AIRBNB DATABASE MANAGEMENT SYSTEM

Amritha Subburayan
Engineering and Applied Sciences
Buffalo, New York
asubbura@buffalo.edu

Kamalnath Sathyamurthy
Engineering and Applied Sciences
Buffalo, New York
Kamalnat@buffalo.edu

Sanjay Aravind Loganathan Ravichandran
Engineering and Applied Sciences
Buffalo, New York
Sanjayar@buffalo.edu

***Abstract* — As the urge of people towards exploring the world keeps on increasing, not only the famous cities but also exotic suburban places are also getting flooded with tourists. Over the past few years (pre-covid), tourism is at boom enabling people to visit and explore the places worldwide. Airbnb has turned into an extremely famous decision among explorers all over the planet for the sort of novel encounters that they give and furthermore to introducing an option in contrast to expensive lodgings. Airbnb does links landlords and tourist visitors by providing an accommodation. The problem that we are going to address is eliminating the inconsistency in Airbnb dataset by creating and normalizing the database system provides efficiency in storing and accessing the huge amount of data and provides hassle free user interface to the users. Normalization using BCNF and query analysis are performed for the dataset.**

Keywords— efficiency, Normalization, user interface, BCNF

I. INTRODUCTION

Airbnb is a web-based commercial center that organizes and offers rooms or houses in which individuals can stay temporarily with family. The organization is only a merchant that acts as an interface between tenants and property owners, then, at that point, gets commissions as a money for each reserving done by a user from landlord. Common people who want to enjoy their vacation by staying in an affordable house to feel like a home-atmosphere will be able to use our database to check the availability of the houses [4]. Our database provides all listings from cheapest till expensive rental places based on user choices with distinctive pricing strategy. The need of database is because of its efficiency. The database can store millions of data efficiently without any disruption whereas excel can be unmanageable at some point of time if excel size is huge. There is no limitation is database whereas excel sheet has limitation of data. Since customer data are crucial, storing in a database will be more secured than saving it has an excel file. Hence, for this project we are storing our customer data in a database in separate schema as per the requirement [5].

II. DATA RESOURCE

The dataset has been extracted from insideairbnb.com website which have airbnb datasets (listings, calendar and pricings) of almost all the major cities in the world. The data behind the Inside Airbnb site is sourced from publicly available information from the Airbnb site. The data has been analyzed, cleansed and aggregated to facilitate public discussion [1]. Among them we used listings of five major cities in United States which are as follows: Chicago, Dallas, New York, San Francisco and New Jersey. These datasets include all the details related to house listings, house availability (calendar) and reviews of that listing. Initially we created 3 tables from dataset obtained. After applying BCNF, we came up with 11 tables and further normalizing the dataset, we ended with 13 relations.

III. ENTITY RELATION SCHEMA

ER diagram illustrates how the entities relate to each other in the database. Here, the original table is normalized to 13 different tables to ensure that there is direct relationship between data and primary key. ER diagrams are related to data structure diagrams (DSDs), which focus on the relationships of elements within entities instead of relationships between entities themselves. ER diagrams also are often used in conjunction with data flow diagrams (DFDs), which map out the flow of information for processes or systems [2]. For our dataset, the listings table, calendar table and reviews table are normalized to following tables – booking, host, listing_address, listing_amenities, listing_details, listing_pricings, listing_reviews, listings_state, listings, location_xwlk, status_xwlk, users.

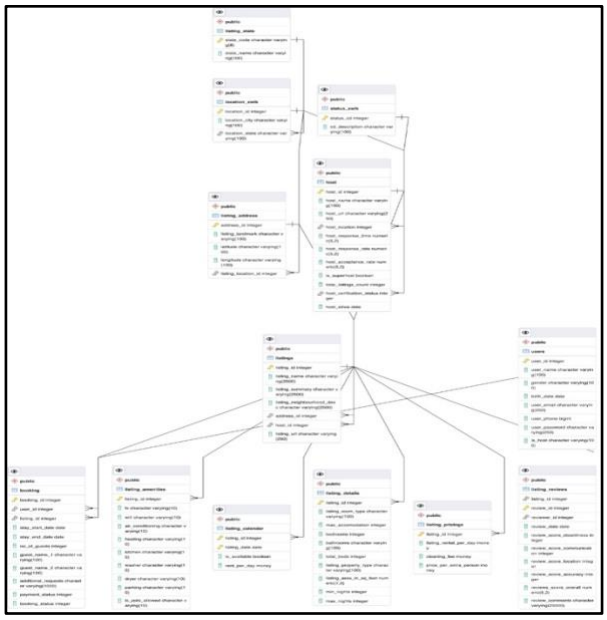


Fig a. Relation and its attributes

Database Summary	
Database	1
Schema	1
Roles	3
Relations/Tables	13
Views	1
Indices	39
Sequences	8
Functions	2
Trigger Functions	2
Triggers	2
Stored Procedures	1

Table b. Summary of database creation

Relations	Description
status_xwlk	We will be using this table as crosswalk table in order to refer the statuses of bookings, host verification. For example, status 1 in this table is with description 'Verified'.
listing_state	This table is crosswalk table to refer all the states in the US with its state code. For Example, NY is used for New York.
location_xwlk	This table is crosswalk table to denote the cities of listings/host. We have 3 columns in this table which denotes city and state code. For example, if host location is with location id 1, it means that host's city is chicago and state is Illinois.
host	This table denotes the host details who organizes and offers rooms or houses. This table contains 11 columns where host_id acts as primary key.
listing_address	This table denotes the listings address details. This table contains 5 columns where address_id is primary key and listings_location_id acts as foreign key.
listings	This table holds all the details related to the listings like listing name, its description, neighborhood. Host_id and address_id acts as a foreign key.
listing_amenities	This table holds the interior details in the house like if TV, WIFI, AC, Heating, Dryer, etc. If these facilities does available or not.
listing_details	This table holds columns that describes the necessary facilities provided by the listings like number of bathrooms, bedrooms, number of people it can accommodate, property type, etc.
listing_pricings	This table denotes the listings pricing details like rent per day, cleaning charge, and fee for extra per person.
listing_calendar	This table denotes the availablity details of each listings like listing posted date, rent per day and whether its available or not.
users	This table defines the users details like user name, gender and date of birth for authentication.
listing_reviews	This table holds the reviews given by user for the listings they stayed. This table has 10 columns where score for each services has been calculated.
booking	This table holds all the details of the users and listings registered by the users. This is to check and analyse the booking details and to close certain listing if its booked by one user to avoid conflict.

Table c. Relations created in the database and its description

Views	Description
host_period_view	We have used this view in order to fetch the time frame of any user being as host. Since we used current_date in this view, it will fetch the total period the user served as host as of current date.

Table d. Views created

Functions	Description
random_between	This function is used to generate random 7 digit big integer between any two input big integers. We have used this function to generate random phone numbers for the users. First 3 digits corresponds to area code for which we used rand operation to select between 3 area codes.
random_string_pavel	This function is used to generate random text and numerics of given length. We used this function to generate password column in the users table.

Table e. Functions and its purpose in database

Trigger function	Description
function_number_of_listings	This function is to update total_listings_count on public.hosts table whenever data is inserted or deleted in listings table.
trg_update_is_host	This function is to update is_host in user table in order to update the column to 'Y' whenever the user is added in hosts table.

Table f. Trigger functions and its usage in database

Stored Procedure	Description
booking	This stored procedure is to insert records into booking table while passing the values that needs to be loaded into table.

Table g. Stored Procedure and its usage in database

IV. Normalizing the dataset using Boyce-Codd Normal Form (BCNF)

BCNF is used to normalize multidimensional databases until there is no more normalization can be carried out in the database. It helps to reduce redundancy and maintains integrity in the database. It ensures that there is no duplicate values in the database. To obtain this, first the relation should be in 3NF and at least one of the reference tables should contain a primary key. For out database, we have normalized our database to ensure that the relation contains only data that are directly related to primary key. After normalizing the original data, there were total 13 tables obtained out of 3 tables which is now in BCNF form. From the original listings table, we had more two primary keys that are referred by another table. Hence, we separated the table that has primary key and the attribute that are directly related to the primary keys. For example, Host_Id primary key has been separated and added to host table which holds details related to hosts. Likewise, listings details have been added with listing_id primary key that can be referred by other tables. We normalized the data by ensuring that attributes are related only to the primary key.

V. Query Implementation

After normalization using BCNF, analyzation has been performed to understand the statistics of the airbnb market. Few assumptions were also made to understand how the database works.

Few analyzations on dataset have been performed using SQL. Below are few findings that was performed on the data.

Query1: How many hosts has been added yearly from the start?

Description of the logic: This query is used to check how many airbnb hosts has been since the start and to check which year played major role in revenue for airbnb

SQL Statement:

```
WITH HOST1 AS ( SELECT DISTINCT host_id, extract(YEAR FROM host_since::date) as YEAR FROM host) SELECT COUNT(host_id) AS TOTAL, YEAR FROM HOST1 GROUP BY YEAR ORDER BY TOTAL DESC
```

Output:

	total bigint	year numeric
1	553	2015
2	513	2016
3	369	2014
4	352	2017
5	344	2018
6	294	2019
7	275	2013
8	180	2021
9	159	2012
10	153	2020
11	104	2011
12	25	2010
13	15	2009
14	4	2008

Query 2: Which place does receive highest rating? Less expensive or more expensive ones?

Description of the logic: This query is used to check whether the highest reviews comes from expensive listings or less expensive listings. This is to provide clear view for people who wants to check in both the places.

SQL Statement:

```
Select a.reviews_score_overall, sum(a.Rental_below_hundred) as total_Rental_below_hundred, sum(a.Rental_between_100_1000) as total_Rental_between_100_1000, sum(a.Rental_above_1000) as total_Rental_above_1000 from (select listing_reviews.reviews_score_overall, CASE WHEN listing_pricings.listing_rental_per_day < '100' then 1 else 0 end as Rental_below_hundred, CASE WHEN listing_pricings.listing_rental_per_day > '100' and listing_pricings.listing_rental_per_day <= '1000' then 1 else 0 end as Rental_between_100_1000, CASE WHEN listing_pricings.listing_rental_per_day > '1000' then 1 else 0 end as Rental_above_1000 from listing_reviews join
```

listing_pricings on listing_reviews.listing_id = listing_pricings.listing_id)a group by a.reviews_score_overall order by reviews_score_overall desc

Output:

Notifications | Data Output | Explain | Messages

	reviews_score_overall numeric (5,2)	total_rental_below_hundred bigint	total_rental_between_100_1000 bigint	total_rental_above_1000 bigint
1	5.00	203	171	0
2	4.75	766	743	5
3	4.50	1999	1926	8
4	4.25	3985	3807	19
5	4.00	6852	6558	35
6	3.75	10348	9653	54
7	3.50	13262	12712	57
8	3.25	15739	15139	69
9	3.00	16756	15913	78
10	2.75	15885	15243	68
11	2.50	13211	12768	67
12	2.25	10216	9731	40
13	2.00	7082	6492	36
14	1.75	3840	3814	22
15	1.50	2060	1868	12
16	1.25	754	754	5
17	1.00	193	184	0

Query 3: Which type of accommodations are in huge number per city?

Description of the logic: This query is to provide clear understanding of the number of type of listings are present in each city.

SQL Statement:

select abc.* from (select ab.*, rank() over (partition by ab.location_city order by ab.total_count desc) from (select c.location_city, count(a.listing_id) as total_count, a.listing_property_type from listing_details a inner join listings b on a.listing_id = b.listing_id inner join listing_address d on b.address_id = d.address_id inner join location_xwlk c on d.listing_location_id = c.location_id group by c.location_city,a.listing_property_type)ab)abc where rank <= 3

Output:

Notifications | Data Output | Explain | Messages

	location_city character varying (100)	total_count bigint	listing_property_type character varying (100)	rank bigint
1	CHICAGO	1049	Entire rental unit	1
2	CHICAGO	193	Private room in rental unit	2
3	CHICAGO	187	Entire condominium (condo)	3
4	DALLAS	827	Entire rental unit	1
5	DALLAS	194	Entire condominium (condo)	2
6	DALLAS	153	Private room in rental unit	3
7	JERSEY CITY	617	Entire rental unit	1
8	JERSEY CITY	179	Entire condominium (condo)	2
9	JERSEY CITY	149	Private room in rental unit	3
10	NEW YORK CITY	592	Entire rental unit	1
11	NEW YORK CITY	160	Entire condominium (condo)	2
12	NEW YORK CITY	122	Private room in rental unit	3

Query 4: How many revenues does each type of properties produces?

Description of the logic: This query lists the total revenue produces by each type of properties seperately. This may help new hosts to decide which type of property they can lease to see some profit.

SQL Statement:

select b.listing_property_type, sum(a.listing_rental_per_day) as total_revenue from listing_pricings a inner join listing_details b on a.listing_id = b.listing_id group by b.listing_property_type order by total_revenue desc

Output:

Notifications | Data Output | Explain | Messages

	listing_property_type character varying (100)	total_revenue money
1	Entire rental unit	$540,741.00
2	Entire residential home	$149,781.00
3	Entire condominium (condo)	$141,020.00
4	Private room in rental unit	$37,450.00
5	Room in boutique hotel	$30,218.00
6	Private room in residential ho...	$28,363.00
7	Entire townhouse	$27,022.00
8	Entire loft	$26,595.00
9	Entire serviced apartment	$26,292.00
10	Private room in condominium...	$20,515.00
11	Entire guest suite	$19,983.00
12	Room in hotel	$8,118.00
13	Private room in townhouse	$4,682.00
14	Entire guesthouse	$4,654.00
15	Room in serviced apartment	$3,677.00

Query 5: Lists top listings based on the review scores.

Description of the logic: This query lists all the listings review from top till bottom. This can help users to analyse before they move in.

SQL Statement:

select distinct listings.listing_name, max(listing_reviews.reviews_score_overall) as max_review from listings join listing_reviews on listings.listing_id = listing_reviews.listing_id group by listing_name order by max_review desc

Output:

	listing_name character varying (2500)	max_review numeric
1	#1 neighborhood in Chicago w/ private balcony	5.0
2	*Comfortable & Spacious Home Close to Downtown*	5.0
3	02a. Full bed in Coed room - 3 mi to NWU	5.0
4	03a. Queen sized in coed room	5.0
5	1 BD Condo by downtown,park,lake	5.0
6	1 Bedroom Apartment in Historic Old Town House	5.0
7	1 mile to USMLE & minutes to O'Hare in Safest Area	5.0
8	15 Minutes to Everything. Skyline View. Quiet.	5.0
9	2 Bed Apartment with Balcony	5.0
10	2 Bedroom city apartment in the heart of Pilsen	5.0
11	2 Bedrooms are perfect for groups up to 3 guests	5.0
12	2 Bedrooms- Comfy Chicago Basement disinfected!	5.0
13	2. MANSION BEDROOM 2- with PRIVATE BATH ROOM	5.0
14	2/1 One Block to Fullerton L Red Line Deck & Yard	5.0
15	2BR Bursting w/ Character in Hip Logan Square!	5.0

QUERY OPTIMIZATION:

Below are few queries that are optimized by using indexing to perform efficiently. Below table lists the query statement, its description and execution time before and after indexing.

QUERY OPTIMIZATION USING INDEXING

Query Statement	Description	Execution time before Optimization	Execution time after
SELECT year, current_year_host, CASE WHEN prev_year_host is null then 0 else prev_year_host end, CASE WHEN prev_year_host is null then 100 else round(((current_year_host - prev_year_host)/prev_year_host:: FLOAT* 100)) end as rate_of_growth FROM (SELECT year, current_year_host, LAG(current_year_host, 1) OVER (ORDER BY year) as prev_year_host FROM (SELECT extract(year FROM host_since::date) as year, count(host_id) as current_year_host FROM host WHERE host_since IS NOT NULL GROUP BY extract(year FROM host_since::date) ORDER BY year asc) a) b;	This query is used to evaluate the growth metrics of airbnb by comparing the current year with previous year. This checks how many new hosts has been added from each year and check if it exceeds previous year addition. If current year hosts is greater than previous year then airbnb growth is positive and increasing in growth.	59.741ms	17.918ms
select a.reviews_score_overall, sum(a.Rental_below_hundred) as total_Rental_below_hundred, sum(a.Rental_between_100_1000) as total_Rental_between_100_1000, sum(a.Rental_above_1000) as total_Rental_above_1000 from (select listing_reviews.reviews_score_overall, CASE WHEN listing_pricings.listing_rental_per_day <'100' then 1 else 0 end as Rental_below_hundred, CASE WHEN listing_pricings.listing_rental_per_day > '100' and listing_pricings.listing_rental_per_day <= '1000' then 1 else 0 end as Rental_between_100_1000,CASE WHEN listing_pricings.listing_rental_per_day > '1000' then 1 else 0 end as Rental_above_1000 from listing_reviews join listing_pricings on listing_reviews.listing_id = listing_pricings.listing_id)a group by a.reviews_score_overall order by reviews_score_overall desc	This query is used to find the relationship between rental pricing and review. Distribution of review around the pricing between below 100, between 100 and 1000 and above 1000.	70.71ms	23.918ms
select abcde.* from (select abcd.*, rank() over (partition by abcd.complete_year order by abcd.total_turnover desc) from (select c.listing_name, abc.complete_year, abc.total_turnover from (select ab.listing_id, sum(ab.total_fee) as total_turnover, ab.complete_year from (SELECT a.listing_id, b.listing_rental_per_day + b.cleaning_fee as total_fee, extract(YEAR FROM a.listing_date::date) as complete_year FROM listing_calendar a inner join listing_pricings b on a.listing_id = b.listing_id where a.is_available = true)ab group by ab.listing_id, ab.complete_year) abc inner join listings c on c.listing_id = abc.listing_id)abcd)abcde where rank<=5 order by abcde.complete_year desc	This query is to find the listing with highest turnover for the complete year for all the cities in database,	45.89ms	27.92ms

Before Optimization

	QUERY PLAN text
1	Subquery Scan on b (cost=248.95..359.47 rows=2105 width=56) (actual time=4.135..4.155 rows=14 loops=1)
2	[...] -> WindowAgg (cost=248.95..306.84 rows=2105 width=48) (actual time=4.132..4.146 rows=14 loops=1)
3	[...] -> Sort (cost=248.95..254.22 rows=2105 width=40) (actual time=3.961..3.964 rows=14 loops=1)
4	[...] Sort Key: (EXTRACT(year FROM host.host_since))
5	[...] Sort Method: quicksort Memory: 25kB
6	[...] -> HashAggregate (cost=106.45..132.76 rows=2105 width=40) (actual time=3.867..3.894 rows=14 loops=1)
7	[...] Group Key: EXTRACT(year FROM host.host_since)
8	[...] Batches: 1 Memory Usage: 121kB
9	[...] -> Seq Scan on host (cost=0.00..89.75 rows=3340 width=36) (actual time=0.063..2.514 rows=3340 loops=1)
10	[...] Filter: (host_since IS NOT NULL)
11	Planning Time: 0.254 ms
12	Execution Time: 59.741 ms

After Optimization

	QUERY PLAN text
1	Subquery Scan on b (cost=248.95..359.47 rows=2105 width=56) (actual time=2.857..2.874 rows=14 loops=1)
2	[...] -> WindowAgg (cost=248.95..306.84 rows=2105 width=48) (actual time=2.853..2.866 rows=14 loops=1)
3	[...] -> Sort (cost=248.95..254.22 rows=2105 width=40) (actual time=2.815..2.817 rows=14 loops=1)
4	[...] Sort Key: (EXTRACT(year FROM host.host_since))
5	[...] Sort Method: quicksort Memory: 25kB
6	[...] -> HashAggregate (cost=106.45..132.76 rows=2105 width=40) (actual time=2.768..2.786 rows=14 loops=1)
7	[...] Group Key: EXTRACT(year FROM host.host_since)
8	[...] Batches: 1 Memory Usage: 121kB
9	[...] -> Seq Scan on host (cost=0.00..89.75 rows=3340 width=36) (actual time=0.074..1.816 rows=3340 loops=1)
10	[...] Filter: (host_since IS NOT NULL)
11	Planning Time: 0.421 ms
12	Execution Time: 17.918 ms

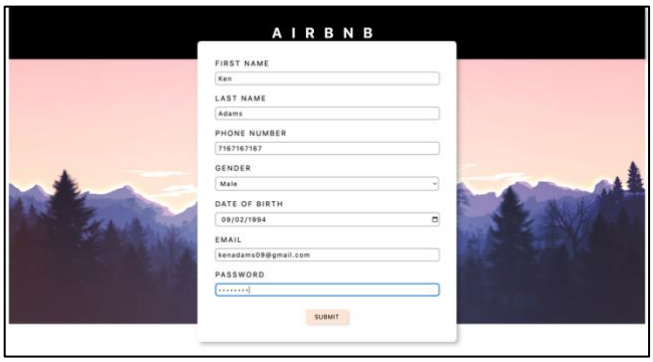
VI. USER INTERFACE

A User Interface has been developed for the users to interact with the airbnb management system. It is developed using Angular Framework, Node.js, HTML5, CSS3, and express server to connect the UI to the database. The UI comprises of the following modules:

- Register
- Sign-in
- User homepage
- Book property
- View my bookings
- Make me host

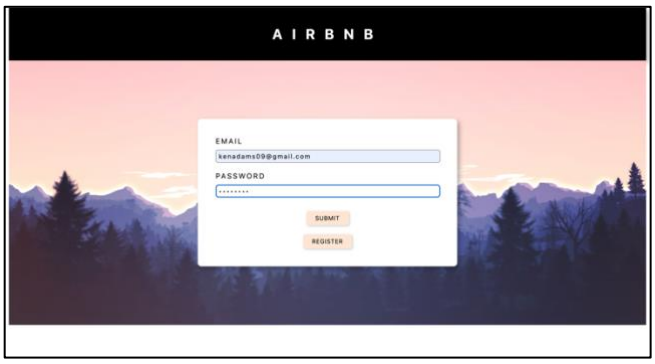
Register:

When a user opens the website for the first time, he/she needs to register in order to use the application. When user fills the register form, the details are added to the Users table in the database.



Sign-in

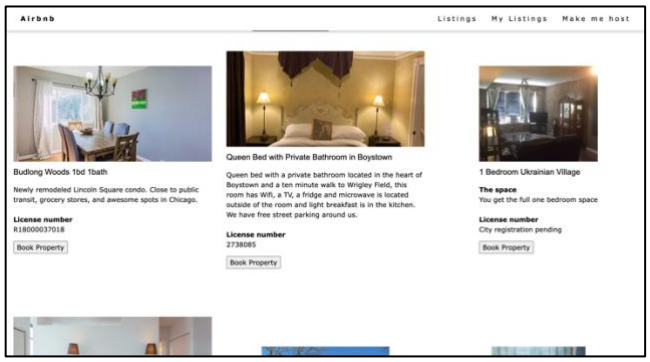
A registered user can log in to the application by entering the credentials. The entered credentials are then checked in the users table and if present, the user will be able to login else error message is displayed.



User homepage

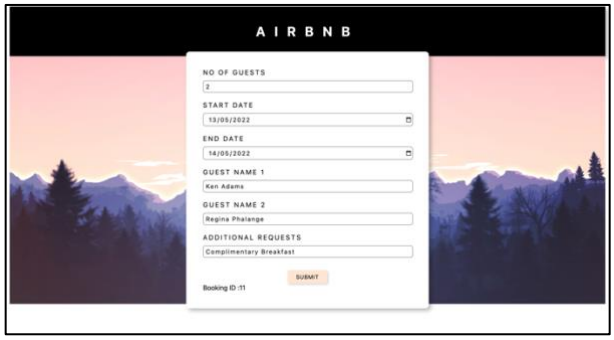
When a user logs in, the user will be able to view all the listings that are available and can book if they wish to. All the

listings from the listings table are fetched and displayed in pagination.



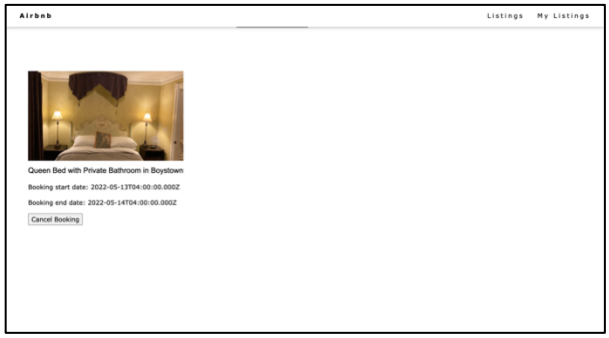
Book property

The user can select a particular listing and book it for a particular period. Once the user clicks the "book property" button present under the listing details, the user will be navigated to the booking form where the user will be asked to fill a form in order to book that property. If the listing is available during that period, the listing will be booked successfully, and a booking ID will be returned. Otherwise, an error message is displayed.



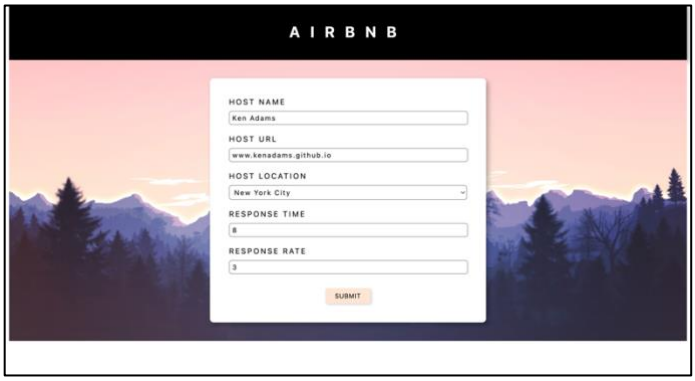
View my listings

A user can view all the bookings made my themselves in this section. All the listings in the bookings table for that particular user is fetched and shown in the UI.



Make me host

If a particular is not a host, the user can nominate themselves to be one. To do this, user can click on the "Make me host" button on the user home page, and they will be asked to fill a form in order to become a host. Once the form is filled, a new record is added in the hosts table, and a trigger is used to make the "is_host" attribute in the Users table as 'Y'.



VII. CONCLUSION:

From this project, we have successfully collected, created and inserted data into the database. With help of normalization technique, 13 tables have been created to implement query analysis on the collected dataset. Also, UI interface has been implemented which integrated with the database and successfully were able to add the booking details to the booking table when registered by the user in the interface of our website. We further tend to improve our database and UI by adding further constraints in future.

VIII. REFERENCES:

1. http://insideairbnb.com/

2. https://www.lucidchart.com/pages/er-diagrams

3. https://www.researchgate.net/publication/335030680_Dynamic_pricing_and_benchmarking_in_AirBnB

4. https://www.researchgate.net/publication/333021103_Understanding_AirBnB_in_Fourteen_European_Cities