COL362: Application Project Milestone 2

Vishwas Kalani 2020CS10411
Aman Bansal 2020CS10319
Nischay Diwan 2020CS50433

March 2023

Contents

1 Slight modification in our database from previous milestone

1. We have added 3 more meals types in our meal types details table to include even more variety of food.
2. We have added two more attributes to the meal table including the *food_id* and *restuarant_id* of the *tracked* food items.

2 Queries that will drive application

2.1 Query 1

2.1.1 Description

Filter restaurants based on filters given as drop down menu to users.
By this query we are selecting restaurants from main table based on choices provided by user.
Below is the query which works for all combinations of filters and only iterate over table once and hence reduces time.
For doing this we are maintaining two variables for each filters one is bool which is true if no filter on that attribute else false and second variable contains the specific value of attribute to filter on that attribute.
After having these variables we are using all these variables in selection condition after where and taking or with bool variable to ensure that if filter is not that attribute then do not reject entries because of some default value of attribute variable.

2.1.2 SQL query

```
--q1--
\set A1 true
\set A2 true
\set A3 true
\set A4 true
\set A5 true
\set A6 true
\set A7 true
\set A8 true
\set A9 true
\set B1 '\'Chokhi Dhani\''
\set B2 '\'Jaipur\''
\set B31 100
\set B32 1000
\set B4 '\'NO\''
\set B5 '\'NO\''
\set B6 '\'NO\''
\set B7 '\'NO\''
\set B8 3
\set B9 100
-- filter restaurants
select * from restaurant
    where
        (:A1 or name=:B1)
        and (:A2 or city_id=:B2)
        and (:A3 or (avg_cost_for_two>=:B31 and avg_cost_for_two<:B32))
        and (:A4 or has_table_booking=:B4)
```

```
        and (:A5 or has_online_delivery=:B5)
        and (:A6 or is_delivering_now=:B6)
        and (:A7 or switch_to_order_menu=:B7)
        and (:A8 or aggregate_rating>=:B8)
        and (:A9 or votes>=:B9)
    order by aggregate_rating desc;
```

2.2 Query 2

2.2.1 Description

For showing restaurants in a particular country along with other filters because country name is not in restaurant table.
There is one new variable to be used in this query which contains the country name given by user.

2.2.2 SQL query

```
--q2--
\set CN '\'India\''
-- filter restaurants based on country
with
    var as
        (select * from restaurant
            where
                (:A1 or name=:B1)
                and (:A2 or city_id=:B2)
                and (:A3 or (avg_cost_for_two>=:B31 and avg_cost_for_two<:B32))
                and (:A4 or has_table_booking=:B4)
                and (:A5 or has_online_delivery=:B5)
                and (:A6 or is_delivering_now=:B6)
                and (:A7 or switch_to_order_menu=:B7)
                and (:A8 or aggregate_rating>=:B8)
                and (:A9 or votes>=:B9)
        ),
    nvar as
        (select * from var,city
            where
                var.city_id=city.city_id
                and country_name=:CN
        )
select * from nvar order by aggregate_rating desc;
```

2.3 Query 3

2.3.1 Description

Suggesting restaurants/food_items based on nutrition requirements of the user.
This query is suggesting restaurants based on the nutrition requirements of user.
For doing this query for all type of nutrition combination we are using same technique as query 1 by using two variables for each nutrition attribute in food table.
After filtering food we are finding restaurants having those food items which satisfy the desired nutrition requirement of users.

2.3.2 SQL query

```
\set C0 true
\set C1 true
\set C2 true
\set C3 true
\set C4 true
\set C5 true
\set C6 true
\set C7 true
\set D7 true
\set D0 '\'Salad\''
\set D1 100
\set D2 100
\set D31 0
\set D32 100
\set D41 0
\set D42 100
\set D5 0
\set D6 '\'Wheat Based\''
\set D7 '\'V\''
-- filter food items based on attributes
select * from food
    where
        (:C0 or cuisine_id=:D0)
        and (:C1 or calories>=:D1)
        and (:C2 or fat<=:D2)
        and (:C3 or (carbohydrates>=:D31 and carbohydrates<=:D32))
        and (:C4 or (protein>=:D41 and protein<=:D42))
        and (:C5 or sodium>=:D5)
        and (:C6 or meal_type_id=:D6)
        and (:C7 or veg_non_veg=:D7)
    order by food_id asc;
```

2.4 Query 4

2.4.1 Description

When a user click on some restaurant after filtering then we have display the food items of that restaurant and we are also providing filters for food items after opening a particular restaurant and for that we are using same food table filter as in query 3.

2.4.2 SQL query

```
--q4--
-- display food items of a given restaurant
\set GR 19687969
with
    nvar as
        (select cuisine_id from restaurant_cuisine
            where restaurant_cuisine.restaurant_id=:GR
        ),
    var2 as
```

```
        (select * from
            (select food.* from food,nvar
                where
                    food.cuisine_id=nvar.cuisine_id
            ) as tvar
        where
            (:C0 or cuisine_id=:D0)
            and (:C1 or calories>=:D1)
            and (:C2 or fat<=:D2)
            and (:C3 or (carbohydrates>=:D31 and carbohydrates<=:D32))
            and (:C4 or (protein>=:D41 and protein<=:D42))
            and (:C5 or sodium>=:D5)
            and (:C6 or meal_type_id=:D6)
            and (:C7 or veg_non_veg=:D7)
        )
select * from var2;
```

2.5 Query 5

2.5.1 Description

We are providing a feature on website which is search restaurants which have food items with nutrition values required by user.
For doing this we are using food table filter based on nutrition values and then finding restaurants having these food items in their menu and ordering them based on number of food items with given nutrition values. Also restaurants filters such as city or other are also available along with this.

2.5.2 SQL query

```
--q5--
--display restuarants with foods having nutrition values
with
    var as
        (select * from food
            where
                (:C0 or cuisine_id=:D0)
                and (:C1 or calories>=:D1)
                and (:C2 or fat<=:D2)
                and (:C3 or (carbohydrates>=:D31 and carbohydrates<=:D32))
                and (:C4 or (protein>=:D41 and protein<=:D42))
                and (:C5 or sodium>=:D5)
                and (:C6 or meal_type_id=:D6)
                and (:C7 or veg_non_veg=:D7)
            order by food_id asc
        ),
    nvar as
        (select cuisine_id,count(*) as noi
            from var group by cuisine_id
        ),
    fvar as
        (select restaurant_id,sum(noi) as tnoi
            from restaurant_cuisine,nvar where
```

```
                restaurant_cuisine.cuisine_id=nvar.cuisine_id
            group by restaurant_id
        ),
    tvar as
        (select * from
            (select * from restaurant,fvar
                where
                    restaurant.restaurant_id=fvar.restaurant_id
            ) as tab
            where
                (:A1 or name=:B1)
                and (:A2 or city_id=:B2)
                and (:A3 or (avg_cost_for_two>=:B31 and avg_cost_for_two<:B32))
                and (:A4 or has_table_booking=:B4)
                and (:A5 or has_online_delivery=:B5)
                and (:A6 or is_delivering_now=:B6)
                and (:A7 or switch_to_order_menu=:B7)
                and (:A8 or aggregate_rating>=:B8)
                and (:A9 or votes>=:B9)
            order by aggregate_rating desc
        )
select * from tvar order by tnoi desc,name asc;
```

2.6 Query 6

2.6.1 Description

There is a dedicated section on our website which shows top 5 restaurants which have most number of ordered dishes since last some fixed time (this can be week or month).

2.6.2 SQL query

```
--q6--
\set TL '\'2023-03-17\''

select restaurant_id,count(*) as noo
    from meal where
        meal.entry_type='Track'
        and meal.meal_date >= :TL
    group by restaurant_id
    order by noo desc limit 5;
```

2.7 Query 7

2.7.1 Description

There is a dedicated section on our website which shows top 5 food items which are most ordered dishes since last some fixed time (this can be week or month).

2.7.2 SQL query

```
--q7--
select food_id,count(*) as noo
```

```
    from meal where
        meal.entry_type='Track'
        and meal.meal_date >= :TL
    group by food_id
    order by noo desc limit 5;
```

2.8 Query 8

2.8.1 Description

There is a dedicated section on our website which shows top 5 meal types which are most ordered since last some fixed time (this can be week or month).

2.8.2 SQL query

```
--q8--
select meal_type_id,count(*) as noo
    from meal where
        meal.entry_type='Track'
        and meal.meal_date >= :TL
    group by meal_type_id
    order by noo desc limit 5;
```

2.9 Query 9

2.9.1 Description

There is a dedicated section on our website which shows top 5 cuisines which have most ordered dishes since last some fixed time (this can be week or month).

2.9.2 SQL query

```
--q9--
with
    var as
        (select food_id,count(*) as noo
            from meal where
                meal.entry_type='Track'
                and meal.meal_date >= :TL
            group by food_id
        ),
    nvar as
        (select cuisine_id,count(*) as noc
            from food,var where
                food.food_id=var.food_id
            group by cuisine_id
            order by noc desc limit 5
        )
select * from nvar;
```

2.10 Query 10

2.10.1 Description

The below query counts the number of tracked meals of a user for a specified duration of time (here the person id is 2 and the duration of time is a week) to keep track of how many times he is eating from outside and adding tracked meals to the meal table.

2.10.2 SQL query

```
--q10--
\set X 2
\set date1 '\'2023-04-17\''
\set date2 '\'2023-04-11\''
select count(*)
    from meal where
        person_id = :X
        and entry_type = 'Track'
        and meal_date >= :date2
        and meal_date <= :date1;
```

2.11 Query 11

2.11.1 Description

The below query shows the score of the user based on the score of the meals he is taking. This score will be shown on the user dashboard.

2.11.2 SQL query

```
--q11--
\set X 2
\set date1 '\'2023-04-17\''
\set date2 '\'2023-04-11\''

with
    t as
        (select *
            from meal where
                person_id = :X
                and meal_date >= :date2
                and meal_date <= :date1
        )
select sum(meal_type_score) as total_score
    from t,meal_type_details where
        t.meal_type_id = meal_type_details.meal_type_id;
```

2.12 Query 12

2.12.1 Description

The below query counts the average number of meals added by a user in a week's time. This will also be shown on the user dashboard.

2.12.2 SQL query

```
--q12--
\set X 2
\set date1 '\'2023-04-17\''
\set date2 '\'2023-04-11\''

select 1.0*count(*)/7 as avg_meals
    from meal where
        person_id = :X
        and meal_date >= :date2
        and meal_date <= :date1;
```

2.13 Query 13

2.13.1 Description

This query is for long term statistics of the meals taken by the user. It scans the entire meal table for the user and sees the types across which his/her meals have been distributed. (shows the percentage of various meal types)

2.13.2 SQL query

```
--q13--
\set X 2

with
    t as
        (select meal_type_id as meal_type,0 as number_of_meals
            from meal_type_details
        union all
        select meal_type_id as meal_type,count(*)
            from meal where
                person_id = :X
            group by meal_type_id
        ),
    sum_table as
        (select
            (case
            when sum(number_of_meals)<=0 then 1
            else sum(number_of_meals)
                end
            ) as total_meals
            from t
        ),
    val_table as
        (select meal_type,sum(number_of_meals) as count_meals
            from t
            group by meal_type
        )
select meal_type,100.0*count_meals/total_meals as percent_of_meals
    from val_table,sum_table;
```

2.14 Query 14

2.14.1 Description

Ordering restaurants based on their distance from position of user.

2.14.2 SQL query

```
--q14--
\set Lt 77.257106
\set Lo 28.570142

select restaurant_id,
        111.111 * DEGREES(ACOS(COS(RADIANS(:Lt)) * COS(RADIANS(latitude)) * COS(RADIANS(longitude)
        - RADIANS(:Lo)) + SIN(RADIANS(:Lt)) * SIN(RADIANS(latitude))))
    AS distance_in_km
    from restaurant
    order by distance_in_km asc
    limit 100;
```

2.15 Query 15

2.15.1 Description

Filtering restaurants based on average cost for two.
For doing this we have converted cost of each restaurant from their currency to INR which is standard used by our website for common filtering.

2.15.2 SQL query

```
--q15--
-- filter restaurants
with
    filter_rest as
        (select *
            from restaurant where
                (:A1 or name=:B1)
                and (:A2 or city_id=:B2)
                and (:A4 or has_table_booking=:B4)
                and (:A5 or has_online_delivery=:B5)
                and (:A6 or is_delivering_now=:B6)
                and (:A7 or switch_to_order_menu=:B7)
                and (:A8 or aggregate_rating>=:B8)
                and (:A9 or votes>=:B9)
        )
    -- currency conversion
    select filter_rest.*,
            currency.Inr_conversion*filter_rest.avg_cost_for_two as avg_cost_for_two_INR,
            currency.Inr_conversion
        from filter_rest, currency, city, country_currency where
            country_currency.currency_id = currency.currency_id
            and filter_rest.city_id = city.city_id
            and city.country_name = country_currency.country_name
            and ((currency.Inr_conversion*filter_rest.avg_cost_for_two >= :B31
```

```
            and currency.Inr_conversion*filter_rest.avg_cost_for_two < :B32) or :A3)
    order by avg_cost_for_two_INR asc, aggregate_rating desc, votes desc;
```

3 Index choices for optimizing the queries

We have created the following indexes on our database :

1. **create index meal_date_index on meal(meal_date);** : Since we show some details about the meals on a weekly basis on user dashboard therefore we have made index on date to allow quick filtering of a week.

2. **create index meal_personid_index on meal(person_id);** : Since we have to show the stats of meals to a particular user therefore we can easily filter his meals based on the *person_id* in meal.

3. **create index restaurant_cityid on restaurant(city_id);** : A lot of users might query restaurant on the basis of the city they are in therefore this is a useful index.

4. **create index food_cuisine on food(cuisine_id);** : This is a very useful index since we often need to reach to the food items of the restaurant using the cuisines in the food table therefore it can help in faster joins and selections.

4 Database size and performance

4.1 Database size

1. We have the following large tables in our database. Moreover some tables in our database are very dynamic and may get even larger in a small course of time for example **meal** and **user** tables. The large tables and their sizes are shown below :

Table	size (number of tuples)
Food	1912
Meal	500000
Restaurant	9552
Restaurant_cuisine	19706

2. We have some small tables in our database which contain some basic information for instance the list of cities, rating colours based on the ratings of the restaurant. The small tables and their sizes are shown below.

Table	size (number of tuples)
Average_cost_for_two	5
Meal_type_details	12
Rating	10
User	10
City	141
Country_currency	15
Currency	15

3. The dump file of our database was of around **32 MB**.

4. We have structured our database in such a way that we don't need huge joins for our query.

4.2 Optimizations apart from indexes

Apart from making indexes on appropriate attributes, we have kept in mind the following optimizations while writing our queries :

1. We have pushed our selections inside and then made the joins to make our query efficient. For example we have first selected meals of a user between some dates and then calculated the score of user using that instead of calculating the score of all users and then selecting.

2. We have devised a new filtering-based query for filtering from a table using multiple attributes. We have used boolean operations for this query which allow us to avoid scanning multiple times and does operation simply by selections.

3. We have **materialized view** for join of tables, Restaurants, Restaurant_Cuisine and Food. This join is repeatedly required in some of the queries, so by creating a materialized view, the repeated computation can be avoided optimizing our queries further.

```
create materialized view my_restaurant_cuisine_food as
select
    restaurant.*, food.food_id
from
    restaurant, restaurant_cuisine, food
where
    restaurant.restaurant_id = restaurant_cuisine.restaurant_id and
    restaurant_cuisine.cuisine_id = food.cuisine_id;
```

4.3 Performance of queries after optimizations

To ananlyse the performance of our queries we have measure the timings of the query using **timing** command in postgresql and we have measured the cost using the **explain** statement with our queries. The results shown below are for a specific set of variables we have set which may vary with the inputs we take from the users on our django interface :

Query number	Timing (in ms)	Cost
1	15.76 ms	922.90..946.78
2	19.579 ms	488.26..495.54
3	7.756 ms	0.28..92.94
4	1.055 ms	8.68..62.79
5	57.006 ms	2616.29..2640.15
6	13.766 ms	5477.85..5477.86
7	12.985 ms	5482.60..5482.61
8	13.611 ms	5477.97..5477.98
9	13.383 ms	5530.07..5530.08
10	2.664 ms	2118.89..2118.90
11	7.438 ms	2151.73..2151.74
12	7.100 ms	2118.54..2118.56
13	25.734 ms	6874.21..6881.73
14	7.315 ms	990.83..991.08
15	37.484 ms	1117.81..1141.69

4.4 Statistics of database

We analysed some statistics of our database using the commands shown below :

1. **pg_stat_database** :

```
datid |  datname  | numbackends | xact_commit | xact_rollback | blks_read | blks_hit
| tup_returned | tup_fetched | tup_inserted | tup_updated | tup_deleted | confli
cts | temp_files | temp_bytes | deadlocks
| checksum_failures | checksum_last_failure | blk_read_time | blk_write_time | stats_reset
43940 | group_5   |           3 |         148 |             2 |     27251 |    72138
|  3502898 |      167992 |       531922 |          32 |       0 |    0
|   8 |   30097408 |    0 |
   |        |       0 |     0 | 2023-04-18 16:39:33.953727+05:30
```

2. **pg_stat_database_conflicts** :

```
datid |datname|confl_tablespace|confl_lock| confl_snapshot | confl_bufferpin | confl_deadlock
43940 | group_5   |       0 |          0 |              0 |               0 |              0
```

3. **pg_stat_bgwriter** :

```
checkpoints_timed | checkpoints_req | checkpoint_write_time | checkpoint_sync_time
------------------+-----------------+-----------------------+---------------------
       15604      |        62       |        2037874        |         3111

buffers_checkpoint | buffers_clean | maxwritten_clean | buffers_backend
-------------------+---------------+------------------+-----------------
 80371      |      19160     |           71          |        999221

buffers_backend_fsync | buffers_alloc | stats_reset
----------------------+---------------+-------------
          0           |     136526    | 2023-02-23 11:31:31.335265+05:30
```

4. **pg_stat_activity** : Using this option we could see the queries we are running, the clients that are making this query and some more details of the query.

5. **pg_stat_all_tables** : Here are some of the statistics of the large tables in our database :

```
relid |  schemaname  |    relname   | seq_scan | seq_tup_read
| idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del
| n_tup_hot_upd | n_live_tup | n_dead_tup | n_mod_since_analyze |last_vacuum
| last_autovacuum  | last_analyze |  last_autoanalyze  | vacuum_count | autovacuum_count
| analyze_count | autoanalyze_count
-------------------------------------------------------------------------
43986 | public        | restaurant_cuisine | 3  | 59115
1     | 0             | 19705              | 0  | 0
0     | 0             | 1911               | 0  | 0
0     |               |                    |    |
2023-04-18 16:40:34.878971+05:30 | 0 | 0     | 1
```

```
43962 | public        | food               | 4  | 7644
4     | 3824          | 1911               | 0  | 0
0     | 0             | 1911               | 0  | 0
0     |               |                    |    |
2023-04-18 16:40:34.287579+05:30 | 0 | 0     | 1
43969 | public        | meal               | 11 | 3000000
18    | 156597        | 500000             | 0  | 0
0     | 0             | 500000             | 0  | 0
0     | 5             |                    |    |
2023-04-18 16:40:34.776443+05:30 | 0 | 0     | 1
43979 | public        | restaurant         | 9  | 85959
2     | 2             | 9551               | 0  | 0
0     | 0             | 9551               | 0  | 0
0     |               |                    |    |
2023-04-18 16:40:34.849273+05:30 | 0 | 0     | 1
```