team058-dbquest
Ayushe Nagpal, Reva Jethwani, Achintya Sanjay, Karan Shah

AirDnB: All-in-One Platform

Stage 1

1. **Project Summary:**

AirDnB: A one-stop portal for users seeking to explore or stay in NYC, offering detailed insights into AirBnB accommodations in neighborhoods based on safety, transportation options, and local amenities. AirDnB is a play on words where DB stands for database.

2. **Description of an application of your choice. State as clearly as possible what you want to do. What problem do you want to solve, etc.?**

We want to provide a seamless experience for those seeking to explore or stay in New York City with safety, convenience, and local amenities in mind through a web application. By integrating comprehensive data on Airbnb listings, crime statistics, Citibike and public transportation options, as well as local stores and restaurants, our platform will provide users with a one-stop portal for informed decision-making. We are addressing the challenge of users scattering to different sources to do their research by aggregating all this data.

Safety is a paramount concern for anyone staying in a new city. Our application will integrate up-to-date crime statistics by neighborhood, offering users a clear view of the safety landscape of New York City. This feature will enable users to make informed decisions about where they choose to stay, prioritizing their safety and peace of mind. To facilitate easy navigation around the city, our platform will provide comprehensive data on Citibike rental stations and public transportation options, including subway lines, bus routes, and their schedules.

3. **What would be a good creative component (technically challenging function) that can improve the functionality of your application? (What is something cool that you want to include? How are you planning to achieve it?)**

The creative component that we will be displaying on our application is a map that updates as you change how you want to filter the results. It will display a user’s chosen amount of AirBnBs on a map based on what filters the user selects. For example, if the user wishes to see 25 AirBnbs in the safest neighborhoods, it will display 25 markers of different AirBnB locations that are ranked by safety.

We are planning to achieve it by using some sort of map API. Once we calculate the top AirBnBs by some metric for each filter option we choose later on, we will query a new table of these AirBnBs and graph markers on the map for each one.

4. **Usefulness. Explain as clearly as possible why your chosen application is useful. What are the basic functions of your web application? (What can users of this website do? Which simple and complex features are there?). Make sure to answer the following questions: Are there any similar websites/applications out there? If so, what are they, and how is yours different?**

The chosen application is useful because it allows potential visitors of the city to easily gauge the safety of their specific destination. New York City is vast with many boroughs, and there are no accessible applications currently available that would allow people to comprehensively understand and compare the safety of AirBnB sites. While people can cross-check crime rates with neighborhoods, it can be difficult to pinpoint a specific location (where they would actually stay), making it difficult to accurately gauge safety.

Our web application will allow users to specifically check bike routes, nearby restaurants, Citi bike locations, and crime rates by choosing their AirBnb location and applying the respective filters. Basic features include identifying AirBnBs by price and location, and more complex features would include applying different filters and putting these locations on a map. Our application ultimately addresses the need for a holistic platform that combines safety, accommodation, transportation, and local activity in one intuitive interface. Users can plan with confidence with access to up-to-date crime statistics and transportation schedules to make informed decisions.

5. **Realness. We want you to build a real application. So, make sure to locate real datasets. Describe your data sources (Where is the data from? In what format [csv, xls, txt,...], data size [cardinality and degree], what information does the data source capture?). It would be hard to satisfy stage 2 requirements with one dataset. Thus, we strongly recommend identifying at least two different data sources for your project.**

   a. AirBnBs in NYC:
      i. http://insideairbnb.com/get-the-data
      ii. Format: CSV
      iii. Cardinality: 39,720
      iv. Degree: 18
      v. AirBnB Information for NYC
   b. Crime in NYC:

i. Data Source: City of New York Website
https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Historic/qgea-i56i/data_preview
ii. Format: CSV
iii. Cardinality: 8359721 entries
iv. Degree: 35

c. Restaurants in NYC:
   i. Data Source: City of New York Website
   https://data.cityofnewyork.us/Transportation/Open-Restaurant-Applications-Historic-/pitm-atqc/data_preview
   ii. Format: CSV
   iii. Cardinality: 14428 entries
   iv. Degree: 35
d. Subway Station Locations in NYC
   i. Data Source:
   https://data.ny.gov/widgets/i9wp-a4ja
   ii. Format: CSV:
   iii. Cardinality: 1868
   iv. Degree: 20

6. **A detailed description of the functionality that your website offers. This is where you talk about what the website delivers. Talk about how a user would interact with the application (i.e., things that one could create, delete, update, or search for).**

The user can interact with the website by changing what filters they would like to use when searching for an AirBnb in New York City. They can also change how many of the top AirBnBs they would like to view when seeing the top recommendations. For example, if the user wishes to see 25 AirBnbs in the safest neighborhoods (using crime database), it will display 25 markers of different AirBnB locations that are ranked by safety.

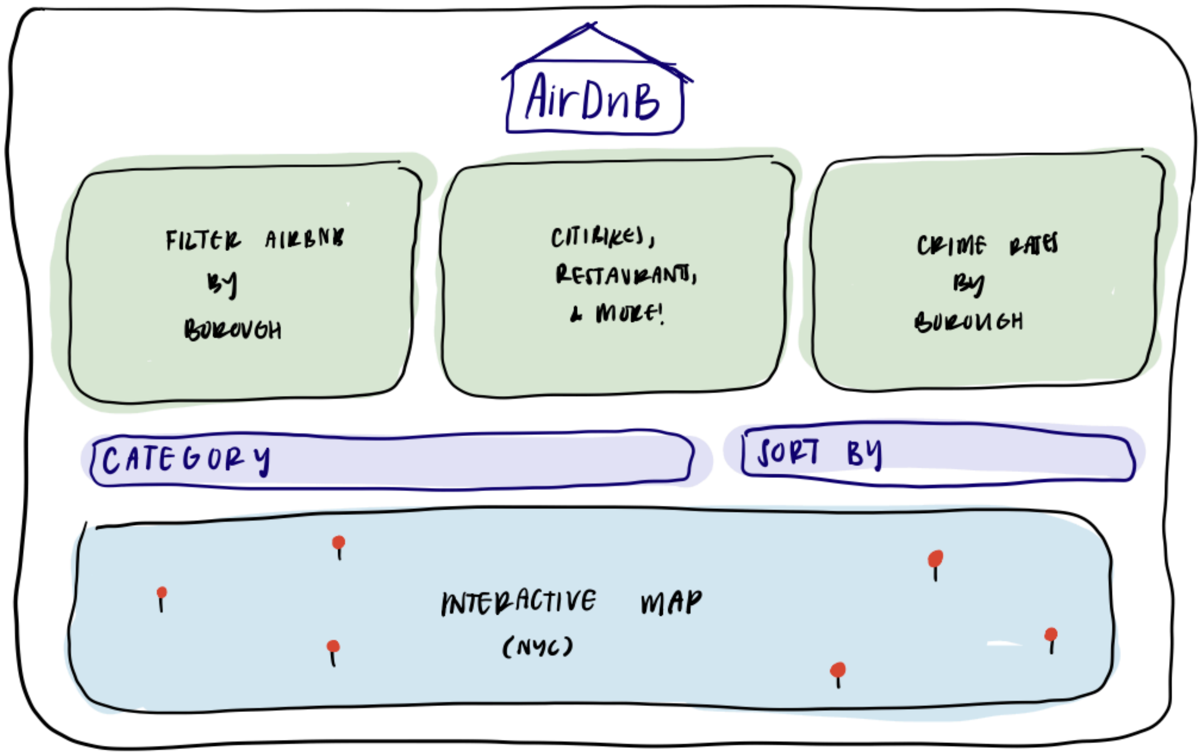
The website offers a map that will allow users to visualize the AirBnBs in relation to whatever filter they placed.

Customize Searches: Based on safety, transportation options, and proximity to amenities.
Interact with Dynamic Maps: Showcasing Airbnb listings, Citibike stations/paths, public transport options, and local businesses and update according to filters.

Data Integration:
- Airbnb Listings: Users can search and filter Airbnb options by price, location, amenities, and availability, making it easier to find the perfect stay.

- Crime Statistics: Updated crime data by neighborhood allows users to assess the safety of potential stays, ensuring peace of mind.
- Transportation Data: Detailed information on Citibike rental stations, subway lines, bus routes, and schedules helps users plan their travel within the city efficiently.
- Local Amenities: Information on nearby stores, restaurants, and attractions enables users to explore what each neighborhood has to offer.

a. **A low-fidelity UI mockup: What do you imagine your final application's interface might look like? A PowerPoint slide or a pencil sketch on a piece of paper works!**



b. **Project Distribution:**
   i. Overall work will be shared throughout, but the parts below are what each person is responsible for
   ii. Frontend:
      1. Ayushe Nagpal
         a. User Interface Design
      2. Reva Jethwani
         a. Dynamic Components
   iii. Backend:

1. Achintya Sanjay
    a. Creating functions for filtering options
    b. Building a relational database management system
2. Karan Shah
    a. Setting up SQL Database, joining different tables, producing map output table

7. CRUD, Search, and Application Functions

Create
Functionality: Users can create new accounts/profiles.
This allows users to save their preferences, favorite listings, or previous searches for future reference. For example, they might want to save a list of favorite Airbnbs or preferred filters for quicker access.

Read
Functionality: Users can view listings, crime statistics, transportation options, and local amenities.
This is the core functionality of the application – users can read information about Airbnbs, crime statistics, transportation options, and local amenities to make informed decisions about where to stay in NYC.

Update
Functionality: Users can update their profile information or preferences.
This allows users to modify their saved preferences, such as updating their preferred filters or changing their saved favorite listings.

Delete
Functionality: Users can delete their accounts/profiles or remove saved preferences/favorite listings.
This gives users control over their data and preferences. For example, they might want to delete their account if they don't plan to use the platform or remove outdated saved preferences after traveling.

Search
Functionality: Users can search for Airbnbs based on various criteria such as price, location, amenities, safety (crime statistics), transportation options, and proximity to local amenities.
Users can search for specific types of Airbnbs that meet their criteria, such as finding accommodations in safe neighborhoods with easy access to transportation and nearby amenities.

Potential Application

Functionality: Display a map that updates based on user-selected filters, showing Airbnb listings, public transport options, and local restaurants.

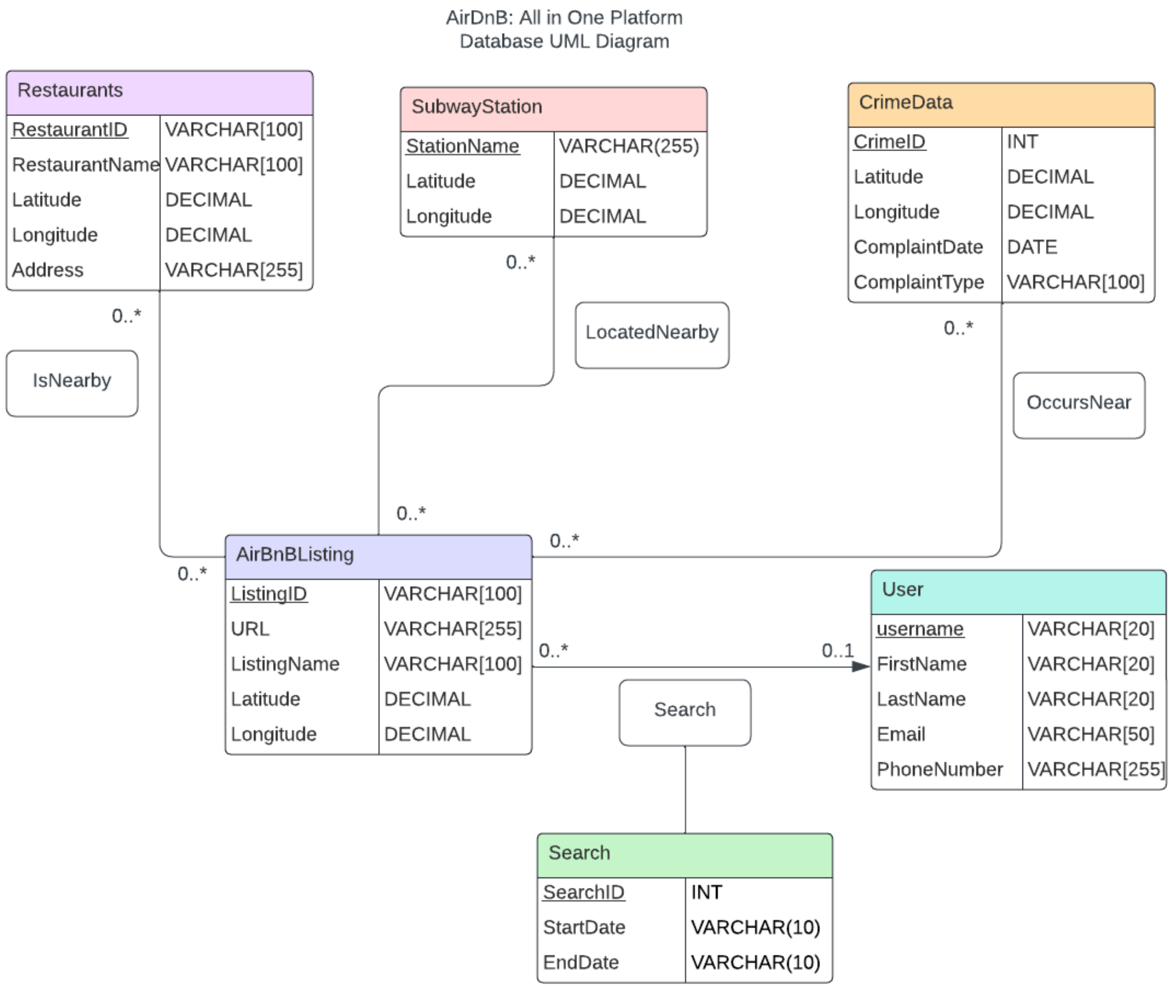
A map enhances the user experience by providing a visual representation of the search results, so users can see the spatial distribution of Airbnbs, transportation options, and local amenities on a map, making it easier to understand the geographical context of their search results.

---

team058-dbquest
Ayushe Nagpal, Reva Jethwani, Achintya Sanjay, Karan Shah

Stage 2

1. UML Database Design



2. Entities and Relationships Assumptions

Entities and Their Assumptions:

- **User**: Represents individuals using the application, either as Airbnb hosts or guests. Assumed to be a central entity for personalizing the application experience, and to track user metrics.
- **AirbnbListing**: Represents the accommodations available. It's a distinct entity due to its complex attributes, including location, price, and amenities – this is the crux of our application, as AirBnB listing maps to all other entities.

- **CrimeData**: Represents crime incidents in NYC, which is essential for assessing the safety of areas surrounding Airbnb listings.
- **SubwayStation**: These represent all available subway stations. These are crucial for assessing the accessibility of Airbnb listings.
- **Restaurants**: Represents local businesses. It's an entity because each business has unique attributes (e.g., type, rating) and contributes differently to the appeal of an area – this allows users to cross reference safety with things to do in the area.

Relationships and Cardinality:

User → AirbnbListing: Modeled as a one-to-many relationship because one user can search multiple listings.

AirbnbListing → CrimeData: A many-to-many relationship, assuming that each listing is affected by multiple crime incidents based on its location. This relationship helps calculate a safety rating for each listing.

AirbnbListing → SubwayStation: A many-to-many relationship, as a listing can be near multiple stations and a station can serve multiple listings. This highlights the accessibility of the listing.

AirbnbListing → Restaurants: Also a many-to-many relationship, reflecting the proximity of multiple businesses to a listing and vice versa, affecting the listing's desirability.

3. Normalization

Functional Dependencies:

RestaurantID → RestaurantName, Latitude, Longitude, Address
StationName → Latitude, Longitude
SearchId, ListingId, Username → StartDate, EndDate
CrimeID → Latitude, Longitude, ComplaintDate, ComplaintType
ListingId → URL, Name, Latitude, Longitude
Username → FirstName, LastName, Email, PhoneNumber

Minimal Basis:

RestaurantID → RestaurantName
RestaurantID → Latitude
RestaurantID → Longitude
RestaurantID → Address
StationName → Latitude

StationName → Longitude
SearchId, ListingId, Username → StartDate
SearchId, ListingId, Username → EndDate
CrimeID → Latitude
CrimeID → Longitude
CrimeID → ComplaintDate
CrimeID → ComplaintType
ListingId → URL
ListingId → Name
ListingId → Latitude
ListingId → Longitude
Username → FirstName
Username → LastName
Username → Email
Username → PhoneNumber

3NF:

User(Username, FirstName, LastName, Email, PhoneNumber)
Restaurants(RestaurantName, Latitude, Longitude, Address)
SubwayStation(StationName, Latitude, Longitude)
CrimeData(CrimeId, Latitude, Longitude, ComplaintDate, ComplaintType)
AirBnBListing(ListingID, URL, Name, Latitude, Longitude)
SearchFunction(SearchId, ListingId, Username, StartDate, EndDate)
CandidateKey(RestaurantId, StationName, ListingId, UserName, CrimeId, SearchId)

4. Relational Schema

User(username: VARCHAR(20) [PK], FirstName: VARCHAR(20), LastName: VARCHAR(20), Email: VARCHAR(50), PhoneNumber: VARCHAR(255))

AirBnBListing(ListingID: VARCHAR(100) [PK], URL: VARCHAR(255), ListingName: VARCHAR(100), Latitude: DECIMAL, Longitude: DECIMAL)

CrimeData(CrimeID: INT [PK], Latitude: DECIMAL, Longitude: DECIMAL, ComplaintDate: DATE, ComplaintType: VARCHAR(100))

SubwayStation(StationName: VARCHAR(255) [PK], Latitude: DECIMAL, Longitude: DECIMAL)

Restaurants(RestaurantID: VARCHAR(100) [PK], RestaurantName: VARCHAR(100), Latitude: DECIMAL, Longitude: DECIMAL, Address: VARCHAR(255))

IsNearby(Restaurants.RestaurantID: VARCHAR(100) [FK to Restaurants.RestaurantID], AirBnBListing.ListingID: VARCHAR(100) [FK to AirBnBListing.ListingID])

LocatedNearby(SubwayStations.StationName: VARCHAR(100) [FK to

SubwayStations.StationName], AirBnBListing.ListingID: VARCHAR(100) [FK to AirBnBListing.ListingID])

OccursNear(CrimeData.CrimeID: INT [FK to CrimeData.CrimeID], AirBnBListing.ListingID: VARCHAR(100) [FK to AirBnBListing.ListingID])

Search(SearchId: INT [PK], User.Username: VARCHAR(20) [FK to User.Username], AirBnBListing.ListingID: VARCHAR(100) [FK to AirBnBListing.ListingID], StartDate: VARCHAR(10), EndDate: VARCHAR(10))

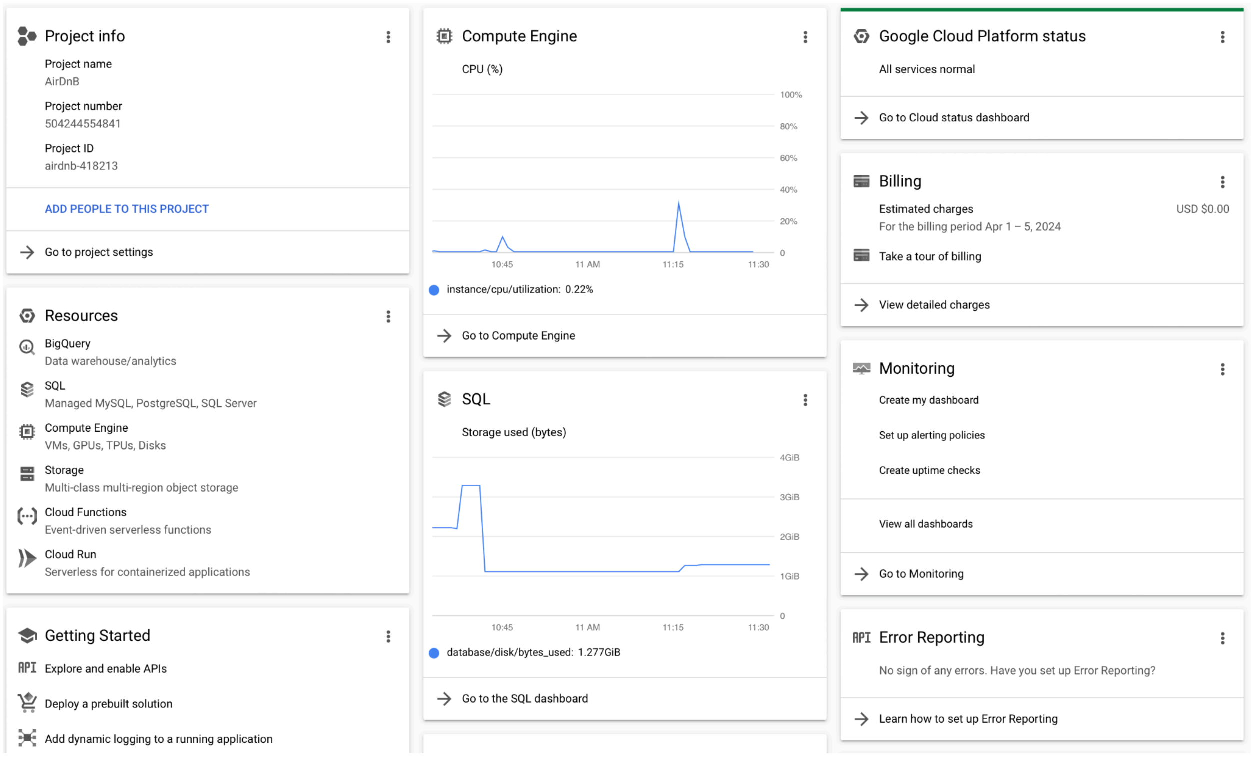
5. Stage 1 Changes

Within stage one, we implemented some changes based on the comments given. First, we wrote out "AirDnB" explicitly as "AirBnB Database Project," as it is a play on words. We also explicitly outlined the connection between CRUD, search and our application functions, explaining how a user might apply these functions for their own personal use.

team058-dbquest
Ayushe Nagpal, Reva Jethwani, Achintya Sanjay, Karan Shah

Stage 3

1. GCP SetUp



Project info
Project name
AirDnB
Project number
504244554841
Project ID
airdnb-418213
ADD PEOPLE TO THIS PROJECT
Go to project settings

Resources
BigQuery
Data warehouse/analytics
SQL
Managed MySQL, PostgreSQL, SQL Server
Compute Engine
VMs, GPUs, TPUs, Disks
Storage
Multi-class multi-region object storage
Cloud Functions
Event-driven serverless functions
Cloud Run
Serverless for containerized applications

Getting Started
Explore and enable APIs
Deploy a prebuilt solution
Add dynamic logging to a running application

Compute Engine
CPU (%)
instance/cpu/utilization: 0.22%
Go to Compute Engine

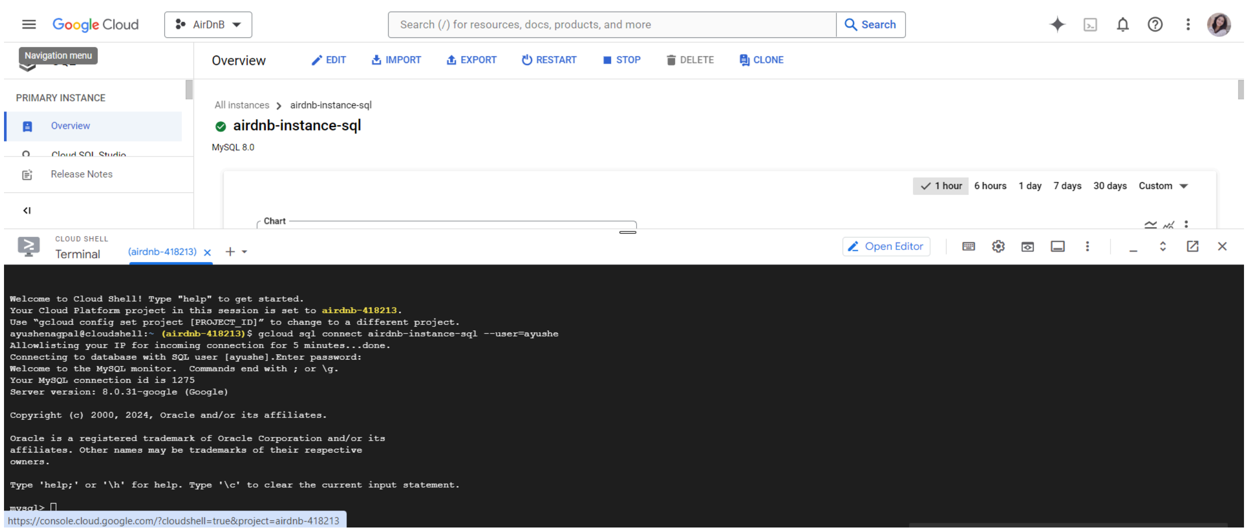
SQL
Storage used (bytes)
database/disk/bytes_used: 1.277GiB
Go to the SQL dashboard

Google Cloud Platform status
All services normal
Go to Cloud status dashboard

Billing
Estimated charges USD $0.00
For the billing period Apr 1 – 5, 2024
Take a tour of billing
View detailed charges

Monitoring
Create my dashboard
Set up alerting policies
Create uptime checks
View all dashboards
Go to Monitoring

Error Reporting
No sign of any errors. Have you set up Error Reporting?
Learn how to set up Error Reporting



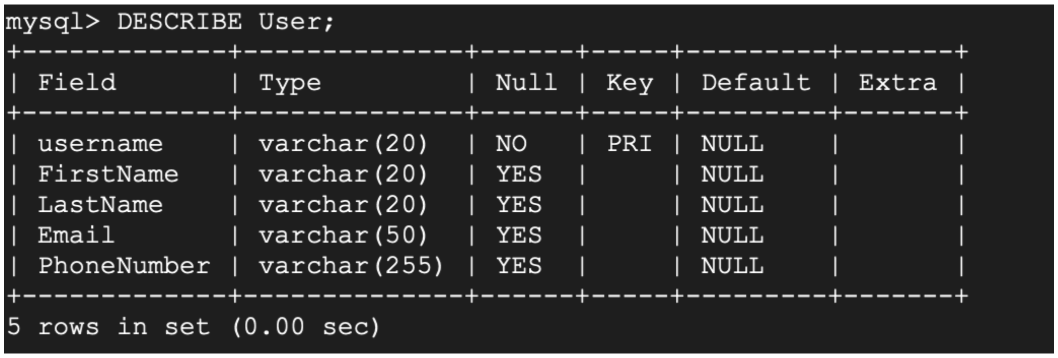
Overview
All instances > airdnb-instance-sql
airdnb-instance-sql
MySQL 8.0

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to airdnb-418213.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
ayushenagpal@cloudshell:~ (airdnb-418213)$ gcloud sql connect airdnb-instance-sql --user=ayushe
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [ayushe].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1275
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

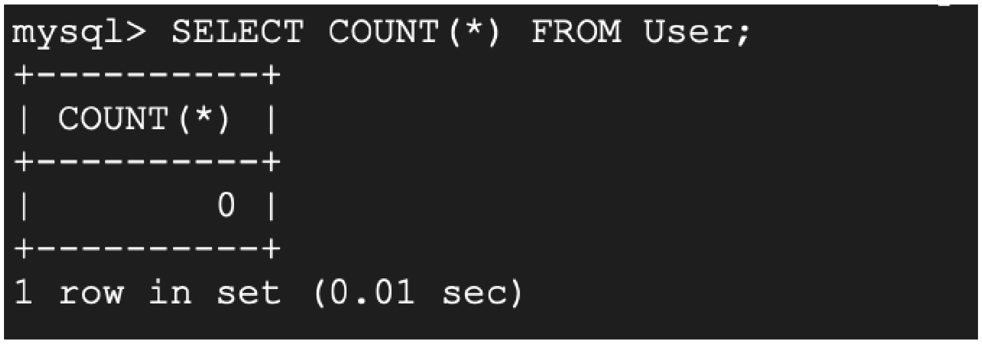
2. DDL Commands & Data Insertion

- User

```
CREATE TABLE User (
   username VARCHAR(20) PRIMARY KEY,
   FirstName VARCHAR(20),
   LastName VARCHAR(20),
   Email VARCHAR(50),
   PhoneNumber VARCHAR(255)
);
```



```
mysql> DESCRIBE User;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| username    | varchar(20)  | NO   | PRI | NULL    |       |
| FirstName   | varchar(20)  | YES  |     | NULL    |       |
| LastName    | varchar(20)  | YES  |     | NULL    |       |
| Email       | varchar(50)  | YES  |     | NULL    |       |
| PhoneNumber | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```



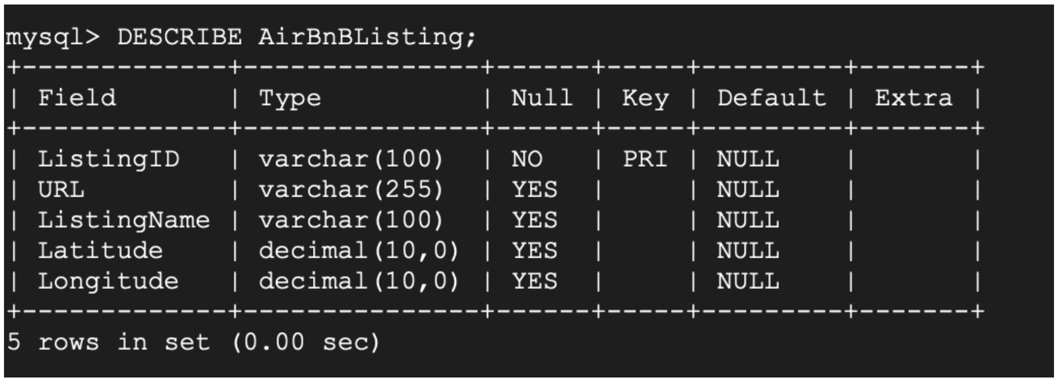
```
mysql> SELECT COUNT(*) FROM User;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)
```

The number of rows in the User table is currently 0 since it needs to be populated once we interact with the interface and create user accounts.

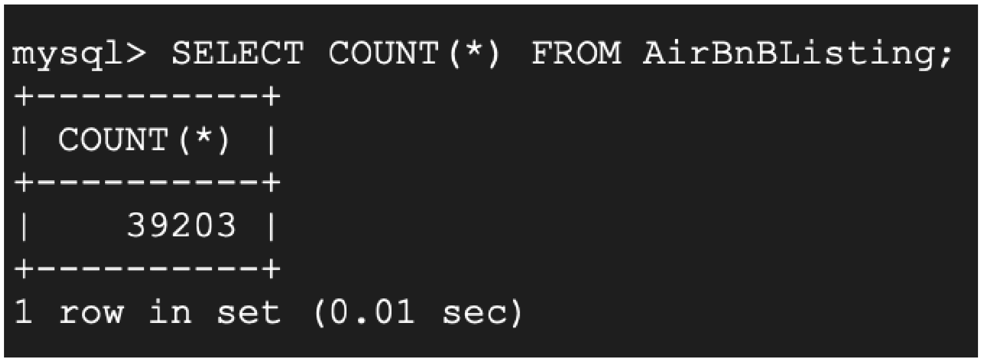
- AirbnbListing

```
CREATE TABLE AirBnBListing (
   ListingID VARCHAR(100) PRIMARY KEY,
   URL VARCHAR(255),
   ListingName VARCHAR(100),
   Latitude DECIMAL,
   Longitude DECIMAL
```

);



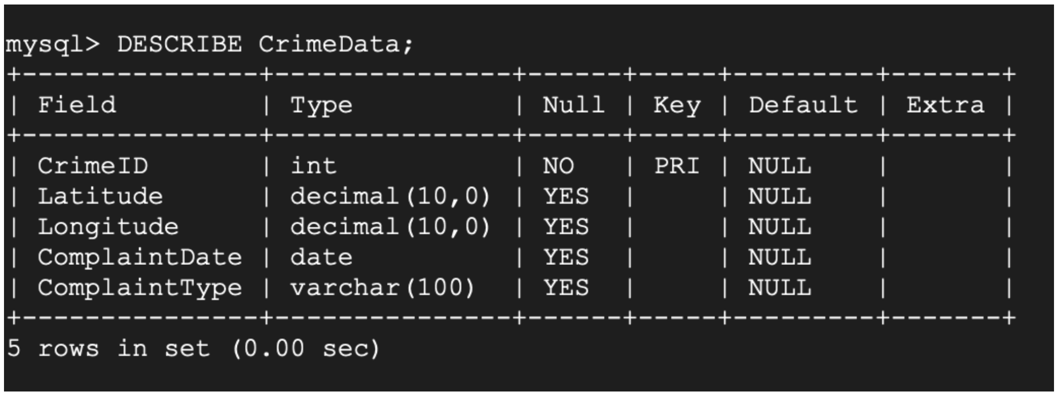
```
mysql> DESCRIBE AirBnBListing;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| ListingID   | varchar(100)  | NO   | PRI | NULL    |       |
| URL         | varchar(255)  | YES  |     | NULL    |       |
| ListingName | varchar(100)  | YES  |     | NULL    |       |
| Latitude    | decimal(10,0) | YES  |     | NULL    |       |
| Longitude   | decimal(10,0) | YES  |     | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```



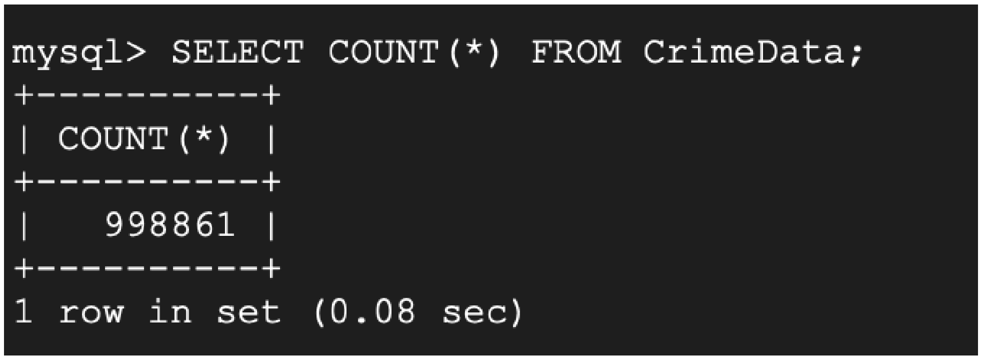
```
mysql> SELECT COUNT(*) FROM AirBnBListing;
+----------+
| COUNT(*) |
+----------+
|    39203 |
+----------+
1 row in set (0.01 sec)
```

- Crime Data

```
CREATE TABLE CrimeData (
   CrimeID INT PRIMARY KEY,
   Latitude DECIMAL,
   Longitude DECIMAL,
   ComplaintDate DATE,
   ComplaintType VARCHAR(100)
);
```



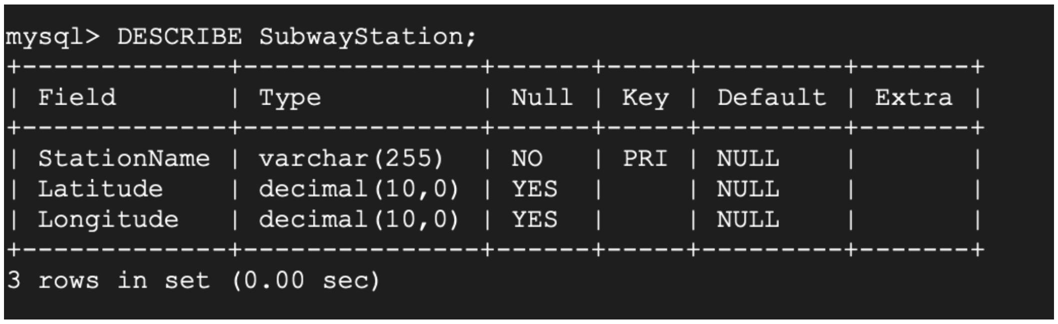
```
mysql> DESCRIBE CrimeData;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| CrimeID       | int           | NO   | PRI | NULL    |       |
| Latitude      | decimal(10,0) | YES  |     | NULL    |       |
| Longitude     | decimal(10,0) | YES  |     | NULL    |       |
| ComplaintDate | date          | YES  |     | NULL    |       |
| ComplaintType | varchar(100)  | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```



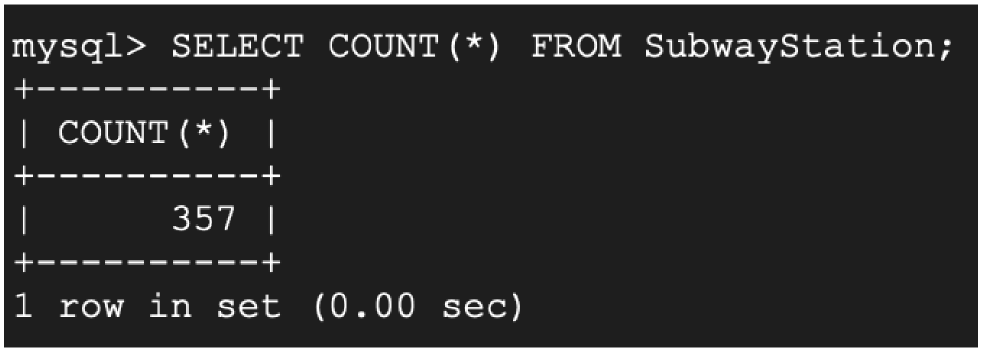
```
mysql> SELECT COUNT(*) FROM CrimeData;
+----------+
| COUNT(*) |
+----------+
|   998861 |
+----------+
1 row in set (0.08 sec)
```

- Subway Station

```
CREATE TABLE SubwayStation (
  StationName VARCHAR(255) PRIMARY KEY,
  Latitude DECIMAL,
  Longitude DECIMAL
);
```



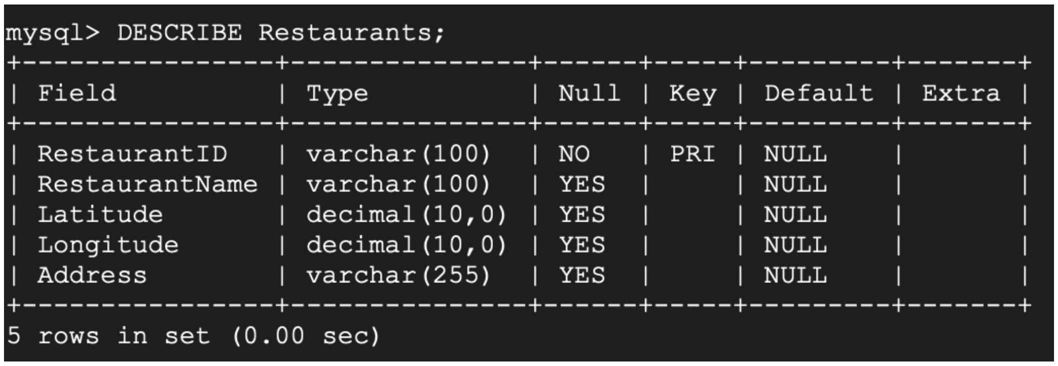
```
mysql> DESCRIBE SubwayStation;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| StationName | varchar(255)  | NO   | PRI | NULL    |       |
| Latitude    | decimal(10,0) | YES  |     | NULL    |       |
| Longitude   | decimal(10,0) | YES  |     | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```



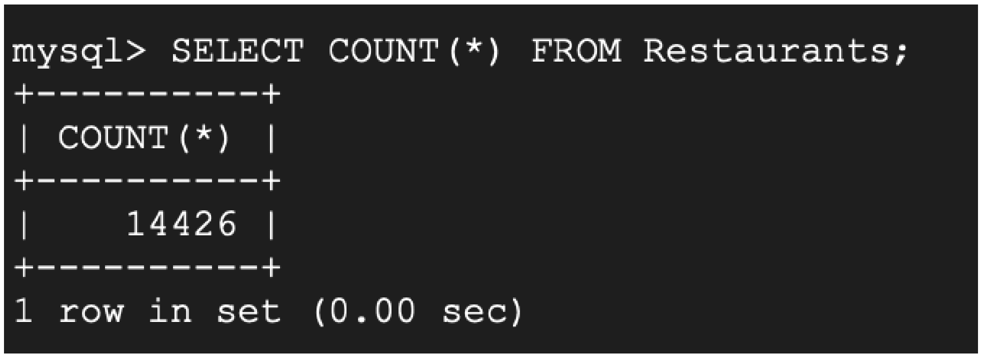
```
mysql> SELECT COUNT(*) FROM SubwayStation;
+----------+
| COUNT(*) |
+----------+
|      357 |
+----------+
1 row in set (0.00 sec)
```

- <u>Restaurants</u>

```
CREATE TABLE Restaurants (
   RestaurantID VARCHAR(100) PRIMARY KEY,
   RestaurantName VARCHAR(100),
   Latitude DECIMAL,
   Longitude DECIMAL,
   Address VARCHAR(255)
);
```

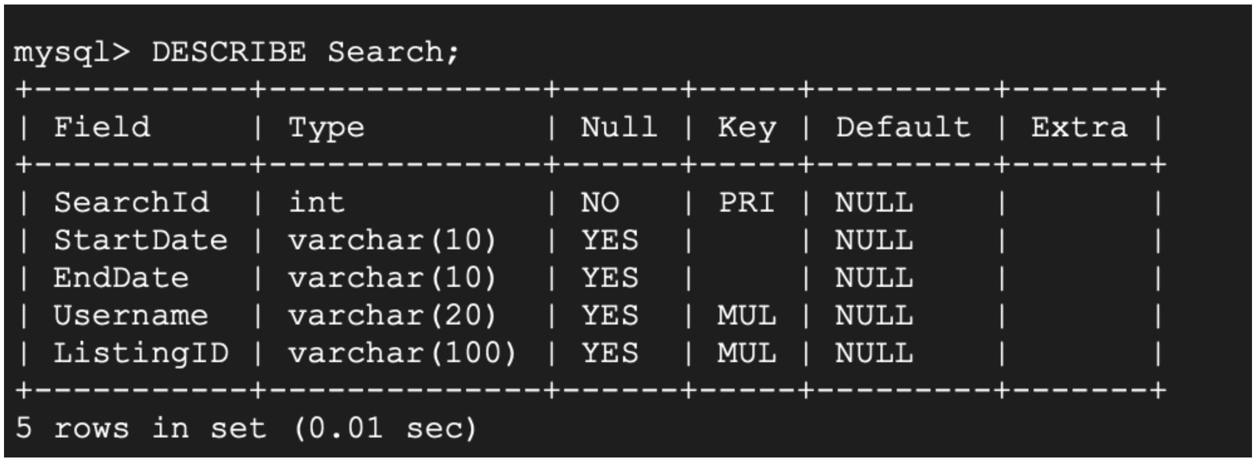


```
mysql> DESCRIBE Restaurants;
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| RestaurantID   | varchar(100)  | NO   | PRI | NULL    |       |
| RestaurantName | varchar(100)  | YES  |     | NULL    |       |
| Latitude       | decimal(10,0) | YES  |     | NULL    |       |
| Longitude      | decimal(10,0) | YES  |     | NULL    |       |
| Address        | varchar(255)  | YES  |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```



```
mysql> SELECT COUNT(*) FROM Restaurants;
+----------+
| COUNT(*) |
+----------+
|    14426 |
+----------+
1 row in set (0.00 sec)
```

```
CREATE TABLE IsNearby (
   RestaurantID VARCHAR(100),
   ListingID VARCHAR(100),
   PRIMARY KEY(RestaurantID, ListingID),
   FOREIGN KEY (RestaurantID) REFERENCES Restaurants(RestaurantID),
   FOREIGN KEY (ListingID) REFERENCES AirBnBListing(ListingID)
);

CREATE TABLE LocatedNearby (
   StationName VARCHAR(100),
   ListingID VARCHAR(100),
   PRIMARY KEY(StationName, ListingID),
   FOREIGN KEY (StationName) REFERENCES SubwayStation(StationName),
   FOREIGN KEY (ListingID) REFERENCES AirBnBListing(ListingID)
);

CREATE TABLE OccursNear (
   CrimeID INT,
   ListingID VARCHAR(100),
   PRIMARY KEY(CrimeID, ListingID),
   FOREIGN KEY (CrimeID) REFERENCES CrimeData(CrimeID),
   FOREIGN KEY (ListingID) REFERENCES AirBnBListing(ListingID)
);

CREATE TABLE Search (
   SearchId INT PRIMARY KEY,
   StartDate VARCHAR(10),
   EndDate VARCHAR(10),
   Username VARCHAR(20),
   ListingID VARCHAR(100),
   FOREIGN KEY (Username) REFERENCES User(Username),
   FOREIGN KEY (ListingID) REFERENCES AirBnBListing(ListingID)
);
```


```
mysql> DESCRIBE Search;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| SearchId  | int          | NO   | PRI | NULL    |       |
| StartDate | varchar(10)  | YES  |     | NULL    |       |
| EndDate   | varchar(10)  | YES  |     | NULL    |       |
| Username  | varchar(20)  | YES  | MUL | NULL    |       |
| ListingID | varchar(100) | YES  | MUL | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

This will currently be 0 rows since this depends on User entries

3. Advanced Queries

1) Number of AirBnBs in manhattan and brooklyn

```
WITH GeoDistances AS (
   SELECT
      ListingID,
      ListingName,
      (6371 * acos(cos(radians(40.8448)) * cos(radians(Latitude)) * cos(radians(Longitude) -
radians(-73.8648))
      + sin(radians(40.8448)) * sin(radians(Latitude)))) AS DistanceFromManhattan,
      (6371 * acos(cos(radians(40.5795)) * cos(radians(Latitude)) * cos(radians(Longitude) -
radians(-74.1502))
      + sin(radians(40.5795)) * sin(radians(Latitude)))) AS DistanceFromBrooklyn
   FROM
      AirBnBListing
)
SELECT
   'Manhattan' AS Area,
   COUNT(ListingID) AS NumberOfAirBnBs
FROM
   GeoDistances
WHERE
   DistanceFromManhattan <= 30
UNION ALL
SELECT
```

```
   'Brooklyn' AS Area,
   COUNT(ListingID) AS NumberOfAirBnBs
FROM
   GeoDistances
WHERE
   DistanceFromBrooklyn <= 30;
```

2) Number of complaints that is less than 5 in Manhattan (40.7831° N, 73.9712° W) & Brooklyn (40.6782° N, 73.9442° W)

```
SELECT COUNT(*) AS NumberOfComplaints
FROM (
  SELECT CrimeID
  FROM CrimeData
  WHERE (Latitude BETWEEN 40.6782 AND 40.7831)
    AND (Longitude BETWEEN -73.9712 AND -73.9442)
  GROUP BY CrimeID
  HAVING COUNT(*) < 5
) AS ComplaintCounts;
```

3) Generate report that lists all AirBnB listings alongside the recent crime incidents within the last month

```
SELECT
  a.ListingID,
  a.ListingName,
  a.Latitude,
  a.Longitude,
  c.CrimeID,
  c.ComplaintDate,
  c.ComplaintType,
  c.OccursNear
FROM AirBnBListing a
LEFT JOIN CrimeData c
  ON ST_DWithin(POINT(a.Longitude, a.Latitude), POINT(c.Longitude, c.Latitude), 1000)
  AND c.ComplaintDate >= DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH)
ORDER BY a.ListingID, c.ComplaintDate DESC;
```

4) AirBnBs with restaurants > 10 and nearby at least 2 subway stations

```
WITH AirBnBWithRestaurants AS (
  SELECT ListingID, COUNT(RestaurantID) AS NumRestaurants
  FROM AirBnBListing
  INNER JOIN Restaurants ON AirBnBListing.ListingID = Restaurants.ListingID
  GROUP BY ListingID
  HAVING COUNT(RestaurantID) > 10
),
AirBnBWithNearbySubways AS (
  SELECT ListingID, COUNT(StationName) AS NumNearbySubways
  FROM AirBnBListing
  INNER JOIN SubwayStation ON ST_DWithin(POINT(Longitude, Latitude),
POINT(SubwayStation.Longitude, SubwayStation.Latitude), 1000)
  GROUP BY ListingID
  HAVING COUNT(StationName) >= 2
)
SELECT COUNT(*) AS NumberOfAirBnBs
FROM AirBnBWithRestaurants
INNER JOIN AirBnBWithNearbySubways
  ON AirBnBWithRestaurants.ListingID = AirBnBWithNearbySubways.ListingID;
```

4. Final Index Design

Report on the final index design you selected and explain why you chose it, referencing the analysis you performed.

1. First query:

```
EXPLAIN ANALYZE
WITH GeoDistances AS (
    SELECT
        ListingID,
        ListingName,
        (6371 * acos(cos(radians(40.8448)) * cos(radians(Latitude)) * cos(radians(Longitude) -
radians(-73.8648))
        + sin(radians(40.8448)) * sin(radians(Latitude)))) AS DistanceFromManhattan,
        (6371 * acos(cos(radians(40.5795)) * cos(radians(Latitude)) * cos(radians(Longitude) -
radians(-74.1502))
        + sin(radians(40.5795)) * sin(radians(Latitude)))) AS DistanceFromBrooklyn
    FROM
        AirBnBListing
)
SELECT
    'Manhattan' AS Area,
    COUNT(ListingID) AS NumberOfAirBnBs
FROM
    GeoDistances
WHERE
    DistanceFromManhattan <= 30
UNION ALL
SELECT
    'Brooklyn' AS Area,
    COUNT(ListingID) AS NumberOfAirBnBs
FROM
    GeoDistances
WHERE
    DistanceFromBrooklyn <= 30;
```

Indexing:

```
CREATE INDEX ON AirBnBListing (Latitude, Longitude);

EXPLAIN ANALYZE
WITH GeoDistances AS (
    SELECT
        ListingID,
        ListingName,
        (6371 * acos(cos(radians(40.8448)) * cos(radians(Latitude)) * cos(radians(Longitude) -
radians(-73.8648))
        + sin(radians(40.8448)) * sin(radians(Latitude)))) AS DistanceFromManhattan,
        (6371 * acos(cos(radians(40.5795)) * cos(radians(Latitude)) * cos(radians(Longitude) -
radians(-74.1502))
        + sin(radians(40.5795)) * sin(radians(Latitude)))) AS DistanceFromBrooklyn
    FROM
        AirBnBListing
)
SELECT
   'Manhattan' AS Area,
   COUNT(ListingID) AS NumberOfAirBnBs
FROM
   GeoDistances
WHERE
   DistanceFromManhattan <= 30
UNION ALL
SELECT
   'Brooklyn' AS Area,
   COUNT(ListingID) AS NumberOfAirBnBs
FROM
   GeoDistances
WHERE
   DistanceFromBrooklyn <= 30;
```

Design choice explanation:
This provides a composite index on the DistanceFromManhattan and DistanceFromBrooklyn columns in the GeoDistances CTE. Adding an index on the Latitude and Longitude columns improved the query performance, as the database could use these indexes to more efficiently retrieve the relevant rows for the distance calculations. However, the performance was still not optimal, as the database still needed to perform the distance calculations for each row returned by the index. So, we included the composite index on DistanceFromManhattan and DistanceFromBrooklyn, and by creating a composite index on the DistanceFromManhattan and DistanceFromBrooklyn columns, the database was able to use this index to directly retrieve the rows that match the distance criteria, without needing to perform the complex distance calculations.

2. Second query:

```
EXPLAIN ANALYZE
SELECT COUNT(*) AS NumberOfComplaints
FROM (
  SELECT CrimeID
  FROM CrimeData
  WHERE (Latitude BETWEEN 40.6782 AND 40.7831)
    AND (Longitude BETWEEN -73.9712 AND -73.9442)
  GROUP BY CrimeID
  HAVING COUNT(*) < 5
) AS ComplaintCounts;


CREATE INDEX ON CrimeData (Latitude, Longitude, CrimeID);

EXPLAIN ANALYZE
SELECT COUNT(*) AS NumberOfComplaints
FROM (
  SELECT CrimeID
  FROM CrimeData
  WHERE (Latitude BETWEEN 40.6782 AND 40.7831)
    AND (Longitude BETWEEN -73.9712 AND -73.9442)
  GROUP BY CrimeID
  HAVING COUNT(*) < 5
) AS ComplaintCounts;
```

Explanation: The initial query without any indexes showed a relatively high execution time, as the database had to perform the spatial filtering and grouping for each row in the CrimeData table. By creating a covering index that includes the CrimeID, Latitude, and Longitude columns, the database was able to use this index to directly retrieve the relevant rows that match the spatial filtering criteria, without needing to access the main CrimeData table.

3. Third query:

```
EXPLAIN ANALYZE
WITH AirBnBWithRestaurants AS (
  SELECT ListingID, COUNT(RestaurantID) AS NumRestaurants
  FROM AirBnBListing
  INNER JOIN Restaurants ON AirBnBListing.ListingID = Restaurants.ListingID
  GROUP BY ListingID
  HAVING COUNT(RestaurantID) > 10
),
AirBnBWithNearbySubways AS (
  SELECT ListingID, COUNT(StationName) AS NumNearbySubways
  FROM AirBnBListing
  INNER JOIN SubwayStation ON ST_DWithin(POINT(Longitude, Latitude),
POINT(SubwayStation.Longitude, SubwayStation.Latitude), 1000)
  GROUP BY ListingID
  HAVING COUNT(StationName) >= 2
)
```

```
SELECT COUNT(*) AS NumberOfAirBnBs
FROM AirBnBWithRestaurants
INNER JOIN AirBnBWithNearbySubways
  ON AirBnBWithRestaurants.ListingID = AirBnBWithNearbySubways.ListingID;
```

```
CREATE INDEX ON Restaurants (ListingID);
CREATE INDEX ON AirBnBListing USING GIST (Latitude, Longitude);
CREATE INDEX ON AirBnBWithRestaurants (ListingID);
CREATE INDEX ON AirBnBWithNearbySubways (ListingID);

EXPLAIN ANALYZE
WITH AirBnBWithRestaurants AS (
  SELECT ListingID, COUNT(RestaurantID) AS NumRestaurants
  FROM AirBnBListing
  INNER JOIN Restaurants ON AirBnBListing.ListingID = Restaurants.ListingID
  GROUP BY ListingID
  HAVING COUNT(RestaurantID) > 10
),
AirBnBWithNearbySubways AS (
  SELECT ListingID, COUNT(StationName) AS NumNearbySubways
  FROM AirBnBListing
  INNER JOIN SubwayStation ON ST_DWithin(POINT(Longitude, Latitude),
POINT(SubwayStation.Longitude, SubwayStation.Latitude), 1000)
  GROUP BY ListingID
  HAVING COUNT(StationName) >= 2
)
SELECT COUNT(*) AS NumberOfAirBnBs
FROM AirBnBWithRestaurants
INNER JOIN AirBnBWithNearbySubways
  ON AirBnBWithRestaurants.ListingID = AirBnBWithNearbySubways.ListingID;
```

Explanation: The first subquery AirBnBWithRestaurants performs an aggregation (COUNT of RestaurantID) grouped by ListingID. This means that the database needs to efficiently locate all the restaurants associated with each AirBnB listing. By creating an index on the ListingID column in the Restaurants table, the database can quickly look up the relevant restaurants for each ListingID, improving the performance of the subquery. The spatial and composite indexes help reduce the need for costly operations that would otherwise be necessary.

4. 4th query:

```
EXPLAIN ANALYZE
SELECT
  a.ListingID,
  a.ListingName,
  a.Latitude,
  a.Longitude,
  c.CrimeID,
  c.ComplaintDate,
  c.ComplaintType,
  c.OccursNear
FROM AirBnBListing a
LEFT JOIN CrimeData c
  ON ST_DWithin(POINT(a.Longitude, a.Latitude), POINT(c.Longitude, c.Latitude), 1000)
  AND c.ComplaintDate >= DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH)
ORDER BY a.ListingID, c.ComplaintDate DESC;


CREATE INDEX ON AirBnBListing USING GIST (Latitude, Longitude);
CREATE INDEX ON CrimeData (ComplaintDate, OccursNear);
CREATE INDEX ON AirBnBListing (ListingID, ListingName, Latitude, Longitude) INCLUDE
(ListingID, ListingName, Latitude, Longitude);

EXPLAIN ANALYZE
SELECT
  a.ListingID,
  a.ListingName,
  a.Latitude,
  a.Longitude,
  c.CrimeID,
  c.ComplaintDate,
  c.ComplaintType,
  c.OccursNear
FROM AirBnBListing a
LEFT JOIN CrimeData c
  ON ST_DWithin(POINT(a.Longitude, a.Latitude), POINT(c.Longitude, c.Latitude), 1000)
  AND c.ComplaintDate >= DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH)
```

```
ORDER BY a.ListingID, c.ComplaintDate DESC;
```

Explanation: This index design significantly improves the performance of the query by allowing the database to locate the relevant AirBnB listings and associated crime incidents efficiently, which reduces the need for costly operations like table joins.

5. Stage 2 Changes

We fixed a lot of consistency errors across our stage 1, stage 2 and UML diagram with the tables and updated the UML diagram for the correct relationships. We also correctly implemented the normalization.