DATA MANAGEMENT AND DATABASE DESIGN
ASSIGNMENT 2

Topic Name: Restaurant Recommendation and Reservation System

GitHub Repository:
https://github.com/snehalpadekar/Restaurant-Recommendation-and-Reservation-System

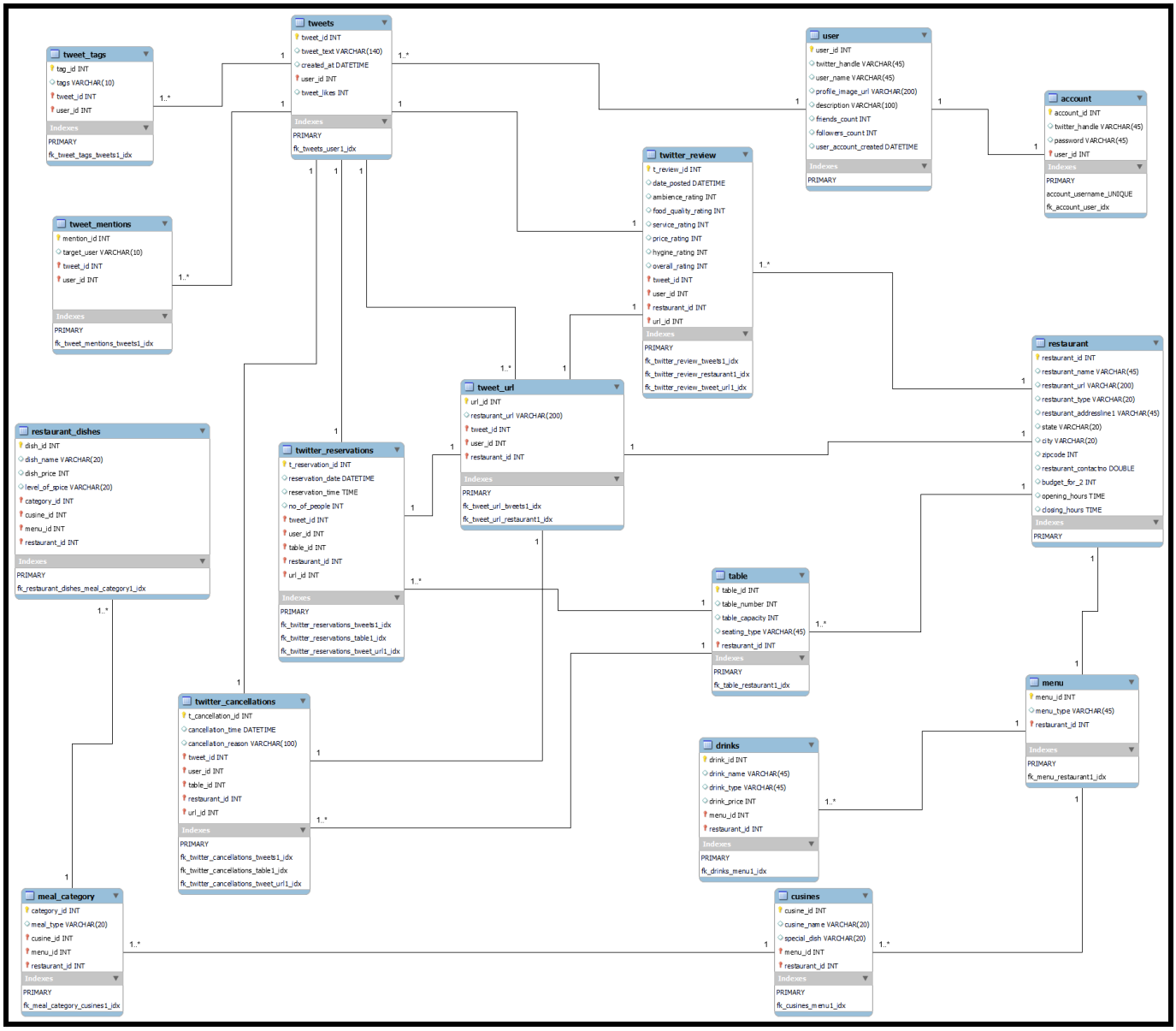
Group Name: R3

Members:
➢Anjali Kshirsagar (002743547)
➢Gayatri Kenkare (002743776)
➢Snehal Padekar (002737903)
➢Mahek Gangadia (002797094)

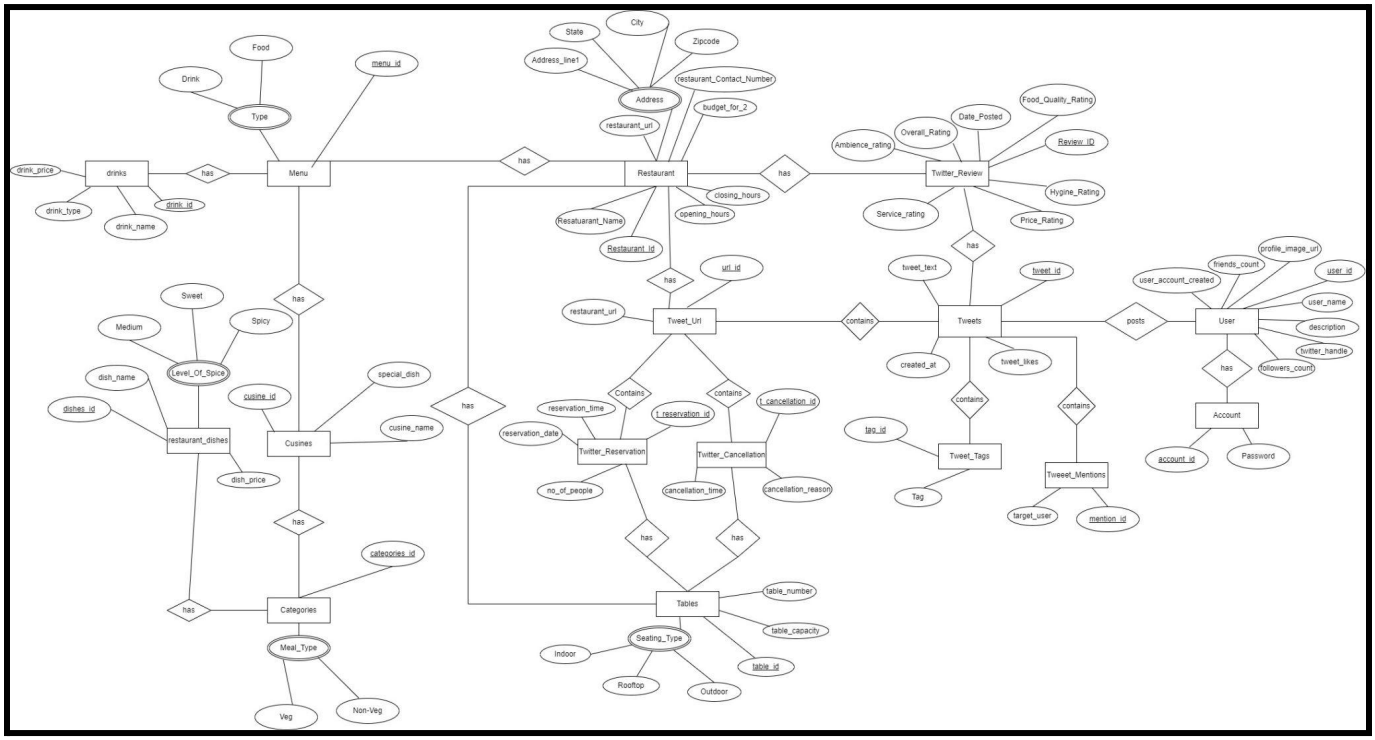
A Model on Restaurant Recommendation and Reservation System

The project idea is to offer a Restaurant Recommendation and Reservation System. It is a platform for people who love to eat. The system will take the user's food preferences into account, which will then suggest good restaurants in the nearby area to the user. This recommendation will solely be based on the cost, the quality of the meal, customer reviews or ratings, accessibility, ambiance, etc. The user can reserve a table after choosing the restaurant. The Reservation System offers features like booking a table at a specific restaurant, canceling the reservation, amending the reservation information, etc. The purpose of this system is to let people get ideas about which restaurant will be great for them. This system can give people some suggestions; also you can get others' opinions from this site. Besides viewing others' opinions, you can give suggestions to other people by rating restaurants. In this system, there are many ways to search restaurants, including by zip code, type, keyword, price, and by recommendation search.
This model also incorporates Twitter database schema. In this model, the user reviews a restaurant, reserves a table, and cancels a particular reservation by tweeting along with the restaurant URL. The restaurant manager can also tweet about their restaurant as a part of promotion and marketing.

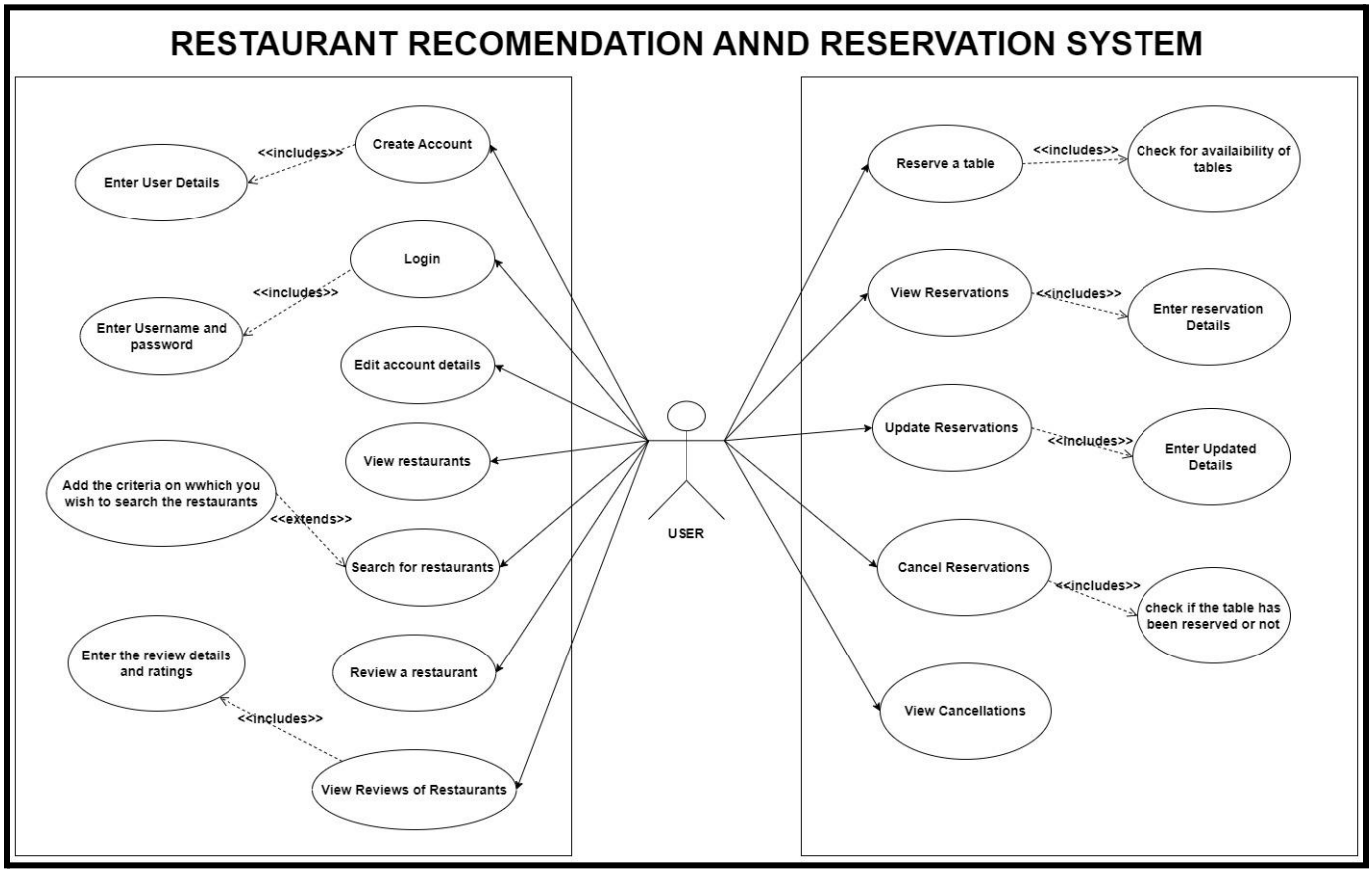
Physical Model diagram of the Restaurant Recommendation and Reservation System.



ER diagram of the online Restaurant Recommendation and Reservation System.



UML diagram (Use Case Diagram)



Explanation of some of the design decisions:

- The Restaurant Recommendation and Reservation System account has a login and password. This login is the same as a user's Twitter handle. The account table contains details related to Twitter accounts. The account table has a primary key named account_id. While every user has a unique id as user_id which is the primary key of the **user** table.
- Each user can tweet any number of **tweets**. Each tweet is uniquely identified by a primary key 'tweet_id'. The restaurant manager who tweets about the promotional offer and ads for marketing purposes is also one of the users and this information can be stored in the user table itself.
- Each tweet can have multiple tags which are stored in the **tweet_tags** table where each tag is uniquely identified by 'tag_id'.
- Users can mention multiple accounts in their tweets. This is stored in the **tweet_mentions** table where each account that is mentioned can be uniquely identified with mention_id which is a primary key for the table.

- A user can review a restaurant through Twitter by tweeting about his experience at the restaurant and mentioning the restaurant URL. This restaurant URL mentioned in a tweet is stored in the '**tweet_url**' table.
- Every tweet that has a URL in it, will have an entry in the **'tweet_url'** table. The 'tweet_url' table also has a primary key as url_id which uniquely defines each URL.
- The **'twitter_reservation'** has a primary key as 't_reservation_id' which uniquely defines each reservation. It also has a foreign key 'tweet_id' of the tweet which uniquely distinguishes each tweet. Each restaurant can have many reservations.
- The '**twitter_cancellation**' has a primary key as 't_cancellation_id' which uniquely defines each cancellation. It also has a foreign key 'tweet_id' of the tweet which uniquely distinguishes each tweet. Each restaurant can have many cancellations.
- The **'twitter_review**' has a primary key as 't_review_id' which uniquely defines each review. It also has a foreign key 'tweet_id' of the tweet which uniquely distinguishes each tweet. Each restaurant can have many reviews.
- The list of all the restaurants is stored in a '**restaurants**' table. In this table, each restaurant is uniquely identified by the restaurant_id key which is a primary key of the table.
- A restaurant has many tables which can be reserved/canceled by the user.
- Each restaurant has various types of tables, whose data is stored in a '**table**' table. It comprises the table number and each table's capacity. 'table_id' is the primary key assigned to this table and it also has a foreign key 'restaurant_id' of the restaurant table which connects the two tables. Users can reserve /cancel the table already reserved by him/her.
- The '**restaurants**' table is also linked with the '**menu**' table whose primary key is menu_id and the foreign key is restaurant_id.
- Menu consists of two main types, that is drinks and cuisine which are respectively saved in '**drinks'** and **' cuisine'** tables. Each table had two foreign keys i.e. 'restaurant_id' and 'table_id'.
- All the data related to drinks and beverages like their name, type, and the price is saved in the '**drinks**' table.
- The type of cuisine served and its id is stored in **'cuisine'** tables. It is further linked with the '**meal_category**' table which specifies the type of meal in each cuisine. Three foreign keys are passed to this table to compare with their respective tables. They are 'cusine_id', 'restaurant_id', and 'menu_id'. 'category_id' is set as the primary key for this table.
- The '**restaurant_dishes'** table with 'dish_id' as its primary key consists of all the names, prices, and levels of spice of each dish in a particular category served in the restaurant. 'category_id', 'cusine_id', 'restaurant_id' and 'menu_id' are passed to this table as foreign keys. The user can select any dish with his/her preference using this system.

SQL STATEMENTS FOR THE CONCEPTUAL MODEL

User Table:

```
CREATE TABLE user (
        user_id INT,
        twitter_handle VARCHAR(45),
        user_name VARCHAR(45),
        profile_img_url VARCHAR(200),
        tweet_description  VARCHAR(100),
        friends_count INT,
        followers_count INT,
        User_account_created DATETIME,
        PRIMARY KEY  (user_id)
);
```

Account Table:

```
CREATE TABLE account (
        account_id INT NOT NULL,
        user_id INT,
        twitter_handle VARCHAR(45),
        password VARCHAR(10),
        PRIMARY KEY  (account_id),
        FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Tweets Table:

```
CREATE TABLE tweets (
        tweet_id INT,
        user_id  INT,
        tweet_text VARCHAR(140),
        created_at DATETIME,
        PRIMARY KEY  (tweet_id),
        FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Tweet_Tags Table:

```
CREATE TABLE tweet_tags (
        tag_id INT,
```

```
        user_id INT,
        tags VARCHAR(10),
        tweet_id INT,
        PRIMARY KEY  (tag_id),
        FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id)
        FOREIGN KEY (user_id) REFERENCES User(user_id)
    );
```

Tweet_Mentions Table:

```
    CREATE TABLE tweet_mentions (
        mention_id INT,
        tweet_id INT,
        user_id INT,
        target_user VARCHAR(10),
        PRIMARY KEY  (mention_id),
        FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id)
        FOREIGN KEY (user_id) REFERENCES User(user_id)
    );
```

Tweet_Url Table:

```
    CREATE TABLE tweet_url (
        url_id INT NOT NULL,
        tweet_id INT NOT NULL,
        user_id INT,
        restaurant_id INT,
        restaurant_url VARCHAR(200)
        PRIMARY KEY  (url_id),
        FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id)
        FOREIGN KEY (user_id) REFERENCES User(user_id)
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
    );
```

Restaurant Table:

```
    CREATE TABLE restaurant (
        restaurant_id INT,
        restaurant_name VARCHAR(45),
        restaurant_url VARCHAR(200),
        restaurant_type VARCHAR(20),
        restaurant_addressline1 VARCHAR(45),
```

```
        state VARCHAR(20),
        zip_code INT,
        city VARCHAR(20),
        restaurant_contactno DOUBLE,
        budget_for_2 INT,
        opening_hours TIME,
        closing_hours TIME,
        PRIMARY KEY  (restaurant_id)
    );
```

Twitter_Review Table:

```
    CREATE TABLE twitter_review (
        t_review_id INT,
        tweet_id INT,
        restaurant_id INT,
        user_id INT,
        url_id,
        date_posted DATETIME,
        ambience_rating INT,
        food_quality_rating INT,
        service_rating INT,
        price_rating INT,
        hygine_rating INT,
        overall_rating INT
        PRIMARY KEY  (t_review_id),
        FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id),
        FOREIGN KEY (url_id) REFERENCES twitter_url(url_id),
        FOREIGN KEY(user_id) REFERENCES user(user_id)
    );
```

Twitter_Reservation Table:

```
    CREATE TABLE twitter_reservations (
        t_reservation_id INT NOT NULL,
        tweet_id INT,
        restaurant_id INT,
        user_id INT,
        url_id,
        table_id INT,
```

```
        reservation_date DATETIME,
        reservation_time TIME,
        no_of_people INT,
        PRIMARY KEY  (t_reservation_id),
        FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id),
        FOREIGN KEY (url_id) REFERENCES twitter_url(url_id),
        FOREIGN KEY (table_id) REFERENCES table(table_id),
        FOREIGN KEY(user_id) REFERENCES user(user_id)
    );
```

Twitter_Cancellation Table:

```
    CREATE TABLE twitter_cancellations (
        t_cancellation_id INT NOT NULL,
        tweet_id INT,
        restaurant_id INT,
        user_id INT,
        url_id,
        table_id INT,
        cancellation_time TIME,
        cancellation _reason Varchar (100),
        PRIMARY KEY  (t_cancellation_id),
        FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id),
        FOREIGN KEY (url_id) REFERENCES twitter_url(url_id),
        FOREIGN KEY (table_id) REFERENCES table(table_id),
        FOREIGN KEY(user_id) REFERENCES user(user_id)
    );
```

Restaurant_Tables Table:

```
    CREATE TABLE table (
        table_id INT NOT NULL,
        restaurant_id INT,
        table_number INT,
        table_capacity INT,
        seating_type varchar(20),
        PRIMARY KEY  (table_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
    );
```

Restaurant_menu:

```
CREATE TABLE menu (
        menu_id INT NOT NULL,
        restaurant_id INT,
        menu_type varchar(20),
        PRIMARY KEY  (menu_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
);
```

Restaurant_cuisine:

```
CREATE TABLE cuisines (
        cuisine_id INT NOT NULL,
        restaurant_id INT,
        cuisine_name varchar(20),
        special_dish varchar(20),
        PRIMARY KEY  (cuisine_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
);
```

Meal_catgeories:

```
CREATE TABLE meal_category (
        categories_id INT NOT NULL,
        restaurant_id INT,
        meal_type varchar(20),
        PRIMARY KEY  (categories_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
);
```

Restaurant_dishes:

```
CREATE TABLE restaurant_dishes (
        dish_id INT NOT NULL,
        restaurant_id INT,
        dish_name varchar(20),
        dish_price INT,
        level_of_spice varchar(20),
        PRIMARY KEY  (dishes_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id)
);
```

Restaurant_drinks:

```
CREATE TABLE drinks (
        drink_id INT NOT NULL,
        restaurant_id INT,
        menu_id INT,
        drink_name varchar(45),
        drink_type varchar(45),
        drink_price INT,
        PRIMARY KEY  (drink_id),
        FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id),
        FOREIGN KEY (menu_id) REFERENCES menu(menu_id)
 );
```

SQL QUERIES AND RELATIONAL ALGEBRA FOR TWITTER DATABASE

1. **What user posted this tweet?**

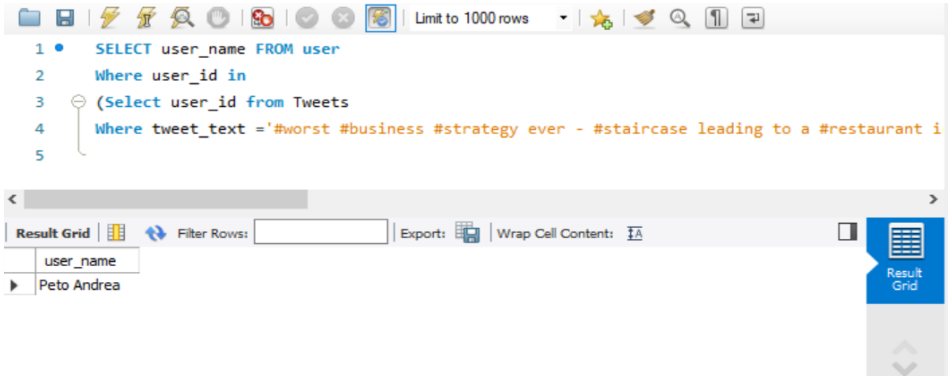
SQL Query:

```
SELECT user_name FROM user
Where user_id in (
        Select user_id from Tweets
        Where tweet_text ='#worst #business #strategy ever - #staircase leading to a #restaurant in #amsterdam #dutch #way #count ⚖🏋 https://t.co/l9mG30Vttj');
```

Relational Algebra:

$\pi_{user\_name}(\sigma_{User.user\_id = Tweets.user\_id \wedge tweet\_text = \text{'\#worst \#business \#strategy ever - \#staircase leading to a \#restaurant in \#amsterdam \#dutch \#way \#count ⚖🏋 https://t.co/l9mG30Vttj'}}(User \times Tweets))$

OUTPUT:



2. **When did the user post this tweet?**

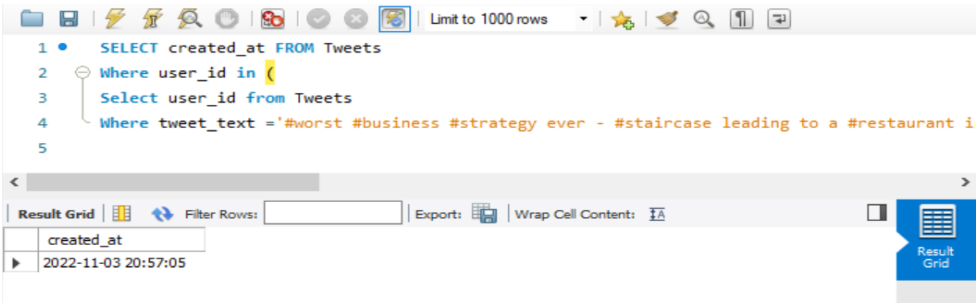
SQL Query:

```
SELECT created_at FROM Tweets
Where user_id in (
        Select user_id from Tweets
Where tweet_text ='#worst #business #strategy ever - #staircase leading to a #restaurant in #amsterdam #dutch #way #count ⚖🏋 https://t.co/l9mG30Vttj');
```

Relational Algebra:

$\pi_{created\_at}(\sigma_{User.user\_id = Tweets.user\_id \wedge user\_name = \text{'\#worst \#business \#strategy ever - \#staircase leading to a \#restaurant in \#amsterdam \#dutch \#way \#count ⚖🏋 https://t.co/l9mG30Vttj'}}(Tweets \times User))$

Output:



3. **What tweets has this user posted in the past 24 hours?**

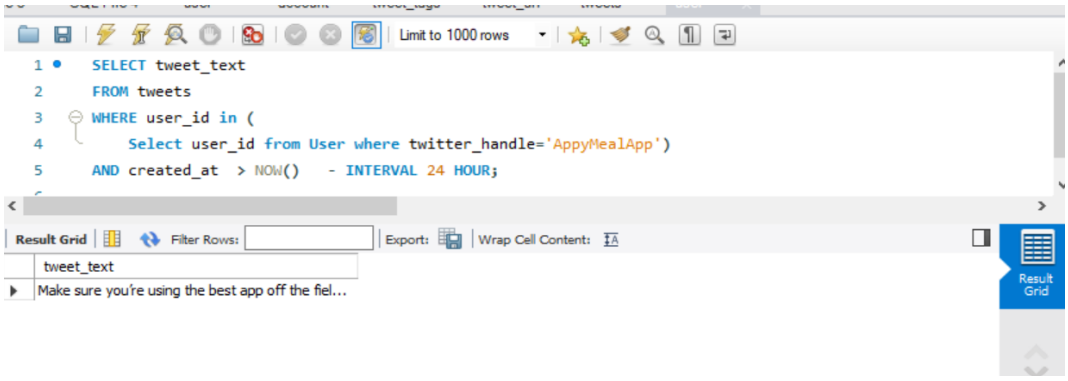
SQL Query:

```
SELECT tweet_text
FROM tweets
WHERE user_id in (
        Select user_id from User where twitter_handle='AppyMealApp')
AND created_at  > NOW()   - INTERVAL 24 HOUR;
```

Relational Algebra:

$\pi_{\text{tweet\_text}}(\sigma_{\text{twitter\_handle='AppyMealApp' ^ created\_at > now()- interval 24 hour}}(\text{Tweets x User}))$

Output:



4. **How many tweets has this user posted in the past 24 hours?**

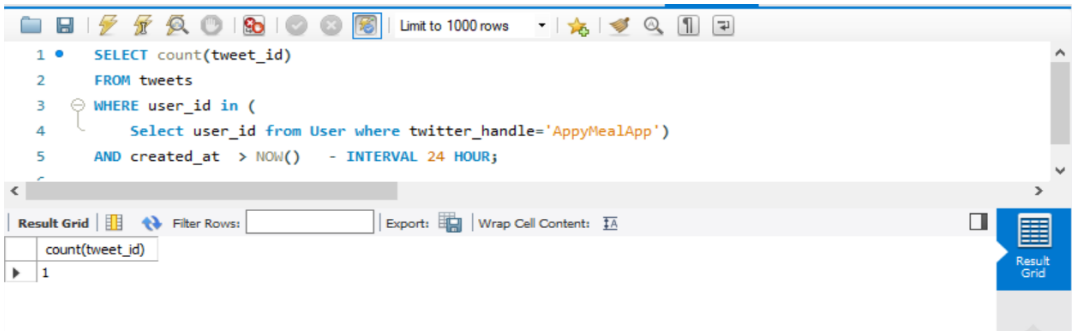
SQL Query:

```
SELECT count(tweet_id)
FROM tweets
WHERE user_id in (
        Select user_id from User where twitter_handle='AppyMealApp')
AND created_at  > NOW()   - INTERVAL 24 HOUR;
```

Relational Algebra:

$\pi_{count(tweet\_id)} (\sigma_{twitter\_handle='AppyMealApp' \wedge created\_at > now()- interval\ 24\ hour}(Tweets \times User))$

Output:



5. **When did this user join Twitter?**

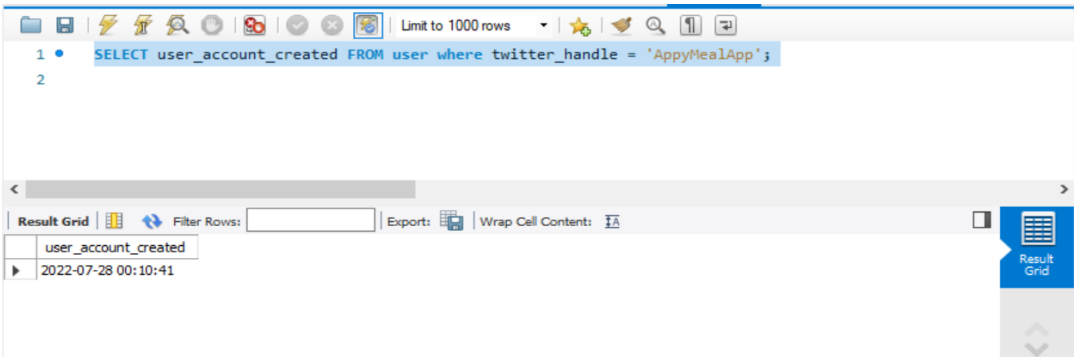
SQL Query:

SELECT user_account_created FROM user where twitter_handle = 'AppyMealApp';

Relational Algebra:

$\pi_{user\_account\_created} (\sigma_{twitter\_handle = 'AppyMealApp'}(User))$

Output:



6. **What keywords/ hashtags are popular?**

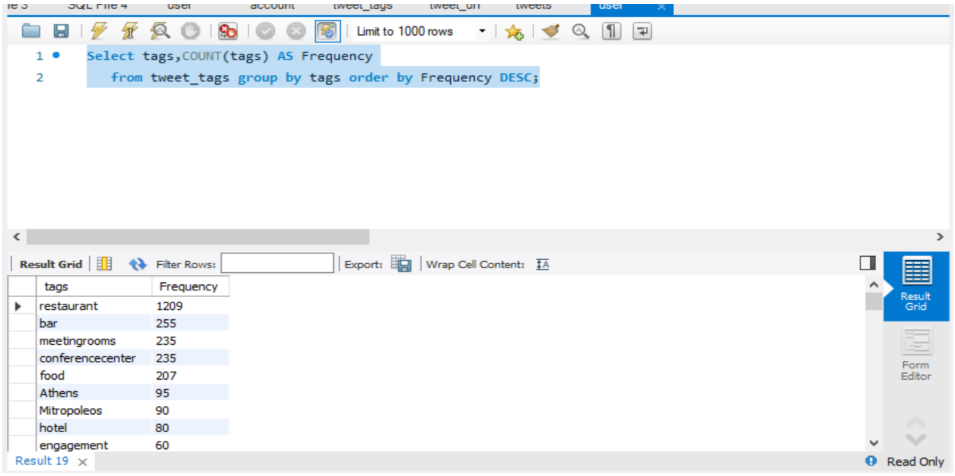
SQL Query:

Select tags,COUNT(tags) AS Frequency
from tweet_tags group by tags order by Frequency DESC;

Relational Algebra:

$\pi_{tweet\_tags}\ G_{count(tweet\_tags)}(Tweets)$

Output:



7. **What tweets are popular?**

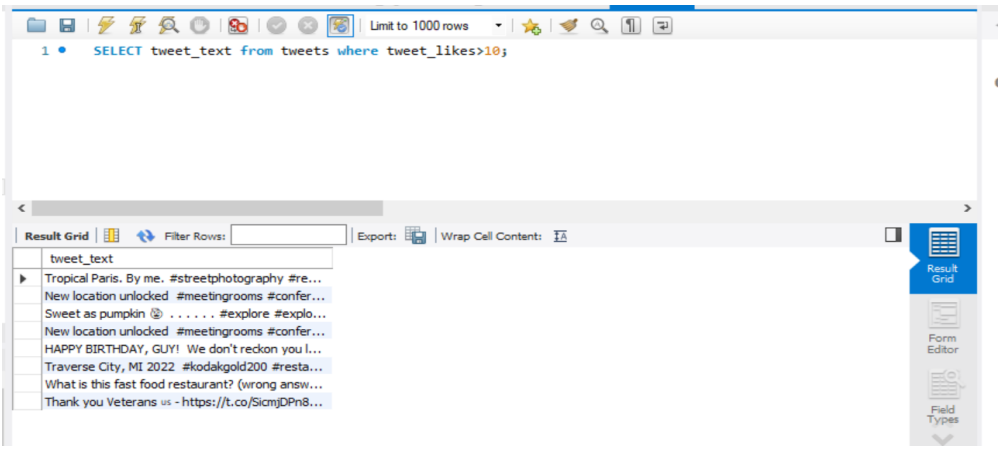
SQL Query:

SELECT tweet_text from tweets where tweet_likes>10

Relational Algebra:

$\pi_{tweet\_likes}(\sigma_{tweet\_likes>10}(Tweets))$

Output:



USE-CASES - SNEHAL PADEKAR

Use case 1: Log in for an account in Restaurant Recommendation and Reservation System

Description: User logins for an account in Restaurant Recommendation and Reservation System

Actor: User

Precondition: When a user wants to search for a restaurant, make a booking or check reviews for any restaurant, firstly he will need to login into the system.

Steps:

Actor action: User request for login

System Responses: If user information is correct then the user is successfully logged in to the system and the use case ends.

Post Condition: The user is successfully logged in.

Alternate Path: The user request is not correct and the system throws an error

Error: User information is incorrect

SQL Query:

INSERT INTO User (user_id, twitter_handle, user_name, profile_image_url, description, friends_count, followers_count, user_account_created) values [(“1”, “ABC”,”abc”,”https:abc.com”,”hi am abc”,”100”,”100”,”2022-09-08”), (“2”, “XYZ”,”xyz”,”https:abc.com”,”hi am abc”,”100”,”100”,”2012-02-02”)]

SELECT * FROM User

Relational-Algebra:

User <- User U { (“1”, “ABC”,”abc”,”https:abc.com”,”hi am abc”,”100”,”100”,”2022-09-08”), (“2”, “XYZ”,”xyz”,”https:abc.com”,”hi am abc”,”100”,”100”,”2012-02-02”) }

$\pi_{user\_id,user\_name,user\_account\_created}$ (User)

Use Case 2: Make a restaurant booking in the Restaurant Recommendation and Reservation System.

Description: The user makes a booking at a restaurant.

Actors: User

Precondition: The user must have a unique Twitter handle to tweet.

Steps:

Actor action: The user tweets about a restaurant booking along with the restaurant URL.

System Responses: A booking is made for the restaurant that matches the restaurant URL.
Post Condition: A booking is added to the Twitter_Reservation table for the restaurant the user tweeted.
Alternate Path: The restaurant currently does not have available tables in the system.
Error: Restaurant Not Available.

SQL Query:
Insert into Twitter_Reservation (t_reservation_id , Tweet_id , restaurant_id, user_id, url_id, Table_id ,reservation_date, reservation_time, no_of_people) values ("1","1234","1","12","1","1",'1999-09-08','23:12:56.98','6')
Select * from Twitter_Reservation

Relational-Algebra:
Twitter_Reservation <- Twitter_Reservation U { ("1","1234","1","12","1","1",'1999-09-08','23:12:56.98','6') }

$\pi_{reservation\_date,reservation\_time,no\_of\_people}$(Twitter_Reservation)

Use Case 3: View a restaurant review already posted through Twitter by a user.
Description: The user views a restaurant review already posted.
Actors: User
Precondition: The user must be logged in.
Steps:
Actor action: The user views a restaurant review from its tweets.
System Responses: Restaurant reviews would be displayed.
Post Condition: system displays the restaurant URL.

SQL Query:-
SELECT food_quality_review, ambiance_rating, service_review, price_review, hygine_review, overall_review FROM tweet_review WHERE restuarant_id in (SELECT restaurant_id FROM restaurant WHERE restaurnt_name =" Mumbai Spice")

Relational-Algebra:-
$\pi_{food\_quality\_review,ambiance\_rating,service\_review,price\_review,hygiene\_review,overall\_review}$ ($\sigma_{tweet\_review.restaurant\_id = restaurant.restaurant\_id \wedge restaurant\_name='Mumbai\ Spice'}$(tweet_review x restaurant))

Use Case 4: View a restaurant with a specific budget (say less than $30)

Description: User views a restaurant within a specific price.
Actors: User
Precondition: The user must be logged in.
Steps:
Actor action – User views a restaurant from its URL.
System Responses – restaurant reviews would be displayed.
Post Condition: system displays restaurant reviews.
Error: No restaurants found within the user's budget.

SQL Query:-

SELECT restaurant_name FROM restaurant WHERE budget_for_2 < 30

Relational-Algebra:-

$\pi_{\text{restaurant\_name}}(\sigma_{\text{budget\_for\_2<30}}(\text{restaurant}))$

Use Case 5: Search for a restaurant that serves pesto pasta.

Description: The user searches for different restaurants that serve pesto pasta
Actor: User
Precondition: The user needs to log in to his account.
Steps:
Actor action: User requests for the Restaurant details which serve pesto pasta
System Responses: Details of all the Restaurants serving pesto pasta will be displayed to the user.
Post Condition: The user will be able to filter out many more features and select a particular restaurant that he/she likes.
Error: The user cannot find the restaurant that serves pesto pasta.

SQL Query:-

SELECT restaurant_id,restaurant_name FROM restaurant WHERE restaurant_id in (SELECT restaurant id FROM restuarant_dishes were dish_name = ‘pesto-pasta’)

Relational-Algebra:-

$\pi_{\text{restaurant\_id,restaurant\_name}}(\sigma_{\text{restaurant.restaurant\_id=restaurant\_dishes.restaurant\_id ^ dish\_name = ‘pesto-pasta’}}(\text{restaurant x restaurant\_dishes}))$

USE-CASES - ANJALI KSHIRSAGAR

Use case 1: Search for a restaurant offering Italian cuisine.

Description: The user searches for different restaurants offering Italian cuisines
Actor: User
Precondition: The user needs to log in to his account.
Steps:
Actor action: User requests for the Details of Restaurants having Italian Cuisine
System Responses: Details of all the Restaurants offering Italian Cuisines will be displayed to the user.
Post Condition: The user will be able to filter out many more features and select a particular restaurant that he/she likes.
Alternate Path: If no such cuisine is present in the database the system will show a message that no such cuisine is provided by the restaurant's
Error: Non-alpha-numeric characters allowed.

SQL Query:-

SELECT * FROM restaurant WHERE restaurant_id in (SELECT restaurant id FROM cuisines WHERE cuisine_type = 'Italian')

Relational-Algebra:-

$\pi\ (\sigma_{\text{restaurant.restaurant\_id = cuisines.restaurant\_id ^ cuisine\_type = 'Italian'}}(\text{restaurant x cuisines}))$

Use case 2: View the cancellations made through Twitter by a user.

Description: The user views the tweets made by him/her to cancel a reservation at a restaurant
Actor: User
Precondition: The user must have made at least one tweet to cancel a reservation at a restaurant.
Steps:
Actor action: User views the history of cancellation tweets.
System Responses: Displays all the cancellation tweets made by the user.
Post Condition: The user will be able to view his/her cancellations details.
Alternate Path: There are no cancellations made by the user.
Error: No history of cancellations available.

SQL Query:

SELECT * FROM twitter_cancellations WHERE user_id in (select user_id from user where user_name='Anjali Kshirsagar'

Relational-Algebra:

π ($\sigma_{\text{twitter\_cancellation.user\_id = user.user\_id ^ user\_name = 'Anjali Kshirsagar'}}$(twitter_cancellation x user))

Use case 3: Update the no of people in the reservation made by the user through Twitter.

Description: The user updates a reservation detail at a restaurant via Twitter.
Actor: User
Precondition: The user must be logged in to his/her Twitter account and the User must have at least one reservation at that restaurant.
Steps:
Actor action: The user updates the reservation details through Twitter.
System Responses: Reservation details updated.
Post Condition: The user can view the updated reservation details.
Alternate Path: There are no reservations made by the user.
Error: The user has not reserved a table for this restaurant

SQL Query:

UPDATE twitter_reservations SET no_of_people = 8
WHERE user_id='xxx' AND reservation_date = 'xxx' AND reservation_time= 'xxx'

Relational-Algebra:

twitter_reservations <- t_reservation_id, reservation_date, reservation_time,no_of_people = 8(user_id='xxx' AND reservation_date = 'xxx' AND reservation_time= 'xxx'(twitter_reservations))

Use Case 4: View the restaurant which serves alcoholic drinks.

Actor: User
Precondition: The user must be logged into his account.
Steps:
Actor action: The user requests a list of restaurants that serve alcoholic drinks.
System Responses: Details of the restaurants meeting the criteria are displayed.

Post Condition: The user will be able to filter out many more features and select a particular restaurant that he/she likes and further reserve a table if he/she wants.
Alternate Path: The user has not logged into his account.
Error: User not logged in.

SQL Query:

SELECT * FROM restaurant WHERE restaurant_id in (SELECT restaurant_id FROM drinks WHERE drink_type = 'Alcohol')

Relational-Algebra:

π ($\sigma_{\text{restaurant.restaurant\_id = drinks.restaurant\_id ^ drink\_type = 'Alcohol'}}$(restaurant x drinks))

Use Case 5: View a specialty of every cuisine from a specific restaurant.

Actor: User
Precondition: The user must be logged into his account
Steps:
Actor action: The user requests the details of a specific restaurant with a special dish of cuisine provided by that restaurant.
System Responses: Displays the list of cuisines provided by the restaurant with its specialty.
Post Condition: The user can decide which dish he wants to order when he checks in at the restaurant.
Alternate Path: The user enters the wrong restaurant name.
Error: No such restaurant is available.

SQL Query:

```
Select cuisine_name, a special dish from cuisines where restaurant_id in (
    Select restauarant_id from the restaurant
    where restauarnt_name='Mumbai Spice')
```

Relational-Algebra:

$\pi_{\text{cuisine\_name,special\_dish}}$($\sigma_{\text{restaurant.restaurant\_id = cuisines.restaurant\_id ^ restaurant\_name = 'Mumbai Spice'}}$(restaurant x cuisines))

USE-CASES - GAYATRI KENKARE

Use Case 1: Search for restaurants that offer entirely vegetarian food

Description: The user searches for restaurants that offer vegetarian food.
Actors: User
Precondition: The user must be logged in from his account.
Steps:
Actor action: User searches for details of restaurants that offer entirely vegetarian dishes.
System Responses: Displays details of the restaurants offering vegetarian food.
Post Condition: Users will be able to select restaurants by viewing other features and previous reviews of those restaurants.
Alternate Path: If no such restaurant is available which offers vegetarian food, the system will show an error.
Error: No restaurants found that offer vegetarian food.

SQL Query:

SELECT restaurant_name FROM restaurant WHERE restaurant_id in(SELECT restaurant_id FROM meal_category WHERE meal_type = “veg”)

Relational-Algebra:

$\pi_{\text{restaurant\_name}}(\sigma_{\text{restaurant.restaurant\_id = meal\_category.restaurant\_id ^ meal\_type = veg}}(\text{restaurant x meal\_category}))$

Use Case 2: To submit a review for a certain restaurant on Twitter.

Description: The user tweets a review for a certain restaurant from his/her own Twitter handle.
Actors: User
Precondition: The user must have a Twitter handle to tweet.
Steps:
Actor action: The user submits a review for a restaurant on Twitter.
System Responses: The review will be submitted and can be viewed by others.
Post Condition: The system displays the review for that restaurant for other twitter handlers too.
Alternate Path: User not logged in
Error: No review is submitted

SQL Query:

Insert into twitter_review (t_review_id, tweet_id, restaurant_id, user_id, url_id, Date_posted ,food_quality_review, ambiance_rating, service_review, price_review, hygine_review, overall_review) WHERE restuarant_id in(SELECT restaurant_id FROM restaurant WHERE restaurnt_name =" Mumbai Spice")

Relational-Algebra:

twitter_review <- twitter_review U { ("1","21234627489","1234","23456","2022-09-08","1","4","3","5","2","4"), ("2","9786543678","1224","2213956","2022-05-08","3", "4","5","3","4","4") }

Use Case 3: View the top 5 restaurants with the highest overall rating in Jamaica Plain.

Description: The user views the restaurant's highly recommended in Boston already posted.
Actors: User
Precondition: The user must be logged in.
Steps:
Actor action: The user views the details of the top 5 restaurants shown by the system.
System Responses: Displays details of those restaurants.
Post Condition: The system will display those restaurants.
Alternate Path: The user is not logged in.
Error: User not logged into his account.

SQL Query:

SELECT restaurant_name FROM restaurant WHERE zip_code=02130 AND restaurant_id in(SELECT restaurant_id FROM twitter_review WHERE ambiance_rating>4)

Relational-Algebra:

$\pi_{\text{restaurant\_name}}(\sigma_{\text{restaurant.restaurant\_id = twitter\_review.restaurant\_id ^ zip\_code = '02130'}}(\text{restaurant x twitter\_review}))$

Use Case 4: View a restaurant that offers indoor seating.

Description: The user views a restaurant that offers indoor seating.
Actors: User

Precondition: The user must be logged in from his/her account.
Steps:
Actor action: The user views a restaurant that offers indoor seating.
System Responses: Displays details of those restaurants.
Post Condition: System will display those restaurants.
Error: User not logged into his account or No restaurants found that offer indoor seating.

SQL Query:

SELECT restaurant_name FROM restaurant WHERE restaurant_id in(SELECT restaurant_id FROM table WHERE seating_type =" indoor seating")

Relational-Algebra:

$\pi_{\text{restaurant\_name}}(\sigma_{\text{restaurant.restaurant\_id = table.restaurant\_id ^ seating\_type = indoor seating}}(\text{restaurant x table}))$

Use Case 5: Search for restaurants that offer mildly spiced dishes.
Description: The user searches for a restaurant that offers mildly spiced dishes.
Actors: User
Precondition: The user must be logged in from his/her account.
Steps:
Actor action – The user searches for restaurants offering mildly spiced dishes.
System Responses – Displays all the restaurants offering mildly spiced dishes.
Post Condition: The user Can view all the restaurants and reserve a table at a restaurant of his choice.
Alternate Path: The user request is not correct and the system throws an error.
Error: No restaurants found that offer this combination of features.

SQL Query:

SELECT restaurant_name FROM restaurant WHERE restaurant_id in(SELECT restaurant_id FROM restaurant_dishes WHERE level_of_spice =" mildly spiced")

Relational-Algebra:

$\pi_{\text{restaurant\_name}}(\sigma_{\text{restaurant.restaurant\_id = restaurant\_dishes.restaurant\_id ^ level\_of\_spice = 'mildly spiced'}}(\text{restaurant x restaurant\_dishes}))$

Use Case 6: Book a restaurant which has outdoor seating and offers Mexican Cuisine from Twitter

Description: The user books a restaurant which has outdoor seating and offers Mexican cuisine from Twitter.
Actors: User
Precondition: The user should have a Twitter handle.
Steps:
Actor action: The user looks for a restaurant with outdoor seating and Mexican cuisine.
System Responses: Displays the list of all restaurants that fulfill the requirements.
Post Condition: Users will be able to select restaurants by viewing other features of those restaurants.
Alternate Path: The restaurant currently does not have available tables.
Error: No restaurants found that offer this combination of features.

SQL Query:-

Insert into Twitter_Reservation (t_reservation_id , Tweet_id , restaurant_id, user_id, url_id, Table_id ,reservation_date, reservation_time, no_of_people) values (“1”,”1234”,”1”,”12”,”1”,”1”,‘1999-09-08’,’23:12:56.98’,’6’) WHERE restuarant_id in(SELECT restaurant_id FROM table WHERE seating_type =” indoor seating”)

Relational-Algebra:-

Twitter_Reservation <- Twitter_Reservation U { (“1”,”1234”,”1”,”12”,”1”,”1”,‘1999-09-08’,’23:12:56.98’,’6’) }

USE-CASES - MAHEK GANGADIA

Use Case 1: Check if the table is available for Reservation for a specified restaurant.

Description:- The user checks if a table is available for reservation in a restaurant.
Actors: User
Precondition: The user must be logged in to his/her account
Steps:
Actor Action: The user searches for the restaurant and tries to make reservations
System Response: Displays all the tables available for making a reservation
Post Condition: User will decide if he/her wants to book that particular table or not.

SQL Query:

SELECT table_no, table_capacity, table_type From table WHERE restaurant_id in(SELECT restaurant_id FROM restaurant WHERE restaurant_name = ” Mumbai Spice”)

Relational-Algebra:

$\pi_{table\_no,table\_capacity,table\_type}(\sigma_{table.restaurant\_id = restaurant.restaurant\_id \wedge restaurant\_name = \text{‘Mumbai Spice’}}(table \times restaurant))$

Use Case 2: Search for the spice level of a dish

Description:
Actors: User
Precondition: The user must be logged in to his/her account.
Steps:
Actor Action: The user looks for cuisine at the restaurant he/she wants to dine in.
System Response: DIsplays the cuisine menu along with the level of spice of each cuisine.
Post Condition: The user will decide if the level of spice in that cuisine works for him/her.

SQL Query:

SELECT level_of_spice FROM restaurant_dishes WHERE dish_name = “Chicken Biryani”

Relational-Algebra:

$\pi_{level\_of\_spice}(\sigma_{dish\_name = \text{'Chicken Biryani'}}(restaurant\_dishes))$

Use Case 3: Search for a Non-Veg Restaurant.

Description: The user searches for restaurants that offer non-vegetarian food.
Actors: User
Precondition: The user must be logged in.
Steps:
Actor Action: The user looks for restaurants that offer non_veg cuisine.
System Response: Display all the details of the restaurants that offer non-vegetarian cuisines.
Postcondition: The user will decide which restaurant to visit.

SQL Query:

SELECT restaurant_name FROM restaurant WHERE restaurant_id in(SELECT restaurant_id FROM meal_category WHERE meal_type = “non-veg”)

Relational-Algebra:

$\pi_{restaurant\_name}(\sigma_{restaurant.restaurant\_id = meal\_category.restaurant\_id \wedge meal\_type = \text{'nonveg}}(restaurant \times meal\_category))$

Use Case 4: Check restaurant with ambiance rating.

Description: The user checks which restaurant has the best ambiance rating.
Actors: User
Precondition: The user must have a Twitter account.
Steps:
Actor Action: The user searches for ambiance ratings for restaurants in tweet reviews.
System Response: Show all the restaurants according to ambiance rating.
Post Condition: System will display the list of restaurant list according to ambiance rating.

SQL Query:

SELECT restaurant_name FROM restaurant WHERE restaurant_id in(SELECT restaurant_id FROM twitter_review WHERE ambiance_rating>4)

Relational-Algebra:

$\pi_{restaurant\_name}(\sigma_{restaurant.restaurant\_id = twitter\_review.restaurant\_id \wedge ambiance\_rating > 4}(restaurant \times twitter\_review))$

Use Case 5: Find restaurants URL which is included in user tweets.

Description: The user searches for the restaurant's URL.
Actors: User
Precondition: The User must have a Twitter account.
Steps:
Actor Action: The user looks for the restaurant's URL.
System Response: The system shows all tweets containing Restaurant's URL.
Post Condition: The user can view the restaurants using that URLs.

SQL Query:-

SELECT unique(restaurant_url) FROM tweet_url;

Relational-Algebra:-

$\pi_{restaurant\_url}(tweet\_url))$