# A Smart Solution for Tracking, Monitoring, and Visualizing Bar Lines

DS 5110: Introduction to Data Management and Processing

Cameron Jester and Matt Ferri

# Table of Contents

# Introduction

## Objectives and Goals

- Simulate data and implement an intuitive database.
- Create a robust and stable back-end for a potential mobile app.
- Mathematically aggregate data on determined intervals.
- Design a mock-up of useful user-interfaces with visualizations.

## Project Scope

- Provide users with real-time line estimates.
- Design a friendly interface.
- Provide bar and restaurant owners with analytics to forecast attendance and sales.
- Ensure the solution is scalable to expand to various types of establishments.

# Literature Review



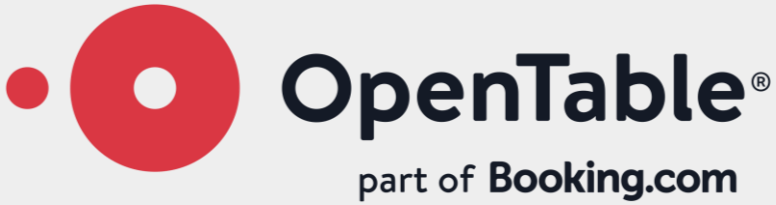



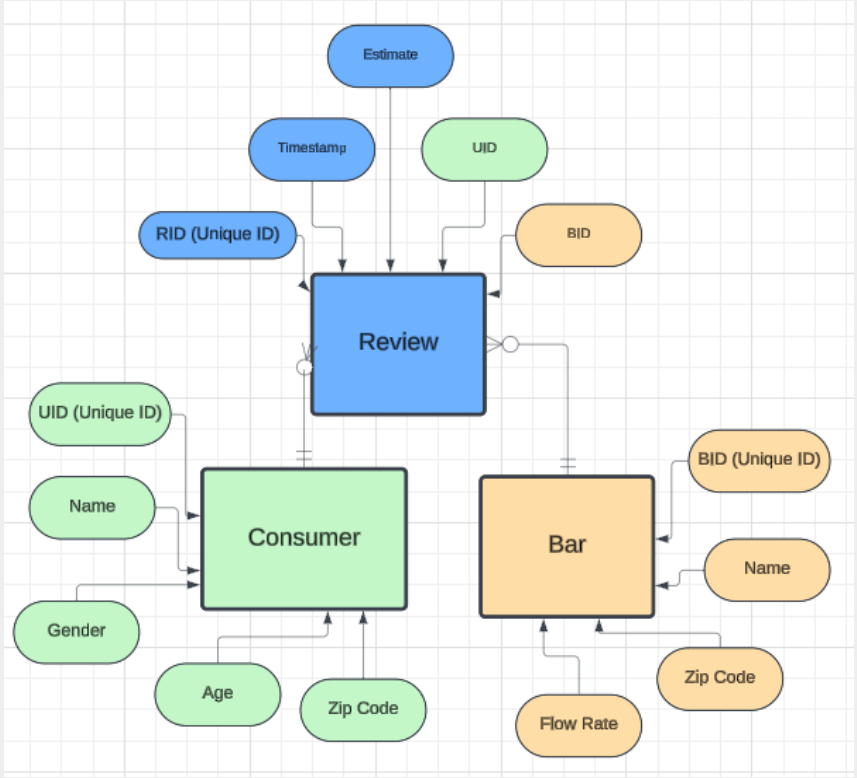

Image courtesy of OpenTable.com

- Aggregates data from users who share their phone location.
- Requires multiple searches to find wait times for various bars.

- Primarily designed for restaurant owners to manage queues.
- Limited insights for patrons.

# Methodology

- Design a normalized relational database (3NF).
- Simulate data using Python for our 3 entities: Users, Bars, and 'Reviews'.
- Create a MySQL server to store data.
- Build pipelines for stable data flow.
- Aggregate and visualize data to create reports.



# Methodology contd.

- **Language/packages:** Python, Pandas, MySQL, sqlalchemy, matplotlib, plotly, ipywidgets
- **Platforms:** JupyterLab and Google Colab (IDE), MySQL Server (storage), phpMyAdmin (DB interface)

```python
# Create tables per our ERD
try:
    connection = mysql.connector.connect(
        host='sql5.freemysqlhosting.net',
        user='sql5743967',
        password='FsUwn8fAfT',
        database='sql5743967'
    )

    if connection.is_connected():
        cursor = connection.cursor()

        cursor.execute(create_table_query)
        print("Table created successfully")

except Error as e:
    print("Error:", e)
```

```
Table created successfully
```

```
create_table_query = """
    CREATE TABLE Bars (
        BID INT PRIMARY KEY,
        Name TEXT NOT NULL,
        Zip_code INT,
        Flow_rate INT
    );

    CREATE TABLE Users (
        UID INT PRIMARY KEY,
        Name TEXT NOT NULL,
        Gender INT CHECK (Role in (0, 1, 2)),
        Age INT,
        Zip_code INT
    );

    CREATE TABLE Reviews (
        RID INT PRIMARY KEY,
        Timestamp DATETIME NOT NULL,
        UID INT,
        BID INT,
        Estimate INT,
        FOREIGN KEY (UID) REFERENCES Users(UID),
        FOREIGN KEY (BID) REFERENCES Bars(BID)
    );
    """
```

| | BID | Name | Zip_code | Flow_rate |
|---|---|---|---|---|
| Edit Copy Delete | 563 | Harbor & Hops | 2127 | 29 |
| Edit Copy Delete | 564 | Beacon Brews | 2127 | 7 |
| Edit Copy Delete | 565 | Fenway Tavern | 2127 | 13 |
| Edit Copy Delete | 566 | The Green Line Pub | 2127 | 8 |
| Edit Copy Delete | 567 | Back Bay Barrel | 2127 | 40 |
| Edit Copy Delete | 568 | Southie Spirits | 2127 | 34 |
| Edit Copy Delete | 569 | The Freedom Taproom | 2127 | 36 |
| Edit Copy Delete | 570 | Charles River Brew House | 2127 | 40 |
| Edit Copy Delete | 571 | Cobblestone & Co. | 2127 | 22 |
| Edit Copy Delete | 572 | The Bostonian Barrel | 2127 | 31 |

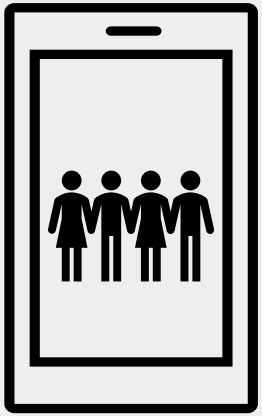
# Time Complexity Analysis

- Data Loading into and Data Querying from MySQL – full tables
  - O(n*m)
  - (n = rows, m = columns) Virtually linear time, as we are appending into an existing database and there is only an insertion operation into a tabular format.
- Subquerying Data from MySQL – querying based on criteria
  - O(log(n) + r*m)
  - (n = rows, r = matching rows, m = columns) Log OR linear time, depending on the number of matching rows. MySQL indexes data using binary trees and therefore reduces the search to log(n) compared to a non-indexed database that would take O(n) to search. It then costs r*m to retrieve the data, where r is the matching rows and m is the columns.
- Aggregations – analysis of data
  - O(log(n) + r)
  - (n = rows, r = matching rows) This is greater than log time, but less than linear time. Time complexity approaches linear as matching rows increases. Log(n) to search for matching values, then 1 operation per matching row, or r.

# Analysis and Results

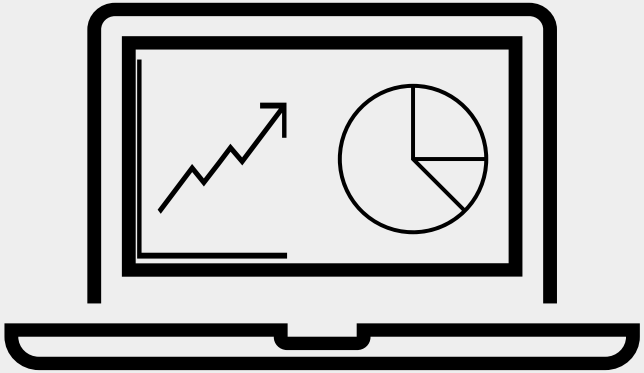
Our Stakeholders Include:

## Patrons



Patrons can access a mobile application to report and view lines

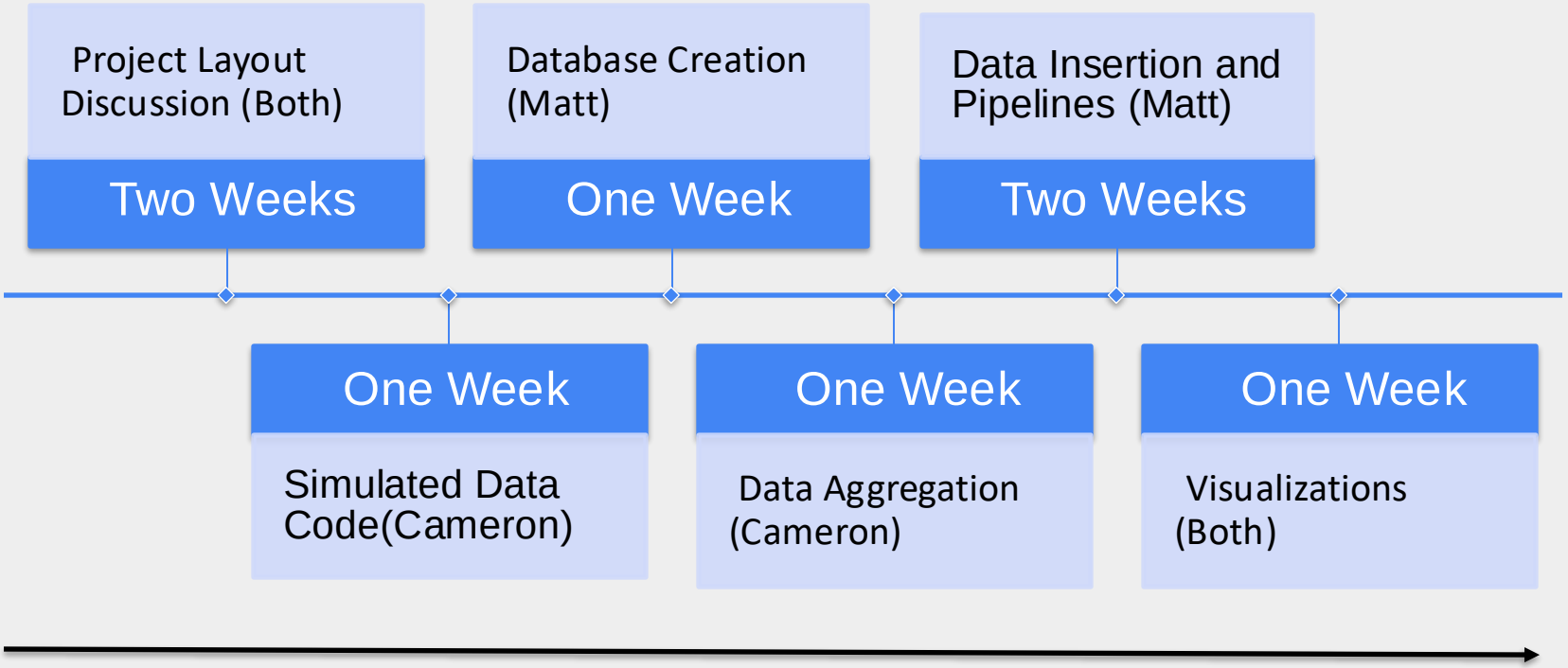
## Bar Owners



Owners can access trends to enhance their business

# Code Walkthrough and Demo

# Time and Responsibilities Breakdown




GitHub Maintenance and the Final Report were done synchronously throughout the project

# Discussion

## Implications of Findings

- Even seemingly simple mobile apps require complex back-end development.
- More "Reviews" per time period smooths averages and reduces dramatic outliers.
- Increasing data size slows down aggregations and calculations.
- Several features could be added to enhance the main project goals.

## Future Work

- Develop an actual mobile app front-end for ease of access and usability.
- Include an estimate from the Bar to augment the accuracy of user-reported line data.
- Modify our aggregation to weight older estimates lower than more recent ones.

# Conclusion

| Streamlined Processes | Sales Growth | Broad Applications |
| --- | --- | --- |
| • A real-time queuing app can improve the bar selection process. | • Bar owners benefit from forecasting customer demographics and trends. | • The app has potential applications beyond the beverage industry. |

# References


Bansal, A. (2024, December 2). *Dictionaries in python*. GeeksforGeeks. https://www.geeksforgeeks.org/python-dictionary/

Google. (n.d.). *About popular times, Wait Times & Visit Duration Data*. Google Business Profile Help. https://support.google.com/business/answer/6263531?hl=en

*Restaurants and restaurant bookings*. OpenTable. (n.d.). https://www.opentable.com/

# Question and Answer