Hotel Room Reservation Database Management System

Nithil Eshwar Mani
School of Engineering and Applied Sciences
University at Buffalo, USA
nithiles@*buffalo.edu*

Arvind Prashanth Sathish Kumar
School of Engineering and Applied Sciences
University at Buffalo, USA
arvindpr@buffalo.edu

Vishal Srinivas Ramesh
School of Engineering and Applied Sciences
University at Buffalo, USA
vishalsr@buffalo.edu

***Abstract*—The objective of our database management system is to efficiently manage and manipulate data related to hotel reservations. In the hospitality industry, it is mandatory to have a database management system to keep track of users' data. It helps hospitality providing companies enhance customer experience, improve operational efficiency and achieve their service goals. With an efficient booking system, we can keep track of guests' preferences and requests. The database system offers several advantages, including enforcing data integrity, scalability to handle large volumes of data, security and access controls, real-time collaboration and sharing, and advanced query and analysis capabilities. In this phase, the project focuses on normalizing the previous ER diagram by decomposing the relations to BCNF form and deploying it into a website. This ensures that the data is organized into appropriate tables and that redundant data is minimized, which helps to maintain data integrity. Additionally, the response of indexing, stored procedures, and triggers is shown. These features improve the efficiency and reliability of our hotel rooms database system. The deployment of this system to a website enables users to access the database remotely, enhancing real-time collaboration and data sharing.**

I. Introduction

The hotel room reservation system contains vast amounts of data, including information of hosts, guests, listings, amenities offered, prices, reviews, booking information and experience offered by each listing. Managing this data efficiently and accurately is a critical challenge for reservation sites. Traditionally, Excel has been often used to to manage this data. However, Excel files are limited in their ability to manage large volumes of data, enforce data integrity, provide advanced query and analysis capabilities, and ensure secure access control. Excel can accommodate only a maximum of 1,040,000 entries and 16,384 features. This is so less compared to a DBMS which can fit in much more.

A database system is a reliable and efficient solution for managing the complex data requirements of a room reservation database. Compared to an Excel file, a database system offers several advantages, such as enforcing data integrity, scalability to handle large volumes of data, security and access controls, real-time collaboration and sharing, and advanced query and analysis capabilities. Therefore, a database system is a more preferable solution for managing data on hotel room listings.

A database system can ensure data integrity by imposing constraints that ensure consistent, accurate, and reliable data. This is an important feature for any database because inaccuracies or errors can have serious consequences for both the hosts and the guests. Moreover, it is scalable and can handle large volumes of data without compromising performance, making it suitable for managing substantial amounts of data generated during processing a room reservation or creating a listing.

In addition, a database system can provide security and access controls that ensure only authorized users can access sensitive data. This ensures that confidential information is protected from unauthorized access and that the integrity of the data is maintained. Furthermore, a database system enables real-time collaboration and data sharing among multiple users, allowing for efficient collaboration. This is particularly useful for hotel booking sites with multiple hosts who are in charge of managing listings.

Finally, a database system provides complex query and analysis capabilities that offer insights and trends that may be difficult or impossible to obtain from an Excel file. Advanced query and analysis capabilities enable data administrators and analysts to gain a better understanding of applicant data and make data-driven decisions that are critical to their operations.

In summary, the proposed hotel room reservation database system is an essential tool for managing complex data requirements. The system offers several advantages over an Excel file, including enforcing data integrity, scalability, security and access controls, real-time collaboration and sharing, and advanced query and analysis capabilities. By using advanced query and analysis capabilities, the system can provide valuable insights and trends that are critical to decision-making processes..

II. Advantages Of Database Over Excel

A room reservation database management system offers several advantages over Excel. In this section, I will explain each of these advantages in detail and how they apply to managing a hotel room reservation database.

A. Search

Being able to search and conveniently fetch records for listings is a top priority in a room reservation database management system. Databases provide advanced search capabilities that enable administrators to search for data using specific criteria. These search capabilities allow administrators to filter and sort data efficiently, saving time and improving accuracy. Excel, on the other hand, has limited search capabilities and cannot handle large datasets efficiently.

B. Data Integrity

Data integrity is crucial when managing thousands of records. A room reservation database management system must ensure that the data entered into the system is complete, accurate, and consistent. A DBMS only allows valid data in the database, preserving data integrity and enforcing constraints These mechanisms check data for completeness, consistency, and accuracy before entering it into the database. Additionally, databases enforce data constraints, which minimize the risk of errors and inconsistencies.

C. Data visualization and analytics

Data analysis and visualization in hotel room reservation systems can significantly enhance both business insights and customer experiences. By analysing booking patterns, seasonal trends, and customer preferences, hotels can optimize room pricing dynamically, anticipate demand surges, and manage inventory effectively. Visualization tools can present complex data in an accessible format, aiding in quick decision-making. This could lead to targeted marketing campaigns and personalized offers based on customer behaviour, improving engagement and loyalty. For customers, data visualization enhances the booking experience, enabling them to easily compare options, understand pricing structures, and make informed decisions. Overall, these practices lead to increased operational efficiency, revenue maximization, and higher customer satisfaction.

D. Security

Security is a top priority for managing any database. DBMS ensures robust data security through features like user authentication, authorization, and access control. This helps to protect sensitive data from unauthorized access, alteration or theft Administrators can define user roles and permissions, which limit access to sensitive data. Additionally, databases provide data encryption, which protects data from unauthorized access. Excel, on the other hand, is not secure, and it can be easily copied and modified, making it challenging to maintain data confidentiality and security.

E. Data Insertion

Data insertion is a critical part of any room reservation database management system. As booking requests come in, they must be entered into the database accurately and efficiently. Databases provide a structured way to input data, ensuring consistency and accuracy. Administrators can create forms and templates that ensure consistent data entry and minimize the risk of errors. In contrast, Excel relies on manual input, which is prone to errors and inconsistencies.

F. Views

Databases allow administrators to create different views of the data, which can be customized based on user needs. These views enable administrators to analyze data efficiently, identify trends, and make informed decisions. Excel, on the other hand, has limited views and cannot handle large datasets efficiently.

G. Duplication

Databases provide mechanisms to prevent duplicate data entries, which minimize the risk of errors and inconsistencies. These mechanisms ensure that each record is unique, eliminating the risk of duplication. Excel, on the other hand, lacks built-in systems for detecting and preventing duplicate data entries, making it challenging to ensure data consistency.

H. Interactive UI

Databases provide interactive user interfaces that make it easy to input, view, and modify data. These interfaces enable administrators to navigate the data efficiently, minimizing the risk of errors and inconsistencies. Excel, on the other hand, has limited user interfaces and can be challenging to navigate, especially with large datasets.

I. Availability

Databases are available 24/7, providing administrators with instant access to data. This availability ensures that administrators can access data when needed, improving productivity and efficiency. Excel, on the other hand, is limited by the availability of the file, making it challenging to access data when needed.

J. Recommendations

Databases can provide recommendations using data gathered from analytics, enabling users to make informed decisions. These recommendations can be used to improve the admission process, identify trends, and predict outcomes. Excel, on the other hand, lacks advanced data analytics features and cannot provide recommendations based on data analysis.

III. Target Users

The two main groups who could benefit the most from a hotel room reservation database system are **Hosts** and **Guests**.

Hosts are individuals or management companies who list their properties on hotel room recommendation websites. With a robust database, hosts can manage listings, availability, pricing, and guest interactions smoothly. Host-guest interactions can include responding to booking enquiries, providing check-in instructions, addressing concerns, receiving feedback and handling issues. An efficient and robust database can handle all these interactions effectively by proving all these details to the guests in advance. Thus, in a way, a DBMS ensures clear communication and good customer service. Also, with proper data analysis and visualizations, hosts can identify trends and various factors of profitable listings. This enables them to plan their listings and improve on them over time.

Guests can use the capabilities of a DBMS to easily search for accommodations that fits their preferences. They can access elaborate details about properties, communicate with hosts, and post reviews of their experiences. The system provides an extensive selection of accommodations, ensuring transparency in costs, and simplifying the reservation procedure. With the amount of information available, guests can refine their choices, considering a variety of factors such as location, amenities, and unique property features, all of which contribute to a tailored and satisfactory travel planning process.

Other users include **Customer Support, Analysis and Business Intelligence Teams, Marketing Team and Administrators**. Customer Support teams can assist guests with enquiries and issues related to listings, reservations and payments. The process becomes smoother when all the required information is compiled in a well-organized manner. Hosts can sometimes get help from analysis and business intelligence teams to know more about the performance of the listings posted. With appropriate tools, analysis can be obtained on the various features in the database.

Business intelligence teams can also help marketing teams by generating insights from the data available. This is helpful for promoting listings and managing promotions. DBMSs' capability of handling voluminous data benefits marketing teams by providing a lot of options.

In summary, the database will serve as an essential tool in managing room reservations and listings' information.

IV. Database

A. Schema

The relations used in our hotel reservation system are as follows

1. Host
2. Guest
3. Listings
4. Price
5. Review
6. Bookings
7. Country
8. Cancellation
9. Experience
10. Bedroom
11. Bathroom
12. Property
13. Rating
14. Response
15. Status
16. Booking Status
17. Room
18. Amenity

We have used the above relations to design this database. It consists of 17 tables, each with a specific purpose and containing relevant data. The "**Host**" table stores information about the person who has listed the room, including a unique identifier (Host ID) and other relevant details. This table is used to maintain a list of hosts and associate them with their listings. The "**Guest**" table stores information about the guests, including a unique identifier (Guest ID) and other relevant details. This table is used to maintain details of guests and associate them with listings, bookings and reviews.

The "**Listings**" table stores information about all the necessary details about each listing, including a unique identifier (Listing ID) and details like the availability status, the maximum and minimum number of days provided and the guests accommodating the room. This table is used to maintain a of listings and associate them Guests, Hosts, Bookings and Reviews. The **"Price"** table stores information about the price details of the listing like Cleaning Fee and Security Deposit. It also includes a unique identifier (Price_ID). The "**Review"** table contains the rating details of each listing. It is associated with the Hosts, Guests and Listings. This table is used to maintain information about the listings' rating in terms of cleanliness and location. The "**Bookings**" table stores information about the booking status of each listing. It contains information like the check-in date, check-out date, number of days spent and booking date. This table is associated with Guests and Listings. It contains two identifiers. Booking ID and Booking status ID. This table is used to keep track of the booking status for each listing.

The "**Country**" table stores information the countries associated with each host, guest and listing. It contains a unique identifier Country ID. The "**Cancellation**" table is used to tell if a listing is cancelled and the manner in which the cancellation is processed. It is linked with the Listings table and contains a unique identifier Cancellation Id. The "Experience" table stores information about the experience offered to the guest. It has a unique identifier (Experience ID) and is linked to the Listings table. The "Bedroom" and "Bathroom" tables describe their respective facilities. They are associated with the Listings table and are associated with the Listings table. They have their own identifiers (Bedroom ID and Bathroom ID). The "**Property**" table stores information about the Property type the listing has. It has its own identifier and is associated with the Listings table.

The "**Rating**" table contains information about the listings rating and the level of rating. It has its own identifier and is associated with the Review table. The "**Response"** table contains information about the time taken for each host to respond to the bookings posted on their listings. It has its own identifier and is associated with the Host. The "**Status**" table contains information about the status of the Hosts and Guests. We can know whether a host has posted their listing or whether a guest has got a moved in based on the information provided from this table. It is linked with the Host and Guest tables has its own unique identifier (Status ID). Meanwhile the "**Booking Status**" table contains information about the booking status of each listing. It is associated with the Listing table and has its own identifier.

The "**Room**" table contains information about the type of room provided in each listing. It is associated with the Listings table and has its own identifier. The "**Amenity**" table is used to maintain information about amenity levels and also has information about the features provided. It is associated with the Listings table.

Overall, this database provides a structured and organized way to manage Hotel reservations, which is critical in ensuring that the application process runs smoothly and efficiently.

B. Dataset

To populate the tables with data, the Python Faker module was used to generate random data that closely resembles real world data. This ensures that the database can handle real world scenarios and can be used for testing and analysis purposes. The generated data was then inserted into the respective tables in the database.

To make the data accessible and shareable, it was exported to CSV files with the corresponding table names. These CSV files contain all the data that was inserted into the tables, including the primary keys and foreign keys. The files are organized and labelled appropriately to ensure easy access and management. The CSV files can be easily imported into the database, making it simple to migrate the data to a different database or to create backups. Overall, storing the data in CSV files provides a simple and effective way to manage and store the data for future use.

C. Milestone 1 to Milestone 2

The project involved creating a database management system for hotel room reservations, with a focus on managing the vast amounts of data related to applicants, universities, and their requirements. The project was completed in two milestones.

In milestone 1, the initial database schema was designed and implemented. Tables were created to store data related to hosts, guests, listings, prices pertaining to each listing, listings' reviews, bookings made, amenities, statuses for hosts and guests and responses. Each table was designed in such a way that respective primary key and foreign keys to establish relationships with other tables.

In milestone 2, the initial database schema was refined to improve its performance and optimize its storage. Some of the changes that were made included updating the table attributes, modifying primary keys, and renaming tables.

Overall, the project's goal was to create a robust and scalable database management system that could handle the vast amounts of data related to hotel room reservations. The system's design and implementation were focused on ensuring data integrity, security, scalability, and performance. The project's two milestones were crucial in achieving these goals and refining the system to ensure it meets the needs of users.

D. Table's Constraints and Description

Host:
- Guest_ID – Unique identifier for Host, RIMARY KEY
- Listing_ID – Foreign key referencing Listings table
- Booking_ID - Foreign key referencing Bookings table
- Country_ID - Foreign key referencing Country table
- Status_ID - Foreign key referencing Status table
- Response_ID - Foreign key referencing Response table
- FirstName – First Name of the host
- LastName – Last name of the host
- Date_of_birth – Date of birth of the host
- Gender – Gender of the host
- Email – email of the host
- Phone – Phone number of the host
- No_Of_Listings – Number of listings for the host
- SuperHost – Indicates whether the Host is a super host

Guest:
- Guest_ID – Unique identifier for Guest, PRIMARY KEY
- Status_ID - Foreign key referencing Status table
- Listing_ID – Foreign key referencing Listings table
- Booking_ID - Foreign key referencing Bookings table
- Country_ID - Foreign key referencing Country table
- FirstName – First Name of the Guest
- LastName – Last name of the Guest
- Date_of_birth – Date of birth of the Guest
- Gender – Gender of the Guest
- Email – email of the Guest
- Phone – Phone number of the Guest

Listings:
- Listing_ID - Unique identifier for each Listing, PRIMARY KEY
- Host_ID - Foreign key referencing Host table
- Guest_ID - Foreign key referencing Guest table
- Booking_ID - Foreign key referencing Booking table
- Price_ID - Foreign key referencing Price table
- Cancellation_ID - Foreign key referencing Cancellation table
- Experience_ID - Foreign key referencing Experience table
- Bedroom_ID - Foreign key referencing Bedroom table
- Bathroom_ID - Foreign key referencing Bathroom table
- Property_ID - Foreign key referencing Property table
- Response_ID - Foreign key referencing Response table
- Availability_ID – Identifier for the Availability details
- Amenity_ID - Foreign key referencing Amenity table
- Room_ID - Foreign key referencing Room table
- Country_ID - Foreign key referencing Country table
- Name - Name of the listing
- Description – Verbose Description of the listing,
- Location – Location of the listing
- Availability - Boolean Values indicating whether the listing is available,
- Bedrooms – Number of bedrooms in the listing
- Bathrooms – Number of bathrooms in the listing
- Minimum_Days – Minimum Number of days guests can reside in the listing
- Maximum_Days - Minimum Number of days guests can reside in the listing
- Accommodates – Details of the Guests residing the listing

Price:
- Price_ID – Unique identifier for the Price, PRIMARY KEY

- Price – The cost for the listing
- Cleaning Fee – The total cleaning fee for the listing
- Security Deposit - The total security deposit fee for the listing
- Extra Person - The total extra person fee for the listing
- Monthly Price – The total Monthly fee for the listing
- Weekly Price – The total weekly fee for the listing

Review:
- Host_ID - Foreign key referencing Host table
- Guest_ID - Foreign key referencing Guest table
- Listing_ID - Foreign key referencing Listing table
- Host_ID, Guest_ID and the Listing_ID together make up the PRIMARY KEY
- Rating_ID - Foreign key referencing Rating table
- Review Date – The date in which the review was posted for the listing.
- Total Rating – The total rating given to the listing.
- Communication_Rating – The rating given to the hosts based on communication.
- Cleanliness_Rating – The rating given to the listing based on its cleanliness.
- Location_Rating – The rating given to the listing's location.
- CheckIn_Rating – The rating given to the check-in experience

Bookings:
- Booking_ID – Unique identifier for each booking, PRIMARY KEY
- Guest_ID – Foreign Key referencing the Guest table
- Listing_ID – Foreign Key referencing the Listings table
- Booking_Status_ID – Foreign Key referencing the Booking Status table
- CheckInDate – The check in date for that reservation
- CheckOutDate – The check-out date for that reservation
- Number_Of_Days – The total number of days spent
- Booking_Date – The date in which the reservation was made.

Country:
- Country_ID – Unique identifier for the country, PRIMARY KEY
- Country – The name of the Country

Cancellation:
- Cancellation_ID – Unique identifier for the cancellation type, PRIMARY KEY
- Country – The type of Cancellation

Experience:
- Experience_ID – Unique identifier for the Experience Offered, PRIMARY KEY
- Country – The type of Experience potentially offered to the guests

Bedroom:
- Bedroom_ID – Unique identifier for the Bed type, PRIMARY KEY
- BedType – The type of Bed available in the listing

Bathroom:
- Bathroom_ID – Unique identifier for the Bathroom type, PRIMARY KEY
- BathroomType – The type of Bathrooms available in the listing

Property:
- Property_ID – Unique identifier for the Property type, PRIMARY KEY
- PropertyType – The listing's Property type

Rating:
- Rating_ID – Unique identifier for the Rating given to the listing, PRIMARY KEY
- RatingLevel – The listing's Rating level
- Rating – The rating given to the listing

Response:
- Response _ID – Unique identifier for the Response time, PRIMARY KEY
- ResponseTime – The time taken by the hosts to respond to the guest's requests

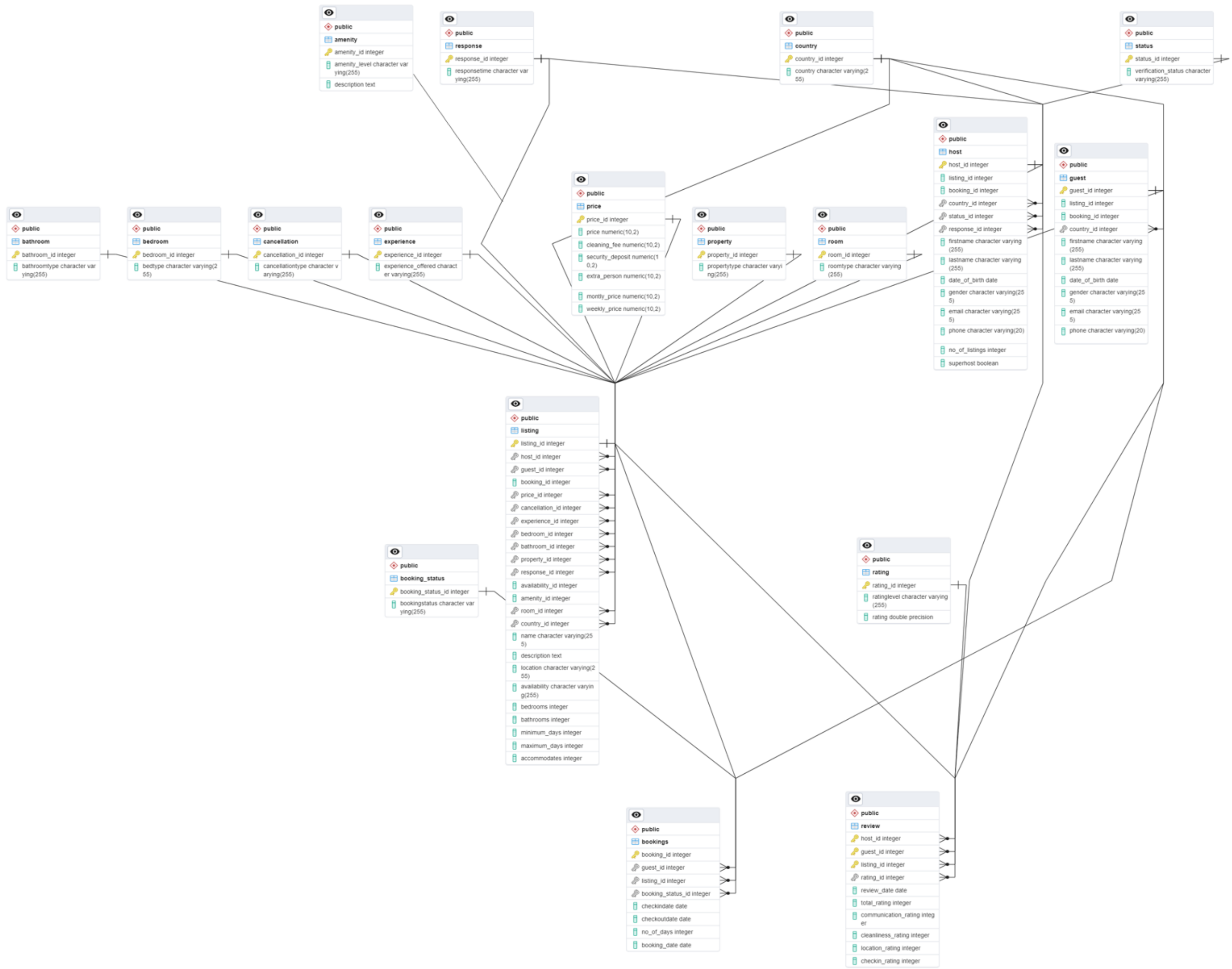
Status:
- Status_ID – Unique identifier for the Verification status, PRIMARY KEY
- Verification_Status – The verification status of the host and the guest

Booking Status:
- Booking_Status_ID – Unique identifier for the Booking status, PRIMARY KEY
- Verification_Status – The booking status of the listing

Room:
- Room_ID – Unique identifier for the listing's room type, PRIMARY KEY
- RoomType – The listing's room type

Amenity:
- Amenity_ID - Unique identifier for the listing's room type, PRIMARY KEY
- Amenity_level – The level of amenities in the listing
- Description – Verbose description of the amenities provided.

V. ER Diagram



VI. Normalization

Normalization is the process of organizing data in a database to eliminate redundancy and dependency. It involves dividing the data into smaller, more manageable tables, and establishing relationships between them to improve data consistency, integrity, and efficiency. Normalization helps prevent data anomalies and inconsistencies, making it easier to maintain and modify the database over time.

For a table to be in BCNF (Boyce-Codd Normal Form), it must satisfy the following conditions:

1. Every determinant (i.e., attribute that determines another attribute) must be a candidate key.
2. Every non-trivial functional dependency (i.e., dependency between non-prime attributes) must have a determinant that is a candidate key.

Based on the columns in the tables and the functional dependencies we have obtained the following decompositions:

1.**Host** (Host_ID,Listing_ID,Booking_ID,Country_ID, Status_ID,Response_ID,FirstName,LastName,Date_Of_Birth,Gender,Email,Phone, No_Of_Listings,SuperHost)

Functional dependencies

- Phone-> FirstName, LastName
- Email->FirstName, LastName

Given the host's phone number or email id, we can determine the person's first name and last name as it is unique for each person. We won't get another person's record given these details. Hence we have considered these two as functional dependencies. Here the left side is not a superkey. Therefore they are not in BCNF. So this table must be decomposed into 2 tables.

The 2 tables are as follows:

- **Host** (Host_ID,Phone,Listing_ID,Booking_ID,Country_ID,Status_ID,Response_ID,Date_Of_Birth, Gender,Email,No_Of_Listings,SuperHost)
- **Host_Details** (Phone,FirstName,LastName)

2.**Guest**(Guest_ID,Status_ID,Listing_ID,Booking_ID,Country_ID,FirstName,LastName,Date_Of_Birth,Gender,Email,Phone)

Functional dependencies

- Phone-> FirstName,LastName
- Email->FirstName,LastName

Similarly for the Guest table, given the host's phone number or email id, we can determine the person's first name and last name. So the left side is not a super key. Therefore they are not in BCNF. So this table must be decomposed into 2 tables.

The 2 tables are as follows:

- **Guest** (Guest_ID,Status_ID,Listing_ID,Booking_ID,Country_ID,Date_Of_Birth,Gender,Email,Phone)
- **Guest_Details** (Email, FirstName, LastName)

3.**Listings** (Listing_ID,Host_ID,Guest_ID,Booking_ID,Price_ID, Cancellation_ID, Experience_ID,Bedroom_ID, Bathroom_ID, Property_ID, Response_ID Availability_ID,Amentity_ID,Room_ID,Country_ID,Name, Description,Location, Availability, Bedrooms,Bathrooms,Minimum_Days,Maximum_Days, Accommodates)

Functional dependencies

- Listing_ID->Host_ID, Guest_ID, Booking_ID, Price_ID, Cancellation_ID, Experience_ID,Bedroom_ID, Bathroom_ID, Property_ID, Response_ID Availability_ID, Amentity_ID, Room_ID, Country_ID, Name, Description, Location, Availability, Bedrooms, Bathrooms, Minimum_Days, Maximum_Days, Accommodates

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

4.**Price** (Price_ID,Price,Cleaning_Fee,Security_Deposit,Extra_Person,Montly_Price,Weekly_Price)

Functional dependencies

- Price_ID>Price, Cleaning_Fee, Security_ Deposit,Extra_ Person,Montly_ Price, Weekly_Price

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed

5.**Review** (Host_ID,Guest_ID,Listing_ID,Rating_ID,Review_Date,Total_Rating,Communication_Rating,Cleanliness_Rating,Location_Rating,CheckIn_Rating)

Functional dependencies

- Host_ID, Guest_ID, Listing_ID –> Rating_ID, Review_Date, Total_Rating, Communication_Rating, Cleanliness_Rating, Location_Rating, CheckIn_Rating

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

6.**Bookings** (Booking_ID,Guest_ID,Listing_ID,Booking_Status_ID,CheckInDate,CheckOutDate,No_Of_Days, Booking_Date)

Functional dependencies

- CheckInDate, CheckOutDate -> No_of_days

Here No_of_days can be obtained by subtracting Check_out_date and Check_in_date, it could be considered a trivial dependency. Since No_of_days might be obvious from the schema itself this is a trivial dependency. Therefore the particular table is in BCNF.

7.**Country** (Country_ID, Country)

Functional dependencies

- Country_ID-> Country

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

8.**Cancellation** (Cancellation_ID,CancellationType)

Functional dependencies

- Cancellation_ID -> CancellationType

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

9.**Experience** (Experience_ID,Experience_offered)

Functional dependencies

- Experience_ID->Experience_offered

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

10.**Bedroom** (Bedroom_ID,BedType)

Functional dependencies

- Bedroom_ID->BedType

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

11.**Bathroom** (Bathroom_ID,BathroomType)

Functional dependencies

- Bathroom_ID->BathroomType

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

12.**Property** (Property_ID,PropertyType)

Functional dependencies

- Property_ID->PropertyType

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

6. **Rating** (Rating_ID,RatingLevel,Rating)

Functional dependencies

- Rating_ID-> RatingLevel,Rating
- Rating -> RatingLevel

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. This table is already in BCNF. Hence it can't be further decomposed.

7. **Response** (Response_ID,ResponseTime)

Functional dependencies

- Response_ID -> ResponseTime

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

8. **Status** (Status_ID,Verification_Status)

Functional dependencies

- Status_ID-> Verification_Status

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

9. **Booking_Status**(Booking_Status_ID,BookingStatus)

Functional dependencies

- Booking_Status_ID-> BookingStatus

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed

10. **Room** (Room_ID,RoomType)

Functional dependencies

- Room_ID-> RoomType

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

11. **Amentity**(Amentity_ID, Amentity_level,Description)

Functional dependencies

- Amentity_ID->Amentity_level,Description

There are no trivial functional dependencies apart from the one where primary key defines all the attributes. Hence this table is already in BCNF and hence can't be further decomposed.

VII. DATABASE LOOKUP

Host Table

```
1 select * from host;
```

Data Output Messages Notifications

	host_id [PK] integer	listing_id integer	booking_id integer	country_id integer	status_id integer	response_id integer	firstname character varying (255)	lastname character varying (255)	date_of_birth date	gender character varying (255)	email character v
1	1	1	1	9	1	1	Carol	Gonzalez	1967-02-24	Female	diaznichola
2	2	2	2	6	1	1	Rodney	Boyd	1996-10-02	Male	santiagowil
3	3	3	3	6	2	1	Amanda	Glass	1992-10-10	Female	shannon94
4	4	4	4	9	2	1	Ashley	Melton	1965-08-18	Male	bonilladavi
5	5	5	5	7	2	3	Robert	Cook	1971-08-28	Male	dennis84@
6	6	6	6	4	2	1	April	Cox	1987-08-04	Female	ifrey@exam
7	7	7	7	4	2	3	Jerry	Hawkins	1993-05-22	Male	bwhite@ex
8	8	8	8	7	1	2	Andrew	Mccormick	1990-12-13	Male	gabriela80
9	9	9	9	3	1	3	Jessica	Sanchez	1996-08-08	Male	scott62@e
10	10	10	10	2	1	3	David	Gonzalez	1998-03-21	Female	amber19@
11	11	11	11	5	2	2	Jessica	Stanley	2001-07-11	Male	tylerjoseph
12	12	12	12	9	2	1	Mitchell	Turner	1992-10-28	Male	sstanley@e
13	13	13	13	2	2	2	Daniel	Smith	1965-12-04	Male	wgarcia@e
14	14	14	14	5	2	2	Bianca	Cobb	1967-06-21	Female	eric58@ex
15	15	15	15	2	1	1	William	Wilson	1983-05-27	Female	mjohnson@
16	16	16	16	1	2	3	Jason	Robinson	1973-09-06	Male	gpeterson@
17	17	17	17	9	2	1	Kevin	Jordan	2002-08-05	Male	brian23@e
18	18	18	18	6	2	3	Joseph	Brooks	1972-04-19	Male	sophiaboyl
19	19	19	19	4	2	1	Emily	[illegible]	[illegible]	[illegible]	[illegible]

✓ Successfully run. Total query runtime: 137 msec. 10000 rows affected.

HostDetails Table

	phone [PK] character varying (20)	firstname character varying (255)	lastname character varying (255)
1	2375515606	Kyle	Perez
2	7062644971	Shawn	Ramos
3	8346534566	David	Copeland
4	7265311026	Jamie	Glover
5	2539332399	Steven	Martinez
6	8056410598	Frances	Massey
7	9049583311	Tasha	Meyer
8	6176376670	Stephanie	Brown
9	5773722939	Brandon	Wright
10	7156415329	James	Johnson

Total rows: 1000 of 10000 Query complete 00:00:00.147

Listing Table

```
1 select * from listing;
```

Data Output Messages Notifications

	listing_id [PK] integer	host_id integer	guest_id integer	booking_id integer	price_id integer	cancellation_id integer	experience_id integer	bedroom_id integer	bathroom_id integer	property_id integer	response_id integer	availability_id integer	ameni intege
1	1	1	1	1	1	3	5	4	1	8	3	2	
2	2	2	2	2	2	2	3	1	3	2	1	5	
3	3	3	3	3	3	3	3	3	1	8	3	1	
4	4	4	4	4	4	3	3	2	1	6	2	2	
5	5	5	5	5	5	2	5	1	2	7	1	3	
6	6	6	6	6	6	3	4	3	3	5	2	2	
7	7	7	7	7	7	1	1	2	2	6	1	4	
8	8	8	8	8	8	3	1	1	3	3	3	2	
9	9	9	9	9	9	2	1	1	3	4	1	5	
10	10	10	10	10	10	2	3	4	2	7	1	1	
11	11	11	11	11	11	3	2	3	3	6	3	1	
12	12	12	12	12	12	1	1	4	1	7	2	5	
13	13	13	13	13	13	1	5	2	2	6	2	2	
14	14	14	14	14	14	2	5	1	2	6	3	1	
15	15	15	15	15	15	2	2	4	1	7	3	4	
16	16	16	16	16	16	2	4	2	2	8	1	2	
17	17	17	17	17	17	3	4	2	2	8	3	2	
18	18	18	18	18	18	2	3	2	1	5	2	4	
19	19	19	19	19	19	1	4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

✓ Successfully run. Total query runtime: 208 msec. 10000 rows affected.

Guest Table

```
1 select * from guest;
```

Data Output Messages Notifications

	guest_id [PK] integer	listing_id integer	booking_id integer	country_id integer	firstname character varying (255)	lastname character varying (255)	date_of_birth date	gender character varying (255)	email character varying (255)	phone chara
1	1	1	1	1	Travis	Williams	1990-04-15	Male	hamiltonchristopher@example.com	2918
2	2	2	2	3	Austin	Hall	1975-08-29	Male	david49@example.org	5939
3	3	3	3	5	Brenda	Harper	1989-08-29	Female	jonathan39@example.com	3207
4	4	4	4	8	Kelly	Miller	1998-01-21	Male	lopezdavid@example.com	3214
5	5	5	5	5	Sarah	Zhang	1971-10-11	Male	kyle09@example.org	4755
6	6	6	6	7	Derek	Brown	1973-09-16	Male	ljohnson@example.org	2943
7	7	7	7	5	Tamara	Smith	1977-05-19	Male	wmoore@example.com	3377
8	8	8	8	8	John	Robinson	1979-06-14	Male	othomas@example.net	4549
9	9	9	9	1	Jennifer	Kim	1963-11-10	Male	christopher90@example.net	7086
10	10	10	10	7	Connor	Solis	1990-02-09	Female	michellewoodard@example.com	9022
11	11	11	11	1	Brittany	Wilson	1976-07-02	Male	william48@example.com	8437
12	12	12	12	10	Francisco	Lee	1998-12-01	Female	monica20@example.net	8316
13	13	13	13	1	Terri	Hawkins	1969-02-28	Male	udavis@example.com	3738
14	14	14	14	8	Victoria	Scott	1995-02-16	Male	bushchristopher@example.net	6539
15	15	15	15	5	Diana	Gonzalez	1963-03-23	Female	acevedojames@example.com	3544
16	16	16	16	2	Karen	Oconnell	1969-08-08	Female	briannarogers@example.com	2968
17	17	17	17	3	Carol	Rice	1972-03-10	Female	sandra59@example.net	3142
18	18	18	18	6	Lucas	Walker	1990-04-08	Male	samuel88@example.com	2777
19	19	19	19	5	Patricia	Kim	[illegible]	[illegible]	[illegible]	[illegible]

✓ Successfully run. Total query runtime: 127 msec. 10000 rows affected.

GuestDetails Table

	email [PK] character varying (255)	firstname character varying (255)	lastname character varying (255)
1	jesasmith0587@example.com	Jesa	Smith
2	danicline0597@example.com	Dani	Cline
3	erikkey0795@example.com	Erik	Key
4	christywilliams0278@example.com	Christy	Williams
5	joanyoung0885@example.com	Joan	Young
6	ronaldwhitehead0178@example.com	Ronald	Whitehead
7	davidburgess0672@example.com	David	Burgess
8	keithjones0588@example.com	Keith	Jones
9	noahpierce1074@example.com	Noah	Pierce
10	kevinclarke1197@example.com	Kevin	Clarke

Total rows: 1000 of 10000 Query complete 00:00:00.141

Price Table

```
1 select * from price;
```

Data Output Messages Notifications

	price_id [PK] integer	price numeric (10,2)	cleaning_fee numeric (10,2)	security_deposit numeric (10,2)	extra_person numeric (10,2)	montly_price numeric (10,2)	weekly_price numeric (10,2)
1	1	70.00	20.00	40.00	10.00	2450.00	150.00
2	2	80.00	10.00	80.00	10.00	1230.00	110.00
3	3	50.00	30.00	50.00	30.00	2520.00	170.00
4	4	10.00	30.00	10.00	40.00	2670.00	270.00
5	5	10.00	30.00	90.00	20.00	1740.00	210.00
6	6	40.00	10.00	50.00	30.00	2530.00	230.00
7	7	50.00	20.00	70.00	50.00	2590.00	200.00
8	8	20.00	30.00	30.00	40.00	1980.00	170.00
9	9	30.00	30.00	80.00	40.00	1380.00	300.00
10	10	50.00	10.00	70.00	30.00	2750.00	150.00
11	11	50.00	30.00	50.00	30.00	1580.00	200.00
12	12	30.00	30.00	90.00	30.00	2920.00	130.00
13	13	60.00	20.00	70.00	30.00	2260.00	230.00
14	14	20.00	20.00	60.00	30.00	1610.00	100.00
15	15	70.00	30.00	50.00	50.00	1700.00	140.00
16	16	40.00	10.00	90.00	10.00	2910.00	260.00
17	17	10.00	30.00	80.00	40.00	2160.00	110.00
18	18	70.00	20.00	20.00	10.00	2910.00	230.00
19	19	50.00	10.00	70.00	10.00	2780.00	[illegible]
20	20	90.00	20.00	30.00	20.00	1660.00	[illegible]

Successfu

Review Table

```
1 select * from review;
```

Data Output Messages Notifications

	host_id [PK] integer	guest_id [PK] integer	listing_id [PK] integer	rating_id integer	review_date date	total_rating integer	communication_rating integer	cleanliness_rating integer	location_rating integer	checkin_rating integer
1	1	1	1	1	2023-10-03	3	1	5	1	4
2	2	2	2	2	2023-03-20	4	4	2	5	1
3	3	3	3	1	2023-10-05	1	4	2	1	1
4	4	4	4	4	2024-01-28	1	4	4	5	1
5	5	5	5	4	2023-03-23	4	4	4	4	1
6	6	6	6	1	2023-06-29	4	4	3	1	4
7	7	7	7	5	2020-10-15	5	1	1	1	4
8	8	8	8	4	2022-05-04	1	5	5	3	5
9	9	9	9	1	2019-05-16	5	4	3	3	4
10	10	10	10	2	2022-05-02	2	4	2	4	3
11	11	11	11	4	2019-03-22	5	5	4	5	3
12	12	12	12	3	2020-10-04	5	5	4	1	4
13	13	13	13	2	2022-05-09	1	3	4	1	3
14	14	14	14	5	2020-10-27	2	5	4	5	5
15	15	15	15	2	2022-12-20	4	3	2	5	4
16	16	16	16	2	2023-04-22	3	2	3	4	3
17	17	17	17	4	2024-02-22	5	3	3	5	5
18	18	18	18	2	2020-08-31	2	5	4	4	4
19	19	19	19	5	2022-09-24	4	1			

✓ Successfully run. Total query runtime: 74 msec. 10 r

Bookings Table

```
1 select * from bookings;
```

Data Output Messages Notifications

	booking_id [PK] integer	guest_id integer	listing_id integer	booking_status_id integer	checkindate date	checkoutdate date	no_of_days integer	booking_date date
1	1	1	1	2	2023-07-21	2023-07-26	5	2023-07-11
2	2	2	2	1	2023-04-27	2023-05-01	4	2023-04-18
3	3	3	3	3	2023-12-08	2023-12-09	1	2023-11-22
4	4	4	4	3	2023-10-23	2023-11-04	12	2023-10-19
5	5	5	5	1	2023-12-19	2023-12-26	7	2023-12-05
6	6	6	6	2	2023-06-09	2023-06-18	9	2023-05-19
7	7	7	7	1	2024-01-05	2024-01-09	4	2023-12-06
8	8	8	8	2	2024-01-08	2024-01-20	12	2023-12-17
9	9	9	9	1	2023-05-26	2023-05-29	3	2023-05-01
10	10	10	10	3	2023-11-27	2023-12-11	14	2023-11-07
11	11	11	11	2	2023-07-30	2023-08-01	2	2023-07-08
12	12	12	12	2	2023-02-27	2023-03-05	6	2023-02-08
13	13	13	13	2	2023-05-03	2023-05-15	12	2023-05-03
14	14	14	14	2	2023-10-29	2023-11-11	13	2023-10-03
15	15	15	15	3	2024-02-06	2024-02-10	4	2024-01-07
16	16	16	16	3	2023-04-09	2023-04-15	6	2023-04-09
17	17	17	17	2	2023-03-04	2023-03-18	14	2023-02-14
18	18	18	18	2	2024-02-11	2024-02-24	13	2024-01-23
19	19	19	19	2	2024-01-30	2024-02-12	13	2024-01-19
20	20	20	20	3	2024-01-15	2024-01-16	1	2023-12-24

Country Table

```
1 select * from country;
```

Data Output Messages Notifications

	country_id [PK] integer	country character varying (255)
1	1	USA
2	2	Canada
3	3	UK
4	4	Australia
5	5	Germany
6	6	France
7	7	Italy
8	8	Spain
9	9	China
10	10	India

Cancellation Table

	cancellation_id [PK] integer	cancellationtype character varying (255)
1	1	Flexible
2	2	Moderate
3	3	Strict
4	4	Non-Refundable

Experience Table

	experience_id [PK] integer	experience_offered character varying (255)
1	1	Business
2	2	Family
3	3	Romantic
4	4	Adventure
5	5	Leisure

Bedroom Table

	bedroom_id [PK] integer	bedtype character varying (255)
1	1	Standard Bed
2	2	Sofa Bed
3	3	Futon
4	4	Bunk Bed

Bathroom Table

	bathroom_id [PK] integer	bathroomtype character varying (255)
1	1	Ensuite Bathroom
2	2	Shared Bathroom
3	3	Full Bathroom

Property Table

	property_id [PK] integer	propertytype character varying (255)
1	1	Apartment
2	2	House
3	3	Bed and Breakfast
4	4	Loft
5	5	Townhouse
6	6	Condo
7	7	Cabin
8	8	Villa
9	9	Castle
10	10	Mansion

Rating Table

	rating_id [PK] integer	ratinglevel character varying (255)	rating double precision
1	1	Excellent	5
2	2	Very Good	4.5
3	3	Good	4
4	4	Fair	3.5
5	5	Poor	3

Response Table

	response_id [PK] integer	responsetime character
1	1	Immediate
2	2	Within a few ho...
3	3	Within a day

Status Table

	status_id [PK] integer	verification_status character varying (255)
1	1	Verified
2	2	Unverified

Booking_Status Table

	booking_status_id [PK] integer	bookingstatus character varying (255)
1	1	Confirmed
2	2	Cancelled
3	3	Pending

Room Table

	room_id [PK] integer	roomtype character varying (255)
1	1	Entire Place
2	2	Private Room
3	3	Shared Room
4	4	Hotel Room
5	5	Dormitory

Amenity Table

	amenity_id [PK] integer	amenity_level character varying (255)	description text
1	1	Basic	wifi,heating,air-conditioning
2	2	Standard	wifi,heating,air-conditioning,hair dryer,iron
3	3	Comfort	wifi,heating,air-conditioning,hair dryer,iron,TV,Kitchen
4	4	Premium	wifi,heating,air-conditioning,hair dryer,iron,TV,Kitchen,Patio,Hot water
5	5	Luxury	wifi,heating,air-conditioning,hair dryer,iron,TV,Kitchen,Patio,Hot water,Fire pit,BBQ ...

VIII. SQL QUERIES

1. Find the top 5 super hosts in terms of number of listings and the average rating given to them.

```sql
SELECT
    H.Host_ID,
    HD.FirstName,
    HD.LastName,
    H.No_Of_Listings AS NumberOfListings,
    round(AVG(R.Total_Rating)) AS AverageRating
FROM
    Host H
INNER JOIN
    HostDetails HD ON H.Phone = HD.Phone
INNER JOIN
    Review R ON H.Host_ID = R.Host_ID
WHERE
    H.SuperHost = 'Yes'
GROUP BY
    H.Host_ID, HD.FirstName, HD.LastName
ORDER BY
    NumberOfListings DESC, AverageRating DESC
LIMIT 5;
```

	host_id integer	firstname character varying (255)	lastname character varying (255)	numberoflistings integer	averagerating numeric
1	7944	Danielle	Herrera	10	5
2	8907	Michelle	Mcdonald	10	5
3	4207	Christina	Lozano	10	5
4	8064	Stanley	Pineda	10	5
5	7090	Rachel	Cohen	10	5

2. Find the highest-rated listings of a specific property type within locations containing 'MA', sorting results by property type and descending ratings

```sql
SELECT
Property.PropertyType,
Listing.Listing_ID,
Listing.Name,
Listing.Location,
MaxReview.Total_Rating
FROM
Property
INNER JOIN
Listing ON Property.Property_ID = Listing.Property_ID
INNER JOIN
(SELECT
Listing_ID,
MAX(Total_Rating) AS Total_Rating
FROM
Review
GROUP BY
Listing_ID) MaxReview ON Listing.Listing_ID = MaxReview.Listing_ID
WHERE
Listing.Location LIKE '%MA%'
ORDER BY
Property.PropertyType,
MaxReview.Total_Rating DESC;
```

	propertytype character varying (255)	listing_id integer	name character varying (255)	location character varying (255)	total_rating integer
1	Apartment	645	Education fund while.	05954 Sharon Rue	5
2	Apartment	3276	Remain middle charge.	5657 Todd Fall Suite 917	5
3	Apartment	6414	Keep score.	24126 Kevin Course Apt. 197	4
4	Apartment	5806	Under tree world draw.	750 Laura Extension	4
5	Apartment	6965	Take myself wall decide.	317 Gonzalez Spurs	4
6	Apartment	1201	Fill opportunity.	21524 Valencia Village Apt. 525	3
7	Apartment	347	Attorney hot stop do might.	8713 Barnes Groves Suite 055	3
8	Apartment	4959	Serious feeling bad social account.	6633 Henry Prairie	2
9	Apartment	4322	Face relationship couple marriage.	972 Jeffrey Manors Suite 913	2
10	Apartment	1253	Kid down seem.	7900 Joshua Garden	2

3. Find the check-in date, check-out date, listing's name, description, location, price, property type, room type, amenity level and bathroom type for the guest with Guest_ID 23.

Query Query History Scratch Pad

```sql
SELECT
Bookings.CheckInDate,
Bookings.CheckOutDate,
Listing.Name ,
Listing.Description,
Listing.Location,
Booking_Status.BookingStatus,
price.Price,
property.PropertyType,
room.RoomType,
amenity.Amenity_level,
bathroom.BathroomType
FROM
Bookings
JOIN Listing  ON Bookings.Listing_ID = Listing.Listing_ID
JOIN Booking_Status ON Bookings.Booking_Status_ID = booking_status.Booking_Status_ID
JOIN Price ON Listing.Price_ID = Price.Price_ID
JOIN Property ON Listing.Property_ID = property.Property_ID
JOIN Room ON Listing.Room_ID = room.Room_ID
JOIN Amenity ON Listing.Amenity_ID = amenity.Amenity_ID
JOIN Bathroom ON Listing.Bathroom_ID = bathroom.Bathroom_ID
WHERE Bookings.Guest_ID = 23;
```

Data Output Messages Notifications

	checkindate date	checkoutdate date	name character varying (255)	description text	location character varying (255)	book char
1	2023-06-03	2023-06-13	Under ground.	Dinner I recently wall account. Easy executive front contain. See whom run third important fat...	7009 Adam Branch	Can

4. Find the Listing ID, name, description, location, property type, bed type, price, bathroom type, amenity level, rating level and cancellation type of the available listings.

Query Query History

```sql
SELECT
listing.Listing_ID,
listing.Name,
listing.Description,
listing.Location,
property.propertytype,
bedroom.bedtype,
price.price,
bathroom.bathroomtype,
amenity.amenity_level,
rating.RatingLevel,
cancellation.CancellationType
FROM
 listing
INNER JOIN property ON listing.Property_ID = property.Property_ID
INNER JOIN bedroom ON listing.Bedroom_ID = bedroom.Bedroom_ID
INNER JOIN bathroom ON listing.Bathroom_ID = bathroom.Bathroom_ID
INNER JOIN amenity ON listing.Amenity_ID = amenity.amenity_ID
INNER JOIN price ON listing.Price_ID = price.price_id
INNER JOIN review ON listing.listing_id = review.listing_id
INNER JOIN rating ON review.rating_id = rating.rating_id
INNER JOIN cancellation ON listing.Cancellation_ID = cancellation.Cancellation_ID
INNER JOIN country ON listing.Country_ID = country.Country_ID
WHERE listing.Availability = 'Available'
```

	listing_id integer	name character varying (255)	description text
1	1	Cup though class.	Social say people language. Right spend but artist age true quite.
2	3	Market me record.	Already voice tonight note any cause through full. Bag music trade against individual tell find. Through movement radio. East indicate group democra
3	4	Responsibility foot policy power go.	Identify need fly require. Seek technology detail job.
4	5	Foot care along.	Significant heart less many. Sport floor administration successful dream whom.
5	6	Address rather all.	Dog police church collection.
6	7	Industry population international hundred ho...	Reality might financial determine civil program. Whole moment American all six. Prevent must bit minute industry.
7	8	Free know mention.	Media oil structure head accept. Her meet few with new. Structure various understand actually song nature character.
8	9	Idea street article.	Discussion join full democratic player card ready. Official finally threat clearly shake type until.
9	11	Between wind both yard.	Recent student age subject executive heavy make partner. Live forward effort picture activity.
10	12	Third across large better computer.	Girl nothing manage green line trade fall. Sit figure offer final city material every.

5. Find the listings available for booking, count for each pending bookings order by number of pending bookings.

```sql
WITH PendingBookings AS (
    SELECT
        B.Listing_ID,
        COUNT(B.Booking_ID) AS PendingCount
    FROM
        Bookings B
    JOIN
        Booking_Status BS ON B.Booking_Status_ID = BS.Booking_Status_ID
    WHERE
        BS.BookingStatus = 'Pending'
    GROUP BY
        B.Listing_ID
),
AvailableListings AS (
    SELECT
        L.Listing_ID,
        L.Name,
        L.Availability,
        L.Location
```

Data Output Messages Notifications

	listing_id [PK] integer	name character varying (255)	location character varying (255)	pendingcount bigint
1	2335	Also five window.	Unit 4409 Box 2584	1
2	273	Information director American able.	984 Hays Crossing	1
3	51	Ahead fire.	22248 Teresa Shoals Apt. 927	1
4	6906	Positive contain listen.	546 Stewart Crest Apt. 258	1
5	5468	Analysis edge economy.	21126 Desiree Mountain Apt. 853	1

Total rows: 1000 of 2294 Query complete 00:00:00.091

6. Find the listings with the highest average ratings and the most reviews, ranked the highest within each country and return these top-ranked listings.

```sql
WITH RankedListings AS (
  SELECT
    C.Country,
    L.Listing_ID,
    L.Name,
    COUNT(R.Rating_ID) AS NumberOfReviews,
    ROUND(AVG(R.Total_Rating)) AS AverageRating,
    RANK() OVER (
      PARTITION BY C.Country_ID
      ORDER BY AVG(R.Total_Rating) DESC, COUNT(R.Rating_ID) DESC
    ) AS RatingRank
  FROM
    Listing L
    JOIN Review R ON L.Listing_ID = R.Listing_ID
    JOIN Country C ON L.Country_ID = C.Country_ID
  GROUP BY
    C.Country_ID, L.Listing_ID
)
```

Data Output Messages Notifications

	country character varying (255)	listing_id integer	name character varying (255)	numberofreviews bigint	averagerating numeric	ratingrank bigint
1	USA	483	Himself really brother whole.	1	5	1
2	USA	8549	Continue know view sea chance.	1	5	1
3	USA	5321	During why though.	1	5	1
4	USA	7509	Hard land since authority.	1	5	1
5	USA	3875	Soon campaign current.	1	5	1
6	USA	9114	Term stop agreement.	1		

✓ Successfully ru

Total rows: 1000 of 2073 Query complete 00:00:00.082

7. Update the Hosts table. Changing the host name as 'Jack Daniels' and set their super host attribute to 'No' for the host with id = '12345' and phone number 123456789.

```sql
Insert into hostdetails(Phone,Firstname,Lastname)
Values('123456789','Peter','Parker');

INSERT INTO Host (Host_ID,Phone,Listing_ID,Booking_ID,Country_ID,Status_ID,Response_ID,Date_Of_Birth,Gender,Email,No_Of_Listings,SuperHost)
VALUES ('12345','123456789',NULL,NULL,'1','1','1','1999-07-15','Male','newhost@example.com',0,'Yes');
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 37 msec.
```

```sql
UPDATE Host
SET
No_Of_Listings = 10,
SuperHost = 'No'
WHERE Host_ID = '12345';


UPDATE HostDetails
SET
FirstName = 'Jack',
LastName = 'Daniels'
WHERE Phone = '123456789'
```

Data Output Messages Notifications

```
UPDATE 1

Query returned successfully in 41 msec.
```

Query Query History

```sql
DELETE FROM Host
WHERE Host_ID = '12345';

DELETE FROM HostDetails
WHERE Phone = '123456789';
```

Data Output Messages Notifications

```
DELETE 1

Query returned successfully in 41 msec.
```

8. Find the verified super hosts with an immediate response time, ordered by average review rating and with a rating above 3.

```sql
SELECT H.Host_ID, HD.FirstName, HD.LastName, ROUND(AVG(R.Total_Rating)) as AverageRating
FROM Host as H
JOIN Status as S ON H.Status_ID = S.Status_ID AND S.Verification_Status = 'Verified'
JOIN Response as Res ON H.Response_ID = Res.Response_ID AND Res.ResponseTime = 'Immediate'
JOIN Review as R ON H.Host_ID = R.Host_ID
JOIN HostDetails as HD ON H.Phone = HD.Phone
WHERE H.SuperHost = 'Yes'
GROUP BY H.Host_ID, HD.FirstName, HD.LastName
HAVING AVG(R.Total_Rating) > 3
ORDER BY AverageRating DESC;
```

	host_id integer	firstname character varying (255)	lastname character varying (255)	averagerating numeric
1	5237	Rickey	Roberts	5
2	5255	Douglas	Wilson	5
3	5259	Elizabeth	Marquez	5
4	2665	Anna	Hudson	5
5	2692	Jake	Jones	5
6	5308	Robert	Evans	5
7	2738	Diane	Camacho	5
8	1373	Monique	Swanson	5
9	1377	Jerry	Richards	5
10	2976	Rhonda	Frye	5

9. Find the IDs and names of ‘Flexible’ cancellation type. The results are grouped by the IDs, names and cancellation types

```sql
SELECT
  Listing.Listing_ID,Listing.Name,Cancellation.CancellationType
FROM
  Listing
JOIN
  Cancellation ON Listing.Cancellation_ID = Cancellation.Cancellation_ID
WHERE
  Cancellation.CancellationType = 'Flexible'
GROUP BY
   Listing.Listing_ID,Listing.Name,Cancellation.CancellationType;
```

Data Output Messages Notifications

	listing_id integer	name character varying (255)	cancellationtype character varying (255)
1	7	Industry population international hundred hope.	Flexible
2	12	Third across large better computer.	Flexible
3	13	Indicate tough account.	Flexible
4	19	Say loss look.	Flexible
5	21	Per guess statement.	Flexible
6	28	Herself tonight project.	Flexible
7	31	Better relationship billion player mention.	Flexible
8	34	Now must although.	Flexible
9	35	Ready word effect rather heart.	Flexible
10	38	Mind another card church big.	Flexible
11	40	Number truth part.	Flexible
12	41	Wall office everybody investment next.	Flexible

10. Find the count of unique guests with ‘confirmed ‘ booking status.

```sql
SELECT COUNT(DISTINCT a.Guest_Id) AS No_Of_Guests_Booked
FROM Guest a
INNER JOIN Listing b ON a.Listing_ID = b.Listing_ID
INNER JOIN Bookings c ON b.Listing_ID = c.Listing_ID
INNER JOIN Booking_Status d ON c.Booking_Status_ID = d.Booking_Status_ID
WHERE d.BookingStatus = 'Confirmed';
```

Data Output Messages Notifications

	no_of_guests_booked bigint
1	3263

IX. INDEXING

Index 1

Query with indexing

```sql
SELECT
    Price_ID, Weekly_Price
FROM
    Price
WHERE
    Weekly_Price > 200
ORDER BY
    Weekly_Price ASC;
```

Data Output Messages Notifications

	price_id [PK] integer	weekly_price numeric (10,2)
1	5	210.00
2	21	210.00
3	23	210.00
4	48	210.00
5	75	210.00
6	76	210.00
7	78	210.00
8	111	210.00
9	157	210.00
10	165	210.00
11	171	210.00
12	175	210.00
13	176	210.00

Total rows: 1000 of 4875 Query complete 00:00:00.073

We have created an index for the weekly_price in the price table. With indexing we only look for the rows in which the weekly_price is greater than 200.

Query without indexing

```sql
SELECT
    Price_ID, Weekly_Price
FROM
    Price
WHERE
    Weekly_Price > 200
ORDER BY
    Weekly_Price ASC;
```

Data Output Messages Explain Notifications

	price_id [PK] integer	weekly_price numeric (10,2)
1	1936	210.00
2	1935	210.00
3	6817	210.00
4	4852	210.00
5	6808	210.00
6	9572	210.00
7	288	210.00
8	1898	210.00
9	1891	210.00
10	4596	210.00
11	1886	210.00
12	2629	210.00
13	111	210.00
14	9990	210.00

Total rows: 1000 of 4875 Query complete 00:00:00.075

Without indexing the whole table is scanned and the values that match the criteria are returned.

Compared to queries without indexing, the results are returned much quicker because instead of looking in the entire table, we directly return the values that match the criteria. This can significantly improve the performance of the query.

Index 2

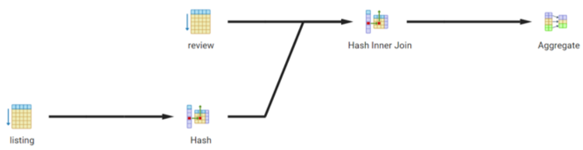
Query and output without indexing

```sql
SELECT
    L.Listing_ID,
    L.Name AS ListingName,
    L.Maximum_Days,
    L.Accommodates,
    H.Host_ID,
    HD.FirstName,
    HD.LastName
FROM
    Listing L
INNER JOIN Host H ON L.Host_ID = H.Host_ID
INNER JOIN HostDetails HD ON H.Phone = HD.Phone
WHERE
    L.Maximum_Days > 6
    AND L.Accommodates > 5
```

Data Output Messages Notifications

	listing_id integer	listingname character varying (255)	maximum_days integer	accommodates integer	host_id integer	firstname character varying (255)	lastname character varying (255)
1	3236	Apply not federal.	7	8	3236	Frances	Massey
2	4067	Hand must rule image.	7	6	4067	Valerie	Rubio
3	2080	Drug price show.	7	6	2080	Taylor	Wright
4	1038	Box cut popular.	7	8	1038	Kathy	Allen
5	9741	Collection rather power become emplo...	7	7	9741	Richard	Kelly
6	3713	Firm herself.	7	6	3713	William	Evans
7	1507	Method seat where still.	7	8	1507	Amber	Farley
8	8669	Agreement need.	7	6	8669	Fernando	Bray

Total rows: 1000 of 1368 Query complete 00:00:00.090



Without indexing, the 'listing' and 'review' tables are joined using a hash join method and the result is then passed to an aggregation step. Indexes provide a pre-sorted. They have a tree based structure and can quickly find the matching entries without having to scan the entire table.

Query for indexing

```sql
create index idx_listing_bedroom on listing(bedrooms);
create index idx_listing_bathroom on listing(bathrooms);
create index idx_review_rating on review(total_rating);
```

Data Output Messages Explain × Notifications

```
CREATE INDEX

Query returned successfully in 57 msec.
```

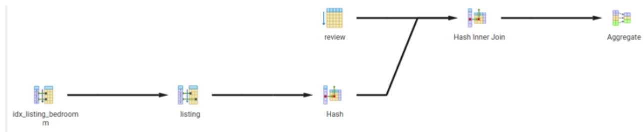
Here we have created indexes for the bedroom and bathroom attributes in the listings table. We also have created an index for the total_rating column in the review table.

Output with indices

	listing_id integer	listingname character varying (255)	maximum_days integer	accommodates integer	host_id integer	firstname character varying (255)	lastname character varying (255)
1	3236	Apply not federal.	7	8	3236	Frances	Massey
2	4067	Hand must rule image.	7	6	4067	Valerie	Rubio
3	2080	Drug price show.	7	6	2080	Taylor	Wright
4	1038	Box cut popular.	7	8	1038	Kathy	Allen
5	9741	Collection rather power become emplo...	7	7	9741	Richard	Kelly
6	3713	Firm herself.	7	6	3713	William	Evans
7	1507	Method seat where still.	7	8	1507	Amber	Farley
8	8669	Agreement need.	7	6	8669	Fernando	Bray

Total rows: 1000 of 1368 Query complete 00:00:00.084

The query runs faster when the indices are created for the tables based on which the output is being generated.

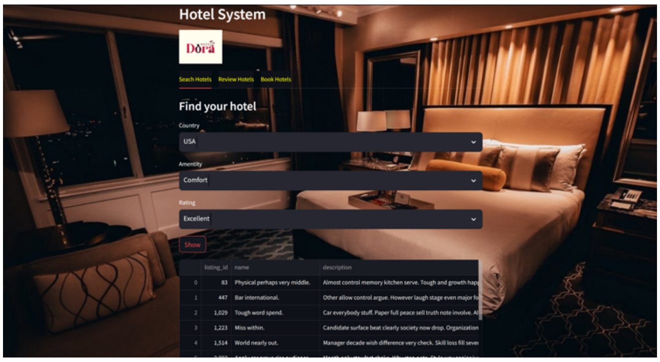


To summarize, indices provide structured, quick paths for data retrieval with minimal scanning. Because of their tree based data structure, rows can be fetched efficiently rather than having to scan the entire dataset.

X. WEBSITE

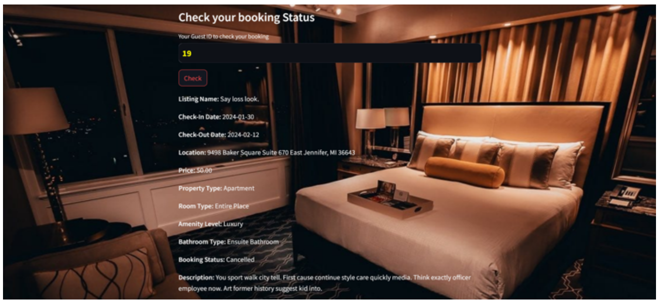
Website link - https://hotelbookingsystem.streamlit.app/

1. Find Your Hotel



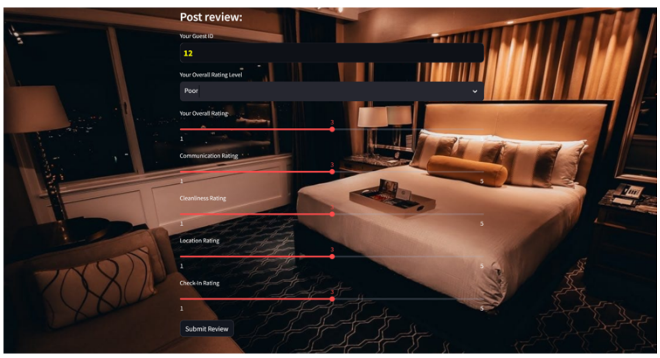
The first webpage of our website. Given the desired Country, Amenity category and the Rating, the records matching the given criteria are fetched and displayed.

2. Check Your Booking Status



In this page, given a Guest_ID, we can check their booking status. We can also get the listing's name, check-in date, check-out date, listing's location, price, property type, room type, amenity level, bathroom type, bedroom type and the description.

3. Post Review



In this page, we can post the review for the listing. This page is used by the guest and they can give their rate their overall experience, their communication experience, location and check-in experience.

XI – REFERENCES

1. Professor's lecture slides - Functional dependency and BCNF normalization
2. Database Systems – Ulman 2nd edition
3. https://www.w3schools.com/
4. https://public.opendatasoft.com/explore/dataset/airbnb-listings/api/
5. https://docs.streamlit.io/develop/api-reference
6. https://www.kaggle.com/datasets/arianazmoudeh/airbnbopendata/data