Table of Contents

INTRODUCTION

In today's competitive hospitality industry, providing exceptional guest experiences is more important than ever, and a robust hotel booking system is essential to achieving this goal. Our Hotel Booking System is designed to enhance the overall guest journey by focusing on three core pillars: personalized service, seamless reservation, and efficient hotel management. By offering personalized service, the system tailors guest experiences based on individual preferences, enabling hotels to create memorable stays through data-driven insights and customized recommendations. The seamless reservation process ensures that guests can easily book rooms, manage their reservations, and receive instant confirmations, whether through the hotel's website or third-party platforms. For hotel management teams, the system streamlines day-to-day operations by automating tasks such as room inventory management, payment processing, and guest request handling. This allows staff to focus on delivering high-quality service and improving overall efficiency. Ultimately, our Hotel Booking System is designed to not only elevate the guest experience but also optimize hotel operations, ensuring long-term customer satisfaction and operational success.

MISSION STATEMENT

Our mission is to deliver unforgettable, personalized experiences through innovative technology and sustainable practices. We aim to create lasting impressions by tailoring every stay to individual preferences and supporting local communities. By empowering our team with growth opportunities, we foster a culture of excellence and ensure exceptional service for every guest.

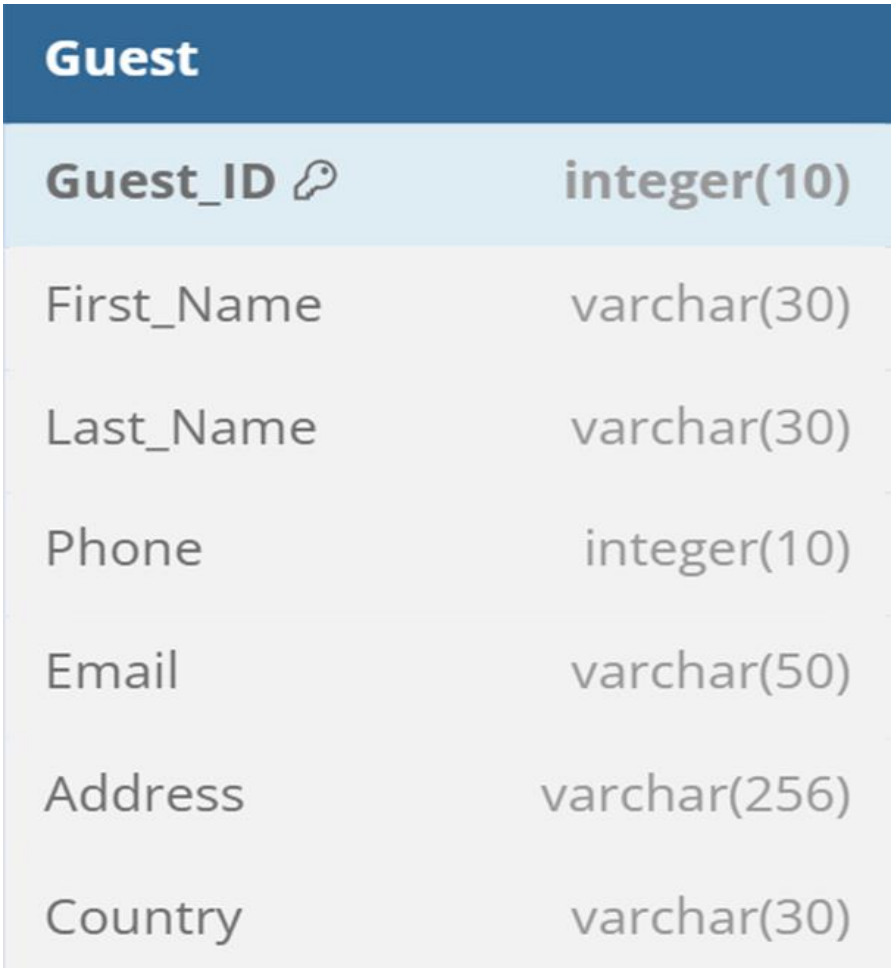
\

OBJECTIVES

- **Improve Guest Engagement**: Use data-driven insights to create personalized guest interactions, enhancing loyalty and satisfaction.
- **Streamline Communication**: Integrate messaging systems to facilitate smooth communication between guests and staff for real-time assistance.
- **Boost Operational Efficiency**: Automate repetitive tasks like billing and housekeeping requests to improve workflow and reduce human error.
- **Enhance Data Security**: Implement robust security measures to ensure guest data is protected and compliant with privacy regulations.
- **Increase Direct Bookings**: Offer incentives or discounts for direct bookings through the hotel's own website to reduce dependency on third-party platforms.
- **Support Sustainability**: Integrate eco-friendly initiatives such as energy-saving features and paperless check-ins to reduce the hotel's environmental impact.
- **Maximize Revenue**: Utilize dynamic pricing tools to adjust room rates based on demand, ensuring competitive pricing and higher occupancy.
- **Expand Market Reach**: Integrate with global distribution systems (GDS) to attract a wider audience and increase bookings from diverse markets.

TABLES

GUEST TABLE:



Explanation:

This table records guest details, including personal information (name, phone, email), location (address, country), and a unique identifier (Guest_ID). It likely supports systems like hotel management or event booking, with data types optimized for efficient storage.

- **Guest_ID**:

  Data Type: integer(10)

Purpose: Serves as the primary key, uniquely identifying each guest.**Key Icon**: Indicates that this column is the primary key.

First_Name:

Data Type: varchar(30)
Purpose: Stores the first name of the guest.

Length: Up to 30 characters.

- **Last_Name**:
  Data Type: varchar(30)
  Purpose: Stores the last name of the guest.
  Length: Up to 30 characters.
- **Phone**:
  Data Type: integer(10)
  Purpose: Stores the guest's phone number.
  Limit: Holds numbers with a maximum of 10 digits (assumes integers only).
- **Email**:
  Data Type: varchar(50)
  Purpose: Stores the guest's email address.
  Length: Up to 50 characters.
- **Address**:

  Data Type: varchar(256)

  Purpose: Stores the guest's full address.

  Length: Up to 256 characters.

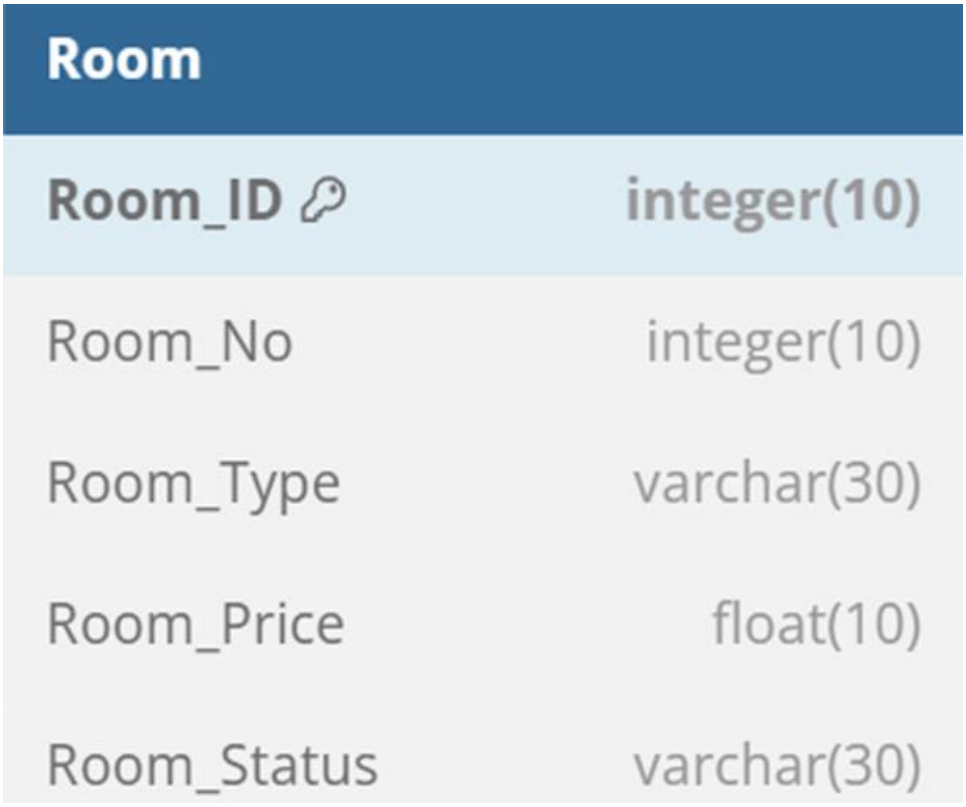
- **Country**:

  Data Type: varchar(30)

  Purpose: Stores the country of the guest.

  Length: Up to 30 characters.

ROOM TABLE:



Explanation:

The **"Room"** table is designed to manage detailed information about the rooms in a system, such as for a hotel or property management platform.

- **Room_ID**:

  Acts as the primary key.

  Uniquely identifies each room in the database.

  Ensures no duplication of room records.

- **Room_No**:

  Represents the actual room number assigned within the facility.

Allows for easy mapping of database entries to real-world rooms.

- **Room_Type**:

  Categorizes the room based on type, such as "Single", "Double", "Suite", or other descriptors.

  Helps in filtering and querying rooms based on guest requirements.

- **Room_Price**:

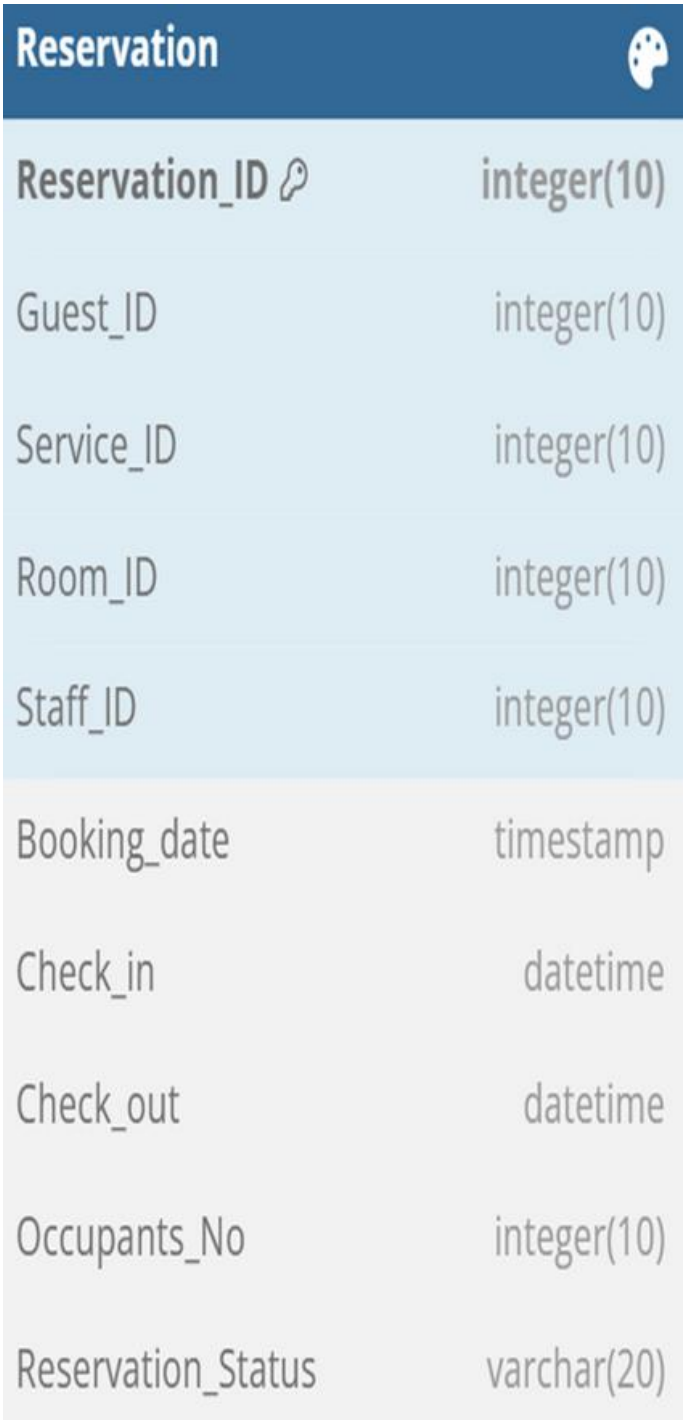
  Stores the cost of booking the room.

  Using a float data type supports pricing with decimal values, such as $100.50.

- **Room_Status**:

- Indicates the current state of the room (e.g., "Available", "Occupied", "Under Maintenance").
- Useful for operational purposes, such as assigning rooms to guests or marking them for service.

RESERVATION TABLE:



Explaination:

The **"Reservation"** table is designed to manage the reservations made for rooms or services in a system, typically in a hotel or booking platform.

- **Reservation_ID:**

  Serves as the primary key for the table.

  Uniquely identifies each reservation.

  Prevents duplication of reservation entries.

- **Guest_ID:**

  A foreign key linking to the "Guest" table (not shown here).

  Represents the unique identifier for the guest making the reservation.

  Helps track reservations made by a specific guest.

- **Service_ID:**

  A foreign key linking to a "Services" table (if present).

  Refers to any additional services (e.g., spa, room service, or t transportation) associated with the reservation.

  Allows tracking of extra amenities.

- **Room_ID:**

  A foreign key referencing the "Room" table.

  Specifies the room reserved by the guest.

  Ensures a valid room is associated with the reservation.

- **Staff_ID:**

  A foreign key referencing a "Staff" table (if present).

  Identifies the staff member managing or confirming the reservation.

  Helps in assigning responsibilities for reservation handling.

- **Booking_date**:

  A timestamp field storing the exact date and time when the reservation was created.

  Useful for auditing and reporting purposes.

- **Check_in:**

  A datetime field indicating the check-in date and time for the reservation.

  Ensures accurate planning for room availability.

- **Check_out:**

  A datetime field representing the check-out date and time.

  Helps calculate the duration of stay and associated charges.

- **Occupants_No:**

  An integer field storing the number of people included in the reservation.

  Important for ensuring the room's capacity is not exceeded.

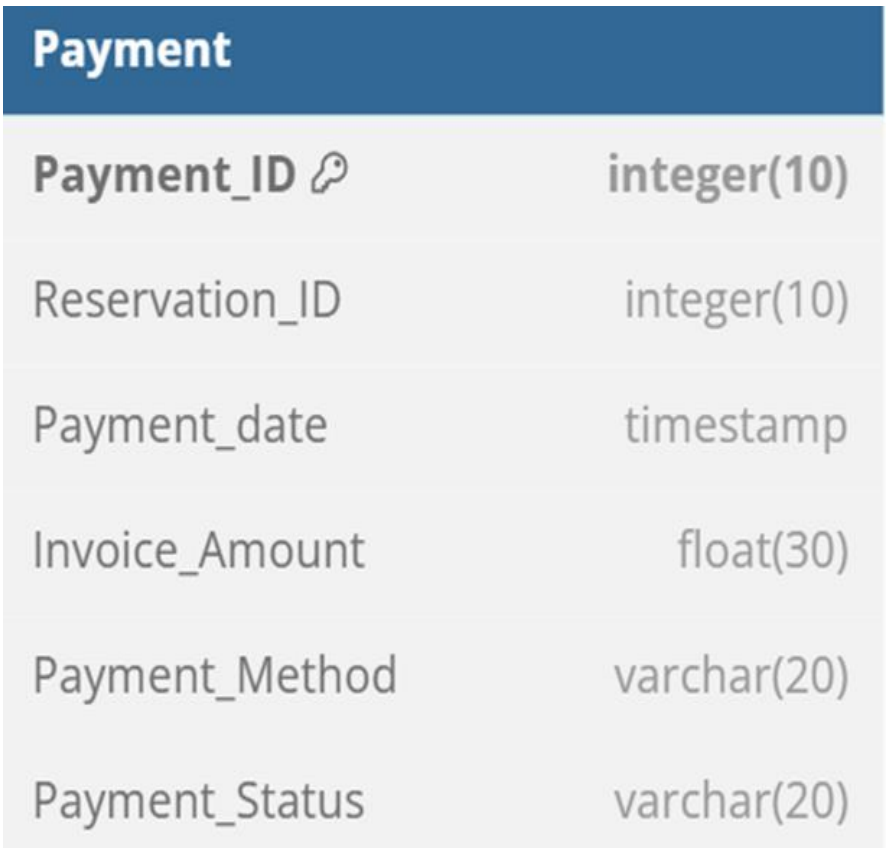
- **Reservation_Status:**

  A varchar(20) field indicating the current status of the reservation.

  Possible values might include "Confirmed," "Pending," "Cancelled," or "Completed."

  Useful for tracking the reservation lifecycle.

PAYMENT TABLE:



Explanation:

This **Payment** table is part of a relational database and is used to store information about financial transactions related to reservations. Below is a more detailed explanation of its structure, purpose, and functionality.

- **Payment_ID (integer(10), Primary Key 🔑)**

  A unique identifier for each payment record.

  Defined as an integer with a length of 10.

- **Reservation_ID (integer(10))**

  Foreign key (likely) referencing a **Reservation** table.

  Links the payment to a specific reservation.

- **Payment_date (timestamp)**

  Stores the date and time when the payment was made.

  Uses a **timestamp** data type for automatic date/time tracking.

- **Invoice_Amount (float(30))**

  Represents the total invoice amount for the payment.

  Defined as a **float** to store large numbers with decimal precision.

- **Payment_Method (varchar(20))**

  Specifies the method used for payment (e.g., "Credit Card," "Cash," "PayPal").

  Stored as a **varchar(20)**, meaning it can hold up to 20 characters.

- **Payment_Status (varchar(20))**

  Indicates the status of the payment (e.g., "Pending," "Completed," "Failed").

  Stored as a **varchar(20)** to accommodate different status values.

STAFF TABLE:

Staff	
Staff_ID	**integer(10)**
Staff_FirstName	varchar(30)
Staff_LastName	varchar(30)
Staff_Phone	integer(10)
Staff_Email	varchar(50)
Staff_Address	varchar(256)
Staff_Role	varchar(30)
Staff_Dept	varchar(20)

Explanation:

This **Staff** table represents the structure of an employee database, storing details about staff members working in an organization. On the next page is a detailed explanation of each column:

- **Staff_ID (integer(10), Primary Key 🔑)**

  Definition: A unique identifier for each staff member.

  Data Type: integer(10), meaning it stores numerical values up to 10 digits.

  Primary Key: Ensures each staff member has a unique ID.

  Auto-increment: Usually, this field is set to **auto-increment**, so each new employee gets a sequentially generated ID.

- **Staff_FirstName (varchar(30))**

  Definition: The first name of the staff member.

  Data Type: varchar(30), meaning it can store up to 30 characters.

  Example Values: "John", "Alice", "David"

- **Staff_LastName (varchar(30))**

  Definition: The last name (surname) of the staff member.

  Data Type: varchar(30), meaning it can store up to 30 characters.

  Example Values: "Doe", "Smith", "Johnson"

- **Staff_Phone (integer(10))**

  Definition: The contact phone number of the staff member.

  Data Type: integer(10), meaning it can store a 10-digit phone number.

  Example Values: 9876543210, 1234567890

  Possible Issue: Phone numbers should ideally be stored as a VARCHAR(15) to support country codes (e.g., +1-9876543210).

- **Staff_Email (varchar(50))**

  Definition: The email address of the staff member.

Data Type: varchar(50), allowing up to 50 characters.

Example Values:

- "john.doe@example.com"
- "alice.smith@company.com"

- **Purpose**: Used for communication, login credentials, and notifications.
- **Staff_Address (varchar(256))**

  Definition: The physical address of the staff member.

  Data Type: varchar(256), meaning it can store up to 256 characters.

  Example Values:

  - "123 Main Street, New York, NY"
  - "45B Baker Street, London"

  Purpose: Useful for employee records, payroll processing, and official documentation.

- **Staff_Role (varchar(30))**

  Definition: The role or designation of the staff member in the organization.

  Data Type: varchar(30), allowing up to 30 characters.

  Example Values:

  - "Manager"
  - "Receptionist"
  - "IT Technician"

  Purpose: Helps define job responsibilities and authority levels.

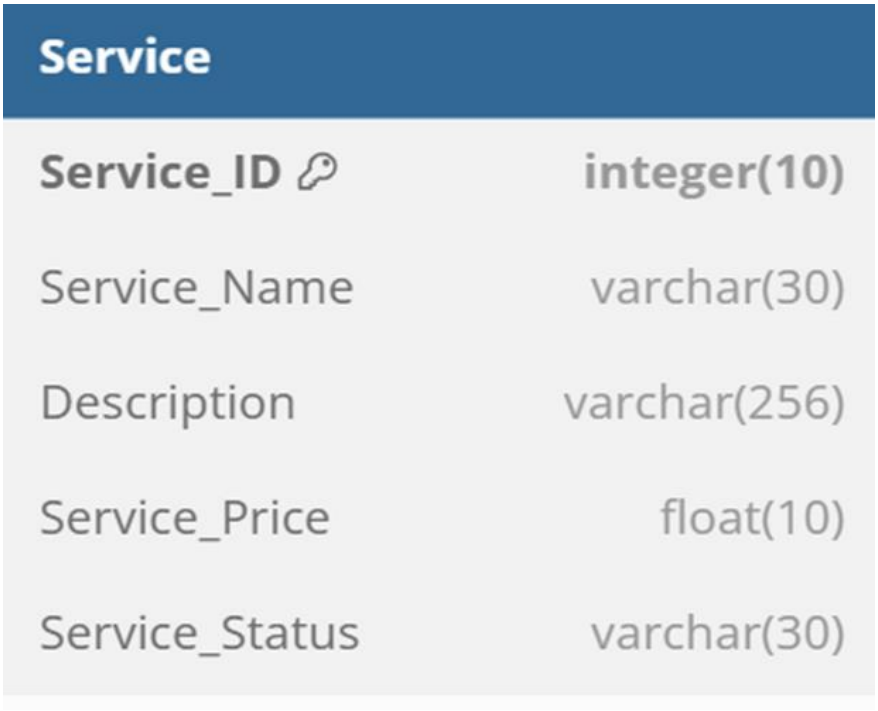
- **Staff_Dept (varchar(20))**

  Definition: The department where the staff member works.

  Data Type: varchar(20), allowing up to 20 characters.

  Example Values:"HR"

SERVICE TABLE:



Explanation:

The **Service** table keeps track of different services that a company provides, including their names, descriptions, pricing, and current status.

- **Service_ID (integer(10), Primary Key 🔑)**

  Definition: A unique identifier for each service.

  Data Type: integer(10), meaning it stores numerical values up to 10 digits.

  Primary Key: Ensures that each service has a unique ID.

  Auto-increment: Typically set to **auto-increment**, so each new service gets a sequentially generated ID.

- **Service_Name (varchar(30))**

  Definition: The name of the service.

  Data Type: varchar(30), meaning it can store up to 30 characters.

  Example Values:

  - "Haircut"
  - "Car Wash"
  - "Software Installation"

- **Description (varchar(256))**

  Definition: A brief description of the service.

  Data Type: varchar(256), meaning it can store up to 256 characters.

  Example Values:

  - "Professional haircut and styling service"
  - "Complete interior and exterior car wash"

  Purpose: Helps users understand what the service includes.

- **Service_Price (float(10))**

  Definition: The price of the service.

  Data Type: float(10), allowing decimal values.

  Example Values:

  - 29.99 (for a haircut)
  - 49.50 (for a car wash)

  Purpose: Stores the cost of each service.

- **Service_Status (varchar(30))**

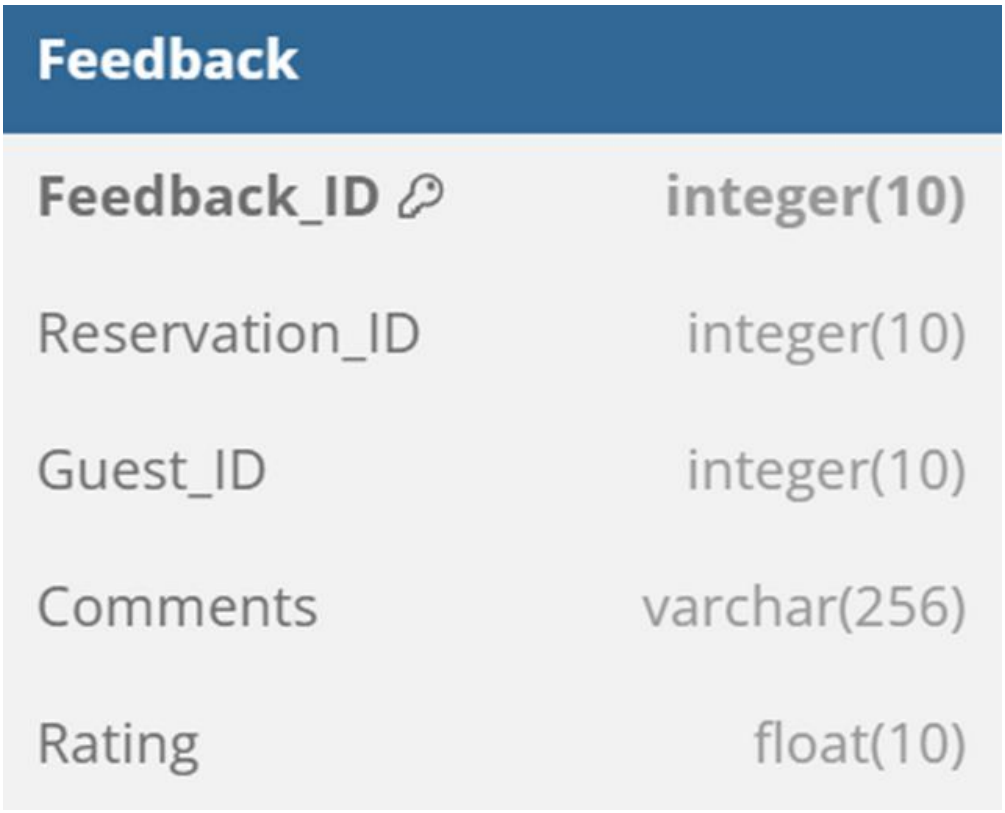
  Definition: The current status of the service.

  Data Type: varchar(30), allowing up to 30 characters.

  Example Values:

  - "Available"
  - "Temporarily Unavailable"

FEEDBACK TABLE:



Explanation:

The **Feedback** table helps collect guest reviews, ratings, and comments related to their reservations. This information is useful for businesses to assess customer satisfaction and improve services.

- **Feedback_ID (integer(10), Primary Key 🔑)**

  Definition: A unique identifier for each feedback entry.

  Data Type: integer(10), meaning it stores numerical values up to 10 digits.

  Primary Key: Ensures that each feedback entry has a unique ID.

Auto-increment: Likely set to **auto-increment**, meaning each new feedback entry gets a sequentially generated ID.

- **Reservation_ID (integer(10))**

  Definition: Links feedback to a specific reservation.

  Data Type: integer(10), meaning it stores numerical values up to 10 digits.

  Foreign Key: Likely references a **Reservation** table to track which reservation this feedback is related to.

  Purpose: Ensures feedback is connected to a specific customer reservation.

- **Guest_ID (integer(10))**

  Definition: Identifies the guest who provided the feedback.

  Data Type: integer(10), meaning it stores numerical values up to 10 digits.

  Foreign Key: Likely references a **Guest** or **Customer** table.

  Purpose: Helps track which guest submitted the review.

- **Comments (varchar(256))**

  Definition: Stores customer feedback in text format.

  Data Type: varchar(256), allowing up to 256 characters.

  Example Values:

  - "Great service! The room was clean and comfortable."
  - "The food was excellent, but the waiting time was too long."

  Purpose: Captures qualitative feedback from guests.

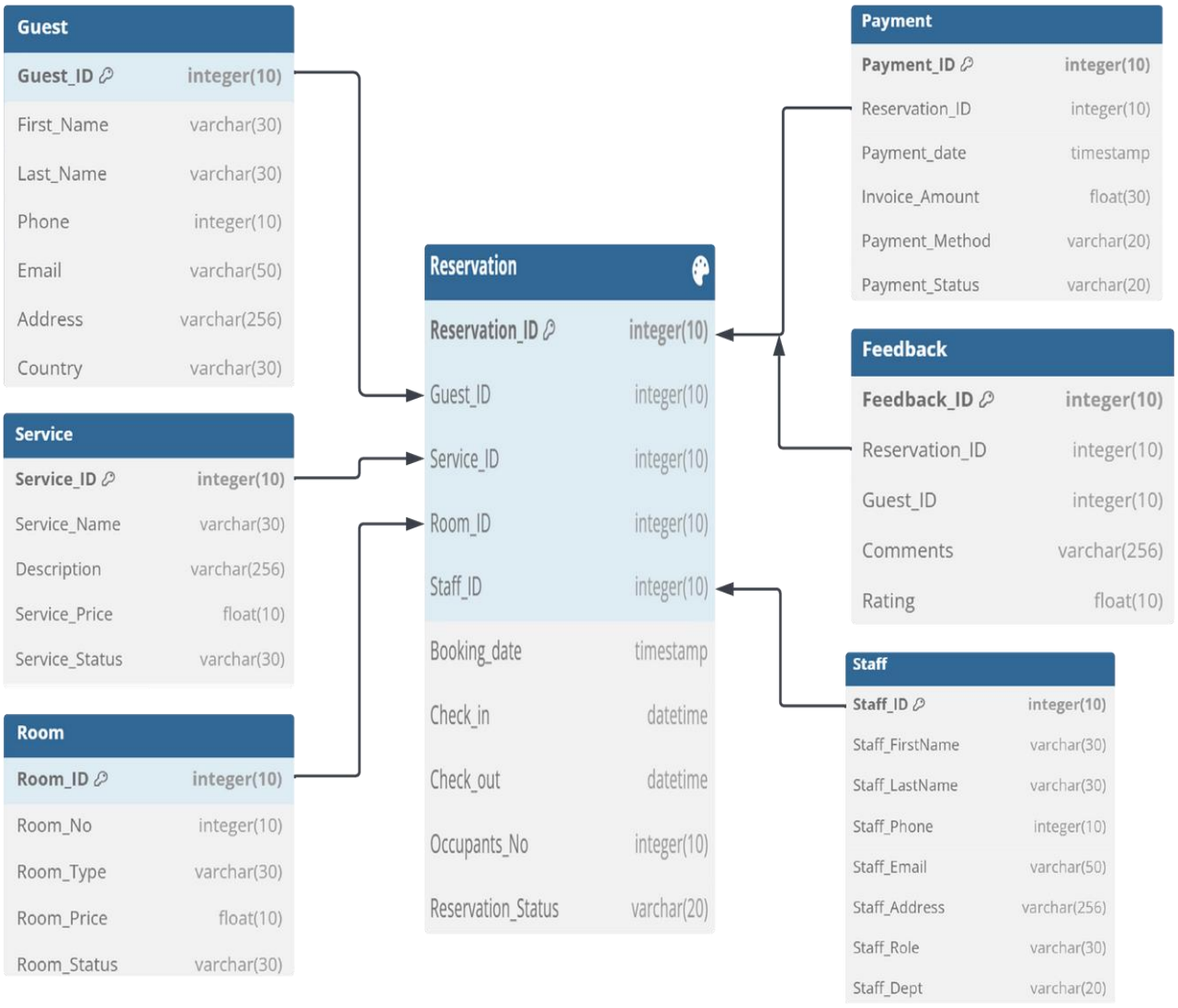
- **Rating (float(10))**

Definition: A numerical rating provided by the guest.

Data Type: float(10), meaning it can store decimal values.

Example Values: 4.5 (out of 5)

ENTITY RELATIONSHIP DIAGRAM:



This **Entity-Relationship Diagram (ERD)** represents a **hotel reservation management system**, detailing the relationships between different entities involved in the booking and management process.

- The **Guest** table stores guest details, including their unique **Guest_ID**, name, phone number, email, address, and country. Each guest can make one or more **reservations**, which are recorded in the **Reservation** table. This table links guests to their booked **rooms** and **services**, assigning a **Reservation_ID** to each booking. It includes booking details like **booking date, check-in/check-out dates, number of occupants, and reservation status**. Each reservation is managed by a staff member (linked through **Staff_ID**).
- The **Room** table stores room details, including **Room_ID, Room_No, Room_Type, Room_Price, and Room_Status**. The **Service** table records additional services available to guests, such as spa, dining, or laundry, identified by **Service_ID**, with attributes including **Service_Name, Description, Price, and Status**.
- Payments for reservations are stored in the **Payment** table, which records each **Payment_ID**, the associated **Reservation_ID**, the **Payment Date**, **Invoice Amount**, **Payment Method**, and **Payment Status**.
- Customer feedback is stored in the **Feedback** table, where each **Feedback_ID** links back to a specific **Reservation_ID** and **Guest_ID**. It contains customer **comments and ratings**, providing insights into guest satisfaction.
- Lastly, the **Staff** table maintains records of hotel employees, each identified by a **Staff_ID**. It includes details like **first name, last name, phone number, email, address, role, and department**, helping manage staff responsibilities.

This database structure ensures efficient hotel operations, tracking guest stays, payments, services, and feedback while maintaining organized staff and room records.

RELATIONSHIP:

Guests & Reservations:

The One-to-Many relationship between Guests and Reservations means that:

- A single guest can make multiple reservations over time. For example, a guest may book a room multiple times on different dates.
- Each reservation is linked to only one guest, ensuring that every booking is associated with a specific individual.

Foreign Key Implementation:

- The GuestID in the Reservations table acts as a foreign key referencing the GuestID in the Guests table.
- This enforces referential integrity, meaning a reservation cannot exist without a valid guest record.

Rooms and Reservations:

The One-to-Many relationship between Rooms and Reservations means that:

- A single room can be booked multiple times by different guests on different dates. For example, Room 101 can be reserved by multiple guests at different times.
- Each reservation is for only one specific room, ensuring that a booking is linked to a particular room at a given time.

Foreign Key Implementation:

- The RoomID in the Reservations table acts as a foreign key referencing the RoomID in the Rooms table.
- This maintains referential integrity, ensuring that a reservation is always associated with an existing room.

Reservations & Payments

The One-to-Many relationship between Reservations and Payments means that:

- A single reservation can have multiple payments associated with it. For example, a guest may pay in installments or make separate payments for additional services.
- Each payment is linked to only one reservation, ensuring that every transaction corresponds to a specific booking.

Foreign Key Implementation:

- The ReservationID in the Payments table serves as a foreign key, referencing the ReservationID in the Reservations table.
- This ensures referential integrity, meaning every payment must be associated with an existing reservation.

Reservations & Feedback

The One-to-One relationship between Reservations and Feedback means that:

- Each reservation can have only one feedback entry provided by the guest after their stay.

- Each feedback is linked to a specific reservation, ensuring that reviews and ratings correspond to the correct booking.

Foreign Key Implementation:

- The ReservationID in the Feedback table acts as a foreign key, referencing the ReservationID in the Reservations table.
- This maintains referential integrity, ensuring that feedback exists only for valid reservations.

Guests & Feedback

The One-to-Many relationship between Guests and Feedback means that:

- A single guest can leave multiple feedback entries for different reservations (stays).
- Each feedback entry is linked to only one guest, ensuring that reviews are associated with the correct individual.

Foreign Key Implementation:

- The GuestID in the Feedback table serves as a foreign key, referencing the GuestID in the Guests table.
- This ensures that every feedback entry is tied to a valid guest, maintaining data integrity and allowing businesses to track guest experiences over multiple visits.

CONCLUSION

The **Hotel Booking System** enhances efficiency, sustainability, and customer satisfaction in the hospitality industry. It streamlines reservations, ensures secure transactions, and personalizes guest experiences. By promoting eco-friendly practices and improving operational efficiency, the system helps hotels attract more customers and stay competitive