (actual time=0.050..0.313 rows=77 loops=1)
                -> Table scan on rt  (cost=7.95 rows=77) (actual time=0.035..0.045 rows=77 loops=1)
                -> Single-row index lookup on rc using PRIMARY (id=rt.RecipeID)  (cost=1.00 rows=1) (actual time=0.003..0.003 rows=1 loops=7
7)
 |
```

Query 2:

```
EXPLAIN ANALYZE
SELECT
   *
FROM
   ingredients_main.Recipes r
WHERE
   NOT EXISTS( SELECT
         *
      FROM
         ingredients_main.Ingredients ing
            JOIN
         ingredients_main.Inventory inv ON (ing.ingredient_id = inv.id)
      WHERE
         r.id = ing.id AND inv.isVeg = 0);
EXPLAIN ANALYZE
SELECT r.name
FROM ingredients_main.Recipes r
WHERE NOT EXISTS
(
SELECT *
FROM ingredients_main.Ingredients i JOIN ingredients_main.Inventory i2 ON (i.ingredient_id = i2.id)
WHERE i.id = r.id
AND i2.isVeg = 0
);
```

This is the result without indexing. We can see it is relatively fast and it is limited to 1000 rows.

-> Nested loop antijoin (cost=298974066853822.10 rows=2989740668118750)

(actual time=1620.147..1869.863 rows=120544 loops=1)

-> Table scan on r (cost=26082.14 rows=158650) (actual time=0.041..106.578 rows=178266 loops=1)

-> Single-row index lookup on <subquery2> using <auto_distinct_key> (id=ingredients_main.r.id) (actual time=0.001..0.001 rows=0 loops=178266)

-> Materialize with deduplication (cost=3768979136.83..3768979136.83 rows=18844882875) (actual time=1740.253..1740.253 rows=57722 loops=1)

-> Filter: (ingredients_main.i.id is not null) (cost=1884490849.33 rows=18844882875) (actual time=7.458..1426.899 rows=712641 loops=1)

-> Inner hash join (ingredients_main.i.ingredient_ids = ingredients_main.i2.id) (cost=1884490849.33 rows=18844882875) (actual time=7.456..1365.535 rows=712641 loops=1)

-> Table scan on i (cost=13.73 rows=1601775) (actual time=0.021..1067.550 rows=1605473 loops=1)

-> Hash

-> Filter: (ingredients_main.i2.isVeg = 0) (cost=1200.75 rows=11765) (actual time=0.011..7.218 rows=849 loops=1)

-> Table scan on i2 (cost=1200.75 rows=11765) (actual time=0.009..6.461 rows=11660 loops=1)-

This is with indexing on Ingredients(ingredient_id). Surprisingly the time to index has increased the query time by about ~1.5 times. This could possibly be a result of bad data and/or primary key problems.

> Nested loop antijoin (cost=44676380923.85 rows=446763389767) (actual time=2574.202..2818.783 rows=120544 loops=1)

-> Table scan on r (cost=26082.14 rows=158650) (actual time=0.383..105.373 rows=178266 loops=1)

-> Single-row index lookup on <subquery2> using <auto_distinct_key> (id=ingredients_main.r.id) (actual time=0.001..0.001 rows=0 loops=178266)

-> Materialize with deduplication (cost=1268414.90..1268414.90 rows=2816031) (actual time=2689.750..2689.750 rows=57722 loops=1)

-> Filter: (ingredients_main.i.id is not null) (cost=986811.76 rows=2816031) (actual time=14.347..2237.692 rows=712641 loops=1)

-> Nested loop inner join (cost=986811.76 rows=2816031) (actual time=14.345..2184.831 rows=712641 loops=1)

-> Filter: (ingredients_main.i2.isVeg = 0) (cost=1200.75 rows=11765) (actual time=0.030..9.669 rows=849 loops=1)

-> Table scan on i2 (cost=1200.75 rows=11765) (actual time=0.014..7.967 rows=11660 loops=1)

-> Index lookup on i using idx (ingredient_ids=ingredients_main.i2.id) (cost=59.84 rows=239) (actual time=0.702..2.502 rows=839 loops=849)

This is the result of indexing on Inventory(id) and it shows similar results as last time which is interesting. Again, the reasons for this could be indexing time and/or key conflicts. Technically the indexing time should be lower so there may be problems in the dataset leading to this, which can be resolved by more data filtering, processing and cleanups.

-> Nested loop antijoin (cost=37264664845.94 rows=372646228988) (actual time=17625.847..17866.586 rows=120544 loops=1)

-> Table scan on r (cost=26082.14 rows=158650) (actual time=0.058..103.591 rows=178266 loops=1)

-> Single-row index lookup on <subquery2> using <auto_distinct_key> (id=ingredients_main.r.id) (actual time=0.001..0.001 rows=0 loops=178266)

-> Materialize with deduplication (cost=1218164.84..1218164.84 rows=2348857) (actual time=17740.036..17740.036 rows=57722 loops=1)

-> Filter: (ingredients_main.i.id is not null) (cost=983279.10 rows=2348857) (actual time=0.038..17423.010 rows=712641 loops=1)

-> Nested loop inner join (cost=983279.10 rows=2348857) (actual time=0.037..17367.446 rows=712641 loops=1)

-> Table scan on i (cost=161179.00 rows=1601775) (actual time=0.012..860.498 rows=1605473 loops=1)

-> Filter: (ingredients_main.i2.isVeg = 0) (cost=0.37 rows=1) (actual time=0.009..0.010 rows=0 loops=1605473)

-> Index lookup on i2 using idx2 (id=ingredients_main.i.ingredient_ids) (cost=0.37 rows=1) (actual time=0.002..0.010 rows=6 loops=1605473)