Property Rental System for London

Aakruti Ambasana, Shyam Vadalia

Abstract—We intend to develop a Property Rental System for London by using the London Airbnb dataset to populate our database. A company named Airbnb, is managing the lodging of tourists and provide specific type of property needed for vacation rentals. Our proposed system is basically a marketplace for rental properties in London, wherein the users will be able to book, cancel a booking, provide review for a property they rented, and post new properties. User can become a host, if he wants to rent the property. Host can view all the customers he served, reviews provided by users and many other things. Admin can manage all users, properties, and bookings. Our overall goal for this project is to understand and implement various aspects of database system design like data modeling, defining a relational schema.

Index Terms—Airbnb, London, Rent, Property, Listing, Host

1 Introduction

Nowadays, people tend to travel a lot. For travelling, in unknown country, you need to get few answers like where to stay, what are the prices per day, what facilities we will get. To get all this answers, we have made Property Rental System by using Airbnb London dataset. The customer can look at the properties based on their preferences like price range, required for number of people, duration to stay, availability. Customer can check the details such as type of property, room type, host details and many more other details. After checking all details regarding Property, customer can book the Property.

The Property Rental System also allows to host the property. If you want to become the host, then you can add all your details regarding property you own. If any guest is searching for property with specific requirements and if your property meets the requirement of customer, then customer will book your property. Additionally, host will also create its own profile and view, how many number of customers he served, how many properties one owns, what are the reviews provided to his properties and many other details.

The third part of our system is Admin side or Server side in Property Rental System. In this, admin can manage all details about properties, customers, hosts, bookings, reviews, and other details.

1.1 Existing Dataset Issues

Existing dataset of Airbnb London has many drawbacks, such as 106 columns of listing relation and half of the attributes are replicate the values. The uncontrolled replication causes errors. So, we try to solve these issues. The host details are every time repeated whenever new listing is added. There are few multivalued attributes like host verification. Like phone number, there are many host whose phone number is verified and many host whose phone number is not. We have normalized the dataset based on atomicity.

Another issue is null values. There is total 85057 property (listing) values in dataset. But some attributes have almost 80000 null values. So, we have solved this kind of issues in our new database design.

2 ER Modelling

ER Modelling helps to understand, the entities and relationship among entities. In Fig.1, we have created separate Host entity from Property because one host can have multiple properties and each time new property is added, host details should not be updated. Many properties have same zipcode, so we have separated address entity because zipcode determines street, city, country, neighbourhood values. We have separated pricing attributes in separate entity because out of 85000 data records, this values are present only in 7000 data records, for the rest it was NULL. Property_OptionalPricing has total specialization to Property relation. Every customer is user, if want to rent a property, then it can become host. So host is specialized entity of user with total participation. Booking is a weak entity because if the user books the property then only, booking will have data, otherwise not. Review has total specialization with booking entity. As user cannot provide review if he has not done booking. The cardinality constraints for weak relationship books are:

- User:Property - 0:*, as user can book more than one property, or may not book
- User:Booking - 0:*, as user can book the property more than one time or does not book the property.
- Property:User - 0:*, as property can be booked many times by different users
- Property:Booking - 0:*, as property can be booked multiple times or not booked at all.
- Booking:Property - 1:1, as single booking contains single property at a time
- Booking:User - 1:1 - as property is booked by single user at a time.

3 Relational Schema

Listings relation of old dataset contained 106 attributes, which contained all host details, address details, optional pricing, details, and big values like description, space, house rules, and many more. We have decomposed it into 4 relations.

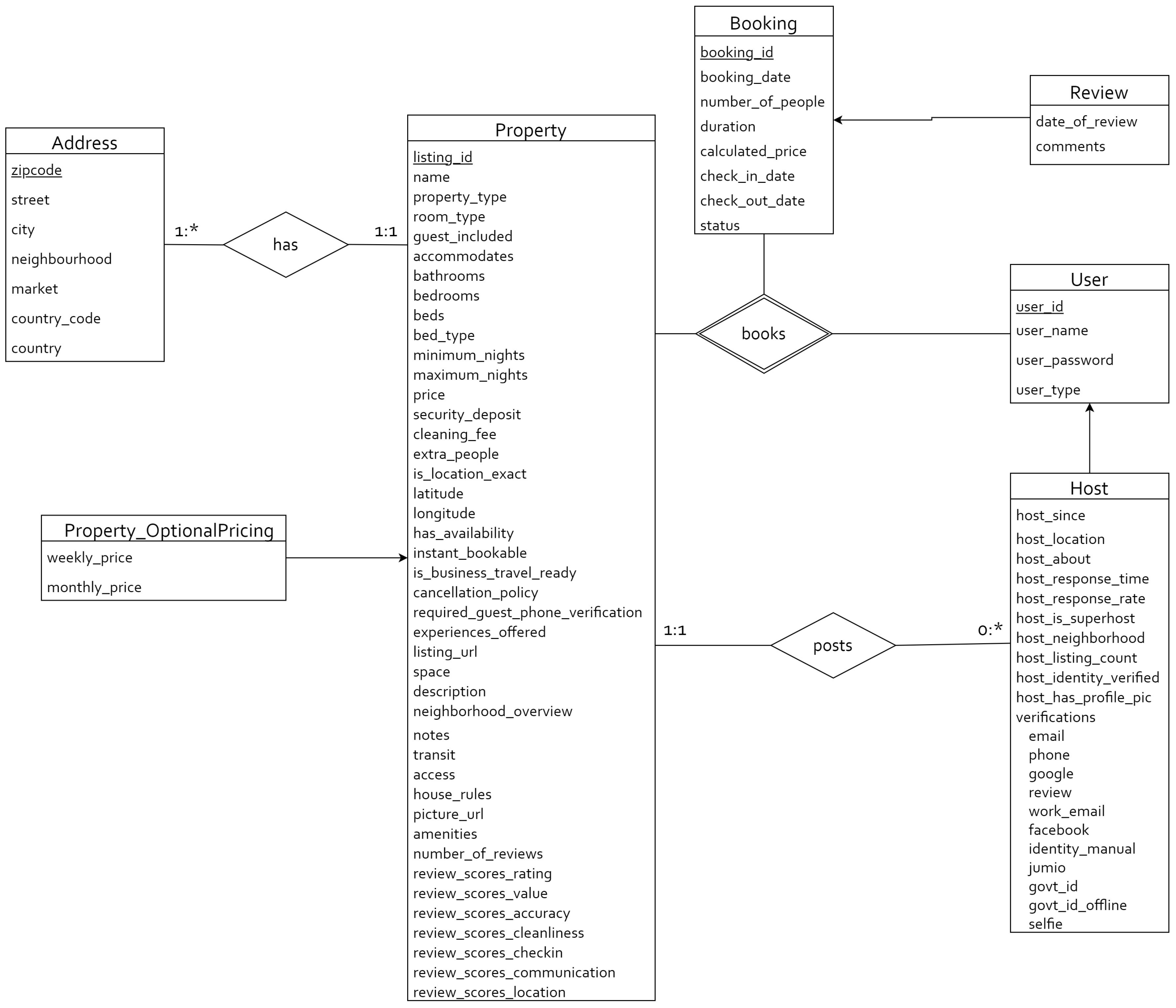

Fig. 1. Entity Relationship Modelling

Address: We have created separate Address relation because zipcode determines street, city, and many other details. And many properties have same zipcode. So, to reduce error, we have created Address relation and made zipcode as unique key, because sometimes zipcode is null, so it cannot be primary key. Data cleaning is performed while loading the statements. In this street data is "street,city,country". As this is London Airbnb dataset, city has only 2 values London and Greater London, country is United Kingdom always. So removed city and country from street value. And placed null where string is empty or contains garbage values like (., 1, Please select, City).

```
CREATE TABLE Address (
    zipcode char(10),
    street varchar(80),
    city char(40),
    neighbourhood_cleansed char(25),
    market char(30),
    country_code char(8),
    country char(15),
    UNIQUE KEY (zipcode));
```

Host: We have created Host as separate relation because one host can have many properties. So, to reduce errors, host details are specified in separate relation, and each property references host_id (host_id values are available in dataset). Data is cleaned in load statement, inserted NULL instead od empty string, or N/A values. Converted t and f values to boolean values. And Verifications is multivalued attribute in format: (phone, email, etc..). So created several other boolean attributes, like phone_verified, email_verified, and many others and removed the multivalued attribute.

```
CREATE TABLE Host (
    host_id int PRIMARY KEY,
    host_since date,
    host_location varchar(80),
    host_about text,
    host_response_time
        ENUM ('within an hour',
        'within a few hours',
        'within a day',
        'a few days or more'),
    host_response_rate decimal(5,2),
    host_is_superhost boolean,
    host_neighbourhood char(40),
    host_listings_count int,
    host_identity_verified boolean,
    host_has_profile_pic boolean,
    email_verified boolean,
    phone_verified boolean,
    google_verified boolean,
    review_verified boolean,
    work_emai_verified boolean,
    facebook_verified boolean,
    identity_manual_verified boolean,
    jumio_verified boolean,
    govt_id_verified boolean,
    govt_id_offline_verified boolean,
    selfie_verified boolean,
    CHECK (host_response_rate<=100.00),
    FOREIGN KEY (host_id)
        REFERENCES User(user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE);
```

Property_Big_Values: We have created separate relations for all big values because, almost all attributes are of datatype varchar(1000). So, the attribute which requires big space are placed in one relation. We have removed, summary attribute because prefix of description is summary, it was same data. It also contained amenities attribute which is multivalued. There are more than 150+ values in amenities and each property contains different and many amenities. So we have not decomposed it. Instead given a proper format to this attribute.

```
CREATE TABLE Property_Big_Values (
    listing_id int PRIMARY KEY,
    listing_url char(45),
    space varchar(1000),
    description varchar(1000),
    neighborhood_overview varchar(1000),
    notes varchar(1000),
    transit varchar(1000),
    access varchar(1000),
    house_rules varchar(1000),
    picture_url varchar(255),
    amenities varchar(1500),
    FOREIGN KEY (listing_id)
    REFERENCES Property(listing_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE);
```

Property_OptionalPricing: We have created separate relation for weekly and monthly prices because from around 85000 data records, we only have 7000+ data records for these two attributes. So, to reduce wastage of space.

```
CREATE TABLE Property_OptionalPricing (
    listing_id int PRIMARY KEY,
    weekly_price decimal(7,2),
    monthly_price decimal(7,2),
    CHECK(
        (weekly_price IS NOT NULL) OR
        (monthly_price IS NOT NULL)),
    FOREIGN KEY (listing_id)
        REFERENCES Property(listing_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE);
```

Property_Review_Statistics: The access to this relation is only to User and Admin. Host can only view this relation cannot change it. Based on this attributes admin can also determine performance of each host and can determine super host also. Not much data cleaning was required for these attributes.

```
CREATE TABLE
Property_Review_Statistics(
    listing_id int PRIMARY KEY,
    number_of_reviews int,
    number_of_reviews_ltm int,
    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,
    reviews_per_month decimal(5,2),
    FOREIGN KEY (listing_id)
    REFERENCES Property(listing_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE);
```

Property: All the remaining attributes are included in this relation which is unique for each property. We have specified enum for room_type because it contained only 4 distinct values. We have included regular expression check on zipcode entered. And converted all prices attributes into proper decimal format. And converted values 't' and 'f' to correct boolean values in few attributes.

```
CREATE TABLE Property (
    listing_id int PRIMARY KEY
        AUTO_INCREMENT,
    name varchar(255),
    host_id int,
    zipcode char(10),
    property_type char(25) NOT NULL,
    room_type
        ENUM('Entire home/apt',
        'Private room','Hotel room',
        'Shared room'),
    guests_included int,
    accommodates int NOT NULL,
    bathrooms int,
    bedrooms int,
    beds int,
    bed_type char(15),
    minimum_nights int NOT NULL,
    maximum_nights int NOT NULL,
    price decimal(7,2) NOT NULL,
    security_deposit decimal(7,2),
    cleaning_fee decimal(6,2),
    extra_people decimal(6,2) NOT NULL,
    is_location_exact boolean,
    latitude decimal(12,7),
    longitude decimal(12,7),
    has_availability boolean,
    instant_bookable boolean,
    is_business_travel_ready boolean,
    cancellation_policy char(70),
    required_guest_phone_verification
        boolean,
    experiences_offered char(10),
    FOREIGN KEY (host_id)
        REFERENCES Host(host_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (zipcode)
        REFERENCES Address(zipcode)
        ON UPDATE CASCADE
        ON DELETE RESTRICT);
```

User: We do not have any user data, except reviews provided by each user. But we want user data to implement functionality booking of property, and cancellation of booking. From old dataset, review relation, we get reviewer_id, which is user_id in this relation and host_id, is also user. Every host is a user first. In user_type attribute, if user is guest, then user_type=0, if user is host also then user_type=1, if it is admin then user_type=2. For authentication purpose, to implement login functionality, this relation was needed. We have initially considered user_id as user_password.

```
CREATE TABLE User (
    user_id int PRIMARY KEY
        AUTO_INCREMENT,
    user_name char(50),
    user_password char(20),
    user_type int);
```

Booking: We also didn't have any booking data of user, so we have considered id which uniquely determines each review from review relation of old dataset as our booking_id, listing_id from review relation, guest_id from reviewer_id of review relation, and date from review relation as check out date. And kept status as 'Checked out', because after guest is checked out then only, one can provide review. For the new booking all values will be filled.

```
CREATE TABLE Booking (
    booking_id int PRIMARY KEY
        AUTO_INCREMENT,
    listing_id int,
    guest_id int,
    booking_date date,
    number_of_people int,
    duration int,
    calculated_price decimal(7,2),
    check_in_date date,
    check_out_date date,
    status ENUM(
        'Waiting for approval',
        'Booked','Checked out',
        'Canceled'),
    FOREIGN KEY (listing_id)
    REFERENCES Property(listing_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (guest_id)
    REFERENCES User(user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE);
```

Review: It contains booking_id as primary key because it is unique, and if you have booked the property then only you can provide review to it. user_id and listing_id can be finded out by booking id, so we will know which user is providing review to which listing.

```
CREATE TABLE Review (
    booking_id int PRIMARY KEY,
    date_of_review date,
    comments varchar(5000),
    FOREIGN KEY (booking_id)
    REFERENCES Booking(booking_id)
    ON UPDATE CASCADE
    ON DELETE CASCADE);
```

4 Client Application

We have created our command line user interface using python.

4.1 Database Connection

```
mydb = mysql.connector.connect(
    host="marmoset03.shoshin.uwaterloo.ca",
    user="\textit{user_name}",
    password="\textit{password}",
    database="db656_aaambasa"
)
mycursor = mydb.cursor(buffered=True)
```

4.2 Overview of Functionalities

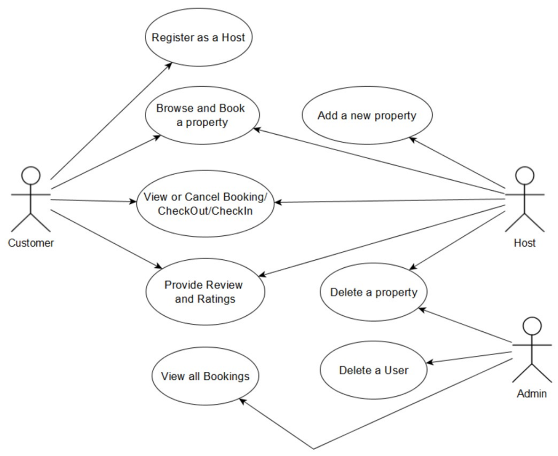

Fig. 2. Use Case Diagram

4.3 User Application

4.3.1 Functionality Implemented

1. Browse rental properties based on preferences

```
select listing_id, name, property_type,
    neighbourhood_cleansed, price,
    has_availability, review_scores_rating
    from Property
        inner join Host using(host_id)
        inner join Address using(zipcode)
        inner join Property_Review_Statistics
            using(listing_id)
    where price<%s and price>%s and
        has_availability=%s and
        host_is_superhost=%s and
        maximum_nights>%s and
        minimum_nights<%s and
        review_scores_rating>=%s
```

2. Book the Property: Before booking the property, we check whether the property is available or not. If not then, user cannot book the property

```
insert into Booking
    (listing_id, guest_id,
    booking_date, number_of_people,
    duration, calculated_price,
    check_in_date, status)
values
    (%s,%s,%s,%s,%s,%s,%s,'Booked')

update Property set has_availability=0
    where listing_id=%s
```

After booking, has_availability attribute is set to zero.

3. View booking history

```
SELECT booking_id, listing_id,
    Property.name,booking_date,
    number_of_people,duration,
    calculated_price, check_in_date,
    check_out_date,status
    from Booking
    inner join Property using (listing_id)
    where guest_id=%s
```

4. View reviews

```
select user_name, date_of_review, comments
    from Review
    inner join Booking
        using(booking_id)
    inner join User
        on(Booking.guest_id=User.user_id)
    where listing_id=%s
    order by date_of_review desc;
```

5. Provide reviews

```
select booking_id, listing_id,
    Property.name, booking_date,
    duration, check_in_date,
    check_out_date, status
    from Booking inner join Property
        using(listing_id)
    where guest_id=%s and
        status='Checked out' and
        booking_id not in
            (select booking_id from Review)
```

User can only provide the review, if this query returns the booking id.

```
insert into Review values (%s,%s,%s)
update Property_Review_Statistics
    set number_of_reviews=%s,
    review_scores_rating=%s,
    review_scores_cleanliness=%s,
    review_scores_checkin=%s,
    review_scores_communication=%s,
    review_scores_location=%s,
    review_scores_value=%s
        where listing_id=%s
```

6. Cancellation of Booking: In this before, cancellation we check for checking date, if the user is not checked in, then user cannot check out.

```
update Booking set status='Canceled'
    where booking_id=%s
update Property set has_availability=1
    where listing_id=%s
```

After cancellation of booking, it sets the has_availability to 1, as it will be available now.

7. Become a host

```
INSERT INTO Host
    (host_id, host_since, host_location,
    host_about, host_response_time,
    host_response_rate, host_is_superhost,
    host_neighbourhood, host_listings_count,
    host_identity_verified,
    host_has_profile_pic)
VALUES (%s, %s, %s, %s, %s, %s, %s,
    %s, %s, %s, %s)

update User set user_type=1
    where user_id=%s
```

User type will be updated after user becomes host.

4.4 Host Application

4.4.1 Functionality Implemented

1. View Rental Properties Details

```
select
    listing_id, name, zipcode, property_type,
    room_type, guests_included, accommodates,
    bathrooms, bedrooms, beds, bed_type,
    minimum_nights, maximum_nights,
    experiences_offered, price,
    cancellation_policy, security_deposit,
    cleaning_fee, extra_people,
    is_location_exact, latitude, longitude,
    has_availability, instant_bookable,
    is_business_travel_ready,
    required_guest_phone_verification
from Property
where host_id=%s
```

For optional Pricing information

```
with s1 as (select listing_id from Property
    where host_id=%s)
select listing_id, weekly_price,
    monthly_price from s1
    inner join Property_OptionalPricing
        using (listing_id)
    where listing_id=%s
```

To view review statistics

```
with s1 as (select listing_id from Property
    where host_id=%s)
select listing_id, number_of_reviews,
    number_of_reviews_ltm, 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, reviews_per_month
from s1 inner join Property_Review_Statistics
    using (listing_id)
    where listing_id=%s
```

For detailed description of Property

```
with s1 as (select listing_id from Property
    where host_id=%s)
select listing_id, listing_url, space,
    description, neighborhood_overview,
    notes, transit, access, house_rules,
    picture_url, amenities from s1
        inner join Property_Big_Values
        using (listing_id)
        where listing_id=%s
```

For viewing reviews of property

```
with s1 as (select listing_id from Property
    WHERE host_id=%s),
s2 as (select booking_id from s1
    inner join Booking using(listing_id)
    where listing_id=%s)
select date_of_review,comments from Review
    where booking_id in
        (select booking_id from s2)
```

2. Number of Customers Served History

```
WITH t1 AS
(SELECT listing_id
    FROM Property
    WHERE host_id = %s)
SELECT
    booking_id, listing_id,
    user_name, booking_date,
    number_of_people, duration,
    calculated_price,
    check_in_date,
    check_out_date, status
FROM t1
INNER JOIN Booking
    USING(listing_id)
INNER JOIN User
ON (Booking.guest_id=User.user_id)
```

3. Own Travel Rental History

```
SELECT
    booking_id, Property.name,
    booking_date, number_of_people,
    duration, calculated_price,
    check_in_date, check_out_date,
    status
FROM Booking INNER JOIN
Property USING (listing_id)
WHERE guest_id = %s
```

4. New Rental Property

```
insert into Address
    (zipcode, street, city,
    neighbourhood_cleansed,
    market, country_code, country)
values (%s, %s, %s, %s, %s, %s, %s);

insert into Property
    (host_id, name, zipcode, property_type,
    room_type, guests_included,
    accommodates, bathrooms, bedrooms,
    beds, bed_type, minimum_nights,
    maximum_nights, price, security_deposit,
    cleaning_fee, extra_people,
    is_location_exact, latitude, longitude,
    has_availability, instant_bookable,
    is_business_travel_ready,
    cancellation_policy,
    required_guest_phone_verification,
    experiences_offered)
values (%s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s)

insert into Property_Big_Values
(listing_id, space, description,
neighborhood_overview, notes, transit,
access, house_rules, amenities)
values(%s, %s, %s, %s, %s, %s, %s, %s, %s)

insert into Property_OptionalPricing
    (listing_id, weekly_price, monthly_price)
    values (%s, %s, %s)
```

After inserting new Property, host_listings_count from Host will be incremented to 1

```
UPDATE Host SET host_listings_count=%s
    WHERE host_id=%s
```

5. View Profile

```
SELECT
    host_since, host_location,
    host_about, host_response_time,
    host_response_rate,
    host_is_superhost,
    host_neighbourhood,
    host_listings_count,
    host_identity_verified
FROM Host
WHERE host_id = %s
```

6. Edit Profile

```
Edit Username or Host name:
UPDATE User SET user_name = %s
    WHERE user_id = %s
Edit Host Location:
UPDATE Host SET host_location = %s
    WHERE host_id = %s
Edit Host About:
UPDATE Host SET host_about = %s
    WHERE host_id = %s
Edit Host Response Time:
UPDATE Host
    SET host_response_time = %s
    WHERE host_id = %s
Edit Host Neighbourhood:
UPDATE Host
    SET host_neighbourhood = %s
    WHERE host_id = %s
```

7. Login

```
SELECT
    user_name, user_id, user_type
    FROM User
    WHERE user_name=%s
        AND user_password=%s
```

4.5 Admin Application

1. Browse and Manage rental properties: It is included in above functionalities

2. View Booking history

```
select booking_id, listing_id,
    user_name, booking_date,
    number_of_people, duration,
    calculated_price, check_in_date,
    check_out_date, status
    from Booking inner join User
        on(Booking.guest_id=User.user_id)
```

3. Manage Users

```
select user_id, user_name, user_type
    from User where user_name=%s"
delete from User where user_id=%s
```

5 TEST CASES

5.1 Server Side Test cases

Case 1:
When host adds new property for rental then, property details will be inserted to Property relations. But host_listings_count in Host relation should be incremented by 1, as that host added new Property. The count of number of listings per host from Property relation should be equal to host_listings_count from Host relation all the time. It below query should always return Empty set.

```
SELECT
    host_id, count(host_id) as cnt,
    host_listings_count
    FROM Property
    INNER JOIN Host
        USING (host_id)
    GROUP BY host_id
    HAVING
        cnt <> host_listings_count;
```

Case 2:
The count of distinct number of host from Property should be greater than or equal to row count in host. As if there is property, then there should be compulsory host. Its possible that host does not have any property yet.

```
WITH T1 AS
(SELECT
    count(distinct host_id)
    AS cnt_host_from_property
    FROM Property),
T2 AS
(SELECT
    count(host_id)
    AS cnt_hostid_from_host
    FROM Host)
SELECT * from T1,T2;
```

Case 3:
User can only provide review on properties that they have booked and checked out and once per booking
(On server side it only checks if the booking id is present in the Booking table before inserting the values into Review table. Remaining constraints are implemented on client side)
Input: Inserting a booking id which is not present in Booking table
Expected Outcome: Error. Foreign key constraint.
Input given:

```
insert into Review (booking_id,
    date_of_review, comments)
    values (
        (select max(booking_id)+1
            from Booking),
        "2021-12-23", "Testing review.");
```

Outcome received: ERROR 1452 (23000): Cannot add or update a child row. A foreign key constraint fails

Case 4:
CheckIn date should be less than CheckOut date in Booking table
Input: Inserting a tuple into Booking table where check in date is greater than check out date
Expected Outcome: Error. Check constraint is violated.
Input given:

```
insert into Booking
    (listing_id, guest_id, booking_date,
    number_of_people, duration,
calculated_price, check_in_date,
check_out_date, status)
values (11551, 1, "2021-12-01", 3, 5, 504,
"2021-12-10", "2021-12-08", 'Checked out');
```

Outcome received: ERROR 3819 (HY000): Check constraint 'Date' is violated

5.2 Client Test cases

Case 1:
User cannot provide review before checking out of a booked property.
Input: User tries to enter a booking id which already been reviewed or cannot be reviewed yet.
Expected Behaviour of application: User will get an Error message saying that the booking id does not match with the list of valid booking ids.

Case 2:
User cannot book a property that is not available (i.e availability=0).
Input: User tries to book a property which has availabilty=0.

Expected Behaviour of application: User will get an Error message: "Error: You cannot book this property. It is not available".

Case 3:
Customer cannot delete a property.
Input: A user with user type of customer tries to delete a property by entering "-" (minus symbol) when prompted to enter "1" to book a selected property.
Expected Behaviour of application: User will get an Error message: "Error: Invalid Input. Please enter an appropriate input.".

Case 4:
User cannot CheckOut before the CheckIn date.
Input: A user tries to checkout from a booked property, in the View Booking History section of application, before the CheckIn date.
Expected Behaviour of application: User will get an Error message: "Sorry. You cannot checkout before checking in."

Case 5:
User cannot cancel a booking after CheckIn date.
Input: A user tries to cancel a booking from the list of booked properties, in the View Booking History section of application, after the CheckIn date.
Expected Behaviour of application: User will get an Error message: "Sorry. You cannot cancel this booking. Today's date is greater than or equal to CheckIn date."

6 Conclusion

From this project, we have understood and implemented various aspects of database design like data modelling, entity-relationship modelling, creating relational schema based on ER modelling, and many other things. In server side, we have created relational schema with all constraints required, and have loaded and cleaned the data from CSV files of London Airbnb dataset. In client application, we have implemented three sites, which is User site, Host site, and Admin site. Our user interface is command line using python. User can search and book the property, cancel the booking, view own booking history. Host can add new properties to rent, view number of customers served, view and edit its profile and few other things. The admin can manage all users, properties, and booking details. We have learnt to produce MySQL queries based on different functionality requirement.