UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
ACCREDITED UNIVERSITY

CASE STUDY

ON

Hotel Management System

Subject Code : 24CAH-606

Master In Computer Application

Submitted By:

Name : Abhijeet Kumar
UID : 24MCA20373
Class/Section : 24MCA-6(A)

Submitted To:

Mrs.Palwinder Kaur Mangat
(Asistent Proffesor**)**

Teacher Sign:

UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

Chandigarh University , Gharun

ABSTRACT

This case study explores the development of a Hotel Management System using Python's Tkinter library for the graphical user interface (GUI) and MySQL for backend database management. The system is designed to automate core operations of small to medium-sized hotels, including room booking, customer data management, billing, and real-time room availability tracking. The manual processes often used by small hotels are prone to inefficiencies and human error, which this system addresses through automation and a user-friendly interface. The system allows hotel staff to easily input customer details, calculate billing based on the stay duration, and update room availability in real time. The integration with MySQL ensures permanent record storage and data retrieval. The study details the design, implementation, and challenges faced during development, demonstrating how Python can be effectively applied to solve real-world problems in the hospitality industry. This system improves operational efficiency, reduces errors, and enhances customer service by simplifying hotel management tasks.

Introduction

Hotel management involves numerous complex tasks, including handling customer reservations, billing, room availability tracking, and maintaining customer records. For small to medium-sized hotels, these tasks are often managed manually, leading to inefficiencies and increased chances of human error. This case study focuses on the development of an automated **Hotel Management System** using Python's Tkinter library for the graphical user interface (GUI) and MySQL for backend database management. The system simplifies the booking process, manages customer information, calculates billing, and updates room availability in real-time. The integration of a robust database ensures efficient data handling and permanent storage, allowing staff to easily retrieve and update customer records. By automating daily operations, the system significantly enhances the overall efficiency of hotel management, leading to improved customer service and smoother operational workflows. This study showcases how technology can streamline hospitality services and reduce manual effort.

.

Problem Definition

Small and medium-sized hotels often struggle with keeping track of room availability, customer billing, and basic record-keeping. Manual processes lead to inefficiency, human error, and poor customer experiences. The hotel management system solves these issues by:

1. Automating the room booking process.
2. Managing customer details and contacts.
3. Handling billing calculations for hotel stays.
4. Providing an intuitive user interface for non-technical hotel staff.

Objectives

1. **Automation of customer booking:** The system automates the booking process by keeping track of available rooms and allowing hotel staff to easily input customer information.

2. **Billing System:** It calculates bills based on the number of nights a customer stays, automating manual processes and reducing errors.

3. **Real-time Room Availability:** The system tracks available rooms and updates them automatically when a booking is made.

4. **Data Management:** Integration with MySQL allows for the permanent storage of customer records and room availability data.

5. **User-friendly Interface:** The system uses Tkinter to create an intuitive GUI for hotel staff to interact with.

System Design and Features

1. **Graphical User Interface (GUI):**
   The front-end is designed using Python's Tkinter library, providing a clean, simple, and user-friendly interface for hotel staff to:
   - Enter customer details.
   - Track the available rooms.
   - Display customer bills after processing.

   The GUI consists of:

   - Input fields for customer details (name, ID, contact number).
   - Buttons for submitting the booking and inserting new data into the database.
   - A listbox for displaying customer bills.

2. Database Connectivity:

- The system uses MySQL (via pymysql) to store and retrieve customer and room data.
- It inserts customer details and updates room availability dynamically.
- The database table hotel_info is used to store key information like available rooms and rent per night.

3. Billing System:

The system calculates the total bill for a customer based on the number of nights stayed and the per-night room rent. If rooms are available, the booking is confirmed and room availability is updated in the database.

4. Error Handling :

The system includes error handling for cases such as attempting to book when no rooms are available. In such cases, a message box informs the user that all rooms are reserved.

Code Implementation

The Hotel Management System is built using Python, Tkinter, and MySQL. Below is a breakdown of the core features.

```
import tkinter as tk

from tkinter import messagebox, scrolledtext

import pymysql

from datetime import datetime


# Constants

ROOM_COST_PER_DAY = 1200
```

```
# Initialize available rooms

available_rooms = list(range(1, 11))

checked_in_guests = [] # To store guest information


class HotelManagement:

    def __init__(self, root):

        self.root = root

        self.root.title("Hotel Management System")


        # Create a scrolled text area for displaying guest information

        self.guest_info_display = scrolledtext.ScrolledText(root, width=50, height=15,
state='disabled')

        self.guest_info_display.pack(pady=10)


        # Create buttons

        btn_fetch_rooms = tk.Button(root, text="Fetch Available Room Data",
command=self.fetch_available_rooms)

        btn_check_in = tk.Button(root, text="Check In", command=self.check_in)

        btn_check_out = tk.Button(root, text="Check Out", command=self.check_out)

        btn_show_guest_list = tk.Button(root, text="Show All Guest List",
command=self.show_guest_list)

        btn_show_history = tk.Button(root, text="Show Guest History",
command=self.show_guest_history)


        # Arrange buttons in the window

        btn_fetch_rooms.pack(pady=10)
        btn_check_in.pack(pady=10)

        btn_check_out.pack(pady=10)

        btn_show_guest_list.pack(pady=10)

        btn_show_history.pack(pady=10)


    def fetch_available_rooms(self):

        if available_rooms:

            messagebox.showinfo("Fetch Available Rooms", f"Available Rooms:
{available_rooms}")

        else:

            messagebox.showinfo("Fetch Available Rooms", "No rooms available.")


    def check_in(self):

        def submit():

            name = entry_name.get()

            phone = entry_phone.get()

            gender = entry_gender.get()

            email = entry_email.get()

            days = entry_guests.get()


            if not name or not phone or gender == "Select" or not email or not days.isdigit():

                messagebox.showwarning("Input Error", "Please fill in all fields correctly.")

                return
            if not available_rooms:

                messagebox.showwarning("No Rooms Available", "No available rooms to check
in.")

                return


            room_number = available_rooms.pop(0)  # Assign the first available room

            total_cost = ROOM_COST_PER_DAY * int(days)  # Calculate total cost


            # Insert guest information into the database

            self.db_connect(name, phone, gender, email, days, room_number, total_cost)


            self.update_guest_info_display()  # Update display after check-in

            messagebox.showinfo("Check In", f"Check-in Successful!\nRoom Number:
{room_number}\nTotal Bill Amount: ₹{total_cost}")

            check_in_window.destroy()  # Close the popup after submission


        # Create a new popup window

        check_in_window = tk.Toplevel(self.root)

        check_in_window.title("Check In")


        # Create input fields

        tk.Label(check_in_window, text="Guest Name:").pack()

        entry_name = tk.Entry(check_in_window)

        entry_name.pack()
```

```
tk.Label(check_in_window, text="Phone Number:").pack()

entry_phone = tk.Entry(check_in_window)

entry_phone.pack()


# Gender selection using a dropdown menu

tk.Label(check_in_window, text="Gender:").pack()

entry_gender = tk.StringVar(check_in_window)

entry_gender.set("Select")  # Default value

gender_menu = tk.OptionMenu(check_in_window, entry_gender, "Male", "Female",
"Other")

gender_menu.pack()


tk.Label(check_in_window, text="Email:").pack()

entry_email = tk.Entry(check_in_window)

entry_email.pack()


tk.Label(check_in_window, text="Number of days you will stay:").pack()

entry_guests = tk.Entry(check_in_window)

entry_guests.pack()


# Submit button

btn_submit = tk.Button(check_in_window, text="Submit", command=submit)

btn_submit.pack(pady=10)
```

```
def check_out(self):
    def confirm_checkout():
        selected_guest = guest_var.get()
        if selected_guest == "Select Guest":
            messagebox.showwarning("Select Guest", "Please select a guest to check out.")
            return

        # Find guest in the list and remove them
        for guest in checked_in_guests:
            if guest["name"] == selected_guest:
                # Move guest data to checked_out_guests table
                self.move_to_checked_out(guest)

                available_rooms.append(guest["room"])  # Free up the room
                checked_in_guests.remove(guest)  # Remove guest from the list
                self.update_guest_info_display()  # Update display after check-out
                messagebox.showinfo("Check Out", f"Checked out {selected_guest} from
Room {guest['room']}.")
                break
        else:
            messagebox.showwarning("Checkout Error", "Guest not found.")

        checkout_window.destroy()
        # Create a new popup window for checkout

        checkout_window = tk.Toplevel(self.root)

        checkout_window.title("Check Out")


        tk.Label(checkout_window, text="Select Guest to Check Out:").pack()


        # Dropdown for selecting guest

        guest_var = tk.StringVar(checkout_window)

        guest_var.set("Select Guest")  # Default value

        guest_names = [guest['name'] for guest in checked_in_guests] + ["Select Guest"]

        guest_menu = tk.OptionMenu(checkout_window, guest_var, *guest_names)

        guest_menu.pack()


        # Confirm checkout button

        btn_confirm = tk.Button(checkout_window, text="Confirm Check Out",
command=confirm_checkout)

        btn_confirm.pack(pady=10)


    def show_guest_list(self):

        try:

            con = pymysql.connect(host='localhost', user='root', passwd='Abhi@8340',
database='hotel')

            cur = con.cursor()

            cur.execute("SELECT * FROM guests")

            guest_data = cur.fetchall()
            con.close()

            self.guest_info_display.configure(state='normal')  # Allow editing
            self.guest_info_display.delete(1.0, tk.END)  # Clear existing text

            if not guest_data:
                self.guest_info_display.insert(tk.END, "No guests currently checked in.")
            else:
                for row in guest_data:
                    guest_details = (f"Name: {row[1]}\n"
                                     f"Phone: {row[2]}\n"
                                     f"Gender: {row[3]}\n"
                                     f"Email: {row[4]}\n"
                                     f"Days: {row[5]}\n"
                                     f"Room: {row[6]}\n"
                                     f"Total Bill: ₹{row[7]}\n"
                                     f"{'='*30}\n")
                    self.guest_info_display.insert(tk.END, guest_details)

            self.guest_info_display.configure(state='disabled')  # Make the text area read-only
        except Exception as e:
            messagebox.showerror("Database Error", f"An error occurred: {e}")

    def show_guest_history(self):
        try:
            con = pymysql.connect(host='localhost', user='root', passwd='Abhi@8340',
database='hotel')
            cur = con.cursor()
            cur.execute("SELECT * FROM checked_out_guests")
            history = cur.fetchall()
            con.close()

            if not history:
                messagebox.showinfo("Guest History", "No guests have checked out yet.")
                return

            history_info = "\n".join([f"Name: {row[1]}, Phone: {row[2]}, Gender: {row[3]}, "
                                      f"Email: {row[4]}, Days: {row[5]}, Room: {row[6]}, "
                                      f"Total Bill: ₹{row[7]}, Checkout Date: {row[8]}"
                                      for row in history])
            messagebox.showinfo("Guest History", f"Checked Out Guests:\n{history_info}")
        except Exception as e:
            messagebox.showerror("Database Error", f"An error occurred: {e}")

    def update_guest_info_display(self):
        self.guest_info_display.configure(state='normal')  # Allow editing
        self.guest_info_display.delete(1.0, tk.END)  # Clear existing text
        if not checked_in_guests:
            self.guest_info_display.insert(tk.END, "No guests currently checked in.")
        else:
            for guest in checked_in_guests:
                guest_details = (f"Name: {guest['name']}\n"
                                 f"Phone: {guest['phone']}\n"
                                 f"Gender: {guest['gender']}\n"
                                 f"Email: {guest['email']}\n"
                                 f"Days: {guest['days']}\n"
                                 f"Room: {guest['room']}\n"
                                 f"Total Bill: ₹{guest['total_cost']}\n"
                                 f"{'='*30}\n")
                self.guest_info_display.insert(tk.END, guest_details)


        self.guest_info_display.configure(state='disabled')  # Make the text area read-only


    # Database connection and insertion function
    def db_connect(self, name, phone, gender, email, days, room_number, total_cost):
        try:
            con = pymysql.connect(host='localhost', user='root', passwd='Abhi@8340', 
database='hotel')
            cur = con.cursor()
            cur.execute('''
            INSERT INTO guests (name, phone, gender, email, days, room_number, 
total_cost)
```

```
            VALUES (%s, %s, %s, %s, %s, %s, %s);

            ''', (name, phone, gender, email, days, room_number, total_cost))

            con.commit()

            con.close()

            checked_in_guests.append({

                "name": name,

                "phone": phone,

                "gender": gender,

                "email": email,

                "days": days,

                "room": room_number,

                "total_cost": total_cost

            })

            messagebox.showinfo("Success", "Guest information saved successfully.")

        except Exception as e:

            messagebox.showerror("Database Error", f"An error occurred: {e}")



    def move_to_checked_out(self, guest):

        try:

            con = pymysql.connect(host='localhost', user='root', passwd='Abhi@8340',
database='hotel')

            cur = con.cursor()

            cur.execute('''

            INSERT INTO checked_out_guests (name, phone, gender, email, days,
room_number, total_cost)
```

```
            VALUES (%s, %s, %s, %s, %s, %s, %s);
            ''', (guest["name"], guest["phone"], guest["gender"], guest["email"],
                guest["days"], guest["room"], guest["total_cost"]))
            con.commit()
            con.close()
            messagebox.showinfo("Success", "Guest checked out successfully.")
        except Exception as e:
            messagebox.showerror("Database Error", f"An error occurred: {e}")

# Create the main application
if __name__== "__main__":
    root = tk.Tk()
    app = HotelManagement(root)
    root.mainloop()
```

Main Application: The main application window contains input fields for customer information and buttons to process bookings and insert data into the database.

Database Functions: These methods handle database interactions such as inserting data and retrieving room availability.

Booking Logic and Billing Calculation:

The system dynamically checks room availability and processes the customer booking, updating the available room count accordingly.

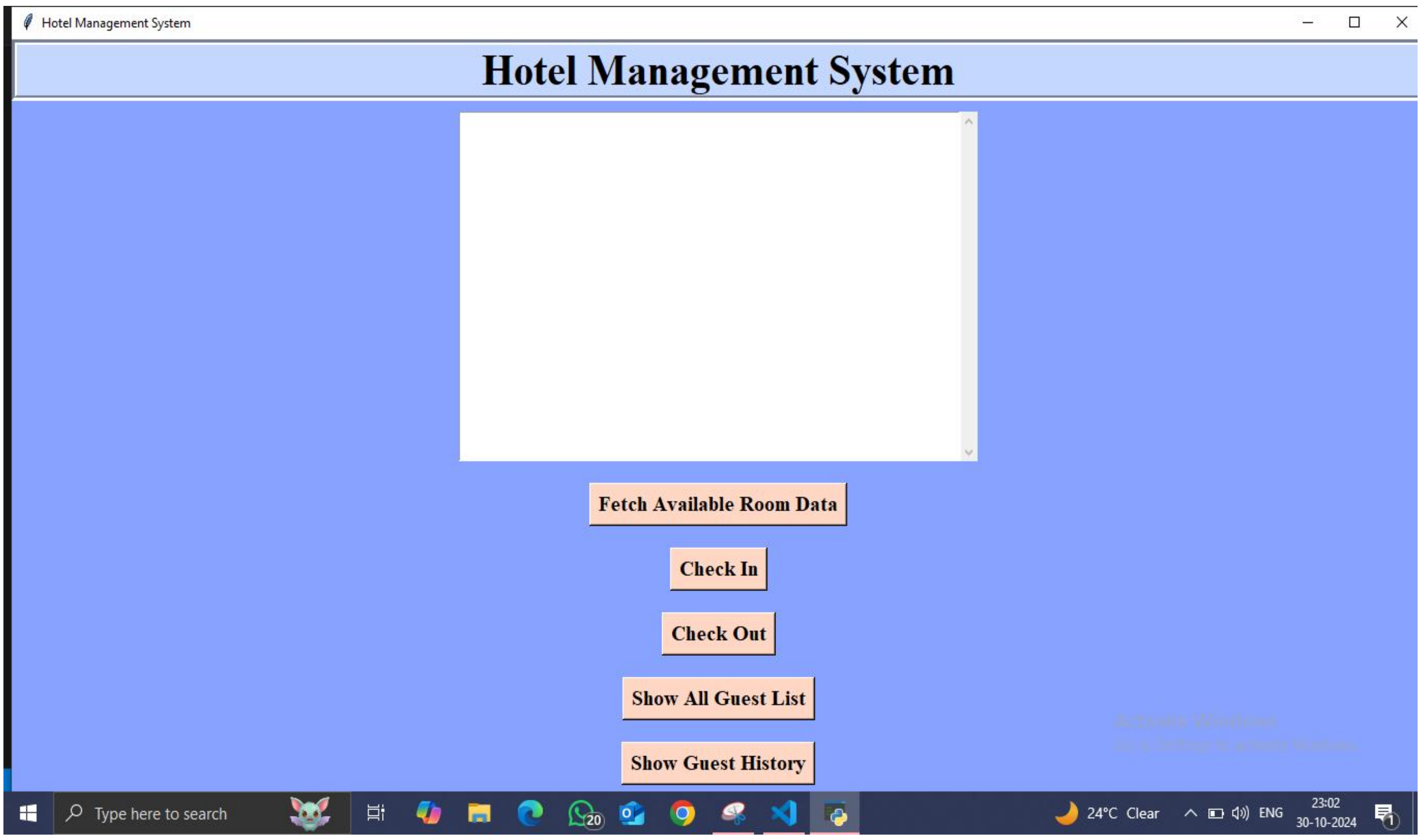
Challenges Encountered

1. **Database Synchronization:**
   One challenge was ensuring that the database and the GUI remained synchronized, especially in cases where multiple bookings could occur simultaneously.

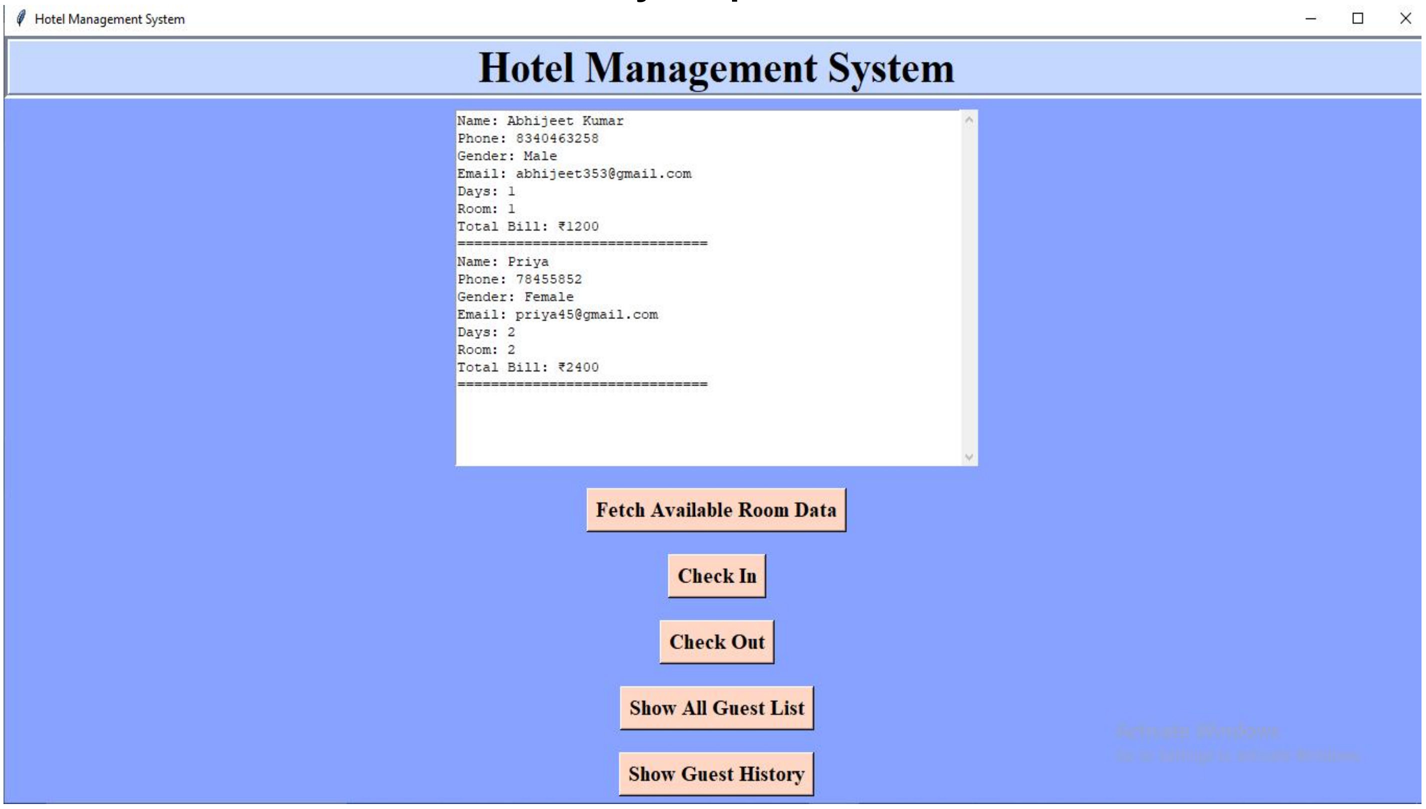
2. **Error Handling:**
   Ensuring that the system properly handles edge cases, such as trying to book rooms when none are available, was critical for a smooth user experience.

3. **GUI Design:**
   Designing a GUI that is simple enough for non-technical hotel staff while ensuring it handles all the necessary functionalities was a key challenge.

Output:



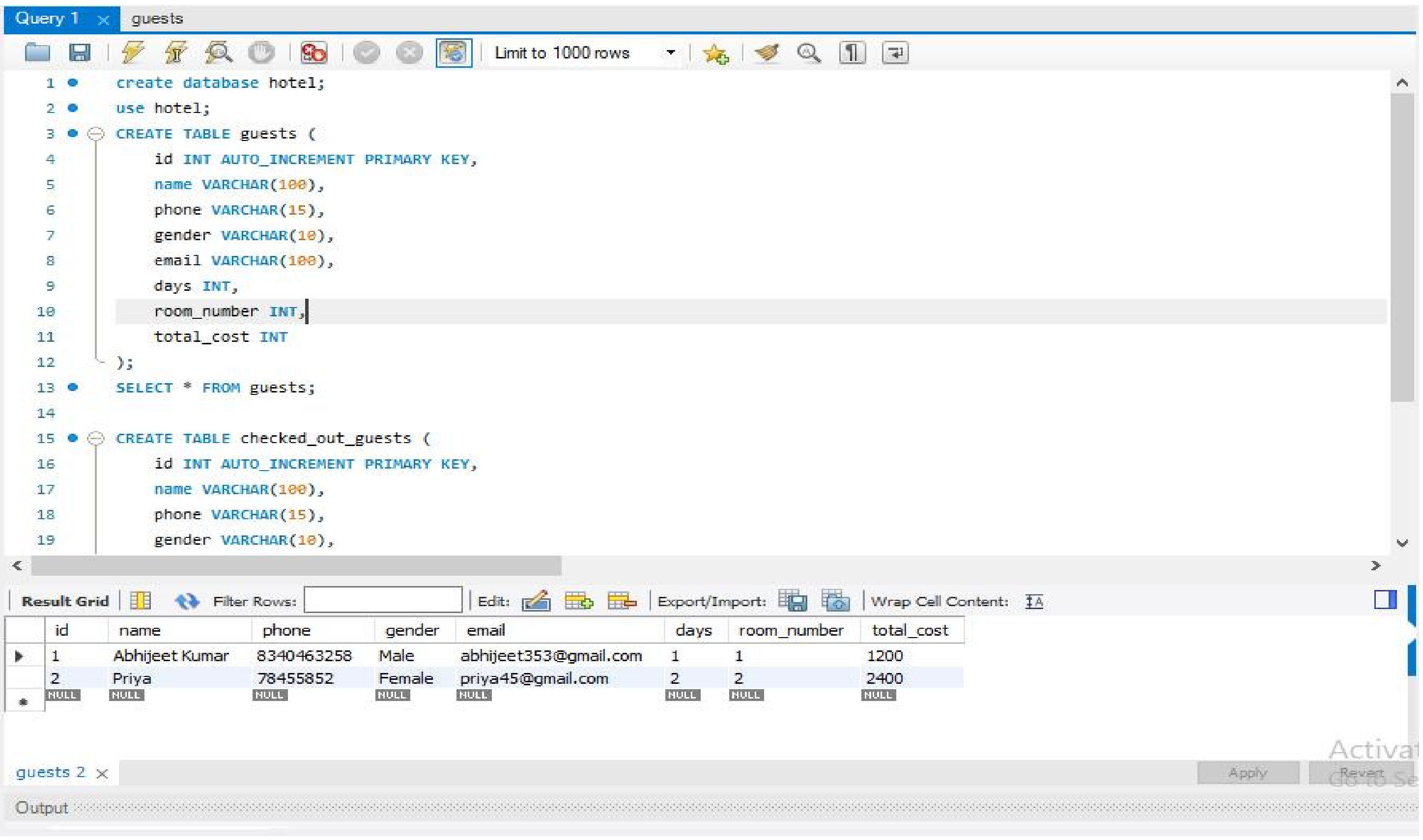
Entry Output:



Hotel Management System

```
Name: Abhijeet Kumar
Phone: 8340463258
Gender: Male
Email: abhijeet353@gmail.com
Days: 1
Room: 1
Total Bill: ₹1200
==============================
Name: Priya
Phone: 78455852
Gender: Female
Email: priya45@gmail.com
Days: 2
Room: 2
Total Bill: ₹2400
==============================
```

SQL Output:



```
create database hotel;
use hotel;
CREATE TABLE guests (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    phone VARCHAR(15),
    gender VARCHAR(10),
    email VARCHAR(100),
    days INT,
    room_number INT,
    total_cost INT
);
SELECT * FROM guests;

CREATE TABLE checked_out_guests (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    phone VARCHAR(15),
    gender VARCHAR(10),
```

id	name	phone	gender	email	days	room_number	total_cost
1	Abhijeet Kumar	8340463258	Male	abhijeet353@gmail.com	1	1	1200
2	Priya	78455852	Female	priya45@gmail.com	2	2	2400
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Conclusion

The Hotel Management System proved to be a transformative tool for Glamorous Getaways Hotel, addressing key operational challenges in booking management, billing, and room availability. By automating previously manual processes, the system enhanced efficiency, minimized human error, and improved the customer experience. Staff could now quickly check room availability, generate accurate bills, and manage customer information, creating a seamless workflow that allowed them to focus on providing quality service. This case demonstrates how a custom-built, user-friendly solution tailored to the specific needs of a small hotel can significantly boost productivity, reduce costs, and elevate customer satisfaction, making it a valuable asset for any boutique or independent hotel operation.

Learning Outcomes

Implementing the Hotel Management System for Glamorous Getaways Hotel provided several key insights and takeaways:

1. **Importance of Automation**: Automating routine tasks such as booking management and billing significantly reduces time and labor costs, highlighting the value of technology in streamlining operations in the hospitality industry.
2. **Data Accuracy and Consistency**: With automated calculations and real-time data updates, the risk of human error is minimized, leading to more reliable records and billing. This reinforces the importance of accuracy in customer-facing businesses where errors can impact customer satisfaction and trust.

3. **User-Friendly Interface Design**: A simple, intuitive interface improves staff efficiency, demonstrating how usability is crucial for technology adoption, especially in businesses with minimal technical staff training.
4. **Enhanced Customer Experience**: Real-time availability tracking and quick billing processes contribute to a smooth, hassle-free experience for guests, emphasizing how internal efficiency directly impacts customer satisfaction.

References

1. **Python Tkinter Documentation**
   Python Software Foundation. (n.d.). *Tkinter — Python interface to Tcl/Tk*. Retrieved from https://docs.python.org/3/library/tkinter.html

2. **MySQL Database Documentation**
   Oracle Corporation. (n.d.). *MySQL 8.0 Reference Manual*. Retrieved from https://dev.mysql.com/doc/

3. **Hotel Management Best Practices**
   Hospitality Net. (2022). *Hotel Operations Management Best Practices for Efficiency*. Retrieved from https://www.hospitalitynet.org/
4. **Case Studies in Hotel Automation**
   Varma, A., & Sharma, S. (2020). *Automation in the Hospitality Industry: Benefits and Case Studies*. Journal of Hospitality and Tourism Technology, 11(3), 245-259. Retrieved from https://doi.org/10.1108/JHTT-09-2019-0117

GitHub Link - https://github.com/abhijeet8340/Hotel-Management-System