HOTEL BOOKING MANAGEMENT SYSTEM

Project Report

Submitted by: Sagan gangwar

Course: 25BAI10057

Date: November 25, 2025

Table of Contents

---

1. Introduction

The hospitality industry relies heavily on efficient management of room inventory and guest information. For small to medium-sized hotels, bed & breakfasts, and guest houses, relying on

manual paper-based systems or disjointed spreadsheets can lead to errors, doublebookings, and inefficient operational workflows.

The **Hotel Booking Management System** is a desktop application designed to address these challenges. Built using Python, it provides a user-friendly graphical interface for front-desk staff to manage the entire guest cycle, from check-in to check-out. By centralizing data in a secure local database, the system ensures accuracy, provides real-time visibility into room occupancy, and streamlines billing processes, ultimately enhancing operational efficiency and guest satisfaction.

2. Problem Statement

Small and independent lodging establishments often lack access to expensive, complex property management software. As a result, they frequently depend on outdated manual methods for managing reservations.

Key problems identified include:

- **Risk of Double-Booking:** Manual logs make it difficult to instantly verify room availability, leading to errors where the same room is promised to multiple guests.
- **Operational Inefficiency:** Checking guests in and out, calculating bills, and updating room status manually is time-consuming and prone to human error.
- **Lack of Real-Time Visibility:** Owners and staff cannot easily see a snapshot of current hotel occupancy at any given moment.
- **Data Loss & Insecurity:** Paper records can be easily lost, damaged, or accessed by unauthorized personnel.

There is a clear need for a simple, cost-effective, and digital solution to centralize and automate these essential front-desk tasks.

3. Functional Requirements

The system is designed with the following core functional modules:

- **3.1 Database Management & Initialization:**
    - The system shall automatically create a local SQLite database file (hotel.db) upon its first execution.
    - It shall define the necessary tables for rooms and bookings.
    - It shall automatically populate the rooms table with a predefined inventory of room varying by type and price (e.g., Single, Double, Deluxe).

- **3.2 Booking and Check-In Module:**
    - The system shall provide a graphical input form for capturing guest details, including name and phone number.
    - It shall allow the user to specify the duration of the stay in days.
    - It shall provide a dynamic dropdown list that displays *only* rooms that are currently marked as 'Available'. ○ Upon confirming a booking, the system shall insert a new registration record into the bookings table and immediately update the status of the selected room to 'Booked'.
    - The system shall validate input fields to prevent incomplete bookings.
- **3.3 Current Bookings View Module:**
    - The system shall present a real-time, tabular dashboard (Treeview) displaying all active guests currently checked into the hotel. ○ Displayed information shall include Guest Name, Phone Number, assigned Room Number, Check-In Date, and stay duration.
- **3.4 Check-Out and Billing Module:**
    - The system shall allow staff to select an active booking from the dashboard to initiate the check-out process. ○ Upon confirmation, it shall automatically calculate the total bill amount by multiplying the room's nightly price by the number of days stayed. ○ It shall display the final bill amount to the user in a pop-up dialog. ○ Upon completion, the system shall remove the booking record and revert the room's status back to 'Available' for future guests.

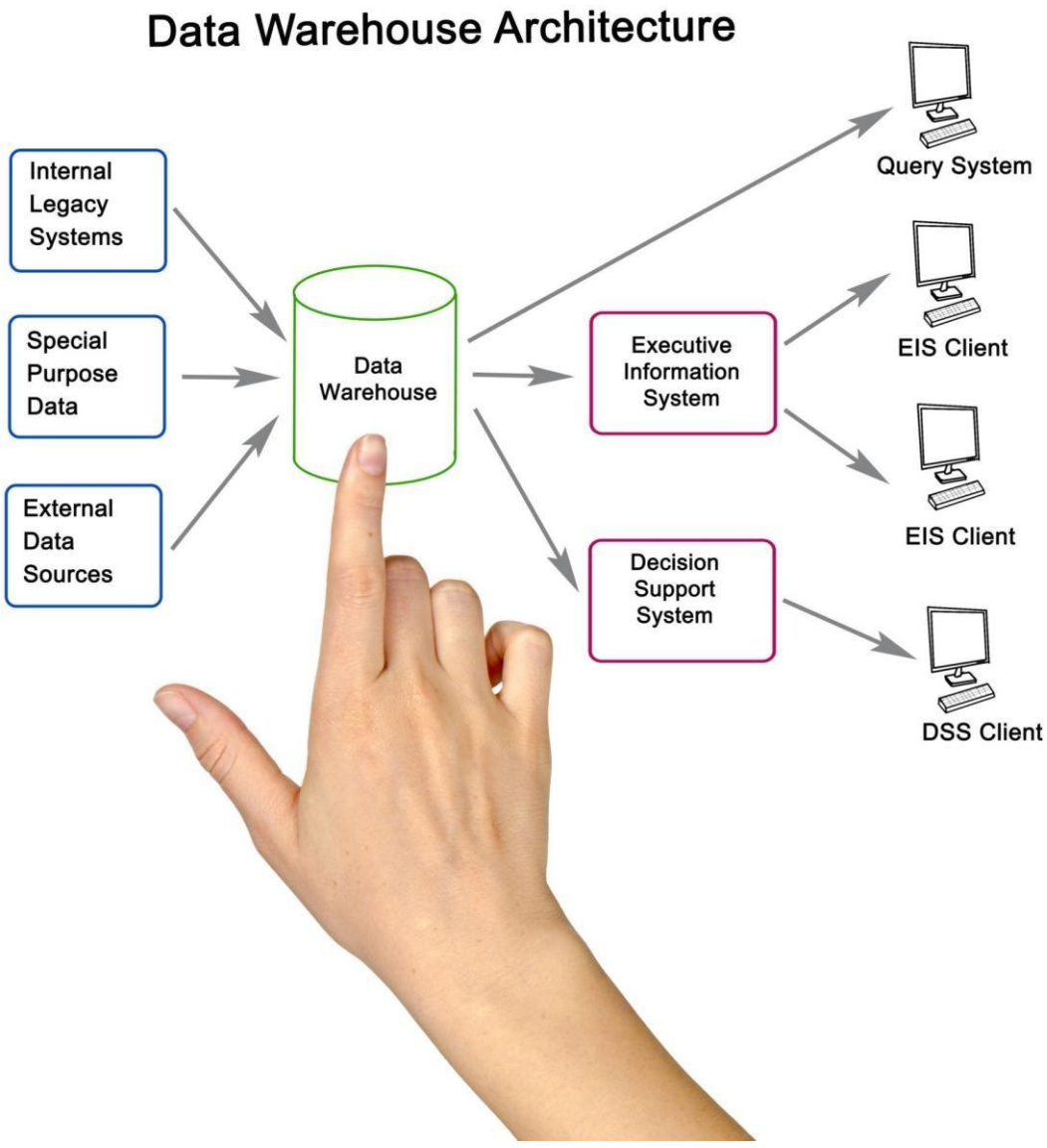
4. Non-functional Requirements

- **4.1 Usability:** The application shall feature a clean, intuitive Graphical User Interface (GUI) built with Tkinter. The layout shall be divided logistically, with data entry on one side and data visualization on the other, allowing staff with minimal technical training to use it effectively.
- **4.2 Reliability & Data Persistence:** All critical data, including room inventory and guest bookings, must be stored persistently in a local relational database (SQLite). This ensures that data is not lost when the application is closed or re-opened.
- **4.3 Data Integrity:** The system shall enforce referential integrity between bookings and rooms. It must also prevent invalid operations, such as attempting to book a room that is already occupied or creating a booking without necessary guest details.

- **4.4 Performance:** The application is designed for a single-user desktop environment. Database queries for fetching available rooms and current bookings shall execute promptly to ensure a responsive user experience without noticeable lag.
- **4.5 Maintainability:** The source code shall be organized using Object-Oriented Programming (OOP) principles, encapsulating logic within a class structure. Database operations shall be separated from GUI event handling code to improve readability and ease of future updates.

5. System Architecture

The system follows a monolithic desktop application architecture with a clear separation of concerns, mimicking a basic Model-View-Controller (MVC) pattern.



Getty Images

- **User Interface (View):** Built using Python's tkinter library, this layer presents forms, buttons, and data tables to the user and captures their input.
- **Application Logic (Controller):** Implemented within the HotelManagementApp Python class. Methods like book_room and checkout_guest contain the business logic that acts on user input and coordinates with the data layer.
- **Data Storage (Model):** Python's built-in sqlite3 module manages all interactions with the local hotel.db file, handling SQL queries to create, read, update, and delete data records.

6. Design Diagrams

6.1 Use Case Diagram

This diagram illustrates the primary interactions between the Front-Desk Staff user and the system.

6.2 Workflow Diagram

This flowchart details the step-by-step process for the core "Book Room" function.

6.3 Sequence Diagram

This diagram visualizes the sequence of operations and object interactions during a guest check-out.

6.4 Class Diagram

Represents the static structure of the application's main class.

6.5 Entity-Relationship (ER) Diagram

Shows the logical structure of the database and relationships between entities.

7. Design Decisions & Rationale

- **Technology Stack (Python & Tkinter):** Python was chosen for its simplicity and rapid development capabilities. Tkinter was selected as the GUI framework because it is built into Python's standard library, requiring no external dependencies and ensuring cross-platform compatibility for desktop deployment.
- **Database (SQLite):** SQLite was chosen as the database engine because it is serverless, zero-configuration, and self-contained within a single file. This makes it ideal for a

lightweight, single-user desktop application where setting up a fullfledged database server (like MySQL or PostgreSQL) would be overkill.

- **Automated Initialization:** The decision to include a setup_database function that runs on startup was made to simplify deployment. It ensures the application is "plug-and-play," ready to be used immediately without requiring manual SQL script execution by the user.
- **Dynamic Room ComboBox:** Instead of a static list, the room selection dropdown is dynamically populated with an SQL query (SELECT ... WHERE status = 'Available'). This is a critical design choice to prevent double-booking errors by physically restricting the user's choices to only free rooms.

8. Implementation Details

The final solution is implemented as a single Python script (hotel_app.py) containing the HotelManagementApp class.

Key Implementation Highlights:

- **Initialization (__init__)**: Sets up the main GUI window, defines styles, calls the database setup method, builds the GUI layout (input frame and treeview frame), and loads initial data.
- **Database Handling**: A helper method run_query encapsulates the boilerplate code for connecting to the SQLite database, creating a cursor, executing a query, and committing changes. This promotes code reusability.
- **GUI Layout**: The interface uses Tkinter's pack and grid geometry managers. LabelFrame widgets are used to group related controls logically for "New Booking" and "Current Bookings".
- **Treeview Widget**: The ttk.Treeview widget is used to display tabular booking data. Columns are configured with headings and specific widths for a neat presentation.
- **Event Handling**: Button clicks (command=...) are linked to specific methods in the class (e.g., self.book_room), which contain the logic to validate input, update the database, and refresh the GUI.

9. Screenshots / Results

- **Figure 1: Application Launch & Empty State** The application starts with a populated room inventory and an empty booking list. *(Place screenshot here)*

- **Figure 2: Creating a New Booking** A user fills in guest details, selects an available room from the dropdown, and clicks "Book Room". *(Place screenshot here)*
- **Figure 3: Active Bookings Dashboard** Following successful bookings, the data is displayed in the "Current Bookings" table, and the booked rooms are removed from the available list. *(Place screenshot here)*
- **Figure 4: Guest Check-Out and Billing** A staff member selects a booking and clicks "Check Out". A confirmation pop-up shows the calculated bill amount. *(Place screenshot here)*

10. Testing Approach

A manual testing approach was adopted to verify the system's functionality against requirements. Test Case

ID	Description	Expected Outcome	Result
TC001	First Run Initialization	Database file created; dummy rooms inserted.	Pass
TC002	Booking with Empty	Error message pop-up: "Please fill all fields". Fields	Pass
TC003	Successful Booking	Guest added to table; room removed from dropdown.	Pass
TC004	Check-Out without Selection	Warning pop-up: "Please select a booking".	Pass
TC005	Successful Check-Out	Confirmation dialog; correct bill calculated; room marked available.	Pass
TC006	Data Persistence	Close and reopen app; previous bookings are still visible.	Pass

Export to Sheets

11. Challenges Faced

- **SQLite Bulk Insert:** Initially, an error occurred when trying to insert multiple dummy rooms using a generic query function designed for single records. This was resolved by learning about and implementing the cursor.executemany() method for bulk operations.

- **GUI Layout Management:** Getting Tkinter widgets to align correctly and resize gracefully using pack and grid required trial and error to achieve a balanced and visually appealing layout.
- **Dynamic Dropdown Updates:** Ensuring the "Available Rooms" combo box refreshed correctly after every booking and check-out action required careful orchestration of database queries and GUI update module calls.

12. Learnings & Key Takeaways

- Gained practical experience in building a complete, functional desktop application using Python and Tkinter.
- Deepened understanding of relational database concepts, including table creation, primary/foreign keys, and performing CRUD (Create, Read, Update, Delete) operations via SQL.
- Learned the importance of separating database logic from GUI code for better maintainability.
- Understood how to handle user events and validate input to build a robust application.

13. Future Enhancements

- **Date Range Bookings:** Implement a date picker to allow bookings for future dates, rather than just immediate check-ins. This would require a more complex database schema to track room availability by date.
- **Guest History:** Create a separate table to store historical guest data, allowing for returning guest recognition and past booking reporting.
- **Reporting Module:** Add functionality to generate simple reports, such as total revenue for the day/month or occupancy rate statistics.
- **Form Validation:** Implement more robust validation, such as ensuring phone numbers contain only digits.

14. References

- Python Software Foundation. (n.d.). *Python 3.12.0 documentation: sqlite3 — DB-API 2.0 interface for SQLite databases*. Retrieved from https://docs.python.org/3/library/sqlite3.html
- Python Software Foundation. (n.d.). *Python 3.12.0 documentation: tkinter — Python interface to Tcl/Tk*. Retrieved from https://docs.python.org/3/library/tkinter.html

- Lundh, F. (n.d.). *An Introduction to Tkinter*. Retrieved from https://tkdocs.com/