**Final Project Report**

# 

Course: CSIS 3300 070

Submitted By:

300352195 Tushar Bhatia

300375339 Gurkanwal Singh

300357973 Saksham Vasudev

300377652 Lovish Dhanda

Table of Contents

[Introduction 4](#_Toc173621716)

[Create Tables(Script) 4](#_Toc173621717)

[Admin 9](#_Toc173621718)

[Create 9](#_Toc173621719)

[Read 9](#_Toc173621720)

[Update 11](#_Toc173621721)

[Delete 11](#_Toc173621722)

[Admin Methods (script) 11](#_Toc173621723)

[Customer 18](#_Toc173621724)

[Create 18](#_Toc173621725)

[Read 19](#_Toc173621726)

[Update 21](#_Toc173621727)

[Delete 21](#_Toc173621728)

[Customer Methods (script) 21](#_Toc173621729)

[Manager 29](#_Toc173621730)

[Create 29](#_Toc173621731)

[Read 29](#_Toc173621732)

[Update 29](#_Toc173621733)

[Delete 30](#_Toc173621734)

[Other 30](#_Toc173621735)

[Manager Methods(script) 31](#_Toc173621736)

[Staff 39](#_Toc173621737)

[Create 39](#_Toc173621738)

[Read 40](#_Toc173621739)

[Update 41](#_Toc173621740)

[Delete 42](#_Toc173621741)

[Staff Methods (script) 42](#_Toc173621742)

[Menu System 49](#_Toc173621743)

[User Registration and Login 49](#_Toc173621744)

[Role-Based Menus 50](#_Toc173621745)

[Menu System(Script) 53](#_Toc173621746)

[Database Connection (script) 55](#_Toc173621747)

[MongoDB Table Structure/Document Structure 56](#_Toc173621748)

[Room Booking Status Cache 56](#_Toc173621749)

[Check In status 56](#_Toc173621750)

[Room Service Request Cache 56](#_Toc173621751)

[Services Cache 57](#_Toc173621752)

[ER Diagram 57](#_Toc173621753)

[Video Link 58](#_Toc173621754)

[Conclusion 58](#_Toc173621755)

# Introduction

This report details the implementation of Starline hotel management system using MySQL and MongoDB, focusing on CRUD (Create, Read, Update, Delete) functionality for different user roles: Admin, Customer, Manager, and Staff. Each section outlines the relevant methods and database interactions for each role.

# Create Tables(Script)

import mysql.connector

from db\_connection import create\_connection

def create\_database():

connection = mysql.connector.connect(

host='localhost',

user='Saksham',

password='Saksham'

)

cursor = connection.cursor()

cursor.execute("CREATE DATABASE IF NOT EXISTS StarlightHotel")

cursor.close()

connection.close()

print("Database 'StarlightHotel' checked and created if not exists.")

def create\_connection\_with\_db():

create\_database()

return create\_connection()

def check\_and\_create\_table(connection, create\_table\_sql, table\_name):

if connection is not None and connection.is\_connected():

cursor = connection.cursor()

cursor.execute(f"SELECT COUNT(\*) FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_SCHEMA = 'StarlightHotel' AND TABLE\_NAME = '{table\_name}';")

if cursor.fetchone()[0] == 1:

print(f"Table '{table\_name}' already exists.")

else:

cursor.execute(create\_table\_sql)

connection.commit()

print(f"Table '{table\_name}' created successfully.")

cursor.close()

else:

print("Failed to establish a database connection.")

def create\_all\_tables(connection):

if connection is not None and connection.is\_connected():

create\_users\_table(connection)

create\_rooms\_table(connection)

create\_bookings\_table(connection)

create\_services\_table(connection)

create\_room\_services\_table(connection)

insert\_rooms\_data(connection)

insert\_services\_data(connection)

insert\_users\_data(connection)

connection.close()

else:

print("Failed to connect to the database.")

def create\_users\_table(connection):

sql = """

CREATE TABLE IF NOT EXISTS Users (

UserID INT AUTO\_INCREMENT PRIMARY KEY,

Username VARCHAR(255) NOT NULL UNIQUE,

Password VARCHAR(255) NOT NULL,

Role ENUM('Admin', 'Manager', 'Staff', 'Customer') NOT NULL,

FirstName VARCHAR(255),

LastName VARCHAR(255),

Email VARCHAR(255) UNIQUE,

Phone VARCHAR(20)

);

"""

check\_and\_create\_table(connection, sql, 'Users')

def create\_rooms\_table(connection):

sql = """

CREATE TABLE IF NOT EXISTS Rooms (

RoomID INT AUTO\_INCREMENT PRIMARY KEY,

RoomType ENUM('Single', 'Double', 'Suite') NOT NULL,

Availability ENUM('Available', 'Booked', 'Maintenance') NOT NULL DEFAULT 'Available',

Price DECIMAL(10,2) NOT NULL

);

"""

check\_and\_create\_table(connection, sql, 'Rooms')

def create\_bookings\_table(connection):

sql = """

CREATE TABLE IF NOT EXISTS Bookings (

BookingID INT AUTO\_INCREMENT PRIMARY KEY,

UserID INT,

RoomID INT,

BookingStatus ENUM('Pending', 'Approved', 'Rejected', 'Checked In', 'Checked Out', 'Cancelled') NOT NULL DEFAULT 'Pending',

CheckInDate DATE NOT NULL,

CheckOutDate DATE NOT NULL,

CheckedInTime DATETIME NULL,

CheckedOutTime DATETIME NULL,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

TotalCost DECIMAL(10,2) NULL,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)

);

"""

check\_and\_create\_table(connection, sql, 'Bookings')

def create\_services\_table(connection):

sql = """

CREATE TABLE IF NOT EXISTS Services (

ServiceID INT AUTO\_INCREMENT PRIMARY KEY,

ServiceType VARCHAR(255) NOT NULL,

Description TEXT,

Price DECIMAL(10,2)

);

"""

check\_and\_create\_table(connection, sql, 'Services')

def create\_room\_services\_table(connection):

sql = """

CREATE TABLE IF NOT EXISTS RoomServices (

RoomServiceID INT AUTO\_INCREMENT PRIMARY KEY,

BookingID INT,

ServiceID INT,

Status ENUM('Requested', 'Completed') NOT NULL DEFAULT 'Requested',

RequestTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CompletionTime TIMESTAMP,

CompletedByStaffID INT NULL,

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID),

FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID),

FOREIGN KEY (CompletedByStaffID) REFERENCES Users(UserID)

);

"""

check\_and\_create\_table(connection, sql, 'RoomServices')

def check\_table\_empty(connection, table\_name):

"""

Check if a given table is empty.

"""

try:

cursor = connection.cursor()

cursor.execute(f"SELECT COUNT(\*) FROM {table\_name}")

count = cursor.fetchone()[0]

return count == 0

except mysql.connector.Error as err:

print(f"Error: {err}")

return False

finally:

cursor.close()

def insert\_rooms\_data(connection):

if check\_table\_empty(connection, 'Rooms'):

sql = """

INSERT INTO Rooms (RoomID, RoomType, Availability, Price)

VALUES

(101, 'Single', 'Available', 100.00),

(102, 'Single', 'Available', 100.00),

(103, 'Single', 'Booked', 100.00),

(104, 'Double', 'Available', 150.00),

(105, 'Double', 'Available', 150.00),

(106, 'Double', 'Maintenance', 150.00),

(201, 'Suite', 'Available', 250.00),

(202, 'Suite', 'Available', 250.00),

(203, 'Suite', 'Booked', 250.00),

(204, 'Suite', 'Maintenance', 250.00),

(301, 'Single', 'Available', 120.00),

(302, 'Single', 'Booked', 120.00),

(303, 'Double', 'Available', 170.00),

(304, 'Double', 'Booked', 170.00),

(305, 'Suite', 'Maintenance', 270.00);

"""

try:

cursor = connection.cursor()

cursor.execute(sql)

connection.commit()

print("Rooms data inserted successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

else:

print("Rooms table already contains data. Insertion skipped.")

def insert\_services\_data(connection):

if check\_table\_empty(connection, 'Services'):

sql = """

INSERT INTO Services (ServiceID, ServiceType, Description, Price)

VALUES

(1, 'Room Cleaning', 'Daily room cleaning service', 20.00),

(2, 'Laundry', 'Laundry service', 15.00),

(3, 'Spa', 'Spa service', 50.00),

(4, 'Gym', 'Access to gym facilities', 10.00),

(5, 'Breakfast', 'Daily breakfast', 25.00),

(6, 'Dinner', 'Daily dinner', 35.00),

(7, 'Parking', 'Secure parking service', 10.00),

(8, 'Internet', 'High-speed internet access', 5.00),

(9, 'Mini Bar', 'In-room mini bar access', 30.00),

(10, 'Pool', 'Access to swimming pool', 15.00);

"""

try:

cursor = connection.cursor()

cursor.execute(sql)

connection.commit()

print("Services data inserted successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

else:

print("Services table already contains data. Insertion skipped.")

def insert\_users\_data(connection):

if check\_table\_empty(connection, 'Users'):

sql = """

INSERT INTO Users (UserID, Username, Password, Role, FirstName, LastName, Email, Phone)

VALUES

(1, 'admin', 'admin', 'Admin', 'Saksham', 'Vasudev', 'admin@example.com', '1234567890'),

(2, 'manager', 'manager', 'Manager', 'Tushar', 'Bhatia', 'manager@example.com', '1234567890'),

(3, 'staff', 'staff', 'Staff', 'Lovish', 'Dhanda', 'staff@example.com', '1234567890'),

(4, 'customer', 'customer', 'Customer', 'Gurkanwal', 'Singh', 'customer@example.com', '1234567890');

"""

try:

cursor = connection.cursor()

cursor.execute(sql)

connection.commit()

print("Users data inserted successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

else:

print("Users table already contains data. Insertion skipped.")

if \_\_name\_\_ == '\_\_main\_\_':

connection = create\_connection\_with\_db()

create\_all\_tables(connection)

# Admin

## Create

* **add\_user(connection)**
  + **Purpose**: Adds a new user to the Users table.
  + **Procedure**: Admin enters user details, and the function inserts the new user into the database.

## Read

* **view\_all\_users(connection)**
  + **Purpose**: Displays all users.
  + **Procedure**: Executes a SELECT statement to retrieve and display all users.
* **view\_all\_bookings(connection)**
  + **Purpose**: Displays all bookings.
  + **Procedure**: Executes a SELECT statement to retrieve and display all bookings.
* **view\_bookings\_by\_first\_name(connection)**
  + **Purpose**: Displays bookings filtered by the customer's first name.
  + **Procedure**: Admin enters the first name, and the function retrieves matching bookings.
* **generate\_financial\_reports(connection)**
  + **Purpose**: Generates financial reports for a specified period.
  + **Procedure**: Admin enters start and end dates, and the function calculates and displays revenue.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

## Update

* **update\_user(connection)**
  + **Purpose**: Updates user details.
  + **Procedure**: Admin selects a user and detail to update, and the function modifies the database accordingly.

## Delete

* **delete\_user(connection)**
  + **Purpose**: Deletes a user.
  + **Procedure**: Admin enters the username of the user to be deleted, and the function removes the user from the database.

## Admin Methods (script)

from db\_connection import create\_connection

import mysql.connector

import getpass

from datetime import datetime, timedelta

def add\_user(connection):

cursor = connection.cursor()

username = input("Enter Username: ")

password = getpass.getpass("Enter Password: ")

role = input("Enter Role (Admin, Manager, Staff, Customer): ")

first\_name = input("Enter First Name: ")

last\_name = input("Enter Last Name: ")

email = input("Enter Email: ")

phone = input("Enter Phone: ")

sql = """

INSERT INTO Users (Username, Password, Role, FirstName, LastName, Email, Phone)

VALUES (%s, %s, %s, %s, %s, %s, %s)

"""

values = (username, password, role, first\_name, last\_name, email, phone)

try:

cursor.execute(sql, values)

connection.commit()

print("User added successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

def update\_user(connection):

cursor = connection.cursor()

username = input("Enter the Username of the user to update: ")

cursor.execute("SELECT \* FROM Users WHERE Username = %s", (username,))

user = cursor.fetchone()

if not user:

print("User not found.")

return

while True:

print("\nWhat do you want to update?")

print("1. Username")

print("2. Password")

print("3. Role")

print("4. First Name")

print("5. Last Name")

print("6. Email")

print("7. Phone")

print("8. Exit")

choice = input("Enter your choice: ")

if choice == '1':

new\_value = input("Enter new Username: ")

field = 'Username'

elif choice == '2':

new\_value = getpass.getpass("Enter new Password: ")

field = 'Password'

elif choice == '3':

new\_value = input("Enter new Role (Admin, Manager, Staff, Customer): ")

field = 'Role'

elif choice == '4':

new\_value = input("Enter new First Name: ")

field = 'FirstName'

elif choice == '5':

new\_value = input("Enter new Last Name: ")

field = 'LastName'

elif choice == '6':

new\_value = input("Enter new Email: ")

field = 'Email'

elif choice == '7':

new\_value = input("Enter new Phone: ")

field = 'Phone'

elif choice == '8':

print("Exiting update menu.")

break

else:

print("Invalid choice. Please try again.")

continue

sql = f"UPDATE Users SET {field} = %s WHERE Username = %s"

values = (new\_value, username)

try:

cursor.execute(sql, values)

connection.commit()

print(f"{field} updated successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

cursor.close()

def view\_all\_users(connection):

cursor = connection.cursor()

sql = "SELECT UserID, Username, Role, FirstName, LastName, Email, Phone FROM Users"

cursor.execute(sql)

users = cursor.fetchall()

if users:

print("\n--- All Users ---")

for user in users:

print(f"\n\nUserID: {user[0]}, \nUsername: {user[1]}, \nRole: {user[2]}, \nFirstName: {user[3]}, \nLastName: {user[4]}, \nEmail: {user[5]}, \nPhone: {user[6]}")

input("\nEnter to go to the Main Menu")

else:

print("No users found.")

cursor.close()

def delete\_user(connection):

cursor = connection.cursor()

view\_all\_users(connection);

user\_name = input("\n\nEnter Username to delete: ")

sql = "DELETE FROM Users WHERE Username = %s"

values = (user\_name,)

try:

cursor.execute(sql, values)

connection.commit()

print("User deleted successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

def view\_all\_bookings(connection):

cursor = connection.cursor()

sql = """

SELECT b.BookingID, u.Username, r.RoomID, b.BookingStatus, b.CheckInDate, b.CheckOutDate, b.CheckedInTime, b.CheckedOutTime, b.CreatedAt

FROM Bookings b

JOIN Users u ON b.UserID = u.UserID

JOIN Rooms r ON b.RoomID = r.RoomID

"""

cursor.execute(sql)

bookings = cursor.fetchall()

if bookings:

print("\n--- All Bookings ---")

print(f"{'Booking ID':<12} | {'Username':<15} | {'Room ID':<10} | {'Status':<12} | {'Check-In Date':<15} | {'Check-Out Date':<15} | {'Checked In Time':<20} | {'Checked Out Time':<20} | {'Created At':<20}")

print("-" \* 180)

for booking in bookings:

booking\_id, username, room\_id, status, check\_in\_date, check\_out\_date, checked\_in\_time, checked\_out\_time, created\_at = booking

check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'

check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'

checked\_in\_time = checked\_in\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_in\_time else 'N/A'

checked\_out\_time = checked\_out\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_out\_time else 'N/A'

created\_at = created\_at.strftime('%Y-%m-%d %H:%M:%S') if created\_at else 'N/A'

print(f"{booking\_id:<12} | {username:<15} | {room\_id:<10} | {status:<12} | {check\_in\_date:<15} | {check\_out\_date:<15} | {checked\_in\_time:<20} | {checked\_out\_time:<20} | {created\_at:<20}")

print("-" \* 180)

else:

print("No bookings found.")

cursor.close()

def view\_bookings\_by\_first\_name(connection):

cursor = connection.cursor()

first\_name = input("Enter the first name of the person to see their bookings: ")

sql\_check\_person = "SELECT UserID FROM Users WHERE FirstName = %s"

cursor.execute(sql\_check\_person, (first\_name,))

user\_ids = cursor.fetchall()

if not user\_ids:

print(f"No person found with the first name '{first\_name}'.")

cursor.close()

return

user\_ids = [user\_id[0] for user\_id in user\_ids]

format\_strings = ','.join(['%s'] \* len(user\_ids))

sql\_view\_bookings = f"""

SELECT b.BookingID, u.Username, r.RoomID, b.BookingStatus, b.CheckInDate, b.CheckOutDate, b.CheckedInTime, b.CheckedOutTime, b.CreatedAt, b.TotalCost

FROM Bookings b

JOIN Users u ON b.UserID = u.UserID

JOIN Rooms r ON b.RoomID = r.RoomID

WHERE b.UserID IN ({format\_strings})

"""

cursor.execute(sql\_view\_bookings, tuple(user\_ids))

bookings = cursor.fetchall()

if bookings:

print("\n--- Bookings for First Name: {} ---".format(first\_name))

print(f"{'Booking ID':<12} | {'Username':<15} | {'Room ID':<10} | {'Status':<12} | {'Check-In Date':<15} | {'Check-Out Date':<15} | {'Checked In Time':<20} | {'Checked Out Time':<20} | {'Created At':<20} | {'Total Cost':<10}")

print("-" \* 180)

for booking in bookings:

booking\_id, username, room\_id, status, check\_in\_date, check\_out\_date, checked\_in\_time, checked\_out\_time, created\_at, total\_cost = booking

check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'

check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'

checked\_in\_time = checked\_in\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_in\_time else 'N/A'

checked\_out\_time = checked\_out\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_out\_time else 'N/A'

created\_at = created\_at.strftime('%Y-%m-%d %H:%M:%S') if created\_at else 'N/A'

total\_cost = f"${total\_cost:.2f}" if total\_cost is not None else 'N/A'

print(f"{booking\_id:<12} | {username:<15} | {room\_id:<10} | {status:<12} | {check\_in\_date:<15} | {check\_out\_date:<15} | {checked\_in\_time:<20} | {checked\_out\_time:<20} | {created\_at:<20} | {total\_cost:<10}")

print("-" \* 180)

else:

print(f"No bookings found for the person with the first name '{first\_name}'.")

cursor.close()

def generate\_financial\_reports(connection):

cursor = connection.cursor()

print("\n--- Generate Financial Reports ---")

print("1. Daily Report")

print("2. Monthly Report")

print("3. Custom Date Range Report")

choice = input("Enter your choice: ")

if choice == '1':

start\_date = datetime.now().date()

end\_date = start\_date

elif choice == '2':

start\_date = datetime.now().replace(day=1).date()

end\_date = (start\_date + timedelta(days=32)).replace(day=1) - timedelta(days=1)

elif choice == '3':

start\_date\_str = input("Enter start date (YYYY-MM-DD): ")

end\_date\_str = input("Enter end date (YYYY-MM-DD): ")

try:

start\_date = datetime.strptime(start\_date\_str, '%Y-%m-%d').date()

end\_date = datetime.strptime(end\_date\_str, '%Y-%m-%d').date()

except ValueError:

print("Invalid date format. Please try again.")

cursor.close()

return

else:

print("Invalid choice. Please try again.")

cursor.close()

return

financial\_summary\_query = """

SELECT

COALESCE(SUM(b.TotalCost), 0) AS TotalRoomRevenue,

COALESCE(SUM(CASE WHEN rs.Status = 'Completed' THEN s.Price ELSE 0 END), 0) AS TotalServiceRevenue

FROM Bookings b

LEFT JOIN RoomServices rs ON b.BookingID = rs.BookingID

LEFT JOIN Services s ON rs.ServiceID = s.ServiceID

WHERE (DATE(b.CreatedAt) BETWEEN %s AND %s)

OR (DATE(rs.RequestTime) BETWEEN %s AND %s)

OR (DATE(rs.CompletionTime) BETWEEN %s AND %s)

"""

cursor.execute(financial\_summary\_query, (start\_date, end\_date, start\_date, end\_date, start\_date, end\_date))

financial\_summary = cursor.fetchone()

total\_room\_revenue = financial\_summary[0]

total\_service\_revenue = financial\_summary[1]

total\_revenue = total\_room\_revenue + total\_service\_revenue

print("\n" + "="\*50)

print(" Financial Report")

print("="\*50)

print(f"Report Period: {start\_date} to {end\_date}")

print("-" \* 50)

print(f"Total Revenue: ${total\_revenue:.2f}")

print(f"Revenue from Room Bookings: ${total\_room\_revenue:.2f}")

print(f"Revenue from Services: ${total\_service\_revenue:.2f}")

print("="\*50)

print("Press Enter to go back to the main menu")

input()

cursor.close()

def admin\_menu(connection):

while True:

print("\n--- Admin Menu ---")

print("1. View All Users")

print("2. Add User")

print("3. Update User")

print("4. Delete User")

print("5. View All Bookings")

print("6. View Bookings by First Name")

print("7. Generate Financial Reports")

print("8. Sign Out / Switch User")

print("9. Exit")

choice = input("Enter your choice: ")

if choice == '1':

view\_all\_users(connection)

elif choice == '2':

add\_user(connection)

elif choice == '3':

update\_user(connection)

elif choice == '4':

delete\_user(connection)

elif choice == '5':

view\_all\_bookings(connection)

elif choice == '6':

view\_bookings\_by\_first\_name(connection)

elif choice == '7':

generate\_financial\_reports(connection)

elif choice == '8':

print("Signing out...")

return 'signout'

elif choice == '9':

print("Exiting...")

return 'exit'

else:

print("Invalid choice. Please try again.")

# Customer

## Create

* **book\_room(connection, user\_id)**
  + **Purpose**: Allows a customer to book a room.
  + **Procedure**: Customer selects room and dates, and the function inserts the booking into the database.

A screenshot of a computer

Description automatically generated

* **request\_room\_service(connection, user\_id)**
  + **Purpose**: Allows a customer to request room service.
  + **Procedure**: Customer selects a service for an approved booking, and the function inserts the request into the database.

A screenshot of a computer

Description automatically generated

## Read

* **view\_services(connection)**
  + **Purpose**: Displays all available services.
  + **Procedure**: Executes a SELECT statement to retrieve and display services.

A screen shot of a computer

Description automatically generated

* **view\_room\_availability(connection)**
  + **Purpose**: Shows all available rooms.
  + **Procedure**: Executes a SELECT statement to retrieve and display room availability.
* **view\_customer\_bookings(connection, user\_id)**
  + **Purpose**: Lists all bookings made by the customer.
  + **Procedure**: Executes a SELECT statement to retrieve and display the customer's bookings.
* **view\_services\_for\_booking(connection, booking\_id)**
  + **Purpose**: Displays services requested for a specific booking.
  + **Procedure**: Executes a SELECT statement to retrieve and display the services for a booking.

A screenshot of a computer screen

Description automatically generated

* **view\_customer\_approved\_bookings(connection, user\_id)**
  + **Purpose**: Shows all approved bookings for the customer.
  + **Procedure**: Executes a SELECT statement to retrieve and display approved bookings.

## Update

* Not applicable for customers.

## Delete

* **cancel\_booking(connection, user\_id)**
  + **Purpose**: Enables a customer to cancel a booking.
  + **Procedure**: Customer specifies the booking ID, and the function deletes the booking from the database.

## Customer Methods (script)

* from db\_connection import create\_connection
* import mysql.connector
* from datetime import datetime
* def view\_services(connection):
* cursor = connection.cursor()
* sql = "SELECT ServiceID, ServiceType, Description, Price FROM Services"
* cursor.execute(sql)
* services = cursor.fetchall()
* if services:
* print("\n" + "="\*60)
* print(" Available Services")
* print("="\*60)
* print(f"{'Service ID':<12} | {'Service Type':<20} | {'Price':<10} | {'Description':<20}")
* print("-" \* 60)
* for service in services:
* service\_id, service\_type, description, price = service
* print(f"{service\_id:<12} | {service\_type:<20} | ${price:<10.2f} | {description:<20}")
* print("="\*60 + "\n")
* else:
* print("No services available.")
* cursor.close()
* def view\_room\_availability(connection):
* cursor = connection.cursor()
* sql = "SELECT RoomID, RoomType, Price FROM Rooms WHERE Availability = 'Available'"
* cursor.execute(sql)
* rooms = cursor.fetchall()
* if rooms:
* print("\n" + "="\*50)
* print(" Available Rooms")
* print("="\*50)
* print(f"{'Room Number':<10} | {'Room Type':<15} | {'Price':<10}")
* print("-" \* 50)
* for room in rooms:
* room\_id, room\_type, price = room
* print(f"{room\_id:<10} | {room\_type:<15} | ${price:<10.2f}")
* print("="\*50 + "\n")
* print("Press Enter to go to menu")
* input()
* else:
* print("No rooms available.")
* cursor.close()
* def book\_room(connection, user\_id):
* cursor = connection.cursor()
* view\_room\_availability(connection)
* while True:
* check\_in\_date\_str = input("Enter Check-In Date (YYYY-MM-DD): ")
* check\_out\_date\_str = input("Enter Check-Out Date (YYYY-MM-DD): ")
* try:
* check\_in\_date = datetime.strptime(check\_in\_date\_str, '%Y-%m-%d').date()
* check\_out\_date = datetime.strptime(check\_out\_date\_str, '%Y-%m-%d').date()
* current\_date = datetime.now().date()
* if check\_in\_date < current\_date:
* print("Check-in date cannot be before the current date. Please try again.")
* continue
* if check\_out\_date <= check\_in\_date:
* print("Check-out date must be after the check-in date. Please try again.")
* continue
* break
* except ValueError:
* print("Invalid date format. Please enter the date in YYYY-MM-DD format.")
* continue
* room\_id = input("Enter Room ID: ")
* num\_days = (check\_out\_date - check\_in\_date).days
* cursor.execute("SELECT Price FROM Rooms WHERE RoomID = %s", (room\_id,))
* room\_price = cursor.fetchone()
* if not room\_price:
* print("Invalid Room ID.")
* cursor.close()
* return
* room\_price = room\_price[0]
* total\_cost = num\_days \* room\_price
* sql = """
* INSERT INTO Bookings (UserID, RoomID, CheckInDate, CheckOutDate, BookingStatus, TotalCost)
* VALUES (%s, %s, %s, %s, 'Pending', %s)
* """
* values = (user\_id, room\_id, check\_in\_date, check\_out\_date, total\_cost)
* try:
* cursor.execute(sql, values)
* connection.commit()
* print(f"Room booking request submitted. Total cost for {num\_days} days is ${total\_cost:.2f}.")
* except mysql.connector.Error as err:
* print(f"Error: {err}")
* finally:
* cursor.close()
* def view\_customer\_bookings(connection, user\_id):
* cursor = connection.cursor()
* sql = """
* SELECT b.BookingID, r.RoomID, b.BookingStatus, b.CheckInDate, b.CheckOutDate, b.CheckedInTime, b.CheckedOutTime, b.CreatedAt, b.TotalCost
* FROM Bookings b
* JOIN Rooms r ON b.RoomID = r.RoomID
* WHERE b.UserID = %s
* """
* cursor.execute(sql, (user\_id,))
* bookings = cursor.fetchall()
* if bookings:
* print("\n--- Your Bookings ---")
* print(f"{'Booking ID':<12} | {'Room ID':<10} | {'Status':<12} | {'Check-In Date':<15} | {'Check-Out Date':<15} | {'Checked In Time':<20} | {'Checked Out Time':<20} | {'Created At':<20} | {'Total Cost':<10}")
* print("-" \* 180)
* for booking in bookings:
* booking\_id, room\_id, status, check\_in\_date, check\_out\_date, checked\_in\_time, checked\_out\_time, created\_at, total\_cost = booking
* check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'
* check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'
* checked\_in\_time = checked\_in\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_in\_time else 'N/A'
* checked\_out\_time = checked\_out\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_out\_time else 'N/A'
* created\_at = created\_at.strftime('%Y-%m-%d %H:%M:%S') if created\_at else 'N/A'
* total\_cost = f"${total\_cost:.2f}" if total\_cost is not None else 'N/A'
* print(f"{booking\_id:<12} | {room\_id:<10} | {status:<12} | {check\_in\_date:<15} | {check\_out\_date:<15} | {checked\_in\_time:<20} | {checked\_out\_time:<20} | {created\_at:<20} | {total\_cost:<10}")
* print("-" \* 180)
* else:
* print("You have no bookings.")
* cursor.close()
* def view\_customer\_approved\_bookings(connection, user\_id):
* cursor = connection.cursor()
* sql = """
* SELECT b.BookingID, r.RoomID, b.BookingStatus, b.CheckInDate, b.CheckOutDate, b.CheckedInTime, b.CheckedOutTime, b.CreatedAt, b.TotalCost
* FROM Bookings b
* JOIN Rooms r ON b.RoomID = r.RoomID
* WHERE b.UserID = %s AND b.BookingStatus IN ('Approved', 'Checked In')
* """
* cursor.execute(sql, (user\_id,))
* bookings = cursor.fetchall()
* if bookings:
* print("\n--- Your Valid Bookings ---")
* print(f"{'Booking ID':<12} | {'Room ID':<10} | {'Status':<12} | {'Check-In Date':<15} | {'Check-Out Date':<15} | {'Checked In Time':<20} | {'Checked Out Time':<20} | {'Created At':<20} | {'Total Cost':<10}")
* print("-" \* 180)
* for booking in bookings:
* booking\_id, room\_id, status, check\_in\_date, check\_out\_date, checked\_in\_time, checked\_out\_time, created\_at, total\_cost = booking
* check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'
* check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'
* checked\_in\_time = checked\_in\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_in\_time else 'N/A'
* checked\_out\_time = checked\_out\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_out\_time else 'N/A'
* created\_at = created\_at.strftime('%Y-%m-%d %H:%M:%S') if created\_at else 'N/A'
* total\_cost = f"${total\_cost:.2f}" if total\_cost is not None else 'N/A'
* print(f"{booking\_id:<12} | {room\_id:<10} | {status:<12} | {check\_in\_date:<15} | {check\_out\_date:<15} | {checked\_in\_time:<20} | {checked\_out\_time:<20} | {created\_at:<20} | {total\_cost:<10}")
* print("-" \* 180)
* return True
* else:
* return False
* cursor.close()
* def cancel\_booking(connection, user\_id):
* cursor = connection.cursor()
* sql = """
* SELECT b.BookingID, r.RoomID, b.BookingStatus, b.CheckInDate, b.CheckOutDate, b.CheckedInTime, b.CheckedOutTime, b.CreatedAt, b.TotalCost
* FROM Bookings b
* JOIN Rooms r ON b.RoomID = r.RoomID
* WHERE b.UserID = %s AND b.BookingStatus IN ('Approved', 'Pending')
* """
* cursor.execute(sql, (user\_id,))
* bookings = cursor.fetchall()
* if bookings:
* print("\n--- Your Approved or Pending Bookings ---")
* print(f"{'Booking ID':<12} | {'Room ID':<10} | {'Status':<12} | {'Check-In Date':<15} | {'Check-Out Date':<15} | {'Checked In Time':<20} | {'Checked Out Time':<20} | {'Created At':<20} | {'Total Cost':<10}")
* print("-" \* 180)
* for booking in bookings:
* booking\_id, room\_id, status, check\_in\_date, check\_out\_date, checked\_in\_time, checked\_out\_time, created\_at, total\_cost = booking
* check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'
* check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'
* checked\_in\_time = checked\_in\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_in\_time else 'N/A'
* checked\_out\_time = checked\_out\_time.strftime('%Y-%m-%d %H:%M:%S') if checked\_out\_time else 'N/A'
* created\_at = created\_at.strftime('%Y-%m-%d %H:%M:%S') if created\_at else 'N/A'
* total\_cost = f"${total\_cost:.2f}" if total\_cost is not None else 'N/A'
* print(f"{booking\_id:<12} | {room\_id:<10} | {status:<12} | {check\_in\_date:<15} | {check\_out\_date:<15} | {checked\_in\_time:<20} | {checked\_out\_time:<20} | {created\_at:<20} | {total\_cost:<10}")
* print("-" \* 180)
* booking\_id = input("Enter Booking ID to cancel: ")
* sql\_check = """
* SELECT BookingID, RoomID FROM Bookings
* WHERE BookingID = %s AND UserID = %s AND BookingStatus IN ('Approved', 'Pending')
* """
* cursor.execute(sql\_check, (booking\_id, user\_id))
* valid\_booking = cursor.fetchone()
* if valid\_booking:
* room\_id = valid\_booking[1]
* sql\_cancel = "UPDATE Bookings SET BookingStatus = 'Cancelled' WHERE BookingID = %s"
* sql\_update\_room = "UPDATE Rooms SET Availability = 'Available' WHERE RoomID = %s AND Availability = 'Booked'"
* sql\_delete\_services = "DELETE FROM RoomServices WHERE BookingID = %s"
* try:
* cursor.execute(sql\_cancel, (booking\_id,))
* cursor.execute(sql\_update\_room, (room\_id,))
* cursor.execute(sql\_delete\_services, (booking\_id,))
* connection.commit()
* print("Booking cancelled and room status updated to 'Available' successfully.")
* except mysql.connector.Error as err:
* print(f"Error: {err}")
* else:
* print("Invalid Booking ID or Booking cannot be cancelled.")
* else:
* print("You have no approved or pending bookings.")
* cursor.close()
* def view\_services\_for\_booking(connection, booking\_id):
* cursor = connection.cursor()
* sql = """
* SELECT rs.RoomServiceID, s.ServiceType, rs.Status, rs.RequestTime, rs.CompletionTime, u.Username AS CompletedBy, s.Price
* FROM RoomServices rs
* JOIN Services s ON rs.ServiceID = s.ServiceID
* LEFT JOIN Users u ON rs.CompletedByStaffID = u.UserID
* WHERE rs.BookingID = %s
* """
* cursor.execute(sql, (booking\_id,))
* services = cursor.fetchall()
* if services:
* print("\n--- Services for Booking ID {} ---".format(booking\_id))
* print(f"{'Service ID':<12} | {'Service Type':<20} | {'Status':<12} | {'Request Time':<20} | {'Completion Time':<20} | {'Completed By':<15} | {'Price':<10}")
* print("-" \* 140)
* for service in services:
* service\_id, service\_type, status, request\_time, completion\_time, completed\_by, price = service
* request\_time = request\_time.strftime('%Y-%m-%d %H:%M:%S') if request\_time else 'N/A'
* completion\_time = completion\_time.strftime('%Y-%m-%d %H:%M:%S') if completion\_time else 'N/A'
* completed\_by = completed\_by if completed\_by else 'N/A'
* print(f"{service\_id:<12} | {service\_type:<20} | {status:<12} | {request\_time:<20} | {completion\_time:<20} | {completed\_by:<15} | ${price:<10.2f}")
* print("-" \* 140)
* else:
* print("No services found for this booking.")
* cursor.close()
* def request\_room\_service(connection, user\_id):
* cursor = connection.cursor()
* has\_approved\_bookings = view\_customer\_approved\_bookings(connection, user\_id)
* if not has\_approved\_bookings:
* print("You have no approved bookings to request room service.")
* cursor.close()
* return
* booking\_id = input("Enter Booking ID to request room service: ")
* view\_services(connection)
* service\_id = input("Enter Service ID: ")
* sql = """
* INSERT INTO RoomServices (BookingID, ServiceID, Status)
* VALUES (%s, %s, 'Requested')
* """
* values = (booking\_id, service\_id)
* try:
* cursor.execute(sql, values)
* connection.commit()
* print("Room service requested successfully.")
* except mysql.connector.Error as err:
* print(f"Error: {err}")
* finally:
* cursor.close()
* def customer\_menu(connection, user\_id):
* while True:
* print("\n--- Customer Menu ---")
* print("1. View Available Rooms")
* print("2. Book Room")
* print("3. Request Room Service")
* print("4. View My Bookings")
* print("5. Cancel Booking")
* print("6. View Services for Booking")
* print("7. Sign Out / Switch User")
* print("8. Exit")
* choice = input("Enter your choice: ")
* if choice == '1':
* view\_room\_availability(connection)
* elif choice == '2':
* book\_room(connection, user\_id)
* elif choice == '3':
* request\_room\_service(connection, user\_id)
* elif choice == '4':
* view\_customer\_bookings(connection, user\_id)
* elif choice == '5':
* cancel\_booking(connection, user\_id)
* elif choice == '6':
* if view\_customer\_approved\_bookings(connection, user\_id):
* booking\_id = input("Enter Booking ID to view services: ")
* view\_services\_for\_booking(connection, booking\_id)
* elif choice == '7':
* print("Signing out...")
* return 'signout'
* elif choice == '8':
* print("Exiting...")
* return 'exit'
* else:
* print("Invalid choice. Please try again.")

# Manager

## Create

* Not applicable for managers.

## Read

* **view\_pending\_bookings(connection)**
  + **Purpose**: Displays a list of pending bookings.
  + **Procedure**: Executes a SELECT statement to retrieve and display pending bookings.
* **view\_room\_availability(connection)**
  + **Purpose**: Displays room availability.
  + **Procedure**: Executes a SELECT statement to retrieve and display room availability.

## Update

* **approve\_booking(connection, booking\_id)**
  + **Purpose**: Approves a booking.
  + **Procedure**: Manager selects a booking, and the function updates its status to 'Approved'.

A screenshot of a computer

Description automatically generated

* **reject\_booking(connection, booking\_id)**
  + **Purpose**: Rejects a booking.
  + **Procedure**: Manager selects a booking, and the function updates its status to 'Rejected'.

## Delete

* Not applicable for managers.

## Other

* **generate\_daily\_activity\_reports(connection)**
  + **Purpose**: Generates daily activity reports.
  + **Procedure**: Executes various SELECT statements to generate and display the report.

A screenshot of a computer screen

Description automatically generated

* **generate\_bill(connection)**
  + **Purpose**: Generates a bill for a customer.
  + **Procedure**: Calculates room and service costs for checked-out bookings and displays the bill.

## Manager Methods(script)

from db\_connection import create\_connection

import mysql.connector

from decimal import Decimal

from datetime import datetime

def view\_pending\_bookings(connection):

cursor = connection.cursor()

sql = """

SELECT b.BookingID, u.Username, r.RoomID, r.RoomType, b.CheckInDate, b.CheckOutDate, b.BookingStatus

FROM Bookings b

JOIN Users u ON b.UserID = u.UserID

JOIN Rooms r ON b.RoomID = r.RoomID

WHERE b.BookingStatus = 'Pending'

"""

cursor.execute(sql)

pending\_bookings = cursor.fetchall()

if pending\_bookings:

print("\n--- Pending Bookings ---")

print(f"{'Booking ID':<12} | {'Username':<15} | {'Room Number':<12} | {'Room Type':<10} | {'Check-In Date':<15} | {'Check-Out Date':<15} | {'Status':<10}")

print("-" \* 110)

for booking in pending\_bookings:

booking\_id, username, room\_id, room\_type, check\_in\_date, check\_out\_date, status = booking

check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'

check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'

print(f"{booking\_id:<12} | {username:<15} | {room\_id:<12} | {room\_type:<10} | {check\_in\_date:<15} | {check\_out\_date:<15} | {status:<10}")

print("-" \* 110)

return True

else:

print("No pending bookings.")

return False

cursor.close()

def approve\_booking(connection, booking\_id):

cursor = connection.cursor()

sql\_get\_room\_id = "SELECT RoomID FROM Bookings WHERE BookingID = %s"

cursor.execute(sql\_get\_room\_id, (booking\_id,))

room\_id = cursor.fetchone()

if room\_id:

room\_id = room\_id[0]

sql\_update\_booking = "UPDATE Bookings SET BookingStatus = 'Approved' WHERE BookingID = %s"

cursor.execute(sql\_update\_booking, (booking\_id,))

sql\_update\_room = "UPDATE Rooms SET Availability = 'Booked' WHERE RoomID = %s"

cursor.execute(sql\_update\_room, (room\_id,))

connection.commit()

print("Booking approved and room status updated to 'Booked'.")

else:

print("Invalid Booking ID.")

cursor.close()

def reject\_booking(connection, booking\_id):

cursor = connection.cursor()

sql\_get\_room\_id = "SELECT RoomID FROM Bookings WHERE BookingID = %s"

cursor.execute(sql\_get\_room\_id, (booking\_id,))

room\_id = cursor.fetchone()

if room\_id:

room\_id = room\_id[0]

sql\_update\_booking = "UPDATE Bookings SET BookingStatus = 'Rejected' WHERE BookingID = %s"

cursor.execute(sql\_update\_booking, (booking\_id,))

connection.commit()

print("Booking rejected.")

else:

print("Invalid Booking ID.")

cursor.close()

def view\_room\_availability(connection):

cursor = connection.cursor()

sql = "SELECT RoomID, RoomType, Availability FROM Rooms"

cursor.execute(sql)

rooms = cursor.fetchall()

if rooms:

print("\n--- Rooms ---")

print(f"{'Room ID':<10}{'Room Type':<15}{'Availability':<15}")

print("-" \* 50)

for room in rooms:

room\_id, room\_type, availability = room

print(f"{room\_id:<10}{room\_type:<15}{availability:<15}")

print("-" \* 50)

input("Press Enter to go to the Main Menu")

else:

print("No rooms available.")

cursor.close()

def generate\_daily\_activity\_reports(connection):

cursor = connection.cursor()

current\_date = datetime.now().date()

sql\_checkins = "SELECT COUNT(\*) FROM Bookings WHERE DATE(CheckedInTime) = %s"

sql\_checkouts = "SELECT COUNT(\*) FROM Bookings WHERE DATE(CheckedOutTime) = %s"

cursor.execute(sql\_checkins, (current\_date,))

checkins\_count = cursor.fetchone()[0]

cursor.execute(sql\_checkouts, (current\_date,))

checkouts\_count = cursor.fetchone()[0]

sql\_bookings = """

SELECT BookingStatus, COUNT(\*)

FROM Bookings

WHERE DATE(CreatedAt) = %s

GROUP BY BookingStatus

"""

cursor.execute(sql\_bookings, (current\_date,))

bookings\_summary = cursor.fetchall()

sql\_room\_availability = """

SELECT Availability, COUNT(\*)

FROM Rooms

GROUP BY Availability

"""

cursor.execute(sql\_room\_availability)

room\_availability = cursor.fetchall()

sql\_room\_services = """

SELECT Status, COUNT(\*)

FROM RoomServices

WHERE DATE(RequestTime) = %s OR DATE(CompletionTime) = %s

GROUP BY Status

"""

cursor.execute(sql\_room\_services, (current\_date, current\_date))

room\_services\_summary = cursor.fetchall()

sql\_financial\_summary = """

SELECT

SUM(b.TotalCost) AS TotalRoomRevenue,

SUM(CASE WHEN rs.Status = 'Completed' THEN s.Price ELSE 0 END) AS TotalServiceRevenue,

SUM(b.TotalCost + CASE WHEN rs.Status = 'Completed' THEN s.Price ELSE 0 END) AS TotalRevenue

FROM Bookings b

LEFT JOIN RoomServices rs ON b.BookingID = rs.BookingID

LEFT JOIN Services s ON rs.ServiceID = s.ServiceID

WHERE DATE(b.CreatedAt) = %s

"""

cursor.execute(sql\_financial\_summary, (current\_date,))

financial\_summary = cursor.fetchone()

total\_room\_revenue = financial\_summary[0] if financial\_summary[0] is not None else 0.0

total\_service\_revenue = financial\_summary[1] if financial\_summary[1] is not None else 0.0

total\_revenue = financial\_summary[2] if financial\_summary[2] is not None else 0.0

print("\n" + "="\*50)

print(" Daily Activity Report")

print("="\*50)

print(f"Date: {current\_date}")

print("-" \* 50)

print(f"Check-ins Today: {checkins\_count}")

print(f"Check-outs Today: {checkouts\_count}")

print("\nBookings Summary:")

for status, count in bookings\_summary:

print(f"{status}: {count}")

print("\nRoom Availability:")

for availability, count in room\_availability:

print(f"{availability}: {count}")

print("\nRoom Services Summary:")

for status, count in room\_services\_summary:

print(f"{status}: {count}")

print(f"\nTotal Revenue: ${total\_revenue:.2f}")

print(f"Revenue from Room Bookings: ${total\_room\_revenue:.2f}")

print(f"Revenue from Services: ${total\_service\_revenue:.2f}")

print("="\*50)

print("Press Enter to go back to the main menu")

input()

cursor.close()

def generate\_bill(connection):

cursor = connection.cursor()

sql\_checked\_out = """

SELECT b.BookingID, u.Username, r.RoomID, b.CheckInDate, b.CheckOutDate, b.TotalCost

FROM Bookings b

JOIN Users u ON b.UserID = u.UserID

JOIN Rooms r ON b.RoomID = r.RoomID

WHERE b.BookingStatus = 'Checked Out'

"""

cursor.execute(sql\_checked\_out)

bookings = cursor.fetchall()

if not bookings:

print("No checked-out bookings found.")

cursor.close()

return

print("\n--- Checked-Out Bookings ---")

print(f"{'Booking ID':<12} | {'Username':<15} | {'Room ID':<10} | {'Check-In Date':<15} | {'Check-Out Date':<15} | {'Total Cost':<10}")

print("-" \* 90)

for booking in bookings:

booking\_id, username, room\_id, check\_in\_date, check\_out\_date, total\_cost = booking

check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'

check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'

total\_cost = f"${total\_cost:.2f}" if total\_cost is not None else 'N/A'

print(f"{booking\_id:<12} | {username:<15} | {room\_id:<10} | {check\_in\_date:<15} | {check\_out\_date:<15} | {total\_cost:<10}")

print("-" \* 90)

booking\_id = input("Enter Booking ID to generate bill: ")

sql\_validate\_booking = """

SELECT b.BookingID, u.Username, r.RoomID, b.CheckInDate, b.CheckOutDate, b.TotalCost

FROM Bookings b

JOIN Users u ON b.UserID = u.UserID

JOIN Rooms r ON b.RoomID = r.RoomID

WHERE b.BookingID = %s AND b.BookingStatus = 'Checked Out'

"""

cursor.execute(sql\_validate\_booking, (booking\_id,))

booking = cursor.fetchone()

if not booking:

print("Invalid Booking ID or Booking is not checked out.")

cursor.close()

return

booking\_id, username, room\_id, check\_in\_date, check\_out\_date, room\_total\_cost = booking

num\_days = (check\_out\_date - check\_in\_date).days

num\_nights = num\_days

sql\_services = """

SELECT rs.ServiceID, s.ServiceType, s.Price

FROM RoomServices rs

JOIN Services s ON rs.ServiceID = s.ServiceID

WHERE rs.BookingID = %s AND rs.Status = 'Completed'

"""

cursor.execute(sql\_services, (booking\_id,))

services = cursor.fetchall()

total\_service\_cost = sum(service[2] for service in services)

total\_cost = room\_total\_cost + total\_service\_cost

pst\_rate = Decimal('0.08')

pst\_amount = total\_cost \* pst\_rate

final\_total\_cost = total\_cost + pst\_amount

print("\n" + "="\*50)

print(" Starlight Hotel")

print(" Where stars meet comfort")

print("="\*50)

print(f"Booking ID: {booking\_id}")

print(f"Customer: {username}")

print(f"Room ID: {room\_id}")

print(f"Check-In Date: {check\_in\_date.strftime('%Y-%m-%d')}")

print(f"Check-Out Date: {check\_out\_date.strftime('%Y-%m-%d')}")

print(f"Duration of Stay: {num\_days} Days and {num\_nights} Nights")

print("="\*50)

print("Services:")

print(f"{'Service ID':<12} | {'Service Type':<20} | {'Price':<10}")

print("-" \* 45)

for service in services:

service\_id, service\_type, price = service

print(f"{service\_id:<12} | {service\_type:<20} | ${price:<10.2f}")

print("-" \* 45)

print(f"Room Cost: ${room\_total\_cost:.2f}")

print(f"Total Service Cost: ${total\_service\_cost:.2f}")

print(f"PST (8%): ${pst\_amount:.2f}")

print(f"Total Cost: ${final\_total\_cost:.2f}")

print("="\*50)

print("Thank you for choosing Starlight Hotel!")

print("="\*50)

print("Press Enter to go back to the main menu")

input()

cursor.close()

def handle\_booking\_approval\_rejection(connection):

if not view\_pending\_bookings(connection):

return

while True:

booking\_id = input("Enter Booking ID to approve or reject (or press 0 to go back to the main menu): ")

if booking\_id == '0':

return

cursor = connection.cursor()

sql\_validate\_booking = "SELECT BookingID FROM Bookings WHERE BookingID = %s AND BookingStatus = 'Pending'"

cursor.execute(sql\_validate\_booking, (booking\_id,))

valid\_booking = cursor.fetchone()

cursor.close()

if not valid\_booking:

print("Invalid Booking ID. Please enter a valid pending Booking ID.")

continue

while True:

print("\n1. Approve Booking")

print("2. Reject Booking")

choice = input("Enter your choice: ")

if choice == '1':

approve\_booking(connection, booking\_id)

break

elif choice == '2':

reject\_booking(connection, booking\_id)

break

else:

print("Invalid choice. Please try again.")

break

def manager\_menu(connection):

while True:

print("\n--- Manager Menu ---")

print("1. Bookings")

print("2. View Room Availability")

print("3. Generate Daily Activity Reports")

print("4. Generate Bill for Customer")

print("5. Sign Out / Switch User")

print("6. Exit")

choice = input("Enter your choice: ")

if choice == '1':

handle\_booking\_approval\_rejection(connection)

elif choice == '2':

view\_room\_availability(connection)

elif choice == '3':

generate\_daily\_activity\_reports(connection)

elif choice == '4':

generate\_bill(connection)

elif choice == '5':

print("Signing out...")

return 'signout'

elif choice == '6':

print("Exiting...")

return 'exit'

else:

print("Invalid choice. Please try again.")

# Staff

## Create

* **provide\_room\_service(connection, staff\_id)**
  + **Purpose**: Marks a room service request as completed.
  + **Procedure**: Staff selects a request, and the function updates its status to 'Completed'.

A screenshot of a computer

Description automatically generated

## Read

* **view\_approved\_bookings(connection)**
  + **Purpose**: Fetches and displays approved bookings.
  + **Procedure**: Executes a SELECT statement to retrieve and display approved bookings.
* **view\_checked\_in\_bookings(connection)**
  + **Purpose**: Fetches and displays checked-in bookings.
  + **Procedure**: Executes a SELECT statement to retrieve and display checked-in bookings.
* **view\_room\_service\_requests(connection)**
  + **Purpose**: Fetches and displays room service requests.
  + **Procedure**: Executes a SELECT statement to retrieve and display room service requests.

A screenshot of a computer

Description automatically generated

* **view\_serviceable\_rooms(connection)**
  + **Purpose**: Fetches and displays rooms in maintenance status.
  + **Procedure**: Executes a SELECT statement to retrieve and display rooms under maintenance.
* **view\_completed\_services(connection)**
  + **Purpose**: Fetches and displays completed room services.
  + **Procedure**: Executes a SELECT statement to retrieve and display completed room services.

## Update

* **handle\_checkin(connection)**
  + **Purpose**: Marks an approved booking as checked in.
  + **Procedure**: Staff selects a booking, and the function updates its status to 'Checked In'.
* **handle\_checkout(connection)**
  + **Purpose**: Marks a checked-in booking as checked out.
  + **Procedure**: Staff selects a booking, and the function updates its status to 'Checked Out'.

A screenshot of a computer

Description automatically generated

* **update\_room\_status(connection)**
  + **Purpose**: Updates the status of a room from maintenance to available.
  + **Procedure**: Staff selects a room, and the function updates its status in the database.

## Delete

* Not applicable for staff.

## Staff Methods (script)

from db\_connection import create\_connection

import mysql.connector

def view\_approved\_bookings(connection):

cursor = connection.cursor()

sql = """

SELECT b.BookingID, u.Username, r.RoomID, b.CheckInDate, b.CheckOutDate

FROM Bookings b

JOIN Users u ON b.UserID = u.UserID

JOIN Rooms r ON b.RoomID = r.RoomID

WHERE b.BookingStatus = 'Approved'

"""

cursor.execute(sql)

bookings = cursor.fetchall()

if bookings:

print("\n--- Approved Bookings ---")

print(f"{'Booking ID':<12} | {'Username':<15} | {'Room ID':<10} | {'Check-In Date':<15} | {'Check-Out Date':<15}")

print("-" \* 80)

for booking in bookings:

booking\_id, username, room\_id, check\_in\_date, check\_out\_date = booking

check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'

check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'

print(f"{booking\_id:<12} | {username:<15} | {room\_id:<10} | {check\_in\_date:<15} | {check\_out\_date:<15}")

print("-" \* 80)

cursor.close()

return True

else:

print("No approved bookings.")

cursor.close()

return False

def view\_checked\_in\_bookings(connection):

cursor = connection.cursor()

sql = """

SELECT b.BookingID, u.Username, r.RoomID, b.CheckInDate, b.CheckOutDate

FROM Bookings b

JOIN Users u ON b.UserID = u.UserID

JOIN Rooms r ON b.RoomID = r.RoomID

WHERE b.BookingStatus = 'Checked In'

"""

cursor.execute(sql)

bookings = cursor.fetchall()

if bookings:

print("\n--- Checked-In Bookings ---")

print(f"{'Booking ID':<12} | {'Username':<15} | {'Room ID':<10} | {'Check-In Date':<15} | {'Check-Out Date':<15}")

print("-" \* 80)

for booking in bookings:

booking\_id, username, room\_id, check\_in\_date, check\_out\_date = booking

check\_in\_date = check\_in\_date.strftime('%Y-%m-%d') if check\_in\_date else 'N/A'

check\_out\_date = check\_out\_date.strftime('%Y-%m-%d') if check\_out\_date else 'N/A'

print(f"{booking\_id:<12} | {username:<15} | {room\_id:<10} | {check\_in\_date:<15} | {check\_out\_date:<15}")

print("-" \* 80)

cursor.close()

return True

else:

print("No checked-in bookings.")

cursor.close()

return False

def handle\_checkin(connection):

cursor = connection.cursor()

if view\_approved\_bookings(connection):

booking\_id = input("Enter Booking ID for check-in: ")

sql = """

UPDATE Bookings

SET BookingStatus = 'Checked In', CheckedInTime = NOW()

WHERE BookingID = %s

"""

values = (booking\_id,)

try:

cursor.execute(sql, values)

connection.commit()

print("Check-in successful.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

else:

cursor.close()

def handle\_checkout(connection):

cursor = connection.cursor()

if view\_checked\_in\_bookings(connection):

booking\_id = input("Enter Booking ID for check-out: ")

sql\_update\_booking = """

UPDATE Bookings

SET BookingStatus = 'Checked Out', CheckedOutTime = NOW()

WHERE BookingID = %s

"""

sql\_get\_room\_id = "SELECT RoomID FROM Bookings WHERE BookingID = %s"

cursor.execute(sql\_get\_room\_id, (booking\_id,))

room\_id = cursor.fetchone()

if room\_id:

room\_id = room\_id[0]

sql\_update\_room = "UPDATE Rooms SET Availability = 'Maintenance' WHERE RoomID = %s"

try:

cursor.execute(sql\_update\_booking, (booking\_id,))

cursor.execute(sql\_update\_room, (room\_id,))

connection.commit()

print("Check-out successful and room status set to 'Maintenance'.")

except mysql.connector.Error as err:

print(f"Error: {err}")

else:

print("Invalid Booking ID.")

cursor.close()

else:

cursor.close()

def view\_room\_service\_requests(connection):

cursor = connection.cursor()

sql = """

SELECT rs.RoomServiceID, rs.BookingID, rs.ServiceID, rs.Status, rs.RequestTime, s.ServiceType

FROM RoomServices rs

JOIN Services s ON rs.ServiceID = s.ServiceID

WHERE rs.Status = 'Requested'

"""

cursor.execute(sql)

service\_requests = cursor.fetchall()

if service\_requests:

print("\n--- Room Service Requests ---")

print(f"{'Room Service ID':<15} | {'Booking ID':<12} | {'Service ID':<10} | {'Service Type':<15} | {'Status':<10} | {'Request Time':<20}")

print("-" \* 100)

for request in service\_requests:

room\_service\_id, booking\_id, service\_id, status, request\_time, service\_type = request

request\_time = request\_time.strftime('%Y-%m-%d %H:%M:%S') if request\_time else 'N/A'

print(f"{room\_service\_id:<15} | {booking\_id:<12} | {service\_id:<10} | {service\_type:<15} | {status:<10} | {request\_time:<20}")

print("-" \* 100)

else:

print("No room service requests.")

cursor.close()

def view\_serviceable\_rooms(connection):

cursor = connection.cursor()

sql = """

SELECT RoomID, RoomType

FROM Rooms

WHERE Availability = 'Maintenance'

"""

cursor.execute(sql)

rooms = cursor.fetchall()

if rooms:

print("\n--- Serviceable Rooms (Maintenance) ---")

print(f"{'Room ID':<10} | {'Room Type':<15}")

print("-" \* 30)

for room in rooms:

room\_id, room\_type= room

print(f"{room\_id:<10} | {room\_type:<15}")

print("-" \* 30)

input("Press Enter to return")

else:

print("No rooms currently in maintenance status.")

cursor.close()

def provide\_room\_service(connection, staff\_id):

cursor = connection.cursor()

sql\_check\_requested = """

SELECT COUNT(\*)

FROM RoomServices

WHERE Status = 'Requested'

"""

cursor.execute(sql\_check\_requested)

requested\_count = cursor.fetchone()[0]

if requested\_count == 0:

print("No room service requests.")

cursor.close()

return

view\_room\_service\_requests(connection)

room\_service\_id = input("Enter Room Service ID to complete the service: ")

sql = """

UPDATE RoomServices

SET Status = 'Completed', CompletionTime = CURRENT\_TIMESTAMP, CompletedByStaffID = %s

WHERE RoomServiceID = %s

"""

values = (staff\_id, room\_service\_id)

try:

cursor.execute(sql, values)

connection.commit()

print("Room service completed successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

def update\_room\_status(connection):

cursor = connection.cursor()

sql\_check\_maintenance = """

SELECT RoomID, RoomType

FROM Rooms

WHERE Availability = 'Maintenance'

"""

cursor.execute(sql\_check\_maintenance)

maintenance\_rooms = cursor.fetchall()

if not maintenance\_rooms:

print("There are no rooms currently in maintenance status.")

cursor.close()

return

view\_serviceable\_rooms(connection)

room\_id = input("Enter Room ID to update status: ")

sql\_validate = "SELECT RoomID FROM Rooms WHERE RoomID = %s AND Availability = 'Maintenance'"

cursor.execute(sql\_validate, (room\_id,))

room = cursor.fetchone()

if room:

sql\_update\_room = "UPDATE Rooms SET Availability = 'Available' WHERE RoomID = %s"

values = (room\_id,)

try:

cursor.execute(sql\_update\_room, values)

connection.commit()

print("Room status updated successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

else:

print("Invalid Room ID or Room is not in maintenance.")

cursor.close()

def view\_completed\_services(connection):

cursor = connection.cursor()

sql = """

SELECT rs.RoomServiceID, rs.BookingID, rs.ServiceID, rs.Status, rs.RequestTime, rs.CompletionTime, s.ServiceType, u.Username AS CompletedBy

FROM RoomServices rs

JOIN Services s ON rs.ServiceID = s.ServiceID

LEFT JOIN Users u ON rs.CompletedByStaffID = u.UserID

WHERE rs.Status = 'Completed'

"""

cursor.execute(sql)

completed\_services = cursor.fetchall()

if completed\_services:

print("\n--- Completed Room Services ---")

print(f"{'Room Service ID':<15} | {'Booking ID':<12} | {'Service ID':<10} | {'Service Type':<15} | {'Status':<10} | {'Request Time':<20} | {'Completion Time':<20} | {'Completed By':<15}")

print("-" \* 150)

for service in completed\_services:

room\_service\_id, booking\_id, service\_id, status, request\_time, completion\_time, service\_type, completed\_by = service

request\_time = request\_time.strftime('%Y-%m-%d %H:%M:%S') if request\_time else 'N/A'

completion\_time = completion\_time.strftime('%Y-%m-%d %H:%M:%S') if completion\_time else 'N/A'

completed\_by = completed\_by if completed\_by else 'N/A'

print(f"{room\_service\_id:<15} | {booking\_id:<12} | {service\_id:<10} | {service\_type:<15} | {status:<10} | {request\_time:<20} | {completion\_time:<20} | {completed\_by:<15}")

print("-" \* 150)

else:

print("No completed room services.")

cursor.close()

def staff\_menu(connection, staff\_id):

while True:

print("\n--- Staff Menu ---")

print("1. Handle Check-In")

print("2. Handle Check-Out")

print("3. Provide Room Service")

print("4. Update Room Status")

print("5. View Completed Services")

print("6. Rooms Which need Maintenance ")

print("7. Sign Out / Switch User")

print("8. Exit")

choice = input("Enter your choice: ")

if choice == '1':

handle\_checkin(connection)

elif choice == '2':

handle\_checkout(connection)

elif choice == '3':

provide\_room\_service(connection, staff\_id)

elif choice == '4':

update\_room\_status(connection)

elif choice == '5':

view\_completed\_services(connection)

elif choice == '6':

view\_serviceable\_rooms(connection)

elif choice == '7':

print("Signing out...")

return 'signout'

elif choice == '8':

print("Exiting...")

return 'exit'

else:

print("Invalid choice. Please try again.")

# Menu System

## User Registration and Login

* **register\_user(connection)**
  + **Purpose**: Registers a new user.
  + **Procedure**: Admin enters user details, and the function inserts the new user into the database.
* **login\_user(connection)**
  + **Purpose**: Authenticates a user.
  + **Procedure**: User enters credentials, and the function validates them against the database.

A screenshot of a computer

Description automatically generated

## Role-Based Menus

* **admin\_menu(connection)**
  + **Purpose**: Displays the admin menu.
  + **Procedure**: Admin selects an option, and the corresponding function is executed.

**A screenshot of a computer

Description automatically generated**

* **manager\_menu(connection)**
  + **Purpose**: Displays the manager menu.
  + **Procedure**: Manager selects an option, and the corresponding function is executed.

A screenshot of a computer

Description automatically generated

* **staff\_menu(connection, user\_id)**
  + **Purpose**: Displays the staff menu.
  + **Procedure**: Staff selects an option, and the corresponding function is executed.

A screenshot of a computer program

Description automatically generated

* **customer\_menu(connection, user\_id)**
  + **Purpose**: Displays the customer menu.
  + **Procedure**: Customer selects an option, and the corresponding function is executed.

**A screenshot of a computer

Description automatically generated**

## Menu System(Script)

from db\_connection import create\_connection

from admin\_methods import admin\_menu

from manager\_methods import manager\_menu

from staff\_methods import staff\_menu

from customer\_methods import customer\_menu

import getpass

def register\_user(connection):

cursor = connection.cursor()

username = input("Enter Username: ")

password = getpass.getpass("Enter Password: ")

role = input("Enter Role (Admin, Manager, Staff, Customer): ")

first\_name = input("Enter First Name: ")

last\_name = input("Enter Last Name: ")

email = input("Enter Email: ")

phone = input("Enter Phone: ")

sql = """

INSERT INTO Users (Username, Password, Role, FirstName, LastName, Email, Phone)

VALUES (%s, %s, %s, %s, %s, %s, %s)

"""

values = (username, password, role, first\_name, last\_name, email, phone)

try:

cursor.execute(sql, values)

connection.commit()

print("User registered successfully.")

except mysql.connector.Error as err:

print(f"Error: {err}")

finally:

cursor.close()

def login\_user(connection):

cursor = connection.cursor()

username = input("Enter Username: ")

password = getpass.getpass("Enter Password: ")

sql = "SELECT \* FROM Users WHERE Username = %s AND Password = %s"

values = (username, password)

cursor.execute(sql, values)

user = cursor.fetchone()

cursor.close()

if user:

print(f"Login successful. Welcome, {user[4]} {user[5]} ({user[3]}).")

return user

else:

print("Invalid username or password.")

return None

def main():

connection = create\_connection()

if connection and connection.is\_connected():

while True:

user = None

while not user:

user = login\_user(connection)

if not user:

print("Login failed. Please try again.")

user\_id = user[0]

role = user[3]

action = None

if role == 'Admin':

action = admin\_menu(connection)

elif role == 'Manager':

action = manager\_menu(connection)

elif role == 'Staff':

action = staff\_menu(connection, user\_id)

elif role == 'Customer':

action = customer\_menu(connection, user\_id)

if action == 'exit':

break

elif action == 'signout':

continue

connection.close()

else:

print("Failed to connect to the database.")

if \_\_name\_\_ == '\_\_main\_\_':

main()

# Database Connection (script)

import mysql.connector

def create\_connection():

connection = mysql.connector.connect(

host='127.0.0.1',

user='Saksham',

password='Saksham',

database='StarlightHotel'

)

return connection

# MongoDB Table Structure/Document Structure

## Room Booking Status Cache

{

"\_id": "<booking\_id>",

"user\_id": "<user\_id>",

"room\_id": "<room\_id>",

"status": "Approved", // or other statuses like Pending, Rejected, Checked In, Checked Out

"check\_in\_date": "<YYYY-MM-DD>",

"check\_out\_date": "<YYYY-MM-DD>",

"last\_updated": "<timestamp>"

}

Check In status  
  
{

"\_id": "<booking\_id>",

"user\_id": "<user\_id>",

"room\_id": "<room\_id>",

"checked\_in\_time": "<timestamp>",

"checked\_out\_time": "<timestamp>",

"is\_checked\_in": true, // Boolean flag for quick check-in status

"is\_checked\_out": false,

"last\_updated": "<timestamp>"

}

## Room Service Request Cache

{

"\_id": "<service\_request\_id>",

"booking\_id": "<booking\_id>",

"service\_id": "<service\_id>",

"status": "Requested", // or Completed

"request\_time": "<timestamp>",

"completion\_time": "<timestamp>",

"completed\_by\_staff\_id": "<staff\_id>",

"service\_price": "<decimal>",

"last\_updated": "<timestamp>"

}

## Services Cache

{

"\_id": "<service\_id>",

"service\_type": "Room", // or Service

"description": "Standard room fee",

"price": "<decimal>",

"applicable\_to": ["Single", "Double", "Suite"], // Array of room types this service applies to

"last\_updated": "<timestamp>"

}

# ER Diagram

A diagram of a hotel service

Description automatically generated

Video Link

[20240803\_163211363\_iOS.mp4](https://collegedouglas-my.sharepoint.com/:v:/g/personal/vasudevs_student_douglascollege_ca/EcKU5AmL54NGiAh9TAYSwggBYndMLgAlsGmCh-7dlbtjgw?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=i8Rgua)

Conclusion

This report outlines the CRUD functionalities for different roles within the Starline hotel management system, detailing the relevant methods and database interactions for each. The system effectively manages user registration, room bookings, booking approval and rejection, check-in and check-out processes, room service requests, and administrative tasks, ensuring a comprehensive and efficient management solution.