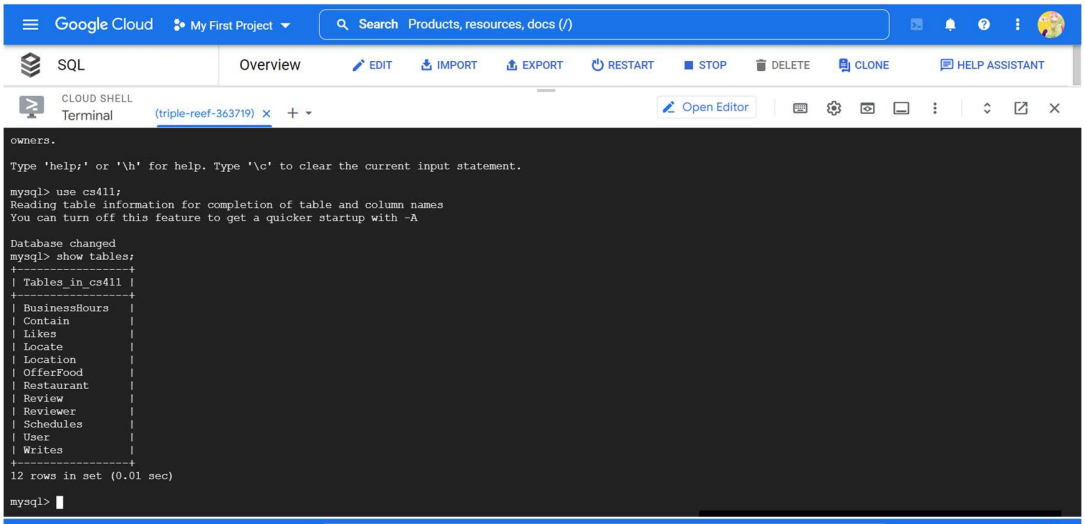
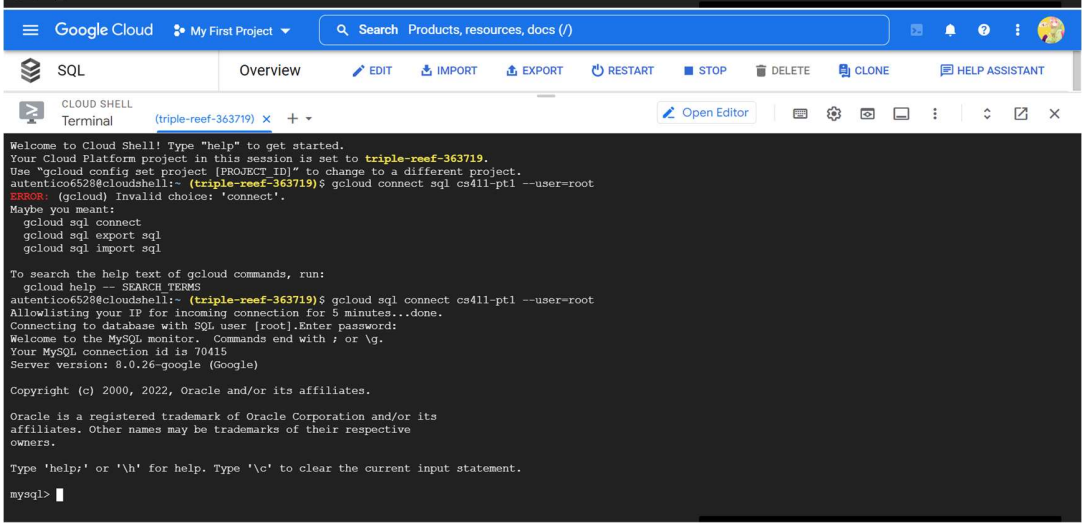
2.1 provide a screenshot of the connection





2.2 providing the DDL commands for your tables.

```
DROP TABLE IF EXISTS Locate;
DROP TABLE IF EXISTS BusinessHours;
DROP TABLE IF EXISTS Likes;
DROP TABLE IF EXISTS Writes;
DROP TABLE IF EXISTS Contain;
DROP TABLE IF EXISTS OfferFood;
DROP TABLE IF EXISTS Location;
DROP TABLE IF EXISTS Restaurant;
DROP TABLE IF EXISTS Schedule;
DROP TABLE IF EXISTS Reviewer;
DROP TABLE IF EXISTS Review;
DROP TABLE IF EXISTS User;
```

-- entities

```
CREATE TABLE Location (
    LocationID int Primary Key,
    Address VARCHAR(255),
    Street VARCHAR(255),
    City VARCHAR(255),
    State VARCHAR(30),
    Zipcode int
);

CREATE TABLE Schedule(
    ScheduleID int Primary key,
    DayOfWeek VARCHAR(10),
    OpenTime time,
    OpeningDuration int
);

CREATE TABLE Restaurant (
    RestaurantID int Primary key,
    Name VARCHAR(255),
    PhoneNumber VARCHAR(255),
    PriceLevel int,
    AverageRating real,
    Cuisine VARCHAR(255),
    Delivery bool,
    Dine_In bool,
    RatingNum int
);

CREATE TABLE User (
    UserID INT Primary key,
    UserName VARCHAR(40) NOT NULL,
    Password VARCHAR(30) NOT NULL,
    Email VARCHAR(50)
);

CREATE TABLE Reviewer (
    ReviewerID int primary key,
    Name VARCHAR(255)
);

CREATE TABLE Review (
    ReviewID int,
    Rating int NOT NULL,
    Comment text,
    ReviewDate timestamp,
    PRIMARY KEY (ReviewID)
);

-- Weak entities
CREATE TABLE OfferFood (
    FoodID int NOT NULL,
```

```
    FoodName VARCHAR(255) NOT NULL,
    RestaurantId int NOT NULL,
    Price ,
    PRIMARY KEY (FoodID, RestaurantId),
      FOREIGN KEY (RestaurantId) REFERENCES Restaurant(RestaurantId) ON DELETE
CASCADE ON UPDATE CASCADE
);


-- relationships
CREATE TABLE Locate (
    RestaurantID int,
    LocationID int,
    PRIMARY KEY (RestaurantID),
    FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID),
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

CREATE TABLE BusinessHours(
    RestaurantID int NOT NULL,
    ScheduleID int,
    PRIMARY KEY (RestaurantID, ScheduleID),
    FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID),
    FOREIGN KEY (ScheduleID) REFERENCES Schedule(ScheduleID)
);

CREATE TABLE Likes(
    UserID int,
    RestaurantID int,
    LikeTime timestamp,
    Notes text,
    PRIMARY KEY (UserID, RestaurantID),
    FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID),
    FOREIGN KEY (UserID) REFERENCES User(UserID)

);

CREATE TABLE Writes(
    ReviewerID int NOT NULL,
    ReviewID int,
    PRIMARY KEY(ReviewID),
    FOREIGN KEY (ReviewerID) REFERENCES Reviewer(ReviewerID),
    FOREIGN KEY (ReviewID) REFERENCES Review(ReviewID)
);

CREATE TABLE Contain(
    RestaurantID int NOT NULL,
    ReviewID int,
    PRIMARY KEY(ReviewID),
    FOREIGN KEY (ReviewID) REFERENCES Review(ReviewID),
    FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID)
```

);

2.3 inserting at least 1000 rows in the tables.

```
mysql> SELECT COUNT(*) FROM OfferFood;
+----------+
| COUNT(*) |
+----------+
|    87059 |
+----------+
1 row in set (0.02 sec)
```

```
mysql> SELECT COUNT(*) FROM Restaurant;
+----------+
| COUNT(*) |
+----------+
|     1182 |
+----------+
1 row in set (0.03 sec)
```

```
mysql> SELECT COUNT(*) FROM Review;
+----------+
| COUNT(*) |
+----------+
|     5706 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Reviewer;
+----------+
| COUNT(*) |
+----------+
|     4292 |
+----------+
1 row in set (0.00 sec)
```

3 and 4
Advanced Query #1

```
SELECT FoodName, ROUND(AVG(AverageRating), 2) as AVGRating
FROM OfferFood NATURAL JOIN Restaurant NATURAL JOIN Locate NATURAL JOIN Location
WHERE Zipcode = 60660
GROUP BY FoodName
ORDER BY AVGRating DESC;
```

```
1  use cs411;
2  SELECT FoodName, ROUND(AVG(AverageRating), 2) as AVGRating
3  FROM OfferFood NATURAL JOIN Restaurant NATURAL JOIN Locate NATURAL
4  JOIN Location
5  WHERE Zipcode = 60660
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

FoodName	AVGRating
Bean Thread Noodle Soup	4.7
6 Pot Sticker	4.7
6 Piece Shrimp Rolls	4.7
8 Piece Shumai	4.7
Tom Kha Vegetable Soup	4.7
Bean Curd Soup	4.7
Pineapple smoothie	4.7
4 Piece Egg Rolls	4.7
Tom Kha Tofu Soup	4.7
6 Piece Crab Rangoon	4.7
Tom Yum Vegetable Soup	4.7
Tom Yum Tofu Soup	4.7
3 Piece Chive Dumpling	4.7
Chicken Satay	4.7
Fish Cake	4.7

Result Grid | Form Editor | Field Types | Query Stats

Default

-> Sort: round(avg(Restaurant.AverageRating),2) DESC (actual time=6.717..6.855 rows=1643 loops=1)
    -> Table scan on <temporary> (actual time=0.001..0.116 rows=1643 loops=1)
        -> Aggregate using temporary table (actual time=6.105..6.340 rows=1643 loops=1)
            -> Nested loop inner join (cost=4305.85 rows=11721) (actual time=1.009..4.443 rows=1840 loops=1)
                -> Nested loop inner join (cost=203.44 rows=118) (actual time=0.674..1.154 rows=23 loops=1)
                    -> Nested loop inner join (cost=162.07 rows=118) (actual time=0.666..1.099 rows=23 loops=1)
                        -> Filter: (Location.Zipcode = 60660) (cost=120.70 rows=118) (actual time=0.654..1.028 rows=23 loops=1)
                            -> Table scan on Location (cost=120.70 rows=1182) (actual time=0.649..0.952 rows=1182 loops=1)
                        -> Index lookup on Locate using LocationID (LocationID=Location.LocationID) (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=23)
                    -> Single-row index lookup on Restaurant using PRIMARY (RestaurantID=Locate.RestaurantID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=23)
                -> Index lookup on OfferFood using RestaurantId (RestaurantId=Locate.RestaurantID) (cost=24.87 rows=99) (actual time=0.098..0.137 rows=80 loops=23)

0.031 seconds

Index on food name:
CREATE INDEX food_name on OfferFood(FoodName);

-> Sort: round(avg(Restaurant.AverageRating),2) DESC (actual time=7.795..7.929 rows=1643 loops=1)
    -> Table scan on <temporary> (actual time=0.001..0.164 rows=1643 loops=1)
        -> Aggregate using temporary table (actual time=7.121..7.377 rows=1643 loops=1)

```
        -> Nested loop inner join  (cost=4305.85 rows=11721) (actual time=0.471..4.953 rows=1840
loops=1)
            -> Nested loop inner join  (cost=203.44 rows=118) (actual time=0.085..0.711 rows=23 loops=1)
                -> Nested loop inner join  (cost=162.07 rows=118) (actual time=0.076..0.624 rows=23
loops=1)
                    -> Filter: (Location.Zipcode = 60660)  (cost=120.70 rows=118) (actual time=0.060..0.490
rows=23 loops=1)
                        -> Table scan on Location  (cost=120.70 rows=1182) (actual time=0.057..0.412
rows=1182 loops=1)
                    -> Index lookup on Locate using LocationID (LocationID=Location.LocationID)  (cost=0.25
rows=1) (actual time=0.004..0.005 rows=1 loops=23)
                -> Single-row index lookup on Restaurant using PRIMARY
(RestaurantID=Locate.RestaurantID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=23)
            -> Index lookup on OfferFood using RestaurantId (RestaurantId=Locate.RestaurantID)
(cost=24.87 rows=99) (actual time=0.129..0.178 rows=80 loops=23)

0.033 seconds

Index on zip code:
CREATE INDEX z_code ON Location(Zipcode);

-> Sort: round(avg(Restaurant.AverageRating),2) DESC  (actual time=7.881..7.999 rows=1643 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.139 rows=1643 loops=1)
        -> Aggregate using temporary table  (actual time=7.243..7.474 rows=1643 loops=1)
            -> Nested loop inner join  (cost=817.31 rows=2281) (actual time=0.536..5.454 rows=1840
loops=1)
                -> Nested loop inner join  (cost=19.04 rows=23) (actual time=0.045..0.238 rows=23 loops=1)
                    -> Nested loop inner join  (cost=10.99 rows=23) (actual time=0.029..0.127 rows=23 loops=1)
                        -> Index lookup on Location using z_code (Zipcode=60660)  (cost=2.94 rows=23) (actual
time=0.015..0.025 rows=23 loops=1)
                    -> Index lookup on Locate using LocationID (LocationID=Location.LocationID)  (cost=0.25
rows=1) (actual time=0.003..0.004 rows=1 loops=23)
                -> Single-row index lookup on Restaurant using PRIMARY
(RestaurantID=Locate.RestaurantID)  (cost=0.25 rows=1) (actual time=0.004..0.005 rows=1 loops=23)
            -> Index lookup on OfferFood using RestaurantId (RestaurantId=Locate.RestaurantID)
(cost=25.22 rows=99) (actual time=0.166..0.221 rows=80 loops=23)

0.033 seconds


Both:
CREATE INDEX z_code ON Location(Zipcode);
CREATE INDEX food_name on OfferFood(FoodName);

-> Sort: round(avg(Restaurant.AverageRating),2) DESC  (actual time=6.501..6.632 rows=1643 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.134 rows=1643 loops=1)
        -> Aggregate using temporary table  (actual time=5.821..6.063 rows=1643 loops=1)
            -> Nested loop inner join  (cost=817.31 rows=2281) (actual time=0.347..3.984 rows=1840
loops=1)
```

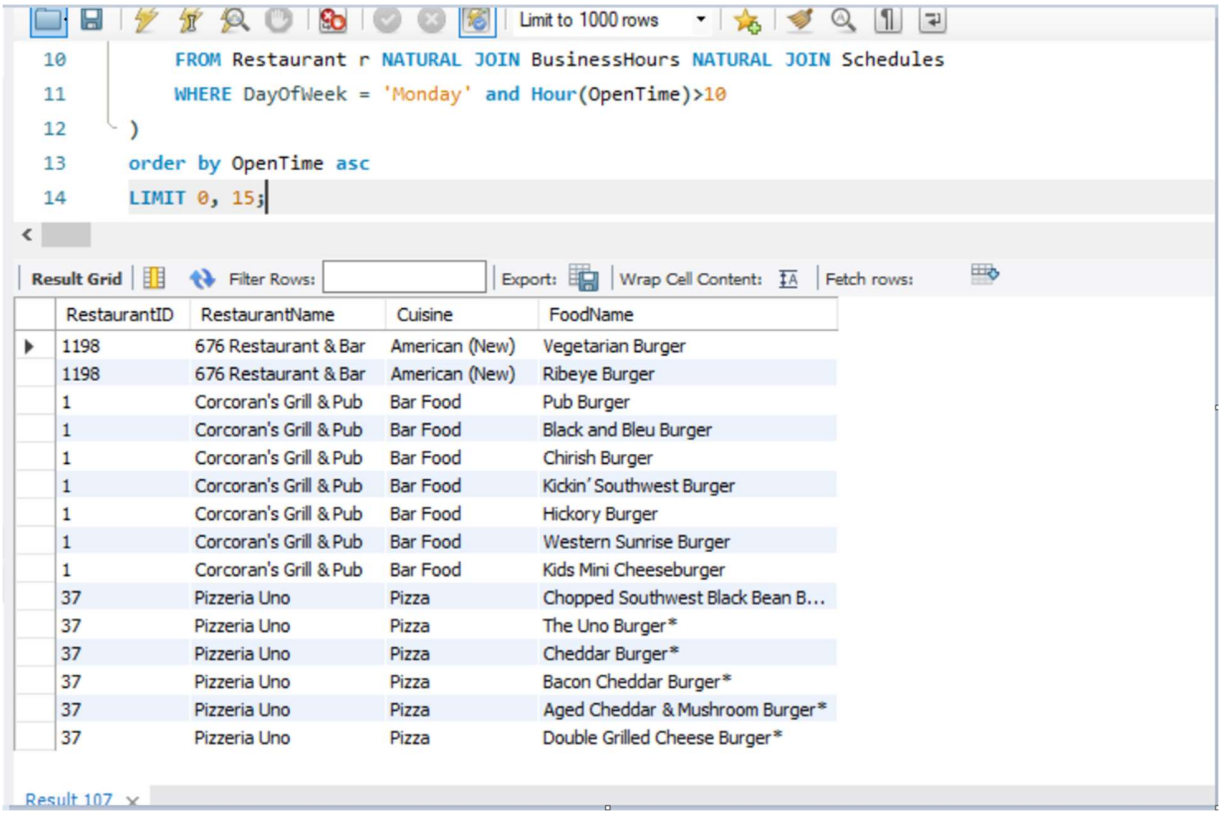
-> Nested loop inner join (cost=19.04 rows=23) (actual time=0.041..0.201 rows=23 loops=1)
-> Nested loop inner join (cost=10.99 rows=23) (actual time=0.031..0.122 rows=23 loops=1)
-> Index lookup on Location using z_code (Zipcode=60660) (cost=2.94 rows=23) (actual time=0.020..0.030 rows=23 loops=1)
-> Index lookup on Locate using LocationID (LocationID=Location.LocationID) (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=23)
-> Single-row index lookup on Restaurant using PRIMARY (RestaurantID=Locate.RestaurantID) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=23)
-> Index lookup on OfferFood using RestaurantId (RestaurantId=Locate.RestaurantID) (cost=25.22 rows=99) (actual time=0.117..0.157 rows=80 loops=23)

0.035 seconds

We used the index method that used both the FoodName and the Zipcode indexes. Although there is almost no actual time difference between the different indexing strategies, we believe it is because the speed of the query is already quite fast. A query that executes in ~0.03 seconds cannot be optimized much further. By using the Zipcode index it changes our location scan from being a full table scan to an index scan, which should increase speed. We also index by FoodName because it is the variable used to group in the final step of the query. When this query was run multiple times with EXPLAIN ANALYZE, we had varying results for the speed. These mySQL queries can have inconsistencies for any number of reasons.
You can find it through the red part, but the statement using index really improves the speed of the statement. The time of other parts may vary with each execution, resulting in a decrease in the total time. we will choose to use both indices.

Advanced Query #2

```
SELECT RestaurantID, RestaurantName, Cuisine, FoodName
FROM  OfferFood o NATURAL JOIN Restaurant r NATURAL JOIN BusinessHours NATURAL JOIN
Schedules s1
WHERE (o.FoodName like "%burger%" or r.RestaurantName like "%burger%" )
AND DayOfWeek = 'Monday'
and RestaurantID in
(
        SELECT RestaurantID
        FROM Restaurant r NATURAL JOIN BusinessHours NATURAL JOIN Schedules
        WHERE DayOfWeek = 'Tuesday' and Hour(OpenTime)>10
)
order by OpenTime asc;
```



RestaurantID	RestaurantName	Cuisine	FoodName
1198	676 Restaurant & Bar	American (New)	Vegetarian Burger
1198	676 Restaurant & Bar	American (New)	Ribeye Burger
1	Corcoran's Grill & Pub	Bar Food	Pub Burger
1	Corcoran's Grill & Pub	Bar Food	Black and Bleu Burger
1	Corcoran's Grill & Pub	Bar Food	Chirish Burger
1	Corcoran's Grill & Pub	Bar Food	Kickin' Southwest Burger
1	Corcoran's Grill & Pub	Bar Food	Hickory Burger
1	Corcoran's Grill & Pub	Bar Food	Western Sunrise Burger
1	Corcoran's Grill & Pub	Bar Food	Kids Mini Cheeseburger
37	Pizzeria Uno	Pizza	Chopped Southwest Black Bean B...
37	Pizzeria Uno	Pizza	The Uno Burger*
37	Pizzeria Uno	Pizza	Cheddar Burger*
37	Pizzeria Uno	Pizza	Bacon Cheddar Burger*
37	Pizzeria Uno	Pizza	Aged Cheddar & Mushroom Burger*
37	Pizzeria Uno	Pizza	Double Grilled Cheese Burger*

Default, with no index added

```
-> Nested loop inner join  (cost=20565.52 rows=56134) (actual time=0.371..102.218 rows=1685 loops=1)
    -> Nested loop inner join  (cost=918.74 rows=566) (actual time=0.044..5.136 rows=760 loops=1)
        -> Nested loop inner join  (cost=720.62 rows=566) (actual time=0.036..3.742 rows=760 loops=1)
            -> Filter: (s1.DayOfWeek = 'Monday')  (cost=531.95 rows=528) (actual time=0.028..1.894
rows=703 loops=1)
                -> Table scan on s1  (cost=531.95 rows=5282) (actual time=0.025..1.387 rows=5459 loops=1)
```

-> Index lookup on BusinessHours using ScheduleID (ScheduleID=s1.ScheduleID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=703)
-> Single-row index lookup on r using PRIMARY (RestaurantID=BusinessHours.RestaurantID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=760)
-> Filter: ((o.FoodName like '%burger%') or (r.RestaurantName like '%burger%')) (cost=24.81 rows=99) (actual time=0.107..0.127 rows=2 loops=760)
-> Index lookup on o using RestaurantId (RestaurantId=BusinessHours.RestaurantID) (cost=24.81 rows=99) (actual time=0.062..0.095 rows=73 loops=760)

0.140 seconds

Create index indexfood on OfferFood(FoodName);

-> Nested loop inner join (cost=20565.52 rows=56134) (actual time=0.337..99.936 rows=1685 loops=1)
-> Nested loop inner join (cost=918.74 rows=566) (actual time=0.063..4.997 rows=760 loops=1)
-> Nested loop inner join (cost=720.62 rows=566) (actual time=0.054..3.641 rows=760 loops=1)
-> Filter: (s1.DayOfWeek = 'Monday') (cost=531.95 rows=528) (actual time=0.040..1.956 rows=703 loops=1)
-> Table scan on s1 (cost=531.95 rows=5282) (actual time=0.038..1.421 rows=5459 loops=1)
-> Index lookup on BusinessHours using ScheduleID (ScheduleID=s1.ScheduleID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=703)
-> Single-row index lookup on r using PRIMARY (RestaurantID=BusinessHours.RestaurantID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=760)
-> Filter: ((o.FoodName like '%burger%') or (r.RestaurantName like '%burger%')) (cost=24.81 rows=99) (actual time=0.104..0.125 rows=2 loops=760)
-> Index lookup on o using RestaurantId (RestaurantId=BusinessHours.RestaurantID) (cost=24.81 rows=99) (actual time=0.060..0.093 rows=73 loops=760)

0.135 seconds

Also index on CREATE INDEX day_idx on Schedules(DayOfWeek);

-> Nested loop inner join (cost=26737.40 rows=74710) (actual time=0.299..106.956 rows=1685 loops=1)
-> Nested loop inner join (cost=588.82 rows=753) (actual time=0.052..3.772 rows=760 loops=1)
-> Nested loop inner join (cost=325.12 rows=753) (actual time=0.044..2.246 rows=760 loops=1)
-> Index lookup on s1 using day_idx (DayOfWeek='Monday') (cost=74.02 rows=703) (actual time=0.034..0.321 rows=703 loops=1)
-> Index lookup on BusinessHours using ScheduleID (ScheduleID=s1.ScheduleID) (cost=0.25 rows=1) (actual time=0.002..0.003 rows=1 loops=703)
-> Single-row index lookup on r using PRIMARY (RestaurantID=BusinessHours.RestaurantID) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=760)
-> Filter: ((o.FoodName like '%burger%') or (r.RestaurantName like '%burger%')) (cost=24.80 rows=99) (actual time=0.114..0.135 rows=2 loops=760)
-> Index lookup on o using RestaurantId (RestaurantId=BusinessHours.RestaurantID) (cost=24.80 rows=99) (actual time=0.067..0.102 rows=73 loops=760)

0.143 seconds

We find that using index really improves the speed of retrieval. It is obvious that using indexfood improves the speed, because the total time is 0.135<0.140. We can also see from a specific statement that the cost of the sentence is reduced from scan table to index.

As for why using the schedule (DayOfWeek) index increases the time.

According to the part marked in red, we can see that the statement uses index using day_ Idx and its cost time is shorter. “0.034..0.321 with index” compare with “0.038..1.421 without index”

The problem is the ‘nested loop inner join’. It seems that each execution will take different time, because the time consumed is accidental. it seems that adding the index day_ Idx increases the time instead.

we will choose to use both indices.