601.615 Database

Final Project Phase II

12.21.2020

Nanxi Ye, Linghao Jin

nye3@jhu.edu and ljin23@jhu.edu

Description: This project provides an interface for users looking for Airbnb listings with concerns of Covid cases and parks in the neighborhood. We want to demonstrate how different datasets can provide a comprehensive view on user searching. The project focuses on a user-friendly interface and some data analysis. The project is deployed on https://ugrad.cs.jhu.edu/~nye3/

1. Some change from phase I

a) New datasets: We incorporate two sets of new data. One is the **national park data** in San Francisco. Considering that travelers might have interests in visiting the nearby parks during vacation. It will be useful to employ park data on our snow bnb website. The other is **COVID-19 dataset** in San Francisco. Travelers now normally concern most about the covid-19 situation in their destinations. Providing the cumulative covid-19 cases would help them to make better decisions.
b) Besides searching listings and displaying corresponding hosts, covid, park, and reviews information relating to the listing. We add a **database management system (DBMS)** for our SNOW BNB workers to work on. They can not only insert, delete new records of listing and hosts, but also ask for some analysis of current existing listings.

2. Data source

We have three sources for our dataset.

- The dataset on Airbnb listings, hosts and reviews are obtained from their official website Inside Airbnb: http://insideairbnb.com/get-the-data.html;
- The dataset on parks in San Francisco is obtained from Kaggle: https://www.kaggle.com/danofer/sf-parks;
- The data on Covid-19 cases and neighborhoods are obtained from San Francisco Open Data: https://data.sfgov.org/stories/s/Map-of-Cumulative-Cases/adm5-wq8i#cumulative-cases-map https://data.sfgov.org/stories/s/Map-of-Cumulative-Cases/adm5-wq8i#new-cases-map.

3. How to load our database

We obtained our dataset mostly in the format of csv files from online sources and used sql code “load data local infile” to insert data from csv files into tables. Some minor tweaks are added to make sure the data fits in SQL tables. An example code is given as follows:

```
LOAD DATA LOCAL INFILE 'data/airbnb_covid_neighborhood.csv'
INTO TABLE Zipcode
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(airbnb_neighbourhood, covid_neighborhood, zipcode);
```

4. How to run our code

Once you have unzipped the submitted code file, you will see 4 directories: data, doc, sql and website. Data and doc directories contain raw data and documentations of this project. Under sql directory, connect to the database. If you are on a local host, try:

```
/usr/local/mysql/bin/mysql --local-infile=1 -u root -p
```

And then prompt the password. Or you can connect to mariaDB if you are on ugrad. After successfully connect to sql server, run:

```
\. load_data.sql
```

This may take a minute or two to import all the data into tables. After it's done, run:

```
\. procedures.sql
```

This command creates all the procedures needed for the php files in the project.
After the previous steps, go to the website directory and modify the file php/dbase-conf.php to the corresponding database connection info of your local host. Then in terminal, run:

```
php -S localhost:8000
```

This opens a localhost for php files. Now you can put localhost:8000 in your browser can see where it takes you.

To login to DBMS page, use "admin" for account # and "adminpwd" for password.

5. Project specializations

1. Concise and modern HTML interface: We designed an interface using HTML/CSS/PHP/JS/Mysql. The interface would display the users and workers the most simple operation to achieve what they want to do.
2. Data Analysis: In the DBMS, we provide queries to display some data analysis according to the current database. Specifically, we have different panels for different table analysis:
    a. *Listing data analysis* shows the aggregated results of listings in different neighborhoods.
    b. *Covid data analysis* gives a view on Covid cases in every neighborhood in San Francisco, and some corresponding statistics.
    c. *Park data analysis* shows the aggregated ratings and average size of parks in each neighborhood. Click the links below to modify listing data.
    d. *Location data analysis* shows the relations between covid cases, airbnb listing ratings and park ratings based on their locations. Click the links below to modify listing data.
    e. *Health recommendation* lists a recommendation to the neighborhoods based on their covid new to cumulative cases ratio, location rating and park scores in that order.
    f. *Joined data analysis* gives a joined view of Airbnb listings, park scores, Covid cases in all neighborhoods in San Francisco.

6.Project selling points

1. The interface and operations provide users the most intuition and simple experiences. We show users the most important information they might want to do during booking an house. Such as pictures of the houses, basic information of the house. All the information displayed in an elegant style gives the most straightforward experience.
2. We also implemented Google Map to display current pages' listings. And designed a pagination to make each page only show 10 listings to make the interface cleaner and faster.
3. COVID-19 information can be easily captured using our interface. Instead of searching google again "how many cases are in San Francisco until now ?", we incorporate covid-19 dataset in our database and display it on the interface.
4. All the listing search restrictions are implemented by button to provide user the most simple input format. Also, insert and delete new records in DBMS is easy to operate because we designed a good input interface.

7. Project Limitations

1. Our data might not be most up to date. Because the data we retrieved from Airbnb is not most recent. This might be solved by using API to fetch newest airbnb and covid data in the future.
2. Our data analysis functions are not very comprehensive. Increasingly complicated queries might be able to be implemented in the future to satisfy workers' demand to analyze the listings situation more thoroughly.
3. The dataset obtained from Airbnb contains certain restrictions. For example, we found many repeated values in id fields, which is supposed to be unique. We also found the reviewer data and review data have discrepancies. You can notice that from warning messages when data is being loaded.

8. Use of outside components

We employed ***Mobirise*** to design our interface. Mobirise helps us achieve a good-looking interface with better css/html design. Basically, we can select the template components provided in Mobirise and then change it with our own design. It provides a structure of the html and we wrote our own PHP and some html files. See Mobirise: https://mobirise.com/

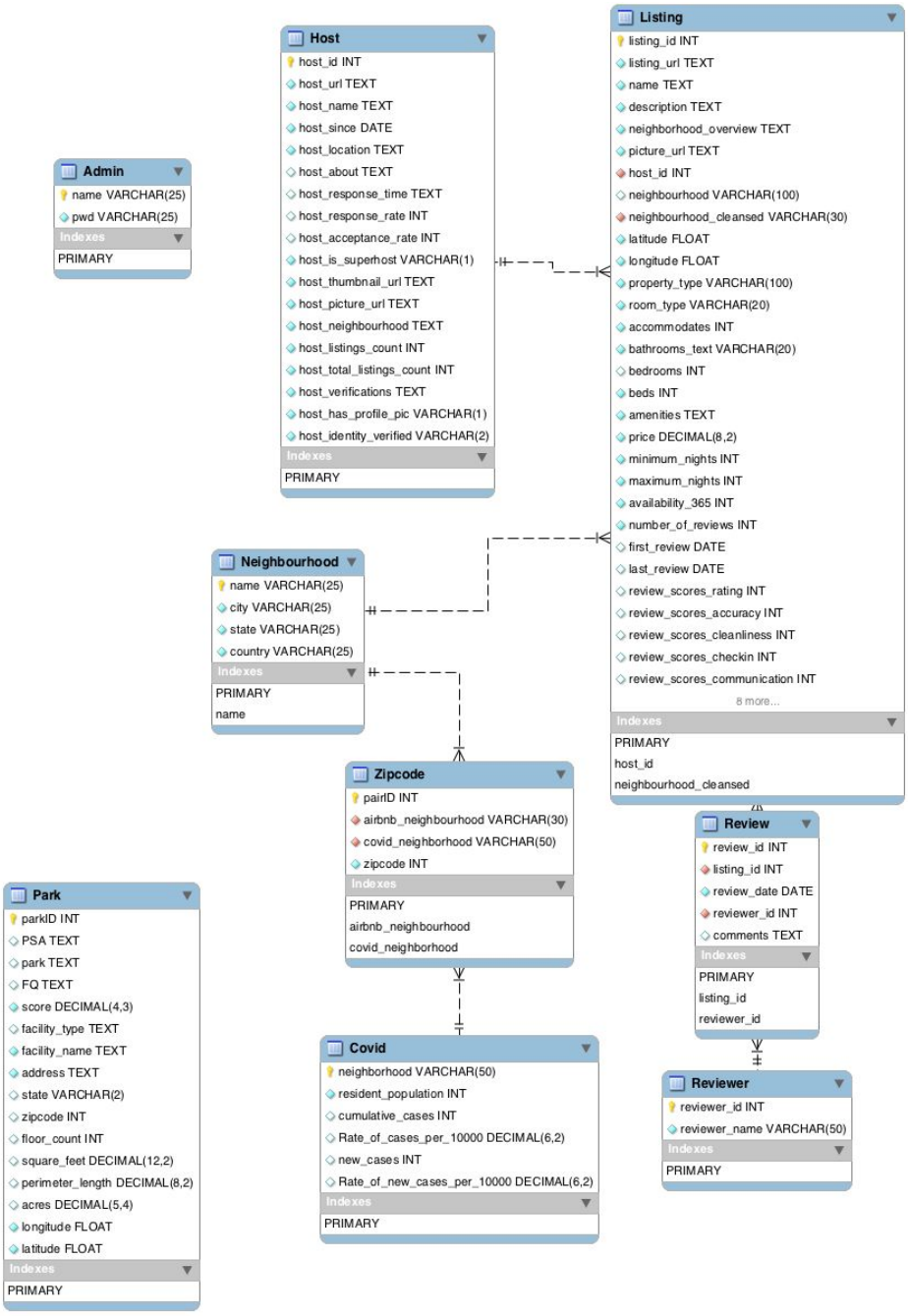
We employed Google Maps as a component in the html page to show the listing locations in the map by fetching its API.

9. Formatted output

We recorded a brief demo to show you the output from our project. See link below.
https://youtu.be/eNUmMqli0E0

10. Full Relational Table Specification

Our table specification is not much different from phase I submission. The ER diagram is listed as follows:



11. Copy of Phase I

You can refer to our phase I written-up using this link:
https://github.com/maxye-frz/615_project/blob/main/doc/Phase%20I.md

12. Copy of all SQL code

Please refer to specific .sql files under /sql directory for a better view. You can find all code we use for this project in gradescope submission.

load_data.sql:

```
-- load data before use:

DROP DATABASE IF EXISTS final_project;

CREATE DATABASE final_project;

USE final_project;

-- set MySQL server not in strict mode, so that table can be inserted with NULL for int type fields
SET @@global.sql_mode= '';

-- drop and create tables:
DROP TABLE IF EXISTS Admin;
CREATE TABLE Admin(
   name VARCHAR(25) NOT NULL,
   pwd VARCHAR(25) NOT NULL,
   PRIMARY KEY (name)
);


DROP TABLE IF EXISTS Neighbourhood;
CREATE TABLE Neighbourhood(
   name              VARCHAR(25) NOT NULL,
   city              VARCHAR(25) NOT NULL,
   state             VARCHAR(25) NOT NULL,
   country           VARCHAR(25) NOT NULL,
   PRIMARY KEY (name),
   UNIQUE (name, city, state, country)
);


DROP TABLE IF EXISTS Host;
CREATE TABLE Host(
   host_id INT NOT NULL,
   host_url TEXT NOT NULL,
   host_name TEXT NOT NULL,
   host_since DATE NOT NULL,
   host_location TEXT NOT NULL,
```

```
    host_about TEXT,
    host_response_time TEXT,
    host_response_rate INT,
    host_acceptance_rate INT,
    host_is_superhost VARCHAR(1) NOT NULL,
    host_thumbnail_url TEXT NOT NULL,
    host_picture_url TEXT NOT NULL,
    host_neighbourhood TEXT NOT NULL,
    host_listings_count INT NOT NULL,
    host_total_listings_count INT NOT NULL,
    host_verifications TEXT NOT NULL,
    host_has_profile_pic VARCHAR(1) NOT NULL,
    host_identity_verified VARCHAR(2) NOT NULL,
    PRIMARY KEY (host_id)
);


DROP TABLE IF EXISTS Listing;
CREATE TABLE Listing(
    listing_id INT NOT NULL,
    listing_url TEXT NOT NULL,
    name TEXT NOT NULL,
    description TEXT NOT NULL,
    neighborhood_overview TEXT NOT NULL,
    picture_url TEXT NOT NULL,
    host_id INT NOT NULL,
    neighbourhood VARCHAR(100),
    neighbourhood_cleansed VARCHAR(30) NOT NULL, -- Neighborhood.id?
    latitude FLOAT NOT NULL,
    longitude FLOAT NOT NULL,
    property_type VARCHAR(100) NOT NULL,
    room_type VARCHAR(20) NOT NULL,
    accommodates INT NOT NULL,
    bathrooms_text VARCHAR(20) NOT NULL,
    bedrooms INT,
    beds INT NOT NULL,
    amenities TEXT NOT NULL,
    price DECIMAL(8, 2) NOT NULL,
    minimum_nights INT NOT NULL,
    maximum_nights INT NOT NULL,
    availability_365 INT NOT NULL,
    number_of_reviews INT NOT NULL,
    first_review DATE,
    last_review DATE,
```

```
    review_scores_rating INT,
    review_scores_accuracy INT,
    review_scores_cleanliness INT,
    review_scores_checkin INT,
    review_scores_communication INT,
    review_scores_location INT,
    review_scores_value INT,
    instant_bookable VARCHAR(1) NOT NULL,
    calculated_host_listings_count INT NOT NULL,
    calculated_host_listings_count_entire_homes INT NOT NULL,
    calculated_host_listings_count_private_rooms INT NOT NULL,
    calculated_host_listings_count_shared_rooms INT NOT NULL,
    reviews_per_month DECIMAL(3, 2) NOT NULL,
    PRIMARY KEY (listing_id),
    FOREIGN KEY (host_id) REFERENCES Host(host_id),
    FOREIGN KEY (neighbourhood_cleansed) REFERENCES Neighbourhood(name)
);


DROP TABLE IF EXISTS Reviewer;
CREATE TABLE Reviewer(
    reviewer_id INT NOT NULL,
    reviewer_name VARCHAR(50) NOT NULL,
    PRIMARY KEY (reviewer_id)
);


DROP TABLE IF EXISTS Review;
CREATE TABLE Review(
    review_id INT NOT NULL,
    listing_id INT NOT NULL,
    review_date DATE NOT NULL,
    reviewer_id INT NOT NULL,
    comments TEXT,
    PRIMARY KEY (review_id),
    FOREIGN KEY (listing_id) REFERENCES Listing(listing_id),
    FOREIGN KEY (reviewer_id) REFERENCES Reviewer(reviewer_id)
);


DROP TABLE IF EXISTS Covid;
CREATE TABLE Covid(
    neighborhood varchar(50) NOT NULL,
    resident_population INTEGER DEFAULT 0 NOT NULL,
```

```
    cumulative_cases INTEGER,
    new_cases INTEGER,
    PRIMARY KEY (neighborhood)
);


INSERT INTO Admin VALUES
    ('admin', 'adminpwd'),
    ('cs615', 'database');


INSERT INTO Covid VALUES
    ('Bayview Hunters Point', 37394, 1962, 352),
    ('Tenderloin', 29588, 1234, 191),
    ('Mission', 59639, 2274, 456),
    ('Visitacion Valley', 19005, 692, 154),
    ('Excelsior', 40701, 1350, 292),
    ('Outer Mission', 24853, 736, 152),
    ('Portola', 16563, 473, 110),
    ('Japantown', 3532, 97, 12),
    ('South of Market', 21771, 522, 104),
    ('Bernal Heights', 25858, 546, 135),
    ('Oceanview/Merced/Ingleside', 58517, 545, 121),
    ('Western Addition', 22638, 432, 92),
    ('Mission Bay', 12180, 228, 63),
    ('Treasure Island', 3064, 56, 14),
    ('Potrero Hill', 14209, 252, 54),
    ('Hayes Valley', 19678, 304, 76),
    ('Financial District/South Beach', 19458, 300, 90),
    ('Marina', 25331, 371, 145),
    ('Presidio', 4119, 56, 15),
    ('Pacific Heights', 24462, 317, 84),
    ('Castro/Upper Market', 22284, 287, 102),
    ('Russian Hill', 17830, 227, 72),
    ('Nob Hill', 26579, 319, 99),
    ('North Beach', 11636, 137, 37),
    ('Presidio Heights', 10699, 123, 55),
    ('Twin Peaks', 8019, 87, 22),
    ('Noe Valley', 23507, 239, 79),
    ('Lakeshore', 14643, 141, 27),
    ('Inner Richmond', 22220, 203, 64),
    ('West of Twin Peaks', 39496, 355, 123),
    ('Lone Mountain/USF', 17831, 153, 55),
    ('Glen Park', 8641, 73, 19),
```

```
    ('Outer Richmond', 45891, 374, 113),
    ('Inner Sunset', 29307, 217, 58),
    ('Sunset/Parkside', 82410, 583, 213),
    ('Haight Ashbury', 18524, 130, 41),
    ('Chinatown', 14737, 102, 25),
    ('Golden Gate Park', 66, NULL, NULL),
    ('Lincoln Park', 305, NULL, NULL),
    ('McLaren Park', 669, NULL, NULL),
    ('Seacliff', 2490, 12, 3);

SET SQL_SAFE_UPDATES = 0;

ALTER TABLE Covid
ADD COLUMN Rate_of_cases_per_10000 DECIMAL(6, 2)
AFTER cumulative_cases;

UPDATE Covid
SET Rate_of_cases_per_10000 = cumulative_cases/(resident_population/10000);

ALTER TABLE Covid
ADD COLUMN Rate_of_new_cases_per_10000 DECIMAL(6, 2)
AFTER new_cases;

UPDATE Covid
SET Rate_of_new_cases_per_10000 = new_cases/(resident_population/10000);


DROP TABLE IF EXISTS Park;
CREATE TABLE Park(
    parkID INT NOT NULL,
    PSA TEXT,
    park TEXT,
    FQ TEXT,
    score DECIMAL(4, 3) NOT NULL,
    facility_type TEXT,
    facility_name TEXT NOT NULL,
    address TEXT NOT NULL,
    state VARCHAR(2) DEFAULT 'CA',
    zipcode INT,
    floor_count INT,
    square_feet DECIMAL(12, 2),
    perimeter_length DECIMAL(8, 2),
    acres DECIMAL(5, 4),
    longitude FLOAT NOT NULL,
```

```
    latitude FLOAT NOT NULL,
    PRIMARY KEY (parkID)
);

DROP TABLE IF EXISTS Zipcode;
CREATE TABLE Zipcode(
    pairID INT NOT NULL AUTO_INCREMENT,
    airbnb_neighbourhood VARCHAR(30) NOT NULL,
    covid_neighborhood VARCHAR(50) NOT NULL,
    zipcode INT NOT NULL,
    PRIMARY KEY (pairID),
    FOREIGN KEY (airbnb_neighbourhood) REFERENCES Neighbourhood(name),
    FOREIGN KEY (covid_neighborhood) REFERENCES Covid(neighborhood)
);

-- load data from '/data'

LOAD DATA LOCAL INFILE 'data/airbnb_covid_neighborhood.csv'
INTO TABLE Zipcode
FIELDS TERMINATED BY ','
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(airbnb_neighbourhood, covid_neighborhood, zipcode);


LOAD DATA LOCAL INFILE 'data/neighbourhoods.csv'
INTO TABLE Neighbourhood
FIELDS TERMINATED BY ','
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(name, city, state, country);


LOAD DATA LOCAL INFILE 'data/hosts.csv'
IGNORE
INTO TABLE Host
FIELDS TERMINATED BY ','
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(host_id, host_url, host_name, @host_since, host_location, host_about, host_response_time,
```

```
    @host_response_rate, @host_acceptance_rate, host_is_superhost,  host_thumbnail_url,
host_picture_url,
    host_neighbourhood, host_listings_count, host_total_listings_count, host_verifications,
host_has_profile_pic,
    host_identity_verified)
SET host_since = STR_TO_DATE(@host_since, '%m/%d/%Y'),
    host_response_rate = TRIM(TRAILING '%' FROM NULLIF(@host_response_rate, '')),
    host_acceptance_rate = TRIM(TRAILING '%' FROM NULLIF(@host_acceptance_rate, ''));


LOAD DATA LOCAL INFILE 'data/listings.csv'
IGNORE
INTO TABLE Listing
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(listing_id,listing_url,name,description,neighborhood_overview,picture_url,host_id,neighbourhoo
d,neighbourhood_cleansed,

latitude,longitude,property_type,room_type,accommodates,bathrooms_text,bedrooms,beds,am
enities,@price,

minimum_nights,maximum_nights,availability_365,number_of_reviews,@first_review,@last_rev
iew,

review_scores_rating,review_scores_accuracy,review_scores_cleanliness,review_scores_chec
kin,review_scores_communication,

review_scores_location,review_scores_value,instant_bookable,calculated_host_listings_count,
    calculated_host_listings_count_entire_homes,calculated_host_listings_count_private_rooms,
    calculated_host_listings_count_shared_rooms,reviews_per_month)
SET price = TRIM(LEADING '$' FROM NULLIF(@price, '')),
    first_review = STR_TO_DATE(@first_review, '%m/%d/%Y'),
    last_review = STR_TO_DATE(@last_review, '%m/%d/%Y');


LOAD DATA LOCAL INFILE 'data/reviewers.csv'
IGNORE
INTO TABLE Reviewer
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
```

```
(reviewer_id, reviewer_name);


LOAD DATA LOCAL INFILE 'data/reviews.csv'
IGNORE
INTO TABLE Review
FIELDS TERMINATED BY ','
ENCLOSED BY ""
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS
(listing_id, review_id, @review_date, reviewer_id, comments)
SET review_date = STR_TO_DATE(@review_date, '%m/%d/%Y');


LOAD DATA LOCAL INFILE 'data/SF_Park_Scores.csv'
INTO TABLE Park
FIELDS TERMINATED BY ','
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(parkID, PSA, park, FQ, score, facility_type, facility_name, address, state, zipcode, floor_count,
square_feet, perimeter_length, acres, longitude, latitude);
```

procedures.sql

```
DELIMITER //

-- view

DROP VIEW IF EXISTS ParkScore;//
CREATE View ParkScore AS
SELECT Zipcode, COUNT(Zipcode) AS count, AVG(score) AS avg_score
FROM Park
WHERE Zipcode >= 90000
GROUP BY Zipcode;
//


-- show neighborhood info
DROP PROCEDURE IF EXISTS NeighborhoodInfo;//
CREATE PROCEDURE NeighborhoodInfo(IN neighborhood VARCHAR(30))
BEGIN
   IF EXISTS (SELECT *
            FROM Zipcode AS Z
```

```
            WHERE Z.airbnb_neighbourhood = neighborhood) THEN
        SELECT Z.zipcode, Z.airbnb_neighbourhood, C.resident_population, C.cumulative_cases,
C.Rate_of_cases_per_10000, C.new_cases, C.Rate_of_new_cases_per_10000, P.avg_score
        FROM Zipcode as Z
        Join Covid as C ON Z.covid_neighborhood = C.neighborhood
        LEFT JOIN ParkScore as P ON Z.zipcode = P.Zipcode
        WHERE Z.airbnb_neighbourhood = neighborhood;
    ELSE
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
//

-- CALL NeighborhoodInfo('Western Addition');//

-- listing search procedure
DROP PROCEDURE IF EXISTS ListingSearch; //
CREATE PROCEDURE ListingSearch(IN neighborhood VARCHAR(30), IN room_type
VARCHAR(20), IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low
DECIMAL(8,2), IN price_high DECIMAL(8,2), IN offset INT, IN no_of_records_per_page INT)

BEGIN
    IF EXISTS (SELECT listing_id
            FROM Listing AS L
            WHERE L.neighbourhood_cleansed = neighborhood
                AND L.room_type = room_type
                AND L.accommodates >= accommodates
                AND L.bedrooms = bedrooms
                AND L.beds = beds
                AND L.price > price_low
                AND L.price < price_high) THEN
        SELECT L.listing_id, L.listing_url, L.name, L.description, L.neighborhood_overview,
L.picture_url, L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
L.review_scores_rating, L.bathrooms_text, L.latitude, L.longitude
        FROM Listing AS L
        WHERE L.neighbourhood_cleansed = neighborhood
            AND L.room_type = room_type
            AND L.accommodates >= accommodates
            AND L.bedrooms >= bedrooms
            AND L.beds >= beds
            AND L.price >= price_low
            AND L.price <= price_high
            LIMIT offset, no_of_records_per_page;
    ELSE
```

```
    SELECT 'No data is found.' AS 'Error Message';
  END IF;
END;
//

-- listing search procedure and sort by price
DROP PROCEDURE IF EXISTS ListingSearchSortByPrice; //
CREATE PROCEDURE ListingSearchSortByPrice(IN neighborhood VARCHAR(30), IN
room_type VARCHAR(20),
    IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN
price_high DECIMAL(8,2),
    IN offset INT, IN no_of_records_per_page INT)

BEGIN
  IF EXISTS (SELECT listing_id
        FROM Listing AS L
        WHERE L.neighbourhood_cleansed = neighborhood
          AND L.room_type = room_type
          AND L.accommodates >= accommodates
          AND L.bedrooms = bedrooms
          AND L.beds = beds
          AND L.price > price_low
          AND L.price < price_high) THEN
    SELECT L.listing_id, L.listing_url, L.name, L.description, L.neighborhood_overview,
L.picture_url,
      L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
L.review_scores_rating,
      L.bathrooms_text, L.latitude, L.longitude
    FROM Listing AS L
    WHERE L.neighbourhood_cleansed = neighborhood
      AND L.room_type = room_type
      AND L.accommodates >= accommodates
      AND L.bedrooms >= bedrooms
      AND L.beds >= beds
      AND L.price >= price_low
      AND L.price <= price_high
    ORDER BY L.price
    LIMIT offset, no_of_records_per_page;
  ELSE
    SELECT 'No data is found.' AS 'Error Message';
  END IF;
END;
//
```

```
-- listing search procedure and sort by price
DROP PROCEDURE IF EXISTS ListingSearchSortByReview; //
CREATE PROCEDURE ListingSearchSortByReview(IN neighborhood VARCHAR(30), IN
room_type VARCHAR(20),
   IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN
price_high DECIMAL(8,2),
   IN offset INT, IN no_of_records_per_page INT)

BEGIN
   IF EXISTS (SELECT listing_id
            FROM Listing AS L
            WHERE L.neighbourhood_cleansed = neighborhood
               AND L.room_type = room_type
               AND L.accommodates >= accommodates
               AND L.bedrooms = bedrooms
               AND L.beds = beds
               AND L.price > price_low
               AND L.price < price_high) THEN
      SELECT L.listing_id, L.listing_url, L.name, L.description, L.neighborhood_overview,
         L.picture_url, L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
         L.review_scores_rating, L.bathrooms_text, L.latitude, L.longitude
      FROM Listing AS L
      WHERE L.neighbourhood_cleansed = neighborhood
         AND L.room_type = room_type
         AND L.accommodates >= accommodates
         AND L.bedrooms >= bedrooms
         AND L.beds >= beds
         AND L.price >= price_low
         AND L.price <= price_high
      ORDER BY L.number_of_reviews DESC
      LIMIT offset, no_of_records_per_page;
   ELSE
      SELECT 'No data is found.' AS 'Error Message';
   END IF;
END;
//

-- listing search procedure and sort by price
DROP PROCEDURE IF EXISTS ListingSearchSortByRating; //
CREATE PROCEDURE ListingSearchSortByRating(IN neighborhood VARCHAR(30), IN
room_type VARCHAR(20),
   IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN
price_high DECIMAL(8,2),
```

```
    IN offset INT, IN no_of_records_per_page INT)

BEGIN
    IF EXISTS (SELECT listing_id
                FROM Listing AS L
                WHERE L.neighbourhood_cleansed = neighborhood
                    AND L.room_type = room_type
                    AND L.accommodates >= accommodates
                    AND L.bedrooms = bedrooms
                    AND L.beds = beds
                    AND L.price > price_low
                    AND L.price < price_high) THEN
        SELECT L.listing_id, L.listing_url, L.name, L.description, L.neighborhood_overview,
L.picture_url,
            L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
L.review_scores_rating, L.bathrooms_text,
            L.latitude, L.longitude
        FROM Listing AS L
        WHERE L.neighbourhood_cleansed = neighborhood
            AND L.room_type = room_type
            AND L.accommodates >= accommodates
            AND L.bedrooms >= bedrooms
            AND L.beds >= beds
            AND L.price >= price_low
            AND L.price <= price_high
        ORDER BY L.review_scores_rating DESC
        LIMIT offset, no_of_records_per_page;
    ELSE
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
//

DROP PROCEDURE IF EXISTS FindListingByID; //
CREATE PROCEDURE FindListingByID(IN listing_id INT)
BEGIN
    IF EXISTS(SELECT listing_id FROM Listing) THEN
        SELECT L.name, L.neighbourhood_cleansed, L.picture_url, L.description, L.amenities,
L.host_id, L.neighborhood_overview,
            L.property_type, L.price, L.number_of_reviews, L.review_scores_rating,
L.bathrooms_text, L.latitude, L.longitude
        FROM Listing AS L
        WHERE L.listing_id = listing_id;
    ELSE
```

```
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
//

DROP PROCEDURE IF EXISTS FindHostIDByListingID; //
CREATE PROCEDURE FindHostIDByListingID(IN listing_id INT)
BEGIN
    IF EXISTS(SELECT listing_id FROM Listing) THEN
        SELECT host_id
        FROM Listing AS L
        WHERE L.listing_id = listing_id;
    ELSE
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
//

DROP PROCEDURE IF EXISTS FindHostByID; //
CREATE PROCEDURE FindHostByID(IN host_id INT)
BEGIN
    IF EXISTS(SELECT host_id FROM Host) THEN
        SELECT host_id, host_name, host_since, host_about, host_picture_url,
host_neighbourhood
        FROM Host AS H
        WHERE H.host_id = host_id;
    ELSE
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
//

DROP PROCEDURE IF EXISTS FindReviewByListingID; //
CREATE PROCEDURE FindReviewByListingID(IN listing_id INT)
BEGIN
    IF EXISTS(SELECT listing_id FROM Review) THEN
        SELECT Review.review_date, Review.comments, Reviewer.reviewer_name
        FROM Review, Reviewer
        WHERE Review.reviewer_id = Reviewer.reviewer_id
            AND Review.listing_id = listing_id;
    ELSE
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
```

```
//


DROP PROCEDURE IF EXISTS UpdateHost;  //
CREATE PROCEDURE UpdateHost(IN host_id INT,
                    IN host_url TEXT,
                    IN host_name TEXT,
                    IN host_since DATE,
                    IN host_location TEXT,
                    IN host_about TEXT,
                    IN host_response_time TEXT,
                    IN host_response_rate INT,
                    IN host_acceptance_rate INT,
                    IN host_is_superhost VARCHAR(1),
                    IN host_thumbnail_url TEXT,
                    IN host_picture_url TEXT,
                    IN host_neighbourhood TEXT,
                    IN host_listings_count INT,
                    IN host_total_listings_count INT,
                    IN host_verifications TEXT,
                    IN host_has_profile_pic VARCHAR(1),
                    IN host_identity_verified VARCHAR(2))
BEGIN
   INSERT INTO Host
   VALUES (host_id,
         host_url,
         host_name,
         host_since,
         host_location,
         host_about,
         host_response_time,
         host_response_rate,
         host_acceptance_rate,
         host_is_superhost,
         host_thumbnail_url,
         host_picture_url,
         host_neighbourhood,
         host_listings_count,
         host_total_listings_count,
         host_verifications,
         host_has_profile_pic,
         host_identity_verified)
   ON DUPLICATE KEY UPDATE
   host_url = VALUES(host_url),
```

```
    host_name = VALUES(host_name),
    host_since = VALUES(host_since),
    host_location = VALUES(host_location),
    host_about = VALUES(host_about),
    host_response_time = VALUES(host_response_time),
    host_response_rate = VALUES(host_response_rate),
    host_acceptance_rate = VALUES(host_acceptance_rate),
    host_is_superhost = VALUES(host_is_superhost),
    host_thumbnail_url = VALUES(host_thumbnail_url),
    host_picture_url = VALUES(host_picture_url),
    host_neighbourhood = VALUES(host_neighbourhood),
    host_listings_count = VALUES(host_listings_count),
    host_total_listings_count = VALUES(host_total_listings_count),
    host_verifications = VALUES(host_verifications),
    host_has_profile_pic = VALUES(host_has_profile_pic),
    host_identity_verified = VALUES(host_identity_verified);
END;
//

-- CALL UpdateHost(1234, 'real url', 'fake host', '2020-12-12', 'baltimore', 'this is a fake host',
'reponse time', 99, 99, 'f', 'thumbnail_url', 'picture_url',
-- 'neighbourhood', 9, 9, 'email', 't', 'f');//
-- CALL UpdateHost(1234, 'change url', 'fake host', '2020-12-12', 'baltimore', 'this is a fake host',
'reponse time', 99, 99, 'f', 'thumbnail_url', 'picture_url',
-- 'neighbourhood', 9, 9, 'email', 't', 'f');//

DROP PROCEDURE IF EXISTS DeleteHost; //
CREATE PROCEDURE DeleteHost(IN host_id INT)
BEGIN
    IF EXISTS (SELECT Host.host_id FROM Host WHERE Host.host_id = host_id) THEN
    DELETE FROM Host WHERE Host.host_id = host_id;
    END IF;
END;
//
-- CALL DeleteHost(1234);//


DROP PROCEDURE IF EXISTS CheckHost; //
CREATE PROCEDURE CheckHost(IN host_id INT)
BEGIN
    IF EXISTS (SELECT Host.host_id FROM Host WHERE Host.host_id = host_id) THEN
        SELECT 'Host exists' AS 'Message';
    ELSE
        SELECT 'Host does not exists' AS 'Message';
```

```
   END IF;
END;
//

-- CALL CheckHost(1169);//
-- CALL CheckHost(1234);//

DROP PROCEDURE IF EXISTS UpdateListing;  //
CREATE PROCEDURE UpdateListing(IN listing_id INT,
   IN listing_url TEXT,
   IN name TEXT,
   IN description TEXT,
   IN neighborhood_overview TEXT,
   IN picture_url TEXT,
   IN host_id INT,
   IN neighbourhood VARCHAR(100),
   IN neighbourhood_cleansed VARCHAR(30),
   IN latitude FLOAT,
   IN longitude FLOAT,
   IN property_type VARCHAR(100),
   IN room_type VARCHAR(20),
   IN accommodates INT,
   IN bathrooms_text VARCHAR(20),
   IN bedrooms INT,
   IN beds INT,
   IN amenities TEXT,
   IN price DECIMAL(8, 2),
   IN minimum_nights INT,
   IN maximum_nights INT,
   IN availability_365 INT,
   IN number_of_reviews INT,
   IN first_review DATE,
   IN last_review DATE,
   IN review_scores_rating INT,
   IN review_scores_accuracy INT,
   IN review_scores_cleanliness INT,
   IN review_scores_checkin INT,
   IN review_scores_communication INT,
   IN review_scores_location INT,
   IN review_scores_value INT,
   IN instant_bookable VARCHAR(1),
   IN calculated_host_listings_count INT,
   IN calculated_host_listings_count_entire_homes INT,
   IN calculated_host_listings_count_private_rooms INT,
```

```
    IN calculated_host_listings_count_shared_rooms INT,
    IN reviews_per_month DECIMAL(3, 2))
BEGIN
    INSERT INTO Listing
    VALUES (listing_id,
            listing_url,
            name,
            description,
            neighborhood_overview,
            picture_url,
            host_id,
            neighbourhood,
            neighbourhood_cleansed,
            latitude,
            longitude,
            property_type,
            room_type,
            accommodates,
            bathrooms_text,
            bedrooms,
            beds,
            amenities,
            price,
            minimum_nights,
            maximum_nights,
            availability_365,
            number_of_reviews,
            first_review,
            last_review,
            review_scores_rating,
            review_scores_accuracy,
            review_scores_cleanliness,
            review_scores_checkin,
            review_scores_communication,
            review_scores_location,
            review_scores_value,
            instant_bookable,
            calculated_host_listings_count,
            calculated_host_listings_count_entire_homes,
            calculated_host_listings_count_private_rooms,
            calculated_host_listings_count_shared_rooms,
            reviews_per_month)
    ON DUPLICATE KEY UPDATE
       listing_url = VALUES(listing_url),
```

```
        name = VALUES(name),
        description = VALUES(description),
        neighborhood_overview = VALUES(neighborhood_overview),
        picture_url = VALUES(picture_url),
        host_id = VALUES(host_id),
        neighbourhood = VALUES(neighbourhood),
        neighbourhood_cleansed = VALUES(neighbourhood_cleansed),
        latitude = VALUES(latitude),
        longitude = VALUES(longitude),
        property_type = VALUES(property_type),
        room_type = VALUES(room_type),
        accommodates = VALUES(accommodates),
        bathrooms_text = VALUES(bathrooms_text),
        bedrooms = VALUES(bedrooms),
        beds = VALUES(beds),
        amenities = VALUES(amenities),
        price = VALUES(price),
        minimum_nights = VALUES(minimum_nights),
        maximum_nights = VALUES(maximum_nights),
        availability_365 = VALUES(availability_365),
        number_of_reviews = VALUES(number_of_reviews),
        first_review = VALUES(first_review),
        last_review = VALUES(last_review),
        review_scores_rating = VALUES(review_scores_rating),
        review_scores_accuracy = VALUES(review_scores_accuracy),
        review_scores_cleanliness = VALUES(review_scores_cleanliness),
        review_scores_checkin = VALUES(review_scores_checkin),
        review_scores_communication = VALUES(review_scores_communication),
        review_scores_location = VALUES(review_scores_location),
        review_scores_value = VALUES(review_scores_value),
        instant_bookable = VALUES(instant_bookable),
        calculated_host_listings_count = VALUES(calculated_host_listings_count),
        calculated_host_listings_count_entire_homes =
VALUES(calculated_host_listings_count_entire_homes),
        calculated_host_listings_count_private_rooms =
VALUES(calculated_host_listings_count_private_rooms),
        calculated_host_listings_count_shared_rooms =
VALUES(calculated_host_listings_count_shared_rooms),
        reviews_per_month = VALUES(reviews_per_month);
END;
//
```

```
-- CALL UpdateListing(1234, 'www', 'big room', 'desc', 'nb overview', 'url', 1169, 'sf', 'Western Addition', 37.7000, -122.333, 'Entire', 'apt', 4, 'no bath??', 1, 1, 'ps5', 99.99, 1, 30, 200, 200, '2008-08-08', '2020-10-15', 99, 9, 9, 9, 9, 8, 9, 'f', 2, 3, 0, 0, 1.3);//


DROP PROCEDURE IF EXISTS DeleteListing; //
CREATE PROCEDURE DeleteListing(IN listing_id INT)
BEGIN
   IF EXISTS (SELECT Listing.listing_id FROM Listing WHERE Listing.listing_id = listing_id) THEN
   DELETE FROM Listing WHERE Listing.listing_id = listing_id;
   END IF;
END;
//
-- CALL DeleteListing(1234);//

DROP PROCEDURE IF EXISTS WordSearch; //
CREATE PROCEDURE WordSearch(IN inputWord TEXT, IN offset INT, IN no_of_records_per_page INT)
BEGIN
   SELECT listing_id
   FROM Listing
   WHERE name LIKE CONCAT('%', inputWord, '%') OR description LIKE CONCAT('%', inputWord, '%') OR neighborhood_overview LIKE CONCAT('%', inputWord, '%') OR amenities LIKE CONCAT('%', inputWord, '%');
END;
//

-- listing search with words procedure
DROP PROCEDURE IF EXISTS WordListingSearch; //
CREATE PROCEDURE WordListingSearch(IN inputWord TEXT, IN neighborhood VARCHAR(30), IN room_type VARCHAR(20), IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN price_high DECIMAL(8,2), IN offset INT, IN no_of_records_per_page INT)

BEGIN
   IF EXISTS (SELECT listing_id
            FROM Listing AS L
            WHERE L.neighbourhood_cleansed = neighborhood
               AND L.room_type = room_type
               AND L.accommodates >= accommodates
               AND L.bedrooms = bedrooms
               AND L.beds = beds
               AND L.price > price_low
```

```
                AND L.price < price_high) THEN
        SELECT s.listing_id, s.listing_url, s.name, s.description, s.neighborhood_overview,
s.picture_url,
            s.host_id, s.property_type, s.amenities, s.price, s.number_of_reviews,
s.review_scores_rating,
            s.bathrooms_text, s.latitude, s.longitude
        FROM (SELECT L.listing_id, L.listing_url, L.name, L.description,
L.neighborhood_overview, L.picture_url,
                L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
L.review_scores_rating,
                L.bathrooms_text, L.latitude, L.longitude
                FROM Listing as L
                WHERE L.neighbourhood_cleansed = neighborhood
                AND L.room_type = room_type
                AND L.accommodates >= accommodates
                AND L.bedrooms >= bedrooms
                AND L.beds >= beds
                AND L.price >= price_low
                AND L.price <= price_high) AS s
        WHERE s.name LIKE CONCAT('%', inputWord, '%') OR s.description LIKE CONCAT('%',
inputWord, '%') OR
            s.neighborhood_overview LIKE CONCAT('%', inputWord, '%') OR s.amenities LIKE
CONCAT('%', inputWord, '%')
        LIMIT offset, no_of_records_per_page;
    ELSE
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
//
-- call WordListingSearch('cul de sac', 'Western Addition', 'Entire home/apt', 3, 1, 2, 100, 150, 0,
10);//


-- listing search procedure and sort by price
DROP PROCEDURE IF EXISTS WordListingSearchSortByPrice; //
CREATE PROCEDURE WordListingSearchSortByPrice(IN inputWord TEXT, IN neighborhood
VARCHAR(30), IN room_type VARCHAR(20), IN accommodates INT, IN bedrooms INT, IN
beds INT, IN price_low DECIMAL(8,2), IN price_high DECIMAL(8,2), IN offset INT, IN
no_of_records_per_page INT)

BEGIN
    IF EXISTS (SELECT listing_id
                FROM Listing AS L
                WHERE L.neighbourhood_cleansed = neighborhood
```

```
                AND L.room_type = room_type
                AND L.accommodates >= accommodates
                AND L.bedrooms = bedrooms
                AND L.beds = beds
                AND L.price > price_low
                AND L.price < price_high) THEN
        SELECT s.listing_id, s.listing_url, s.name, s.description, s.neighborhood_overview,
s.picture_url, s.host_id,
            s.property_type, s.amenities, s.price, s.number_of_reviews, s.review_scores_rating,
            s.bathrooms_text, s.latitude, s.longitude
        FROM (SELECT L.listing_id, L.listing_url, L.name, L.description,
L.neighborhood_overview,
                L.picture_url, L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
                L.review_scores_rating, L.bathrooms_text, L.latitude, L.longitude
                FROM Listing as L
                WHERE L.neighbourhood_cleansed = neighborhood
                AND L.room_type = room_type
                AND L.accommodates >= accommodates
                AND L.bedrooms >= bedrooms
                AND L.beds >= beds
                AND L.price >= price_low
                AND L.price <= price_high) AS s
        WHERE s.name LIKE CONCAT('%', inputWord, '%') OR s.description LIKE CONCAT('%',
inputWord, '%') OR
            s.neighborhood_overview LIKE CONCAT('%', inputWord, '%') OR s.amenities LIKE
CONCAT('%', inputWord, '%')
        ORDER BY s.price
        LIMIT offset, no_of_records_per_page;
    ELSE
        SELECT 'No data is found.' AS 'Error Message';
    END IF;
END;
//


-- listing search procedure and sort by price
DROP PROCEDURE IF EXISTS WordListingSearchSortByReview; //
CREATE PROCEDURE WordListingSearchSortByReview(IN inputWord TEXT, IN
neighborhood VARCHAR(30), IN room_type VARCHAR(20),
    IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN
price_high DECIMAL(8,2),
    IN offset INT, IN no_of_records_per_page INT)

BEGIN
```

```
   IF EXISTS (SELECT listing_id
            FROM Listing AS L
            WHERE L.neighbourhood_cleansed = neighborhood
               AND L.room_type = room_type
               AND L.accommodates >= accommodates
               AND L.bedrooms = bedrooms
               AND L.beds = beds
               AND L.price > price_low
               AND L.price < price_high) THEN
      SELECT s.listing_id, s.listing_url, s.name, s.description, s.neighborhood_overview,
s.picture_url,
         s.host_id, s.property_type, s.amenities, s.price, s.number_of_reviews,
s.review_scores_rating,
         s.bathrooms_text, s.latitude, s.longitude
      FROM (SELECT L.listing_id, L.listing_url, L.name, L.description,
L.neighborhood_overview,
            L.picture_url, L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
            L.review_scores_rating, L.bathrooms_text, L.latitude, L.longitude
            FROM Listing as L
            WHERE L.neighbourhood_cleansed = neighborhood
            AND L.room_type = room_type
            AND L.accommodates >= accommodates
            AND L.bedrooms >= bedrooms
            AND L.beds >= beds
            AND L.price >= price_low
            AND L.price <= price_high) AS s
      WHERE s.name LIKE CONCAT('%', inputWord, '%') OR s.description LIKE CONCAT('%',
inputWord, '%') OR
      s.neighborhood_overview LIKE CONCAT('%', inputWord, '%') OR s.amenities LIKE
CONCAT('%', inputWord, '%')
      ORDER BY s.number_of_reviews DESC
      LIMIT offset, no_of_records_per_page;
   ELSE
      SELECT 'No data is found.' AS 'Error Message';
   END IF;
END;
//

-- listing search procedure and sort by price
DROP PROCEDURE IF EXISTS WordListingSearchSortByRating; //
CREATE PROCEDURE WordListingSearchSortByRating(IN inputWord TEXT, IN neighborhood
VARCHAR(30), IN room_type VARCHAR(20),
   IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN
price_high DECIMAL(8,2),
```

```
   IN offset INT, IN no_of_records_per_page INT)

BEGIN
   IF EXISTS (SELECT listing_id
            FROM Listing AS L
            WHERE L.neighbourhood_cleansed = neighborhood
               AND L.room_type = room_type
               AND L.accommodates >= accommodates
               AND L.bedrooms = bedrooms
               AND L.beds = beds
               AND L.price > price_low
               AND L.price < price_high) THEN
      SELECT s.listing_id, s.listing_url, s.name, s.description, s.neighborhood_overview,
s.picture_url,
         s.host_id, s.property_type, s.amenities, s.price, s.number_of_reviews,
s.review_scores_rating,
         s.bathrooms_text, s.latitude, s.longitude
      FROM (SELECT L.listing_id, L.listing_url, L.name, L.description,
L.neighborhood_overview,
            L.picture_url, L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
            L.review_scores_rating, L.bathrooms_text, L.latitude, L.longitude
            FROM Listing as L
            WHERE L.neighbourhood_cleansed = neighborhood
            AND L.room_type = room_type
            AND L.accommodates >= accommodates
            AND L.bedrooms >= bedrooms
            AND L.beds >= beds
            AND L.price >= price_low
            AND L.price <= price_high) AS s
      WHERE s.name LIKE CONCAT('%', inputWord, '%') OR s.description LIKE CONCAT('%',
inputWord, '%') OR
      s.neighborhood_overview LIKE CONCAT('%', inputWord, '%') OR s.amenities LIKE
CONCAT('%', inputWord, '%')
      ORDER BY s.review_scores_rating DESC
      LIMIT offset, no_of_records_per_page;
   ELSE
      SELECT 'No data is found.' AS 'Error Message';
   END IF;
END;
//


DROP PROCEDURE IF EXISTS ParkAnalysis; //
CREATE PROCEDURE ParkAnalysis()
```

```
BEGIN

   Select zipcode, COUNT(park) AS park_count, AVG(score) AS average_park_score,
AVG(square_feet) AS average_square_feet
   FROM Park
   WHERE zipcode != 0
   GROUP BY zipcode
   ORDER BY zipcode ASC;

END;
//

DROP PROCEDURE IF EXISTS ListingAnalysis; //
CREATE PROCEDURE ListingAnalysis()
BEGIN
   SELECT neighbourhood_cleansed as neighborhood,
         COUNT(listing_id) AS listing_count,
         AVG(review_scores_rating) AS average_review_rating,
         AVG(review_scores_accuracy) AS average_accuracy,
         AVG(review_scores_cleanliness) AS average_cleanliness,
         AVG(review_scores_checkin) AS average_checkin,
         AVG(review_scores_communication) AS average_communication,
         AVG(review_scores_location) AS average_location,
         AVG(review_scores_value) AS average_value
   FROM Listing
   GROUP BY neighbourhood_cleansed
   ORDER BY neighbourhood_cleansed ASC;
END;
//


DROP PROCEDURE IF EXISTS CovidAnalysis; //
CREATE PROCEDURE CovidAnalysis()
BEGIN
   SELECT * FROM Covid
   ORDER BY cumulative_cases DESC;
END;
//


DROP PROCEDURE IF EXISTS AllAnalysis; //
CREATE PROCEDURE AllAnalysis()
BEGIN
   SELECT z.airbnb_neighbourhood AS neighborhood,
```

```
        l.listing_count, l.average_review_rating,
        p.park_count, p.average_park_score,
        c.cumulative_cases, c.new_cases
    FROM Zipcode as z,
      (SELECT neighbourhood_cleansed as neighborhood,
        COUNT(listing_id) AS listing_count,
        AVG(review_scores_rating) AS average_review_rating,
        AVG(review_scores_accuracy) AS average_accuracy,
        AVG(review_scores_cleanliness) AS average_cleanliness,
        AVG(review_scores_checkin) AS average_checkin,
        AVG(review_scores_communication) AS average_communication,
        AVG(review_scores_location) AS average_location,
        AVG(review_scores_value) AS average_value
      FROM Listing
      GROUP BY neighbourhood_cleansed) as l,
      (Select zipcode, COUNT(park) AS park_count, AVG(score) AS average_park_score
      FROM Park
      WHERE zipcode != 0
      GROUP BY zipcode) as p,
      Covid as c
    WHERE z.airbnb_neighbourhood = l.neighborhood
    AND z.covid_neighborhood = c.neighborhood
    AND z.zipcode = p.zipcode
    ORDER BY z.airbnb_neighbourhood;
END;
//

DROP PROCEDURE IF EXISTS Location;  //
CREATE PROCEDURE Location()
BEGIN
    SELECT z.airbnb_neighbourhood AS neighborhood,
        l.listing_count, l.average_review_rating,
        p.park_count, p.average_park_score,
        c.cumulative_cases, c.new_cases
    FROM Zipcode as z,
      (SELECT neighbourhood_cleansed as neighborhood,
        COUNT(listing_id) AS listing_count,
        AVG(review_scores_rating) AS average_review_rating,
        AVG(review_scores_accuracy) AS average_accuracy,
        AVG(review_scores_cleanliness) AS average_cleanliness,
        AVG(review_scores_checkin) AS average_checkin,
        AVG(review_scores_communication) AS average_communication,
        AVG(review_scores_location) AS average_location,
        AVG(review_scores_value) AS average_value
```

```
        FROM Listing
        GROUP BY neighbourhood_cleansed) as l,
        (Select zipcode, COUNT(park) AS park_count, AVG(score) AS average_park_score
        FROM Park
        WHERE zipcode != 0
        GROUP BY zipcode) as p,
        Covid as c
    WHERE z.airbnb_neighbourhood = l.neighborhood
    AND z.covid_neighborhood = c.neighborhood
    AND z.zipcode = p.zipcode
    ORDER BY z.airbnb_neighbourhood;
END;
//


DROP PROCEDURE IF EXISTS HealthRec;  //
CREATE PROCEDURE HealthRec()
BEGIN
    SELECT z.airbnb_neighbourhood AS neighborhood,
            l.listing_count, l.average_location_rating,
            p.park_count,
            c.cumulative_cases,
            c.new_cases,
            c.new_cases/c.cumulative_cases AS new_to_cumulative_ratio
    FROM Zipcode as z,
        (SELECT neighbourhood_cleansed as neighborhood,
            COUNT(listing_id) AS listing_count,
            AVG(review_scores_location) AS average_location_rating
        FROM Listing
        GROUP BY neighbourhood_cleansed) as l,
        (Select zipcode, COUNT(park) AS park_count, AVG(score) AS average_park_score
        FROM Park
        WHERE zipcode != 0
        GROUP BY zipcode) as p,
        Covid as c
    WHERE z.airbnb_neighbourhood = l.neighborhood
    AND z.covid_neighborhood = c.neighborhood
    AND z.zipcode = p.zipcode
    AND cumulative_cases != 'NULL'
    ORDER BY new_to_cumulative_ratio ASC, l.average_location_rating DESC, p.park_count
DESC;
END;
//
```

```
DROP PROCEDURE IF EXISTS CountListingSearch;  //
CREATE PROCEDURE CountListingSearch(IN neighborhood VARCHAR(30), IN room_type
VARCHAR(20), IN accommodates INT,
   IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN price_high DECIMAL(8,2))
BEGIN
   SELECT COUNT(*)
   FROM Listing AS L
   WHERE L.neighbourhood_cleansed = neighborhood
      AND L.room_type = room_type
      AND L.accommodates >= accommodates
      AND L.bedrooms >= bedrooms
      AND L.beds >= beds
      AND L.price >= price_low
      AND L.price <= price_high;
END;
//

DROP PROCEDURE IF EXISTS CountWordSearch;  //
CREATE PROCEDURE CountWordSearch(IN inputWord TEXT, IN neighborhood
VARCHAR(30), IN room_type VARCHAR(20),
   IN accommodates INT, IN bedrooms INT, IN beds INT, IN price_low DECIMAL(8,2), IN
price_high DECIMAL(8,2))
BEGIN
   SELECT COUNT(*)
   FROM (SELECT L.listing_id, L.listing_url, L.name, L.description, L.neighborhood_overview,
L.picture_url,
         L.host_id, L.property_type, L.amenities, L.price, L.number_of_reviews,
L.review_scores_rating, L.bathrooms_text
            FROM Listing as L
            WHERE L.neighbourhood_cleansed = neighborhood
            AND L.room_type = room_type
            AND L.accommodates >= accommodates
            AND L.bedrooms >= bedrooms
            AND L.beds >= beds
            AND L.price >= price_low
            AND L.price <= price_high) AS s
   WHERE s.name LIKE CONCAT('%', inputWord, '%') OR s.description LIKE CONCAT('%',
inputWord, '%')
      OR s.neighborhood_overview LIKE CONCAT('%', inputWord, '%') OR s.amenities LIKE
CONCAT('%', inputWord, '%');
END;
//
```

Final Project Phase I

data source:

Inside Airbnb

covid total cases in SF by neighborhood

covid cases in last 30 days in SF by neighborhood

sentiment analysis

(1) Who are your team members

Nanxi Ye and Linghao Jin

(2) Target domain

Airbnb listing information in San Francisco, CA combined with SF neighborhood data including covid cases, demographic, income level etc.

(3) List of questions

```
(1) Show the listings start hosting in 2008 with review score over 95.
(2) Show the name, host_acceptance_time and host_response_rate of the hosts who have more than 3 listings currently.
(3) Show the listings by hosts who is related to UCSF in their host_about.
(4) Show host_response_time and host_response_rate of those who have different host_location and listing location.
(5) Show listings that offers entire home/apartment with more than 3 accommodations and coffee maker in neighborhood Financial Distr
(6) Show the average price of listings in Financial District.
(7) Show the average rating of listings with price less than $100, between $100 and $200, between $200 and $300 and above $300.
(8) Show listings from hosts who were originally not from San Francisco. (from host_about)
(9) Show the average rental price of each neighborhood in San Francisco.
(10) Show the entire houses that are 20 miles (Euclidean distance) away from my current location.
(11) Show the percentage of reviewers have reviewed multiple listings under 50 in Airbnb.
(12) Show the review rating score stats(average, max, min) or listings group by property type, order by average review rating score.
(13) Show the listings that are instant bookable, have over 50 reviews, rated above 90 and have over 30 days available in a year.
(14) Show all the listings that owned by the same superhost who responses within an hour and are verified by government id.
(15) Show the name of reviewers who reviewed most each year.
(16) Show the listings that have good views (name including view) that are rated highest in each neighborhood.
```

(4) Relational data model

```
DROP TABLE Listing;
CREATE TABLE Listing(
    id INT NOT NULL,
    listing_url VARCHAR(100) NOT NULL,
    name VARCHAR(100) NOT NULL,
    description VARCHAR(2000) NOT NULL,
    neighborhood_overview VARCHAR(2000) NOT NULL,
    picture_url VARCHAR(100) NOT NULL,
    host_id INT NOT NULL,
    neighbourhood VARCHAR(20) NOT NULL,
    neighbourhood_cleansed VARCHAR(20) NOT NULL, -- Neighborhood.id?
    latitude DECIMAL(8, 5) NOT NULL,
    longitude DECIMAL(8, 5) NOT NULL,
    property_type VARCHAR(20) NOT NULL,
    room_type VARCHAR(20) NOT NULL,
    accommodates INT NOT NULL,
    bathrooms_text VARCHAR(20) NOT NULL,
    bedrooms INT NOT NULL,
    beds INT NOT NULL,
    amenities VARCHAR(1000) NOT NULL,
    price DECIMAL NOT NULL,
    minimum_nights INT NOT NULL,
    maximum_nights INT NOT NULL,
    availability_365 INT NOT NULL,
    number_of_reviews INT NOT NULL,
    first_review DATE NOT NULL,
    last_review DATE NOT NULL,
    review_scores_rating INT NOT NULL,
    review_scores_accuracy INT NOT NULL,
    review_scores_cleanliness INT NOT NULL,
```

```
    review_scores_checkin INT NOT NULL,
    review_scores_communication INT NOT NULL,
    review_scores_location INT NOT NULL,
    review_scores_value INT NOT NULL,
    instant_bookable BOOLEAN NOT NULL,
    calculated_host_listings_count INT NOT NULL,
    calculated_host_listings_count_entire_homes INT NOT NULL,
    calculated_host_listings_count_private_rooms INT NOT NULL,
    calculated_host_listings_count_shared_rooms INT NOT NULL,
    reviews_per_month DECIMAL(3, 2) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (host_id) REFERENCES Host.host_id
);

DROP TABLE Host;
CREATE TABLE Host(
    host_id INT NOT NULL,
    host_url VARCHAR(100) NOT NULL,
    host_name VARCHAR(20) NOT NULL,
    host_since VARCHAR(10)NOT NULL,
    host_location VARCHAR(20) NOT NULL, -- Neighborhood.id?
    host_about VARCHAR(2000)NOT NULL,
    host_response_time VARCHAR(50),
    host_response_rate INT,
    host_acceptance_rate INT NOT NULL,
    host_is_superhost VARCHAR(1) NOT NULL,
    host_thumbnail_url VARCHAR(100) NOT NULL,
    host_picture_url VARCHAR(100) NOT NULL,
    host_neighbourhood VARCHAR(20) NOT NULL,
    host_listings_count INT NOT NULL,
    host_total_listings_count INT NOT NULL,
    host_verifications VARCHAR(100) NOT NULL,
    host_has_profile_pic VARCHAR(1) NOT NULL,
    host_identity_verified VARCHAR(1) NOT NULL,
    PRIMARY KEY (host_id)
);

DROP TABLE Neighbourhood;
CREATE TABLE Neighbourhood(
    id INT NOT NULL,
    name VARCHAR(20) NOT NULL,
    city VARCHAR(20) NOT NULL,
    state VARCHAR(20) NOT NULL,
    country VARCHAR(20) NOT NULL,
    covid_case INT
    PRIMARY KEy (id)
);

DROP TABLE Review;
CREATE TABLE Review(
    id INT NOT NULL,
    listing_id INT NOT NULL,
    date DATE NOT NULL,
    reviewer_id INT NOT NULL,
    comments VARCHAR(2000),
    sentiment_score INT,
    PRIMARY KEY (id),
    FOREIGN KEY (listing_id) REFERENCES Listing.id,
    FOREIGN KEY (reviewer_id) REFERENCES Reviewer.reviewer_id
);

DROP TABLE Reviewer;
CREATE TABLE Reviewer(
    reviewer_id INT NOT NULL,
    reviewer_name VARCHAR(20) NOT NULL,
    PRIMARY KEY (reviewer_id)
```

```
    PRIMARY KEY (reviewer_id)
);
```

(5) SQL statements for representative sample of target queries

```
/* Show the name, host_acceptance_time and host_response_rate of the hosts who have more than 3 listings currently. */
SELECT host_name, host_acceptance_time, host_response_rate
FROM Hosts AS H
WHERE EXISTS (
    SELECT *
    FROM listings AS L1, Listings AS L2, Listings AS L3
    WHERE L1.host_id = L2.host_id AND
    L2.host_id = L3.host_id AND
    H.host_id = L1.host_id AND
    L1.id <> L2.id AND
    L2.id <> L3.id AND
    L1.id <> L3.id);


/* Show host_response_time and host_response_rate of those who have different host_location and listing location. */
SELECT host_response_time, host_response_rate
FROM Hosts AS H, Neighborhood N1, Listings as L, Neighborhood N2
WHERE H.id = L.host_id AND
    N1.id = H.neighborhood_id AND
    N2.id = L.neighborhood_id AND
    N1.id <> N2.id;


/* Show the average rental price of each neighborhood in San Francisco. */
SELECT N.name, AVG(price)
FROM Listings AS L
JOIN Neighborhood AS N ON L.neighborhood_id = N.id
WHERE N.city = 'San Francisco'
GROUP BY N.id;


/* Show the percentage of reviewers have reviewed over 10 listings in Airbnb. */
SELECT COUNT(distinct reviewer_id) / r2_ids
FROM Reviews, (SELECT COUNT(distinct reviewer_id) AS r2_ids
        FROM Reviews
        GROUP BY reviewer_id
        HAVING COUNT(listing_id) > 10) AS R2;


/* Show the listings that are instant bookable, have over 50 reviews, rated above 90 and have over 30 days available in a year. */
SELECT distinct id
FROM Listings
JOIN Reviews ON Listings.id = Reviews.listing_id
WHERE instant_bookable = 't' AND
    review_scores_rating > 90 AND
    availability_365 > 30
HAVING COUNT(Reviews.id) > 50;


/* Show the name of reviewers who reviewed most each year. */
SELECT reviewer_name
FROM Reviews
GROUP BY reviewer_id
HAVING COUNT(id) = (SELECT COUNT(id)
            FROM Reviews
            GROUP BY reviewer_id);

/* Show the listings that have good views (name including view) that are rated highest in each neighborhood. */
SELECT id, name
FROM Listings
WHERE name LIKE %View% OR
```

```
    name LIKE %view% AND
GROUP BY neighborhood_id
ORDER BY review_scores_rating DESC
LIMIT 1;


/* Show the review rating score stats(average, max, min) or listings group by property type, order by average review rating score. *
SELECT property_type, AVG(review_scores_rating), MAX(review_scores_rating), MIN(review_scores_rating)
FROM Listings
GROUP BY property_type,
ORDER BY AVG(review_scores_rating)
```

(6) How to load the database with values

We plan to use sql to bulk load the data from csv files to sql. There will be issues related to datatype conversion (such as format of date, 95% to 95), duplicate tuples, NULL values and data parsing problems ("", escape /, line break in quoted text). An example of how we load data is:

```
 load data local infile 'listings2.csv' into  table listings
fields terminated by ',' optionally enclosed by '"'
(id, name, description...);
```

(7) Result of project

We plan to deploy a web application that has a search feature, which would be useful for users to search for specific listings they look for on Airbnb. The specific implementation is still under discussion. At this stage we have some prelimiary thoughts on such search website. In addition to traditional search on Airbnb listing (location, date, avaliability), we want the search engine to allow user to search for additional information in specific neighborhoods, such as covid cases(total/last 30 days), listing density, demongraphic, living cost, income levels etc. The users could also do a "vague" search. For example, a user can search for "sunset view" which would not be included as part of amentities in the database, but could be mentioned in listing description or reviewers comments. We want the search results beyond the scope of simple queries so that they provide further understanding of the data.

(8) Topics of database design

Some potential topics include data mining, complex data extraction issues from online sources and some fields in natural language interfaces. We will conduct some data mining practice on our dataset to generate useful analysis for users. One example of using natural language related knowledge in data mining is to categorize/sort review comments by their sentiment scores, which is evaluated based on how postive/negative some adjetive words used in the comments. This would be provide an additional way to reflect how positive a user evalute its experience other than old-fashioned numeric rating mechanism.