Project – Technical Report

BUAN 6320

Group 13

JSOM – UTD

Introduction

This Database Design Document outlines the design and implementation of a database that will store guests' reservations, services used by guests, and the billing records for the hotel named "Express Suites," which is run by the hotel chain "Stars Inc." This document is written along with developing the database system for the hotel, and it should be solely used by hotel staff, owners, and business partners to obtain necessary information. This document contains confidential information about "Express Suites."

Overview

The upcoming hotel "Express Suites" is a highly anticipated hotel by the public. Located in the heart of Dallas, it gives the guests easy access to the main attractions of the city. In addition to the comfortable stay and great location, guests can also enjoy services like spa, in-room dining, and other luxury amenities at the "Express Suites." The Star Inc. has been commissioned to create the database allowing the hotel to support these features and store guest reservation details and the amenities used by them in the database. This database will interact directly with the hotel system and the hotel staff and indirectly with the guests staying at the hotel.

Literature Review

Source: http://dx.doi.org/10.2991/icetis-13.2013.260

Research and Design of Hotel Management System Model: This paper is to make the hotel management system more optimized and systematic. The paper tries to optimize customer information management and hotel information management. We utilize these ideas by making a centralized reservation system that relates the guest to their room and billing as well as room service.

Assumptions

Assumptions

- Each room is allowed to have multiple reservations, but the reservations should not overlap with each other.
- Each reservation is associated with one and only one guest, but one reservation could include multiple rooms under the same guest's name.

Design Decisions

Key Factors Influencing Design

The database design was established through the creation of entity relationship model diagram. There are five entities defined in the database: A Guest entity represents the guests that have made reservations in the hotel; A Reservation entity includes all the details of the type of reservation made by the guest; a Billing_records entity represents the total bill the guest is charged (including the room service charges along with the stay); a Room entity represents the details of the room and the reservation info to which the room is assigned; a Room_service entity represents the details of the room services ordered by the guests staying in a particular room.

The database includes four views. The reservation_details view fetches data from the reservation, guest, room and billing_records table to provide a detailed summary of a reservation made by a guest. The room_availability view is used to fetch data regarding the availability of the rooms including the room number and the type of room that is available. The e. The total_billed_amount fetches the data of the total bill charged to guest including the reservation and the room_service provided to them. The guest_reservations view obtains the data regarding the guests and the reservations they made.

Functional Design Decisions

The database will usually interact with four main systems. A security and Authentication System will verify user credentials such as staff and manager's usernames and passwords before giving them access to the hotel database. A Property Management System will act as central hub for hotel operations and will perform functions like interpreting user requests, data insertion and modification and such other functions. A Channel Management System will be used to manage listing across many online hotel booking websites. A Payment Processing System will be used to verify the payment information and process the payments.

Database Managements System Decisions

For the implementation of the database, we have selected PostgreSQL 15. The PostgreSQL 15 DBMS is used because it is highly cost effective. It eliminates the licensing fees that usually come with other database management systems. Moreover, it has a strong foundation in relational database management, allowing to create a flexible data model with high efficiency by automating complex tasks. PostgreSQL's scalability and performance also allows it to handle large datasets and large magnitude of transactions effectively, allowing hotel operations to run smoothly.

Security and Privacy Design Decisions

The system will use layered security to ensure the data integrity and privacy. A login and password-based authentication system will be implemented to verify the credentials and ensure that only authorized personnel have access and can interact with data. Furthermore, a mediator system will be used to translate commands from non—SQL format into PL/SQL. This mechanism is implemented because end users like guests will not be allowed direct access to the database system.

The system will have three types of users. The end users such as guests will not be allowed direct access to the database, and they will interact through a user-friendly application. internal users such as staff will have controlled access to the database through the application layer. er. The database administrators will have full access to the database for the maintenance and troubleshooting tasks.

Statement of Work

Overview

This project oversees the design and creation of a database that will be used to track the information of the guests' reservation at a hotel along with the services they take from the hotel and their billing information. This database will serve as a data store and allow the staff and managers to access the details of guests, their reservations, payments methods and status and the services they received from the hotel. This database will allow the hotel to operate efficiently by allowing the staff and managers to manage the guests' reservations, billing history and the services provided to them properly. By properly maintaining the information and allowing the guests to enjoy their stay, the hotel will attract more customers in future.

Purpose and Objective

The purpose of the database is to manage all the guest information and hotel operations in the central repository. This database will allow staff and managers to access and maintain the guest profiles, including their contact details and reservation history. They will be able to easily manage the offline and online reservations and manage the room availability, check-in and check-out processes, and extra services provided to the guests. This database will also allow them to manage and record the financial transactions of all the guests.

Project Scope

This project's scope is limited to the design and implementation of the hotel database. The in-scope work includes writing down the requirements of the project, creating the entity relationship diagram for the hotel database, writing the data definition language (DDL) script to create and implement the database, writing the example of data manipulation language (DML) and examples of SQL scripts to highlight the usage of the database. All the out-of-scope work like implementing the server backend to access the data and having a user-friendly application to access and manage the room and reservation dates.

In-Scope Work

- Requirements definition document
- Entity relationship diagram
- Data Definition Language script
- Example of Data Manipulation Language script
- Example of SQL script
- Technical Report

Out-of-Scope Work

- Implementation of server backend
- User-friendly application to access the data such as reservation and guest data

Database Goals, Expectations, and Deliverables

The completed database will contain the metadata within the dedicated fields. Each field will also contain a primary key (a unique identifier) to prevent collisions of the guests with same name. The deliverables of the projects will include the entity relationship diagram of the database, the DDL scripts, examples of the DML scripts and SQL scripts to highlight the usage of the database and a technical report on the project.

Database Benefits

The primary benefit of this database is to provide data regarding the guests who are staying in the hotel and the reservations received by the hotel. This database will allow the staffs and managers of the hotel to increase their functionality by allowing them to easily fetch data regarding the reservations, room services provided to the guests, and the total amount that needs to be collected from the guests. This database will also allow the staff to keep track of check-in and check-out times of the guests.

Project Hardware and Software Tools

Diagram Tool

ER-Assistant 2.10, running on Windows 10

Office Productivity Tools

Microsoft Office 365

Database

PostgreSQL 15, using pgAdmin on Windows 10

Hardware and Software

Intel Core running on Window 10

Client Access Method

User friendly application for staff and managers to access the data. The application will run on Windows, macOS or Linux.

SQL Usage and Style

Adapted from Simon Holywell's SQL style guide, available at http://sqlstyle.guide/

General

- Use consistent and descriptive identifiers and names.
- Use white space and indentation
- Store time and date information in ISO 8601 format
- Use standard SQL functions instead of vendor-specific functions
- Keep code concise and avoid redundant SQL
- Add comments where necessary
- Avoid camelCase
- Avoid writing entity and attribute name in plurals
- Do not apply object-oriented design principles

Naming Conventions

- Keep the name unique and avoid using reserved keywords for name
- Name must begin with a letter and should not end with an underscore
- Avoid underscore between the two words instead of camelCase
- Avoid using abbreviations
- Keep the length of names to a maximum of 30 characters

Query syntax

- Use uppercase for reserved words
- Add spaces, indentation and line spacing to make the code easier to read
- Use BETWEEN instead of AND to combine multiple sentences
- Avoid using UNION clause where possible

Create syntax

- Avoid using vendor-specific data types
- Only use REAL or FLOAT types
- Default value should follow data type declaration
- The key should be unique

Project Management Methodology

In the context of our ongoing project, the initial database design phase is executed with a structured approach, reminiscent of elements from the waterfall model, aiming to fulfill the foundational database requirements delineated at the project's outset. As the project progresses, however, it becomes evident that a more flexible and iterative approach is necessary to accommodate evolving project dynamics and stakeholder needs. Thus, the database team seamlessly transitions to an agile project management methodology, mirroring the practices adopted by our software development counterparts. This shift enables us to foster close collaboration and synergy between the two teams, facilitating rapid adaptation to changing project requirements and ensuring alignment between database design and software development efforts. With this iterative approach, the database team continually refines the database design based on evolving project specifications and feedback from the software development team, ensuring that the database remains responsive to the project's evolving needs. By embracing an agile project

management methodology, we empower our team to deliver a database solution that not only meets but exceeds project expectations, driving the success of the overall endeavor.

Requirements Definition Document

Business Rules

1. A guest can make zero or more reservations.
2. Each reservation must be made by one guest.
3. A reservation can have zero or many rooms.
4. Each room must have one and only one reservation.
5. Each reservation is associated with one and only one billing record.
6. Each billing record is associated with one and only one reservation.
7. A room can receive zero or more room services.
8. A room service can be received by a room.

Entity and Attribute Description

Entity Name: GUEST

Entity Description: The entity includes all the details of the guests staying in the hotel.

Main Attributes of GUEST:

- Guest_id: (Primary Key) A unique identifier for the guests.
- First_name: The first name of the guest.

- Last_name: The last name of the guest.
- Phone_no: The contact number of the guest.
- Email: This attribute includes the email address of the guest.
- Guest_address: The permanent residence address of the guest.

Entity Name: RESERVATION

Entity Description: The entity includes all the reservations made from the guest to the hotel, for a specific room/rooms

Main Attributes of RESERVATION:

- Reservation_id (Primary Key): A unique identifier for the reservation.
- Guest_id_fk (Foreign Key, Primary Key): The ID of the guest staying in the hotel.
- Room_id_fk (Foreign Key): The ID of the room booked for the reservation.
- Billing_id_fk (Foreign Key): The ID of the bill issued for the reservation
- Check-in_date: The date when the guest checks in to stay at the hotel.
- Check-out_date: The date when the guest checks out of the hotel.
- Number_of_guests: The number of guests staying in the hotel under a particular reservation.
- Status: The status of the reservation (confirmed, cancelled, pending).

Entity Name: BILLING_RECORDS

Entity Description: The entity contains the details related to the total amount charged to the guest.

Main Attributes of BILLING_RECORDS:

- Billing_id(Primary Key): A unique identifier of each billing record.
- Reservation_id_fk(Foreign Key): The unique number assigned to each reservation made.
- Total_amount: The total amount charged to the guests for thier stay in the hotel and the services they took.
- Payment_method: The methods through which the guest makes the payment.
- Billing_date: The date the bill is generated and paid.
- Billing_address: The address of the guest on which the bill is generated.

Entity Name: ROOM

Entity Description: The entity includes the room details of the room assigned to the guest reservation.

Main Attributes OF ROOM:

- Room_id (Primary Key): A unique identifier for the room.
- Room_no: The number of the room in the hotel.

- Room_type: The type of room (such as single, double, etc) assigned to the guest.
- Capacity: The total number of guests that can stay in that room.
- Availability: The availability status of the specific room to be reserved.
- Reservation_id_fk (Foreign Key): The unique number assigned to each reservation made.

Entity Name: ROOM_SERVICE

Entity Description: The entity contains the details of the room services provided to each room.

Main Attributes of ROOM_SERVICE:

- Service_id (Primary Key): A unique identifier of the service provied to the room.
- Service_name: The name of the service provided to the room.
- Service cost: The cost of the room service provided to the room.
- Service_payment_status: The status of the service payment, whether the payment is made or pending.
- Service_description: The details of the service provided to the room.
- Room_id_fk (Foreign Key): The unique number given to each room present in the hotel.

Relationship and Cardinality Description

Relationship: makes between GUEST and RESERVATION

Cardinality: 1:M between GUEST and RESERVATION

Business rule: Each guest can make zero or more reservations in the hotel. Each reservation can be made by one guest.

Relationship: associated with between RESERVATION and BILLING_RECORDS

Cardinality: 1:1 between RESERVATION and BILLING_RECORDS

Business rule: Each reservation is associated with one and only one billing record. Each billing record is associated with one and only one reservation.

Relationship: books between RESERVATION and ROOM

Cardinality: 1:M between RESERVATION and ROOM

Business rule: Each reservation can book zero or many rooms. Each room can be booked by one and only one reservation.

Relationship: receives between ROOM and ROOM_SERVICE

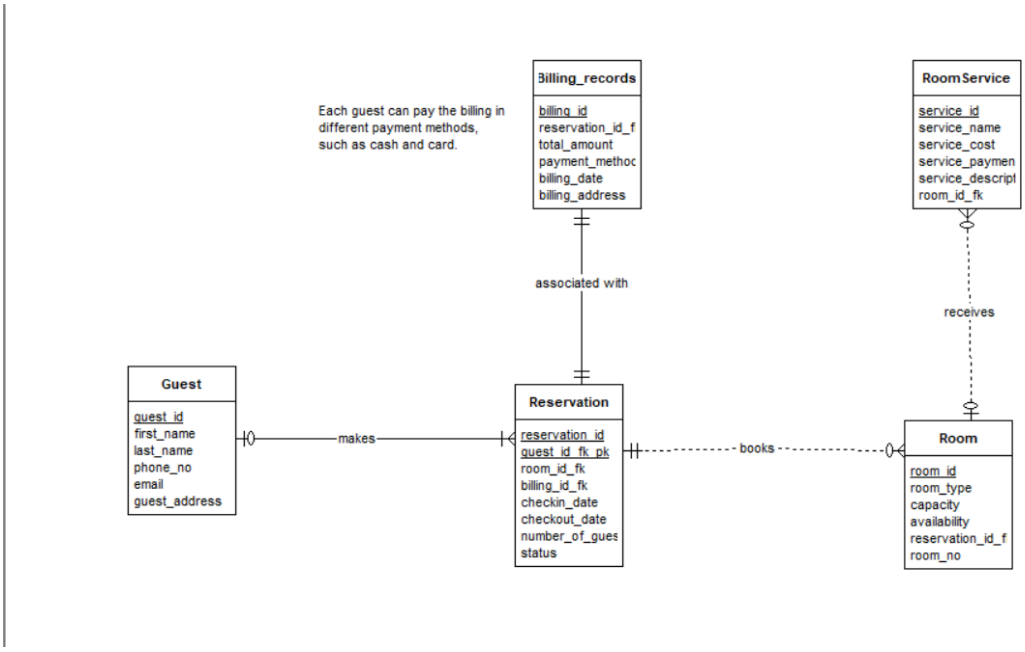
Cardinality: 1:M between ROOM and ROOM_SERVICE

Business rule: Each room can receive zero or many room services. Each room service can be received by zero or one room.

Detailed Database Design

Entity Relationship Diagram

(Note: It was not possible to resize the boxes of entities in the ER assistant due to its limitation. So, there is a possibility of entity names or attributes being truncated.)



DDL Source Code

set search_path to hotel_db;

/*DROP statements to clean up objects from previous run*/

--Triggers

DROP TRIGGER IF EXISTS set_billing_date ON billing_records;

DROP TRIGGER IF EXISTS update_room_availability ON reservation;

```
DROP TRIGGER IF EXISTS generate_reservation_id ON reservation;
```

--Sequences

```
DROP SEQUENCE IF EXISTS reservation_id_seq;

DROP SEQUENCE IF EXISTS guest_id_seq;

DROP SEQUENCE IF EXISTS billing_id_seq;

DROP SEQUENCE IF EXISTS room_id_seq;

DROP SEQUENCE IF EXISTS room_service_id_seq;
```

--Views

```
DROP VIEW IF EXISTS reservation_details;

DROP VIEW IF EXISTS room_availability;

DROP VIEW IF EXISTS total_billed_amount;

DROP VIEW IF EXISTS guest_reservations;
```

--Indices

```
DROP INDEX IF EXISTS idx_guest_guest_id;

DROP INDEX IF EXISTS idx_guest_last_name;
```

```
DROP INDEX IF EXISTS idx_reservation_reservation_id;

DROP INDEX IF EXISTS idx_reservation_check_in_date;

DROP INDEX IF EXISTS idx_reservation_check_out_date;

DROP INDEX IF EXISTS idx_reservation_guest_id_fk_pk;


DROP INDEX IF EXISTS idx_billing_records_payment_method;

DROP INDEX IF EXISTS idx_billing_records_billing_date;

DROP INDEX IF EXISTS idx_billing_records_reservation_id_fk;


DROP INDEX IF EXISTS idx_room_room_id;

DROP INDEX IF EXISTS idx_room_room_no;

DROP INDEX IF EXISTS idx_room_availability;

DROP INDEX IF EXISTS idx_room_reservation_id_fk;


DROP INDEX IF EXISTS idx_room_service_service_name;

DROP INDEX IF EXISTS idx_room_service_service_payment_status;

DROP INDEX IF EXISTS idx_room_service_room_id_fk;
```

--Tables

```
DROP TABLE guest CASCADE;

DROP TABLE reservation CASCADE;

DROP TABLE billing_records CASCADE;

DROP TABLE room CASCADE;

DROP TABLE room_service CASCADE;
```

/*Create tables based on entities*/

```
create table guest (

guest_id int primary key,

first_name varchar(255),

last_name varchar(255),

phone_no varchar(20),

email varchar(255),

guest_address varchar(255)

);
```

```
create table reservation (
reservation_id int primary key,
guest_id int,
room_id int,
billing_id int,
check_in_date date,
check_out_date date,
number_of_guests int,
status varchar(50),
foreign key (guest_id) references guest(guest_id)
);

create table billing_records (
    billing_id serial primary key,
    reservation_id int,
    total_amount numeric(10, 2),
    payment_method varchar(50),
    billing_date date,
```

```
    billing_address varchar(255),
    foreign key (reservation_id) references reservation(reservation_id)
);

create table room (
    room_id serial primary key,
    room_no varchar(20),
    room_type varchar(50),
    capacity int,
    availability varchar(20),
    reservation_id int,
    foreign key (reservation_id) references reservation(reservation_id)
);

create table room_service (
    service_id serial primary key,
    service_name varchar(100),
    service_cost numeric(10, 2),
```

```
    service_payment_status varchar(20),
    service_description text,
    room_id int,
    foreign key (room_id) references room(room_id)
);
```

/*Create indices for natural keys, foreign keys, and frequently- queried columns*/

```
--Guest
CREATE INDEX idx_guest_guest_id ON guest(guest_id);
CREATE INDEX idx_guest_last_name ON guest(last_name);

--Reservation
CREATE INDEX idx_reservation_reservation_id ON reservation(reservation_id);
CREATE INDEX idx_reservation_check_in_date ON reservation(check_in_date);
CREATE INDEX idx_reservation_check_out_date ON reservation(check_out_date);
CREATE INDEX idx_reservation_guest_id_fk_pk ON reservation(guest_id);

--Billing_records
```

```
CREATE INDEX idx_billing_records_payment_method ON billing_records(payment_method);

CREATE INDEX idx_billing_records_billing_date ON billing_records(billing_date);

CREATE INDEX idx_billing_records_reservation_id_fk ON billing_records(reservation_id);


--Room

CREATE INDEX idx_room_room_id ON room(room_id);

CREATE INDEX idx_room_room_no ON room(room_no);

CREATE INDEX idx_room_availability ON room(availability);

CREATE INDEX idx_room_reservation_id_fk ON room(reservation_id);


--Room_service

CREATE INDEX idx_room_service_service_name ON room_service(service_name);

CREATE INDEX idx_room_service_service_payment_status ON
room_service(service_payment_status);

CREATE INDEX idx_room_service_room_id_fk ON room_service(room_id);
```

/*Alter table by adding audit tables*/

```
ALTER TABLE guest

ADD COLUMN created_by    VARCHAR(30),
```

```
ADD COLUMN date_created  DATE,

ADD COLUMN modified_by   VARCHAR(30),

ADD COLUMN date_modified DATE;


ALTER TABLE reservation

ADD COLUMN created_by    VARCHAR(30),

ADD COLUMN date_created  DATE,

ADD COLUMN modified_by   VARCHAR(30),

ADD COLUMN date_modified DATE;


ALTER TABLE billing_records

ADD COLUMN created_by    VARCHAR(30),

ADD COLUMN date_created  DATE,

ADD COLUMN modified_by   VARCHAR(30),

ADD COLUMN date_modified DATE;


ALTER TABLE room

ADD COLUMN created_by    VARCHAR(30),
```

```
ADD COLUMN date_created  DATE,

ADD COLUMN modified_by   VARCHAR(30),

ADD COLUMN date_modified DATE;



ALTER TABLE room_service

ADD COLUMN created_by    VARCHAR(30),

ADD COLUMN date_created  DATE,

ADD COLUMN modified_by   VARCHAR(30),

ADD COLUMN date_modified DATE;
```

-- Views

--Reservation details view

```
CREATE OR REPLACE VIEW reservation_details AS

SELECT r.reservation_id, g.first_name || ' ' || g.last_name AS guest_name, r.check_in_date,

r.check_out_date,

    rm.room_no, rm.room_type, rm.capacity, rm.availability,

    b.total_amount, b.payment_method, b.billing_date

FROM reservation r

JOIN guest g ON r.guest_id = g.guest_id
```

```
JOIN room rm ON r.room_id = rm.room_id

JOIN billing_records b ON r.billing_id = b.billing_id;



-- Room Availability View

CREATE OR REPLACE VIEW room_availability AS

SELECT room_no, room_type, availability

FROM room;

 -- Create view for total billed amount including room service

CREATE VIEW total_billed_amount AS (

SELECT

r.reservation_id,

r.check_in_date,

r.check_out_date,

r.number_of_guests,

b.total_amount AS room_bill,

COALESCE(SUM(rs.service_cost), 0) AS room_service_bill,

b.total_amount + COALESCE(SUM(rs.service_cost), 0) AS total_billed_amount

FROM
```

```
reservation r

LEFT JOIN

billing_records b ON r.reservation_id = b.reservation_id

LEFT JOIN

room_service rs ON r.room_id = rs.room_id

GROUP BY

r.reservation_id, r.check_in_date, r.check_out_date, r.number_of_guests, b.total_amount

);



-- Create view for guest reservations

CREATE VIEW guest_reservations AS

SELECT

    g.first_name || ' ' || g.last_name AS guest_name,

    r.check_in_date,

    r.check_out_date,

    ro.room_no

FROM

    guest g
```

```
JOIN
    reservation r ON g.guest_id = r.guest_id
JOIN
    room ro ON r.room_id = ro.room_id;
```

/* CREATE ITEMS */

-- Sequences

```
CREATE SEQUENCE guest_id_seq
  START WITH 1001
  INCREMENT BY 1
  MAXVALUE 9999
  CYCLE;

CREATE SEQUENCE reservation_id_seq
  START WITH 1
  INCREMENT BY 1
  MAXVALUE 9999
```

```
  CYCLE;


CREATE SEQUENCE billing_id_seq
  START WITH 1
  INCREMENT BY 1
  MAXVALUE 9999
  CYCLE;


CREATE SEQUENCE room_id_seq
  START WITH 1
  INCREMENT BY 1
  MAXVALUE 9999
  CYCLE;


CREATE SEQUENCE room_service_id_seq
  START WITH 1
  INCREMENT BY 1
  MAXVALUE 9999
```

```
  CYCLE;


-- Triggers

-- Create Trigger for unique reservation_id

CREATE OR REPLACE FUNCTION generate_reservation_id()

RETURNS TRIGGER AS $$

BEGIN

  SELECT nextval('reservation_id_seq') INTO NEW.reservation_id;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;


CREATE TRIGGER generate_reservation_id

BEFORE INSERT ON reservation

FOR EACH ROW

EXECUTE FUNCTION generate_reservation_id();


-- Create Trigger to set billing_date to current date
```

```
CREATE OR REPLACE FUNCTION set_billing_date()

RETURNS TRIGGER AS $$

BEGIN

  NEW.billing_date := CURRENT_DATE;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE TRIGGER set_billing_date

BEFORE INSERT ON billing_records

FOR EACH ROW

EXECUTE FUNCTION set_billing_date();
```

-- Create Trigger to update room availability

```
CREATE OR REPLACE FUNCTION update_room_availability()

RETURNS TRIGGER AS $$

BEGIN

  IF TG_OP = 'INSERT' THEN
```

```
    -- Reservation is confirmed, set room availability to occupied
    UPDATE room
    SET availability = 'occupied'
    WHERE room_id = NEW.room_id;
  ELSIF TG_OP = 'DELETE' THEN
    -- Reservation is canceled, set room availability to unoccupied
    UPDATE room
    SET availability = 'unoccupied'
    WHERE room_id = OLD.room_id;
  END IF;
  RETURN NULL;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER update_room_availability
AFTER INSERT OR DELETE ON reservation
FOR EACH ROW
EXECUTE FUNCTION update_room_availability();
```

DML and Query Source Code

```
set search_path to hotel_db;


INSERT INTO guest (guest_id, first_name, last_name, phone_no, email, guest_address)
VALUES
(1001, 'Liam', 'Smith', '979-123-4567', 'liam.smith@gmail.com', '123 Main St, Dallas, TX'),
(1002, 'Stephanie', 'Garcia', '848-456-7890', 'stephanie.garcia@yahoo.com', '456 Elm St, Dallas, TX'),
(1003, 'John', 'Davis', '979-111-2345', 'john.davis@gmail.com', '789 Pine St, Houston, TX'),
(1004, 'Lily', 'Chavez', '878-222-4848', 'lilychavez123@gmail.com', '101 Oak St, Los Angeles, CA'),
(1005, 'Will', 'Davis', '979-234-5678', 'willdavis5678@gmail.com', '202 Maple St, New York, NY');


-- DML for inserting data into reservation Entity
INSERT INTO reservation (reservation_id, guest_id, room_id, billing_id, check_in_date, check_out_date, number_of_guests, status) VALUES
(1,1001, 1, 1111, '2023-02-15', '2023-02-17', 2, 'confirmed'),
```

(2,1002, 2, 1112, '2023-08-18', '2023-08-22', 1, 'cancelled'),

(3,1003, 3, 1113, '2024-05-25', '2024-05-26', 3, 'confirmed'),

(4,1004, 4, 1114, '2024-12-01', '2024-12-02', 4, 'confirmed'),

(5, 1005, 5, 1115, '2025-01-08', '2025-01-10', 2, 'cancelled');

-- DML for inserting data into billing_records Entity

INSERT INTO billing_records (reservation_id, total_amount, payment_method, billing_date, billing_address) VALUES

(1, 500.00, 'Credit Card', '2023-02-15', '123 Main St, Dallas, TX'),

(2, 600.00, 'Debit Card', '2023-08-18', '110 Pine St, Houston, TX'),

(3, 450.00, 'Cash', '2024-05-25', '432 Damascus St, Austin, TX'),

(4, 500.00, 'Credit Card', '2024-12-01', '101 Oak St, Los Angeles, CA'),

(5, 350.00, 'Debit Card', '2025-01-08', '789 Greenwood St, Houston, TX');

-- DML for inserting data into room Entity

INSERT INTO room (room_id, room_no, room_type, capacity, availability, reservation_id) VALUES

(1, 101, 'Single', 2, 'Unoccupied', 1),

(2, 102, 'Single', 2, 'Occupied', 2),

(3, 201, 'Double', 4, 'Unoccupied', 3),

(4, 202, 'Double', 4, 'Unoccupied', 4),

(5, 203, 'Suite', 2, 'Occupied', 5);

-- DML for inserting data into room_service Entity

INSERT INTO room_service (service_name, service_cost, service_payment_status, service_description, room_id) VALUES

('Laundry', 15.00, 'Paid', 'Same-day laundry service', 1),

('Breakfast', 20.00, 'Paid', 'Continental breakfast delivered to room', 2),

('Laundry', 15.00, 'Unpaid', 'Same-day laundry service', 3),

('Dinner', 25.00, 'Unpaid', 'Three-course dinner served in room', 4),

('Spa', 50.00, 'Paid', 'In-room massage and spa service', 5);

COMMIT;

--Query 1: Select all columns and all rows from one table

SELECT * FROM guest;

--Query 2: Select five columns and all rows from one table

```
SELECT guest_id, first_name, last_name, phone_no, email FROM guest;
```

--Query 3: Select all columns from all rows from one view

```
SELECT * FROM total_billed_amount;
```

--Query 4: Using a join on 2 tables, select all columns and all rows from the tables without the use of a Cartesian product

```
SELECT * FROM reservation
JOIN guest ON reservation.guest_id = guest.guest_id;
```

--Query 5: Select and order data retrieved from one table

```
SELECT * FROM room ORDER BY room_type;
```

--Query 6: Using a join on 3 tables, select 5 columns from the 3 tables. Use syntax that would limit the output to 3 rows

```
SELECT r.*, g.first_name, g.last_name, rm.room_type
FROM reservation r
JOIN guest g ON r.guest_id = g.guest_id
JOIN room rm ON r.room_id = rm.room_id
```

```
LIMIT 3;
```

--Query 7: Select distinct rows using joins on 3 tables

```
SELECT DISTINCT r.*, g.first_name, g.last_name, rm.room_type

FROM reservation r

JOIN guest g ON r.guest_id = g.guest_id

JOIN room rm ON r.room_id = rm.room_id;
```

--Query 8: Use GROUP BY and HAVING in a select statement using one or more tables

```
SELECT room_type, COUNT(*)

FROM room

GROUP BY room_type

HAVING COUNT(*) > 1;
```

--Query 9: Use IN clause to select data from one or more tables

```
SELECT * FROM guest WHERE guest_id IN (SELECT guest_id FROM reservation);
```

--Query 10: Select length of one column from one table (use LENGTH function)

```
SELECT LENGTH(first_name) AS name_length FROM guest;
```

--Query 11: Delete one record from one table. Use select statements to demonstrate the table contents before and after the DELETE statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed

```
-- Before DELETE

SELECT * FROM room_service;

-- Delete

DELETE FROM room_service

WHERE service_name = 'Breakfast';

-- After DELETE

SELECT * FROM room_service;

-- Rollback

ROLLBACK;
```

--Query 12: Update one record from one table. Use select statements to demonstrate the table contents before and after the UPDATE statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed

```
-- Before UPDATE

SELECT * FROM guest WHERE guest_id = 1002;

-- Update

UPDATE guest SET first_name = 'John' WHERE guest_id = 1002;

-- After UPDATE

SELECT * FROM guest WHERE guest_id = 1002;

-- Rollback

ROLLBACK;
```

--Query 13 (Advanced Queries): Retrieve the total revenue generated from room service for each guest

```
SELECT g.guest_id, g.first_name, g.last_name, COALESCE(SUM(rs.service_cost), 0) AS
total_service_revenue

FROM guest g

LEFT JOIN reservation r ON g.guest_id = r.guest_id

LEFT JOIN room_service rs ON r.room_id = rs.room_id

GROUP BY g.guest_id, g.first_name, g.last_name;
```

--Query 14 (Advanced Queries): Retrieve the top 3 most popular room types based on the number of reservations

```
SELECT room_type, COUNT(*) AS reservations_count

FROM reservation r

JOIN room rm ON r.room_id = rm.room_id

GROUP BY room_type

ORDER BY reservations_count DESC

LIMIT 3;
```

DDL, DML, and Query Output

- **DDL Output**

NOTICE: drop cascades to constraint reservation_guest_id_fkey on table reservation

NOTICE: drop cascades to 2 other objects NOTICE: drop cascades to constraint room_service_room_id_fkey on table room_service

CREATE TRIGGER

Query returned successfully in 115 msec.

- **DML Output**

WARNING: there is no transaction in progress

COMMIT

Query returned successfully in 37 msec.

- **Query Output**

Query 1: Select all columns and all rows from one table

"guest_id" "first_name" "last_name" "phone_no" "email" "guest_address" "created_by" "date_created" "modified_by" "date_modified"

1001 "Liam" "Smith""979-123-4567" "liam.smith@gmail.com" "123 Main St, Dallas, TX"

1002 "Stephanie" "Garcia" "848-456-7890" "stephanie.garcia@yahoo.com" "456 Elm St, Dallas, TX"

1003 "John" "Davis""979-111-2345" "john.davis@gmail.com" "789 Pine St, Houston, TX"

1004 "Lily" "Chavez" "878-222-4848" "lilychavez123@gmail.com" "101 Oak St, Los Angeles, CA"

1005 "Will" "Davis""979-234-5678" "willdavis5678@gmail.com" "202 Maple St, New York, NY"

Query 2: Select five columns and all rows from one table

"guest_id" "first_name" "last_name" "phone_no" "email"

1001 "Liam" "Smith" "979-123-4567" "liam.smith@gmail.com"

1002 "Stephanie" "Garcia" "848-456-7890"

"stephanie.garcia@yahoo.com"

1003 "John" "Davis" "979-111-2345" "john.davis@gmail.com"

1004 "Lily" "Chavez" "878-222-4848" "lilychavez123@gmail.com"

1005 "Will" "Davis" "979-234-5678" "willdavis5678@gmail.com"

Query 3: Select all columns from all rows from one view

"reservation_id"	"check_in_date"	"check_out_date"	"number_of_guests"	"room_bill"	"room_service_bill"	"total_billed_amount"
4	"2024-12-01"	"2024-12-02"	4	500.00	25.00	525.00
2	"2023-08-18"	"2023-08-22"	1	600.00	20.00	620.00
3	"2024-05-25"	"2024-05-26"	3	450.00	15.00	465.00
1	"2023-02-15"	"2023-02-17"	2	500.00	15.00	515.00
5	"2025-01-08"	"2025-01-10"	2	350.00	50.00	400.00

Query 4 : Using a join on 2 tables, select all columns and all rows from the tables without the use of a Cartesian product (5 points)

"reservation_id"	"guest_id"	"room_id"	"billing_id"	"check_in_date"	"check_out_date"	"number_of_guests"	"status"	"created_by"	"date_created"	"modified_by"	"date_modified"	"guest_id-2"	"first_name"	"last_name"	"phone_no"	"email"	"guest_address"	"created_by-2"	"date_created-2"	"modified_by-2"	"date_modified-2"
1	1001	1	1111	"2023-02-15"	"2023-02-17"	2	"confirmed"	1001	"Liam"	"Smith"	"979-123-4567"	"liam.smith@gmail.com"	"123 Main St, Dallas, TX"								
2	1002	2	1112	"2023-08-18"	"2023-08-22"	1	"cancelled"	1002	"Stephanie"	"Garcia"	"848-456-7890"	"stephanie.garcia@yahoo.com"	"456 Elm St, Dallas, TX"								
3	1003	3	1113	"2024-05-25"	"2024-05-26"	3	"confirmed"	1003	"John"	"Davis"	"979-111-2345"	"john.davis@gmail.com"	"789 Pine St, Houston, TX"								
4	1004	4	1114	"2024-12-01"	"2024-12-02"	4	"confirmed"	1004	"Lily"	"Chavez"	"878-222-4848"	"lilychavez123@gmail.com"	"101 Oak St, Los Angeles, CA"								
5	1005	5	1115	"2025-01-08"	"2025-01-10"	2	"cancelled"	1005	"Will"	"Davis"	"979-234-5678"	"willdavis5678@gmail.com"	"202 Maple St, New York, NY"								

Query 5: Select and order data retrieved from one table

"room_id"	"room_no"	"room_type"	"capacity"	"availability"	"reservation_id"	"created_by"	"date_created"	"modified_by"	"date_modified"

3	"201"	"Double"	4	"Unoccupied"	3
4	"202"	"Double"	4	"Unoccupied"	4
1	"101"	"Single"	2	"Unoccupied"	1
2	"102"	"Single"	2	"Occupied"	2
5	"203"	"Suite"	2	"Occupied"	5

Query 6: Using a join on 3 tables, select 5 columns from the 3 tables. Use syntax that would limit the output to 3 rows

"reservation_id"	"guest_id"	"room_id"	"billing_id"	"check_in_date"	"check_out_date"	"number_of_guests"	"status"	"created_by"	"date_created"	"modified_by"	"date_modified"	"first_name"	"last_name"	"room_type"
1	1001	1	1111	"2023-02-15"	"2023-02-17"	2	"confirmed"	"Liam"	"Smith"	"Single"				
2	1002	2	1112	"2023-08-18"	"2023-08-22"	1	"cancelled"	"Stephanie"	"Garcia"	"Single"				
3	1003	3	1113	"2024-05-25"	"2024-05-26"	3	"confirmed"	"John"	"Davis"	"Double"				

Query 7: Select distinct rows using joins on 3 tables

"reservation_id"	"guest_id"	"room_id"	"billing_id"	"check_in_date"	"check_out_date"	"number_of_guests"	"status"	"created_by"	"date_created"	"modified_by"	"date_modified"	"first_name"	"last_name"	"room_type"
4	1004	4	1114	"2024-12-01"	"2024-12-02"	4	"confirmed"	"Lily"	"Chavez"	"Double"				
3	1003	3	1113	"2024-05-25"	"2024-05-26"	3	"confirmed"	"John"	"Davis"	"Double"				
2	1002	2	1112	"2023-08-18"	"2023-08-22"	1	"cancelled"	"Stephanie"	"Garcia"	"Single"				
5	1005	5	1115	"2025-01-08"	"2025-01-10"	2	"cancelled"	"Will"	"Davis"	"Suite"				
1	1001	1	1111	"2023-02-15"	"2023-02-17"	2	"confirmed"	"Liam"	"Smith"	"Single"				

Query 8: Use GROUP BY and HAVING in a select statement using one or more tables

"room_type" "count"

"Single" 2

"Double" 2

Query 9: Use IN clause to select data from one or more tables

"guest_id" "first_name" "last_name" "phone_no" "email" "guest_address" "created_by" "date_created" "modified_by" "date_modified"

1001 "Liam" "Smith" "979-123-4567" "liam.smith@gmail.com" "123 Main St, Dallas, TX"

1002 "Stephanie" "Garcia" "848-456-7890" "stephanie.garcia@yahoo.com" "456 Elm St, Dallas, TX"

1003 "John" "Davis" "979-111-2345" "john.davis@gmail.com" "789 Pine St, Houston, TX"

1004 "Lily" "Chavez" "878-222-4848" "lilychavez123@gmail.com" "101 Oak St, Los Angeles, CA"

1005 "Will" "Davis" "979-234-5678" "willdavis5678@gmail.com" "202 Maple St, New York, NY"

Query 10: Select length of one column from one table (use LENGTH function)

"name_length"

4

9

4

4

4

Query 11 Delete one record from one table. Use select statements to demonstrate the table contents before and after the DELETE statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed

"service_id" "service_name" "service_cost" "service_payment_status" "service_description" "room_id" "created_by" "date_created" "modified_by" "date_modified"

1 "Laundry" 15.00 "Paid" "Same-day laundry service" 1

3 "Laundry" 15.00 "Unpaid" "Same-day laundry service" 3

4 "Dinner" 25.00 "Unpaid" "Three-course dinner served in room" 4

5 "Spa" 50.00 "Paid" "In-room massage and spa service" 5

Query 12: Update one record from one table. Use select statements to demonstrate the table contents before and after the UPDATE statement. Make sure you use ROLLBACK afterwards so that the data will not be physically removed

"guest_id" "first_name" "last_name" "phone_no" "email" "guest_address" "created_by" "date_created" "modified_by" "date_modified"

1002 "John" "Garcia" "848-456-7890" "stephanie.garcia@yahoo.com" "456 Elm St, Dallas, TX"

Query 13 (Advanced Query)

"guest_id" "first_name" "last_name" "total_service_revenue"

1002 "Stephanie" "Garcia" 20.00

1005 "Will" "Davis" 50.00

1004 "Lily" "Chavez" 25.00

1003 "John" "Davis" 15.00

1001 "Liam" "Smith" 15.00

Query 14 (Advanced Query)

"room_type" "reservations_count"

"Single" 2

"Double" 2

"Suite" 1

Database Administration and Monitoring

Roles and Responsibilities

1. **System Administrator:** The database administrator shall install, configure and maintain the database management system (BDMS) and related infrastructure, along with ensuring the availability, reliability, and performance of database systems. The SA shall monitor system health, troubleshoot issues, perform system maintenance tasks, and collaborate with developers and analysts to address system-level requirements and challenges.
2. **Database administrator:** The database administrator has the responsibility to install, configure, and upgrade database software and tools, shall manage the database schema, tables, users, roles, and permissions. The DA shall implement backup and recovery strategies to protect data and ensure business continuity and collaborate with system administrators, developers, and analysts to address database-related requirements and issues
3. **Data Analyst:** The role of the data analyst is to write SQL queries and scripts to retrieve, manipulate, and analyze data stored in the database, it shall create reports, dashboards, and visualization to present data analysis findings to stakeholders.
4. **Security Administrator:** The system security administrator and other security staff shall implement security measures such as access controls to protect sensitive data, it shall monitor the database activity and user privileges to detect and mitigate security threads.

System Information

DBMS: PostgreSQL 15

System requirements:

- Minimum 4 cores per server
- Minimum 16 GB RAM per server
- Minimum 50 GB free hard disk space per server (pre-installation)
- Additional storage spaces for data and backups (for data retention and future growth)

Performance Monitoring and Database Efficiency

The database administration team and the system administration team will both be responsible for monitoring and maintaining the perdormance of the database. The database administration team will be responsible for maintaing the performance and efficiency of the database and the system administration team will be responsible for monitoring and maintaining the software and server of the database. The DBMS shall be monitored for its performance by both the teams.

Data Formats

The database uses several data formats to ensure that the data is stored and retrieved efficiently. The database uses standard datatypes like strings, integers, numeric, and dates . These data are stored in the form of raw binary data. The database will store this raw binary data and it will be transferred by the DBMS.

Backup and Recovery

The database has employed a robust backup and recovery plan to ensure the data integrity and reduce the downtimes in unforeseen circumstances. Due to frequent updates made in the system

from the new reservations, room services requested and changes in the guest information and such other circumstances, delta backups will be performed once daily and. There will also be full backups performed weekly at 1 AM CST every Monday.

References

Yang, J. (2013). Research and design of Hotel Management System Model. *Proceedings of the 2013 the International Conference on Education Technology and Information Systems*. https://doi.org/10.2991/icetis-13.2013.260