COMP3030: Group Project Report. "VinKun: Database Design for Hotel Booking Management"

Students:
Vu Hong Phuc
Tran Quang Thanh
Huynh Quynh Anh

Supervisor
Prof. Nguyen Hoang Ha

1 Introduction

Project Overview VinKun, a renowned luxury hotel chain, requires a booking management system to streamline and enhance operational efficiency. The system should cover various aspects of hotel booking management, including guest reservations, room and service booking, and overall customer experience. The goal is to provide a seamless and delightful experience for guests while optimizing internal for hotel staff.

2 Business Requirements

2.1 Room Booking

Name of Use Case: Room Booking

Goal: Enables guests to book a hotel room through an online booking system.

Actor(s): Guest

Trigger: Guest decides to book a hotel room.

Entry Conditions: None

Post Conditions: Room is successfully booked, and booking details are sent to the guest.

Basic Flow of Events:

1. Guest accesses the hotel booking system.
2. The system presents a list of hotels, including **name**, **address**, **city**, **country**, **phone**, **email**, **rating**, and **facilities**. Guest can filter by **city**, **country**, **rating**, and **facilities**. Guest chooses their preferred hotel.
3. Guest inputs their **check-in** and **check-out dates**.
4. Based on the entered dates, the system shows a list of available rooms, including **room type (name and description)**, **amenities**, **capacity**, **available rooms** and **price**. Guests can sort and filter by **price** and **amenities**.
5. Guest selects preferred room types and the number of rooms for each type.
6. Guest fills out personal information including **first name**, **last name**, **date of birth** (optional), **address** (optional), **email**, and **phone number**.

7. The system provides a summary of the booking for review, including **start and end dates**, **hotel name**, **room type**, **number of corresponding rooms**, **total price**, and **booking deposit**.
8. Guest selects a **payment method** and completes the payment process.
9. Upon successful booking, the system sends a confirmation email to the guest with all booking details including **booking id**, **hotel name**, **check-in time**, **check-out time**, **room type**, **number of corresponding rooms**, **total price**, **deposit**, and **QR code** for additional services. The system should mark the **room type**, **number of corresponding rooms** as reserved during **check-in time**, **check-out time**.
10. Guest submits service requests by scanning QR in their confirmation email (more on this in section 2.2).

Alternate Flows:

3. Guest modifies the selected dates or hotel leading to an updated list of available rooms.
7. Guest returns to modify room selection or personal information before finalizing the booking.

Exceptions:

6. Incomplete or invalid personal information submitted.
8. The payment process fails or is declined.

Data Constraints:

- **Hotel Information:** Each hotel must have a valid name, address, city, country, rating, phone number, email address, and a list of facilities (optional but recommended for better guest experience). The hotel's name, email, and phone number should be unique.
- **Date Constraints:** The check-in date must be on or after the current date, and strictly before the check-out date.
- **Room Specifications:** Each room type should have a clear name, detailed description, defined capacity (minimum of 1 person), a set price (> 0) per night, and an optional list of amenities.
- **Booking Volume:** The number of rooms booked should be at least one and must not exceed the total number of available rooms of the selected type.
- **Room Availability:** Room availability is determined based on the total number of rooms of the specified type at the hotel, minus the number of rooms already booked. This calculation must be updated in real time to reflect current availability.
- **Room Status:** Each room should have a status categorized as "Available", "Maintenance", or "Occupied". Rooms labeled as "Maintenance" cannot be booked.
- **Guest Information:** Mandatory information includes first name, last name, email, and phone number. Address and date of birth are optional but recommended for more personalized services.
- **Payment Constraints:** The booking deposit should be a fixed amount (≥ 0) as defined by each hotel or the final price of the booking, whichever is smaller. This ensures that the deposit reflects the hotel's policy while also considering the total cost of the booking.
- **Payment Methods:** Guests must choose from the hotel's accepted payment methods. The amount of payment should be greater than 0. The system should validate payment information for accuracy and authenticity.

- **Price Calculation:** The price of the booking must match the number of rooms, room types, and duration of stay.

2.2 Service Ordering

Name of Use Case: Service Ordering

Goal: Enables guests to order services through a digital menu accessible via QR code before and during their stay.

Actor(s): Guest, Staff

Trigger: Guest decides to order a service using the digital service menu.

Entry Conditions: Guest must have a valid booking id to access the service menu. Guest must be able to scan the QR code sent to their email.

Post Conditions: Service order is successfully placed and confirmed by guest and staff.

Basic Flow of Events:

1. Guest scans the QR code containing their **booking id** to access the service menu.
2. The system presents a digital service menu, including service items with **name**, **description**, and **price** (> 0).
3. Guest selects a service item and specifies any **service unit** and **service notes**. The system saves service requests and returns **service booking id**.
4. The system notifies hotel staff of the guest's request, including guest details which are **booking id**, **guest name**, **phone**, **email**, **room number**, **service name**, **service unit**, and **service notes**. The system uses **service booking id** to retrieve these data.
5. Staff reviews the request, confirms it, and updates the **service name** and **service unit**, **service price**, **service status** corresponding to **service booking id** to the system.

Alternate Flows:

5. Staff suggests modifications or alternatives to the service requested based on availability or other constraints.
6. Guest cancels the requested service using **service booking id**.

Exceptions:

1. QR code does not work or fails to load the service menu.

Data Constraints:

- **Booking ID Validation:** The guest must provide a valid booking ID to access the service menu. The system should verify the booking ID against current and future reservations.
- **Service Menu:** The digital service menu must list each service with a clear name, detailed description, and price. All listed services must be currently available at the hotel.

- **Service Selection:** Guests should be able to select services, specify service units (at least 1), and add service notes. Service notes can be at most 1023 characters long.
- **Service Status Tracking:** The system must allow for updating and tracking the status of each service order, with statuses including "Pending", "Confirmed", and "Completed", "Canceled".
- **Order Confirmation:** The staff must confirm the service order in the system, updating the service status and cost as per the selected service units. When the staff confirms the service order, its status should be "Confirmed".
- **Order Time:** The system must store the creation time and the last updated time of each service request.
- **Order Completion:** When the service is delivered, change the status to "Completed". Only "Completed" service should be charged.

2.3 Check-in

Name of Use Case: Guest Check-In and Room(s) Assigning

Goal: Facilitates the process of verifying guest reservation and assigning rooms upon arrival.

Actor(s): Guest, Staff

Trigger: Guest arrives at the hotel and initiates the check-in process.

Entry Conditions: Guest must have a valid reservation. Staff must have access to the hotel management system.

Post Conditions: Guest is successfully checked in and assigned room(s).

Basic Flow of Events:

1. Guest arrives and provides identification along with reservation confirmation to hotel staff.
2. Staff verifies the guest's information in the system by **name**, **email**, **phone**, **check-in date** or **booking id**.
3. Upon successful verification, the system displays the booking details including **booking id**, **guest details (name, email, date of birth, phone, address)**, **check-in date**, **check-out date**, **room (type, number of rooms)**, **service (name, note, status, created date)**.
4. Staff uses the system to retrieve a list of available rooms (**room number**), applying filters by **room type**, **hotel id**, and **current status**. Rooms assigned to guest must be currently available.
5. Staff assigns room(s) to the guest based on booked preferences and availability. System should marked the rooms as occupied.

Alternate Flows:

2. If verification fails, staff rechecks the guest's provided information and attempts verification.

Exceptions:

2. Verification failure due to incorrect or missing information.

Data Constraints:

- **Room Assignment:** Rooms assigned to guests must match their booked preferences in terms of room type and number of rooms. The system should mark assigned rooms as occupied and update the room's status accordingly.
- **Status of Booking:** Status of booking should be changed from "Confirmed" to "CheckedIn".

2.4 Check-out

Name of Use Case: Guest Check-Out and Payment

Goal: Manages the guest's departure process, including verification, billing, and payment.

Actor(s): Guest, Staff

Trigger: Guest approaches the reception to check out.

Entry Conditions: Guest must have completed their stay. Staff must have access to the hotel management system.

Post Conditions: Guest has checked out, payment is processed, and the stay is marked as complete.

Basic Flow of Events:

1. Guest provides identification to hotel staff for system verification during check-out.
2. Staff verifies the **booking id** and the system retrieves the stay details including **guest details (name, email, date of birth, phone, address)**, **check-in date**, **check-out date**, **room (type, number of rooms, price)**, **service (name, unit, note, price, created date)**.
3. The system calculates the final bill including **total price**, **room(s) number**, **duration of stay**, **services name**, **services unit**, **date of service ordering**.
4. Guest reviews the final bill on the system and chooses a **payment method**.
5. After confirming payment and checking the room's condition, staff confirms the guest's stay as complete in the system.
6. The system sends a thank-you email and a feedback form to the guest. The feedback form includes a **star rating** and **review**.

Alternate Flows:

4. Guest disputes charges on the final bill, and staff reviews and adjusts the bill if necessary.

Exceptions:

4. Payment method is declined or payment confirmation fails.
5. Discrepancies in room condition require further action before finalizing check-out.

Data Constraints:

- **Status of Booking:** Status of booking should be changed from "CheckedIn" to "CheckedOut".
- **Payment Method:** The system must validate that the payment method used by the guest is one of the accepted methods offered by the hotel.
- **Feedback:** Any feedback system integrated into the booking process must limit the star rating to a scale of 1 to 5. Additionally, written reviews should be capped at 1023 characters.

2.5 Manager View

Name of Use Case: Manager View

Goal: Enables the hotel manager to review financial reports and customer feedback within a specified time range.

Actor(s): Manager

Trigger: Manager decides to review hotel revenue and customer feedback.

Entry Conditions: Manager must be logged into the system with managerial access rights.

Post Conditions: Manager has received reports on hotel revenue and customer feedback.

Basic Flow of Events:

1. Manager accesses the system.
2. Manager inputs a time range with a **start date** and **end date** for the revenue report.
3. The system calculates and presents the a list of **booking id**, **payment id**, **payment method**, **amount**, **payment time**, and **total revenue** from bookings and services within the specified time range.
4. Manager inputs a time range with a **start date** and **end date** for customer feedback.
5. The system retrieves and presents a list of **reviews, ratings, review time**, and calculates the **average rating** from guests within the specified time range.
6. Manager reviews the data and uses it for business analysis and decision-making.

Alternate Flows:

2. Manager adjusts the time range if the system does not return any data for the initially entered dates.
5. Manager may request more detailed feedback reports based on specific criteria like room types or services.

Exceptions:

2. Manager inputs an invalid date range, and the system prompts to correct the input.

Data Constraints: N/A

3 Database Design

3.1 Data Model - ERD

The provided Entity-Relationship Diagram (ERD) represents the database schema for a hotel booking system. This ERD outlines the intricate relationships between various entities such as Hotel, Guest, Room, Service, Booking, Feedback, Facility, Payment,... The diagram employs a clear structure, displaying entities as rectangles and relationships as diamonds. The cardinality of each relationship is indicated by the notation near the connecting lines, which shows one-to-many (1:N), many-to-one (N:1), or many-to-many (N:M) correspondences.

Each entity has attributes with keys that denote either primary or foreign keys, which are crucial for relational database design. For instance, the Hotel entity is characterized by attributes like HotelID, Name, Address,... with HotelID being the primary key. The Booking entity at the center of the diagram connects multiple entities and represents the core functionality of the system, handling the association between Guests, Rooms, Services,...

The ERD demonstrates normalization through the separation of concerns, such as distinct entities for RoomType and RoomStatus, which simplifies the maintenance of room details and statuses. Payment methods are also abstracted out into a separate entity, which provides flexibility for future extensions.

To fully comprehend the database schema, a careful examination of this ERD is imperative, as it is a blueprint for the creation of the database and the development of the application logic that will interact with it. Below is the visual representation of the ERD:

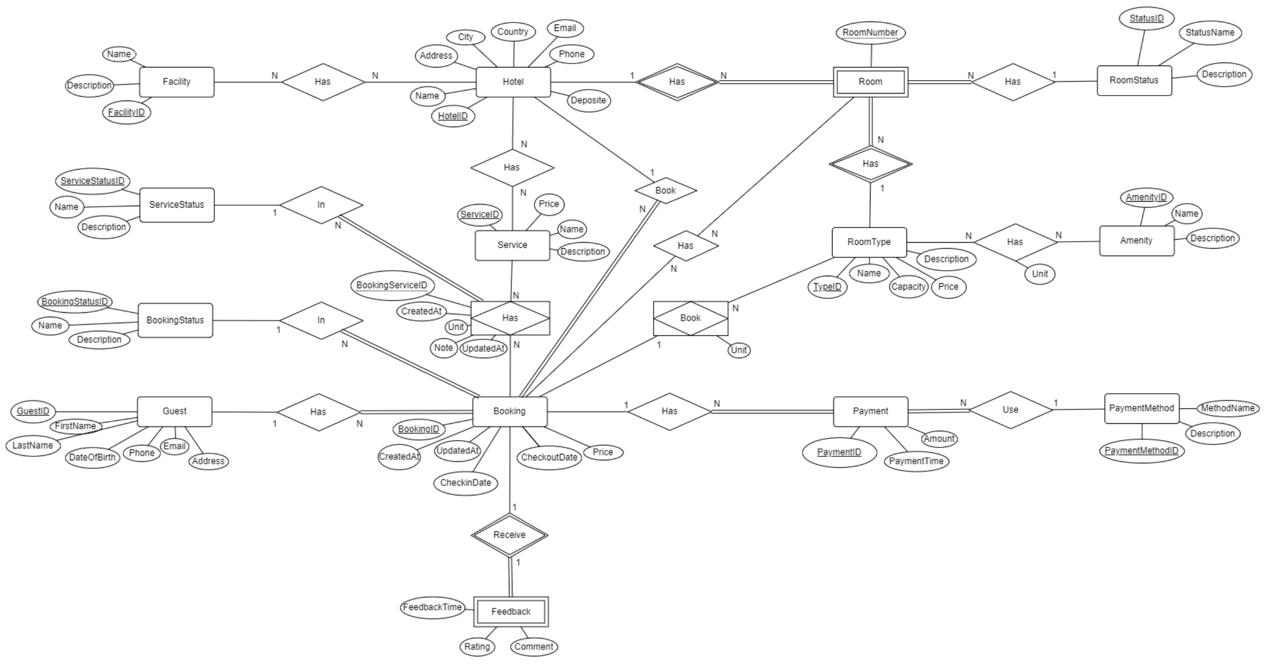


Figure 1: Entity-Relationship Diagram of the Hotel Booking System

3.2 Database Schema

The database schema presented here is derived from an Entity-Relationship Diagram (ERD), which is a conceptual model of the data structure required for a hotel management system. The schema delineates the logical configuration of tables and the relationships between them, essential for the implementation of the database.

This schema includes a variety of tables such as ‘Hotel‘, ‘Guest‘, ‘Room‘, ‘Booking‘, ‘Payment‘, and ‘Feedback‘, among others. Each table is meticulously designed to contain attributes that are crucial for the hotel’s operations. For instance, the ‘Guest‘ table contains personal details of the clients, while the ‘Booking‘ table records the specifics of their stay.

The relationships between tables are indicated by foreign keys that establish connections essential for database integrity and query efficiency. For example, the ‘Booking‘ table references the ‘Guest‘ table to associate each booking with the correct guest, and the ‘Payment‘ table to ensure accurate financial transactions.

The ERD from which this schema is derived provides a visual representation of the data structure, aiding in understanding the database’s organization. The schema is a more technical rendition, signifying the transition from conceptual design to practical database creation.

Please refer to the figure 2 below for the graphical representation of the database schema:

4 SQL Implementation

This section only demonstrates highlighted procedures for use cases mentioned above. The complete SQL code is in SQL file.

4.1 Room Booking Use Case

4.1.1 List Available Room Types Procedure

To efficiently fulfill the use case of presenting guests with a list of available rooms based on entered dates, along with room type details, amenities, capacity, available rooms, and prices, a two-procedure approach is essential.

The first procedure is responsible for retrieving all room types and their associated number of rooms within a hotel. This procedure gathers the essential information such as room type names, descriptions, amenities, capacities, and prices, providing guests with a comprehensive view of the available options.

The second procedure complements the first by determining the number of rooms of each type that are currently unavailable within the specified date range. This information is crucial in ensuring that guests are presented with accurate availability details. By subtracting the unavailable rooms from the total number of rooms of each type, the system can accurately display the number of available rooms, enabling guests to make informed decisions.

Together, these two procedures work in tandem to deliver a seamless and informative room selection experience for guests, allowing them to filter and sort by price and amenities while ensuring real-time availability data.

```sql
-- Procedure to Show Room Types with its number of rooms"

DELIMITER //

CREATE PROCEDURE ListAvailableRooms(
    IN hotel_id INT,
    IN filter_amenity VARCHAR(100),
    IN sort_by_price BOOLEAN
)
BEGIN
    SELECT
        rt.RoomTypeID,
        rt.Name,
        rt.Description,
        rt.Price,
        rt.Capacity,
        GROUP_CONCAT(DISTINCT a.Name SEPARATOR ', ') AS Amenities,
        COUNT(DISTINCT r.RoomID) as TotalRooms
    FROM
        RoomType rt
    LEFT JOIN
        RoomTypeAmenity rta ON rt.RoomTypeID = rta.RoomTypeID
    LEFT JOIN
        Amenity a ON rta.AmenityID = a.AmenityID
  LEFT JOIN
    Room r on r.RoomTypeID = rt.RoomTypeID
  WHERE
    r.HotelID = hotel_id          AND (
            filter_amenity IS NULL
            OR rt.RoomTypeID IN (
                SELECT DISTINCT rt2.RoomTypeID
                FROM RoomType rt2
                LEFT JOIN RoomTypeAmenity rta2 ON rt2.RoomTypeID = rta2.RoomTypeID
                LEFT JOIN Amenity a2 ON rta2.AmenityID = a2.AmenityID
                WHERE a2.Name = filter_amenity
            )
        )
  GROUP BY rt.RoomTypeID

  ORDER BY
    CASE WHEN sort_by_price THEN rt.Price END ASC,
    CASE WHEN sort_by_price  = 0 THEN rt.Price END DESC;
END //

DELIMITER ;
```

Listing 1: Show Room Types with its number of rooms

```sql
-- Procedure to get non available number of room for each room type"

DELIMITER //

CREATE PROCEDURE GetNotAvailableNumberOfRooms(
    IN hotel_id INT,
    IN checkin_date Date,
    IN checkout_date Date
)
BEGIN
    SELECT
  rt.RoomTypeID,
    rt.Name,
    rt.Description,
    rt.Capacity,
    temp.TotalUnits as TotalBookedRooms
FROM RoomType rt
LEFT JOIN (
    SELECT brt.RoomTypeID, SUM(brt.Unit) as TotalUnits
    FROM Booking b
    LEFT JOIN BookingRoomType brt on brt.BookingID = b.BookingID
    WHERE b.CheckinDate >= checkin_date AND b.CheckoutDate <= checkout_date AND b.HotelID =
    hotel_id
    GROUP BY brt.RoomTypeID
) temp on rt.RoomTypeID = temp.RoomTypeID;
END //

DELIMITER ;
```

Listing 2: Get non available number of room for each room type

Procedure demonstration

- Guest wants all rooms in hotel with id = 3 **CALL ListAvailableRooms(3, NULL, False)** (Figure 3)
- Guest wants all rooms with mini bar in hotel with id = 3 and sorted by price **CALL ListAvailable-Rooms(3, 'Mini Bar', True)** (Figure 4)

4.2 Manager View Use Case: Get Average Rating Procedure

4.2.1 Procedure Implementation

The manager view use case requires a mechanism to compute the average rating within a given time range. This is achieved through the SQL procedure `GetAverageRating`. The procedure takes two dates as input, representing the start and end of the period, and outputs the average rating. It ensures that the start date is not after the end date and calculates the average rating based on the feedback received within this range. The SQL code is as follows:

```sql
CREATE PROCEDURE GetAverageRating(IN start_date DATE, IN end_date DATE, OUT average_rating
    DECIMAL(3,2))
BEGIN
    IF start_date <= end_date THEN
        SELECT AVG(f.Rating) INTO average_rating
        FROM Feedback f
        WHERE f.FeedbackTime BETWEEN start_date AND end_date;
    ELSE
        SELECT 'Error: start_date should be less than or equal to end_date.' AS ErrorMessage
    ;
    END IF;
END //

DELIMITER ;
```

Listing 3: Get Average Rating SQL Procedure

4.2.2 Procedure Testing

To validate the functionality of the `GetAverageRating` procedure, a test is conducted. This test involves calling the procedure with a specific start and end date and then retrieving the average rating calculated. The SQL command for this test is provided below:

```sql
-- Retrieve and display the average rating calculated by the previous procedure
SELECT @average AS AverageRating;
```

Listing 4: Test Call to GetAverageRating

The procedure executes successfully, the output of this execution is illustrated in the figure 5.

4.3 Booking Service Use Case: Save Service Request Procedure

4.3.1 Procedure Implementation

In the context of the booking service application, a crucial functionality is implemented through the SQL stored procedure `SaveServiceRequest`. The primary role of this procedure is to process and record service requests made by users. Upon successful execution, it returns a unique service booking ID, which is essential for tracking and managing these requests. The implementation details of this procedure are outlined below.

```sql

-- SQL procedure to saves service requests and returns service booking id

```

```sql
DELIMITER //
DROP PROCEDURE IF EXISTS SaveServiceRequest;

CREATE PROCEDURE SaveServiceRequest(
    IN booking_id INT,
    IN selected_service_id INT,
    IN service_units INT,
    IN service_notes VARCHAR(1023),
    OUT booking_service_id INT
)
BEGIN
    DECLARE service_status_ordered INT DEFAULT 1; -- Assuming 'Ordered' status ID
    DECLARE current_price DECIMAL(10,2);

    -- Retrieve the price of the selected service
    SELECT Price INTO current_price FROM Service WHERE ServiceID = selected_service_id;

    -- Insert the new service request into BookingService table
    INSERT INTO BookingService (BookingID, ServiceID, ServiceStatusID, Unit, Note)
    VALUES (booking_id, selected_service_id, service_status_ordered, service_units,
    service_notes);

    -- Get the last auto-incremented ID which is the service booking ID
    SET booking_service_id = LAST_INSERT_ID();
END //

DELIMITER ;
```

Listing 5: Save Service Request SQL Procedure

This procedure first drops any existing procedure of the same name for consistency. It then creates a new procedure accepting parameters like booking ID, selected service ID, units of service, and service notes. Notably, it retrieves the current price of the selected service, inserts the new service request into the `BookingService` table, and finally, sets the output parameter `booking_service_id` to the last auto-incremented ID, which acts as the service booking ID.

4.3.2 Procedure Testing

After implementing the `SaveServiceRequest` procedure, it is essential to test its functionality. The procedure is tested by calling it with specific parameters to simulate a real service request. The test involves creating a service request for booking ID 5, service ID 5, requesting 2 units, and including notes for a special room setup. The call and its corresponding SQL command are as follows:

```sql
CALL SaveServiceRequest(3, 4, 1, 'Please do it. Thank you', @booking_service_id);
SELECT @booking_service_id AS BookingServiceID;
```

Listing 6: Test Call to SaveServiceRequest

The procedure executes successfully, and the generated service booking ID is stored in the variable `@booking_service_id`. The output of this execution is illustrated in the figure bellow:

4.4 Check-in Use Case

```sql
-- 8 --
-- SQL procedure to assigns a room to a booking and updates the room status to "Occupied"
DELIMITER //

DROP PROCEDURE IF EXISTS `AssignRoomToBooking`;

CREATE PROCEDURE AssignRoomToBooking(IN bookingID INT, IN roomID INT, OUT roomNumber VARCHAR
    (50))
```

```sql
BEGIN
    DECLARE availableStatusID INT DEFAULT 0;
    DECLARE occupiedStatusID INT DEFAULT 0;
    DECLARE currentStatusID INT DEFAULT 0;

    -- Ensuring unique status IDs for 'Available' and 'Occupied'
    IF (SELECT COUNT(*) FROM RoomStatus WHERE Name = 'Available') != 1 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Ambiguous status for Available';
    END IF;
    IF (SELECT COUNT(*) FROM RoomStatus WHERE Name = 'Occupied') != 1 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Ambiguous status for Occupied';
    END IF;

    --  -- Get the status IDs for 'Available' and 'Occupied'
    SELECT RoomStatusID INTO availableStatusID FROM RoomStatus WHERE Name = 'Available';
    SELECT RoomStatusID INTO occupiedStatusID FROM RoomStatus WHERE Name = 'Occupied';

    -- -- Check the current status of the room
    SELECT RoomStatusID INTO currentStatusID FROM Room WHERE Room.`RoomID` = roomID;

    -- Check if the room is available
    IF currentStatusID = availableStatusID THEN
        -- Assign room to booking
        INSERT INTO BookingRoom (BookingID, RoomID) VALUES (bookingID, roomID);

        -- -- Update room status to 'Occupied'
        UPDATE Room SET RoomStatusID = occupiedStatusID WHERE Room.`RoomID` = roomID;

        -- -- Retrieve the room number for the output
        SELECT Room.`RoomNumber` INTO roomNumber FROM Room WHERE Room.`RoomID` = roomID;
    ELSE
        -- Return an error if the room is not available
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: The selected room is not
    available';
    END IF;
END //

DELIMITER ;

-- Examples --

CALL GetDetailedBookingInfoByFirstName('Alice');


CALL GetDetailedBookingInfoByLastName('Johnson');


CALL GetDetailedBookingInfoByPhone('123-456-7890');


CALL GetDetailedBookingInfoByEmail('alice.johnson@example.com');


CALL GetDetailedBookingInfoByBookingID(1);

CALL GetDetailedBookingInfoByCheckinDate('2024-01-10');

CALL GetAvailableRooms(1, 1);

CALL AssignRoomToBooking(1, 10, @roomNumber);
SELECT @roomNumber AS RoomNumber;
```

Listing 7: assigns a room to a booking

4.5 Check-out Use Case

```sql
-- 1 --
-- SQL procedure to retrieve billing information given booking id

DELIMITER //

DROP PROCEDURE IF EXISTS `GetBillingInformation`;

CREATE PROCEDURE GetBillingInformation(IN p_BookingID INT)
BEGIN
    DECLARE completedStatusID INT DEFAULT 0;

    -- Fetch the ServiceStatusID for 'Completed' status
    SELECT ServiceStatusID INTO completedStatusID FROM ServiceStatus WHERE Name = 'Completed
    ' LIMIT 1;

    -- Ensure that there is only one 'Completed' status
    IF (SELECT COUNT(*) FROM ServiceStatus WHERE Name = 'Completed') != 1 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Ambiguous status for Completed';
    END IF;

    -- Retrieve billing information
    SELECT
        b.BookingID,
        b.Price AS BookingPrice,
        (b.Price + COALESCE((SELECT SUM(s.Price * bs.Unit)
                             FROM BookingService bs
                             JOIN Service s ON bs.ServiceID = s.ServiceID
                             WHERE bs.BookingID = p_BookingID AND bs.ServiceStatusID =
    completedStatusID), 0)) AS TotalPrice,
        (SELECT GROUP_CONCAT(CONCAT(r.RoomNumber, ' - ', rt.Name) SEPARATOR ', ')
         FROM BookingRoom br
         JOIN Room r ON br.RoomID = r.RoomID
         JOIN RoomType rt ON r.RoomTypeID = rt.RoomTypeID
         WHERE br.BookingID = b.BookingID) AS RoomsUsed,
        DATEDIFF(b.CheckoutDate, b.CheckinDate) AS DurationOfStay,
        (SELECT GROUP_CONCAT(CONCAT(s.Name, ' (', bs.Unit, ' units @ ', s.Price, '/unit) on
    ', DATE_FORMAT(bs.CreatedAt, '%Y-%m-%d')) SEPARATOR ', ')
         FROM BookingService bs
         JOIN Service s ON bs.ServiceID = s.ServiceID
         WHERE bs.BookingID = p_BookingID AND bs.ServiceStatusID = completedStatusID) AS
    ServicesUsed
        FROM Booking b
        WHERE b.BookingID = p_BookingID;
END //

DELIMITER ;
```

Listing 8: retrieve billing information

5 Team Contribution

Equal contribution between team members.

References

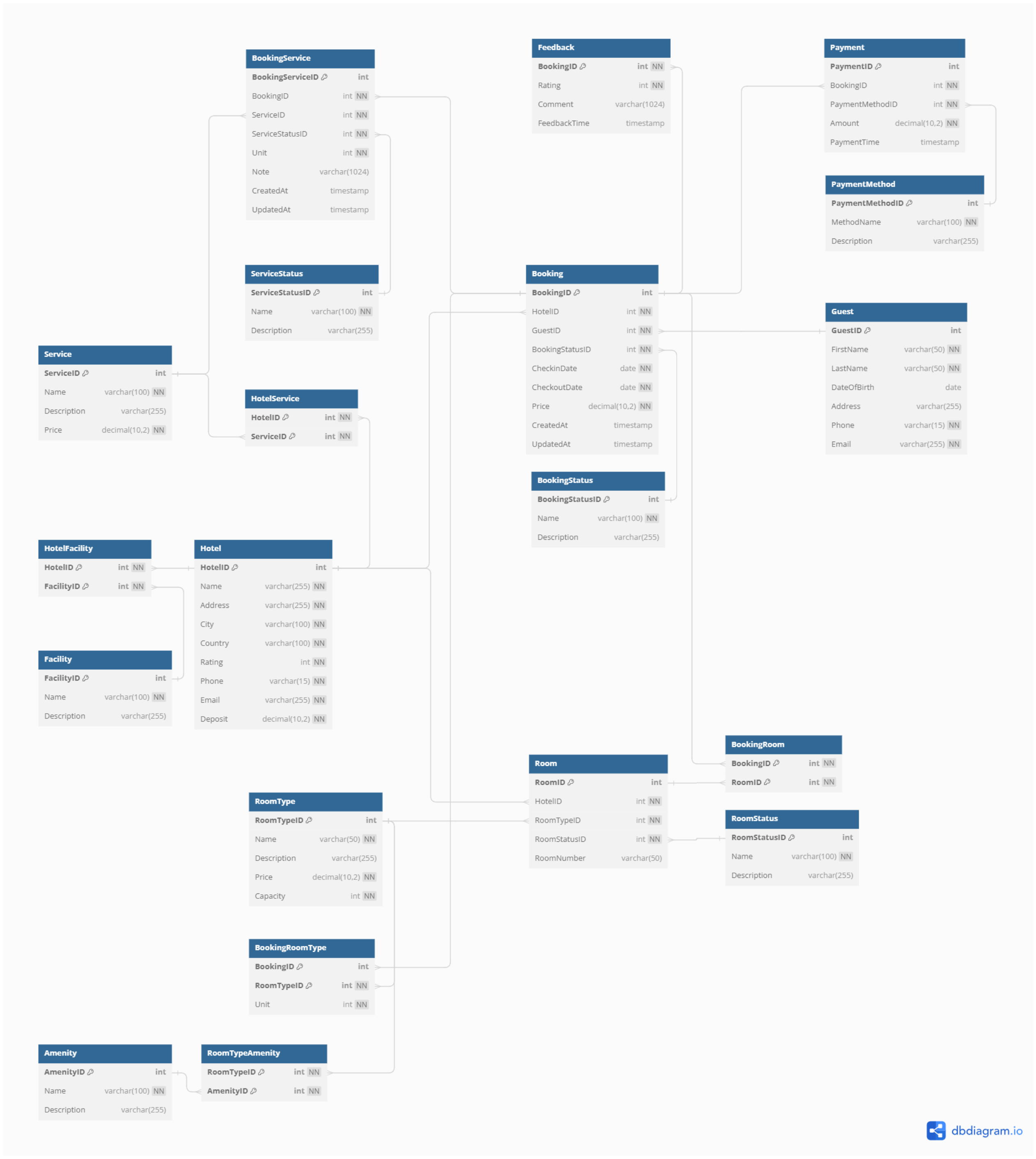
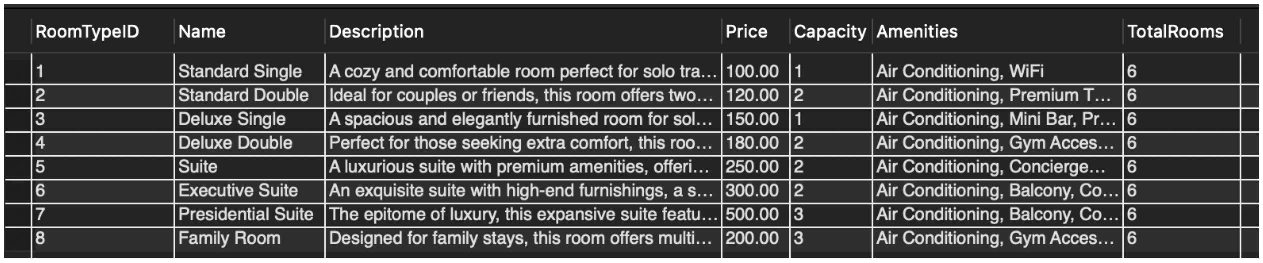
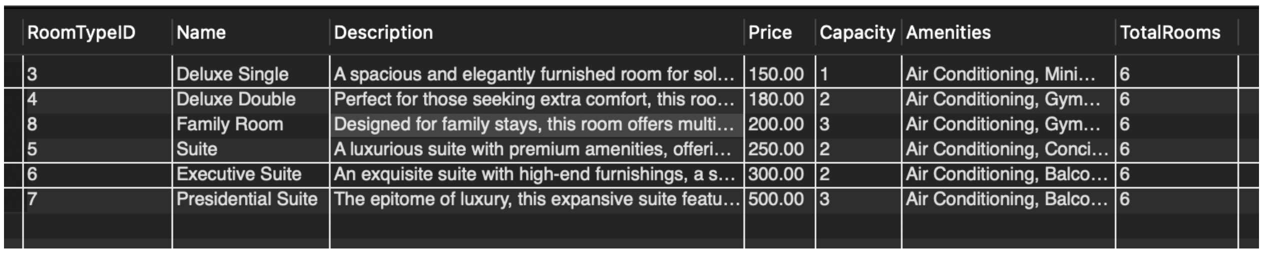


Figure 2: Database Schema



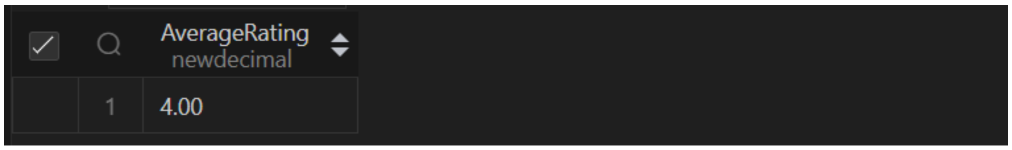
RoomTypeID	Name	Description	Price	Capacity	Amenities	TotalRooms
1	Standard Single	A cozy and comfortable room perfect for solo tra…	100.00	1	Air Conditioning, WiFi	6
2	Standard Double	Ideal for couples or friends, this room offers two…	120.00	2	Air Conditioning, Premium T…	6
3	Deluxe Single	A spacious and elegantly furnished room for sol…	150.00	1	Air Conditioning, Mini Bar, Pr…	6
4	Deluxe Double	Perfect for those seeking extra comfort, this roo…	180.00	2	Air Conditioning, Gym Acces…	6
5	Suite	A luxurious suite with premium amenities, offeri…	250.00	2	Air Conditioning, Concierge…	6
6	Executive Suite	An exquisite suite with high-end furnishings, a s…	300.00	2	Air Conditioning, Balcony, Co…	6
7	Presidential Suite	The epitome of luxury, this expansive suite featu…	500.00	3	Air Conditioning, Balcony, Co…	6
8	Family Room	Designed for family stays, this room offers multi…	200.00	3	Air Conditioning, Gym Acces…	6

Figure 3



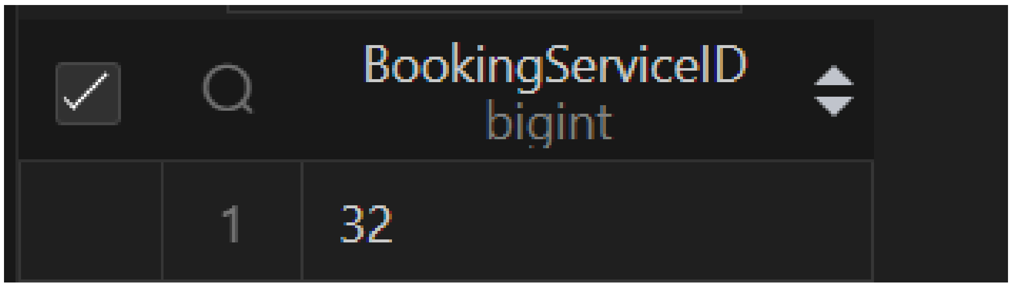
RoomTypeID	Name	Description	Price	Capacity	Amenities	TotalRooms
3	Deluxe Single	A spacious and elegantly furnished room for sol…	150.00	1	Air Conditioning, Mini…	6
4	Deluxe Double	Perfect for those seeking extra comfort, this roo…	180.00	2	Air Conditioning, Gym…	6
8	Family Room	Designed for family stays, this room offers multi…	200.00	3	Air Conditioning, Gym…	6
5	Suite	A luxurious suite with premium amenities, offeri…	250.00	2	Air Conditioning, Conci…	6
6	Executive Suite	An exquisite suite with high-end furnishings, a s…	300.00	2	Air Conditioning, Balco…	6
7	Presidential Suite	The epitome of luxury, this expansive suite featu…	500.00	3	Air Conditioning, Balco…	6

Figure 4



	AverageRating newdecimal
1	4.00

Figure 5: Successful execution of the GetAverageRating procedure



	BookingServiceID bigint
1	32

Figure 6: Successful execution of the SaveServiceRequest procedure