

```
Create Table Recommendation(  
  Id INT,  
  ratingScore INT,  
  userId INT,  
  restaurantName Varchar(255),  
  restaurantId INT,  
  Primary Key(Id),  
  Foreign Key(userId) References User(Id),  
  Foreign Key(restaurantId) References Restaurant(restaurantId)  
);
```

```
Create Table RankingPriority(  
  userId INT,  
  foodTypeRank INT,  
  priceRank INT,  
  locationRank INT,  
  isOpenRank INT,  
  Foreign Key(userId) References User(Id)  
);
```

```
Create Table Restaurant(  
  restaurantName Varchar(255),  
  Image Varchar(255),  
  foodType Varchar(255),  
  averagePrice INT,  
  Address Varchar(255),  
  restaurantId INT,  
  whenOpen INT,  
  whenClosed INT,  
  Primary Key(restaurantId)  
);
```

```
Create Table Review(  
  Id INT,  
  reviewText Varchar(255),  
  userId INT,  
  Date Varchar(255),  
  Primary Key(Id),  
  Foreign Key (userId) References User(Id)  
);
```

```
Create Table User(  
  ID INT,  
  userName Varchar(30),
```

Location Varchar(255),  
Primary Key (ID)  
);

We inserted 1000 values into each table with randomly generated data because we were running short on time due to difficulties with group assignments. Below is the screenshot of the resulting entries:

✓	13	21:28:29	use group51	0 row(s) affected	0.047 sec
✓	14	21:28:29	show tables	5 row(s) returned	0.078 sec / 0.000 sec
✓	15	21:28:29	describe RankingPriority	5 row(s) returned	0.063 sec / 0.000 sec
✓	16	21:28:29	describe Recommendation	5 row(s) returned	0.062 sec / 0.000 sec
✓	17	21:28:29	describe Restaurant	8 row(s) returned	0.063 sec / 0.000 sec
✓	18	21:28:30	describe Review	4 row(s) returned	0.047 sec / 0.000 sec
✓	19	21:28:30	describe User	3 row(s) returned	0.062 sec / 0.000 sec
✓	20	21:28:30	SELECT * FROM RankingPriority LIMIT 2000	1000 row(s) returned	0.063 sec / 0.000 sec
✓	21	21:28:30	SELECT * FROM Recommendation LIMIT 2000	1000 row(s) returned	0.047 sec / 0.000 sec
✓	22	21:28:30	SELECT * FROM Restaurant LIMIT 2000	1000 row(s) returned	0.047 sec / 0.016 sec
✓	23	21:28:30	SELECT * FROM Review LIMIT 2000	1000 row(s) returned	0.078 sec / 0.000 sec
✓	24	21:28:30	SELECT * FROM User LIMIT 2000	1000 row(s) returned	0.062 sec / 0.016 sec

We created 2 sql queries, the first collects the average score of all reviews for a specific restaurant type. The second provides the user id and restaurant for all user reviews where food type is prioritized highest.

```
SELECT foodType, avg(ratingScore) as avgScore
FROM Restaurant
JOIN Recommendation
ON Restaurant.restaurantId = Recommendation.restaurantId
GROUP BY foodType
ORDER BY avgScore LIMIT 15;
```

Result Grid			Filter Rows:
	foodType	avgScore	
▶	American	39.1613	
	Malaysian	46.3934	
	Thai	48.0382	
	Korean	48.7605	
	Japanese	49.5217	
	French	50.8571	
	Chinese	52.2547	

Result 8 x

```

SELECT Recommendation.userId, restaurantName
FROM Recommendation
JOIN RankingPriority
ON Recommendation.userId = RankingPriority.userId
Where foodTypeRank = 1
AND Recommendation.userId IN
(SELECT userId
From Review);
LIMIT 15;

```

Result Grid			Filter Rows:
	userId	restaurantName	
▶	10000	minor	
	10002	quota	
	10003	promote	
	10008	witness	
	10019	brain	
	10020	coup	
	10025	amuse	
	10040	blast	
	10042	poetry	
	10045	well	
	10052	thought	
	10055	pollution	
	10056	response	
	10062	flight	
	10066	farmer	

First Query Original:

-> Limit: 15 row(s) (cost=171.25 rows=15) (actual time=0.061..0.182 rows=15 loops=1)  
 -> Nested loop semijoin (cost=171.25 rows=100) (actual time=0.060..0.180 rows=15 loops=1)  
 -> Nested loop inner join (cost=136.25 rows=100) (actual time=0.052..0.146 rows=15 loops=1)  
 -> Filter: ((RankingPriority.foodTypeRank = 1) and (RankingPriority.userId is not null)) (cost=101.25 rows=100) (actual time=0.029..0.073 rows=15 loops=1)  
 -> Table scan on RankingPriority (cost=101.25 rows=1000) (actual time=0.026..0.062 rows=67 loops=1)  
 -> Index lookup on Recommendation using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.004..0.005 rows=1 loops=15)  
 -> Index lookup on Review using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)

New Index on RankingPriority Column FoodTypeRank:

---

-> Limit: 15 row(s) (actual time=3.098..3.100 rows=7 loops=1)  
 -> Sort: avgScore, limit input to 15 row(s) per chunk (actual time=3.097..3.098 rows=7 loops=1)  
 -> Table scan on <temporary> (actual time=0.001..0.002 rows=7 loops=1)  
 -> Aggregate using temporary table (actual time=3.077..3.078 rows=7 loops=1)  
 -> Nested loop inner join (cost=451.25 rows=1000) (actual time=0.066..2.240 rows=1000 loops=1)

-> Filter: (Recommendation.restaurantId is not null) (cost=101.25 rows=1000) (actual time=0.053..0.478 rows=1000 loops=1)  
 -> Table scan on Recommendation (cost=101.25 rows=1000) (actual time=0.051..0.388 rows=1000 loops=1)  
 -> Single-row index lookup on Restaurant using PRIMARY (restaurantId=Recommendation.restaurantId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)  
 ---

New Index on Recommendation Column ratingScore:

-> Limit: 15 row(s) (actual time=3.970..3.972 rows=7 loops=1)  
 -> Sort: avgScore, limit input to 15 row(s) per chunk (actual time=3.969..3.970 rows=7 loops=1)  
 -> Table scan on <temporary> (actual time=0.002..0.004 rows=7 loops=1)  
 -> Aggregate using temporary table (actual time=3.936..3.939 rows=7 loops=1)  
 -> Nested loop inner join (cost=451.25 rows=1000) (actual time=0.069..2.801 rows=1000 loops=1)  
 -> Filter: (Recommendation.restaurantId is not null) (cost=101.25 rows=1000) (actual time=0.056..0.632 rows=1000 loops=1)  
 -> Table scan on Recommendation (cost=101.25 rows=1000) (actual time=0.055..0.505 rows=1000 loops=1)  
 -> Single-row index lookup on Restaurant using PRIMARY (restaurantId=Recommendation.restaurantId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1000)

New Index on Recommendation Column ratingScore and RankingPriority Column FoodTypeRank:

-> Limit: 15 row(s) (actual time=3.171..3.173 rows=7 loops=1)  
 -> Sort: avgScore, limit input to 15 row(s) per chunk (actual time=3.170..3.171 rows=7 loops=1)  
 -> Table scan on <temporary> (actual time=0.001..0.002 rows=7 loops=1)  
 -> Aggregate using temporary table (actual time=3.146..3.148 rows=7 loops=1)  
 -> Nested loop inner join (cost=451.25 rows=1000) (actual time=0.061..2.271 rows=1000 loops=1)  
 -> Filter: (Recommendation.restaurantId is not null) (cost=101.25 rows=1000) (actual time=0.048..0.479 rows=1000 loops=1)  
 -> Table scan on Recommendation (cost=101.25 rows=1000) (actual time=0.047..0.383 rows=1000 loops=1)  
 -> Single-row index lookup on Restaurant using PRIMARY (restaurantId=Recommendation.restaurantId) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=1000)

#### Second Query Original:

- > Limit: 15 row(s) (actual time=3.099..3.101 rows=7 loops=1)
  - > Sort: avgScore, limit input to 15 row(s) per chunk (actual time=3.099..3.100 rows=7 loops=1)
    - > Table scan on <temporary> (actual time=0.001..0.001 rows=7 loops=1)
      - > Aggregate using temporary table (actual time=3.078..3.079 rows=7 loops=1)
        - > Nested loop inner join (cost=451.25 rows=1000) (actual time=0.065..2.250 rows=1000 loops=1)
          - > Filter: (Recommendation.restaurantId is not null) (cost=101.25 rows=1000) (actual time=0.051..0.502 rows=1000 loops=1)
            - > Table scan on Recommendation (cost=101.25 rows=1000) (actual time=0.050..0.401 rows=1000 loops=1)
              - > Single-row index lookup on Restaurant using PRIMARY (restaurantId=Recommendation.restaurantId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)

#### New Index on RankingPriority Column FoodTypeRank:

- > Limit: 15 row(s) (cost=194.95 rows=15) (actual time=0.049..0.154 rows=15 loops=1)
  - > Nested loop semijoin (cost=194.95 rows=239) (actual time=0.048..0.152 rows=15 loops=1)
    - > Nested loop inner join (cost=111.30 rows=239) (actual time=0.040..0.118 rows=15 loops=1)
      - > Filter: (RankingPriority.userId is not null) (cost=27.65 rows=239) (actual time=0.022..0.050 rows=15 loops=1)
        - > Index lookup on RankingPriority using idx\_RankingPriority\_foodTypeRank (foodTypeRank=1) (cost=27.65 rows=239) (actual time=0.021..0.047 rows=15 loops=1)
          - > Index lookup on Recommendation using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
            - > Index lookup on Review using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)

#### New Index on Recommendation Column RestaurantName:

- > Limit: 15 row(s) (cost=171.25 rows=15) (actual time=0.058..0.205 rows=15 loops=1)
  - > Nested loop semijoin (cost=171.25 rows=100) (actual time=0.058..0.203 rows=15 loops=1)
    - > Nested loop inner join (cost=136.25 rows=100) (actual time=0.050..0.143 rows=15 loops=1)
      - > Filter: ((RankingPriority.foodTypeRank = 1) and (RankingPriority.userId is not null)) (cost=101.25 rows=100) (actual time=0.028..0.072 rows=15 loops=1)

-> Table scan on RankingPriority (cost=101.25 rows=1000) (actual time=0.026..0.062 rows=67 loops=1)

-> Index lookup on Recommendation using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)

-> Index lookup on Review using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)

New Index on RankingPriority Column FoodTypeRank and New Index on Recommendation Column RestaurantName:

-> Limit: 15 row(s) (cost=194.95 rows=15) (actual time=0.051..0.152 rows=15 loops=1)

-> Nested loop semijoin (cost=194.95 rows=239) (actual time=0.050..0.150 rows=15 loops=1)

-> Nested loop inner join (cost=111.30 rows=239) (actual time=0.041..0.116 rows=15 loops=1)

-> Filter: (RankingPriority.userId is not null) (cost=27.65 rows=239) (actual time=0.021..0.046 rows=15 loops=1)

-> Index lookup on RankingPriority using idx\_RankingPriority\_foodTypeRank (foodTypeRank=1) (cost=27.65 rows=239) (actual time=0.020..0.043 rows=15 loops=1)

-> Index lookup on Recommendation using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)

-> Index lookup on Review using userId (userId=RankingPriority.userId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)