Hotel Room Booking System

Table of Contents

1. How It Was Implemented

The development of the Hotel Room Booking System began with an in-depth understanding of the functionality of real-world hotel management systems. After identifying the essential requirements, a relational database was designed and implemented using MySQL.

The process began with the creation of a database named HotelBookingDB. Key tables were created, including Guests, RoomTypes, Rooms, and Bookings. Each table featured a primary key, and relationships were established using foreign keys to maintain referential integrity.

Important design features included:

- Enforcing unique email addresses for guests
- Assigning unique room numbers and associating each room with a specific room type
- Ensuring that each booking record has a check-in date earlier than the check-out date, and preventing overlapping bookings for the same room

To improve query performance, indexes were added to frequently used columns such as room_type_id, booking_status, and check_in/check_out.

Sample data was inserted to test the database functionality:

- Three room types (Single, Double, Suite)
- Three guest records
- Three uniquely numbered rooms
- Bookings using both static and dynamic dates (via the CURDATE() function)

To enhance usability and system intelligence, several SQL views were created:

- **AvailableRooms**: Lists rooms not booked in the next 30 days
- **GuestBookingSummary**: Summarizes total bookings and nights per guest
- **RoomStatus**: Displays whether each room is currently "Occupied" or "Available"

Additional SQL logic was implemented to:

- Confirm pending bookings only when no conflicts exist
- Cancel future bookings for a specific room
- Display a list of guests scheduled for check-in on the current day
- Count total bookings for each day in the current month
- Identify guests who have never made a booking

The system was thoroughly tested and performed efficiently, meeting all outlined requirements, including prevention of double bookings, quick lookups, and clear data summarization.

2. Problem Statement

The Hotel Room Booking System is designed to automate and streamline the reservation process. It facilitates the registration of guests, manages room types, assigns rooms without conflicts, and handles bookings with accurate check-in and check-out dates.

Key goals of the system include:

- Preventing double bookings and managing date overlaps
- Providing visibility into room statuses
- Maintaining data integrity and consistency across all components

The system's functionality includes:

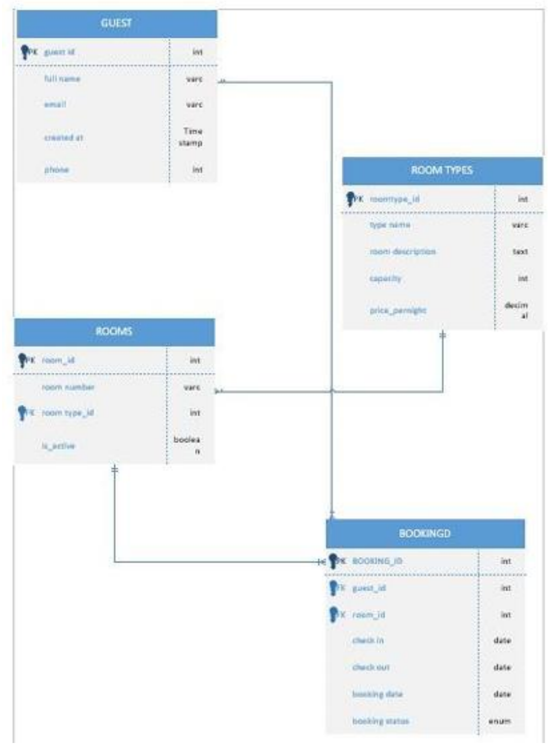
- Separate tables for guests, room types, rooms, and bookings
- Enforced constraints to allow only one active booking per room at a time
- Room types including pricing and capacity
- Validation to ensure check-in precedes check-out
- Summary views for efficient data interpretation

- Use of indexes for performance optimization

This system serves as a practical example of database design in real-world hotel management and demonstrates principles of normalization, indexing, and constraint enforcement.

---

3. Entity Relationship Diagram (ERD)



The ERD illustrates relationships between the core entities: Guests, RoomTypes, Rooms, and Bookings. It outlines how each entity connects and identifies primary and foreign key constraints for maintaining relational consistency.

4. Database Schema with Table Descriptions

Guests Table

Stores guest details including:

- guest_id (Primary Key)
- full_name, email (Unique), phone, and created_at timestamp

RoomTypes Table

Defines room categories with the following attributes:

- room_type_id (Primary Key)
- type_name (Unique), description, capacity, and price_per_night

Rooms Table
Contains room-specific data:

- room_id (Primary Key)
- room_number (Unique), linked room_type_id, and room status (active/inactive)

Bookings Table
Records reservations:

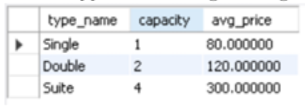
- booking_id (Primary Key)
- Foreign keys to guest_id, room_id, room_type_id
- check_in, check_out, booking_status, and created_at timestamp
- Constraint: check_in must be earlier than check_out

Indexes and Constraints

- Primary keys: guest_id, room_type_id, room_id, booking_id
- Foreign keys:
    - guest_id → Guests
    - room_type_id → RoomTypes
    - room_id → Rooms
- Unique columns:
    - email (Guests), room_number (Rooms), type_name (RoomTypes)
- Indexes:
    - idx_room_type (Rooms)
    - idx_checkin_checkout, idx_booking_status, idx_bookings_guest_id, idx_bookings_room_id (Bookings)

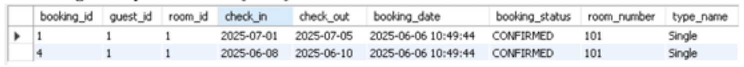
5. Screenshots of Queries and Outputs

- Room Types with Average Pricing



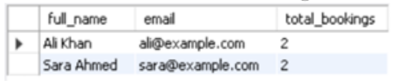
type_name	capacity	avg_price
Single	1	80.000000
Double	2	120.000000
Suite	4	300.000000

- Bookings for Specific Guest (ID: 1)



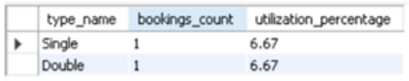
booking_id	guest_id	room_id	check_in	check_out	booking_date	booking_status	room_number	type_name
1	1	1	2025-07-01	2025-07-05	2025-06-06 10:49:44	CONFIRMED	101	Single
4	1	1	2025-06-08	2025-06-10	2025-06-06 10:49:44	CONFIRMED	101	Single

- Guests with More Than One Booking



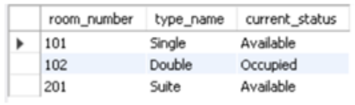
full_name	email	total_bookings
Ali Khan	ali@example.com	2
Sara Ahmed	sara@example.com	2

- Room Utilization (%)



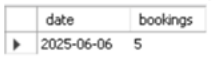
type_name	bookings_count	utilization_percentage
Single	1	6.67
Double	1	6.67

- Room Current Status View



room_number	type_name	current_status
101	Single	Available
102	Double	Occupied
201	Suite	Available

- Guests with Overlapping Bookings (Date: 2025-07-04)



date	bookings
2025-06-06	5

6. Conclusion and Challenges Faced

This project provided practical experience in designing, implementing, and testing a relational database. It enhanced understanding of entity relationships, referential integrity, and query optimization.

Key challenges included:

- **Overlapping Bookings**: Preventing multiple reservations for the same room at the same time required precise SQL logic using NOT EXISTS and date comparisons
- **Data Validity**: Using fixed test data sometimes led to empty query results; this was addressed by implementing dynamic functions like CURDATE()
- **Performance Optimization**: As the dataset grew, indexing became essential to maintain performance in queries involving date and room filters
- **Views and Summarization**: Creating SQL views helped simplify complex query outputs, making results more interpretable

Overall, the system fulfilled all requirements and offered a real-world perspective on database applications in the hospitality industry. The implementation not only met academic goals but also helped build skills relevant to professional database development.