FoodBytes - Restaurant Booking Portal

Rishi Shah(2019CS10394), Ishaan Singh(2019CS10359), Karan Aggarwal(2019CS10699)

1st March, 2022

Contents

1 Section 1 **1**
1.a *Description* 1
1.b *ER Diagram* 2

2 Section 2 - Data **3**
2.a *Sources* 3
2.b *Download Procedure* 3
2.c *Clean-up Procedure* 4
2.d *Statistics* 4

3 Section 3 - Views **4**
3.a *Users View* 4
3.b *System View* 6
3.c *Query Processing Time* 7

1 Section 1

1.a *Description*

In this project, we have designed a portal for users to login and search for their favourite restaurants among 12000+ of them in the city of Bangalore through various kinds of filters like Restaurant Name, Average Cost, Location, Type of Cuisine, and Rating. Apart from that, a user can also book a table in advance as well as write a review for and rate different restaurants. Users can make better decisions by looking at previous reviews and ratings of the restaurants.

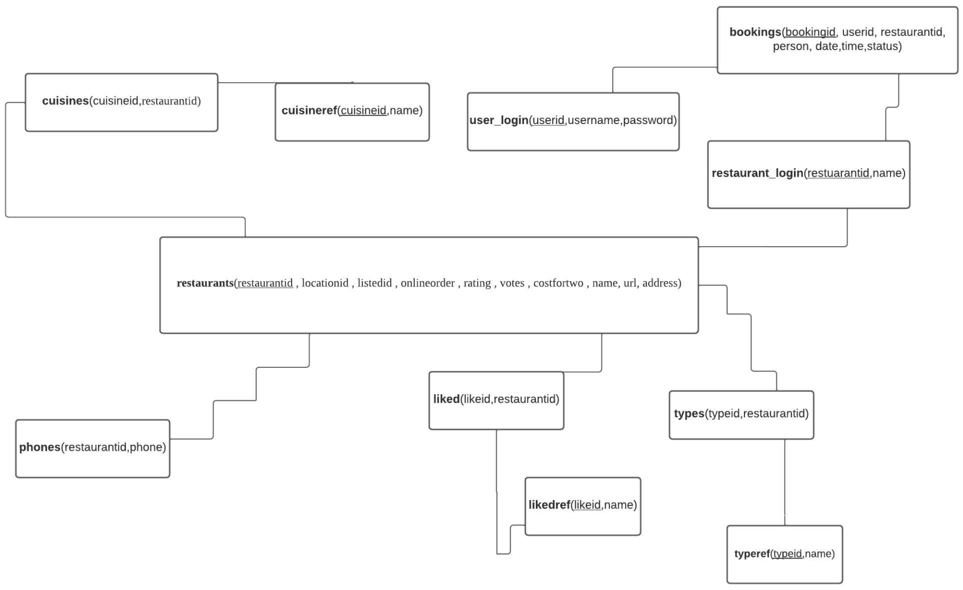
There is a portal for the restaurants too, in which they can see the booking requests by people and can either choose to accept or reject them according to their availability. The restaurants can also keep a track of their past bookings, reviews, and ratings. If a new restaurant wants to join our portal, it can do so by simply entering its basic information like name, address, type of cuisines, etc.

So, overall the complete system allows a person to book a table at his/her favorite restaurant quite easily and conveniently (due to the various filters) as well as rate and review those restaurants.

Our motivation to do this project to ensure a smooth and transparent experience for people to book restaurant tables in advance. Similarly, restaurants can leverage our network to improve their sales and attract new customers. Thus, our platform helps both: the users and the restaurants.

1.b *ER Diagram*

Database	
Entity	Attributes
restaurants	restaurantid (bigint) , locationid (bigint), listedid (bigint), onlineorder (text), rating (float), votes (bigint), costfortwo (bigint), name (text), url (text), address (text)
locationref	locationid (bigint), name (text)
listedref	listedid (bigint), name (text)
liked	restaurantid (bigint), likedid (bigint)
likedref	likedid (bigint), name (text)
phones	restaurantid (bigint), phone (text)
types	restaurantid (bigint), typeid (bigint)
typesref	typeid (bigint), name (text)
cuisines	restaurantid (bigint), cuisineid (bigint)
cuisinesref	cuisineid (bigint), name (text)
reviews	restaurantid (bigint), reviewid (bigint), userid (bigint), rating (float), review (text)
user_login	userid (bigint), username (text), password (text)
restaurant_login	restaurantid (bigint), username (text), password (text)
bookings	bookingid (bigint), userid (bigint), restaurantid (bigint), person (bigint), date (date), time (time), status (text)


bookings(bookingid, userid, restaurantid, person, date,time,status)
cuisines(cuisineid,restaurantid)
cuisineref(cuisineid,name)
user_login(userid,username,password)
restaurant_login(restuarantid,name)
restaurants(restaurantid , locationid , listedid , onlineorder , rating , votes , costfortwo , name, url, address)
liked(likeid,restaurantid)
types(typeid,restaurantid)
phones(restaurantid,phone)
likedref(likeid,name)
typeref(typeid,name)

restaurants: Contains information about all restaurants
Primary Key: restaurantid
Foreign Key: restaurantid references restaurant_login(restaurantid)
Foreign Key: listedid references listedref(listedid)
Foreign Key: locationid references locationref(locationid)
Unique: (name, address)

locationref: Contains information about locationid and name of locations
Primary Key: locationid
Unique: name

listedref: Contains information about category id and name of category of restaurants
Primary Key: listedid
Unique: name

liked: Contains information about dishes of each restaurant liked by people
Foreign Key: likedid references likedref(likedid)
Foreign Key: restaurantid references restaurants(restaurantid)
Unique: (restaurantid, likedid)

likedref: Contains name of dishes of each restaurant liked by people
Primary Key: likedid

phones: Contains phone numbers of all restaurants
Foreign Key: restaurantid references restaurants(restaurantid)
Unique: (restaurantid, phone)

types: Contains information about types of restaurants
Foreign Key: typeid references typesref(typeid)
Foreign Key: restaurantid references restaurants(restaurantid)
Unique: (restaurantid, typeid)

typesref: Contains names of types of restaurants
Primary Key: typeid

cuisines: Contains information about cuisines of restaurants
Foreign Key: cuisineid references cuisinesref(cuisineid)
Foreign Key: restaurantid references restaurants(restaurantid)
Unique: (restaurantid, cuisineid)

cuisinesref: Contains names of cuisines of restaurants
Primary Key: cuisineid

reviews: Contains information about reviews of restaurants
Primary Key: reviewid
Foreign Key: restaurantid references restaurants(restaurantid)
Foreign Key: userid references user_login(userid)
Unique: (restaurantid, userid)

user_login: Contains login information about users
Primary Key: userid
Unique: username

restaurant_login: Contains login information about restaurants
Primary Key: restaurantid
Unique: username

bookings: Contains information about bookings
Primary Key: bookingid
Foreign Key: userid references user_login(userid)
Foreign Key: restaurantid references restaurants(restaurantid)

2 Section 2 - Data

2.a *Sources*

Data was downloaded from kaggle - https://www.kaggle.com/praveenmzzzz876/swiggy-metrocities-dataset.

2.b *Download Procedure*

The database is a ready made csv without any scrapping involved.

2.c *Clean-up Procedure*

The data had many duplicate entries:

- The csv file had 1 big table to begin with.
- Duplicate restaurant entries were identified and remove by running a Python script.
- Duplicate reviews were also removed.
- Some restaurants had the same name and address, resulting in duplicates which were also removed.
- A schema was designed as shown in Part 1.
- The big table was split according to the schema into several tables to get the data into Normal Forms, reduce redundancy, space needed and anomalies.

2.d *Statistics*

Name of tables	**Number of tuples**	**Time to load**	**Size of raw data (After cleanup)**
restaurants	12465	400.802 ms	6.3 MB
locationref	93	10.134 ms	2 KB
listedref	7	3.718 ms	101 B
liked	23490	201.618 ms	236 KB
likedref	2792	10.125 ms	53 KB
phones	16617	63.357 ms	332 KB
types	14118	112.803 ms	111 KB
typesref	25	2.606 ms	312 B
cuisines	28728	188.614	238 KB
cuisinesref	107	40.18 ms	1 KB
reviews	110120	974.435 ms	33.5 MB
user_login (Synthetic)	3	2.597 ms	N.A. (This was generated by users)
restaurant_login (Synthetic)	12465	42.975 ms	1 MB
bookings (Synthetic)	8	2.665 ms	N.A. (This was generated by users)

Size of raw data before cleanup: **92.1 MB**
Size of raw data after cleanup: **41.8 MB**

This is because there were a lot of duplicates in the original data, and several other optimizations were done during cleanup such as minimizing the space used to store strings by making a separate table for them and joining by ids.

3 Section 3 - Views

3.a *Users View*

- Every user has a separate profile and can register his/herself and then login with the credentials. For example, for a user with name 'x', password 'y' performing registration, the fired query is

  ```
  INSERT INTO user_login VALUES ('x','y')
  ```

  Similarly, to authenticate a user who has entered his username as 'x' we obtain its credentials from the login table as:

  ```
  SELECT * FROM user_login WHERE username = 'x'
  ```

- The user can go to the 'Search' tab in his/her profile. The user can use various filters such as **Cost for two**, **Rating**, **Cuisine**, **Location** and **Category** to search for a restaurant.

- Apart from that, he/she can also perform a search by typing the name of the restaurant, in which a prefix match is performed to show the results. The query using the filters as Cost for two:1000-5000, Rating:4-5, Cuisine: Bengali (id x), Location:BTM (id y), Category:Buffet (id z), Prefix:'Ja' is :

```
SELECT restaurants.name, url, restaurants.restaurantid, address,
listedref.name,locationref.name, costfortwo FROM restaurants, cuisines,
listedref, locationref WHERE restaurants.listedid = z and
costfortwo < 5000 and costfortwo >= 1000 and restaurants.name like 'Ja%'
and rating < 5 and rating>=4 and restaurants.locationid= y and
restaurants.restaurantid = cuisines.restaurantid and
cuisines.cuisineid = x and restaurants.listedid = listedref.listedid
and restaurants.locationid=locationref.locationid order by
votes desc limit 100;
```

- The results are displayed as a grid of cards having information about the restaurants and a button to book a table at that restaurant.

- Also, clicking on the restaurant's name redirects the user to another interface where the user can get detailed information about the restaurant, write a review, as well as see his/her previous bookings at the restaurant.
  For example, if user with userid '1' wants to write review with id '45' for a restaurant with id '2', rates it '4', and writes 'good food', the query fired is:

```
INSERT INTO reviews VALUES(2, 45, 1, 4, 'good␣food');
```

  To obtain the previous bookings of user with id '1' , the query fired is:

```
SELECT bookingid, restaurants.name, person, date, time, status,
restaurants.restaurantid FROM bookings, restaurants where userid =1
and restaurants.restaurantid = bookings.restaurantid  order
by date asc, time asc
```

- The page to book a table asks for details like **number of people**, **date** and **time** and also has the option to go back to searching. To book a table for a user with id '1', for '6' people, date '22:03:2022', time: '2:00PM', for restaurant id '2', allotted a booking id '27' query fired is:

```
INSERT INTO bookings VALUES (27,1,2,6,date(22:03:2022),time(2:00PM),
'PENDING')
```

- Similarly, every restaurant has a separate profile and can register itself and then login with the credentials. To register a restaurant with details as username:'u1', password :'p', location 'BTM', Category: 'Desserts', Type: 'Casual Dining', Online order supported: 'Yes', Cost for Two: '3000', url: 'www.u1food.com', address:'Road 3, BTM', Phone number '+91 9832671819', and is allotted an id '57235', the query fired is

```
SELECT listedid FROM listedref WHERE name = 'Desserts';
SELECT locationid FROM locationref WHERE name = 'BTM';
INSERT INTO restaurants VALUES(57235,locationid,listedid,'Yes',None,0,3000,
'u1','www.u1food.com','Road␣3,␣BTM');
INSERT INTO restaurant_login VALUES (1,'u1',hash('p'));
```

- The main page has four tabs: **Information**, **Pending Bookings**, **Reviews** and **Processed Bookings**. Apart from that, there is button to **Edit Profile**, which could be use to change the various profile details.

- The **Information tab** has things like phone number, restaurant type, category, address, average cost for two people, rating, etc. To obtain the information for the restaurant with id 'x', the query fired is:

```
SELECT phone FROM phones WHERE restaurantid = 'x';
SELECT userid, rating, review FROM reviews WHERE restaurantid = x
order by rating desc;
SELECT * FROM restaurants WHERE restaurantid = 'x' limit 1;
```

- The **Pending Bookings** tab shows the various bookings placed by the users, which have not been processed yet. These can be accepted or rejected individually. To obtain the pending bookings for a restaurant with id 'x', the query fired is:

```
SELECT username,person,date,time,bookingid FROM bookings,
user_login where bookings.userid = user_login.userid and
restaurantid =x and status = 'PENDING' order by date asc, time asc
```

- The **Reviews** tab shows the various reviews for the restaurants written by users, along with the rating provided by them. To obtain the reviews for a restaurant with id 'x', the query fired is:

```
SELECT userid, rating, review FROM reviews WHERE
restaurantid = x order by rating desc;
```

- The **Processed Bookings** tab shows the bookings which have been processed by the restaurant (accepted).To obtain the processed bookings for a restaurant with id 'x', the query fired is:

```
SELECT username,person,date,time,bookingid,status FROM bookings,
user_login where bookings.userid = user_login.userid
and restaurantid =x and (status = 'ACCEPTED') order by
status asc, date asc, time asc
```

3.b *System View*

- To perform the search operations efficiently extensive use of indexes has been done. The tables and the attributes on which the indexing has been done are presented below:

Database	
Entity	Attributes
reviews	restaurantid (bigint)
restaurants	locationid (bigint)
restaurants	name (text)
restaurants	listedid (bigint)
bookings	userid (bigint))
bookings	restaurantid (bigint))
bookings	date (date))
cuisines	cuisineid (bigint)
cuisines	restaurantid(bigint)
liked	restaurantid(bigint)
phones	restaurantid(bigint)
types	restaurantid (bigint)

All of the indexing has been done only for those queries which are quite frequently used.

- Indexing reviews on *restaurantid* enables us to find the reviews pertaining to a particular restaurant quickly.
- Indexing restaurants on *locationid* optimizes our search of restaurants based on their location.
- Indexing restaurants on *listedid* optimizes our search of restaurants based on their category.
- Indexing bookings on *userid* enables us to find the bookings done by a particular user quickly.

- Indexing bookings on *restaurantid* enables us to find the bookings done for a particular restaurant quickly.
- Indexing bookings on *date* enables us to check if a user has already booked a restaurant for a particular date efficiently.
- Indexing cuisines on *cuisineid* enables us to find the restaurants having a particular cuisine efficiently.
- Indexing cuisines on *restaurantid* enables us to find all the cuisines served by a particular restaurant quickly.
- Indexing liked by *restaurantid* enables us to find the dishes liked by users for a particular restaurant quickly.
- Indexing phones by *restaurantid* enables us to find the phone numbers for a particular restaurants quickly.
- Indexing types on *restaurantid* enables us to find the type of each restaurant quickly.
- We have also created a view *popular* materialized from *restaurants* which consists of top 100 popular restaurants (decided by maximum number of votes). These are displayed to the user by default when no filtering has been performed.
- The trigger $refresh\_popular$ comes into play when either a new restaurant is added or the *restaurants* table is updated to ensure correctness of the materialzed view *popular*.

3.c *Query Processing Time*

Following are some of the queries which were used -

Database	
Query	Time Taken
SELECT name FROM (SELECT * FROM liked WHERE restaurantid = 1) as temp, likedref where temp.likedid=likedref.likedid;	0.692 ms
SELECT phone FROM phones WHERE restaurantid = 1;	0.235 ms
SELECT name FROM (SELECT * FROM cuisines WHERE restaurantid = 1) as temp, cuisinesref where temp.cuisineid=cuisinesref.cuisineid;	0.618 ms
SELECT * FROM restaurants WHERE restaurantid = 1 limit 1;	0.256 ms
SELECT * FROM user_login WHERE username ='rishi';	0.178 ms
SELECT * FROM restaurant_login WHERE username = 'rishi';	0.258 ms
SELECT username FROM user_login WHERE userid = 1;	0.178 ms
SELECT listedid FROM listedref WHERE name = 'Buffet';	0.183 ms
SELECT restaurants.name, url, restaurants.restaurantid, address, listedref.name, locationref.name, costfortwo FROM restaurants,cuisines, listedref, locationref WHERE restaurants.listedid = 1 and costfortwo<999999 and costfortwo>=0 and restaurants.name like 'Jal%' and rating < 5 and rating>=0 and restaurants.locationid=1 and restaurants.restaurantid = cuisines.restaurantid and cuisines.cuisineid = 1 and restaurants.listedid = listedref.listedid and restaurants.locationid=locationref.locationid order by votes desc limit 100;	1.277 ms
SELECT name FROM cuisinesref ORDER BY name;	0.893 ms
UPDATE bookings SET status = 'ACCEPTED' WHERE bookingid = 1;	0.698 ms
SELECT bookingid,restaurants.name, person, date, time, status, restaurants.restaurantid FROM bookings,restaurants where userid =1 and restaurants.restaurantid = bookings.restaurantid order by date asc, time asc;	0.422 ms
SELECT reviewid, rating FROM reviews WHERE restaurantid = 1 and userid = 1 limit 1;	0.217 ms
INSERT INTO user_login VALUES (1000000,'notme123','pass123');	0.370 ms
SELECT locationid FROM locationref ORDER BY locationid DESC limit 1;	0.178 ms