# A REPORT

**ON**

**Hotel Management System**

| | |
|---|---|
| **Sourabh Bhandari** | **2021A7PS2412P** |
| **Meet Vithalani** | **2021A7PS0555P** |

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI (Rajasthan)**

For the fulfilment of the Assignment in the course- **Database Systems- CS F212**

April 2023

# ***Index***

Some Important links:

1. Videos of frontend documentation and project explanation : https://drive.google.com/drive/folders/1dzlM3BisDAfC1m64CO9TPeHWHNR1K9iw?usp=sharing
2. Github link: https://github.com/SaurabhRBhandari/Hotel-Management-System/tree/master

## Starting The Program

We have tested the program on Linux(Ubuntu); here are the instructions for running the application on Linux.

1. Open the project directory in terminal.
2. Run `sudo ./installs_linux.`
3. All the necessary packages are now installed
4. Create a new mysql user with given credentials
    a. Start mysql CLI by running the following
    ```
    sudo mysql
    ```
    b. Run the following sql queries:
    ```
    CREATE USER ‘user’@’localhost’ IDENTIFIED BY ‘password’;
    GRANT ALL PRIVILEDGES ON *.* TO ‘user’@’localhost
    ```
5. To create the database and populate it with sample data, run
    ```
    python3 reset_db.py
    ```
6. Now the project is ready to run, run the command
    ```
    python3 main.py.
    ```

7. A login window should appear, for user login the username is “user” and the password is ”password”. For admin login the username is “admin” and the password is “password”.

8. You can find the required functionalities implemented .

## Tech Used

We have used the following services/packages

1.mysql : To store and access the database

2.tkinter,tkcalendar: To display the frontend

3.mysql-connector-python: To integrate sql database with python backend.

4.matplotlib: To graph revenues generated by each type of room

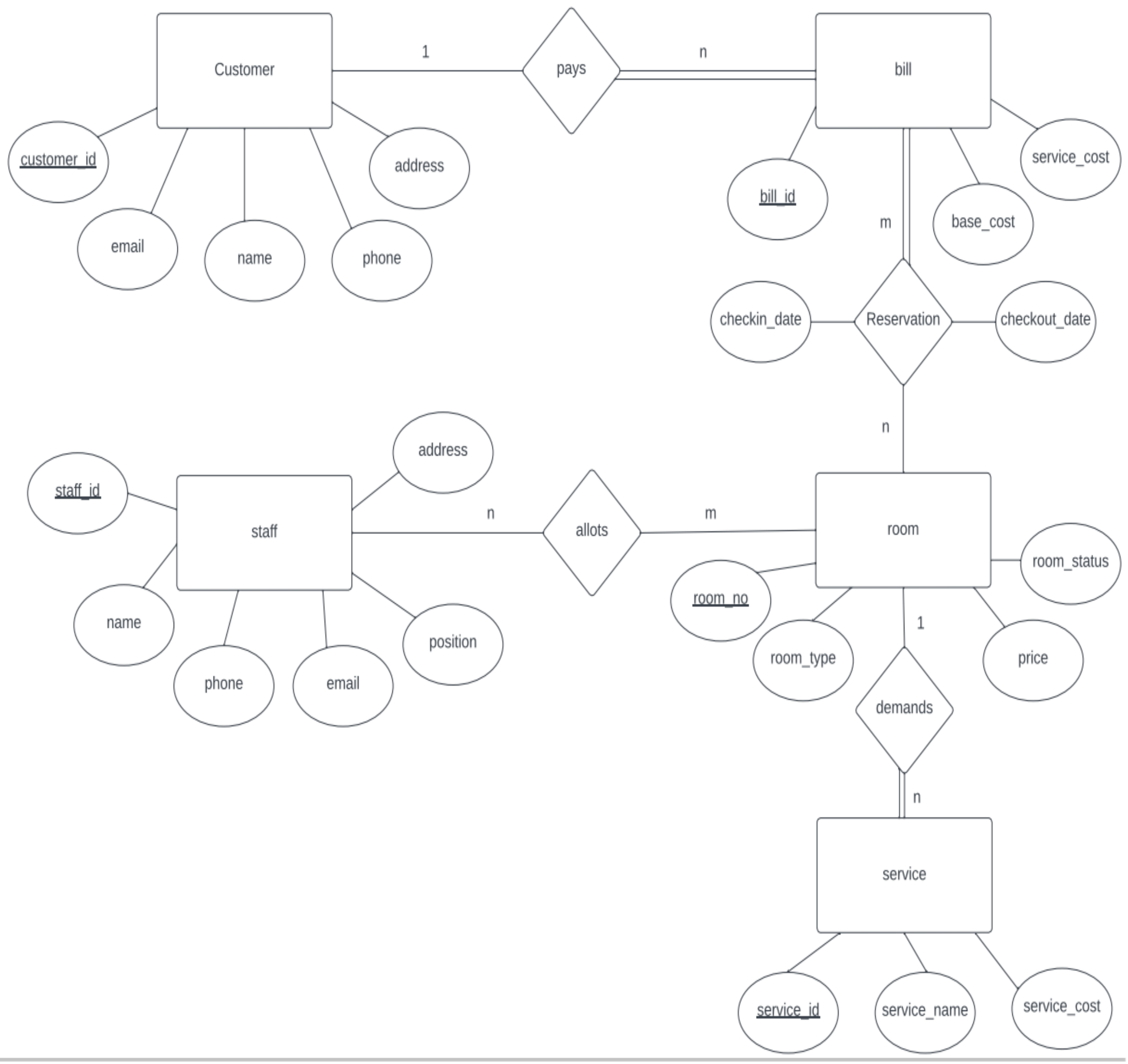
5.PIL: To display button icons

## Anti Plagiarism Statement

I declare in my honour that what has been written in this work has been written by me and that, with the exception of quotations, no part has been copied from scientific publications, the Internet or from research works already presented in the academic field by me or by other students.  In the case of parts taken from scientific publications, from the Internet or from other documents, I have expressly and directly indicated the source at the end of the quotation or at the foot of the page.  I also declare that I have taken note of the sanctions provided for in case of plagiarism by the current Study Regulations

| **Name** | **ID** | **Sign** |
|---|---|---|
| Sourabh Bhandari | 2021A7PS2412P | [illegible] |
| Meet Vithalani | 2021A7PS0555P | Meet |

# ER Diagram


Customer
1
pays
n
bill
customer_id
email
name
phone
address
bill_id
m
base_cost
service_cost
checkin_date
Reservation
checkout_date
n
address
staff_id
staff
n
allots
m
room
room_status
name
phone
email
position
room_no
room_type
1
price
demands
n
service
service_id
service_name
service_cost

## Documentation for ER diagram

Hotel Management System is a software for hotel staff that provides various necessary tools to efficiently manage hotel operations like room booking, room service, billing, etc.

### Entities:

1. Customer entity : Stores customer information and it has the following attributes:
    - customer_id(Primary Key)
    - name
    - phone
    - email
    - address

2. Room entity : Stores information about a particular room like type of room, its price and if it is occupied or empty currently. It has following attributes:
    - room_no(Primary Key)
    - room_type
    - price
    - room_status.

3. Bill entity : Stores information about the amount a customer is charged. It has following attributes:
    - bill_id(Primary Key)
    - base_cost
    - service_cost.

4. Staff entity : Stores staff information and has the following attributes:
    - staff_id (Primary Key)
    - name
    - phone
    - email
    - position
    - address.
5. Service table : Provides information about services used by the customers. It has attributes:
    - service_id(Primary Key)
    - service_name
    - service cost.

Relationships:

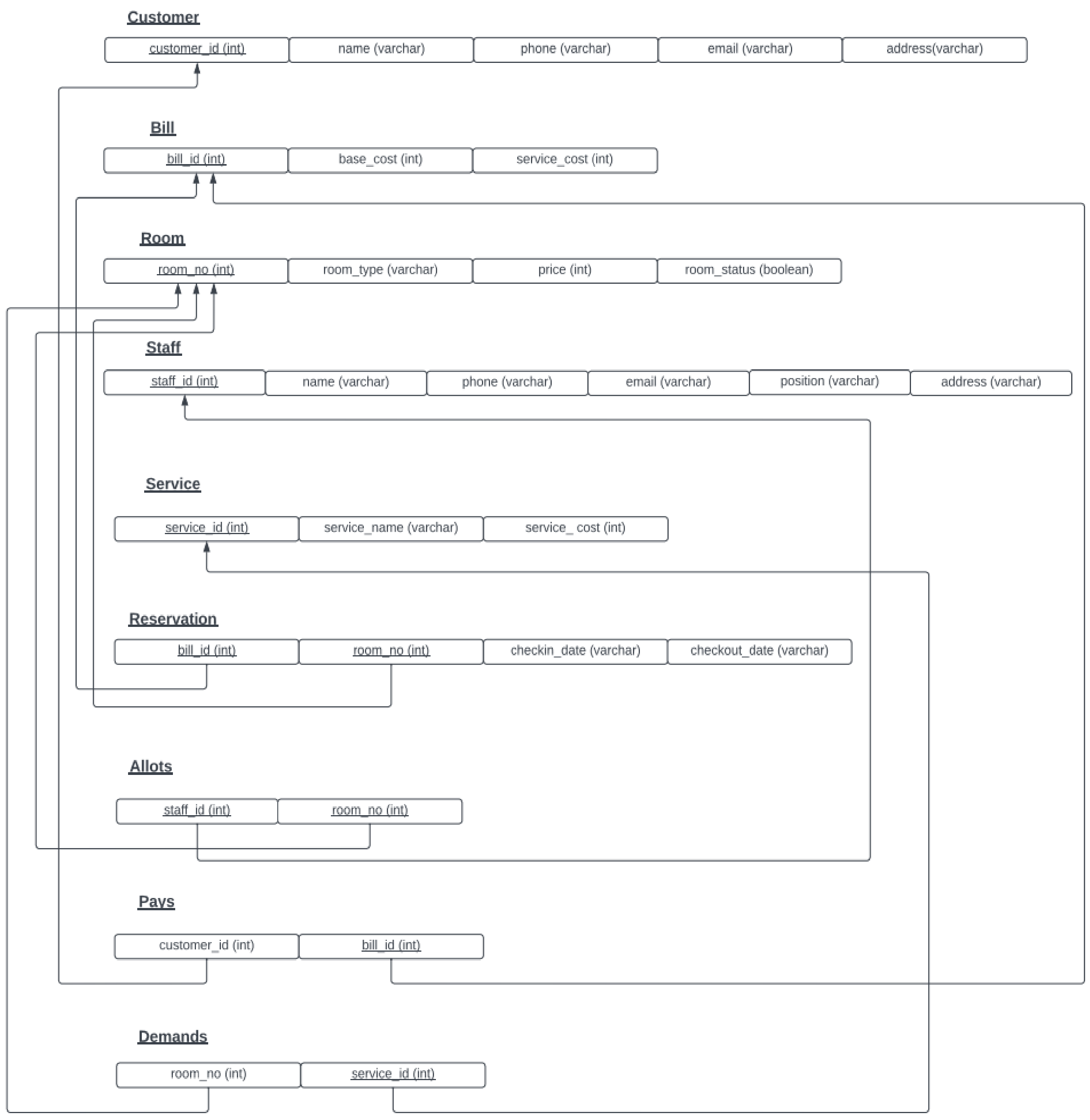
1. Pays : Describes a one-to-many relationship between customer entity and bill entity. One customer can have many bills but each bill belongs to a unique customer. Also, it involves total participation for only the bill table since every bill must belong to some customer. It has only bill_id as the primary key.

2. Reservation: Describes many-to-many relationship between room entity and bill entity. A room can have multiple billing as well as multiple rooms can be booked in a single bill. Also, the participation is total only for bill entity since every bill will have some room reserved but some rooms might not be reserved ever. Its primary key is bill_id along with room_no.  It has following attributes:
   - checkin_date
   - checkout_date.

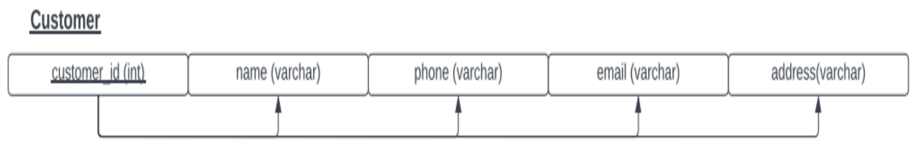
3. Allots : Describes a many-to-many relationship between room entity and staff entity. One room can have multiple staff people involved for its service and one staff person might also be serving multiple rooms. It involves partial participation since some staff might not be allotted to any room or a room might not need any service from staff at a given time. This table has staff_id along with room_no as the primary key.

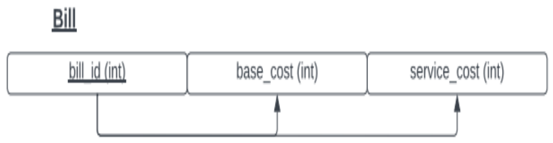
4. Demands : Describes one-to-many relationship between room entity and service entity. One room can demand for multiple services but since every service availed by any room has a unique service id, one service id can only correspond to one room. Also, the participation is total only for service entity since every room might not have availed service but every service availed has a room corresponding to it. This table has only service_id as the primary key.

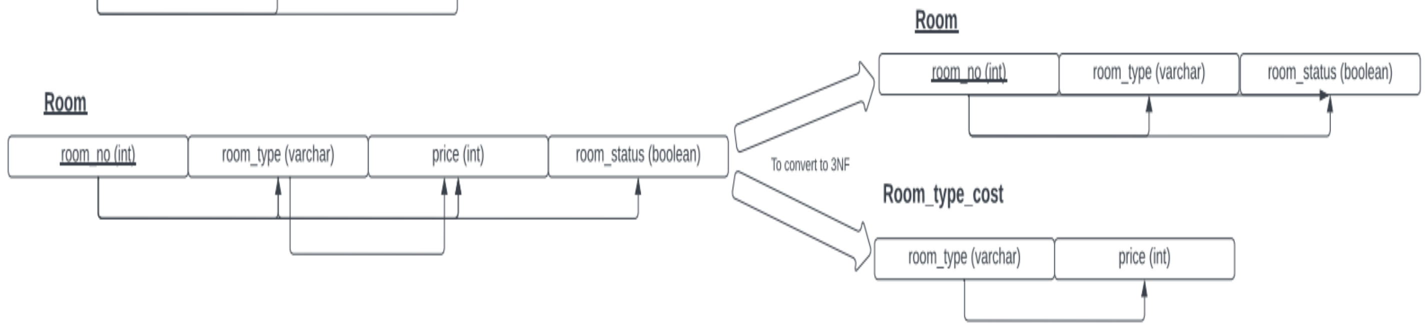
# Relational Schema before normalization

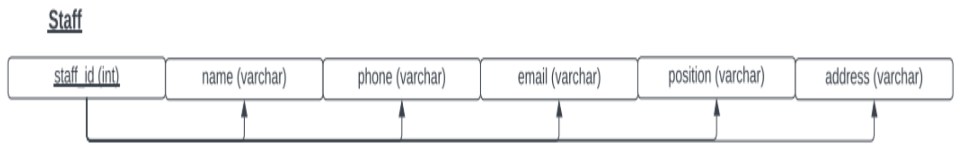


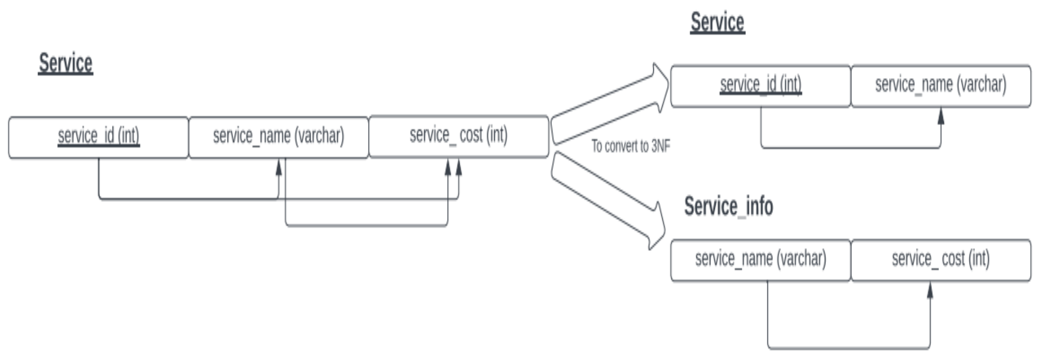
# Functional Dependencies

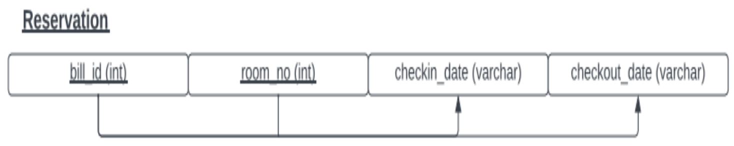












# Conversion to 3NF

Converting to 1NF:

- On looking at the relational schema, we observe that none of the attributes in the relational schema is multivalued or composite.
- Hence, we can conclude that the relational schema is already in 1NF.

Converting to 2NF:

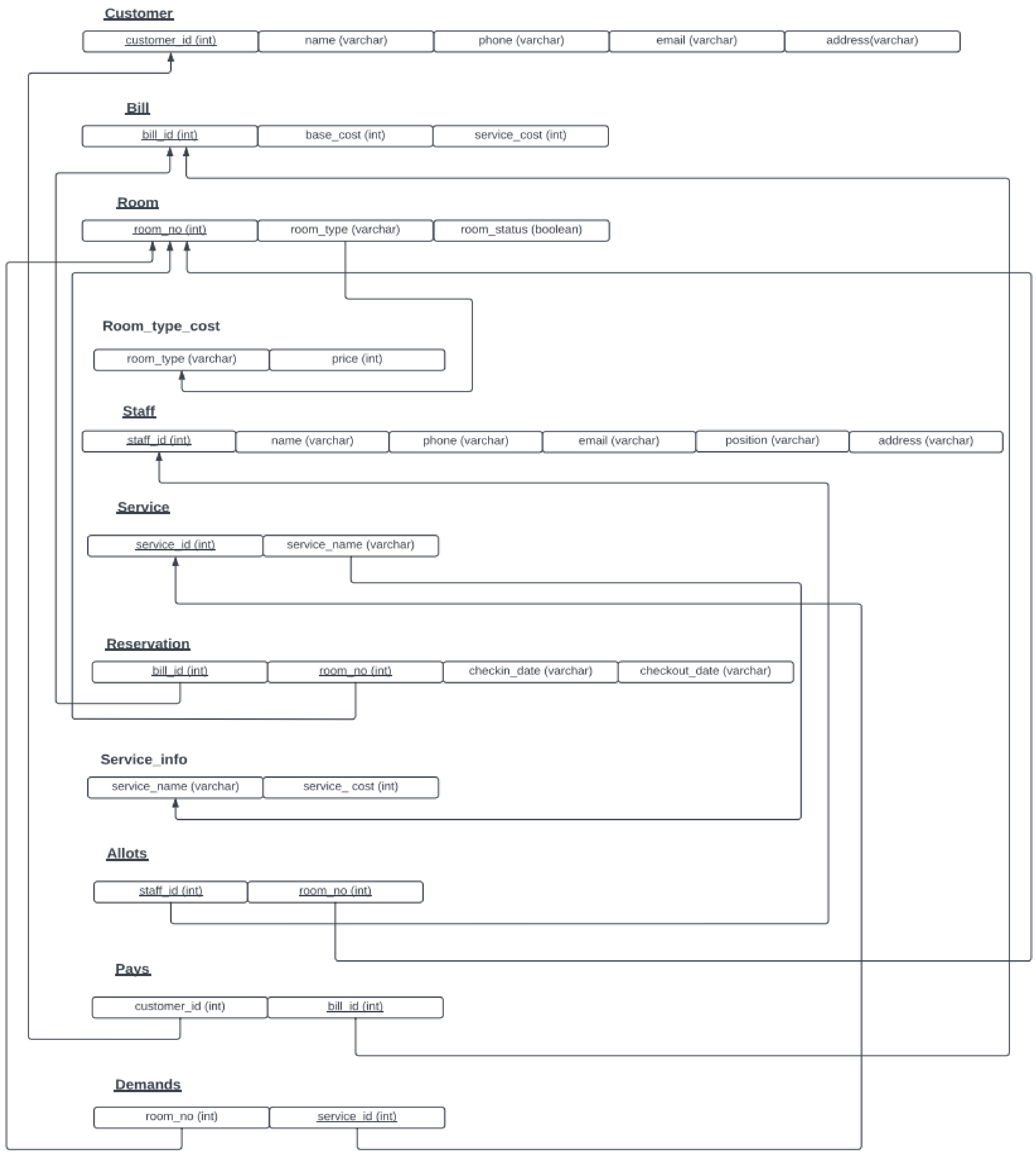
- On looking at the relational schema, we observe that no sub-part of any candidate key of any table could uniquely identify any other attributes i.e every non-prime attribute is fully functional dependent.
- Hence, we can conclude that the relational schema is already in 2NF.

Converting to 3NF:

1. In the room table, the prime attribute is only room_no.
    - Now, we observe that there is a functional dependency room_type→price. Here, both room_type and price are non-prime attributes.
    - This means that we can uniquely determine price by just room_type which is a non-prime attribute.
    - Hence, to remove this dependency, we will remove price as an attribute from the room table and create a separate table named room_type_cost which will have room_type and price as its attributes along with room_type as primary key.

2. In the service table, the prime attribute is only service id.
    - Now, we observe that there is a functional dependency service_name→service_cost. Here, both service_name and service_cost are non-prime attributes.
    - This means that we can uniquely determine service_cost by just service_name which is a non-prime attribute.
    - Hence, to remove this dependency, we will remove service_cost as an attribute from the service table and create a separate table named service_info which will have service_name and service_cost as its attributes along with service_name as primary key.

# Relational Schema after normalization



# SQL Queries

We now explain the important queries used.

## Creating the tables

```
CREATE DATABASE IF NOT EXISTS hoteldb;

USE hoteldb;

CREATE TABLE IF NOT EXISTS customer (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    phone VARCHAR(255),
    email VARCHAR(255),
    address VARCHAR(255));

CREATE TABLE IF NOT EXISTS room_type_cost (
    room_type VARCHAR(255) PRIMARY KEY,
    price INT);

CREATE TABLE IF NOT EXISTS room (
    room_no INT PRIMARY KEY,
    room_type VARCHAR(255) REFERENCES room_type_cost,
    room_status BOOLEAN);

CREATE TABLE IF NOT EXISTS bill (
    bill_id INT AUTO_INCREMENT PRIMARY KEY,
    base_cost INT,
    service_cost INT);

CREATE TABLE IF NOT EXISTS pays (
    customer_id INT REFERENCES customer ON DELETE CASCADE,
    bill_id INT REFERENCES bill,
    PRIMARY KEY (customer_id, bill_id));

CREATE TABLE IF NOT EXISTS staff (
    staff_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    phone VARCHAR(255),
    email VARCHAR(255),
    address VARCHAR(255),
    position VARCHAR(255));

CREATE TABLE IF NOT EXISTS allots (
    staff_id INT REFERENCES staff ON DELETE CASCADE,
    room_no INT REFERENCES room,
    PRIMARY KEY (staff_id, room_no));

CREATE TABLE IF NOT EXISTS service_info (
    service_name VARCHAR(255) PRIMARY KEY ,
    service_cost INT);

CREATE TABLE IF NOT EXISTS service (
    service_id INT AUTO_INCREMENT PRIMARY KEY,
    service_name VARCHAR(255) REFERENCES service_info,
    quantity INT);

CREATE TABLE IF NOT EXISTS demands (
    room_no INT REFERENCES room,
    service_id INT REFERENCES service,
    PRIMARY KEY (service_id));

CREATE TABLE IF NOT EXISTS reservation (
    room_no INT REFERENCES room,
    bill_id INT REFERENCES bill,
    checkin_date DATE,
    checkout_date DATE ,
    PRIMARY KEY (room_no, bill_id));
```

- The "customer" table contains information about hotel customers such as their name, phone number, email, and address.
- The "room_type_cost" table contains the cost of each room type available in the hotel.
- The "room" table contains information about each room such as the room number, room type, and room status.
- The "bill" table contains information about each bill generated by the hotel such as the base cost and service cost.
- The "pays" table contains information about the payment status of each customer's bill.
- The "staff" table contains information about hotel staff such as their name, phone number, email, address, and position.

- The "allots" table contains information about the room assigned to each staff member.
- The "service_info" table contains information about each service available in the hotel such as the service name and service cost.
- The "service" table contains information about the service used by each customer such as the service name and quantity.
- The "demands" table contains information about the service demanded in each room.
- The "reservation" table contains information about each reservation made by the customer such as the room number, bill ID, check-in date, and check-out date.

## Inserting Guest

→This procedure creates a new guest in the customer table with the provided information

```
CREATE PROCEDURE IF NOT EXISTS new_guest(
    IN p_name VARCHAR(255),
    -- Input parameter for the guest name
    IN p_phone VARCHAR(20),
    -- Input parameter for the guest phone number
    IN p_email VARCHAR(255),
    -- Input parameter for the guest email
    IN p_address VARCHAR(255) -- Input parameter for the guest address
) BEGIN
INSERT INTO
    customer -- Insert a new row into the customer table with the
provided information
VALUES
    (NULL, p_name, p_phone, p_email, p_address);

END;
```

- This is a SQL procedure that creates a new guest in the customer table with the provided information. The procedure takes four input parameters: p_name for the guest name, p_phone for the guest phone number, p_email for the guest email, and p_address for the guest address.
- The "IF NOT EXISTS" statement checks if the procedure already exists before creating it to avoid errors.
- Inside the procedure, an "INSERT INTO" statement is used to add a new row to the "customer" table with the provided information. The "NULL" value is used for the

"customer_id" column, which is set to auto-increment, so a new value will be generated automatically.

- To execute the procedure, you can call it with the required input parameters using a SQL editor or a command line interface in a MySQL database management system. For example, to create a new guest with the name "John Doe", phone number "1234567890", email "johndoe@example.com", and address "123 Main St", you can execute the following SQL command:
    - CALL new_guest('John Doe', '1234567890', 'johndoe@example.com', '123 Main St');

## Inserting rooms

→ This procedure creates a new room in the hotel database

```
CREATE PROCEDURE IF NOT EXISTS new_room(
    IN p_room_no INT,
    -- The room number of the new room
    IN p_room_type VARCHAR(255),
    -- The type of the new room (e.g. "Single", "Double", etc.)
    IN p_room_status BOOLEAN -- The status of the new room (True =
available, False = occupied)
) BEGIN
INSERT INTO
    room -- Insert a new row into the "room" table
VALUES
    (
        p_room_no,
        -- Use the value passed in as the room number
        p_room_type,
    -- Use the value passed in as the room type
        p_room_status
    );

-- Use the value passed in as the room status
END;
```

- This is a SQL procedure that creates a new room in the hotel database. The procedure takes three input parameters: p_room_no for the room number of the new room, p_room_type for the type of the new room (e.g. "Single", "Double", etc.), and p_room_status for the status of the new room (True = Available for  use, False = Under maintenance).

- The "IF NOT EXISTS" statement checks if the procedure already exists before creating it to avoid errors.
- Inside the procedure, an "INSERT INTO" statement is used to add a new row to the "room" table with the provided information. The values passed in for p_room_no, p_room_type, and p_room_status are used to populate the corresponding columns in the table.
- To execute the procedure, you can call it with the required input parameters using a SQL editor or a command line interface in a MySQL database management system. For example, to create a new room with the room number 101, room type "Single", and status "True" , you can execute the following SQL command:
    - CALL new_room(101, 'Single', True);

## Inserting Hotel Staff

→ This procedure creates a new staff member in the staff table with the provided information

```
CREATE PROCEDURE IF NOT EXISTS new_staff(
    IN p_name VARCHAR(255),
    -- Input parameter for the staff member's name
    IN p_phone VARCHAR(20),
    -- Input parameter for the staff member's phone number
    IN p_email VARCHAR(255),
    -- Input parameter for the staff member's email
    IN p_address VARCHAR(255),
    -- Input parameter for the staff member's address
    IN p_position VARCHAR(255) -- Input parameter for the staff member's
position
) BEGIN
INSERT INTO
    staff -- Insert a new row into the staff table with the provided
information
VALUES
    (
        NULL,
        p_name,
        p_phone,
        p_email,
        p_address,
        p_position
    );

END;
```

- This is a SQL procedure that creates a new staff member in the staff table with the provided information. The procedure takes five input parameters: p_name for the staff member's name, p_phone for the staff member's phone number, p_email for the staff member's email, p_address for the staff member's address, and p_position for the staff member's position.
- The "IF NOT EXISTS" statement checks if the procedure already exists before creating it to avoid errors.
- Inside the procedure, an "INSERT INTO" statement is used to add a new row to the staff table with the provided information. The values passed in for p_name, p_phone, p_email, p_address, and p_position are used to populate the corresponding columns in the table.
- To execute the procedure, you can call it with the required input parameters using a SQL editor or a command line interface in a MySQL database management system. For example, to create a new staff member with the name "John Smith", phone number "123-456-7890", email "john.smith@example.com", address "123 Main St, Anytown, USA", and position "Front Desk Clerk", you can execute the following SQL command:
    - CALL new_staff('John Smith', '123-456-7890', 'john.smith@example.com', '123 Main St, Anytown, USA', 'Front Desk Clerk');

## Updating guest Personal Information

→ This procedure updates the information of a customer in the customer table based on the provided customer_id

```
CREATE PROCEDURE IF NOT EXISTS update_customer(
    IN id INT,
    -- Input parameter for the customer_id to be updated
    IN name VARCHAR(255),
    -- Input parameter for the updated name of the customer
    IN phone VARCHAR(20),
    -- Input parameter for the updated phone number of the customer
    IN email VARCHAR(255),
    -- Input parameter for the updated email of the customer
    IN address VARCHAR(255) -- Input parameter for the updated address
of the customer
) BEGIN
UPDATE
    customer
SET
    -- Update the row in the customer table with the provided
information
    name = name,
    phone = phone,
    email = email,
    address = address
WHERE
    customer_id = id;

END;
```

This procedure updates the information of a customer in the customer table based on the provided customer_id, with the new values for name, phone, email, and address. The procedure takes the customer_id as the first input parameter and the updated information as subsequent input parameters. The procedure then updates the row in the customer table that matches the provided customer_id with the new information.

## Changing room status between available/under maintenance

→The following stored procedure toggles the status of a specific room.

```
-- It takes in one parameter: the room number.
CREATE PROCEDURE toggle_status(IN p_room_no INT) BEGIN
UPDATE
    room
SET
    room_status = NOT room_status
WHERE
    room_no = p_room_no;

END;
```

This stored procedure toggles the status of a specific room in the hotel database. It takes in one parameter, which is the room number, and changes the room_status from available to occupied or from occupied to available.

## Generating Bills

→Procedure to generate a bill for a given bill ID

```
CREATE PROCEDURE IF NOT EXISTS generate_bill(IN bill_id INT) BEGIN --
Select customer information, base cost, service cost, check-in date,
check-out date, and number of rooms for the given bill ID
SELECT
    c.name,
    c.email,
    b.base_cost,
    b.service_cost,
    r.checkin_date,
    r.checkout_date,
    COUNT(DISTINCT r.room_no) AS num_rooms
FROM
    pays p
    JOIN customer c ON p.customer_id = c.customer_id
    JOIN bill b ON p.bill_id = b.bill_id
    JOIN reservation r ON p.bill_id = r.bill_id
WHERE
    p.bill_id = bill_id
GROUP BY
    c.name,
    c.email,
    b.base_cost,
    b.service_cost,
    r.checkin_date,
    r.checkout_date;

END;
```

- The generate_bill procedure is used to generate a bill for a given bill ID. The procedure takes one input parameter: the bill ID for which to generate the bill.
- The procedure first joins the pays, customer, bill, and reservation tables to gather the necessary information for the bill. It selects the customer name and email, the base cost and service cost of the bill, the check-in and check-out dates, and the number of rooms associated with the bill.
- The procedure then groups the selected data by the customer name, email, base cost, service cost, check-in date, and check-out date.
- Finally, the procedure returns the selected and grouped data to generate the bill.

## Book a Room

```
CREATE PROCEDURE IF NOT EXISTS book_rooms(
    IN customer_id INT,
    IN room_nos VARCHAR(255),
    IN checkin_date DATE,
    IN checkout_date DATE
) BEGIN -- Create a new bill for the customer
INSERT INTO
    bill (base_cost, service_cost)
VALUES
    (0, 0);

SET
    @bill_id = LAST_INSERT_ID();

-- Insert a new record in the pays table to link the customer and bill
INSERT INTO
    pays (customer_id, bill_id)
VALUES
    (customer_id, @bill_id);

-- Split the room numbers string into a temporary table
DROP TEMPORARY TABLE IF EXISTS temp_room_nos;

-- drop temporary table if it exists
CREATE TEMPORARY TABLE temp_room_nos (room_no INT);

-- create temporary table to hold the room numbers
```

```
SET
    @query = CONCAT(
        "INSERT INTO temp_room_nos (room_no) VALUES ",
        room_nos,
        ";"
    );

-- construct query to insert the room numbers into temporary table
PREPARE stmt
FROM
    @query;

-- prepare the query for execution
EXECUTE stmt;

-- execute the query to insert the room numbers into temporary table
DEALLOCATE PREPARE stmt;

-- deallocate the prepared statement to free up resources
-- Book each room in the temporary table
WHILE EXISTS(
    SELECT
        *
    FROM
        temp_room_nos
) DO -- loop while there are still rooms to book
-- Get the first available room of the desired type
SELECT
    room.room_no INTO @room_no
FROM
    room
    INNER JOIN room_type_cost ON room.room_type =
room_type_cost.room_type
WHERE
    room.room_no IN (
        SELECT
            room_no
        FROM
            temp_room_nos
    )
    AND NOT EXISTS (
        SELECT
```

```
                1
            FROM
                reservation
            WHERE
                reservation.room_no = room.room_no
                AND reservation.checkin_date <= checkout_date
                AND reservation.checkout_date >= checkin_date
        )
ORDER BY
    room_type_cost.price ASC
LIMIT
    1;

-- select the cheapest available room of the desired type
-- Book the room by inserting a record into the reservation table
INSERT INTO
    reservation (room_no, bill_id, checkin_date, checkout_date)
VALUES
    (@room_no, @bill_id, checkin_date, checkout_date);

-- Remove the booked room from the temporary table
DELETE FROM
    temp_room_nos
WHERE
    room_no = @room_no;

END WHILE;

-- Update the base cost of the bill to reflect the total room cost
UPDATE
    bill
SET
    base_cost = (
        SELECT
            SUM(
                room_type_cost.price * DATEDIFF(checkout_date,
checkin_date)
            )
        FROM
            reservation
            INNER JOIN room ON reservation.room_no = room.room_no
            INNER JOIN room_type_cost ON room.room_type =
room_type_cost.room_type
        WHERE
            reservation.bill_id = @bill_id
    )
WHERE
    bill_id = @bill_id;

END;
```

- The stored procedure book_rooms is used to book one or more rooms for a customer during a specific period. The procedure takes in four input parameters: customer_id to identify the customer, room_nos as a string containing a comma-separated list of room numbers to be booked, checkin_date as the date of check-in, and checkout_date as the date of check-out.
- The procedure begins by creating a new bill for the customer by inserting a record into the bill table with base_cost and service_cost values set to 0. The LAST_INSERT_ID() function is then used to retrieve the ID of the newly created bill and store it in the variable @bill_id.
- Next, a new record is inserted into the pays table to link the customer with the bill. The customer ID and @bill_id are used as input parameters.
- The room_nos input parameter, which is a comma-separated string of room numbers, is then split into individual room numbers and stored in a temporary table called temp_room_nos. The temporary table is created if it doesn't already exist.
- The procedure then loops through each room number in the temporary table and books the room if it is available during the specified check-in and check-out dates. The loop continues until all the rooms in the temporary table have been booked.
- Within the loop, the procedure checks for the first available room of the desired type and cheapest price by joining the room and room_type_cost tables, filtering out any rooms that have already been reserved during the specified check-in and check-out dates. If a suitable room is found, a new record is inserted into the reservation table to book the room for the customer during the specified period. The booked room is then removed from the temporary table.
- Finally, the procedure calculates the total cost of the booked rooms by multiplying the number of days between checkin_date and checkout_date by the daily rate of each room type. The base_cost field of the bill table is then updated with the calculated total.
- In summary, the book_rooms stored procedure creates a new bill and links the customer to the bill, books the requested rooms, and calculates the total cost of the booking.

## Cancel the booking

→ Procedure to delete a reservation for a given booking ID and room number

```
CREATE PROCEDURE IF NOT EXISTS delete_reservation(IN booking_id INT, IN
room_no INT) BEGIN -- Declare variables to hold reservation cost,
check-in date, and check-out date
DECLARE reservation_cost INT;

DECLARE checkin_date DATE;

DECLARE checkout_date DATE;

-- Retrieve the reservation cost based on the room type associated with
the given room number
SELECT
    price INTO reservation_cost
FROM
    room_type_cost
WHERE
    room_type = (
        SELECT
            room_type
        FROM
            room r
        WHERE
            r.room_no = room_no
    );
```

```
-- Retrieve the check-in and check-out dates for the reservation
matching the given booking ID and room number
SELECT
    r.checkin_date,
    r.checkout_date INTO checkin_date,
    checkout_date
FROM
    reservation r
WHERE
    r.bill_id = booking_id
    AND r.room_no = room_no;

-- Delete the reservation matching the given booking ID and room number
DELETE FROM
    reservation r
WHERE
    r.bill_id = booking_id
    AND r.room_no = room_no;

-- Update the bill associated with the given booking ID to reflect the
change in base cost
UPDATE
    bill b
SET
    base_cost = base_cost - reservation_cost *(DATEDIFF(checkout_date,
checkin_date))
WHERE
    b.bill_id = bill_id;

END;
```

- This stored procedure is used to delete a reservation from the database. The procedure takes in two parameters, the booking ID and the room number of the reservation to be deleted.
- To delete the reservation, the procedure first declares three variables to hold the reservation cost, check-in date, and check-out date. The reservation cost is retrieved from the room_type_cost table based on the room type associated with the given room number. The check-in and check-out dates are retrieved from the reservation table for the reservation matching the given booking ID and room number.
- Once the necessary information is retrieved, the procedure deletes the reservation from the reservation table where the booking ID and room number match the given parameters.

The base cost of the bill associated with the given booking ID is then updated to reflect the change in cost due to the deleted reservation. The base cost is updated by subtracting the reservation cost times the number of days between the check-in and check-out dates of the deleted reservation.

- To summarize, this procedure takes in a booking ID and room number as parameters and deletes the reservation from the reservation table. It then updates the base cost of the bill associated with the given booking ID to reflect the change in cost due to the deleted reservation. Users can also delete some of the rooms of the booking and retain others.

## Allot Staff to room

→ The following stored procedure sets the staff assigned to a specific room.

```
-- It takes in two parameters: the staff ID and the room number.
CREATE PROCEDURE IF NOT EXISTS set_staff(IN p_staff_id INT, IN p_room_no
INT) BEGIN
INSERT INTO
    allots
VALUES
    (p_staff_id, p_room_no);

END;
```

This stored procedure sets the staff assigned to a specific room by inserting a new record into the 'allots' table, which links staff members to rooms. It takes two parameters: 'p_staff_id' to specify the ID of the staff member being assigned to the room, and 'p_room_no' to specify the room number. Once the record is inserted into the 'allots' table, the staff member will be considered assigned to the room. Multiple staff can be allotted for a single room.

## Admin Functionality: View Occupied rooms

```
SELECT r.room_no,
       CASE
```

```
           WHEN res.checkin_date <= CURRENT_DATE() AND res.checkout_date
>= CURRENT_DATE() THEN 0
           ELSE r.room_status+1
       END AS room_status,
       DATEDIFF(res.checkout_date,CURRENT_DATE())
FROM room r
LEFT JOIN (
    SELECT *
    FROM reservation
    WHERE checkin_date <= CURRENT_DATE() AND checkout_date >=
CURRENT_DATE()
) res ON r.room_no = res.room_no
```

This SQL query selects data from two tables: room and reservation, and uses a LEFT JOIN to join the two tables on the room_no column. The query retrieves information about the room status and duration of stay for each room in the hotel.

Here is a breakdown of the different parts of the query:

- SELECT r.room_no,: Selects the room_no column from the room table.
- CASE WHEN res.checkin_date <= CURRENT_DATE() AND res.checkout_date >= CURRENT_DATE() THEN 0 ELSE r.room_status+1 END AS room_status,: Creates a computed column called room_status that has a value of 0 if the current date falls within a reservation for the room, or r.room_status+1 otherwise. This is achieved using a CASE statement that checks whether the current date falls within the reservation dates (res.checkin_date <= CURRENT_DATE() AND res.checkout_date >= CURRENT_DATE()). If the condition is true, the room_status is set to 0. Otherwise, the room_status is set to the current room_status value (r.room_status) plus 1.
- DATEDIFF(res.checkout_date,CURRENT_DATE()): Calculates the duration of stay in days for the current reservation by subtracting the current date from the reservation checkout date (res.checkout_date).
- FROM room r: Specifies that the data should be retrieved from the room table.
- LEFT JOIN (SELECT * FROM reservation WHERE checkin_date <= CURRENT_DATE() AND checkout_date >= CURRENT_DATE()) res ON r.room_no = res.room_no: Joins the room table with the reservation table using a LEFT JOIN on the room_no column. This joins the room table with only those rows from the reservation table where the current date is between the check-in and check-out dates of the reservation (checkin_date <= CURRENT_DATE() AND checkout_date >= CURRENT_DATE()).

- Overall, this SQL query retrieves the room number, status and days of stay remaining for each room in the hotel, with a room_status value of 0 if the room is currently occupied 1 if it is under maintenance and 2 if it is available.

## Total amount generated by each type of room

```
SELECT room.room_type, SUM(bill.base_cost + bill.service_cost) AS
revenue
FROM room
INNER JOIN reservation ON room.room_no = reservation.room_no
INNER JOIN bill ON reservation.bill_id = bill.bill_id
GROUP BY room.room_type
```

- This SQL statement retrieves the total revenue generated by each room type in a hotel.
- The SELECT statement starts by specifying the columns to be returned, which are room.room_type and the calculated column SUM(bill.base_cost + bill.service_cost) AS revenue. room_type is the column from the room table which holds the type of the room, while base_cost and service_cost are columns from the bill table that hold the cost of the room reservation and any additional services used by the guest during their stay.
- The FROM clause specifies the tables involved in the query, which are room, reservation, and bill.
- The INNER JOIN keywords join the tables based on the specified condition, which in this case is room.room_no = reservation.room_no to join the room and reservation tables and reservation.bill_id = bill.bill_id to join the reservation and bill tables. These joins ensure that only rows with matching values in the specified columns are returned.
- The GROUP BY clause groups the results by the room_type column, which means that the results will be aggregated by room type, and the revenue column will show the total revenue generated for each room type.
- Overall, this SQL statement is useful in helping hotel managers track the revenue generated by different room types and make informed decisions regarding pricing, promotions, and resource allocation.

## Update tables when a room goes under maintenance using trigger

```
CREATE TRIGGER IF NOT EXISTS update_checkout_date_and_bill
AFTER UPDATE ON room
```

```
FOR EACH ROW
BEGIN
    IF NEW.room_status = 0 THEN
        -- Update checkout date of reservations for the room
        UPDATE reservation
        SET checkout_date = CURRENT_DATE()
        WHERE room_no = NEW.room_no
        AND checkin_date <= CURRENT_DATE() AND checkout_date >=
CURRENT_DATE();

        -- Update base cost of bills for the reservations of the room
        UPDATE bill b
        SET b.base_cost = (
            SELECT SUM(room_type_cost.price * DATEDIFF(checkout_date,
checkin_date))
            FROM reservation
            INNER JOIN room_type_cost ON NEW.room_type =
room_type_cost.room_type
            WHERE reservation.bill_id = b.bill_id
                AND reservation.room_no = NEW.room_no
                AND reservation.checkin_date <= CURRENT_DATE()
                AND reservation.checkout_date >= CURRENT_DATE()
        )
        WHERE b.bill_id IN (
            SELECT bill_id FROM reservation WHERE room_no = NEW.room_no
AND checkin_date <= CURRENT_DATE() AND checkout_date >= CURRENT_DATE()
        );

    END IF;
END;
```

This is a trigger that gets executed after an update on the "room" table. Specifically, it is triggered for each row that is updated.

The trigger checks if the new "room_status" value is 0. If it is, it updates the checkout date of reservations for that room to the current date. This means that any reservations that were ongoing in that room will now be marked as complete.

Next, the trigger updates the base cost of bills for the reservations of that room. It does this by selecting the sum of the product of the price of the room type and the difference between the checkout date and the check-in date for each reservation in that room. The price of the room type is obtained by joining the "room_type_cost" table. The bill is identified by joining the "reservation" and "bill" tables.

Finally, the trigger updates the bills that are associated with the reservations of that room. It does this by setting the "base_cost" value of each bill to the previously calculated sum of reservation costs. The bills to be updated are identified by selecting the bill IDs of reservations for that room that started before or on the current date and ended after or on the current date.

In summary, this trigger automatically updates the checkout date of reservations and the base cost of bills for a room when the room status is changed to 0, indicating that the room is no longer occupied.

## Use supplementary services

```
CREATE PROCEDURE IF NOT EXISTS new_order(
    IN p_room_no INT,
    IN p_service_name VARCHAR(255),
    IN p_quantity INT
) BEGIN -- Declare variables to store customer ID, bill ID, and service
cost
DECLARE v_customer_id INT;

DECLARE v_bill_id INT;

DECLARE v_service_cost INT;

-- Find the customer ID of the guest currently occupying the room
```

```
SELECT
    customer_id INTO v_customer_id
FROM
    pays
WHERE
    bill_id = (
        SELECT
            bill_id
        FROM
            reservation
        WHERE
            room_no = p_room_no
            AND checkin_date <= NOW()
            AND checkout_date > NOW()
    );

-- Get the bill ID for the customer's current bill
SELECT
    bill_id INTO v_bill_id
FROM
    pays
WHERE
    customer_id = v_customer_id
    AND bill_id = (
        SELECT
            bill_id
        FROM
            reservation
        WHERE
            room_no = p_room_no
            AND checkin_date <= NOW()
            AND checkout_date > NOW()
    );

-- Get the service cost for the requested service
SELECT
    service_cost INTO v_service_cost
FROM
    service_info
WHERE
    service_name = p_service_name;
```

```
-- Add the service cost to the current bill's service_cost field
UPDATE
    bill
SET
    service_cost = service_cost + (v_service_cost * p_quantity)
WHERE
    bill_id = v_bill_id;

-- Add the service to the room's list of provided services
INSERT INTO
    service
VALUES
    (NULL, p_service_name, p_quantity);

INSERT INTO
    demands
VALUES
    (p_room_no, LAST_INSERT_ID());

END;
```

- The CREATE PROCEDURE IF NOT EXISTS new_order statement creates a stored procedure named new_order in the database. This stored procedure takes in three parameters: p_room_no, p_service_name, and p_quantity.
- 
- Inside the stored procedure, several variables are declared to store the customer ID, bill ID, and service cost. The first SELECT statement retrieves the customer_id of the guest currently occupying the room identified by p_room_no. This is done by joining the pays and reservation tables and filtering the results to include only the reservation that matches the provided room_no and has a checkin_date that is less than or equal to the current date and a checkout_date that is greater than the current date.
- The next SELECT statement retrieves the bill_id associated with the customer identified by v_customer_id and the reservation matching the provided room_no, again using the pays and reservation tables and the same filter as before.
- The third SELECT statement retrieves the service_cost associated with the p_service_name parameter from the service_info table.
- The UPDATE statement adds the service cost multiplied by the p_quantity parameter to the service_cost field of the current bill, identified by v_bill_id.
- The last two INSERT statements add the service to the list of provided services for the room by inserting a new record into the service table with the provided p_service_name and p_quantity, and then inserting a new record into the demands table with the room

number (p_room_no) and the ID of the newly inserted service record (LAST_INSERT_ID()).

- Overall, this stored procedure is used to add a new order for a service to the current bill for a guest occupying a specific room. It does this by finding the customer and bill associated with the specified room, updating the bill's service_cost field with the cost of the requested service, and adding the service to the list of provided services for the room.

## Fetch Available rooms

```
SELECT room.room_no, room.room_type, room_type_cost.price
FROM room
INNER JOIN room_type_cost ON room.room_type = room_type_cost.room_type
WHERE room.room_status AND NOT EXISTS (
    SELECT *
    FROM reservation
    WHERE reservation.room_no = room.room_no AND
          reservation.checkin_date <= CURRENT_DATE() AND
          reservation.checkout_date >= CURRENT_DATE()
);
```

- This SQL statement retrieves information about available rooms along with their corresponding room types and prices. The query is composed of two main parts: a SELECT statement and a subquery.
- The SELECT statement begins by selecting the room number, room type, and price columns from the room and room_type_cost tables, respectively. The INNER JOIN clause is used to combine the two tables based on their room_type column, which serves as the common column between them.
- The WHERE clause is then used to filter the results based on two conditions. The first condition is room.room_status, which selects only those rooms that are currently available. This is because the room_status column indicates the current status of the room, with a value of 1 representing an occupied room and 0 representing an available room.
- The second condition is specified using a subquery that checks whether a room is currently reserved or not. The EXISTS operator is used to determine whether the subquery returns any rows or not. If the subquery returns no rows, it means that the room is currently not reserved and is available for booking.
- The subquery uses the reservation table to check if there is any reservation for the room on the current date. The reservation table contains information about all reservations made by guests, including the room number, check-in date, and check-out date. The

WHERE clause in the subquery checks whether the room number matches the room number of the current row and whether the reservation dates overlap with the current date.

- In summary, this SQL statement retrieves information about available rooms along with their corresponding room types and prices by selecting data from the room and room_type_cost tables and filtering the results based on the current room status and reservation status using a subquery.

## Fetch all rooms data

```
SELECT r.room_no, rtc.room_type,
CASE
    WHEN r.room_status = 0 THEN 'Maintenance'
    WHEN NOT EXISTS (
        SELECT *
        FROM reservation
        WHERE reservation.room_no = r.room_no AND
              reservation.checkin_date <= CURDATE() AND
              reservation.checkout_date >= CURDATE()
    ) THEN 'Available'
    ELSE 'Booked'
END AS is_available,
rtc.price,
GROUP_CONCAT(DISTINCT s.name ORDER BY s.staff_id ASC SEPARATOR ', ') AS
staff_allotted,
IFNULL(c.name, 'N/A') AS guest_name
FROM room r
JOIN room_type_cost rtc ON r.room_type = rtc.room_type
LEFT JOIN allots a ON r.room_no = a.room_no
LEFT JOIN staff s ON a.staff_id = s.staff_id
LEFT JOIN (
    SELECT res.room_no, res.bill_id, c.name
    FROM reservation res
    LEFT JOIN pays p ON res.bill_id = p.bill_id
    LEFT JOIN customer c ON p.customer_id = c.customer_id
    WHERE res.checkin_date <= CURDATE() AND (res.checkout_date >=
CURDATE() OR res.checkout_date IS NULL)
) res ON r.room_no = res.room_no
LEFT JOIN pays p ON res.bill_id = p.bill_id
LEFT JOIN customer c ON p.customer_id = c.customer_id
```

```
GROUP BY r.room_no, rtc.room_type, r.room_status, c.name;
```

- This query selects information about each room in a hotel, including its room number, room type, availability status, price, staff allotted, and guest name if the room is currently occupied.
- SELECT r.room_no, rtc.room_type, ... selects the room number, room type, availability status, price, staff allotted, and guest name if the room is currently occupied.
- CASE WHEN r.room_status = 0 THEN 'Maintenance' ... is a conditional statement that checks if the room_status field is 0. If it is, the room is marked as "Maintenance". If it is not, the query checks if there exists a reservation that occupies the room today. If there is no such reservation, the room is marked as "Available". If there is a reservation, the room is marked as "Booked".
- GROUP_CONCAT(DISTINCT s.name ORDER BY s.staff_id ASC SEPARATOR ', ') AS staff_allotted is a function that concatenates the names of staff members allotted to the room. DISTINCT is used to remove duplicates, ORDER BY sorts the names in ascending order of their staff IDs, and , is used as the separator between the names.
- IFNULL(c.name, 'N/A') AS guest_name selects the name of the guest occupying the room, or "N/A" if the room is not currently occupied.
- FROM room r selects the room table.
- JOIN room_type_cost rtc ON r.room_type = rtc.room_type joins the room table with the room_type_cost table on the room_type column.
- LEFT JOIN allots a ON r.room_no = a.room_no left joins the allots table on the room_no column.
- LEFT JOIN staff s ON a.staff_id = s.staff_id left joins the staff table on the staff_id column.
- LEFT JOIN (SELECT res.room_no, res.bill_id, c.name ... left joins a subquery that selects the room_no, bill_id, and name columns from the reservation, pays, and customer tables, respectively. This subquery finds the name of the guest who is currently occupying the room, if there is one.
- LEFT JOIN pays p ON res.bill_id = p.bill_id left joins the pays table on the bill_id column.
- LEFT JOIN customer c ON p.customer_id = c.customer_id left joins the customer table on the customer_id column.
- GROUP BY r.room_no, rtc.room_type, r.room_status, c.name groups the results by room number, room type, room status, and guest name.

**Apart from these we have included many other queries inside the code, which are explained using comments.**

## Screenshots for sql queries

The screenshots for all the SQL queries can be found at: Screenshots

## Screenshot/Documentation for frontend

The screenshots for all the SQL queries can be found at: Frontend Documentation

## Conclusion

Hotel Management System is a software solution that enables hotel staff to efficiently manage hotels. It uses the relational database model and provides functionalities to insert, update, delete, and retrieve data from the database tables. The system can be easily installed and modified as per the requirements of the hotel.