Apartment Listing Management System

Rishi Goswamy Aadesh Mallya Satvik Khetan Meet Vora

1. Introduction

Between ever-changing property values, evolving market trends, or the sheer volume of rental details, agents, investors, and other people are often caught in a labyrinth of information. Our Apartment Listing Management System is a comprehensive database management project aimed at streamlining and enhancing real estate operations. This project addresses the growing complexity of real estate management by integrating a robust database system.

.

1. Database Design

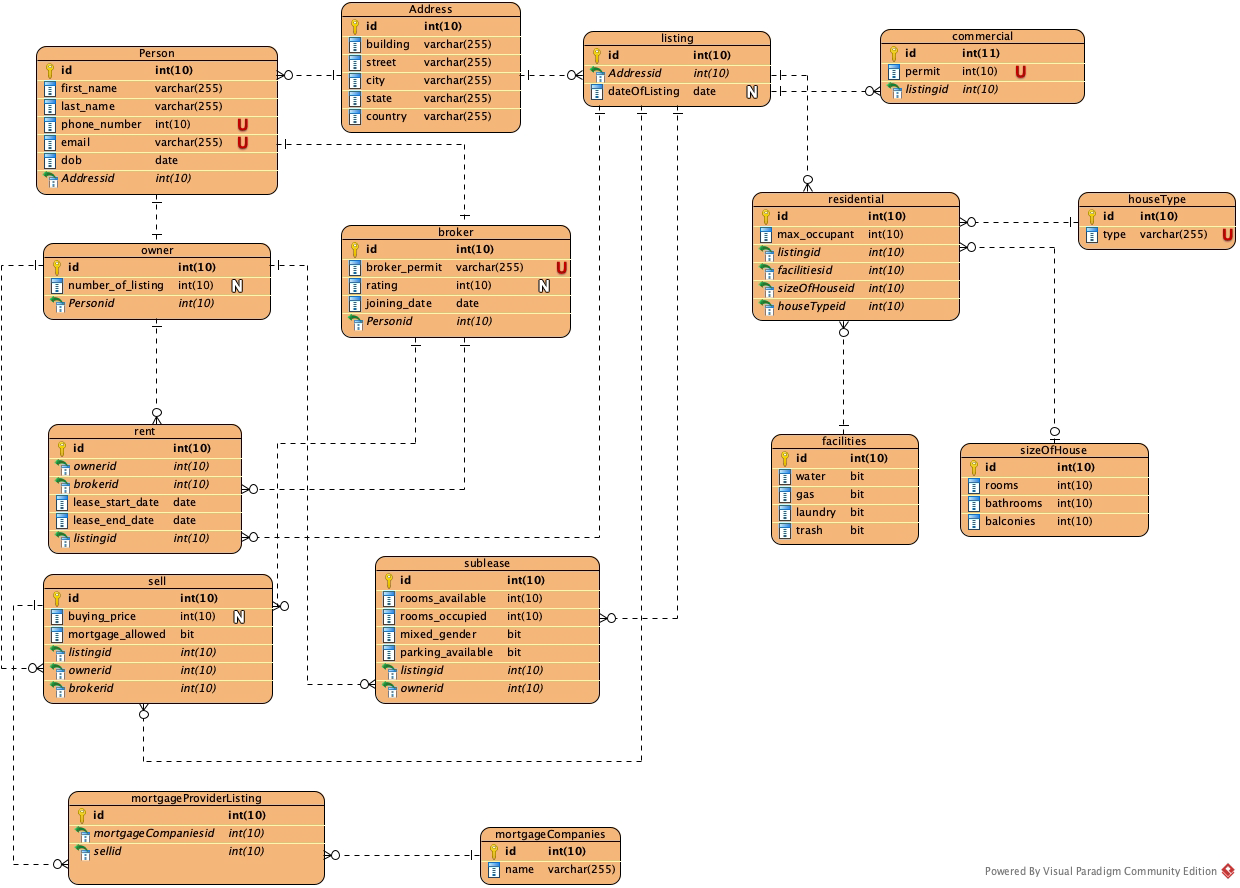
Our database system consists of 15 different tables each serving a specific purpose. The final ERD looks like -   
  


Fig 1. ERD for Apartment Listing Management System.

As we can see from the ERD we have tables for the corresponding entities -

* Person
* Owner
* Broker
* Rent
* Sell
* Sublease
* MortgageProviderListing
* MortgageCompanies
* Address
* Listing
* Residential
* Commercial
* Facilities
* SizeOfHouse
* HouseType

Some of the key decisions that we took while developing this system were –

* A person can be either an owner or a broker.
* A listing can be a residential or a commercial property.
* A listing can be sold or rented, and both owner and broker can be involved in that transaction.
* If it is a residential property, it should have certain attributes such as the house type, the facilities provided, etc.
* When a listing is being sold, it can be done through a mortgage company.
* Tables are normalized to at least 3rd normal form, by using junction tables, eliminating multi valued or composite attributes.

In addition to these entities, we have the following –

* 1. **Views**
* One view for details of property sold through an individual. This view is useful for people who are trying to directly buy or sell a property.
* Second view for properties which have been rented by brokers. This view would be useful for individuals who are just looking to find rented accommodation and are not concerned with the properties up for sale.
  1. **Stored Procedures**
* First procedure for listing properties by state. This could improve the efficiency of the query since most of the users would be searching based on some specific state.
* Second procedure for adding a broker. This procedure allows us to insert a broker detail cautiously and avoid any accidental inserts since brokers are an important part of the system and incorrect or illegitimate broker entries could lead to severe consequences. The procedure also takes care of reusing existing address and person details by relying on functions internally, and comprehensively adds details which are new and do not exist in the database.
* Third procedure for adding a mortgage company. Like brokers, it’s a good idea to add a mortgage company through a stored procedure to avoid any mistakes and assign any pseudo companies.
* Fourth procedure to list properties by zip code. Like state, this would also improve the efficiency of the system by quickly providing the listing in the zip code the user is interested in.
* Fifth procedure for adding new listing. This allows us to add a listing while making sure all the necessary details have been added for it.
  1. **Functions**
* First Function to check if the address already exists to avoid duplicates in addresses which is a common problem when dealing with addresses.
* Second Function to check if the person already exists and avoid any duplicated due to discrepancy in naming of the person.
* Third Function to find number of listings in a state. This can be very helpful in quickly figuring out all the listings state wise and provide quick results.
* Another function to find the average rating of the broker. This would be useful to suggest brokers with higher average rating to improve the user experience.
  1. **Triggers**
* First trigger makes use of one of the functions to check if the address already exists. If the address already exists, it doesn’t allow for duplicate entries. This way we can avoid data getting bad.
* Second trigger is to validate the phone number of the user is a valid one and consist of 10 digits only. This would restrain users to create pseudo accounts with random numbers.

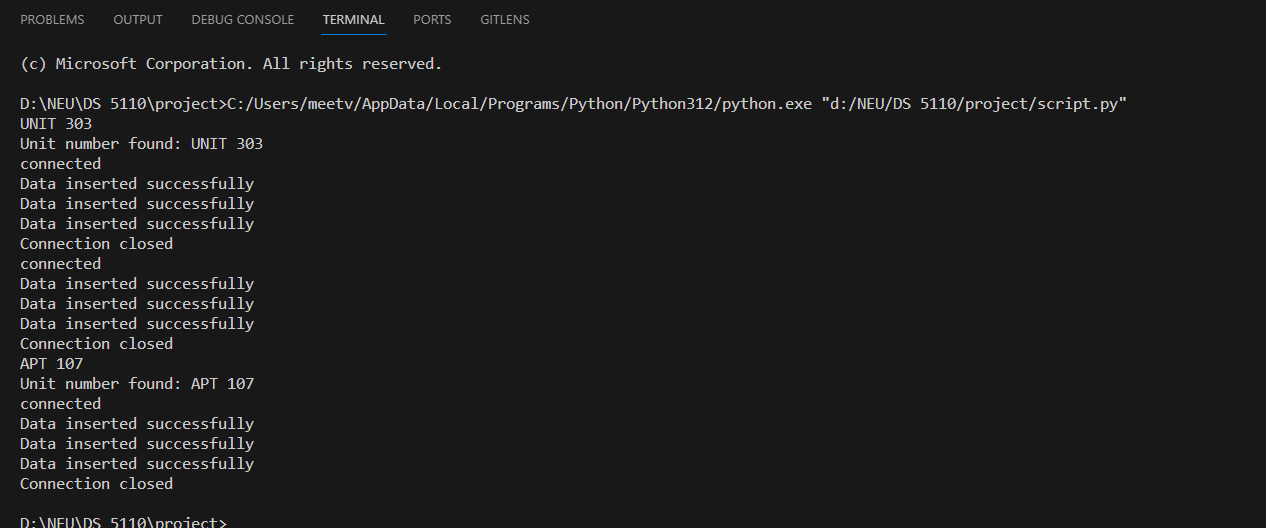


Fig 2. Sample web scrapping.

1. Data Collection

The data collection process for our application is a multi-step procedure that involves interfacing with the Zillow API through the *script.py* file.

1. Zillow API Integration: The *GetData.py* script establishes an HTTPS connection to the Zillow API, facilitated by the RapidAPI service. The script sends a GET request to the Zillow API, incorporating a property URL and authenticating with the provided API key and host information. This integration ensures that our application has real-time access to property data from Zillow.
2. Response Handling and JSON Conversion: Upon receiving the API response, the script meticulously reads the byte data and converts it into a structured JSON format using *json.loads*. This step is crucial for transforming raw data into a more manageable and interpretable structure. The JSON format allows for easy navigation and extraction of specific information, facilitating seamless integration into our application's data model.
3. Data Persistence: The JSON data is not only used for immediate database population but is also persistently stored in a local file (*data/test.json*). This archival approach ensures that a historical record of acquired data is maintained. This record is invaluable for tracking changes over time and facilitates historical data analysis.
4. Dynamic Data Acquisition: While the current script is designed for a specific Zillow property URL, its modular structure allows for dynamic data acquisition. The script can be easily adapted to support multiple URLs and diverse data collection scenarios, providing scalability and versatility in data collection efforts.
5. Automation Potential: The script's design and structure lay the foundation for automation, enabling scheduled or triggered data collection. This adaptability is crucial for scenarios where regular updates or continuous data monitoring are required. Automation streamlines the process, reducing manual intervention and ensuring the most up-to-date information is consistently available.
6. Incorporation of Mock Data: In addition to fetching data from external sources like the Zillow API, our application demonstrates versatility by incorporating mock data. This approach serves dual purposes—it allows the application to function seamlessly in the absence of real-time external data, and it facilitates testing and development in controlled environments. The integration of mock data enhances the application's resilience and adaptability to varying data availability scenarios.

A screenshot of a computer

Description automatically generated

Fig 3. Skeletal code structure of userApp.py.

1. Application Description

Our application, userApp.py, serves as a robust interface for users to interact with a MySQL database. The key features and functionalities are as follows:

1. User-Driven Data Interaction: The application's user interface (*userApp.py*) empowers users to perform various operations on the MySQL database. These operations include data retrieval, insertion, deletion, and modification, providing a comprehensive suite of functionalities for managing database content. The script has implementations for calling functions, procedures along with create, update, and delete queries.
2. Data Mapping and MySQL Integration: The *script.py* file plays a pivotal role in mapping the structured JSON data to Python objects representing entities such as Address, Facilities, SizeOfHouse, HouseType, Listing, Person, and Broker. This mapped data is then seamlessly integrated into the MySQL database, ensuring a coherent and well-organized data structure.
3. Dynamic Query Execution: Users have the flexibility to execute dynamic SQL queries, enabling them to retrieve specific data based on their criteria. This feature enhances the application's flexibility and utility, allowing users to generate custom reports or respond to specific inquiries with tailored data.
4. Data Validation and Sanitization: Integrated data validation and sanitization routines to prevent SQL injection attacks by using parameterized queries. Ensured that the data entered by users meets the database schema requirements.
5. Stored Procedure and Function Usage: The application showcases advanced database interaction capabilities by calling stored procedures and functions. For example, it calls the stored procedure "AddMortgageCompany" to insert data and verifies the insertion. Additionally, it utilizes a function ("TotalListingsInCity") for aggregated data retrieval, demonstrating the application's ability to leverage database logic for complex operations.
6. User-Interactive Operations:The application offers a user-friendly interface for inputting parameters for various operations. Users can specify the number of months for lease end date intervals, provide mortgage company names, and input details for adding or deleting addresses. This interactive design ensures that users can easily navigate and utilize the application's functionalities.
7. Graphs and Reports: There are two implementations for graphs, one for Number of listings by State, and another for Number of Listings by Owner. We have leveraged matplotlib.pyplot for plotting.
8. Robust Error Handling and Rollback: Robust error handling mechanisms are implemented throughout the application to ensure reliability. In case of errors, the application gracefully rolls back transactions to maintain the integrity of the MySQL database. This approach contributes to data consistency and prevents the introduction of incomplete or erroneous information.
9. Comprehensive Documentation and Code quality Developed an extensive suite of documentation covering every aspect of the application. Adopted best coding practices by including clear, concise comments in the code, which explain the purpose and functionality.

In summary, our user application (*userApp.py*) is a comprehensive and interactive tool for managing a MySQL database. It seamlessly integrates with data collected from the Zillow API, providing users with a powerful platform to perform diverse operations and derive valuable insights from the database. The application's features are designed to cater to a range of user needs, from basic data manipulation to advanced querying and analysis, while also accommodating scenarios where real-time external data may not be available.

A graph of a number of listings

Description automatically generated

Fig 4. Graph depicting Number of listings by State.

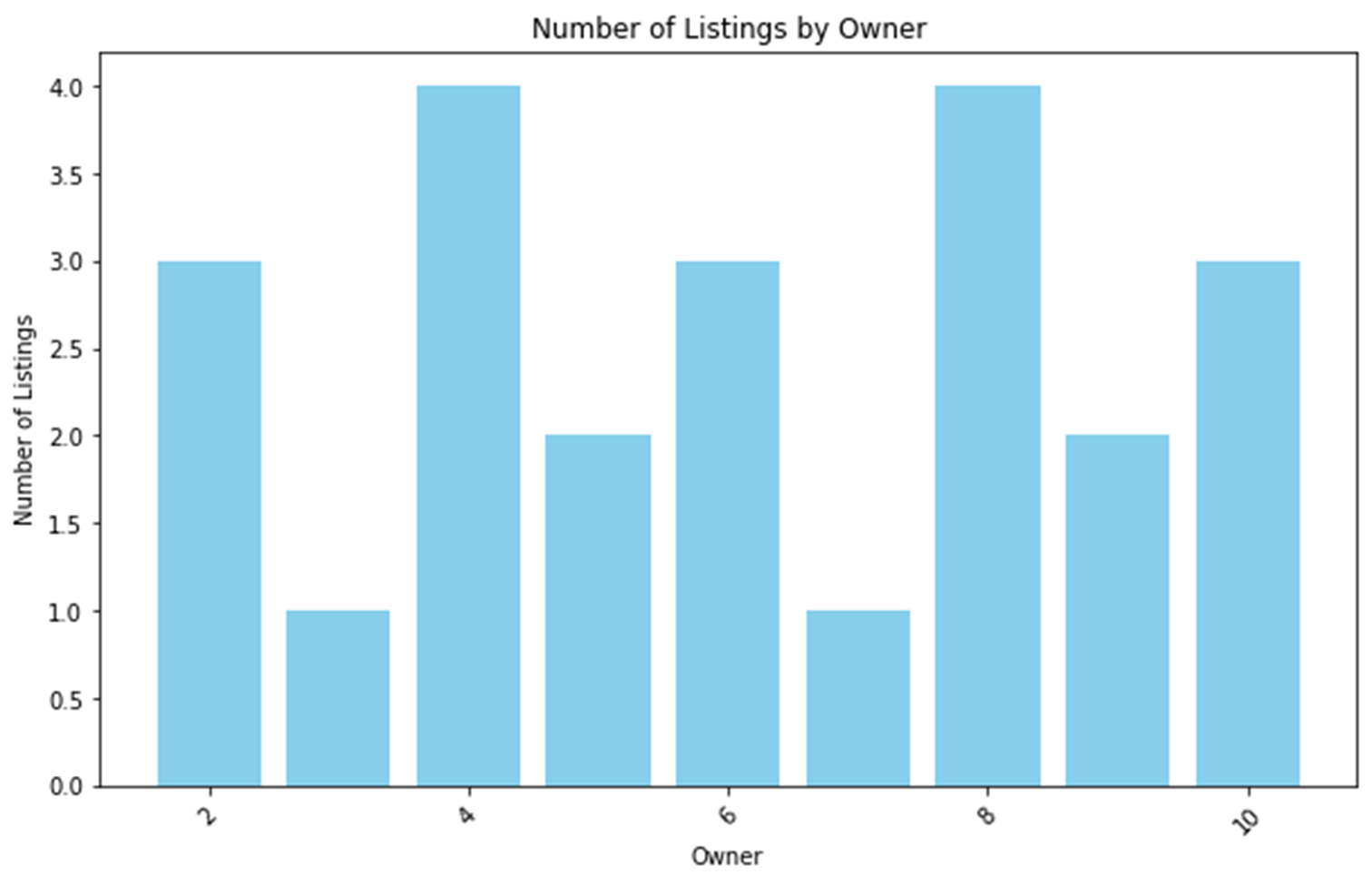


Fig 5. Number of Listings by Owner.

1. Conclusion/Future Direction

Through this project, we have learned a lot regarding how the entire end to end system works with a database system in place, an application to use it and a continuous stream of data input to enrich the dataset. To improve this further, we can add Geospatial Integration which can enhance the property search functionality with geospatial data. We can also add user authentication to have a secure and controlled access. For future students, we can advise them to use this opportunity to build a system which can be a very good learning experience of how different components come into action together.