DBMS Project Report

Hotel reservation system:

The aim of the following project is to implement a simplistic overview of how a Hotel booking/reservation system works from the perspective of data storage and databases. The project works to implement real-world scenarios wherein a customer can check into one or more rooms and each check-in has a unique reservation ID. At the end of the stay, a bill is auto-generated by taking into account the duration of the stay, room category, number of beds, etc..

In this model, the existence of multiple hotels of the chain in different locations is taken into consideration, thus identifying each hotel by a unique value, HotelID. Each hotel has an 'n' number of rooms. and each room is uniquely identified by a RoomID.

With every customer that books a room, the following details about the customer is stored in the database:
CustomerID, SSN, Age, Full name, Address, primary contact number & email.
At the end of the stay a bill is generated for every customer, the attributes of which are : Billid,Total Amount paid, current date, full name of recipient, HotelID, CustomerID & reservationID.
Each bill is uniquely identified by a BillID.

A log of all the income to the Hotel chain is kept by maintaining an audit-trail which keeps track of the total income to the Hotel-chain. Also a track about the history of a customer is kept, whether the User has booked any rooms previously in any of the Hotels.

The following is the detailed report of a simplistic hotel reservation system.

Index

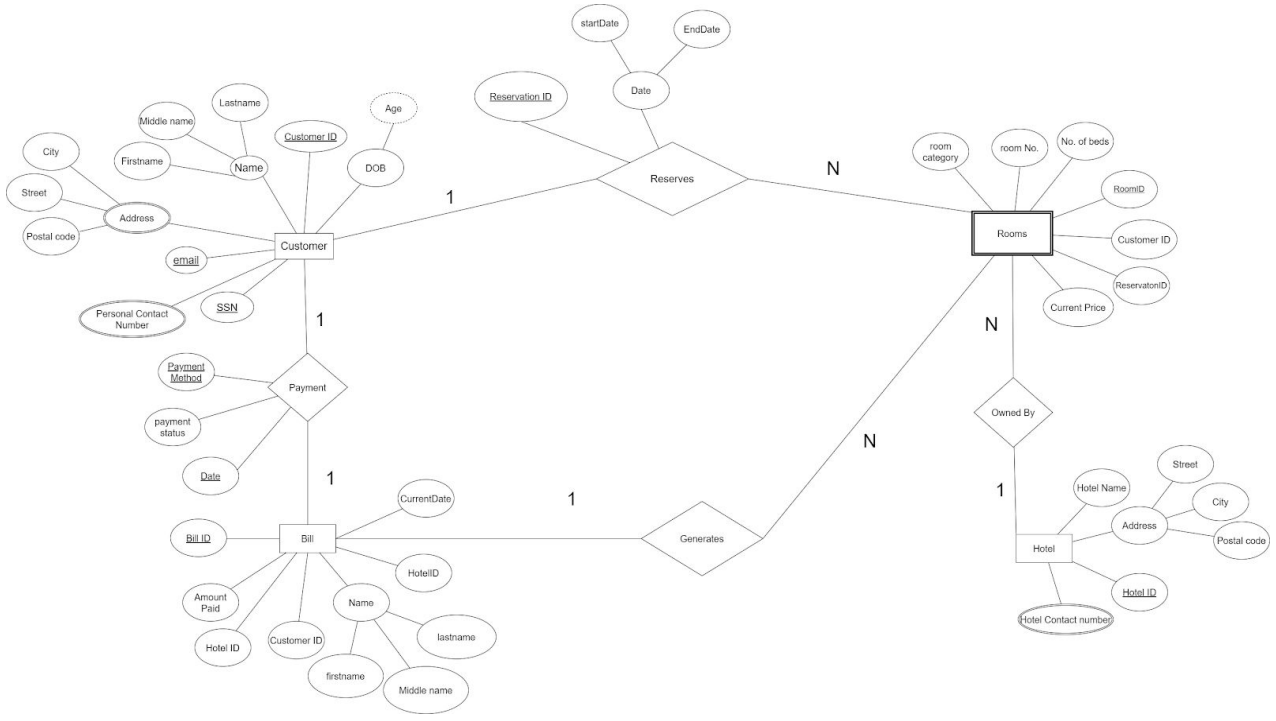
Introduction:

This project runs on any given MySql Client
I have used the Microsoft SQL Server Management Studio for the development of the project.
, a link containing code for the same has been attached at the end of the report.
The following are relation's in the database

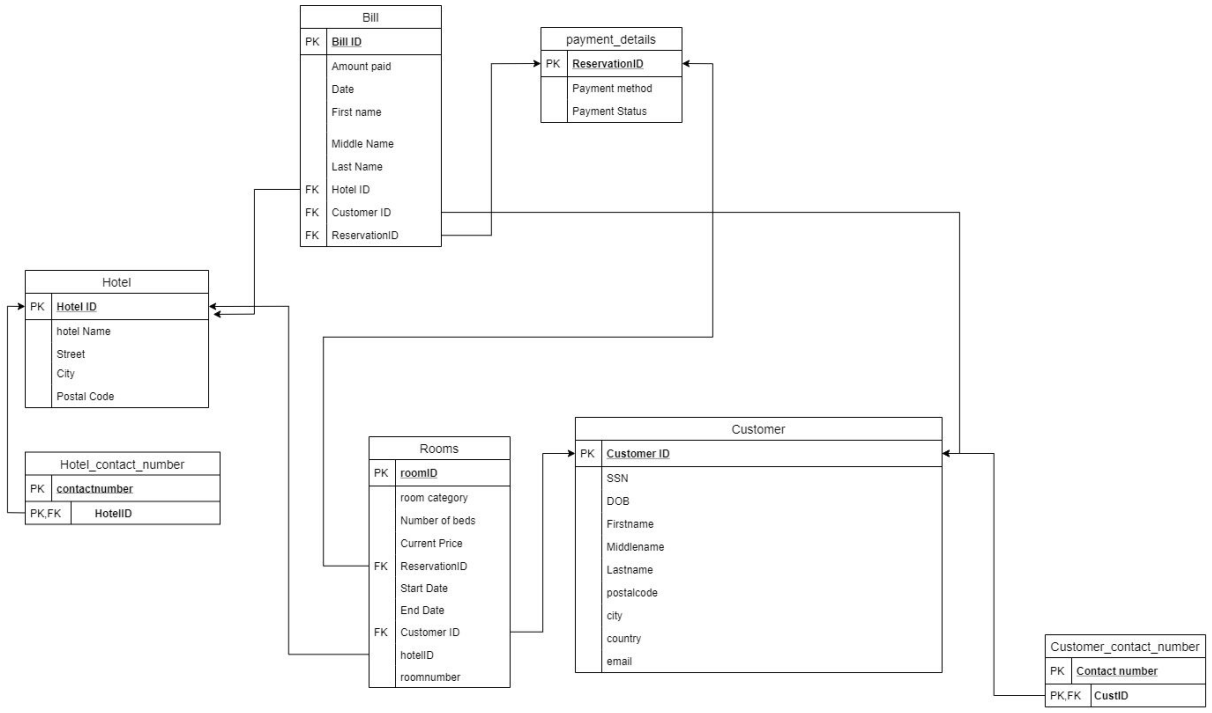
- Customer(SSN,custid,DOB,firstname,middlename,lastname,postalcode,city,country, primarycontact,email)
- Hotel(HotelID,hotelname,street,city,postalcode)
- Bill(billid,amountpaid,date,firstname,middlename,lastname,hotelID,CustomerID, reservationID)
- payment details(reservationid,payment method,payment status)
- Rooms(roomid,roomcategory,numberofbeds,startdate,enddate,currentprice,reservationID,CustomerID,roomnuber,hotelID)

Data Model

ER-Model:



Relational Schema:



The above relational schema represents 9 tables in total(including audit/log tables):

1.Customer:

- Attributes:
    - SSN(Unique),Datatype:INTEGER
    - CustomerID ,Datatype: INTEGER
    - Age, Datatype: INTEGER
    - first name, middle name, Last Name, Datatype: VARCHAR(10)
    - postal code,city,country,Datatype: VARCHAR(20)
    - email, Datatype: VARCHAR(20)
- Primary keys:
    - CustomerID
- Candidate keys:
    - CustomerID
    - email

Justification of obtained candidate keys:

The minimal cover is obtained from the functional dependencies.

we find attributes, not on RHS on FD. Every CK must contain these attributes the set of attributes that appeared on RHS of some FD but not on the LHS is{Age, first name, middle name, Last Name, postal code, city, country, primary contact}

These mentioned can not be in any candidate key.

with the other attributes, we check whether it is a candidate key by checking whether it has a proper subset which is also a superkey.
{CustomerID}: This is a superkey, and it is also a candidate key
{email}: This is a superkey, and it is also a candidate key

2.Customer contact number(multivalued attribute):

- Attributes:
  - Contact number, Datatype: VARCHAR(12)
  - CustomerID,Datatype: INTEGER
- Primary Keys:
  - Contact Number,CustomerID
- Foreign Keys:
  - CustomerID

3.Hotel:

- Attributes:
  - HotelID,Datatype:INTEGER(12)
  - hotel name, Datatype: VARCHAR(20)
  - street,city,postal code,Datatype: VARCHAR(20)
- Primary Keys:
  - HotelID
- Candidate Keys:
  - HotelID

Justification of obtained candidate keys:

The minimal cover is obtained from the functional dependencies.

set of attributes not on the RHS of any FD in the relation, which is Not On RHS is {Hotelid}. Every CK must contain these attributes.
HotelID is a superkey, so it is the only candidate key.

4.Hotel contact number(multivalued attribute):

- Attributes:
  - HotelID,Datatype: INTEGER
  - contact number, Datatype: VARCHAR(12)
- Primary Keys:
  - contact number
- Foreign Keys:
  - HotelID

5.Rooms:

- Attributes:
  - RoomID,Datatype: INTEGER
  - room category, Datatype: VARCHAR(10)
  - number of beds, Datatype: INTEGER
  - start date, end date, Datatype: DATE
  - current price, Datatype: INTEGER

    - ReservationID, Datatype: INTEGER
    - CustomerID, Datatype: INTEGER
    - room number, Datatype: INTEGER
    - HotelID, Datatype: INTEGER
- Primary Keys:
    - RoomID
- Foreign Keys:
    - HotelID
    - ReservationID
    - CustomerID
- Candidate Keys:
    - roomid
    - reservationID
    - {hotelID,room number}

Justification of obtained candidate keys:
The minimal cover is obtained from the functional dependencies.
We find the set of attributes not on RHS of any FD.every candidate key must contain these attributes we call this set set_1={}.
any attribute that appeared on the RHS of some FD but not on the LHS cannot be a candidate key, thus {roomcategory, number of beds, start date, end date,currentprice,CustID} cannot be candidate keys
we find closure set of set_1, (set_1)+ ={}
checking the set_1 UNION {roomid}: this is a superkey, and it is also a candidate key
checking the set_1 UNION {reservationID}: this is a superkey, and it is also a candidate key
checking the set_1 UNION {roomnuber}: this is not a superkey, and it is also not a candidate key
checking the set_1 UNION {HotelID}: this is not a superkey, and it is also not a candidate key
checking { hotelID,roomnuber}: This set is a superkey, and no proper subset of it is a superkey, hence it is a candidate key.

6.Bill:

- Attributes:
    - billid, Datatype: INTEGER
    - amount paid, Datatype: INTEGER
    - current date, Datatype: DATE
    - first name, middle name, last name, Datatype: VARCHAR(10)
    - hotelID, Datatype: INTEGER
    - CustomerID, Datatype: INTEGER
    - reservationID, Datatype: INTEGER
- Primary Keys:
    - BillID
- Foreign Keys:
    - HotelID
    - ReservationID

    - CustomerID
- Candidate Keys:
    - BillID
    - CustomerID

Justification of obtained candidate keys:
The minimal cover is obtained from the functional dependencies.
We find the set of attributes not on the RHS of any FD. Every candidate key must contain these attributes. The following attributes occur on RHS but not on LHS of minimal cover thus they cannot be in any candidate key.
{amount paid,date,middle name,first name,last name,hotel,reservationID}

{billID}: This is a superkey, and it is also a candidate key.
{CustomerID }: This is a superkey, and it is also a candidate key

7.payment Details:

- Attributes:
    - ReservationID, Datatype: INTEGER
    - payment method, payment status, Datatype: VARCHAR(10)
- Primary Keys:
    - ReservationID
- Candidate Keys:
    - ReservationID

justification of candidate keys:
The minimal cover is obtained from the functional dependencies.
ReservationID is a superkey, so it is the only candidate key.

8.Audit details:

- Attributes:
    - BillID, Datatype: INTEGER
    - start date, end date, Datatype: DATE
    - days of stay, Datatype: INTEGER
    - total amount paid, Datatype: INTEGER
- Primary Keys:
    - BillID

9.Rooms_log:

- Attributes:
    - index_1:Auto increment integer
    - CustomerID:INTEGER
    - RoomID:INTEGER
    - start_date,end_date:INTEGER
- Primary Keys:
    - index_1

FD and Normalization

Non-Trivial Functional Dependencies **(List based on your application constraints):**

- **Customer:**
    - Custid->{SSN,DOB,firstname,middlename,lastname,postalcode, city,country,primary contact,email}

- **Hotel**:
    - Hotelid->{hotel name,street,city,postal code}

- **Bill:**
    - billid->{reservationID,CustomerID,amountpaid,date,firstname,middlename,lastname,hotelID }

- **payment details**:
    - reservationid->{payment method,payment status}
- **Rooms:**
    - roomid->{room category,numberbeds,startdate,enddate,current price,CustomerID,room number ,hotelID,ReservationID}

- **Hotel_contact_number:**
    - contact number->{HotelID}
- **Customer_contacr_number**:
    - contact number->{CustomerID}

Normalization and testing for lossless join property:

1NF:

In the problem statement, the Customers table has a multivalued attribute called Customer contact number, this can be considered multivalued as the field can have more than one value associated with the key of the entity(CustomerID).
Similarly, the relation, Hotel Hotel_contact_number is also a multivalued attribute.
We also need to identify composite attributes. Once identified these composite attributes are to be split into atomic data types. In the above-mentioned ER-diagram, we can see that the composite attributes are :

Address-City, Street, Postal code, Country
Date-StartDate,EndDate
Name-Firstname, Middle name, Last Name

In the relational schema, these attributes are represented as atomic attributes.
To resolve this to higher Normal forms we create a separate relation for multivalued attributes. This has been done by the creation of Hotel_contact_number and Customer_contact_number.
All relations in this Database satisfies the following properties :

1. Should contain only Single Valued Attributes.
2. Attribute Domain does not change.
3. There is a unique name for every Attribute/Column.

Thus we can conclude this database is normalized to a normal form equal to or higher than 1NF.

2NF:

In the above-mentioned schema, all the relations in the Database do not have composite primary keys. Second Normal Form (2NF) is based on the concept of full functional dependency. Second Normal Form applies to relations with composite keys, i.e, relations with a primary key composed of two or more attributes. All the relations have Single-attribute primary key, thus it can be concluded that the database is normalized to a normal form equal to or higher than 2NF.

3NF:

A relation is in the third normal form if there is no transitive dependency for non-prime attributes as well as it is in second normal form. We refer to the functional dependencies identified above to check for transitive dependency for non-prime attributes for every relation in the Database. Thus it can be concluded that this database is normalized to a normal form equal to or higher than 3NF.

BCNF:

The relations are in BCNF since there are no non-prime attributes that functionally determine other attributes. In other words for every dependency A->B, A is a superkey.

Since the relations have been attained by first creating an ER model and then converting it to a relational schema, all the relations are in the normalized form of 3NF and BCNF.

Testing for lossless join Property:

from the ER diagram, “Bill” entity has to be a separate table but this table might consist of redundant values thus we can decompose this table into 2 smaller tables.
Bill(Billid,amountpaid,date,firstname,middlename,lastname,hotelID,CustomerID reservationID,PaymentStatus,Paymentmethod)
decomposed into 2 smaller tables
1.Bill(billid,amountpaid,date,firstname,middlename,lastname,hotelID,CustomerID ,reservationID) consider this as R1
2.payment details(reservationid,payment method,payment status) consider this as R2.

Proving the decomposition is lossless by using the algorithm- Non-additive (Lossless) Join Property

Matrix at the start of the algorithm:

	BillID	amount paid	curr date	fname	mname	lname	Hotel ID	CustID	Reservation ID	payment method	payment status
Bill	a1	a2	a3	a4	a5	a6	a7	a8	a9	b1,10	b1,11
payment	b2,1	b2,2	b2,3	b2,4	b2,5	b2,6	b2,7	b2,8	a9	a10	a11

Bill_ID is the primary key for the relation R1.
Reservation_ID is the primary key for the relation R2.
Matrix after applying the algorithm

	BillID	amount paid	curr date	fname	mname	lname	Hotel ID	CustID	Reservation ID	payment method	payment status
Bill	a1	a2	a3	a4	a5	a6	a7	a8	a9	~~b1,10~~ a10	~~b1,11~~ a11
Payment	b2,1	b2,2	b2,3	b2,4	b2,5	b2,6	b2,7	b2,8	a9	a10	a11

We can see that one of the rows completely contains a's .
One other way to make sure that the lossless join property hold is to check:
1. Att(R1) U Att(R2) = Att(R)
2. Att(R1) ∩ Att(R2) != Φ
3. The common attribute must be a key for at least one relation (R1 or R2).
All the relations I have used to satisfy the lossless join property.

Thus we can say that lossless join is verified.

DDL

```
CREATE TABLE Customer
( SSN INT NOT NULL,
  CustomerID INT NOT NULL,
  firstname VARCHAR(10) NOT NULL,
  middlename VARCHAR(10) NOT NULL,
```

```
  lastname VARCHAR(10) NOT NULL,
  email VARCHAR(20) NOT NULL,
  city VARCHAR(20) NOT NULL,
  street VARCHAR(20),
  postalcode VARCHAR(20),
  Age INT NOT NULL,
  country VARCHAR(20) NOT NULL,
  PRIMARY KEY (CustomerID),
  UNIQUE (SSN)
);

CREATE TABLE Customer_contact_number
(
  contact_number VARCHAR(12) NOT NULL,
  CustomerID INT NOT NULL,
  PRIMARY KEY (contact_number,CustomerID),
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE
CASCADE
);
CREATE TABLE Hotel
(
  HotelID INT NOT NULL,
  street VARCHAR(20),
  postalcode VARCHAR(20),
  city VARCHAR(20) NOT NULL,
  hotelname VARCHAR(20) NOT NULL,
  PRIMARY KEY (HotelID)
);
CREATE TABLE payment_details
(
  reservationID INT NOT NULL,
  payment_method VARCHAR(10) NOT NULL,
  payment_status VARCHAR(10) NOT NULL,
  PRIMARY KEY (reservationID)
);
CREATE TABLE Hotel_contact_number
(
  contact_number VARCHAR(12) NOT NULL,
  HotelID INT NOT NULL,
  PRIMARY KEY (contact_number, HotelID),
  FOREIGN KEY (HotelID) REFERENCES Hotel(HotelID) ON DELETE CASCADE
);
CREATE TABLE Rooms
(
  RoomID INT NOT NULL,
```

```
  room_category VARCHAR(10) NOT NULL,
  number_of_beds INT NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  current_price INT NOT NULL,
  ReservationID INT NOT NULL,
  roomnumber INT NOT NULL,
  CustomerID INT NOT NULL,
  HotelID INT NOT NULL,
  PRIMARY KEY (RoomID),
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE
CASCADE,
  FOREIGN KEY (HotelID) REFERENCES Hotel(HotelID) ON DELETE CASCADE,
  FOREIGN KEY (reservationID) REFERENCES payment_details(reservationID)
ON DELETE CASCADE
);
CREATE TABLE Bill
(
  BillID INT NOT NULL,
  Amount INT,
  lastname VARCHAR(10) ,
  firstname VARCHAR(10),
  middlename VARCHAR(10),
  HotelID INT NOT NULL,
  reservationID INT NOT NULL,
  currentdate date DEFAULT GETDATE() ,
  CustomerID INT NOT NULL,
  PRIMARY KEY (BillID),
  FOREIGN KEY (HotelID) REFERENCES Hotel(HotelID) ON DELETE CASCADE,
  FOREIGN KEY (reservationID) REFERENCES payment_details(reservationID)
ON DELETE CASCADE,
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE
CASCADE
);

CREATE TABLE AUDIT_DETAILS
(
 BillID INT,
 start_date date,
 end_date date,
 number_of_days INT,
 total_amount_paid INT,
 PRIMARY KEY (BillID)
);
```

Implementation of check constraints on the above-created tables

1. In the Customer relation, a check constraint is added to validate the email to match the standard format. Also, a constraint on Age is applied where customers under the age of 18 cannot make a reservation.
the postal code of a location should follow a specific format.

```
alter table Customer add constraint customer_valid check (
email like '%_@__%.__%' AND
Age>=18  AND
postalcode like '[0-9][0-9][0-9][0-9][0-9][0-9]'
);
```

2.In the Customer_contact_number we check if all the contact numbers are 10 digits long and contain only numeric instances.

```
alter table  Customer_contact_number add constraint cus_ph_chk check(
contact_number like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
);
```

3. In the room relation, a check constraint is added to ensure the start
the date is before the end date, and also the room category is chosen from the provided values as well as the number of beds is less than or equal to 4.

```
alter table Rooms add constraint chk_room check (
end_date>=start_date AND
room_category IN
('Single','Double','Triple','Quad','King','Connecting','Suite') AND
number_of_beds<=4
);
```

4.In the Hotel_contact_number we check if all the contact numbers are 10 digits long and contain only numeric instances.

```
alter table Hotel_contact_number add constraint chk_hotel_contact check
(
contact_number like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
);
```

5.checks if the postal code of a location should follows a specific format

```
alter table Hotel add constraint chk_hotel check (
```

```
postalcode like '[0-9][0-9][0-9][0-9][0-9][0-9]'
);
```

6. In the Payment_details relation, a check constraint is used to check if the payment method is valid and whether the payment status either takes a value of ‘failed’ or ‘successful’.

```
alter table payment_details add constraint chk_payment_details check(
payment_method IN ('Cash','Card','Netbanking','UPI') AND
payment_status IN ('Failed','successful')
);
```

Triggers

1. The goal of the following trigger is to automatically generate a bill provided CustomerID and the name of the recipient. Here I have used temporary variables to store values which is obtained from multiple other tables. The total amount the customer is supposed to pay is auto-generated depending on the duration of the stay. This trigger is also responsible for logging data onto AUDIT_DETAILS. This tables keeps track of all the income to the organizations.

```
CREATE TRIGGER income_trigger ON Bill AFTER INSERT AS
BEGIN
      DECLARE @bill_temp INT;
      DECLARE @id INT;
      SELECT
            @id=inserted.CustomerID,
            @bill_temp=inserted.BillID
      from inserted;
      DECLARE @sd date;
      DECLARE @ed date;
      DECLARE @current_price INT;
      DECLARE @fname VARCHAR(10);
      DECLARE @mname VARCHAR(10);
      DECLARE @lname VARCHAR(10);
      DECLARE @temp_amount INT;
            SELECT @fname=Customer.firstname,
                     @mname=Customer.middlename,
                     @lname=Customer.lastname
            from Customer WHERE Customer.CustomerID=@id;
```

```
            SELECT @sd=Rooms.start_date,
                     @ed=Rooms.end_date,
                     @current_price=Rooms.current_price,
                     @temp_amount=DATEDIFF(DAY,@sd,@ed)*@current_price
            FROM Rooms WHERE Rooms.CustomerID=@id;
            UPDATE Bill
                  SET Amount=DATEDIFF(DAY,@sd,@ed)*@current_price,
firstname = @fname,lastname= @lname,middlename= @mname
                  WHERE Bill.CustomerID=@id

            INSERT INTO
AUDIT_DETAILS(BillID,total_amount_paid,start_date,end_date)VALUES
(@bill_temp,@temp_amount,@sd,@ed)

            UPDATE AUDIT_DETAILS
                  SET
number_of_days=DATEDIFF(DAY,@sd,@ed),total_amount_paid=DATEDIFF(DAY,@sd,
@ed)*@current_price
                  WHERE AUDIT_DETAILS.BillID=@bill_temp
            DELETE FROM ROOMS WHERE ROOMS.CustomerID=@id
END;
```

2.This trigger is activated after an insert operation has been performed on the table Rooms This trigger is responsible for logging data onto a table called Rooms_log which keeps track of the history of rooms booked by a Customer.

```
CREATE TRIGGER Rooms_logger ON Rooms AFTER INSERT AS
BEGIN
      DECLARE @custid INT;
      DECLARE @roomid INT;
      DECLARE @sd_1 date;
      DECLARE @ed_1 date;
      SELECT
            @custid=inserted.CustomerID,
            @roomid=inserted.RoomID,
            @sd_1=inserted.start_date,
            @ed_1=inserted.end_date
      from inserted;
```

```
      Insert into Rooms_log(CustomerID,RoomID,start_date,end_date)
values(@custid,@roomid,@sd_1,@ed_1)

END;
```

SQL Queries

1. The following SQL query can be used to determine the maximum revenue generated by a single branch of a hotel. We use 2 aggregate functions: MAX() , SUM().

```
SELECT MAX(tmp.total_revenue) as max_revenue
from
      (SELECT Bill.HotelID,SUM(Amount) as total_revenue
      FROM Bill
      GROUP BY HotelID) as tmp
```

2. Given a CustomerID this SQL query can access the details of history/previous bookings of a Customer in any of the Hotels in the DB. I have used Inner join in this query as I had to select all rows from both participating tables as long as there is a match between the columns.

```
Select
Rooms_log.CustomerID,firstname,middlename,lastname,email,RoomID,start_da
te,end_date
from Rooms_log
INNER JOIN Customer
ON Rooms_log.CustomerID=Customer.CustomerID
WHERE Rooms_log.CustomerID='8999'
```

3. This SQL query finds details of customers who have stayed in a room for more than a given number of days. The DATEDIFF() function helps in calculating the length of stay.

```
SELECTfirstname,middlename,lastname,email,city,age
FROM Customer
WHERE CustomerID in
(SELECT CustomerID FROM Rooms_log
WHERE DATEDIFF(DAY,start_date,end_date)>3);
```

4.The following query selects important data like HotelID,reservationID, CustomerID for a given customer and displays it. As I wanted all rows from the left table included I have used Left outer join.

```
SELECT
Customer.firstname,Customer.middlename,Customer.lastname,Bill.HotelID,Bi
ll.reservationID,Bill.CustomerID
FROM   Customer
LEFT OUTER JOIN Bill
ON  Customer.CustomerID=Bill.CustomerID
```

Few other SQL queries:

5. The following SQL query returns the number of times a given customer has booked in any of the hotels in the hotel chain.

```
SELECT Count(CustomerID) as numberoftimes from Rooms_log
WHERE CustomerID='8999'
```

6. The following query returns the total revenue of all the hotels.

```
SELECT SUM(total_amount_paid) as total_rev
FROM AUDIT_DETAILS;
```

Conclusion

The main goal of the above model is to simplify, organize the process of booking of hotel rooms as well create an efficient way to maintain all said records.
The resulting project has made it capable to audit & log all the aforementioned expenditures for the organization, making it easier to come upon their gross income.
A bill is generated by calculating the cost based on parameters such as current price, room type & duration of stay.

The project achieves efficiency by keeping track of returning customers, thus not having the need to reenter all the data into the database. It also covers a multitude of check constraints that ensures the upload of only sensible data into the Database.

While the project achieves a good amount of usability and efficiency, it isn't capable of performing all the operations a professional hotel reservation system does, but it does provide the basics and fundamentals of its working. The Bill generation can aim to be more precise by being made to generate a bill for every single customer.

The project provides a good basis for scalability and holds the ease to be adapted for a more real-life application. It is easy to use and contains all the bones a traditional hotel reservation booking system holds.

The complete code for this writeup and GUI made is open-source and can be found here;

[https://github.com/anand-371/Hotel_reservation_system]