Milestone 5 - SafeAirbnbLA Final Report

Demo Link: **CIS550-Project.mov**

1. Title

SafeAirbnbLA

2. Introduction - Project Goals / Target Problem

Our tool provides a convenient way to view Airbnb listings alongside nearby crime rate data. By integrating Airbnb's listing information with crime data from reliable sources, users can make more informed decisions when choosing accommodations.

Key Functionalities:

- View Airbnb listings with detailed descriptions and ratings.
- Explore a map displaying the locations of Airbnb listings and nearby crime incidents.
- Access crime rate statistics of specific areas.
- Customize search filters to narrow down listings based on preferred crime rate levels. With our tool, users can easily assess the safety of an area before booking an Airbnb stay, empowering them to prioritize their peace of mind and security while traveling.

Group Members & Github Username

- Jingle Wang, gillianw@seas.upenn.edu (gillianwang0325)
- Ziqi He, ziqihe@seas.upenn.edu (HeidiHe2001)
- Lingpei Luo, luol@seas.upenn.edu (JLmm123)
- Guangqiuse Hu, violahu@seas.upenn.edu (violahu930)

3. Architecture

List of technologies, description of system architecture/application:

Colab(data cleaning), MySQL(database), Node.js & React (backend/frontend), VSCode & Github (source control)

A brief description of pages of application and features on each page:

- **Home Page**
  - Highlights a handpicked Airbnb listing to feature daily, promoting a unique and appealing option for users visiting the site
  - Lists the top three safest neighborhoods in Los Angeles based on safety metrics and Airbnb listing data; Each entry provides the area number, neighborhood name, and the average price of Airbnb listings in that area
  - Displays a list of the most popular neighborhoods in Los Angeles according to the total number of Airbnb listings.

- **Neighborhood Page**
    - Google map API showing the LA area maps
    - Lists the neighborhoods with hyperlinks to its neighborhood card page with detailed statistics
- **Rank Page**
    - Shows a table which ranks each neighborhood by crime count and severance level, it also shows the average rating of Airbnb listings in each neighborhood.
- **Search Listing Page**
    - Users can search Airbnbs Listings by inputting keywords in the search bar, or adjusting the filter sliders (Star Rating, # of Bedrooms, # of Bathrooms, and Minimum Nights Stay); Our application checks incorrect/null inputs.
    - Clicking the 'Search' button fetches and displays listings that match the specified criteria

4. Data

For each dataset, a link to the source, a description of the data, relevant summary statistics, an explanation of how you use the data, and details on any pre-processing that you performed on your datasets.

- LA Crime data (Crime Data from 2020 to Present - Catalog)
- LA neighborhood data (manually scrapped from LAPD map)
- LA airbnb listings data (Airbnb listings)

The data above are splitted into 7 datasets, and populated into MySQL databases.

Dataset	***Description***	***Columns***	***# of Rows***	***# of Columns***
CrimeData	Detailed crime case listings	DR_NO, Date Occ, Time Occ, AREA, Crm Cd, Premis Cd, Weapon Used Cd, Status	910,164	8
Areas	Area designations and details	AREA, SUBAREA NAME, AREA NAME	204	3
CrimeCodes	Crime codes and descriptions	Crm Cd, Crm Cd Desc	139	2
Premises	Premises types and details	Premis Cd, Premis Desc	307	2
Weapons	Weapon codes and descriptions	Weapon Used Cd, Weapon Desc	79	2
Status	Status	Status, Status Desc	6	2

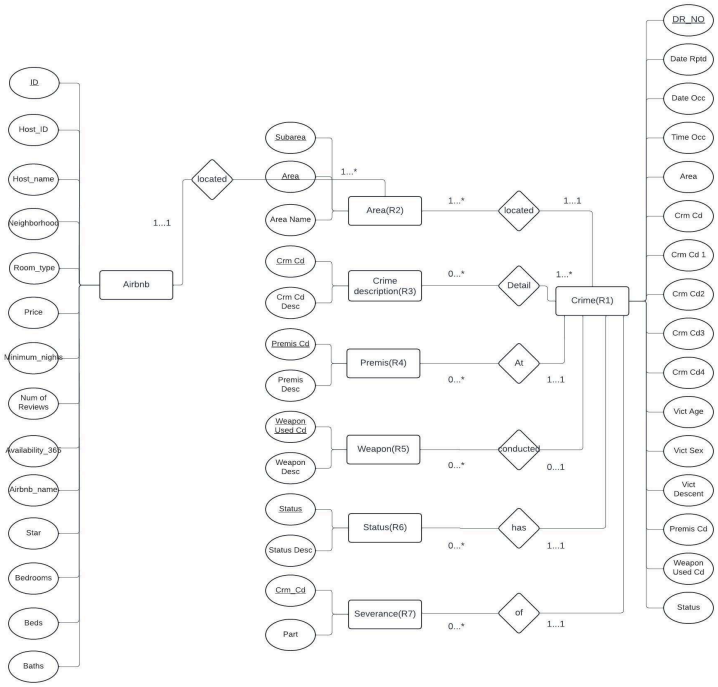
	identifiers and details			
Severance	Codes and their parts	Crm Cd, Part1-2	139	2
Airbnb	Codes and their parts	Id, host_id, host_name, neighborhood, room_type, price, minimum_nights, number_of_reviews, availability_365, airbnb_name, star, bedrooms, beds, baths	22,601	14

5. Database

Explanation of data ingestion procedure and entity resolution efforts, ER diagram, number of instances in each table, and proof of 3NF/BCNF.

- CrimeData (R1): Contains crime incident data with 910163 rows.
- Areas (R2): Contains area information with 223 rows.
- CrimeCode (R3): Contains crime code descriptions with 139 rows.
- Premis (R4): Contains premises descriptions with 307 rows.
- Weapon (R5): Contains weapon descriptions with 79 rows.
- Status (R6): Contains status descriptions with 7 rows.
- Severance (R7): Contains severance information with 139 rows.
- Airbnb: Contains Airbnb listing data with 22601 rows.

*** ER Diagram is attached in Appendix II for reference**



In the provided schema:

- `Premises (R4): Premis_Cd -> Premis_Desc`
- `Areas (R2): { AREA, SUBAREA_NAME } -> AREA_NAME`
- `Weapons (R5): Weapon_Used_Cd -> Weapon_Desc`
- `Status (R6): Status -> Status_Desc`
- `Severance (R7): Crm_Cd -> Part1, Part2`
- `CrimeData (R1): DR_NO -> { Date_Occ, Time_Occ, AREA, Crm_Cd, Premis_Cd, Weapon_Used_Cd, Status }, AREA -> { AREA_NAME, SUBAREA_NAME }, Crm_Cd -> Crm_Cd_Desc, Premis_Cd -> Premis_Desc, Weapon_Used_Cd -> Weapon_Desc, Status -> Status_Desc`
- `Airbnb: ID -> { host_id, host_name, neighborhood, room_type, price, minimum_nights, number_of_reviews, availability_365, airbnb_name, star, bedrooms, beds, baths }`

Proof of 3NF/BCNF:

- Each table has a primary key that uniquely identifies each record.
- There are no partial dependencies (all non-prime attributes are fully functionally dependent on the primary key).
- There are no transitive dependencies between non-prime attributes.

6. Queries

Examples of at least 5 queries in your application and explanations of how they're used. Please report the queries you think are most complex. Include the remaining queries in an appendix.

1. **Search_listings**: This query is used for the search feature. It filters airbnb listings by high/low star; high/low crime rate; high/low bedroom.

```
# Parameters: high_star, low_star, high_crime_rate, low_crime_rate,
high_bedroom,low_bedroom

WITH crime_neighborhood_summary AS (
    SELECT CRIME_AIRBNB.Areas.AREA AS area,
CRIME_AIRBNB.Areas.SUBAREA_NAME AS neighborhood, COUNT(*) AS total_crimes
FROM CRIME_AIRBNB.CrimeData crime
JOIN CRIME_AIRBNB.Areas ON CRIME_AIRBNB.Areas. AREA = crime.AREA
GROUP BY CRIME_AIRBNB.Areas.AREA)

SELECT id
FROM CRIME_AIRBNB.Airbnb a
JOIN crime_neighborhood_summary cns ON a.neighborhood = cns.neighborhood
WHERE cns.total_crimes > low_crime_rate AND cns.total_crimes <
high_crime_rate
AND a.star > low_star AND a.star < high_star
AND a.bedroom > low_bedroom AND a.bedroom < high_bedroom;
```

2. **Rank**: This query ranks each neighborhood by crime count and severance level, and also calculates the average rating of Airbnb listings in each neighborhood.

```
WITH CrimeCounts AS (
SELECT area.SUBAREA_NAME AS neighborhood, COUNT(*) AS crime_count,
SUM(CASE WHEN severance.Part = 1 THEN 1 ELSE 0 END) AS severe_crime_count
FROM CRIME_AIRBNB.CrimeData crime
JOIN CRIME_AIRBNB.Areas area ON crime.AREA = area.AREA
JOIN CRIME_AIRBNB.Severance severance ON crime.Crm_Cd = severance.Crm_Cd
GROUP BY area.SUBAREA_NAME),

AverageRatings AS (
SELECT neighborhood, AVG(star) AS avg_rating
FROM CRIME_AIRBNB.Airbnb airbnb
GROUP BY neighborhood)

SELECT c.neighborhood, c.crime_count, c.severe_crime_count, a.avg_rating,
RANK() OVER (ORDER BY c.crime_count, c.severe_crime_count) AS crime_rank,
RANK() OVER (ORDER BY a.avg_rating DESC) AS rating_rank
FROM CrimeCounts c JOIN AverageRatings a ON c.neighborhood =
a.neighborhood;
```

3. This query identifies High-Demand Airbnb Listings in Areas with Low Crime Rates.

```
WITH crime_summary AS (
SELECT AREA, COUNT(*) AS total_crimes
FROM CRIME_AIRBNB.CrimeData
GROUP BY AREA),
low_crime_areas AS (
SELECT AREA
FROM CRIME_AIRBNB.Areas
GROUP BY AREA
HAVING COUNT(*) < (SELECT AVG(total_crimes) FROM crime_summary))
SELECT airbnb_name, price
FROM CRIME_AIRBNB.Airbnb
WHERE neighborhood IN (
SELECT CRIME_AIRBNB.Areas.SUBAREA_NAME NAME FROM low_crime_areas la
LEFT JOIN CRIME_AIRBNB.Areas ON la.AREA = CRIME_AIRBNB.Areas.Area)
ORDER BY number_of_reviews DESC;
```

4. This query identifies neighborhoods with high average prices and low crime rates. This list is shown on the main page as the default display of listings.

```
WITH avg_price_per_neighbourhood AS (
SELECT CRIME_AIRBNB.Areas.AREA AS area, CRIME_AIRBNB.Areas.AREA_NAME
AS area_name, AVG(air.price) AS avg_price
FROM CRIME_AIRBNB.Airbnb as air
```

```
LEFT JOIN CRIME_AIRBNB.Areas ON air.neighborhood =
CRIME_AIRBNB.Areas.SUBAREA_NAME
GROUP BY CRIME_AIRBNB.Areas.AREA),

crime_summary AS (
SELECT AREA, COUNT(*) AS total_crimes
FROM CRIME_AIRBNB.CrimeData
GROUP BY AREA)

SELECT apn.area, apn.area_name, apn.avg_price, cs.total_crimes
FROM avg_price_per_neighbourhood apn
LEFT JOIN crime_summary cs ON apn.area = cs.AREA
WHERE apn.avg_price > (SELECT AVG(price) FROM CRIME_AIRBNB.Airbnb)
-- Filter by above average price
AND cs.total_crimes < (SELECT AVG(total_crimes) FROM crime_summary)
-- Filter by below average crimes
ORDER BY apn.avg_price
DESC;
```

5. **Area Statistics**: This query retrieves and computes multiple statistical summaries of a given area, including total incidents, unresolved incident rate, average price of Airbnbs, total listings of Airbnbs, and total reviews of Airbnbs. This is used for the info page for a specific area.

```
WITH crime_summary AS (
SELECT AREA, COUNT(*) AS total_incidents,
AVG(CASE WHEN `Status` = 'IC' THEN 1 ELSE 0 END) AS
unresolved_incident_rate
FROM CRIME_AIRBNB.CrimeData
WHERE YEAR(`Date_Occ`) = 2023 -- Consider incidents from the year
2023
GROUP BY AREA),

airbnb_summary AS (
SELECT CRIME_AIRBNB.Areas.AREA AS AREA,
AVG(CRIME_AIRBNB.Airbnb.price) AS avg_price, COUNT(*) AS
total_listings
FROM CRIME_AIRBNB.Airbnb JOIN CRIME_AIRBNB.Areas ON
CRIME_AIRBNB.Airbnb.neighborhood = CRIME_AIRBNB.Areas.SUBAREA_NAME
GROUP BY CRIME_AIRBNB.Areas.AREA),

popular_area AS (
SELECT area.AREA AS AREA, SUM(airbnb.NUMBER_OF_REVIEWS) AS
total_reviews
FROM CRIME_AIRBNB.Airbnb airbnb JOIN CRIME_AIRBNB.Areas area ON
airbnb.neighborhood = area.SUBAREA_NAME
```

```
GROUP BY area.AREA
)

SELECT cs.AREA, cs.total_incidents, cs.unresolved_incident_rate,
asum.avg_price, asum.total_listings, pa.total_reviews
FROM crime_summary cs
LEFT JOIN airbnb_summary asum ON cs.AREA = asum.AREA
LEFT JOIN popular_area pa ON cs.AREA = pa.AREA
ORDER BY cs.total_incidents DESC;
```

***The rest of the queries are included in the appendix.**

7. Performance evaluation

A table consisting of recorded timings before and after optimization for all 4 of your complex queries, descriptions of inputs being used, and explanations of why caching, indexing, and other optimizations improved timings.

Query Name	Cost Before Optimization	Cost After Optimization
Rank	63832.0	94.8
Search_listing	4.85e8	244643.0
High Demand & Low Crime	121306.0	15566.0
Area Statistics	2.41e7	175023.0

Rank – There are three intermediate tables, so runtime can be optimized by adding appropriate indexes on the database schema

```
CREATE INDEX idx_area ON CRIME_AIRBNB.CrimeData(AREA);
CREATE INDEX idx_neighborhood ON CRIME_AIRBNB.Areas(SUBAREA_NAME);
CREATE INDEX idx_airbnb_neighborhood ON CRIME_AIRBNB.Airbnb(neighborhood);
```

Search Listing – use indexes (composite index)

```
CREATE INDEX idx_crime_neighborhood ON CRIME_AIRBNB.CrimeData(AREA,
SUBAREA_NAME);
CREATE INDEX idx_airbnb_star_bedrooms ON CRIME_AIRBNB.Airbnb(star,
bedrooms, neighborhood);
```

High Demand & Low Crime – use indexes, and instead of having a subquery to determine `neighborhoods` from `low_crime_areas`, directly Join the necessary tables.

Area Statistics – create index if not yet created, and apply filter early for the year 2023 directly in the `FROM` clause for `CrimeData` to reduce the dataset size before operations.

```
CREATE INDEX idx_crime_area_date ON CRIME_AIRBNB.CrimeData (AREA,
Date_Occ);
```

8. Technical challenges & Optimization

- **Discrepancy between datasets**: We found out the ‘area’ in airbnb data is actually ‘subarea’ to the ‘area’ in crime data. We bridged the discrepancy by manually scraping geometric data from a LA map, and unwinding the bigger area to mapping the smaller areas to bigger ones.

- **Error encountered when populating database**: When populating the database, it will implicitly check for all foreign keys constraints. If we encounter an unexpectedly long waiting time, we know there is a pitfall in the data cleaning process - often about handling NULL values, converting INT and FLOAT, parsing nonatomic STRING cells, and there is also missing data from the original dataset. We ended up going back and forth several times until the population was successful.

- **Implementation of the interactive map feature**: We originally wanted to implement the neighborhoods as an interactive map of LA. However, after we checked documentation and referenced example code online, it seems a better idea to use a GoogleMap API and table with hyperlinks to each neighborhood’s listing instead.

- **Long waiting time for complex queries**: We tried to improve performance using indexes, and significantly reduce runtime.

Appendix I - Remaining queries excluded from part 6

1. Identifying Airbnb Listings Near High Crime Density Areas

```
WITH crime_density_per_area AS (
      SELECT AREA, COUNT(*) AS crime_density
      FROM CRIME_AIRBNB.CrimeData
      GROUP BY AREA),
avg_density_per_neighborhood AS(
      SELECT AVG(crime_density)
      FROM crime_density_per_area
      )

SELECT airbnb_data.airbnb_name, airbnb_data.price, airbnb_data.area,
crime_density
FROM(
      SELECT a.airbnb_name AS airbnb_name, a.price AS price, R2.AREA AS
      area
      FROM CRIME_AIRBNB.Airbnb a
      LEFT JOIN CRIME_AIRBNB.Areas R2 ON a.neighborhood = R2.SUBAREA_NAME)
      airbnb_data
LEFT JOIN crime_density_per_area cd ON airbnb_data.area = cd.AREA
ORDER BY cd.crime_density DESC;
```

2. Most popular neighborhood: sort neighborhood with the most Airbnb listings

```
SELECT neighborhood, COUNT(*) AS listing_count
FROM CRIME_AIRBNB.Airbnb a
GROUP BY a.neighborhood
ORDER BY COUNT(*) DESC;
```

3. Safest area: sort area with the lowest crime number

```
WITH crime_counts AS (
SELECT AREA AS area_name, COUNT(*) AS crime_count
FROM CRIME_AIRBNB.CrimeData
GROUP BY AREA)
SELECT area_name, crime_count
FROM crime_counts
ORDER BY crime_count ASC;
```

4. Given Airbnb id, retrieve Airbnb Info

```
# Parameters: given_id
SELECT * FROM CRIME_AIRBNB.Airbnb
WHERE CRIME_AIRBNB.Airbnb.id = given_id;
```

5. Given area select all airbnb id in area

```
# Parameters: given_area
```

```
SELECT id
FROM CRIME_AIRBNB.Airbnb a
JOIN CRIME_AIRBNB.Areas ON CRIME_AIRBNB.Areas.SUBAREA_NAME=
a.neighbourhood
WHERE CRIME_AIRBNB.Areas.AREA = given_area;
```

Appendix II - ER Diagram