****

**Minor Project Report**

**on**

**Project Title: - Online Hotel Booking Management System**

**Course: Advanced Database Techniques**

**Course Code: CAP570**

| **Submitted by**  **1. Ritesh Kumar (12405323), Group: - 1**  **2. Akash Arya (12403255), Group: - 1**  **3. Shubham (), Group: - 1** | **Submitted to**  Dr. Heena Khanna  UID: 31463  Assistant Professor, SCA, LPU |
| --- | --- |

**Lovely Faculty of Technology & Sciences**

**School of Computer Applications**

**Lovely Professional University**

**Punjab**

**Acknowledgment**

We would like to express our sincere gratitude to **Dr. Heena Khanna (31463), Assistant Professor, School of Computer Applications, Lovely Professional University**, for her guidance and continuous supervision, as well as for providing essential information and unwavering support throughout the project. Her consistent direction and willingness to share her extensive knowledge enabled us to gain a deep understanding of the project, which greatly assisted us in completing our tasks on time.

We would also like to extend our heartfelt thanks to all the individuals who graciously participated in the interviews and surveys conducted during the project. Their experiences and insights provided a practical understanding of how database systems are used in real-world applications, which greatly enriched the quality of this project.

Finally, we would like to thank our families and friends for their constant encouragement and support during this journey. This project would not have been possible without the contribution of everyone involved.

Thank you all.

## 

## **Table of Contents**

| **Sr. No.** | **Content** | **Page No.** |
| --- | --- | --- |
| 1. | Introduction | 3 |
| 2. | Project Overview | 4 - 9 |
| 3. | Project Objectives | 10 |
| 4. | Problem Statement | 11 |
| 5. | Requirement Gathering· Interviews and Surveys· Website Exploration | 12 - 15 |
| 6. | ER Diagram and Relational Schema | 16 - 17 |
| 7. | Database Design and Normalisation· Tables Description· Normalisation Process | 18 - 25 |
| 8. | SQL Queries and Database Operations· Create Tables· Insert Data· Queries for Data Retrieval | 26 - 55 |
| 9. | Conclusion | 56 - 57 |

### **1. Introduction**

In the modern era, effective database management systems have become essential for managing large volumes of data efficiently and accurately. This Hotel Management System project is designed to provide a comprehensive and scalable database solution to handle the operations and data requirements of a hotel, from booking management and customer data handling to room assignments and billing. Similar to the way large-scale applications like Netflix manage vast libraries of media and customer interactions through robust databases, this system allows for the efficient organisation, retrieval, and manipulation of information critical to hotel operations. This introduction gives a clear purpose and contextual importance to the Hotel Management System database while connecting it to the broader relevance of database systems in contemporary applications. Let me know if you'd like to include additional technical details or any other information.

**Project Overview**

This Hotel Management System database is structured to support core functionalities such as managing room availability, tracking customer reservations, processing payments, and handling staff information. By organising these components in a structured database, the system aims to streamline operational workflows, reduce manual errors, and enhance data accuracy, which ultimately contributes to a better experience for both customers and hotel staff.

**Importance of Database Systems in Modern Applications**

The importance of such systems is evident across numerous industries where massive data processing and real-time interaction are required. Just as Netflix and other streaming platforms rely on complex databases to deliver relevant content to millions of users, a hotel system relies on its database to maintain room availability, track customer preferences, and generate invoices promptly. This project emphasises the importance of a well-architected database in achieving efficiency, security, and data integrity within the hospitality industry.

**2. Project Overview**

The Hotel Management System is a database solution designed to efficiently manage hotel operations by organising, tracking, and processing the essential data related to guests, room availability, staff, bookings, and payments. The primary goal of this database is to facilitate the smooth operation of a hotel by automating data handling tasks, reducing manual workloads, and ensuring accurate record-keeping. This system is crafted to support various operational needs, from reservations and room allocation to billing and inventory management, contributing to a seamless experience for both guests and hotel staff.

**Project Concept and Scope**

This database aims to handle critical areas of hotel management by providing structured data storage and accessibility. It will support functions such as:

* **Managing reservations:** Tracking room bookings, guest check-ins, and check-outs.
* **Customer data management**: Storing guest information, preferences, and stay history.
* **Room inventory and allocation:** Keeping records of room details, availability status, and categories.
* **Billing and payment processing:** Generating and managing bills, handling payments, and maintaining transaction records.
* **Staff management:** Storing staff information, roles, and schedules.
* **Facility Management**: Recording and managing hotel amenities and services offered, ensuring guests have access to accurate information about available facilities.

The database is built to handle multi-user access and large amounts of data, ensuring that all information is accurate, accessible, and secure.

**Core Entities and Their Roles**

**Hotel :** The Hotel entity contains essential information about each hotel, such as its unique ID, name, address, contact details, and the number of rooms. This entity supports efficient hotel management by organising property-specific details, enabling streamlined operations and room allocation. It also plays a key role in associating hotels with their respective facilities, services, and staff, ensuring comprehensive property representation in the system.

**Customer** (Guests): This entity captures information about each guest, including contact details, ID information, and preferences. It enables the system to track guest history, preferences, and other details to enhance personalised service.

**Room**: The Rooms entity stores information on each room, including room numbers, types, rates, and status (available, occupied, or under maintenance). This entity helps in the organisation of room inventory, making it easier to allocate rooms based on guest preferences and availability.

**Reservations**: The Reservations entity links guests with specific rooms for a defined period, including details like check-in and check-out dates, room type, and reservation status. This enables efficient tracking of bookings and room assignments.

**Staff**: This entity maintains information about hotel employees, including their roles, schedules, and contact details. Having a record of the staff supports hotel management in assigning tasks, managing schedules, and ensuring optimal staff utilisation.

**Payment**: The Billing or Payments entity captures billing information, such as room charges, additional services, and total payments. It is crucial for generating invoices and tracking payments made by guests, ensuring smooth billing operations.

**Services**: This entity captures additional services like laundry, dining, and spa services offered by the hotel. Guests can request these services, and they are added to their final bill, ensuring a complete record of all charges during their stay

**Attributes and Relationships**

Each entity in the database includes key attributes that define the necessary details for hotel management operations. For example:

**Hotel Attributes**:

HotelID, Name, Address, City, State, Country, PostalCode, PhoneNo, Email, NoOfRooms

**Customer Attributes**:

CustomerID, FirstName, LastName, Date of Birth, PhoneNo, Email, Address. City, State, Country, PostalCode

**Room Attributes**:

RoomID, HotelID, RoomNo, RoomType, BedType, PricePerNight, AvailabilityStatus

**Reservation Attributes**:

ReservationID, CustomerID, RoomID, HotelID, CheckInDate, CheckOutDate, NoOfPeople, TotalPrice, BookingDate, ReservationStatus.

**Payment Attributes**:

PaymentID, PaymentDate, ReservationID, Amount, PaymentStatus, PaymentMethod

**Facility Attributes**:

FacilityID, HotelID, FacilityName, Description

**Service Attributes**:

ServiceID, ServiceName, ServiceCost, HotelID

**Staff Attributes**:

StaffID, HotelID, FirstName, LastName, JobPosition, PhoneNo, Email, HireDate, Salary

**Review Attributes**:

ReviewID, CustomerID, HotelID, Rating, Remarks, ReviewDate

**Relationships in this system establish how these entities interact with each other:**

**Hotels and Rooms:**

* + **Relationship Type**: One-to-Many
  + **Details**: Each hotel can have multiple rooms, but each room is part of only one hotel.
  + **Foreign Key**: The HotelID in the Room table references the HotelID in the Hotel table.

**Hotels and Facilities:**

* **Relationship Type**: One-to-Many
* **Details**: Each hotel can have multiple facilities, but each facility is associated with only one hotel.
* **Foreign Key**: The HotelID in the Facility table references the HotelID in the Hotel table.

### **Hotels and Services:**

* **Relationship Type:** One-to-Many
* **Details:** Each hotel offers multiple services, but each service is specific to a hotel.
* **Foreign Key:** The HotelID in the Service table references the HotelID in the Hotel table.

**Hotels and Staff:**

* **Relationship Type**: One-to-Many
* **Details**: Each hotel employs multiple staff members, but each staff member belongs to one hotel.
* **Foreign Key**: The HotelID in the Staff table references the HotelID in the Hotel table.

**Hotels and Reviews:**

* **Relationship Type**: One-to-Many
* **Details**: Each hotel can receive multiple reviews, but each review pertains to one hotel.
* **Foreign Key**: The HotelID in the Review table references the HotelID in the Hotel table.

**Customers and Reservations:**

* **Relationship Type**: One-to-Many
* **Details**: Each customer can make multiple reservations, but each reservation is linked to only one customer.
* **Foreign Key**: The CustomerID in the Reservation table references the CustomerID in the Customer table.

**Reservations and Rooms:**

* **Relationship Type**: One-to-Many
* **Details**: Each reservation is associated with one specific room. However, the same room can be reserved multiple times across different periods.
* **Foreign Key**: The RoomID in the Reservation table references the RoomID in the Room table.

**Customers and Reviews:**

* **Relationship Type**: One-to-Many
* **Details**: Each customer can submit multiple reviews, but each review is linked to only one customer.
* **Foreign Key**: The CustomerID in the Review table references the CustomerID in the Customer table.

### **Reservations and Payments:**

* **Relationship Type:** One-to-One
* **Details:** Each reservation has a unique bill associated with it, which includes room charges and any additional services availed.
* **Foreign Key:** The ReservationID in the Payment table references the ReservationID in the Reservation table.

### **Summary of Relationships:**

**One-to-Many**:

* + Customer to Reservation
  + Hotel to Room
  + Hotel to Facility
  + Hotel to Service
  + Hotel to Staff
  + Customer to Review
  + Hotel to Review

**One-to-One**:

* + Reservation to Payment

**3. Project Objectives**

**Objectives of the Hotel Management System Database Project**

Build a structured database to manage hotel operations: Design a comprehensive, relational database to efficiently manage guest information, room inventory, reservations, billing, and staff records.

1. **Automate booking and reservation management**: Implement an automated system that handles room reservations, tracks availability, and updates check-in and check-out statuses, reducing manual booking errors.
2. **Enhance guest experience through personalised service**: Store and manage guest information, including preferences and stay history, enabling the hotel to offer personalised services and improve guest satisfaction.
3. **Maintain accurate room inventory and status**: Track each room’s details, including type, rate, and availability, to simplify room allocation and provide real-time status updates for front desk operations.
4. **Streamline billing and payment processing**: Facilitate efficient billing by generating invoices, managing payment records, and consolidating charges for rooms and additional services availed during the stay.
5. **Support staff management and scheduling**: Record essential staff information, roles, and schedules, helping hotel management allocate tasks and maintain optimal staffing levels.
6. **Provide a secure and scalable data structure**: Develop a secure, multi-user access system that safeguards guest and operational data while supporting the scalability needed for future hotel expansion.
7. **Generate insightful operational reports**: Enable the creation of reports on occupancy rates, revenue, and guest demographics to assist management in decision-making and improve operational efficiency.
8. **Facilitate efficient customer service**: Enable quick access to guest and reservation data, allowing hotel staff to efficiently manage customer queries, check-ins, and check-outs.
9. **Ensure data consistency and integrity**: Implement relational integrity constraints to maintain accurate and consistent data across all related entities, such as reservations, guests, rooms, and billing.

**4. Problem Statement**

Hotels manage a wide range of data and tasks, from guest reservations and room availability to billing and staff coordination. Traditionally, many of these processes are either paper-based or managed using disjointed systems, leading to inefficiencies, data inconsistencies, and delays in accessing critical information. This disorganisation can result in errors in booking, customer dissatisfaction, and challenges in managing hotel operations effectively.

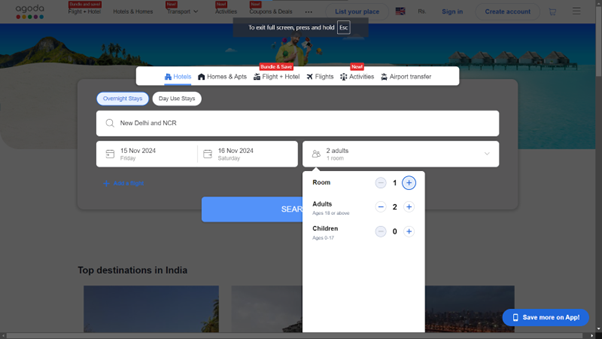
The primary problem addressed by this Hotel Management System database is the need for a centralised, efficient, and reliable system to organise and manage all essential hotel operations. Without such a system, hotels struggle with:

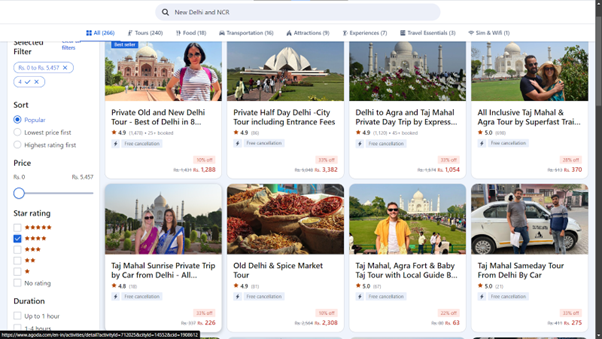
* **Room allocation and availability tracking**: Manually tracking room status can lead to overbooking or inefficient room utilisation.
* **Guest data management**: Disorganised guest information hinders personalised service and effective handling of guest preferences and history.
* **Billing and payment issues**: Managing billing for rooms and additional services separately can lead to billing inaccuracies, delayed check-outs, and frustrated guests.
* **Staff coordination and scheduling**: Inefficient tracking of staff roles and schedules makes it challenging to assign tasks and maintain optimal staffing levels.

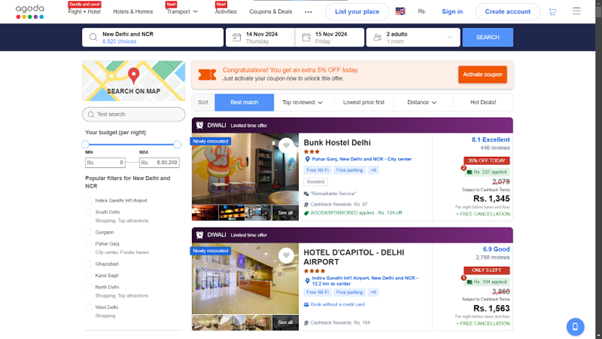
This project aims to create a robust, relational database that streamlines hotel operations, reduces manual errors, and improves overall efficiency by centralising data management. The solution will enable the hotel to enhance guest experiences, support effective resource utilisation, and make informed operational decisions.

**5. Requirement Gathering**

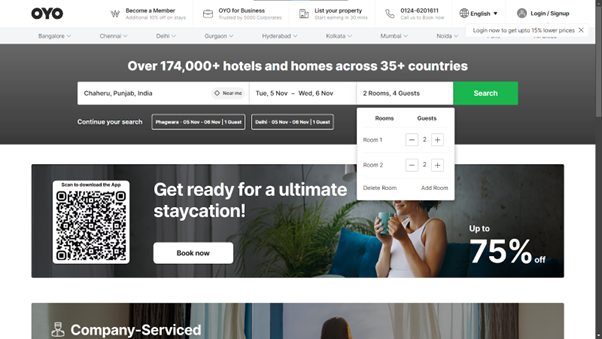
#### **5.1 Website Exploration**

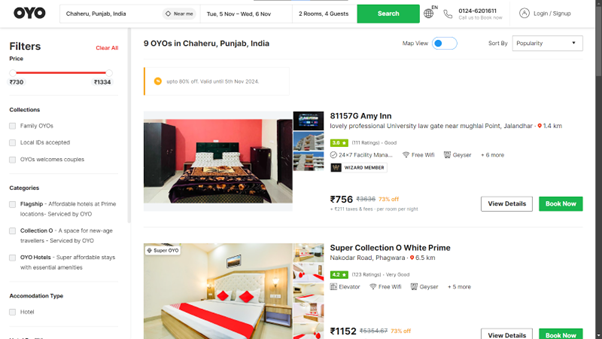
* **Objective:** To explore existing websites or platforms that use similar databases.
* **Websites Explored:**
  + [**Agoda**](https://www.agoda.com/en-in/) : Agoda, a digital travel platform, helps anyone see the world for less with its great value deals on a global network of 4.5M hotels and holiday properties worldwide, plus flights, activities, and more. It is one of the famous companies in the list of companies using MS SQL Server and operates globally and serves customers in over 200 countries and territories.
  + [**OYO Rooms**](https://www.oyorooms.com/) **:** OYO Rooms also known as OYO Hotels & Homes, is an Indian multinational hospitality chain of leased and franchised hotels, homes, and living spaces. Founded in 2012 by Ritesh Agarwal, OYO initially consisted mainly of budget hotels.
* **Screenshots:** Attach relevant screenshots of the websites that students explored.  
  





**OYO**

****



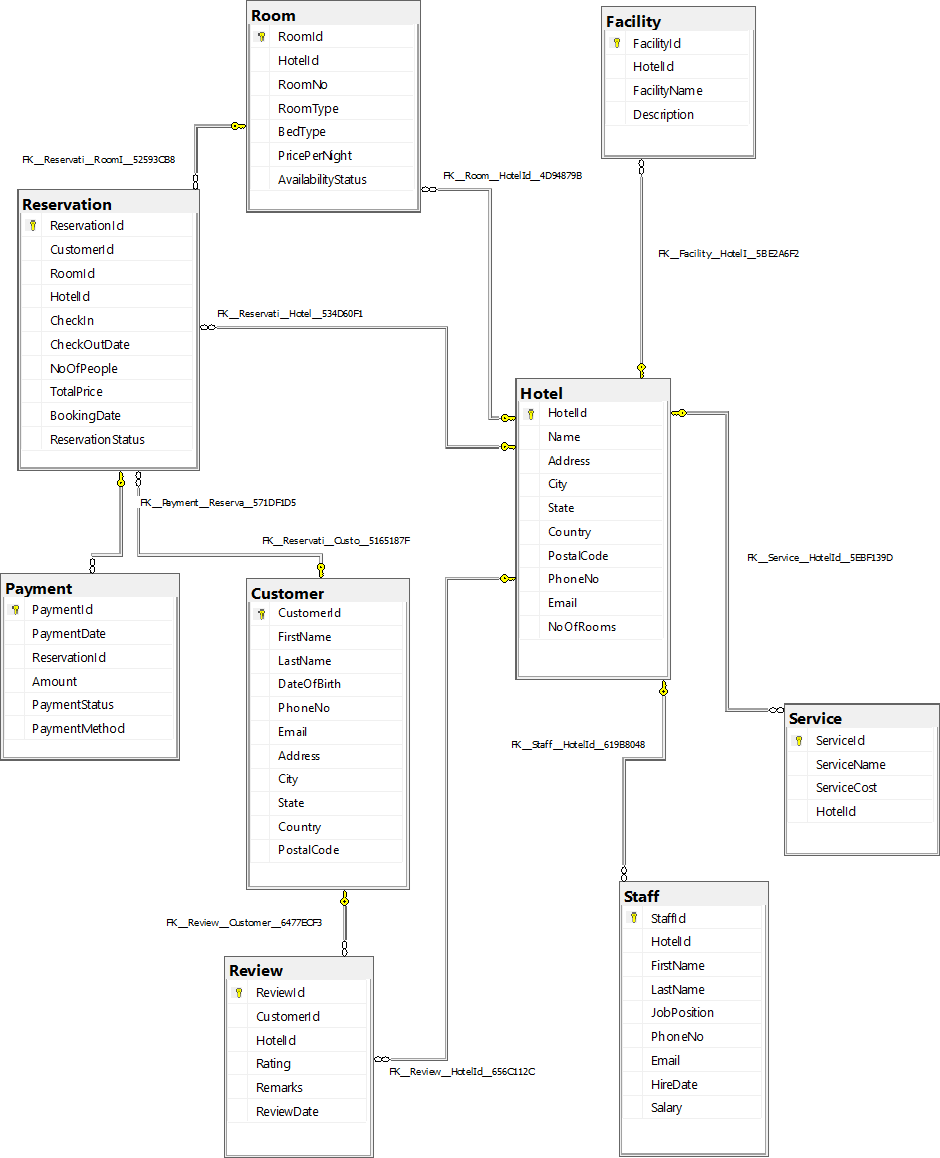
* **Observations:** Summarise the features and structures observed on these platforms.

These platforms leverage Microsoft SQL Server to manage extensive transactional data, supporting real-time availability, customer bookings, and pricing data. It enables high-volume queries, relational data analysis, and insights into booking trends, using stored procedures, indexing & transaction logging for efficient data retrieval and scalability

### **6. ER Diagram and Relational Schema**

The ER Diagram for the Hotel Management System represents key entities like **Customers(Guests),** **Rooms**, **Reservations**, **Payment**, **Staff**, and **Services**, each with their relevant attributes and relationships. Here’s a breakdown of each entity and its connections:

Entity Relationship Diagram

**7. Database Design and Normalisation**

#### **7.1 Table Descriptions**

**Table 1:** **Hotel Table**

* + **Attributes (10):** HotelID, Name, Address, City, State, Country, PostalCode, PhoneNo, Email, NoOfRooms
  + **Primary Key:** HotelID
  + **Description:** Stores information about hotels, including location, contact details, and the number of rooms available.

**Table 2:** **Customer Table**

* + **Attributes (11):** CustomerID, FirstName, LastName, DateOfBirth, PhoneNo, Email, Address, City, State, Country, PostalCode
  + **Primary Key:** CustomerID
  + **Description:** Contains details of customers, including personal and contact information.

**Table 3:** **Room Table**

* + **Attributes (7):** RoomID, HotelID, RoomNo, RoomType, BedType, PricePerNight, AvailabilityStatus
  + **Primary Key:** RoomID
  + **Foreign Key:** HotelID (Foreign Key referencing Hotel(HotelID))
  + **Description:** Represents individual rooms within hotels, detailing their type, pricing, and availability.

**Table 4:** **Reservation Table**

* + **Attributes (9):** ReservationID, CustomerID, RoomID, HotelID, CheckInDate, CheckOutDate, TotalPrice, BookingDate, ReservationStatus
  + **Primary Key:** ReservationID
  + **Foreign Key:**
    - CustomerID (Foreign Key referencing Customer(CustomerID))
    - RoomID (Foreign Key referencing Room(RoomID))
    - HotelID (Foreign Key referencing Hotel(HotelID))
  + **Description:** Holds reservation details, including the check-in and check-out dates, number of people, and status.

**Table 5:** **Payment Table**

* + **Attributes (6):** PaymentID, PaymentDate, ReservationID, Amount, PaymentStatus, PaymentMethod
  + **Primary Key:** PaymentID
  + **Foreign Key:**
    - ReservationID (Foreign Key referencing Reservation (ReservationID))
  + **Description:** Records payments made for reservations, including the method and status of each payment.

**Table 6:** **Facility Table**

* + **Attributes (4):** FacilityID, HotelID, FacilityName, Description
  + **Primary Key:** FacilityID
  + **Foreign Key:** HotelID (Foreign Key referencing Hotel(HotelID))
  + **Description:** Lists facilities available at each hotel, such as swimming pools, gyms, and conference rooms.

**Table 7:** **Service Table**

* + **Attributes (4):** ServiceID, ServiceName, ServiceCost, HotelID
  + **Primary Key:** ServiceID
  + **Foreign Key:** HotelID (Foreign Key referencing Hotel(HotelID))
  + **Description:** Details services provided by hotels, such as room service and spa treatments, along with their costs.

**Table 8:** **Staff Table**

* + **Attributes (8):** StaffID, HotelID, FirstName, LastName, JobPosition, PhoneNo, Email, HireDate, Salary
  + **Primary Key:** StaffID
  + **Foreign Key:** HotelID (Foreign Key referencing Hotel(HotelID))
  + **Description:** Contains information about hotel staff, including job position and salary details.

**Table 9:** **Review Table**

* + **Attributes (6):** ReviewID, CustomerID, HotelID, Rating, Remarks, ReviewDate
  + **Primary Key:** ReviewID
  + **Foreign Key:**
    - CustomerID (Foreign Key referencing Customer(CustomerID))
    - HotelID (Foreign Key referencing Hotel(HotelID))

#### **Description:** Stores customer reviews and ratings for hotels, with optional remarks and automatic timestamp for review date.

#### 

#### **7.2 Normalisation Process**

**Normalisation Process**

The SQL code submitted for the Social Flow social media management database design is in some form of normalisation; however, let us see each table based on principles of normalisation to make sure we have optimised it completely.

**Step 1: First Normal Form (1NF)**

A column in any given table can hold only atomic values, no repeating groups or arrays.

As illustrated by the tables above they are already in 1NF as follows:

At least every column in this table carries one value to each cell.

On each of these tables a minimum one primary key at least will have to feature that would uniquely identify rows.

**Step 2: Second Normal Form (2NF)**

1. It must be in 1NF.

2. It must be so that each and every non-key attributes must be depending completely or entirely on the whole of the primary key.

In the our schema, All of these tables are in 2NF because all of each and every non-key attributes are depending on their corresponding primary key completely.

A foreign key relationship is the kind of relationship between the tables does not cause the partial dependency; hence all attributes must be fully depending on a primary key.

**Step 3: Third Normal Form (3NF)**

Database is in 2NF

There will not exist any transitive dependency. All the non-key attributes must not depend upon certain

columns apart from key.

**Here Schema:**

The schema here is 3NF, as other columns besides some primary column are having dependencies over

themselves, but certainly not over other like these column

**HotelBranchTable**

* **Attributes**: BranchId (PK), HotelName, City, State, Country
* **Analysis**:
  + Non-key attributes (HotelName, City, State, Country) depend directly on BranchId.
  + No additional dependencies among non-key columns.
* **Normalisation Level**: 3NF

**CustomerTable**

* **Attributes**: CustomerId (PK), Name, ContactNumber, Email, Address
* **Analysis**:
  + Each attribute (Name, ContactNumber, Email, Address) depends solely on CustomerId, the primary key.
  + No redundant data or transitive dependencies exist.
* **Normalisation Level**: 3NF

**RoomTable**

* **Attributes**: RoomId (PK), RoomType, PricePerNight, Status
* **Analysis**:
  + Attributes (RoomType, PricePerNight, Status) are directly related to RoomId.
  + No redundant or transitive dependencies exist.
* **Normalisation Level**: 3NF

**ReservationTable**

* **Attributes**: ReservationId (PK), CustomerId (FK referencing CustomerTable), RoomId (FK referencing RoomTable), CheckInDate, CheckOutDate
* **Analysis**:
  + Each attribute (CustomerId, RoomId, CheckInDate, CheckOutDate) is dependent on ReservationId, the primary key.
  + No redundant data or transitive dependencies among non-key columns.
* **Normalisation Level**: 3NF

**PaymentTable**

* **Attributes**: PaymentId (PK), ReservationId (FK referencing ReservationTable), Amount, PaymentDate, PaymentStatus
* **Analysis**:
  + Attributes (Amount, PaymentDate, PaymentStatus) depend solely on PaymentId.
  + No dependencies exist between non-key attributes, meeting the 3NF criteria.
* **Normalisation Level**: 3NF

**StaffTable**

* **Attributes**: StaffId (PK), BranchId (FK referencing HotelBranchTable), Name, Role, ContactNumber
* **Analysis**:
  + Each non-key attribute (Name, Role, ContactNumber) depends solely on the primary key, StaffId.
  + No transitive dependencies among non-key attributes.
* **Normalisation Level**: 3NF

**FacilityTable**

* **Attributes**: FacilityId (PK), BranchId (FK referencing HotelBranchTable), FacilityName, Description
* **Analysis**:
  + Attributes (FacilityName, Description) are dependent only on the primary key, FacilityId.
  + No transitive dependencies or interdependencies among non-key columns.
* **Normalisation Level**: 3NF

**ReviewTable**

* **Attributes**: ReviewId (PK), ReservationId (FK referencing ReservationTable), Rating, Comments
* **Analysis**:
  + Rating and Comments depend solely on ReviewId, the primary key.
  + No redundant data or transitive dependencies between non-key attributes.
* **Normalisation Level**: 3NF

**FacilityTable**

* **Attributes**: RoomFacilityId (PK), RoomId (FK referencing RoomTable), FacilityId (FK referencing FacilityTable)
* **Analysis**:
  + Represents the relationship between rooms and facilities without any additional attributes.
  + Each attribute depends on RoomFacilityId.
* **Normalisation Level**: 3NF

**Summary**

After this table-wise analysis, all tables are in **3NF**, which ensures data integrity and minimises redundancy. Here’s a recap of potential improvements applied in the normalisation process:

1. **1NF (First Normal Form)**:
   * Ensured each table has a unique primary key.
   * Removed repeating groups and organised attributes into individual columns.
2. **2NF (Second Normal Form)**:
   * Removed partial dependencies by ensuring all non-key attributes fully depend on the primary key.
3. **3NF (Third Normal Form)**:
   * Eliminated transitive dependencies, making each non-key attribute directly dependent only on the primary key.

This structure allows the **CheckInn** database to efficiently manage room reservations, guest information, staff roles, hotel details, payment transactions, and reviews.

**8. SQL Queries and Database Operations**

#### **8.1 Create Tables**

Provide SQL scripts for creating all tables.

===================== DATABASE & SCHEMA CREATION QUERIES ======================

-- CREATES THE CheckInn DATABASE

CREATE DATABASE CheckInn;

-- USES DATABASE

USE CheckInn;

--CREATING HOTEL ATTRIBUTES TABLE

CREATE TABLE Hotel(

HotelId INT NOT NULL PRIMARY KEY IDENTITY (100,1),

Name VARCHAR (255) NOT NULL,

Address VARCHAR (255) NOT NULL,

City VARCHAR (255) NOT NULL,

State VARCHAR (255) NOT NULL,

Country VARCHAR (255) NOT NULL,

PostalCode VARCHAR(20) NOT NULL,

PhoneNo VARCHAR(15) NOT NULL,

Email VARCHAR (255) NOT NULL,

NoOfRooms INT

);

-- FETCH ALL COLUMNS FROM Hotel TABLE

SELECT \* FROM Hotel

--CREATING CUSTOMER ATTRIBUTES TABLE

CREATE TABLE Customer(

CustomerId INT NOT NULL PRIMARY KEY IDENTITY (1,1),

FirstName VARCHAR (225) NOT NULL,

LastName VARCHAR (255),

DateOfBirth DATE NOT NULL,

PhoneNo VARCHAR(15) NOT NULL,

Email VARCHAR (255) NOT NULL,

Address VARCHAR (255) NOT NULL,

City VARCHAR (255) NOT NULL,

State VARCHAR (255) NOT NULL,

Country VARCHAR (255) NOT NULL,

PostalCode VARCHAR(20) NOT NULL

);

-- FETCH ALL COLUMNS FROM Customer TABLE

SELECT \* FROM Customer

-- CREATING ROOM ATTRIBUTES TABLE

CREATE TABLE Room(

RoomId INT NOT NULL PRIMARY KEY IDENTITY(1,1),

HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel(HotelId),

RoomNo INT NOT NULL,

RoomType VARCHAR (255) NOT NULL,

BedType VARCHAR (255) NOT NULL,

PricePerNight INT NOT NULL,

AvailabilityStatus VARCHAR(50) NOT NULL,

CHECK (AvailabilityStatus IN ('Available', 'Occupied', 'Reserved', 'Maintenance', 'Out of Service'))

);

-- FETCH ALL COLUMNS FROM Room TABLE

SELECT \* FROM Room

--CREATING RESERVATION ATTRIBUTES TABLE

CREATE TABLE Reservation (

ReservationId INT NOT NULL PRIMARY KEY IDENTITY(1000,1),

CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customer(CustomerId),

RoomId INT NOT NULL FOREIGN KEY REFERENCES Room(RoomId),

HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel(HotelId),

CheckIn DATE NOT NULL,

CheckOutDate DATE NOT NULL,

NoOfPeople INT NOT NULL,

TotalPrice DECIMAL(10,2),

BookingDate DATE,

ReservationStatus VARCHAR (50) NOT NULL,

CHECK ( ReservationStatus IN ('Available', 'Occupied', 'Reserved', 'Maintenance', 'Out of Service'))

);

-- FETCH ALL COLUMNS FROM Reservation TABLE

SELECT \* FROM Reservation

--CREATING PAYMENT ATTRIBUTES TABLE

CREATE TABLE Payment (

PaymentId INT NOT NULL PRIMARY KEY IDENTITY(10000,1),

PaymentDate DATE,

ReservationId INT NOT NULL FOREIGN KEY REFERENCES Reservation(ReservationId),

Amount DECIMAL(10,2) NOT NULL,

PaymentStatus VARCHAR(50) NOT NULL,

CHECK (PaymentStatus IN ('Pending', 'Completed', 'Failed', 'Refunded', 'Cancelled', 'Partially Refunded', 'Authorized', 'Declined')),

PaymentMethod VARCHAR(50) NOT NULL,

CHECK (PaymentMethod IN ('Credit Card', 'Debit Card', 'Cash', 'Bank Transfer', 'PayPal', 'Google Pay', 'Apple Pay', 'Amazon Pay', 'Cryptocurrency', 'Gift Card', 'Net Banking','Cheque'))

);

-- FETCH ALL COLUMNS FROM Payment TABLE

SELECT \* FROM Payment

--CREATING FACILITY ATTRIBUTES TABLE

CREATE TABLE Facility (

FacilityId INT NOT NULL PRIMARY KEY IDENTITY(10,1),

HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel(HotelId),

FacilityName VARCHAR(255) NOT NULL,

Description VARCHAR(500)

);

-- FETCH ALL COLUMNS FROM Facility TABLE

SELECT \* FROM Facility

--CREATING SERVICE ATTRIBUTES TABLE

CREATE TABLE Service (

ServiceId INT NOT NULL PRIMARY KEY IDENTITY(20,1),

ServiceName VARCHAR(50),

ServiceCost DECIMAL(10,2) NOT NULL,

HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel(HotelId)

);

-- FETCH ALL COLUMNS FROM Service TABLE

SELECT \* FROM Service

--CREATING STAFF ATTRIBUTES TABLE

CREATE TABLE Staff (

StaffId INT NOT NULL PRIMARY KEY IDENTITY(50,1),

HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel(HotelId),

FirstName VARCHAR(100) NOT NULL,

LastName VARCHAR(100) NOT NULL,

JobPosition VARCHAR(100) NOT NULL,

PhoneNo VARCHAR(15) NOT NULL,

Email VARCHAR(255) NOT NULL,

HireDate DATE NOT NULL,

Salary DECIMAL(10, 2) NOT NULL

);

-- FETCH ALL COLUMNS FROM Staff TABLE

SELECT \* FROM Staff

--CREATING REVIEW ATTRIBUTES TABLE

CREATE TABLE Review (

ReviewId INT NOT NULL PRIMARY KEY IDENTITY(1,1),

CustomerId INT NOT NULL FOREIGN KEY REFERENCES Customer(CustomerId),

HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel(HotelId),

Rating DECIMAL(3, 2) NOT NULL CHECK (Rating >= 0 AND Rating <= 5),

Remarks VARCHAR(500),

ReviewDate DATETIME NOT NULL DEFAULT GETDATE()

);

-- FETCH ALL COLUMNS FROM Review TABLE

SELECT \* FROM Review

#### **8.2 Insert Data**

Show SQL commands for inserting data into each table.

-- =========== INSERTS DATA INTO HOTEL TABLE ===========

INSERT INTO Hotel (Name, Address, City, State, Country, PostalCode, PhoneNo, Email, NoOfRooms) VALUES

('Hotel Azure', '123, Ocean Drive, Juhu, Mumbai', 'Mumbai', 'Maharashtra', 'India', '400049', '9876543210', 'hotelazure@gmail.com', 100),

('Royal Palace', '456, Kingsway, Connaught Place, New Delhi', 'New Delhi', 'Delhi', 'India', '110001', '9876543211', 'royalpalace@gmail.com', 200),

('Green Valley Resort', '789, Nature Lane, Munnar', 'Munnar', 'Kerala', 'India', '685612', '9876543212', 'greenvalley@gmail.com', 150),

('Beachside Inn', '321, Sunset Boulevard, Goa', 'Goa', 'Goa', 'India', '403507', '9876543213', 'beachsideinn@gmail.com', 75),

('Desert Camp', '654, Sand Dunes, Jaisalmer', 'Jaisalmer', 'Rajasthan', 'India', '345678', '9876543214', 'desertcamp@gmail.com', 50),

('Mountain Retreat', '987, Hilltop Road, Shimla', 'Shimla', 'Himachal Pradesh', 'India', '171001', '9876543215', 'mountainretreat@gmail.com', 100),

('City Center Hotel', '123, MG Road, Bengaluru', 'Bengaluru', 'Karnataka', 'India', '560001', '9876543216', 'citycenter@gmail.com', 120),

('Heritage Lodge', '456, Old Fort Road, Agra', 'Agra', 'Uttar Pradesh', 'India', '282001', '9876543217', 'heritagelodge@gmail.com', 60),

('Seaside Haven', '789, Ocean View, Chennai', 'Chennai', 'Tamil Nadu', 'India', '600001', '9876543218', 'seasidehaven@gmail.com', 80),

('Urban Suites', '321, Main Street, Hyderabad', 'Hyderabad', 'Telangana', 'India', '500001', '9876543219', 'urbansuites@gmail.com', 90);

/\* =========== INSERTS DATA INTO CUSTOMER TABLE =========== \*/

INSERT INTO Customer (FirstName, LastName, DateOfBirth, PhoneNo, Email, Address, City, State, Country, PostalCode) VALUES

('Aarav', 'Singh', '1995-05-12', '9876543210', 'aaravsingh@gmail.com', '123, Maple Street', 'Patna', 'Bihar', 'India', '800001'),

('Vivaan', 'Sharma', '1988-07-25', '9876543211', 'vivansharma@gmail.com', '456, Oak Avenue', 'Mumbai', 'Maharashtra', 'India', '400002'),

('Dhruv', 'Verma', '1994-04-22', '9876543217', 'dhruvverma@gmail.com', '567, Palm Avenue', 'Jaipur', 'Rajasthan', 'India', '302008'),

('Reyansh', 'Chopra', '1996-10-11', '9876543218', 'reyanshchopra@gmail.com', '890, Mango Street', 'Lucknow', 'Uttar Pradesh', 'India', '226009'),

('Krishna', 'Joshi', '1989-08-05', '9876543219', 'krishnajoshi@gmail.com', '321, Lotus Lane', 'Varanasi', 'Uttar Pradesh', 'India', '221010'),

('Kaira', 'Pillai', '1994-09-14', '9876543226', 'kairapillai@gmail.com', '654, Tulip Avenue', 'Nagpur', 'Maharashtra', 'India', '440017'),

('Nia', 'Patel', '1999-04-09', '9876543227', 'niapatel@gmail.com', '987, Lotus Lane', 'Vadodara', 'Gujarat', 'India', '390018');

/\* =========== INSERTS DATA INTO Room TABLE =========== \*/

INSERT INTO Room (HotelId, RoomNo, RoomType, BedType, PricePerNight, AvailabilityStatus) VALUES

(100, 101, 'Deluxe', 'King', 5000, 'Available'),

(100, 102, 'Standard', 'Queen', 3500, 'Occupied'),

(101, 201, 'Suite', 'King', 8000, 'Reserved'),

(101, 202, 'Deluxe', 'Twin', 5500, 'Available'),

(109, 1001, 'Suite', 'Twin', 7200, 'Available'),

(132, 3301, 'Economy', 'Twin', 2800, 'Maintenance'),

(132, 3302, 'Luxury', 'King', 9000, 'Out of Service'),

(133, 3401, 'Suite', 'King', 7500, 'Available'),

(133, 3402, 'Deluxe', 'Queen', 5500, 'Occupied'),

(134, 3501, 'Standard', 'Double', 3800, 'Reserved'),

(134, 3502, 'Economy', 'Twin', 2700, 'Available');

/\* =========== INSERTS DATA INTO Reservation TABLE =========== \*/

INSERT INTO Reservation (CustomerId, RoomId, HotelId, CheckIn, CheckOutDate, NoOfPeople, TotalPrice, BookingDate, ReservationStatus) VALUES

(1, 1, 100, '2024-10-01', '2024-10-05', 2, 20000.00, '2024-09-28', 'Reserved'),

(2, 2, 101, '2024-10-10', '2024-10-12', 1, 7000.00, '2024-10-05', 'Available'),

(3, 3, 102, '2024-10-20', '2024-10-23', 3, 15000.00, '2024-10-10', 'Reserved'),

(4, 4, 103, '2024-11-01', '2024-11-07', 2, 42000.00, '2024-10-18', 'Available'),

(5, 5, 104, '2024-10-05', '2024-10-08', 4, 18000.00, '2024-10-02', 'Reserved'),

(6, 6, 105, '2024-10-11', '2024-10-15', 2, 16000.00, '2024-10-01', 'Occupied'),

(7, 7, 106, '2024-10-20', '2024-10-22', 1, 8000.00, '2024-10-15', 'Available'),

(8, 8, 107, '2024-11-01', '2024-11-05', 3, 24000.00, '2024-10-18', 'Reserved'),

(9, 9, 108, '2024-10-22', '2024-10-25', 2, 15000.00, '2024-10-15', 'Reserved'),

(35, 35, 134, '2024-10-20', '2024-10-23', 2, 16000.00, '2024-10-15', 'Available');

/\* =========== INSERTS DATA INTO Payment TABLE =========== \*/

INSERT INTO Payment (PaymentDate, ReservationId, Amount, PaymentStatus, PaymentMethod) VALUES

('2024-09-28', 1000, 20000.00, 'Completed', 'Credit Card'),

('2024-10-05', 1001, 7000.00, 'Completed', 'Debit Card'),

('2024-10-05', 1017, 12000.00, 'Completed', 'Cash'),

('2024-10-30', 1018, 17000.00, 'Cancelled', 'Net Banking'),

('2024-10-14', 1019, 25000.00, 'Completed', 'Debit Card'),

('2024-10-20', 1020, 16000.00, 'Partially Refunded', 'Credit Card'),

('2024-10-15', 1021, 18000.00, 'Completed', 'PayPal'),

('2024-10-07', 1022, 9000.00, 'Declined', 'Google Pay'),

('2024-10-25', 1023, 12000.00, 'Completed', 'Bank Transfer'),

('2024-10-10', 1025, 8000.00, 'Refunded', 'Gift Card');

/\* =========== INSERTS DATA INTO Facility TABLE =========== \*/

INSERT INTO Facility (HotelId, FacilityName, Description) VALUES

(100, 'Swimming Pool', 'An outdoor pool with temperature control and lifeguard service available.'),

(101, 'Gym', 'Fully equipped gym with personal trainers available from 6 AM to 10 PM.'),

(102, 'Spa', 'A luxury spa offering ayurvedic treatments and massages.'),

(103, 'Restaurant', 'Multi-cuisine restaurant serving Indian, Continental, and Chinese food.');

/\* =========== INSERTS DATA INTO Service TABLE =========== \*/

INSERT INTO Service (ServiceName, ServiceCost, HotelId) VALUES

('Room Cleaning', 500.00, 100),

('Laundry', 300.00, 101),

('In-Room Dining', 800.00, 102),

('Spa Treatment', 2500.00, 103),

('Electric Vehicle Charging', 200.00, 134),

('Banquet Hall Use', 15000.00, 135);

/\* =========== INSERTS DATA INTO Staff TABLE =========== \*/

INSERT INTO Staff (HotelId, FirstName, LastName, JobPosition, PhoneNo, Email, HireDate, Salary) VALUES

(100, 'Rajesh', 'Sharma', 'General Manager', '9876543210', 'rajesh.sharma@gmail.com', '2019-01-10', 80000.00),

(101, 'Priya', 'Verma', 'Receptionist', '9867554321', 'priya.verma@gmail.com', '2021-03-15', 25000.00),

(102, 'Amit', 'Singh', 'Housekeeping Manager', '9876541234', 'amit.singh@gmail.com', '2018-07-22', 40000.00),

(103, 'Suman', 'Kumar', 'Chef', '9876545678', 'suman.kumar@gmail.com', '2020-08-05', 60000.00);

/\* =========== INSERTS DATA INTO Review TABLE =========== \*/

INSERT INTO Review (CustomerId, HotelId, Rating, Remarks, ReviewDate) VALUES

(1, 100, 4.50, 'Excellent service and clean rooms.', '2024-01-15 12:30:00'),

(2, 101, 3.75, 'Good stay, but the food could have been better.', '2024-02-12 09:45:00'),

(3, 102, 4.00, 'Nice location and friendly staff.', '2024-03-22 14:20:00'),

(4, 103, 2.50, 'Rooms need renovation. Not worth the price.', '2024-04-05 17:50:00'),

(5, 104, 5.00, 'Amazing experience! Will visit again.', '2024-05-18 10:15:00'),

(6, 105, 3.25, 'Decent stay but had some issues with AC.', '2024-06-21 11:40:00');

**8.3 Queries for Data Retrieval**

**Provide queries to demonstrate CRUD operations**.

-- 1. SQL Query for displaying the each room type, list the total number of bookings and the average revenue per booking

SELECT Room.RoomType, COUNT(Reservation.ReservationId) AS total\_bookings, AVG(Payment.Amount) AS average\_revenue\_per\_booking

FROM Room

JOIN Reservation ON Room.RoomId = Reservation.RoomId

JOIN Payment ON Reservation.ReservationId = Payment.ReservationId

GROUP BY Room.RoomType;

-- 2. Sql query to Find the city and states of the branches of the hotels, the first and last names of the customers

-- who have stayed in each hotel branch and has reserved a particular room in which the month of Booking and Checking are same

SELECT h.City, h.State, c.FirstName, c.LastName, re.ReservationId, MONTH(re.BookingDate) AS BookingMonth, MONTH(re.CheckIn) AS CheckInMonth

FROM Hotel h JOIN Reservation re ON h.HotelId = re.HotelId

JOIN Customer c ON re.CustomerId = c.CustomerId

WHERE MONTH(re.BookingDate) = MONTH(re.CheckIn);

-- 3. Write sql query to Retrieve the list of guests who have never booked and reserved a room and made payment below a specific rate

-- (say Rs 1000)

SELECT c.CustomerId,c.FirstName,c.LastName,c.PhoneNo,c.Email

FROM Customer c LEFT JOIN Reservation re ON c.CustomerId = re.CustomerId

LEFT JOIN Payment p ON re.ReservationId = p.ReservationId

WHERE re.ReservationId IS NULL -- Ensures the customer has never booked or reserved a room

OR (p.Amount < 1000 AND re.ReservationId IS NOT NULL); -- Checks for payments below Rs 800 if a reservation exists

-- 4. sql query to Display the most common room type booked by guests from each country and their ratings

SELECT

c.Country,

r.RoomType,

COUNT(re.ReservationId) AS total\_bookings,

AVG(rv.Rating) AS average\_rating

FROM

Customer c

JOIN

Reservation re ON c.CustomerId = re.CustomerId

JOIN

Room r ON re.RoomId = r.RoomId

LEFT JOIN

Review rv ON re.CustomerId = rv.CustomerId AND re.HotelId = rv.HotelId

GROUP BY

c.Country, r.RoomType

HAVING

COUNT(re.ReservationId) = (

SELECT MAX(total\_bookings)

FROM (

SELECT

COUNT(sub\_re.ReservationId) AS total\_bookings

FROM

Customer sub\_c

JOIN

Reservation sub\_re ON sub\_c.CustomerId = sub\_re.CustomerId

JOIN

Room sub\_r ON sub\_re.RoomId = sub\_r.RoomId

WHERE

sub\_c.Country = c.Country

GROUP BY

sub\_r.RoomType

) AS sub\_query

)

ORDER BY

c.Country, total\_bookings DESC;

-- 5. Write the Sql query to display the most common room type booked by guests along with their names

-- and the maximum no of rooms booked for a particular room type from each country

SELECT

c.Country,

c.FirstName,

c.LastName,

r.RoomType,

COUNT(re.ReservationId) AS total\_bookings

FROM

Customer c

JOIN

Reservation re ON c.CustomerId = re.CustomerId

JOIN

Room r ON re.RoomId = r.RoomId

GROUP BY

c.Country, c.FirstName, c.LastName, r.RoomType

HAVING

COUNT(re.ReservationId) = (

SELECT MAX(room\_booking\_count)

FROM (

SELECT

sub\_c.Country,

sub\_r.RoomType,

COUNT(sub\_re.ReservationId) AS room\_booking\_count

FROM

Customer sub\_c

JOIN

Reservation sub\_re ON sub\_c.CustomerId = sub\_re.CustomerId

JOIN

Room sub\_r ON sub\_re.RoomId = sub\_r.RoomId

GROUP BY

sub\_c.Country, sub\_r.RoomType

) AS sub\_query

WHERE sub\_query.Country = c.Country

)

ORDER BY

c.Country, total\_bookings DESC;

-- 6. Write the Sql query to Find each hotel branch's least popular room type based on the total number of bookings and display the amount paid

-- by each customer according to room type for each hotel

WITH RoomBookings AS (

SELECT h.HotelId, h.Name AS HotelName, r.RoomType, COUNT(re.ReservationId) AS total\_bookings, SUM(p.Amount) AS total\_paid

FROM Hotel h JOIN Room r ON h.HotelId = r.HotelId

LEFT JOIN Reservation re ON r.RoomId = re.RoomId

LEFT JOIN Payment p ON re.ReservationId = p.ReservationId

GROUP BY h.HotelId, h.Name, r.RoomType

),

LeastPopularRoomTypes AS (

SELECT HotelId, MIN(total\_bookings) AS least\_bookings

FROM RoomBookings GROUP BY HotelId

)

SELECT rb.HotelName, rb.RoomType, rb.total\_bookings, rb.total\_paid

FROM RoomBookings rb

JOIN LeastPopularRoomTypes lpr ON rb.HotelId = lpr.HotelId AND rb.total\_bookings = lpr.least\_bookings

ORDER BY rb.HotelName, rb.RoomType;

-- 7. Write the Sql query to List each guests total spend across all hotel branches booked and reserved and the type of rooms theyve stayed

-- in and the total payment for each room in each hotel reserved and booked

SELECT c.CustomerId,c.FirstName,c.LastName, h.Name AS HotelName, r.RoomType, SUM(p.Amount) AS TotalPayment, COUNT(re.ReservationId) AS TotalBookings

FROM Customer c JOIN Reservation re ON c.CustomerId = re.CustomerId

JOIN Room r ON re.RoomId = r.RoomId

JOIN Hotel h ON r.HotelId = h.HotelId

JOIN Payment p ON re.ReservationId = p.ReservationId

GROUP BY c.CustomerId, c.FirstName, c.LastName, h.Name, r.RoomType

ORDER BY c.CustomerId, h.Name;

-- 8. write a query to find, for each month, list the branch with the highest total revenue, showing the no of customers per hotel reserved and booked with showing

-- only months where revenue is less than 50,00

WITH MonthlyRevenue AS (

SELECT h.HotelId,h.Name AS HotelName, MONTH(re.CheckIn) AS BookingMonth, YEAR(re.CheckIn) AS BookingYear,SUM(p.Amount) AS TotalRevenue, COUNT(DISTINCT re.CustomerId) AS TotalCustomers

FROM Hotel h

JOIN Room r ON h.HotelId = r.HotelId

JOIN Reservation re ON r.RoomId = re.RoomId

JOIN Payment p ON re.ReservationId = p.ReservationId

GROUP BY h.HotelId, h.Name, MONTH(re.CheckIn), YEAR(re.CheckIn)

),

RankedRevenue AS (

SELECT BookingMonth,BookingYear,HotelName,TotalRevenue,TotalCustomers,RANK() OVER (PARTITION BY BookingYear, BookingMonth ORDER BY TotalRevenue DESC) AS RevenueRank

FROM MonthlyRevenue

)

SELECT BookingMonth,BookingYear,HotelName,TotalRevenue,TotalCustomers

FROM RankedRevenue

WHERE RevenueRank = 1 AND TotalRevenue < 50000

ORDER BY BookingYear, BookingMonth;

-- 9. write a query to find the guests who have booked the most popular room type of each hotel in the hotel system and the rating of the hotel

-- according to the most popular room type of each hotel

WITH PopularRoomType AS (

SELECT h.HotelId,h.Name AS HotelName,r.RoomType,COUNT(re.ReservationId) AS TotalBookings,RANK() OVER (PARTITION BY h.HotelId ORDER BY COUNT(re.ReservationId) DESC) AS RoomRank

FROM Hotel h JOIN Room r ON h.HotelId = r.HotelId

JOIN Reservation re ON r.RoomId = re.RoomId

GROUP BY h.HotelId, h.Name, r.RoomType

),

GuestsWithPopularRoom AS (

SELECT c.CustomerId, c.FirstName,c.LastName,h.HotelId,h.Name AS HotelName,r.RoomType,r.PricePerNight, hr.Rating

FROM PopularRoomType prt

JOIN Room r ON prt.HotelId = r.HotelId AND prt.RoomType = r.RoomType

JOIN Reservation re ON r.RoomId = re.RoomId

JOIN Customer c ON re.CustomerId = c.CustomerId

JOIN Hotel h ON prt.HotelId = h.HotelId

JOIN Review hr ON h.HotelId = hr.HotelId AND r.RoomType = hr.RoomType

WHERE prt.RoomRank = 1

)

SELECT g.CustomerId,g.FirstName,g.LastName,g.HotelName,g.RoomType,g.PricePerNight,AVG(g.Rating) AS AverageRating

FROM GuestsWithPopularRoom g

GROUP BY g.CustomerId, g.FirstName, g.LastName, g.HotelName, g.RoomType, g.PricePerNight

ORDER BY g.HotelName, g.RoomType;

-- 10. Write a SQL query to find the no of customers , their names and average price paid by each customer according to each room type in each hotel branch

-- and the total payment made by the customer after booking and reserving a particular room on the date of CheckIn and CheckOut who stayed in each hotel branch.

-- Also list out the Facilities provided by each hotel and the name of the staff and count the no of staffs for each hotel and according to the facilities provided

-- by each hotel branch , display the reviews of each hotel along with the maximum average rating of a particular hotel branch according to its room and Facilities

WITH CustomerPayments AS (

SELECT c.CustomerId,c.FirstName,c.LastName,h.HotelId,h.Name AS HotelName,r.RoomType,AVG(p.Amount) AS AveragePricePaid,SUM(p.Amount) AS TotalPayment,COUNT(DISTINCT re.ReservationId) AS TotalBookings

FROM Customer c

JOIN Reservation re ON c.CustomerId = re.CustomerId

JOIN Room r ON re.RoomId = r.RoomId

JOIN Hotel h ON r.HotelId = h.HotelId

JOIN Payment p ON re.ReservationId = p.ReservationId

GROUP BY c.CustomerId, c.FirstName, c.LastName, h.HotelId, h.Name, r.RoomType

),

HotelFacilities AS (

SELECT h.HotelId,h.Name AS HotelName,f.FacilityName,COUNT(s.StaffId) AS StaffCount

FROM Hotel h LEFT JOIN Facility f ON h.HotelId = f.HotelId

LEFT JOIN Staff s ON h.HotelId = s.HotelId GROUP BY h.HotelId, h.Name, f.FacilityName

),

HotelReviews AS (

SELECT h.HotelId,AVG(r.Rating) AS AverageRating,COUNT(r.ReviewId) AS ReviewCount

FROM Hotel h

LEFT JOIN Review r ON h.HotelId = r.HotelId GROUP BY h.HotelId

)

SELECT cp.CustomerId,cp.FirstName,cp.LastName,cp.HotelName,cp.RoomType,cp.AveragePricePaid,cp.TotalPayment,hf.FacilityName,hf.StaffCount,hr.AverageRating,hr.ReviewCount

FROM CustomerPayments cp

JOIN HotelFacilities hf ON cp.HotelId = hf.HotelId

JOIN HotelReviews hr ON cp.HotelId = hr.HotelId

ORDER BY cp.HotelName, cp.CustomerId;

**Functions for Data Retrieval:**

**Function to calculate Total Revenue for a Hotel**

/\* This function calculates the total revenue generated by a specific hotel from completed payments. \*/

CREATE FUNCTION GetTotalRevenueByHotel (@HotelId INT)

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @TotalRevenue DECIMAL(10, 2);

SELECT

@TotalRevenue = SUM(Amount)

FROM

Payment P

JOIN

Reservation R ON P.ReservationId = R.ReservationId

WHERE

R.HotelId = @HotelId

AND P.PaymentStatus = 'Completed';

RETURN ISNULL(@TotalRevenue, 0);

END;

**Function Call**

DECLARE @TotalRevenue DECIMAL(10, 2);

SELECT @TotalRevenue = dbo.GetTotalRevenueByHotel(101); -- Replace 101 with the desired HotelId

PRINT 'Total Revenue: ' + CAST(@TotalRevenue AS VARCHAR);

**Function to get Full Customer Name**

/\* This function returns a customer’s full name given their CustomerId. \*/

CREATE FUNCTION GetCustomerFullName (@CustomerId INT)

RETURNS VARCHAR(255)

AS

BEGIN

DECLARE @FullName VARCHAR(255);

SELECT

@FullName = FirstName + ' ' + LastName

FROM

Customer

WHERE

CustomerId = @CustomerId;

RETURN @FullName;

END;

**Function Call**

DECLARE @FullName VARCHAR(255);

SELECT @FullName = dbo.GetCustomerFullName(1); -- Replace 1 with the desired CustomerId

PRINT 'Customer Full Name: ' + @FullName;

**Function to check Room Availability**

/\* This function checks if a specific room is available. \*/

CREATE FUNCTION IsRoomAvailable (@RoomId INT)

RETURNS BIT

AS

BEGIN

DECLARE @IsAvailable BIT;

SELECT

@IsAvailable = CASE WHEN AvailabilityStatus = 'Available' THEN 1 ELSE 0 END

FROM

Room

WHERE

RoomId = @RoomId;

RETURN @IsAvailable;

END;

**Function Call**

-- This function checks if a room is available. It returns 1 for available and 0 otherwise.

DECLARE @IsAvailable BIT;

SELECT @IsAvailable = dbo.IsRoomAvailable(10); -- Replace 10 with the desired RoomId

IF @IsAvailable = 1

PRINT 'The room is available.';

ELSE

PRINT 'The room is not available.';

**Stored Procedures:**

**Non-Parameterized Stored Procedure**

**This procedure retrieves all hotel records.**

CREATE PROCEDURE GetAllHotels

AS

BEGIN

SELECT \* FROM Hotel;

END;

**Execute:**

EXEC GetAllHotels;

**Parameterized Stored Procedure**

**This procedure retrieves reservation details for a specific customer by CustomerId.**

CREATE PROCEDURE GetReservationsByCustomer

@CustomerId INT

AS

BEGIN

SELECT

R.ReservationId,

R.RoomId,

R.HotelId,

R.CheckIn,

R.CheckOutDate,

R.NoOfPeople,

R.TotalPrice,

R.ReservationStatus

FROM

Reservation R

WHERE

R.CustomerId = @CustomerId;

END;

**Execute:**

EXEC GetReservationsByCustomer @CustomerId = 1;

**Try-Catch Stored Procedure**

**This procedure attempts to update a room’s availability status and handles any errors that occur.**

CREATE PROCEDURE UpdateRoomAvailability

@RoomId INT,

@NewStatus VARCHAR(50)

AS

BEGIN

BEGIN TRY

-- Update room availability status

UPDATE Room

SET AvailabilityStatus = @NewStatus

WHERE RoomId = @RoomId;

PRINT 'Room availability status updated successfully.';

END TRY

BEGIN CATCH

-- Error handling

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);

END CATCH

END;

**Execute:**

EXEC UpdateRoomAvailability @RoomId = 1, @NewStatus = 'Occupied';

#### **Cursors:**

#### **1. Cursor to List Available Rooms for Each Hotel**

**This cursor iterates through each hotel and lists all available rooms.**

DECLARE @HotelId INT, @HotelName VARCHAR(255), @RoomNo INT;

-- Declare the cursor

DECLARE AvailableRoomsCursor CURSOR FOR

SELECT

HotelId,

Name

FROM

Hotel;

-- Open the cursor

OPEN AvailableRoomsCursor;

-- Fetch the first record

FETCH NEXT FROM AvailableRoomsCursor INTO @HotelId, @HotelName;

-- Loop through the cursor

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'Hotel: ' + @HotelName + ' (ID: ' + CAST(@HotelId AS VARCHAR) + ')';

-- List available rooms for the current hotel

SELECT

RoomNo, RoomType, PricePerNight

FROM

Room

WHERE

HotelId = @HotelId AND AvailabilityStatus = 'Available';

-- Fetch the next record

FETCH NEXT FROM AvailableRoomsCursor INTO @HotelId, @HotelName;

END;

-- Close and deallocate the cursor

CLOSE AvailableRoomsCursor;

DEALLOCATE AvailableRoomsCursor;

#### **2. Cursor to Calculate Total Reservations for Each Customer**

**This cursor calculates the total number of reservations made by each customer.**

DECLARE @CustomerId INT, @CustomerName VARCHAR(255), @TotalReservations INT;

-- Declare the cursor

DECLARE CustomerReservationsCursor CURSOR FOR

SELECT

CustomerId,

FirstName + ' ' + LastName AS CustomerName

FROM

Customer;

-- Open the cursor

OPEN CustomerReservationsCursor;

-- Fetch the first record

FETCH NEXT FROM CustomerReservationsCursor INTO @CustomerId, @CustomerName;

-- Loop through the cursor

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Calculate total reservations for the current customer

SELECT @TotalReservations = COUNT(\*)

FROM Reservation

WHERE CustomerId = @CustomerId;

PRINT 'Customer: ' + @CustomerName + ', Total Reservations: ' + CAST(@TotalReservations AS VARCHAR);

-- Fetch the next record

FETCH NEXT FROM CustomerReservationsCursor INTO @CustomerId, @CustomerName;

END;

-- Close and deallocate the cursor

CLOSE CustomerReservationsCursor;

DEALLOCATE CustomerReservationsCursor;

**9. Conclusion**

**Project Aim:**

The primary aim of this project was to design and implement a structured, efficient, and scalable database for managing hotel operations. The database was created to store and organise essential data about hotels, rooms, customers, bookings, and related information, facilitating streamlined hotel management and data retrieval.

**Problem Objective:**

The project addressed the need for a comprehensive database system to manage data and optimise operations within a hotel environment. By organising data on hotels, bookings, customers, and services, the system enables accurate data handling, minimises redundancies, and provides ease of access for hotel management and staff.

**Problem Statement:**

Managing hotel data manually or through disparate systems leads to inefficiencies, inconsistencies, and data silos. This project aimed to centralise data within a single relational database, ensuring consistent, reliable, and accessible information across various hotel management functions

**Database Creation and Structure:**The database comprises multiple interconnected tables representing essential hotel management entities, including:

* Hotel Table: Captures details of each hotel, such as ID, name, location, contact details, and room capacity.
* Room Table: Records room information, including room number, type, rate, and status.
* Customer Table: Stores customer details, including personal information and contact data.
* Booking Table: Maintains records of reservations, linking customers to specific rooms and timeframes.

These entities are connected through well-defined relationships, allowing for efficient management of bookings, customer information, and room availability across various hotel locations.

**Project Achievements and Objectives Fulfilment:  
The completed system successfully fulfils the objectives set at the project's outset by:**

1. Improving Data Management: All essential information is organised within structured tables, reducing redundancy and ensuring data consistency.
2. Enhancing Accessibility and Reliability: The relational database enables quick access to critical information, supporting hotel staff in decision-making and customer service.
3. Streamlining Operations: By centralising data on bookings, customers, and rooms, the database facilitates a smoother workflow for tasks such as booking management, room allocation, and customer handling.
4. Scalability and Future Extensions: The database is designed to accommodate additional tables or attributes as needed, allowing for easy expansion to support future functionalities like loyalty programs or event management.

In summary, this hotel management database project has created a robust, reliable, and efficient system that supports core hotel operations. It provides a scalable solution for centralising and managing hotel data, thereby fulfilling the project's objectives of improving data management, accessibility, and operational efficiency. ​​