New York Airbnb Open Data 2024

CSE 460: MILESTONE 2

Prachi Vijay Patil
ppatil8
ppatil8@buffalo.edu
50539237

Monika Jangam Prabhudev
monikaja
monikaja@buffalo.edu
50537185

Bhimireddy Sai Gayathri Reddy
saigayat
saigayat@buffalo.edu
50526445

***Abstract*—This dataset, which records New York City's most recent Airbnb listing activity as of January 5, 2024, is designed to assist financial analysts and company managers who are looking for thorough insights into the profitable world of short- and long-term house stays.**

Focusing on more than 20 parameters, this dataset is a valuable tool for financial analysts who want to perform an in-depth financial study of New York's Airbnb operations. Precise market assessments are made possible by the fine details, which make it easier to identify pricing patterns, popular room kinds, and regional concentrations.

The information provides company managers with a comprehensive overview of the varied Airbnb landscape in New York, facilitating well-informed decision-making. Whether planning investments, making the best

We have taken the data set from Kaggle and imported the data to the table through CSV files

I. PROBLEM STATEMENT

Efficient inventory management is paramount for businesses to sustain profitability. The proposed database system endeavors to address the imperative need for efficient storage, organization, and retrieval of Airbnb listing data in New York City. The provided dataset encompasses critical information such as latitude, longitude, room type, price, ratings, bedrooms, beds, and baths, among other pertinent details. However, users often encounter challenges in finding the perfect property within their budget constraints, compounded by factors such as location and neighborhood preferences.

Our project aims to alleviate these challenges by leveraging the comprehensive dataset to assist users in discovering properties that align with their budget and desired location and neighborhood criteria. Separate tables have been designed to retrieve more detailed information for easier business understanding and analysis. For instance, the database schema involves tables for listings, hosts, availability, neighborhoods, ratings, reviews, locations, prices, and rooms.

Along with that, by organizing data into separate tables, stakeholders can gain deeper insights into various aspects of Airbnb listings, such as geographical distribution, pricing trends, room type popularity, and neighborhood preferences. This structured approach facilitates easier analysis to determine which areas are thriving and where potential investments could yield higher returns, enabling more informed decision-making processes for stakeholders in the Airbnb market.

Background and Objectives: Manual handling of Airbnb listing data often leads to inconsistencies, missed opportunities, and operational inefficiencies. These challenges underscore the necessity for a robust database system to streamline the management of Airbnb listings in New York City. Moreover, renters face challenges in finding suitable properties within their budget and preferred location, highlighting the need for a solution that enhances the rental experience.

Our primary objective is to develop a comprehensive database system that optimizes the management of Airbnb listings in New York City. By seamlessly managing listings, host information, availability, pricing, and other relevant data, our system aims to empower stakeholders with actionable insights to maximize returns and capitalize on opportunities in the dynamic Airbnb market of New York City. Additionally, our system will enhance the rental experience for users by providing tailored recommendations based on their budget constraints, location preferences, and other criteria, thereby simplifying the property search process and ensuring a seamless renting experience.

II. TARGET USER

Airbnb Administrators and Analysts: Professionals responsible for monitoring and analyzing overall listing trends, performance metrics, and user behaviors. They may use the database to generate reports, track key performance indicators, and make data-driven decisions.

Property Managers and Hosts: Individuals managing Airbnb properties or hosts renting out their homes. They may use the database to update listing details, view booking information, and track guest reviews.

Travelers and Guests: People searching for accommodations on Airbnb. They may use the database indirectly through the Airbnb platform to find suitable listings based on their preferences and requirements.

Customer Support Representatives: Individuals handling customer inquiries, issues, and support tickets related to Airbnb listings. They may use the database to retrieve information about specific bookings, resolve problems, and provide assistance to users.

Database Administrators (DBAs): Skilled professionals responsible for the design, implementation, maintenance, and security of the database system. They ensure data integrity, optimize performance, and manage backups and recovery processes.management system database:

III. BCNF CHECK AND DECOMPOSITION

To ensure that all relations are in Boyce-Codd Normal Form (BCNF), we need to identify functional dependencies and eliminate any dependencies on non-superkey attributes. Below are the dependencies for each relation:

A. Location

- Dependencies:
  - $LocationID \rightarrow NeighbourhoodGroup,$ $Neighourhood, Longitude, Latitude$
- All attributes are functionally dependent on the $LocationID$ (primary key).
- No partial dependencies or transitive dependencies exist.

B. Listing

- Dependencies:
  - $ID \rightarrow HostID$
- The ID uniquely determines $HostID$.
- No partial dependencies or transitive dependencies exist.

C. Ratings

- Dependencies:
  - $RatingID \rightarrow Rating$
- The $RatingID$ uniquely determines the $Rating$ of the property.
- No partial dependencies or transitive dependencies exist.

D. Reviews

- Dependencies:
  - $ID \rightarrow NumberOfReviews, LastReview,$ $ReviewsPerMonth, RatingID$
- The ID uniquely determines all other attributes.
- No partial dependencies or transitive dependencies exist.

E. Price

- Dependencies:
  - $PriceID \rightarrow Price, LocationID, RatingID$
- The $PriceD$ uniquely determines all other attributes.
- No partial dependencies or transitive dependencies exist.

F. Room

- Dependencies:
  - $RoomID \rightarrow RoomType, Bedrooms, Beds,$ $Bath, PriceID$
- The $RoomID$ uniquely determines all other attributes.
- No partial dependencies or transitive dependencies exist.

G. Availability

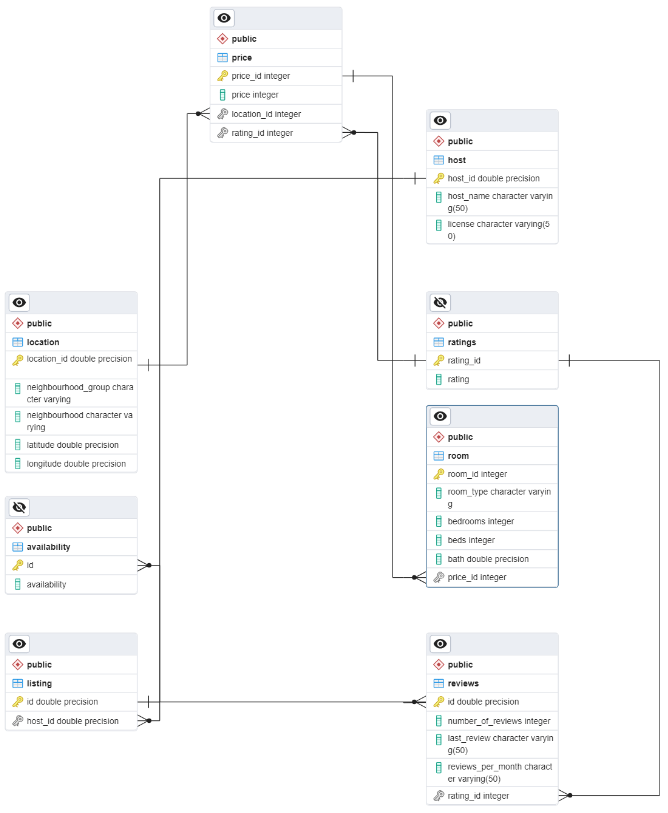
- Dependencies:
  - $ID \rightarrow Availability$
- The ID uniquely determines $Availability$.
- No partial dependencies or transitive dependencies exist.

H. Host

- Dependencies:
  - $HostID \rightarrow HostName, License$
- The $HostID$ uniquely determines $HostName, License$.
- No partial dependencies or transitive dependencies exist.

BCNF Check and Decomposition: All tables appear to be in BCNF as there are no non-trivial functional dependencies on any proper subset of the primary keys. Hence no decomposition is required.

IV. E/R DIAGRAM



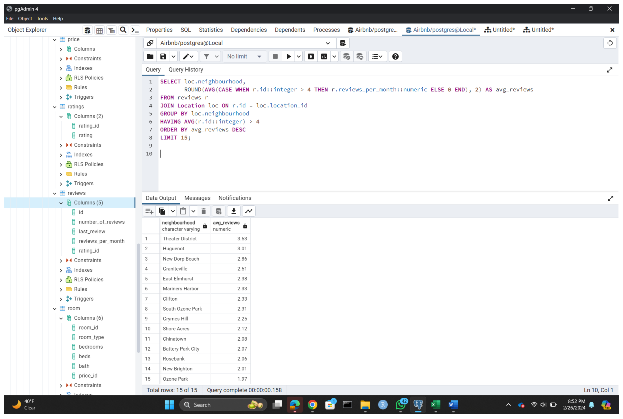
E/R Diagram

V. SQL QUERIES

Here are several sample queries that not only demonstrate different features and functionalities of the system but also serve as proof of its functionality. These queries showcase various clauses such as WHERE, different types of JOINs, ORDER BY, GROUP BY, and other functions:

VI. DATABASE TABLE STRUCTURE

We have modified tables Host, Listing to keep them in BCNF Form.

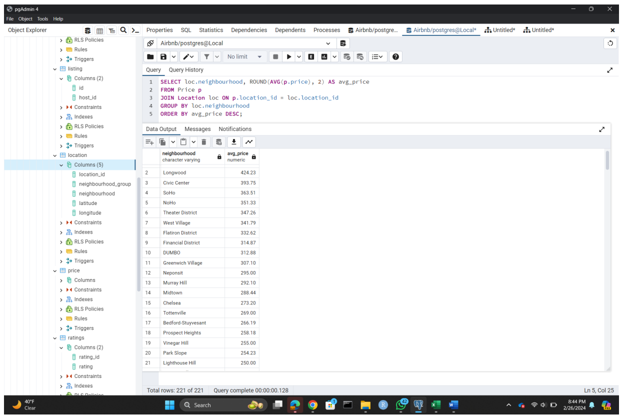


```sql
SELECT loc.neighbourhood,
       ROUND(AVG(CASE WHEN r.id::integer > 4 THEN r.reviews_per_month::numeric ELSE 0 END), 2) AS avg_reviews
FROM reviews r
JOIN Location loc ON r.id = loc.location_id
GROUP BY loc.neighbourhood
HAVING AVG(r.id::integer) > 4
ORDER BY avg_reviews DESC
LIMIT 15;
```

	neighbourhood character varying	avg_reviews numeric
1	Theater District	3.53
2	Huguenot	3.01
3	New Dorp Beach	2.86
4	Graniteville	2.51
5	East Elmhurst	2.38
6	Mariners Harbor	2.33
7	Clifton	2.33
8	South Ozone Park	2.31
9	Grymes Hill	2.25
10	Shore Acres	2.12
11	Chinatown	2.08
12	Battery Park City	2.07
13	Rosebank	2.06
14	New Brighton	2.01
15	Ozone Park	1.97

Total rows: 15 of 15 Query complete 00:00:00.158

Fig. 1. This SQL query retrieves the top 15 neighborhoods with the highest average reviews per month, among those with an average rating greater than 4.

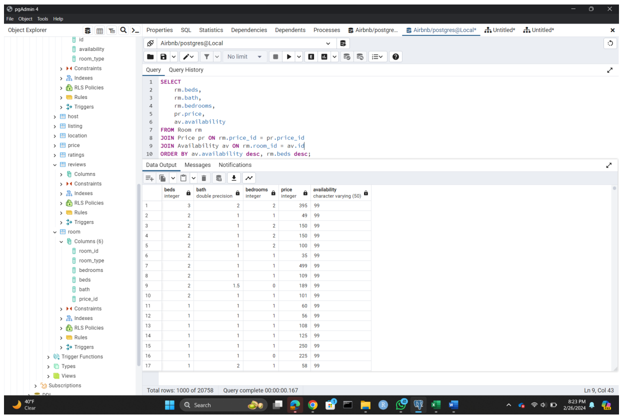


```sql
SELECT loc.neighbourhood, ROUND(AVG(p.price), 2) AS avg_price
FROM Price p
JOIN Location loc ON p.location_id = loc.location_id
GROUP BY loc.neighbourhood
ORDER BY avg_price DESC;
```

	neighbourhood character varying	avg_price numeric
2	Longwood	424.23
3	Civic Center	393.75
4	SoHo	363.51
5	NoHo	351.33
6	Theater District	347.26
7	West Village	341.79
8	Flatiron District	332.62
9	Financial District	314.87
10	DUMBO	312.88
11	Greenwich Village	307.10
12	Neponsit	295.00
13	Murray Hill	292.10
14	Midtown	288.44
15	Chelsea	273.20
16	Tottenville	269.00
17	Bedford-Stuyvesant	266.19
18	Prospect Heights	258.18
19	Vinegar Hill	255.00
20	Park Slope	254.23
21	Lighthouse Hill	250.00

Total rows: 221 of 221 Query complete 00:00:00.128

Fig. 2. This SQL query calculates the average price for each neighborhood, rounding the result to two decimal places and arranging the neighborhoods in descending order based on the average price.

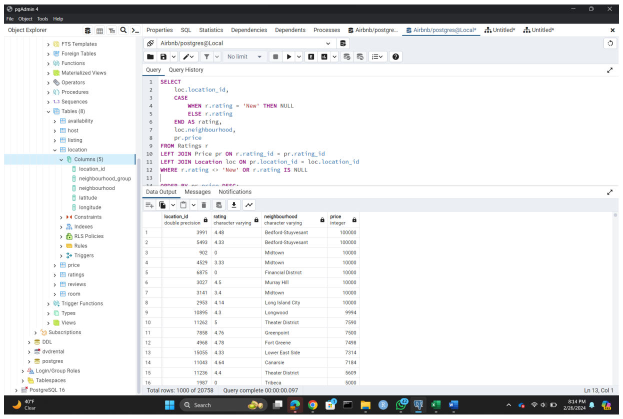


```sql
SELECT
    rm.beds,
    rm.bath,
    rm.bedrooms,
    pr.price,
    av.availability
FROM Room rm
JOIN Price pr ON rm.price_id = pr.price_id
JOIN Availability av ON rm.room_id = av.id
ORDER BY av.availability desc, rm.beds desc;
```

	beds integer	bath double precision	bedrooms integer	price integer	availability character varying (50)
1	3	2	2	395	99
2	2	1	1	49	99
3	2	1	2	150	99
4	2	1	2	150	99
5	2	1	2	100	99
6	2	1	1	35	99
7	2	1	1	499	99
8	2	1	1	109	99
9	2	1.5	0	189	99
10	2	1	1	101	99
11	1	1	1	60	99
12	1	1	1	56	99
13	1	1	1	108	99
14	1	1	1	125	99
15	1	1	1	250	99
16	1	1	0	225	99
17	1	2	1	58	99

Total rows: 1000 of 20758 Query complete 00:00:00.167

Fig. 3. The SQL query selects the number of beds, bath, bedrooms, price, and availability from the Room, Price, and Availability tables, joining them based on their respective keys, and arranges the results by availability in descending order and then by the number of beds in descending order.

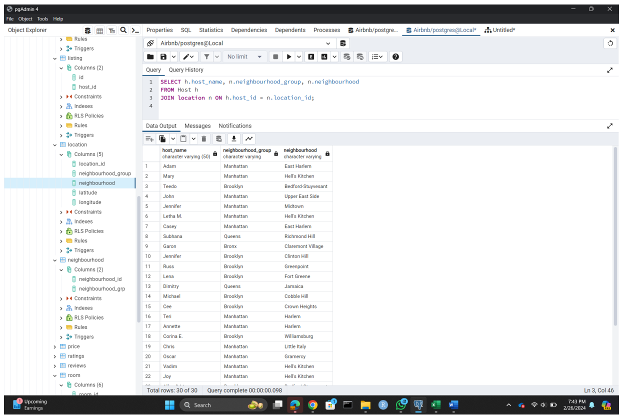


```sql
SELECT
    loc.location_id,
    CASE
        WHEN r.rating = 'New' THEN NULL
        ELSE r.rating
    END AS rating,
    loc.neighbourhood,
    pr.price
FROM Ratings r
LEFT JOIN Price pr ON r.rating_id = pr.rating_id
LEFT JOIN Location loc ON pr.location_id = loc.location_id
WHERE r.rating <> 'New' OR r.rating IS NULL
```

	location_id double precision	rating character varying	neighbourhood character varying	price integer
1	3991	4.48	Bedford-Stuyvesant	100000
2	5493	4.33	Bedford-Stuyvesant	100000
3	902	0	Midtown	10000
4	4529	3.33	Midtown	10000
5	6875	0	Financial District	10000
6	3027	4.5	Murray Hill	10000
7	3141	3.4	Midtown	10000
8	2953	4.14	Long Island City	10000
9	10895	4.3	Longwood	9994
10	11262	5	Theater District	7590
11	7858	4.76	Greenpoint	7500
12	4968	4.78	Fort Greene	7498
13	15055	4.33	Lower East Side	7314
14	11043	4.64	Canarsie	7184
15	11236	4.4	Theater District	5609
16	1987	0	Tribeca	5000

Total rows: 1000 of 20758 Query complete 00:00:00.097

Fig. 4. This SQL query provides the Highest Price based on the rating and neighborhood.

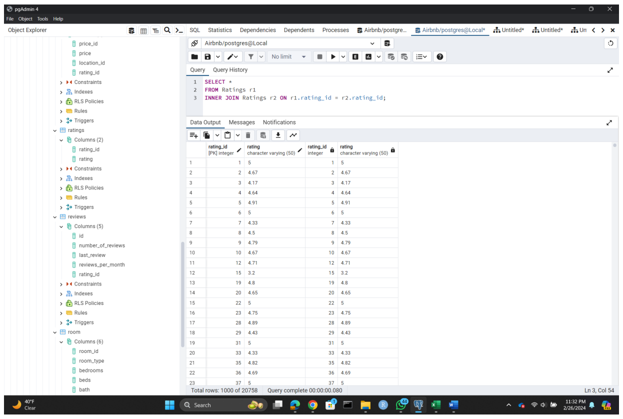


```sql
SELECT h.host_name, n.neighbourhood_group, n.neighbourhood
FROM Host h
JOIN location n ON h.host_id = n.location_id;
```

	host_name character varying (50)	neighbourhood_group character varying	neighbourhood character varying
1	Adam	Manhattan	East Harlem
2	Mary	Manhattan	Hell's Kitchen
3	Teedo	Brooklyn	Bedford-Stuyvesant
4	John	Manhattan	Upper East Side
5	Jennifer	Manhattan	Midtown
6	Letha M.	Manhattan	Hell's Kitchen
7	Casey	Manhattan	East Harlem
8	Subhana	Queens	Richmond Hill
9	Garon	Bronx	Claremont Village
10	Jennifer	Brooklyn	Clinton Hill
11	Russ	Brooklyn	Greenpoint
12	Lena	Brooklyn	Fort Greene
13	Dimitry	Queens	Jamaica
14	Michael	Brooklyn	Cobble Hill
15	Cee	Brooklyn	Crown Heights
16	Teri	Manhattan	Harlem
17	Annette	Manhattan	Harlem
18	Corina E.	Brooklyn	Williamsburg
19	Chris	Manhattan	Little Italy
20	Oscar	Manhattan	Gramercy
21	Vadim	Manhattan	Hell's Kitchen
22	Joy	Manhattan	Hell's Kitchen

Total rows: 30 of 30 Query complete 00:00:00.098

Fig. 5. This SQL query provides Hosts with their property locations in the New York area.



```sql
SELECT *
FROM Ratings r1
INNER JOIN Ratings r2 ON r1.rating_id = r2.rating_id;
```

	rating_id [PK] integer	rating character varying (50)	rating_id integer	rating character varying (50)
1	1	5	1	5
2	2	4.67	2	4.67
3	3	4.17	3	4.17
4	4	4.64	4	4.64
5	5	4.91	5	4.91
6	6	5	6	5
7	7	4.33	7	4.33
8	8	4.5	8	4.5
9	9	4.79	9	4.79
10	10	4.67	10	4.67
11	12	4.71	12	4.71
12	15	3.2	15	3.2
13	19	4.8	19	4.8
14	20	4.65	20	4.65
15	22	5	22	5
16	23	4.75	23	4.75
17	28	4.89	28	4.89
18	29	4.43	29	4.43
19	31	5	31	5
20	33	4.33	33	4.33
21	35	4.82	35	4.82
22	36	4.69	36	4.69
23	37	5	37	5

Total rows: 1000 of 20758 Query complete 00:00:00.080

Fig. 6. This SQL query provides inner join for ratings table

A. Location Table

- **LocationID** (Primary Key, INT): Unique identifier for each location.
- **NeighbourhoodGroup** (VARCHAR(255)): Group information for the neighborhood.

- **Neighbourhood** (VARCHAR(255)): Name of the neighborhood.
- **Longitude** (INT): Longitude of the listing.
- **Latitude** (INT): Latitude of the listing.

B. Listing Table

- **ID** (Primary Key, INT): Unique identifier for each listing
- **HostID** (INT): ID of the host.

C. Ratings Table

- **RatingID** (Primary Key, INT): Unique identifier for each property rating.
- **Rating** (VARCHAR(255)): Rating of the property.

D. Reviews Table

- **ID** (Primary Key, Foreign Key, INT): Unique identifier for each listing.
- **NumberOfReviews** (INT): Number of reviews for a property.
- **LastReview** (VARCHAR(255)): Date of the last review.
- **ReviewsPerMonth** (VARCHAR(255)): Reviews per month.
- **RatingID** (INT, Foreign Key): Unique Rating identifier for each listing.
- RatingID is a foreign key for the Reviews table taken from the Ratings table
- ID is a foreign key and primary key for the Reviews table taken from the Listing table

E. Host Table

- **HostID** (Primary Key, INT): Unique identifier for each host.
- **HostName** (VARCHAR(255)): Name of the host.
- **license** (VARCHAR(255)): Details of the host license.
- HostID is a foreign key for the listing table.

F. Room Table

- **RoomId** (Primary Key, INT): Unique identifier for each Room.
- **RoomType** (VARCHAR(255)): Details of type of room (e.g., private room, entire home/apt).
- **Bedrooms** (INT): Number of bedrooms.
- **Beds** (INT): Number of beds in the room.
- **Bath** (FLOAT): Number of bathrooms.
- **PriceId** (INT): Foreign key referencing the price table associated with the room..

G. Price Table

- **PriceId** (Primary Key, INT): Unique identifier for each price entry.
- **Price** (INT): Price information associated with the location.
- **LocationId** (INT): Foreign key referencing the location.
- **RatingId** (Foreign Key, INT): Foreign key referencing the rating associated with the price.
- *Actions on Foreign Key*: No action on delete.

H. Availability Table

- **ID** (Primary Key, INT): Unique identifier for availability table.
- **Availability** (VARCHAR(50)): Availability status of the listing.
- **ID** Foreign key referencing the id associated with the listing table.
- *Actions on Foreign Key*: No action on delete.

VII. Problems and Solutions Handling Large Databases

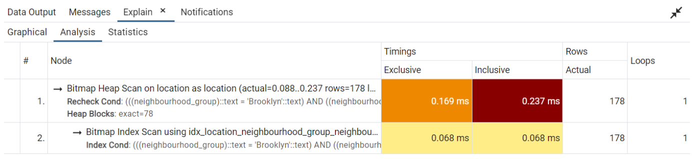
Handling larger datasets, such as here this dataset with 21,000 rows in each relation, can present several challenges that may become more pronounced as the dataset grows. These challenges primarily revolve around performance issues, including slower query execution times and increased resource consumption. Slower query execution times can significantly impact application responsiveness and user experience, particularly when executing complex queries or joining multiple tables. To address this, implementing indexing on frequently queried columns and optimizing SQL queries can improve data retrieval efficiency. Additionally, increased resource consumption, including CPU, memory, and disk space, can strain system resources and impact overall performance and scalability. Here is how they can be tackled:

A. Slow Query Performance

```sql
SELECT *
FROM Location
WHERE neighbourhood_group = 'Brooklyn'
  AND neighbourhood = 'Clinton␣Hill';
```

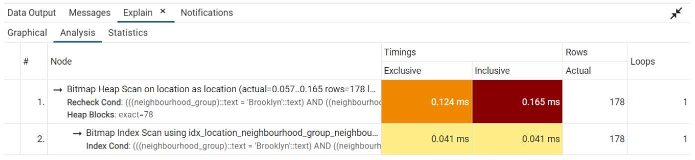
Adding appropriate indexes on the columns involved in the WHERE clause (`neighbourhood_group` and `neighborhood`) allows the database to quickly locate the relevant rows, significantly reducing the time needed to execute the query. This is because indexes enable the database engine to efficiently narrow down the search to only those rows that meet the specified conditions, resulting in faster query execution times

```sql
CREATE INDEX idx_availability_id ON
    ↪ Availability (id);

CREATE INDEX idx_host_host_id ON Host (
    ↪ host_id);

CREATE INDEX idx_listing_host_id ON
    ↪ Listing (host_id);

CREATE INDEX idx_location_location_id ON
    ↪ Location (location_id);
```



#	Node	Timings		Rows	Loops
		Exclusive	Inclusive	Actual	
1.	→ Bitmap Heap Scan on location as location (actual=0.088..0.237 rows=178 l... **Recheck Cond**: (((neighbourhood_group)::text = 'Brooklyn'::text) AND ((neighbourhoo **Heap Blocks**: exact=78	0.169 ms	0.237 ms	178	1
2.	→ Bitmap Index Scan using idx_location_neighbourhood_group_neighbou... **Index Cond**: (((neighbourhood_group)::text = 'Brooklyn'::text) AND ((neighbourh	0.068 ms	0.068 ms	178	1

Fig. 7. **Before Indexing**



#	Node	Timings		Rows	Loops
		Exclusive	Inclusive	Actual	
1.	→ Bitmap Heap Scan on location as location (actual=0.057..0.165 rows=178 l... **Recheck Cond**: (((neighbourhood_group)::text = 'Brooklyn'::text) AND ((neighbourhoo **Heap Blocks**: exact=78	0.124 ms	0.165 ms	178	1
2.	→ Bitmap Index Scan using idx_location_neighbourhood_group_neighbou... **Index Cond**: (((neighbourhood_group)::text = 'Brooklyn'::text) AND ((neighbourh	0.041 ms	0.041 ms	178	1

Fig. 8. **After Indexing**

B. Optimized Resource Utilization

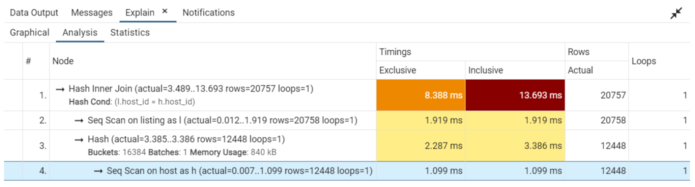
Indexing aids in efficiently managing database server resources like memory and CPU, crucial for handling larger datasets. By optimizing queries and indexes, the strain on resources diminishes, reducing the risk of performance degradation or server crashes.

C. Enhanced Concurrency Management

With indexing, concurrent access by multiple users or processes becomes more manageable, even with expansive datasets. Effective locking mechanisms and transaction management strategies ensure smooth operation, minimizing contention and performance bottlenecks.

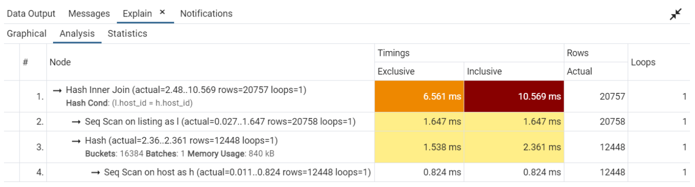
D. Data Integrity Maintenance

Indexes contribute to maintaining data integrity by enforcing constraints such as uniqueness and referential integrity. By enforcing constraints at the index level, indexing helps ensure



#	Node	Timings		Rows	Loops
		Exclusive	Inclusive	Actual	
1.	→ Hash Inner Join (actual=3.489..13.693 rows=20757 loops=1) **Hash Cond**: (l.host_id = h.host_id)	8.388 ms	13.693 ms	20757	1
2.	→ Seq Scan on listing as l (actual=0.012..1.919 rows=20758 loops=1)	1.919 ms	1.919 ms	20758	1
3.	→ Hash (actual=3.385..3.386 rows=12448 loops=1) **Buckets**: 16384 **Batches**: 1 **Memory Usage**: 840 kB	2.287 ms	3.386 ms	12448	1
4.	→ Seq Scan on host as h (actual=0.007..1.099 rows=12448 loops=1)	1.099 ms	1.099 ms	12448	1

Fig. 9. **Before Indexing**



#	Node	Timings		Rows	Loops
		Exclusive	Inclusive	Actual	
1.	→ Hash Inner Join (actual=2.48..10.569 rows=20757 loops=1) **Hash Cond**: (l.host_id = h.host_id)	6.561 ms	10.569 ms	20757	1
2.	→ Seq Scan on listing as l (actual=0.027..1.647 rows=20758 loops=1)	1.647 ms	1.647 ms	20758	1
3.	→ Hash (actual=2.36..2.361 rows=12448 loops=1) **Buckets**: 16384 **Batches**: 1 **Memory Usage**: 840 kB	1.538 ms	2.361 ms	12448	1
4.	→ Seq Scan on host as h (actual=0.011..0.824 rows=12448 loops=1)	0.824 ms	0.824 ms	12448	1

Fig. 10. **After Indexing**

data consistency and accuracy, enhancing data quality and reliability.

E. Efficient Backup and Recovery

Indexing streamlines backup and recovery processes, crucial for large databases. By enabling faster data retrieval, indexing reduces the time and resources needed for backup operations, ensuring robust data protection and quicker recovery in case of emergencies.

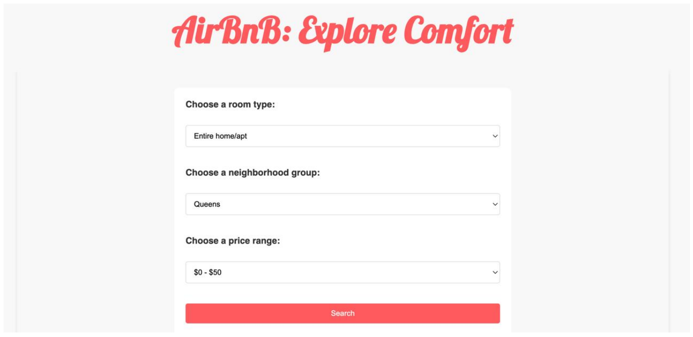
F. Scalability Enhancement

Indexes facilitate database scalability by improving query performance and resource utilization. With indexes in place, the database can handle larger datasets and increase user loads more efficiently, supporting scalability requirements.

VIII. Web Portal Implementation

We have created a web portal where the user can select the room type (such as private room, Entire home/Apt) and neighborhood/Location he is looking and the portal will display The Host Name, Beds, Bedroom, Bath, Price, Availability of the property.

IX. Working Demo

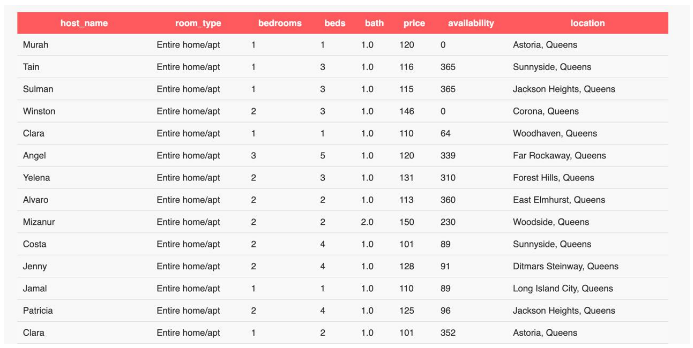

AirBnB: Explore Comfort

Choose a room type:
Entire home/apt

Choose a neighborhood group:
Queens

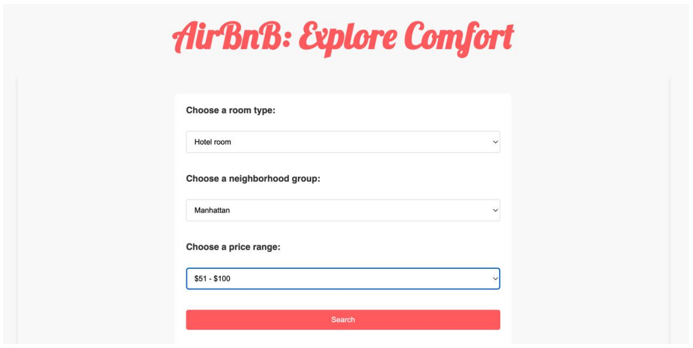
Choose a price range:
$0 - $50

Fig. 11. **Landing page where the user has to select the Room Type and Neighbourhood. Room Type = Entire home/Apt, Neighbourhood = Queens, Price Range = 0-50**



host_name	room_type	bedrooms	beds	bath	price	availability	location
Murah	Entire home/apt	1	1	1.0	120	0	Astoria, Queens
Tain	Entire home/apt	1	3	1.0	116	365	Sunnyside, Queens
Sulman	Entire home/apt	1	3	1.0	115	365	Jackson Heights, Queens
Winston	Entire home/apt	2	3	1.0	146	0	Corona, Queens
Clara	Entire home/apt	1	1	1.0	110	64	Woodhaven, Queens
Angel	Entire home/apt	3	5	1.0	120	339	Far Rockaway, Queens
Yelena	Entire home/apt	2	3	1.0	131	310	Forest Hills, Queens
Alvaro	Entire home/apt	2	2	1.0	113	360	East Elmhurst, Queens
Mizanur	Entire home/apt	2	2	2.0	150	230	Woodside, Queens
Costa	Entire home/apt	2	4	1.0	101	89	Sunnyside, Queens
Jenny	Entire home/apt	2	4	1.0	128	91	Ditmars Steinway, Queens
Jamal	Entire home/apt	1	1	1.0	110	89	Long Island City, Queens
Patricia	Entire home/apt	2	4	1.0	125	96	Jackson Heights, Queens
Clara	Entire home/apt	1	2	1.0	101	352	Astoria, Queens

Fig. 12. **Available results based on the parameter selection providing Host Name, Room Type, bedrooms, Beds, Bath, Price, Availability, and the Location**

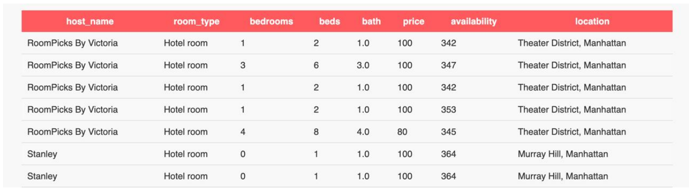

AirBnB: Explore Comfort

Choose a room type: Hotel room

Choose a neighborhood group: Manhattan

Choose a price range: $51 - $100

Fig. 13. **Landing page where the user has to select the Room Type and Neighbourhood. Room Type = Hotel Room, Neighbourhood = Manhattan, Price Range = 51-100**



host_name	room_type	bedrooms	beds	bath	price	availability	location
RoomPicks By Victoria	Hotel room	1	2	1.0	100	342	Theater District, Manhattan
RoomPicks By Victoria	Hotel room	3	6	3.0	100	347	Theater District, Manhattan
RoomPicks By Victoria	Hotel room	1	2	1.0	100	342	Theater District, Manhattan
RoomPicks By Victoria	Hotel room	1	2	1.0	100	353	Theater District, Manhattan
RoomPicks By Victoria	Hotel room	4	8	4.0	80	345	Theater District, Manhattan
Stanley	Hotel room	0	1	1.0	100	364	Murray Hill, Manhattan
Stanley	Hotel room	0	1	1.0	100	364	Murray Hill, Manhattan

Fig. 14. **Available results based on the parameter selection providing Host Name, Room Type, bedrooms, Beds, Bath, Price, Availability, and the Location**

X. Results

Our project meticulously refined the raw database of Airbnb listings through structured segmentation, resulting in the creation of distinct tables tailored to specific facets of listing management and analysis. The transformed database architecture, comprising Listing, Host, Availability, Ratings, Reviews, Location, Price, and Room tables, offers enhanced organization and accessibility for stakeholders. By capturing essential listing details, host profiles, booking management data, guest feedback, spatial analysis, and room configurations, the structured database facilitates in-depth analysis and informed decision-making within the dynamic Airbnb ecosystem. Utilization of PGAdmin as the primary PostgreSQL database management tool facilitated seamless interaction, enabling efficient execution of SQL queries and optimization of database performance. The implementation of indexing techniques further optimized runtime and data retrieval efficiency, empowering stakeholders to maximize returns and capitalize on market opportunities efficiently.

XI. Conclusion

Our project has successfully transformed the raw Airbnb listing database into a structured and analytically rich repository, enabling stakeholders to efficiently manage and analyze listing data in New York City. The development of a web portal further enhances accessibility, allowing users to easily search and filter listings based on location and room type. By implementing indexing techniques and leveraging PGAdmin for database management, we have optimized runtime and data retrieval efficiency, empowering stakeholders to make informed decisions and maximize returns in the dynamic Airbnb market. Overall, our project represents a significant advancement in the management and analysis of Airbnb listing data, offering valuable insights and facilitating seamless interaction for users.

References

[1] https://airbtics.com/
[2] https://www.kaggle.com/datasets/vrindakallu/new-york-dataset
[3] https://medium.com/@aliaformo/
[4] https://medium.com/towards-data-engineering/
[5] https://www.w3schools.com/
[6] https://www.sqlshack.com/