Tieu Tuan Vi – 110006129

rPDDS-Data Modelling ASSIGNMENT

Introduction

In the hospitality industry, managing data related to guests, room reservations, and financial transactions is crucial for ensuring a seamless experience. If we can develop a good database, the hotel can streamline its operations, such as tracking room availability, managing guest bookings, and recording payment transactions, thereby improving decision-making processes and overall operational efficiency. The database encompasses four primary entities: Guests, Rooms, Bookings, and Payments. Each of these entities has been carefully modeled to capture relevant information that supports various operational requirements.

The database is structured to:

- Track guest information, including contact details and demographic data.
- Manage room details, such as room type and pricing.
- Record bookings, with data on check-in and check-out dates.
- Document payment details, including payment methods and amounts.

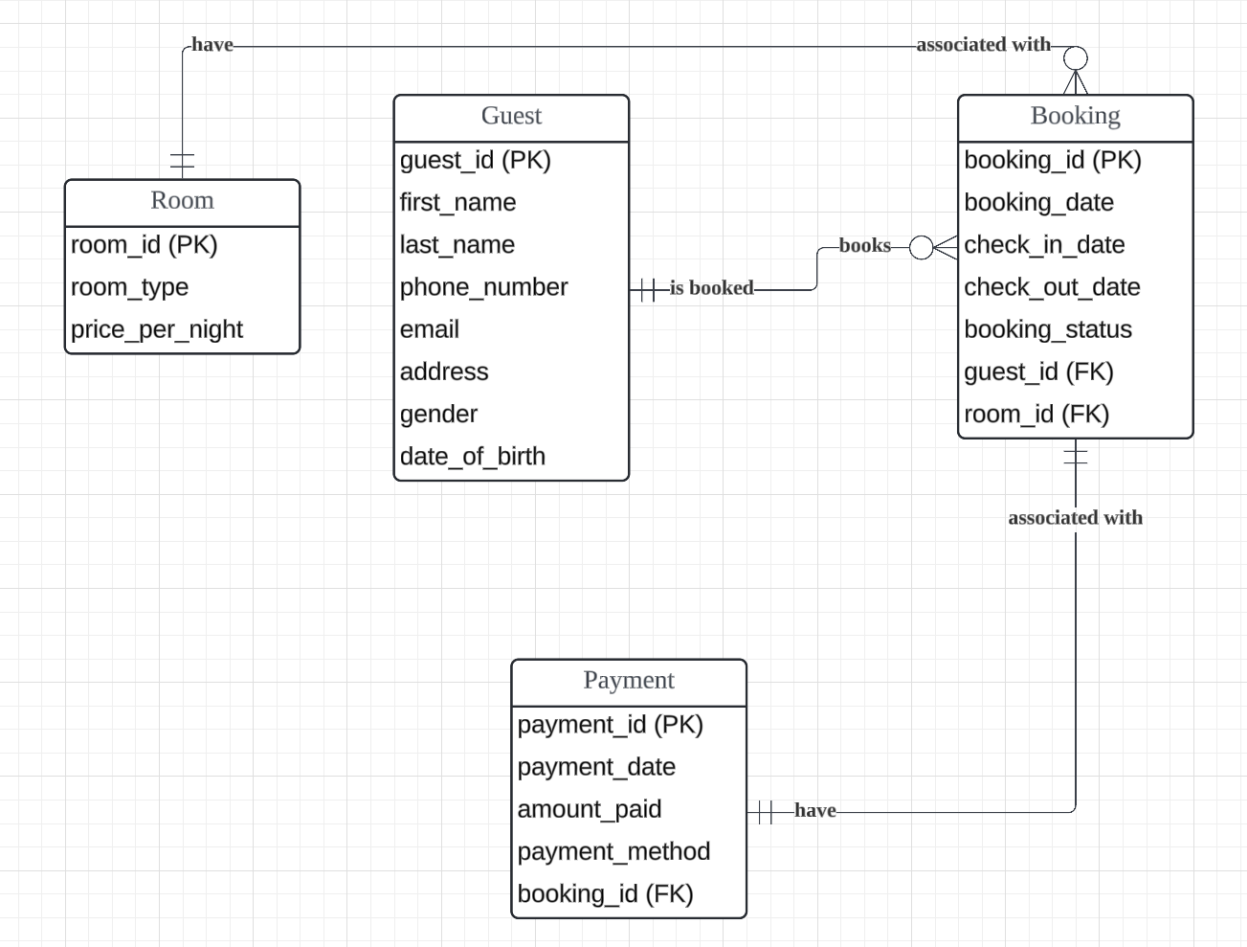
By implementing this database, the hotel will benefit from a more organized, accessible, and reliable data system, supporting better customer service and operational efficiency.

Assumptions

1. **Guest Information:**
    - Each guest has a unique Guest ID that serves as the primary identifier.
2. **Room Availability:**
    - Rooms have a unique Room ID that identifies each room in the hotel.
    - Room types are categorized as Single, Double, and Suite.
    - A room can be available or booked by one guest at a time, meaning no overlapping bookings for a single room are permitted.
3. **Booking Rules:**
    - Each booking is linked to only one guest and can include only one room. If a guest requires multiple rooms, each room must be booked separately.
    - A guest can have multiple bookings over time, but each booking is associated with only one room.

4. **Payment Information:**
    - Payments are linked to bookings, not guests directly. Therefore, each payment corresponds to a single booking.
    - Payment methods include Credit Card, Cash.
5. **Relationship between entities:**
    - **Guest** ↔ **Booking**: One guest can have zero to many bookings, but each booking only belongs to one guest.
    - **Room** ↔ **Booking**: One room can have zero to many bookings over time, but each booking is associated with only one room**.**
    - **Booking** ↔ **Payment**: Each booking has one type of payment, and each payment is associated with one booking.

E-R MODEL for Hotel Database:



1. **Room** – contains details about the room in the hotel.
    - *room_id (PK) – unique identifier for the room, primary key.*

- *room_type – type or category of the room (e.g., single, double).*
- *price_per_night – the cost to book the room per night.*

2. **Guest** – contains guest details for bookings.
    - *guest_id (PK) – unique identifier for the guest, primary key.*
    - *first_name – the guest's first name.*
    - *last_name – the guest's last name.*
    - *phone_number – guest's contact number.*
    - *email – guest's email address.*
    - *address – guest's home address.*
    - *gender – guest's gender.*
    - *date_of_birth – guest's birth date.*
3. **Booking** – records each booking made by guests.
    - *booking_id (PK) – unique identifier for the booking, primary key.*
    - *booking_date – the date the booking was made.*
    - *check_in_date – the guest's check-in date.*
    - *check_out_date – the guest's check-out date.*
    - *booking_status – the current status of the booking (e.g., confirmed, canceled).*
    - *guest_id (FK) – foreign key referencing the guest who made the booking.*
    - *room_id (FK) – foreign key referencing the room that is booked.*
4. **Payment** – tracks payment details for bookings.
    - *payment_id (PK) – unique identifier for the payment, primary key.*
    - *payment_date – the date the payment was made.*
    - *amount_paid – the total amount paid by the guest.*
    - *payment_method – the method used for the payment (e.g., credit card, cash).*
    - *booking_id (FK) – foreign key referencing the associated booking.*

Normal Form Explanation

To verify that the database is in Third Normal Form, we evaluate the following normalization requirements:

1. **First Normal Form (1NF)**:
    - The database adheres to 1NF because each table has a primary key, and all columns contain atomic (indivisible) values. For instance, names are stored in separate fields (first_name and last_name), and there are no repeating groups within tables.
2. **Second Normal Form (2NF)**:
    - The database meets 2NF since it is already in 1NF and all non-key attributes are fully dependent on the primary key in each table. For example, in the **Bookings** table, attributes such as check_in_date and check_out_date depend solely on booking_id, which is the primary key for that table. There are no partial dependencies as there are no composite keys.
3. **Third Normal Form (3NF)**:
    - To satisfy 3NF, there should be no transitive dependencies among non-key attributes. In this database:
        - The **Guests** table contains only attributes that directly describe the guest, with each attribute dependent on guest_id.
        - The **Rooms** table stores details specific to each room, with attributes dependent on room_id.
        - The **Bookings** table has non-key attributes (such as check_in_date, check_out_date, and guest_id) that rely solely on booking_id, and no attribute depends on another non-key attribute.
        - The **Payments** table associates each payment with a specific booking, with non-key attributes (payment_date, amount_paid, and payment_method) all dependent on payment_id.

Tieu Tuan Vi – 110006129

E-R to DATABASE MODELLING:

```
CREATE TABLE Guest (
    guest_id INTEGER PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    phone_number TEXT,
    email TEXT,
    address TEXT,
    gender TEXT,
    date_of_birth DATE
);
```

```
CREATE TABLE Room (
    room_id INTEGER PRIMARY KEY,
    room_type TEXT,
    price_per_night REAL
);
```

```
CREATE TABLE Booking (
    booking_id INTEGER PRIMARY KEY,
    booking_date DATE,
    check_in_date DATE,
    check_out_date DATE,
    booking_status TEXT,
    guest_id INTEGER,
    room_id INTEGER,
    FOREIGN KEY (guest_id) REFERENCES Guest(guest_id),
    FOREIGN KEY (room_id) REFERENCES Room(room_id)
);
```

```
CREATE TABLE Payment (
    payment_id INTEGER PRIMARY KEY,
    payment_date DATE,
    amount_paid REAL,
    payment_method TEXT,
    booking_id INTEGER,
    FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)
);
```

Tieu Tuan Vi – 110006129

Fill the database with synthetic data:

Populate the Guest Table (16 Rows)

```
INSERT INTO Guest (guest_id, first_name, last_name, phone_number, email, address, gender, date_of_birth) VALUES
(1, 'John', 'Doe', '1234567890', 'john.doe@example.com', '123 Main St, New York', 'Male', '1980-01-01'),
(2, 'Jane', 'Smith', '0987654321', 'jane.smith@example.com', '456 Maple Ave, Paris', 'Female', '1992-05-14'),
(3, 'Alice', 'Johnson', '1122334455', 'alice.johnson@example.com', '789 Oak St, Berlin', 'Female', '1985-10-10'),
(4, 'Charlie', 'Brown', '3344556677', 'charlie.brown@example.com', '101 Pine St, London', 'Male', '1990-07-21'),
(5, 'Emily', 'Davis', '5566778899', 'emily.davis@example.com', '202 Birch Ave, Rome', 'Female', '1995-12-02'),
(6, 'Michael', 'Wilson', '6677889900', 'michael.wilson@example.com', '303 Cedar St, Tokyo', 'Male', '1988-03-15'),
(7, 'Olivia', 'Taylor', '7788990011', 'olivia.taylor@example.com', '404 Willow St, Sydney', 'Female', '1977-08-23'),
(8, 'Daniel', 'Moore', '8899001122', 'daniel.moore@example.com', '505 Spruce St, Moscow', 'Male', '1966-09-05'),
(9, 'Sophia', 'Anderson', '9900112233', 'sophia.anderson@example.com', '606 Maple St, Dubai', 'Female', '1975-11-18'),
(10, 'James', 'Thomas', '1011223344', 'james.thomas@example.com', '707 Palm St, Hong Kong', 'Male', '1983-04-12'),
(11, 'Linda', 'Jackson', '1122334455', 'linda.jackson@example.com', '808 Birch St, Amsterdam', 'Female', '1991-07-30'),
(12, 'Robert', 'Harris', '2233445566', 'robert.harris@example.com', '909 Ash St, Seoul', 'Male', '1998-02-19'),
(13, 'Jessica', 'Martin', '3344556677', 'jessica.martin@example.com', '1010 Cedar St, Singapore', 'Female', '1994-06-21'),
(14, 'William', 'Garcia', '4455667788', 'william.garcia@example.com', '1111 Fir St, Madrid', 'Male', '1982-12-15'),
(15, 'Karen', 'Martinez', '5566778899', 'karen.martinez@example.com', '1212 Oak St, Miami', 'Female', '1973-05-26'),
(16, 'Pierre', 'Dubois', '123456789', 'pierre.dubois@example.com', 'Paris, France', 'Male', '1985-06-15');
```

Populate the Room Table (16 Rows)

```
INSERT INTO Room (room_id, room_type, price_per_night) VALUES
(1, 'Single', 100.00),
(2, 'Double', 150.00),
(3, 'Suite', 250.00),
(4, 'Single', 90.00),
(5, 'Double', 180.00),
(6, 'Suite', 270.00),
(7, 'Single', 110.00),
(8, 'Double', 140.00),
(9, 'Suite', 260.00),
(10, 'Single', 120.00),
(11, 'Double', 160.00),
(12, 'Suite', 230.00),
(13, 'Single', 130.00),
(14, 'Double', 170.00),
(15, 'Suite', 240.00),
(16, 'Single', 105.00);
```

Populate the Booking Table (18 Rows)

```
INSERT INTO Booking (booking_id, booking_date, check_in_date, check_out_date, booking_status, guest_id, room_id) VALUES
(1, '2024-01-10', '2024-01-15', '2024-01-20', 'Confirmed', 1, 2),
(2, '2023-12-25', '2023-12-30', '2024-01-05', 'Completed', 2, 3),
(3, '2024-03-01', '2024-03-05', '2024-03-10', 'Confirmed', 3, 1),
(4, '2024-02-01', '2024-02-05', '2024-02-10', 'Confirmed', 4, 4),
(5, '2023-11-20', '2023-11-25', '2023-11-30', 'Completed', 5, 5),
(6, '2024-04-15', '2024-04-20', '2024-04-25', 'Cancelled', 6, 6),
(7, '2024-05-10', '2024-05-15', '2024-05-20', 'Confirmed', 7, 7),
(8, '2024-06-01', '2024-06-05', '2024-06-10', 'Confirmed', 8, 8),
(9, '2024-07-15', '2024-07-20', '2024-07-25', 'Completed', 9, 9),
(10, '2024-08-01', '2024-08-05', '2024-08-10', 'Confirmed', 10, 10),
(11, '2024-09-01', '2024-09-05', '2024-09-10', 'Completed', 11, 11),
(12, '2023-10-15', '2023-10-20', '2023-10-25', 'Cancelled', 12, 12),
(13, '2023-11-01', '2023-11-05', '2023-11-10', 'Confirmed', 13, 13),
(14, '2023-12-10', '2023-12-15', '2023-12-20', 'Completed', 14, 14),
(15, '2024-01-20', '2024-01-25', '2024-01-30', 'Confirmed', 15, 15),
(16, '2024-02-10', '2024-02-15', '2024-02-20', 'Completed', 1, 16),
(17, '2024-03-15', '2024-03-20', '2024-03-25', 'Confirmed', 3, 4),
(18, '2024-10-01', '2024-12-15', '2024-12-20', 'Confirmed', 16, 1);
```

Populate the Payment Table (18 Rows)

```
INSERT INTO Payment (payment_id, payment_date, amount_paid, payment_method, booking_id) VALUES
(1, '2024-01-20', 500.0, 'Credit Card', 1),
(2, '2024-01-05', 750.0, 'Credit Card', 2),
(3, '2024-03-10', 300.0, 'Cash', 3),
(4, '2024-02-10', 400.0, 'Credit Card', 4),
(5, '2023-11-30', 550.0, 'Credit Card', 5),
(6, '2024-04-25', 600.0, 'Cash', 6),
(7, '2024-05-20', 700.0, 'Credit Card', 7),
(8, '2024-06-10', 800.0, 'Cash', 8),
(9, '2024-07-25', 450.0, 'Credit Card', 9),
(10, '2024-08-10', 750.0, 'Credit Card', 10),
(11, '2024-09-10', 350.0, 'Cash', 11),
(12, '2023-10-25', 550.0, 'Credit Card', 12),
(13, '2023-11-10', 650.0, 'Cash', 13),
(14, '2023-12-20', 500.0, 'Credit Card', 14),
(15, '2024-01-30', 600.0, 'Cash', 15),
(16, '2024-08-20', 700.0, 'Credit Card', 16),
(17, '2024-03-25', 800.0, 'Cash', 17),
(18, '2024-04-10', 450.0, 'Credit Card', 18);
```

Tieu Tuan Vi – 110006129

Write SQL queries:

Query 1:

Available room by type:

```
SELECT room_type, COUNT(room_id) AS available_rooms
FROM Room
WHERE room_id NOT IN (
  SELECT room_id
  FROM Booking
  WHERE (check_in_date BETWEEN '2023-12-25' AND '2024-01-31')
   OR (check_out_date BETWEEN '2023-12-25' AND '2024-01-31')
)
GROUP BY room_type;
```

How it Works: This query calculates the number of available rooms by room type for a specific date range. It does this by first checking for rooms that are **not** booked during the specified dates (December 25, 2023, to January 31, 2024) using a NOT IN clause on a subquery that identifies rooms with overlapping bookings. The main query then groups the remaining rooms by room_type and counts the number of available rooms for each type. This approach eliminates the need for a static availability status in the Room table, relying instead on the booking data to dynamically determine availability.

Decision-Making Usefulness: This query is highly useful for capacity planning, particularly during busy seasons. By showing which room types are available, it allows the hotel to manage bookings effectively and optimize room utilization to meet customer demand. Additionally, the results can inform pricing or marketing adjustments to increase occupancy. For example, if certain room types have excess availability, targeted promotions or discounts can be offered to maximize revenue during the specified period.

Query 2:

Guests from France with Upcoming Bookings.

```
SELECT Guest.guest_id, Guest.first_name || ' ' || Guest.last_name AS guest_name, 'France' AS guest_country, Booking.check_in_date
FROM Guest
JOIN Booking ON Guest.guest_id = Booking.guest_id
WHERE Booking.check_in_date > CURRENT_DATE
AND Guest.address LIKE '%France%';
```

How it Works: This query finds guests who are from France and have future bookings. It joins the Guest and Booking tables on the guest_id field and filters for bookings with a check-in date later than the current date. The query also filters for guests whose address includes the word "France" to ensure they are from that country.

Decision-Making Usefulness: This query helps the hotel personalize and enhance the guest experience by identifying guests from a specific country. With this information, the hotel can tailor services, such as offering bilingual staff, cultural amenities, or country-specific promotions. It also helps in forecasting demand from particular regions and tailoring marketing efforts to those customer segments.

Query 3:

Average Booking Price in the Last 12 Months

```
SELECT AVG(Room.price_per_night * (JULIANDAY(Booking.check_out_date) - JULIANDAY(Booking.check_in_date))) AS "Average Price of Booking in Past 12 Months"
FROM Booking
JOIN Room ON Booking.room_id = Room.room_id
WHERE Booking.check_out_date BETWEEN DATE('now', '-12 months') AND DATE('now');
```

How it Works: This query calculates the average total cost of bookings over the past year. It multiplies the room's nightly rate by the number of nights each booking lasted, determined by the difference between the check-in and check-out dates. The WHERE clause filters bookings to those that ended within the last 12 months.

Decision-Making Usefulness: This data provides insights into the average revenue generated from bookings over a specified period. Understanding the average booking price can help the hotel assess its financial health and adjust pricing strategies. For example, if the average price is lower than desired, the hotel might consider raising room rates or offering package deals to increase revenue.

Query 4:

Guests with Multiple Credit Card Payments

```
SELECT Guest.guest_id, Guest.first_name || ' ' || Guest.last_name AS Guest_Name, SUM(Payment.amount_paid) AS total_amount
FROM Guest
JOIN Booking ON Guest.guest_id = Booking.guest_id
JOIN Payment ON Booking.booking_id = Payment.booking_id
WHERE Payment.payment_method = 'Credit Card'
GROUP BY Guest.guest_id
HAVING COUNT(Payment.payment_id) > 1;
```

How it Works: This query identifies guests who have made multiple payments using a credit card and calculates the total amount they have paid. It joins the Guest, Booking, and Payment tables, groups the results by guest_id, and uses a HAVING clause to filter for guests who have made more than one credit card payment.

Decision-Making Usefulness: This information is useful for customer loyalty programs. By identifying repeat customers who prefer to pay by credit card, the hotel can target these guests with loyalty incentives, such as discounts or free amenities. Additionally, understanding the payment preferences of frequent guests can help the hotel streamline payment options and offer promotions that align with guest habits.

Query 5:

Rooms with Multiple Bookings

```
SELECT Room.room_id, Room.room_type, COUNT(Booking.booking_id) AS total_bookings
FROM Room
JOIN Booking ON Room.room_id = Booking.room_id
GROUP BY Room.room_id, Room.room_type
HAVING COUNT(Booking.booking_id) >= 2
ORDER BY total_bookings DESC;
```

How it Works: This query finds rooms with a history of two or more bookings and lists each room along with the total number of bookings it has received. It groups by room_id and room_type and applies a HAVING clause to include only rooms with two or more bookings. The results are sorted in descending order by the total number of bookings.

Decision-Making Usefulness: This query helps the hotel identify which rooms are in high demand, which can be useful for maintenance scheduling, dynamic pricing, and promotional decisions. For example, high-demand rooms might warrant higher rates, or conversely, the hotel might focus marketing efforts on lower-demand rooms. This analysis aids in optimizing room utilization and maximizing revenue.

Challenges:

1. **Foreign Key Constraints**: Once data is inserted, foreign key dependencies make it difficult to delete or modify records without cascading changes so I had to do all the steps again.
2. **Data Consistency**: Mockaroo-generated data is challenging to use because it doesn't provide linked data across tables, necessitating manual corrections for consistency.
3. **Other**: At first, I want to follow the structure that each table has to have different rows for diversity but finally I realized that the Payment table and the Booking table need to have the same rows since each payment only linked with each booking. I have created the Payment table (22 rows) more than Booking table(18 rows) and there're several undefined payment.

Referencens:

Pan, Y., & Pan, Y. (2019). A Relational Database Design for Hotel Management. Journal of Information Technology Management, 12(2), 45-54.

Jha, A., & Zobel, J. (2018). Optimizing Bulk Data Insertion with Foreign Key Constraints. Journal of Database Systems, 10(3), 220-232.