# Database System – S232

# Assignment 2 Report

----o0o-----

*Class: CC02*

*Group No: 5 (your group no. in Registration)*

1- Dương Ngọc Quang Huy - 2052489

2- Nguyễn Nam Kha – 2052515

3- Phan Trần Ý Nhi - 2252583

| **Content** | **Student No.**[1] | **Note**[2] |
|---|---|---|
| **PART 1: CREATE DATABASE (3.5p)** | | |
| I. Create table (2p)<br>- Table 1-6<br>- Table 7-10<br>- Table 11-14 | <br>1<br>3<br>2 | |
| II. Insert (1.5p)<br>- Table 1-6<br>- Table 7-10<br>- Table 11-14 | <br>1<br>3<br>2 | |
| **PART 2: STORE PROCEDURE, FUNCTION, TRIGGER** | | |
| 2.1.1 Procedure `create_booking`: allow guests to create booking.<br>• Input: check in date, check out date, rooms,… | 3 | |
| 2.1.2 Procedure `check_out`: check out a booking. *<br>• Input: bookingid, list of rooms with list of food consumed for each room | 1 | |
| 2.1.3 Function `get_vacant_roomtypes`: allow guests to get list of roomtypes whose vacant rooms meet guests' requirements.<br>• Input: branch, number of guests, number of rooms, check in date, check out date | 2 | |

[1] *Student(s) who do this task. Student No (1..4) is the same as group member in the top of page 1.*

[2] *For lecturer's use only*

| | | |
|---|---|---|
| • Output: list of roomtypes that meet guests' requirements | | |
| 2.1.4 Function `get_branch_statistics`: get statistics of a branch in a year<br>• Input: branch, year.<br>• Output: occupancy rate, total revenue, total guests of each month. | 1 | |
| 2.1.5 Function `get_rooms_calendar`: get all roomtypes, along with its rooms and list of bookings of each room.<br>• Input: branch, year, month<br>• Output: all roomtypes with its rooms and list of bookings. | 2 | |
| 2.1.6 Function `get_bookings`: get list of bookings, filtered by branch and status (upcoming, occupying, checked out).<br>• Input: branch, actual check in (null or not), actual check out (null or not)<br>• Output: list of bookings | 3 | |
| 2.2.1 Trigger to generate derived column values 1: `trigger_calculate_rental_cost`: calculate rental cost after a new booking_room is inserted. | 1 | |
| 2.2.2 Trigger to generate derived column values 2: `trigger_calculate_food_cost`: calculate food cost after a new foodconsumed is inserted. | 3 | |
| 2.2.3 Trigger to enforce business rules `trigger_check_valid_booking_room`: ensure that a guest cannot book at two different branches at the same time. | 2 | |
| **PART 3: BUILDING APPLICATIONS (3.5p)** | | |
| I. Create user (0.5p) | 1, 2, 3 | |

| | | |
|---|---|---|
| II. Implement features | | |
| 2.1.1 Log in (0.25p) | 1, 2, 3 | |
| 2.1.2 Log out (0.25p) | 1, 2, 3 | |
| 2.2.1 View, Insert, Delete, Update (CRUD) (1p) | 1, 2, 3 | |
| 2.2.2 Retrieve list of objects with filtering and sorting (1p) | 1, 2, 3 | |
| 2.2.3. Other feature (0.5p):<br>– Display monthly revenue of each month in a year of a branch<br>– Find vacant rooms that meet customers' requirements. | 1, 2, 3 | |
| Bonus:<br>- User interface is attractive and friendly (0.5p)<br>- Building a data model (0.5p) | | |

## PART 1: CREATE DATABASE IN PostgreSQL

### I. Create table

1. branch (BranchID, Province, Address, Email, Phone)

```sql
-- Table 1
CREATE SEQUENCE branch_seq;
CREATE TABLE branch (
    BranchID VARCHAR(4) DEFAULT ('BR' || lpad(nextval('branch_seq')::text, 2, '0')),
    Province VARCHAR(50) NOT NULL,
    Address VARCHAR(100) NOT NULL UNIQUE,
    Email VARCHAR(64) NOT NULL UNIQUE,
    Phone VARCHAR(10) NOT NULL UNIQUE,
    PRIMARY KEY (BranchID)
);
```

2. branch_image (BranchID, Image)

```sql
-- Table 2
CREATE TABLE branch_image (
    BranchID VARCHAR(4) NOT NULL,
    Image VARCHAR(255) NOT NULL,
    PRIMARY KEY (BranchID,Image),
    FOREIGN KEY (BranchID) REFERENCES branch (BranchID) ON UPDATE CASCADE ON DELETE CASCADE
);
```

3. roomtype (RoomTypeID, RoomName, Area, GuestNum, SingleBedNum, DoubleBedNum)

```sql
-- Table 3
CREATE TABLE roomtype (
    RoomTypeID SERIAL NOT NULL,
    RoomName VARCHAR(45) NOT NULL,
    Area INTEGER NOT NULL,
    GuestNum INTEGER NOT NULL CONSTRAINT LimitGuest CHECK(GuestNum BETWEEN 1 AND 10),
    SingleBedNum INTEGER NOT NULL,
    DoubleBedNum INTEGER NOT NULL,
    Description TEXT DEFAULT NULL,
    PRIMARY KEY (RoomTypeID)
);
```

4. roomtype_image (RoomTypeID, Image)

```sql
-- Table 4
CREATE TABLE roomtype_image (
    RoomTypeID INTEGER NOT NULL,
    Image VARCHAR(255) NOT NULL,
    PRIMARY KEY (RoomTypeID,Image),
    FOREIGN KEY (RoomTypeID) REFERENCES RoomType (RoomTypeID) ON UPDATE CASCADE ON DELETE CASCADE
);
```

5. roomtype_branch (RoomTypeID, BranchID, RentalPrice)

```
-- Table 5
CREATE TABLE roomtype_branch (
    RoomTypeID INTEGER NOT NULL,
    BranchID VARCHAR(4) NOT NULL,
    RentalPrice INTEGER NOT NULL,
    PRIMARY KEY (RoomTypeID, BranchID),
    FOREIGN KEY (BranchID) REFERENCES branch (BranchID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (RoomTypeID) REFERENCES roomtype (RoomTypeID) ON UPDATE CASCADE ON DELETE CASCADE
);
```

6. room (BranchID, RoomNumber, RoomTypeID)

```
-- Table 6
CREATE TABLE room (
    BranchID VARCHAR(4) NOT NULL,
    RoomNumber VARCHAR(3) NOT NULL,
    RoomTypeID INTEGER NOT NULL,
    PRIMARY KEY (BranchID,RoomNumber),
    FOREIGN KEY (BranchID) REFERENCES Branch (BranchID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (RoomTypeID) REFERENCES roomtype (RoomTypeID) ON UPDATE CASCADE ON DELETE CASCADE
);
```

7. supplytype (SupplyTypeID, SupplyTypeName)

```
-- Table 7
CREATE SEQUENCE supplytype_seq;
CREATE TABLE supplytype (
    SupplyTypeID VARCHAR(6)  DEFAULT ('SP' || lpad(nextval('supplytype_seq')::text, 4, '0')),
    SupplyTypeName VARCHAR(50) NOT NULL UNIQUE,
    PRIMARY KEY (SupplyTypeID)
);
```

8. roomtype_supplytype (SupplyTypeID, RoomTypeID, Quantity)

```
-- Table 8
CREATE TABLE roomtype_supplytype (
    SupplyTypeID VARCHAR(6) NOT NULL,
    RoomTypeID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL DEFAULT '1',
    PRIMARY KEY (SupplyTypeID,RoomTypeID),
    FOREIGN KEY (SupplyTypeID) REFERENCES supplytype (SupplyTypeID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (RoomTypeID) REFERENCES roomtype (RoomTypeID) ON UPDATE CASCADE ON DELETE CASCADE
);
```

9. supply (BranchID, SupplyTypeID, SupplyIndex, RoomNumber, Condition)

```
-- Table 9
CREATE TABLE supply (
    BranchID VARCHAR(4) NOT NULL,
    SupplyTypeID VARCHAR(6) NOT NULL,
    SupplyIndex INTEGER NOT NULL,
    RoomNumber VARCHAR(3) DEFAULT NULL,
    Condition VARCHAR(45) DEFAULT 'Good',
    PRIMARY KEY (BranchID,SupplyTypeID,SupplyIndex),
    FOREIGN KEY (BranchID,RoomNumber) REFERENCES room (BranchID,RoomNumber) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (SupplyTypeID) REFERENCES supplytype (SupplyTypeID) ON UPDATE CASCADE ON DELETE CASCADE
) ;
```

10. customer (CustomerID, CitizenID, FullName, DateOfBirth, Phone, Email, Username, Password)

```
-- Table 10
CREATE SEQUENCE customer_seq;
CREATE TABLE customer (
    CustomerID VARCHAR(8) DEFAULT ('CS' || lpad(nextval('customer_seq')::text, 6, '0')),
    CitizenID VARCHAR(12) NOT NULL UNIQUE,
    FullName VARCHAR(45) NOT NULL,
    DateOfBirth DATE NOT NULL,
    Phone VARCHAR(12) NOT NULL UNIQUE,
    Email VARCHAR(45) DEFAULT NULL UNIQUE,
    Username VARCHAR(45) DEFAULT NULL UNIQUE,
    Password VARCHAR(45) DEFAULT NULL,
    PRIMARY KEY (CustomerID)
);
```

11. booking (BookingID, BookingDate, GuestCount, CheckIn, CheckOut, ActualCheckIn, ActualCheckOut, RentalCost, FoodCost, CustomerID)

```
-- Table 11
CREATE SEQUENCE booking_seq;
CREATE TABLE booking (
    BookingID VARCHAR(10) DEFAULT ('BK' || lpad(nextval('booking_seq')::text, 8, '0')),
    BookingDate TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    GuestCount INTEGER NOT NULL,
    CheckIn TIMESTAMP NOT NULL,
    CheckOut TIMESTAMP NOT NULL,
    ActualCheckIn TIMESTAMP,
    ActualCheckOut TIMESTAMP,
    RentalCost INTEGER DEFAULT '0',
    FoodCost INTEGER DEFAULT '0',
    CustomerID VARCHAR(8) NOT NULL,
    PRIMARY KEY (BookingID),
    FOREIGN KEY (CustomerID) REFERENCES customer (CustomerID) ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT CheckIn_After_BookingDate CHECK(CheckIn >= BookingDate),
    CONSTRAINT CheckOut_After_CheckIn CHECK(CheckOut >= CheckIn)
);
```

12. booking_room (BookingID, BranchID, RoomNumber)

```
-- Table 12
CREATE TABLE booking_room (
    BookingID VARCHAR(10),
    BranchID VARCHAR(4) NOT NULL,
    RoomNumber VARCHAR(3) NOT NULL,
    PRIMARY KEY (BookingID, BranchID, RoomNumber),
    FOREIGN KEY (BookingID) REFERENCES booking (BookingID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (BranchID,RoomNumber) REFERENCES room (BranchID,RoomNumber) ON UPDATE CASCADE ON DELETE CASCADE
);
```

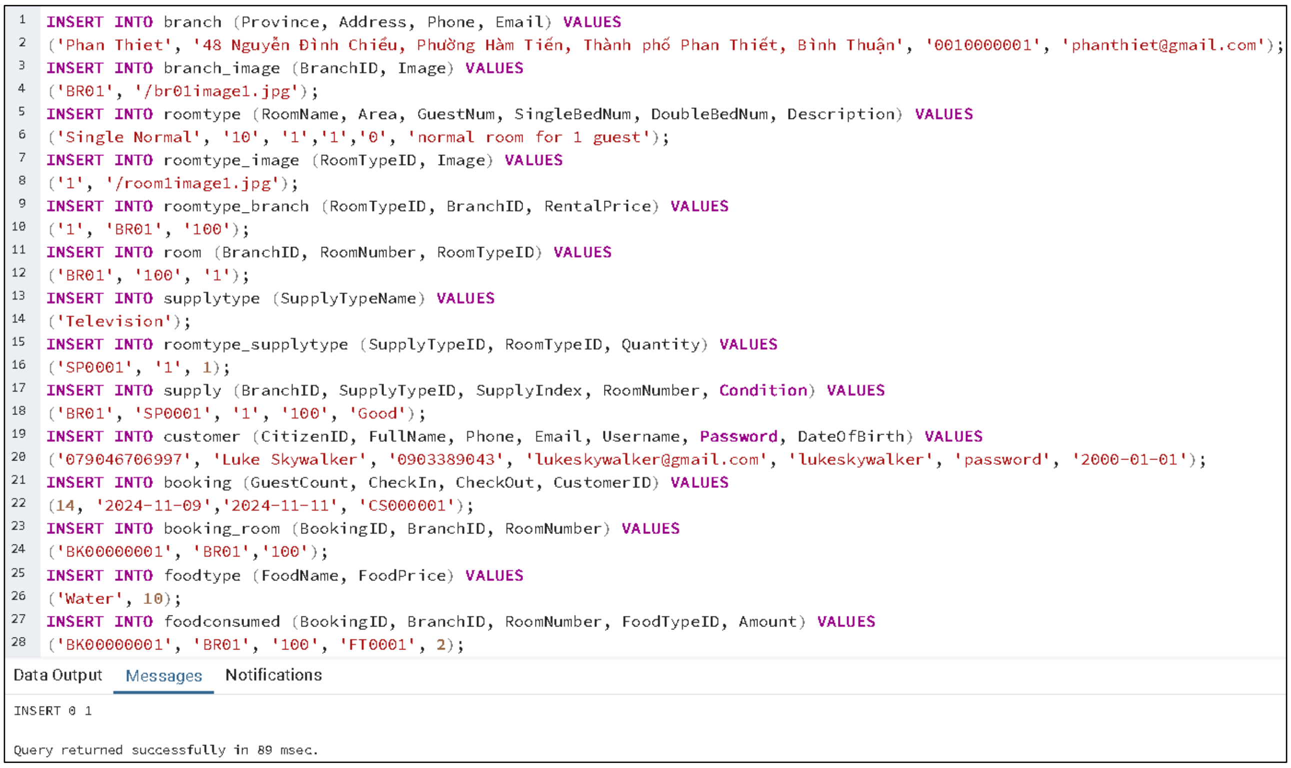
13. foodtype

```
-- Table 13
CREATE SEQUENCE foodtype_seq;
CREATE TABLE foodtype (
    FoodTypeID VARCHAR(6) DEFAULT ('FT' || lpad(nextval('foodtype_seq')::text, 4, '0')),
    FoodName VARCHAR(45) NOT NULL UNIQUE,
    FoodPrice INT NOT NULL,
    PRIMARY KEY (FoodTypeID)
);
```

14. foodconsumed

```
-- Table 14
CREATE TABLE foodconsumed (
    BookingID VARCHAR(10),
    BranchID varchar(4) NOT NULL,
    RoomNumber varchar(3) NOT NULL,
    FoodTypeID VARCHAR(6) NOT NULL ,
    Amount INTEGER NOT NULL DEFAULT '0',
    PRIMARY KEY (BookingID,BranchID,RoomNumber, FoodTypeID),
    FOREIGN KEY (BookingID,BranchID,RoomNumber) REFERENCES booking_room (BookingID,BranchID,RoomNumber) ON UPDATE
    CASCADE ON DELETE CASCADE,
    FOREIGN KEY (FoodTypeID) REFERENCES foodtype (FoodTypeID) ON UPDATE CASCADE ON DELETE CASCADE
);
```

## II. Insert



```sql
INSERT INTO branch (Province, Address, Phone, Email) VALUES
('Phan Thiet', '48 Nguyễn Đình Chiểu, Phường Hàm Tiến, Thành phố Phan Thiết, Bình Thuận', '0010000001', 'phanthiet@gmail.com');
INSERT INTO branch_image (BranchID, Image) VALUES
('BR01', '/br01image1.jpg');
INSERT INTO roomtype (RoomName, Area, GuestNum, SingleBedNum, DoubleBedNum, Description) VALUES
('Single Normal', '10', '1','1','0', 'normal room for 1 guest');
INSERT INTO roomtype_image (RoomTypeID, Image) VALUES
('1', '/room1image1.jpg');
INSERT INTO roomtype_branch (RoomTypeID, BranchID, RentalPrice) VALUES
('1', 'BR01', '100');
INSERT INTO room (BranchID, RoomNumber, RoomTypeID) VALUES
('BR01', '100', '1');
INSERT INTO supplytype (SupplyTypeName) VALUES
('Television');
INSERT INTO roomtype_supplytype (SupplyTypeID, RoomTypeID, Quantity) VALUES
('SP0001', '1', 1);
INSERT INTO supply (BranchID, SupplyTypeID, SupplyIndex, RoomNumber, Condition) VALUES
('BR01', 'SP0001', '1', '100', 'Good');
INSERT INTO customer (CitizenID, FullName, Phone, Email, Username, Password, DateOfBirth) VALUES
('079046706997', 'Luke Skywalker', '0903389043', 'lukeskywalker@gmail.com', 'lukeskywalker', 'password', '2000-01-01');
INSERT INTO booking (GuestCount, CheckIn, CheckOut, CustomerID) VALUES
(14, '2024-11-09','2024-11-11', 'CS000001');
INSERT INTO booking_room (BookingID, BranchID, RoomNumber) VALUES
('BK00000001', 'BR01','100');
INSERT INTO foodtype (FoodName, FoodPrice) VALUES
('Water', 10);
INSERT INTO foodconsumed (BookingID, BranchID, RoomNumber, FoodTypeID, Amount) VALUES
('BK00000001', 'BR01', '100', 'FT0001', 2);
```

```
INSERT 0 1

Query returned successfully in 89 msec.
```

## PART 2: STORE PROCEDURE, FUNCTION, TRIGGER (3 points)

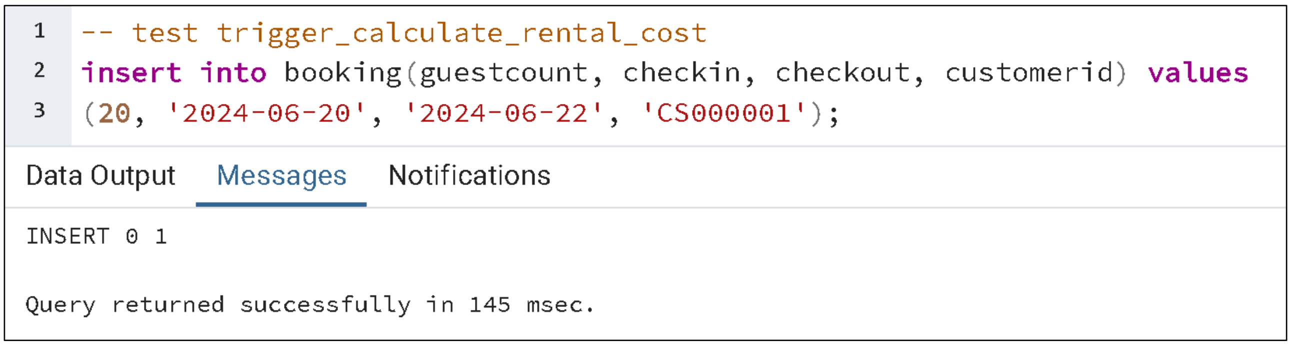
### I. Trigger

1. Trigger 1: `trigger_calculate_rental_cost`

   a. Create trigger

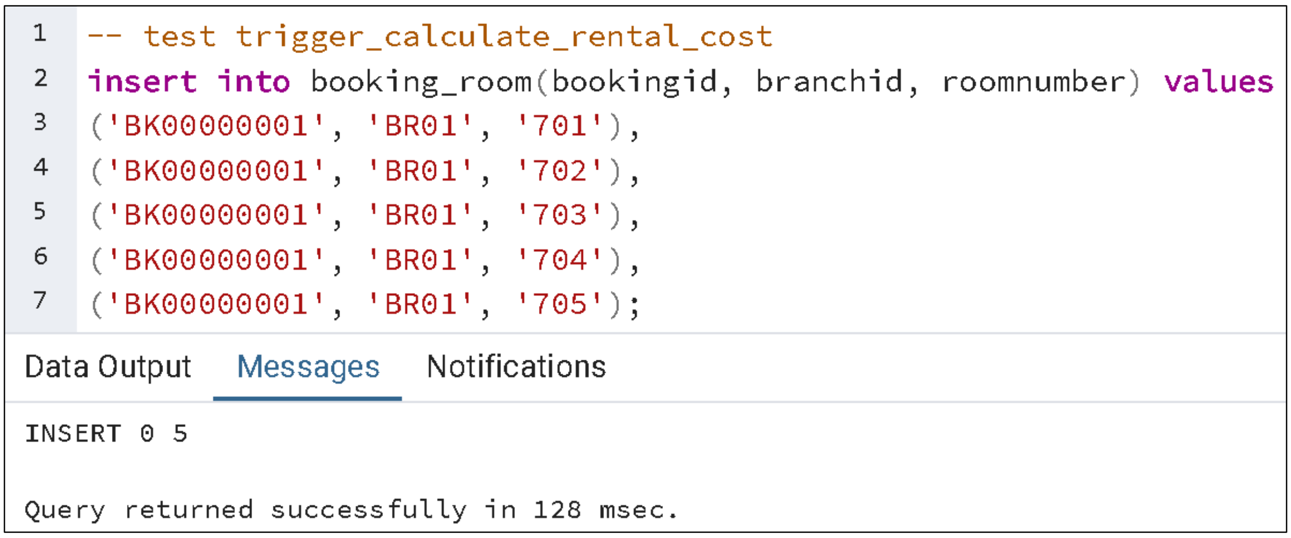
```
CREATE OR REPLACE FUNCTION calculate_rental_cost()
    RETURNS TRIGGER AS
    $body$
        DECLARE
            oneday_cost integer;
            total_day integer;
        BEGIN
            SELECT roomtype_branch.rentalprice, COALESCE(DATE_PART('day', booking.
            CheckOut::timestamp - booking.CheckIn::timestamp) + 1,0)
            into oneday_cost, total_day
            from booking natural JOIN booking_room
            natural join room natural join roomtype natural join roomtype_branch
            where booking.bookingid = new.bookingid;

            UPDATE booking SET rentalcost = rentalcost + oneday_cost*total_day where
            bookingid = new.bookingid;
            return new;
        END;
    $body$
    LANGUAGE plpgsql;
CREATE OR REPLACE TRIGGER trigger_calculate_rental_cost
    AFTER INSERT ON booking_room
    FOR EACH ROW
    EXECUTE FUNCTION calculate_rental_cost();
```

   b. Test trigger



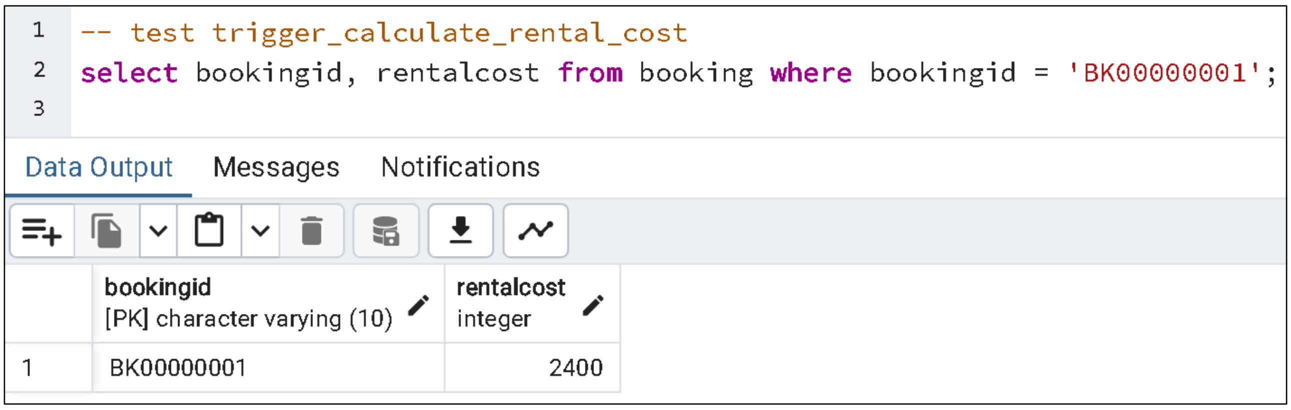
```
-- test trigger_calculate_rental_cost
insert into booking(guestcount, checkin, checkout, customerid) values
(20, '2024-06-20', '2024-06-22', 'CS000001');
```

```
INSERT 0 1

Query returned successfully in 145 msec.
```



```
-- test trigger_calculate_rental_cost
insert into booking_room(bookingid, branchid, roomnumber) values
('BK00000001', 'BR01', '701'),
('BK00000001', 'BR01', '702'),
('BK00000001', 'BR01', '703'),
('BK00000001', 'BR01', '704'),
('BK00000001', 'BR01', '705');
```

```
INSERT 0 5

Query returned successfully in 128 msec.
```



```
-- test trigger_calculate_rental_cost
select bookingid, rentalcost from booking where bookingid = 'BK00000001';
```

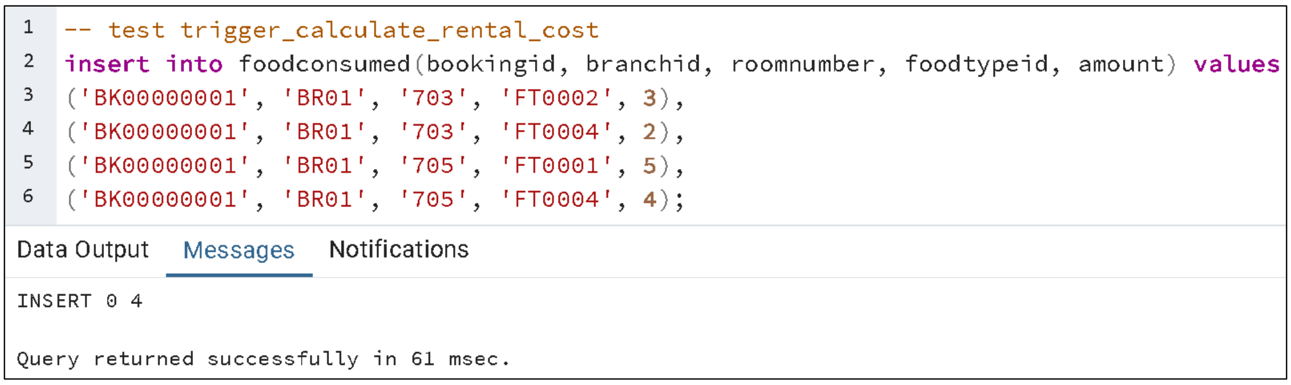
| | bookingid [PK] character varying (10) | rentalcost integer |
|---|---|---|
| 1 | BK00000001 | 2400 |

2. Trigger 2: `trigger_calculate_food_cost`

   a. Create trigger

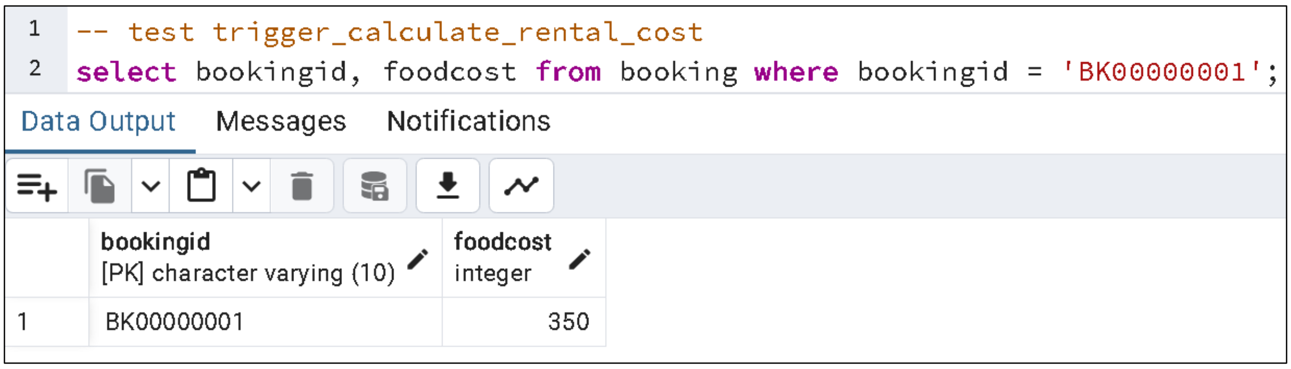
```
CREATE OR REPLACE FUNCTION calculate_food_cost()
    RETURNS TRIGGER AS
    $body$
        DECLARE
            onefood_cost integer;
        BEGIN
            select foodprice into onefood_cost
            from foodtype where foodtypeid=NEW.foodtypeid;

            UPDATE booking SET
                foodcost = foodcost + onefood_cost*NEW.amount,
                ActualCheckOut = CURRENT_TIMESTAMP
            where bookingid = NEW.bookingid;
            return new;
        END;
    $body$
    LANGUAGE plpgsql;
CREATE OR REPLACE TRIGGER trigger_calculate_food_cost
    AFTER INSERT ON foodconsumed
    FOR EACH ROW
    EXECUTE FUNCTION calculate_food_cost();
```

b. Test trigger



```
-- test trigger_calculate_rental_cost
insert into foodconsumed(bookingid, branchid, roomnumber, foodtypeid, amount) values
('BK00000001', 'BR01', '703', 'FT0002', 3),
('BK00000001', 'BR01', '703', 'FT0004', 2),
('BK00000001', 'BR01', '705', 'FT0001', 5),
('BK00000001', 'BR01', '705', 'FT0004', 4);
```

```
INSERT 0 4

Query returned successfully in 61 msec.
```



```
-- test trigger_calculate_rental_cost
select bookingid, foodcost from booking where bookingid = 'BK00000001';
```

| | bookingid [PK] character varying (10) | foodcost integer |
|---|---|---|
| 1 | BK00000001 | 350 |

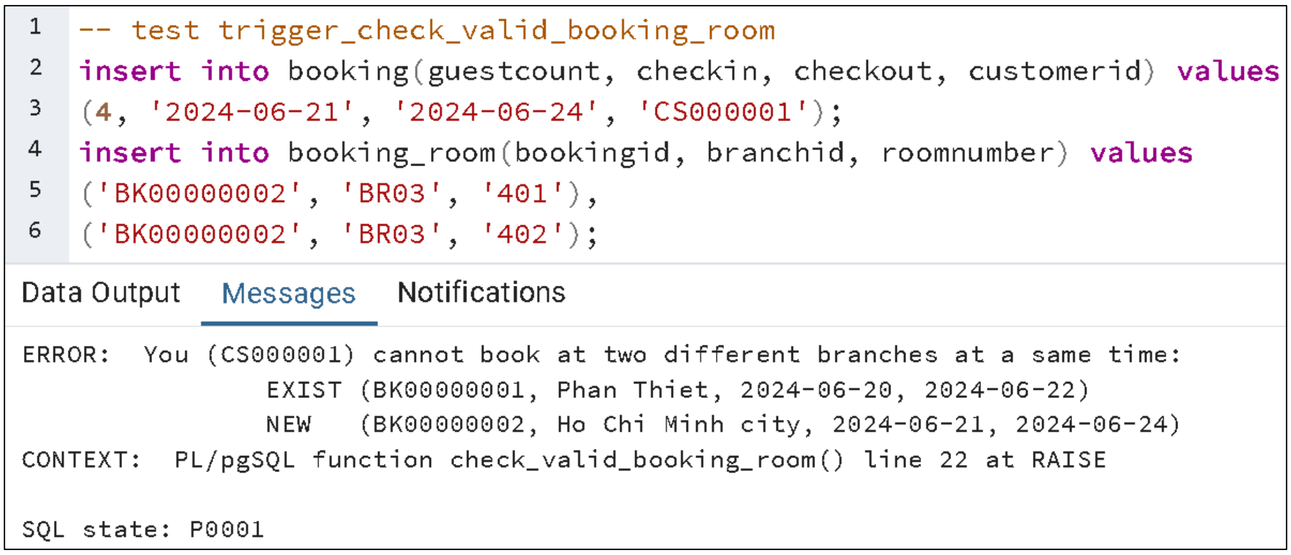
3. Trigger 3: `trigger_check_valid_booking_room`

   a. Create trigger

```
CREATE OR REPLACE FUNCTION check_valid_booking_room()
    RETURNS TRIGGER AS
    $body$
        DECLARE
            InputCustomerID varchar;
            InputCheckIn timestamp;
            InputCheckOut timestamp;
            InputProvince varchar;
            ExistBookingID varchar;
            ExistCheckIn timestamp;
            ExistCheckOut timestamp;
            ExistProvince varchar;
        BEGIN
            SELECT booking.customerid, booking.checkin, booking.checkout
            INTO InputCustomerID, InputCheckIn, InputCheckOut FROM
            booking WHERE booking.bookingid = NEW.bookingid;
            SELECT branch.province into InputProvince from branch where
            branchid = new.branchid;

            SELECT booking.bookingid, br.province, booking.checkin,
            booking.checkout into ExistBookingID, ExistProvince,
            ExistCheckIn, ExistCheckOut FROM booking NATURAL JOIN
            (booking_room natural join branch) as br
                WHERE booking.customerid = InputCustomerID AND NEW.
                BranchID <> br.BranchID
                AND ((InputCheckIn <= booking.CheckIn AND InputCheckOut
                >= booking.CheckIn)
                OR (InputCheckIn >= booking.CheckIn AND InputCheckIn <=
                booking.CheckOut));

            IF ExistBookingID IS NOT NULL THEN
                RAISE EXCEPTION 'You (%) cannot book at two different
                branches at a same time:
                EXIST (%, %, %, %)
                NEW   (%, %, %, %)', InputCustomerID,
                ExistBookingID, ExistProvince, DATE(ExistCheckIn), DATE
                (ExistCheckOut),
                NEW.bookingid, InputProvince, DATE(InputCheckIn), DATE
                (InputCheckOut);
            ELSE
                RETURN NEW;
            END IF;
        END;
    $body$
    LANGUAGE plpgsql;
CREATE OR REPLACE TRIGGER trigger_check_valid_booking_room
    BEFORE INSERT ON booking_room
    FOR EACH ROW
    EXECUTE FUNCTION check_valid_booking_room();
```

b. Test trigger

This insert command violates the rule of this trigger since  this customer (CS000001) already had a booking (BK00000001) in another branch (Phan Thiet) at the same time (2024/06/20 – 2024/06/22)

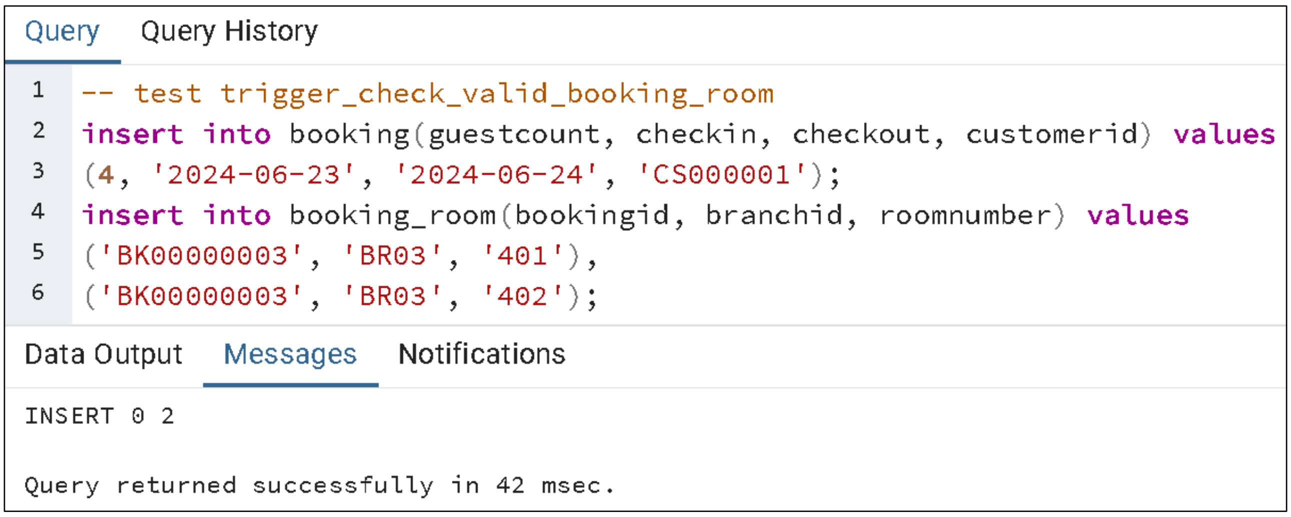


```sql
-- test trigger_check_valid_booking_room
insert into booking(guestcount, checkin, checkout, customerid) values
(4, '2024-06-21', '2024-06-24', 'CS000001');
insert into booking_room(bookingid, branchid, roomnumber) values
('BK00000002', 'BR03', '401'),
('BK00000002', 'BR03', '402');
```

Data Output Messages Notifications

```
ERROR:  You (CS000001) cannot book at two different branches at a same time:
                EXIST (BK00000001, Phan Thiet, 2024-06-20, 2024-06-22)
                NEW   (BK00000002, Ho Chi Minh city, 2024-06-21, 2024-06-24)
CONTEXT:  PL/pgSQL function check_valid_booking_room() line 22 at RAISE

SQL state: P0001
```

This insert command does not violate the rule of this trigger since this booking time period does not overlap with this customer’s other bookings in the database:



Query Query History

```sql
-- test trigger_check_valid_booking_room
insert into booking(guestcount, checkin, checkout, customerid) values
(4, '2024-06-23', '2024-06-24', 'CS000001');
insert into booking_room(bookingid, branchid, roomnumber) values
('BK00000003', 'BR03', '401'),
('BK00000003', 'BR03', '402');
```

Data Output Messages Notifications

```
INSERT 0 2

Query returned successfully in 42 msec.
```

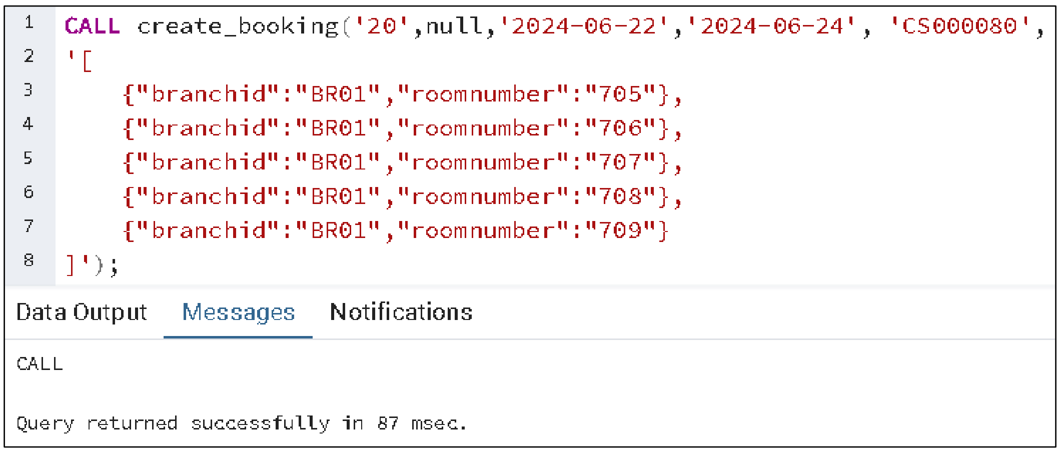
## II. Store Procedure/Function

1. Procedure 1: `create_booking`

   a. Create procedure

```
CREATE OR REPLACE PROCEDURE create_booking(
        in inputguestcount integer,
        in inputbookingdate timestamp,
        in inputcheckin timestamp,
        in inputcheckout timestamp,
        in inputcustomerid varchar,
        in inputbookingroom json
    ) AS
    $body$
        DECLARE
            newbookingid varchar;
            room_rec json;
        BEGIN
            INSERT INTO booking (BookingDate, GuestCount, CheckIn,
            CheckOut, CustomerID) VALUES
            (COALESCE(inputbookingdate, current_timestamp),
            inputguestcount, inputcheckin, inputcheckout,
            inputcustomerid) RETURNING bookingid INTO newbookingid;
            for room_rec in select json_array_elements(inputbookingroom)
                loop
                    insert into booking_room (BookingID, BranchID,
                    RoomNumber)
                    values (
                        newbookingid,
                        (room_rec->>'branchid')::text,
                        (room_rec->>'roomnumber')::text
                    );
                end loop;
        END;
    $body$
    LANGUAGE 'plpgsql';
```

   b. Call procedure



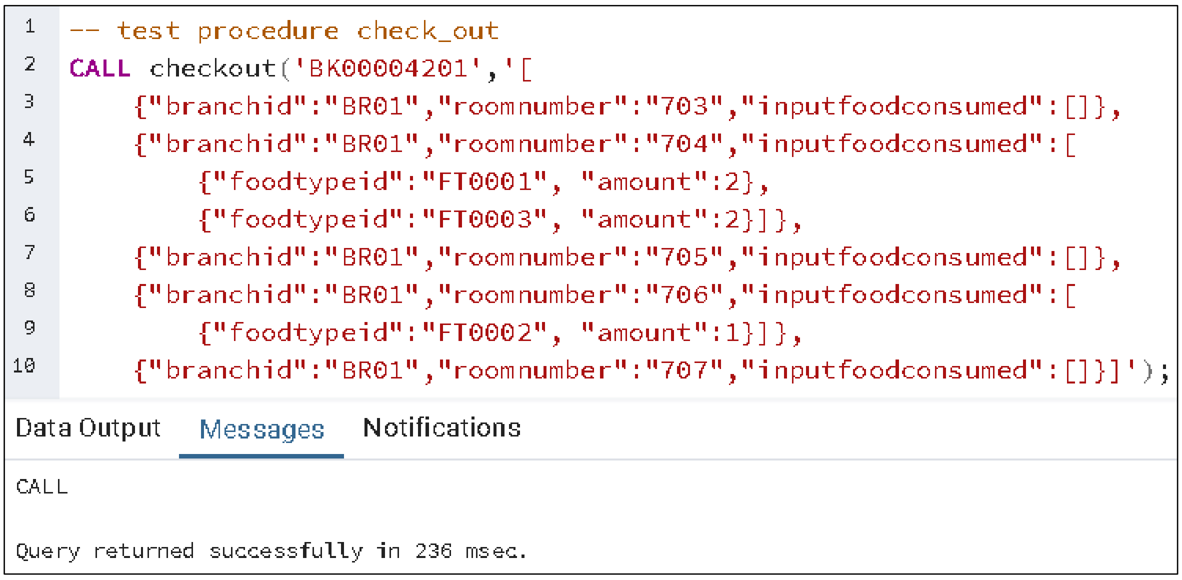
```
CALL create_booking('20',null,'2024-06-22','2024-06-24', 'CS000080',
'[
    {"branchid":"BR01","roomnumber":"705"},
    {"branchid":"BR01","roomnumber":"706"},
    {"branchid":"BR01","roomnumber":"707"},
    {"branchid":"BR01","roomnumber":"708"},
    {"branchid":"BR01","roomnumber":"709"}
]');
```

```
CALL

Query returned successfully in 87 msec.
```

2. Procedure 2: `check_out`
    a. Create procedure

```
CREATE OR REPLACE PROCEDURE checkout(
        inputbookingid varchar, json_data json
    )AS
    $body$
        DECLARE
            booking_record json;
            food_record json;
        BEGIN
            FOR booking_record IN SELECT json_array_elements(json_data)
            LOOP
                FOR food_record IN SELECT json_array_elements
                (booking_record->'inputfoodconsumed')
                LOOP
                    INSERT INTO foodconsumed (BookingID, BranchID,
                    RoomNumber, FoodTypeID, Amount)
                    VALUES (
                        inputbookingid,
                        (booking_record->>'branchid')::text,
                        (booking_record->>'roomnumber')::text,
                        (food_record->>'foodtypeid')::text,
                        (food_record->>'amount')::int
                    );
                END LOOP;
            END LOOP;
        END;
    $body$
    LANGUAGE plpgsql;
```

    b. Call procedure



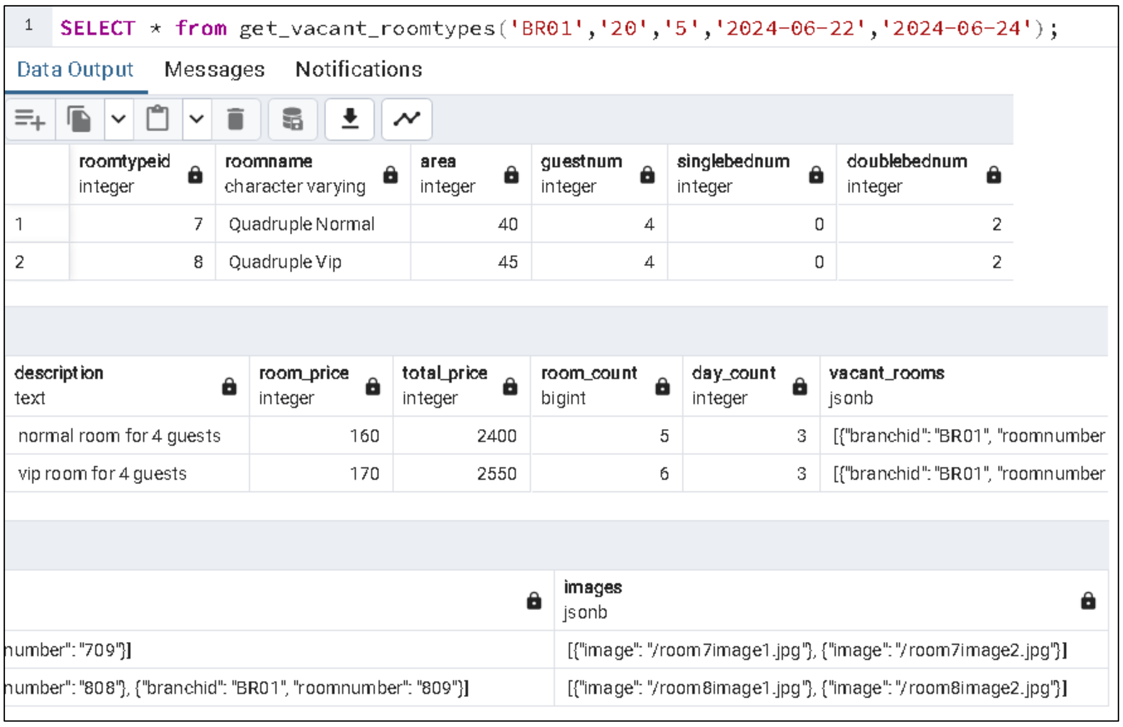
```
-- test procedure check_out
CALL checkout('BK00004201','[
    {"branchid":"BR01","roomnumber":"703","inputfoodconsumed":[]},
    {"branchid":"BR01","roomnumber":"704","inputfoodconsumed":[
        {"foodtypeid":"FT0001", "amount":2},
        {"foodtypeid":"FT0003", "amount":2}]},
    {"branchid":"BR01","roomnumber":"705","inputfoodconsumed":[]},
    {"branchid":"BR01","roomnumber":"706","inputfoodconsumed":[
        {"foodtypeid":"FT0002", "amount":1}]},
    {"branchid":"BR01","roomnumber":"707","inputfoodconsumed":[]}]');
```

```
CALL

Query returned successfully in 236 msec.
```

3. Function 3: `get_vacant_roomtypes`
   a. Create procedure/ function

```
CREATE OR REPLACE FUNCTION get_vacant_roomtypes(
        InputBranch varchar,
        InputGuest integer,
        InputRoom integer,
        InputCheckIn timestamp,
        InputCheckOut timestamp
    ) RETURNS TABLE (
        roomtypeid integer,
        roomname varchar,
        area integer,
        guestnum integer,
        singlebednum integer,
        doublebednum integer,
        description text,
        room_price integer,
        total_price integer,
        room_count bigint,
        day_count integer,
        vacant_rooms jsonb,
        images jsonb
    ) AS
    $body$
        DECLARE
            CalculatedGuestNum integer DEFAULT CEIL(InputGuest::numeric/
            InputRoom::numeric);
            day_count integer DEFAULT EXTRACT(DAY FROM
            (InputCheckOut::timestamp - InputCheckIn::timestamp)) + 1;
        BEGIN
            DROP TABLE IF EXISTS vacant_rooms;
            DROP TABLE IF EXISTS vacant_roomtypes;
            CREATE TEMPORARY TABLE vacant_rooms AS
            SELECT * FROM
            (SELECT * FROM room NATURAL JOIN roomtype WHERE roomtype.
            GuestNum = CalculatedGuestNum) AS r0
            WHERE r0.BranchID = InputBranch
            AND NOT EXISTS
            (SELECT * FROM booking NATURAL JOIN booking_room
            WHERE (r0.RoomNumber = booking_room.RoomNumber AND r0.
            BranchID = booking_room.BranchID)
            AND ((InputCheckIn <= booking.CheckIn AND InputCheckOut >=
            booking.CheckIn)
            OR (InputCheckIn >= booking.CheckIn AND InputCheckIn <=
            booking.CheckOut)));
            CREATE TEMPORARY TABLE vacant_roomtypes AS
            SELECT roomtype.*, roomtype_branch.rentalprice,
            roomtype_branch.rentalprice*day_count*InputRoom, COUNT(*) AS
            room_count, day_count as day_count,
            jsonb_agg(jsonb_build_object(
                'branchid', vacant_rooms.branchid,
                'roomnumber', vacant_rooms.roomnumber
            ))
            FROM (roomtype NATURAL JOIN roomtype_branch) NATURAL JOIN
            vacant_rooms
            WHERE roomtype.GuestNum = CalculatedGuestNum
            GROUP BY roomtype.RoomTypeID, roomtype_branch.rentalprice
            HAVING COUNT(*) >= InputRoom;

            ALTER TABLE vacant_roomtypes ADD PRIMARY KEY (roomtypeid);


            RETURN QUERY
            SELECT vacant_roomtypes.*, jsonb_agg(jsonb_build_object(
                'image', roomtype_image.image
            )) FROM vacant_roomtypes NATURAL LEFT JOIN roomtype_image
            GROUP BY vacant_roomtypes.roomtypeid;
            DROP TABLE IF EXISTS vacant_rooms;
            DROP TABLE IF EXISTS vacant_roomtypes;
        END;
    $body$
    LANGUAGE plpgsql;
```

b. Call procedure/ function



```
SELECT * from get_vacant_roomtypes('BR01','20','5','2024-06-22','2024-06-24');
```

Data Output Messages Notifications

| | roomtypeid integer | roomname character varying | area integer | guestnum integer | singlebednum integer | doublebednum integer |
|---|---|---|---|---|---|---|
| 1 | 7 | Quadruple Normal | 40 | 4 | 0 | 2 |
| 2 | 8 | Quadruple Vip | 45 | 4 | 0 | 2 |

| description text | room_price integer | total_price integer | room_count bigint | day_count integer | vacant_rooms jsonb |
|---|---|---|---|---|---|
| normal room for 4 guests | 160 | 2400 | 5 | 3 | [{"branchid": "BR01", "roomnumber |
| vip room for 4 guests | 170 | 2550 | 6 | 3 | [{"branchid": "BR01", "roomnumber |

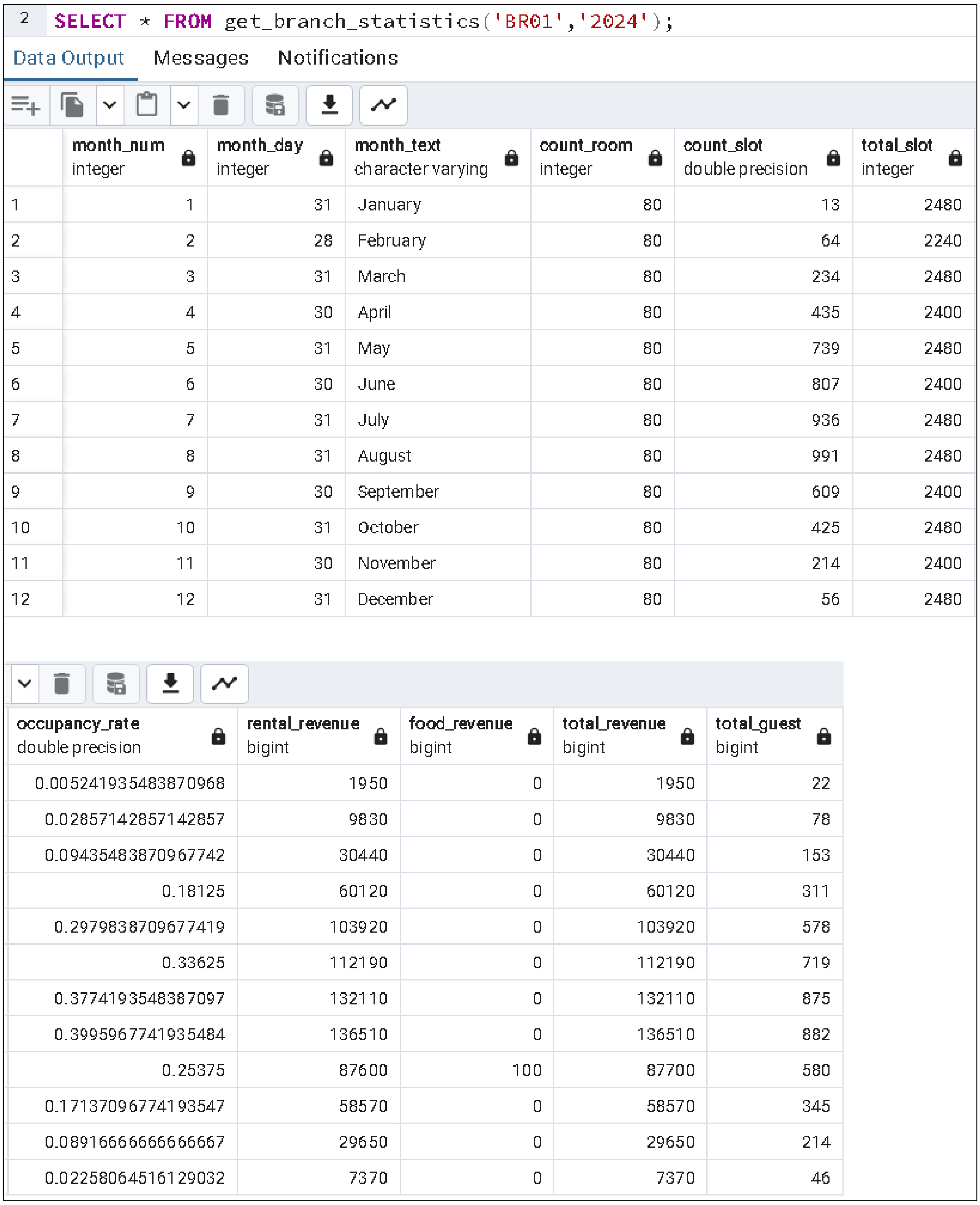
| | images jsonb |
|---|---|
| umber": "709"}] | [{"image": "/room7image1.jpg"}, {"image": "/room7image2.jpg"}] |
| umber": "808"}, {"branchid": "BR01", "roomnumber": "809"}] | [{"image": "/room8image1.jpg"}, {"image": "/room8image2.jpg"}] |

4. Function 4: `get_branch_statistics`

    a. Create procedure/ function

```
CREATE OR REPLACE FUNCTION get_branch_statistics(
        InputBranch varchar,
        InputYear integer
    ) RETURNS TABLE (
        month_num integer,
        month_day integer,
        month_text varchar,
        count_room integer,
        count_slot double precision,
        total_slot integer,
        occupancy_rate double precision,
        rental_revenue bigint,
        food_revenue bigint,
        total_revenue bigint,
        total_guest bigint
    ) AS
    $body$
        DECLARE
            count_room_var integer DEFAULT '0';
        BEGIN
            SELECT COUNT(*) INTO count_room_var FROM room WHERE room.BranchID =
            InputBranch;
            DROP TABLE IF EXISTS all_months;
            DROP TABLE IF EXISTS vacancy_rate;
            CREATE TEMPORARY TABLE all_months (
                    month_num integer,
                    month_day integer,
                    month_text varchar,
                    primary key (month_num)
                );
            INSERT INTO all_months VALUES
            (1, 31, 'January'),
            (2, 28, 'February'),
            (3, 31, 'March'),
            (4, 30, 'April'),
            (5, 31, 'May'),
            (6, 30, 'June'),
            (7, 31, 'July'),
            (8, 31, 'August'),
            (9, 30, 'September'),
            (10, 31, 'October'),
            (11, 30, 'November'),
            (12, 31, 'December');
            CREATE TEMPORARY TABLE vacancy_rate AS
            SELECT
                all_months.month_num AS month_num,
                all_months.month_day AS month_day,
                all_months.month_text AS month_text,
                count_room_var AS count_room,
                COALESCE(temp.count_slot,0) AS count_slot,
                all_months.month_day*count_room_var AS total_slot,
                COALESCE(temp.count_slot,0)/(all_months.
                month_day*count_room_var) AS occupancy_rate
            FROM (all_months NATURAL LEFT JOIN
                (SELECT
                    EXTRACT(month FROM booking.CheckIn::timestamp) as
                    month_num,
                    SUM(COALESCE(DATE_PART('day', booking.
                    CheckOut::timestamp - booking.CheckIn::timestamp) +
                    1,0)) AS count_slot
                FROM room r1 NATURAL LEFT JOIN (booking_room NATURAL
                JOIN booking)
                WHERE r1.BranchID=InputBranch AND EXTRACT(year FROM
                booking.CheckIn::timestamp) = InputYear
                GROUP BY EXTRACT(month FROM booking.CheckIn::timestamp)
                )
            as temp)
            ORDER BY all_months.month_num ASC;
            ALTER TABLE vacancy_rate ADD PRIMARY KEY (month_num);

            RETURN QUERY
            SELECT vacancy_rate.*, SUM(bktemp.rentalcost), SUM(bktemp.
            foodcost),
            SUM(bktemp.rentalcost) + SUM(bktemp.foodcost), SUM(bktemp.
            guestcount)
            FROM vacancy_rate NATURAL JOIN
            (select booking.*, EXTRACT(month FROM booking.
            CheckIn::timestamp) as month_num
            from booking natural join booking_room
            where booking_room.branchid = InputBranch and EXTRACT(year
            FROM booking.CheckIn::timestamp) = InputYear
            group by booking.bookingid) as bktemp
            group by vacancy_rate.month_num;

            DROP TABLE IF EXISTS all_months;
            DROP TABLE IF EXISTS vacancy_rate;
        END;
    $body$
    LANGUAGE plpgsql;
```

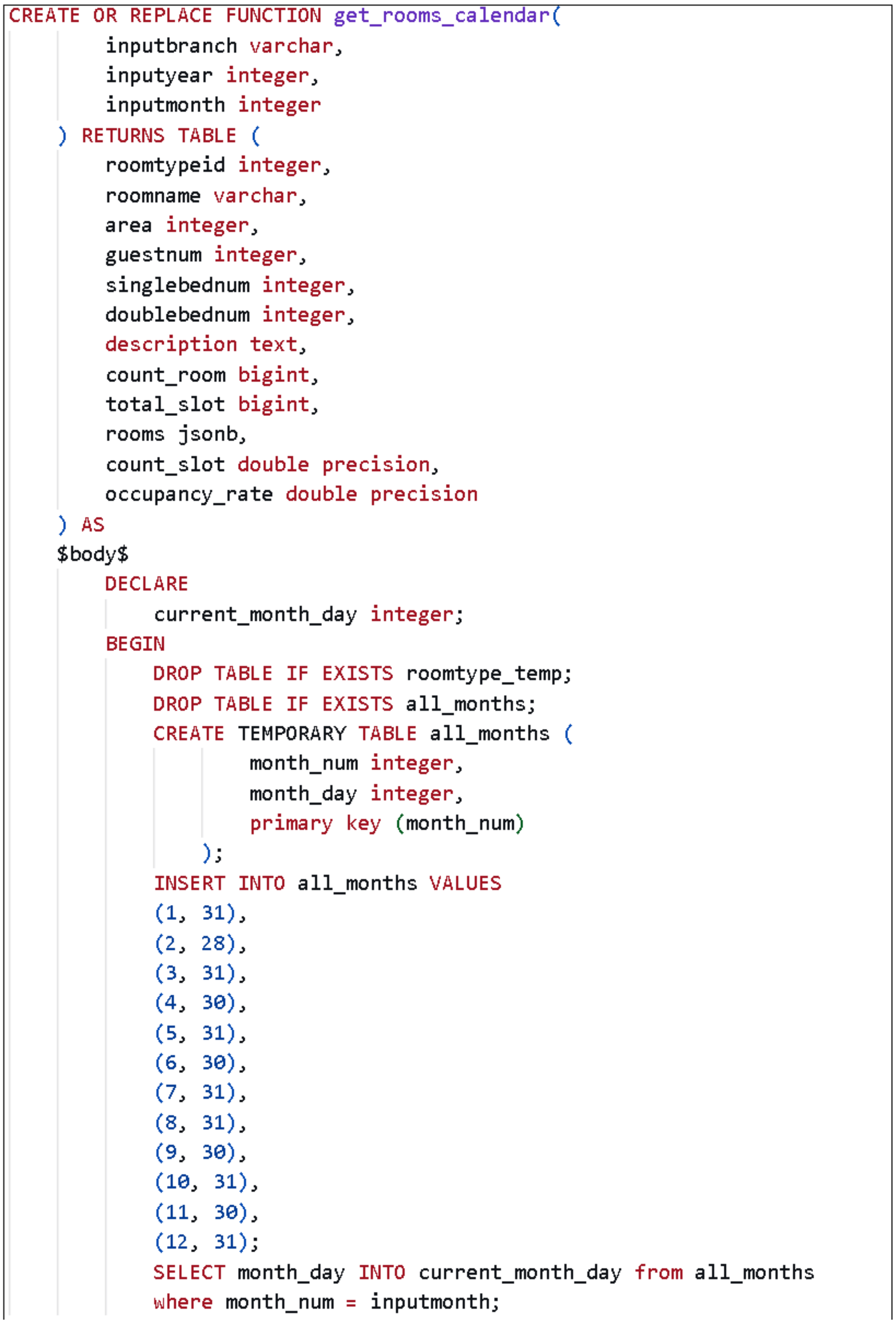
b. Call procedure/ function



```sql
SELECT * FROM get_branch_statistics('BR01','2024');
```

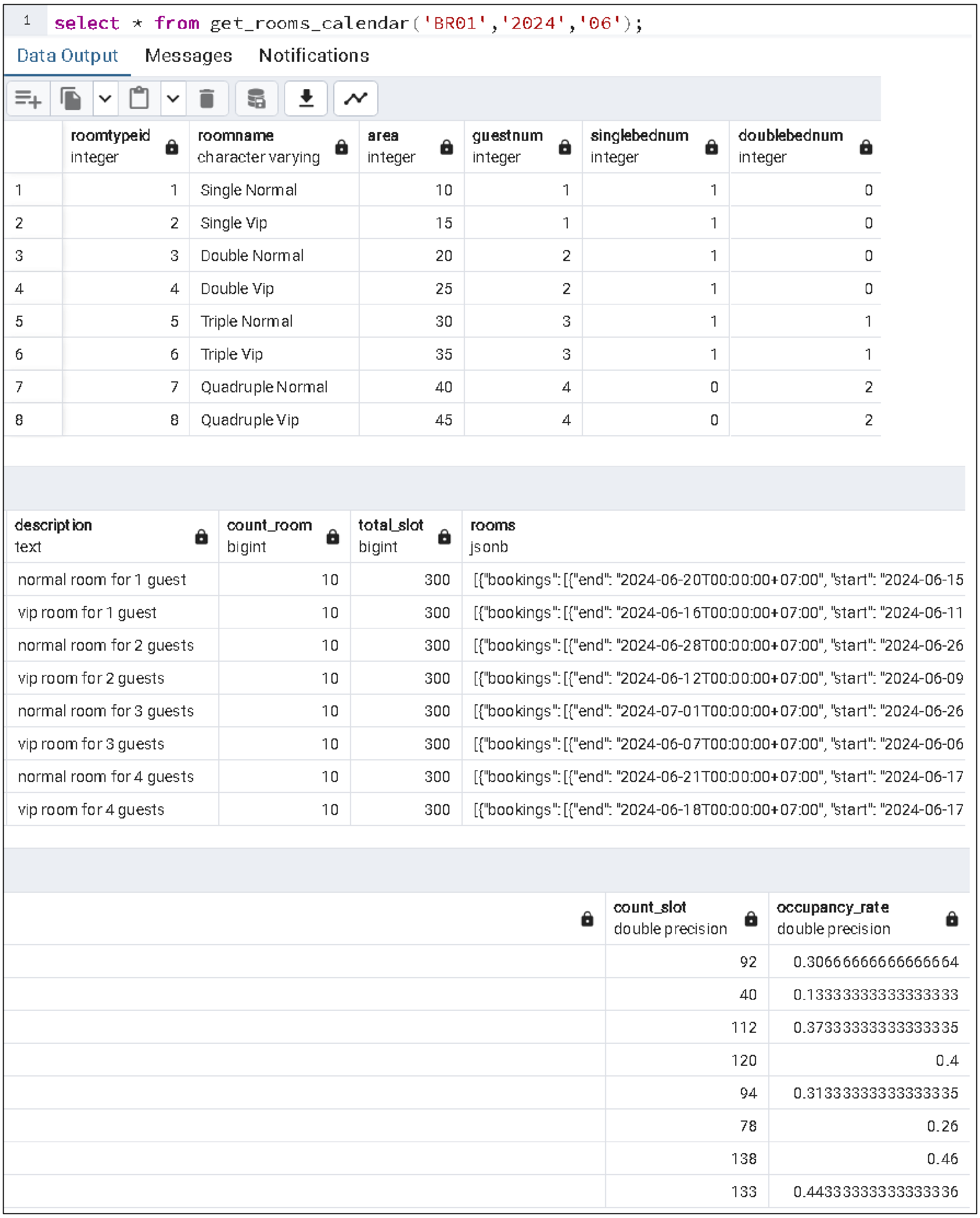
| | month_num integer | month_day integer | month_text character varying | count_room integer | count_slot double precision | total_slot integer | occupancy_rate double precision | rental_revenue bigint | food_revenue bigint | total_revenue bigint | total_guest bigint |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 31 | January | 80 | 13 | 2480 | 0.005241935483870968 | 1950 | 0 | 1950 | 22 |
| 2 | 2 | 28 | February | 80 | 64 | 2240 | 0.02857142857142857 | 9830 | 0 | 9830 | 78 |
| 3 | 3 | 31 | March | 80 | 234 | 2480 | 0.09435483870967742 | 30440 | 0 | 30440 | 153 |
| 4 | 4 | 30 | April | 80 | 435 | 2400 | 0.18125 | 60120 | 0 | 60120 | 311 |
| 5 | 5 | 31 | May | 80 | 739 | 2480 | 0.2979838709677419 | 103920 | 0 | 103920 | 578 |
| 6 | 6 | 30 | June | 80 | 807 | 2400 | 0.33625 | 112190 | 0 | 112190 | 719 |
| 7 | 7 | 31 | July | 80 | 936 | 2480 | 0.3774193548387097 | 132110 | 0 | 132110 | 875 |
| 8 | 8 | 31 | August | 80 | 991 | 2480 | 0.3995967741935484 | 136510 | 0 | 136510 | 882 |
| 9 | 9 | 30 | September | 80 | 609 | 2400 | 0.25375 | 87600 | 100 | 87700 | 580 |
| 10 | 10 | 31 | October | 80 | 425 | 2480 | 0.17137096774193547 | 58570 | 0 | 58570 | 345 |
| 11 | 11 | 30 | November | 80 | 214 | 2400 | 0.08916666666666667 | 29650 | 0 | 29650 | 214 |
| 12 | 12 | 31 | December | 80 | 56 | 2480 | 0.022580645161290322 | 7370 | 0 | 7370 | 46 |

5. Function 5: `get_rooms_calendar`
    a. Create function



```
CREATE OR REPLACE FUNCTION get_rooms_calendar(
        inputbranch varchar,
        inputyear integer,
        inputmonth integer
    ) RETURNS TABLE (
        roomtypeid integer,
        roomname varchar,
        area integer,
        guestnum integer,
        singlebednum integer,
        doublebednum integer,
        description text,
        count_room bigint,
        total_slot bigint,
        rooms jsonb,
        count_slot double precision,
        occupancy_rate double precision
    ) AS
    $body$
        DECLARE
            current_month_day integer;
        BEGIN
            DROP TABLE IF EXISTS roomtype_temp;
            DROP TABLE IF EXISTS all_months;
            CREATE TEMPORARY TABLE all_months (
                    month_num integer,
                    month_day integer,
                    primary key (month_num)
                );
            INSERT INTO all_months VALUES
            (1, 31),
            (2, 28),
            (3, 31),
            (4, 30),
            (5, 31),
            (6, 30),
            (7, 31),
            (8, 31),
            (9, 30),
            (10, 31),
            (11, 30),
            (12, 31);
            SELECT month_day INTO current_month_day from all_months
            where month_num = inputmonth;
            create temporary table roomtype_temp as
            SELECT roomtype.*, COALESCE(count(*),0) as count_room,
            COALESCE(count(*)*current_month_day,0) as total_slot,
            jsonb_agg(jsonb_build_object(
                'branchid', room.branchid,
                'roomnumber', room.roomnumber,
                'bookings', (SELECT jsonb_agg(jsonb_build_object(
                                    'title', bc_join.bookingid,
                                    'start', bc_join.checkin,
                                    'end', date_add(bc_join.
                                    checkout::timestamp, '1
                                    day'::interval), -- for render
                                    purpose
                                    'end_original', bc_join.checkout,
                                    'allDay', true, -- for render purpose
                                    'customerid', bc_join.customerid,
                                    'customername', bc_join.fullname
                                ))
                            FROM booking_room NATURAL JOIN (select *
                            from booking NATURAL JOIN customer) as
                            bc_join
                            WHERE EXTRACT(year from bc_join.
                            checkin::timestamp) = inputyear
                            AND EXTRACT(month from bc_join.
                            checkin::timestamp) = inputmonth
                            AND booking_room.branchid = room.branchid
                            AND booking_room.roomnumber = room.roomnumber
                            GROUP BY room.branchid, room.roomnumber)
            )ORDER BY room.roomnumber) FROM roomtype NATURAL LEFT JOIN
            room
            WHERE room.branchid = inputbranch
            GROUP BY roomtype.roomtypeid
            ORDER BY roomtype.roomtypeid;

            ALTER TABLE roomtype_temp ADD PRIMARY KEY (roomtypeid);

            RETURN QUERY
            SELECT roomtype_temp.*, COALESCE(SUM(br_join.num_days),0) as
            count_slot,
            COALESCE(SUM(br_join.num_days)/roomtype_temp.total_slot,0)
            as occupancy_rate FROM roomtype_temp
            NATURAL LEFT JOIN (SELECT * FROM room NATURAL LEFT JOIN
            (SELECT *, COALESCE(DATE_PART('day', booking.
            checkout::timestamp - booking.checkin::timestamp) + 1,0) as
            num_days
            FROM booking_room NATURAL JOIN booking
            WHERE booking_room.branchid = inputbranch
            AND EXTRACT(year from booking.checkin::timestamp) = inputyear
            AND EXTRACT(month from booking.checkin::timestamp) =
            inputmonth )) as br_join
            GROUP BY roomtype_temp.roomtypeid
            ORDER BY roomtype_temp.roomtypeid;
            DROP TABLE IF EXISTS roomtype_temp;
            DROP TABLE IF EXISTS all_months;
        END;
    $body$
    LANGUAGE 'plpgsql';
```

b. Call function



```sql
select * from get_rooms_calendar('BR01','2024','06');
```

Data Output Messages Notifications

| | roomtypeid integer | roomname character varying | area integer | guestnum integer | singlebednum integer | doublebednum integer |
|---|---|---|---|---|---|---|
| 1 | 1 | Single Normal | 10 | 1 | 1 | 0 |
| 2 | 2 | Single Vip | 15 | 1 | 1 | 0 |
| 3 | 3 | Double Normal | 20 | 2 | 1 | 0 |
| 4 | 4 | Double Vip | 25 | 2 | 1 | 0 |
| 5 | 5 | Triple Normal | 30 | 3 | 1 | 1 |
| 6 | 6 | Triple Vip | 35 | 3 | 1 | 1 |
| 7 | 7 | Quadruple Normal | 40 | 4 | 0 | 2 |
| 8 | 8 | Quadruple Vip | 45 | 4 | 0 | 2 |

| description text | count_room bigint | total_slot bigint | rooms jsonb |
|---|---|---|---|
| normal room for 1 guest | 10 | 300 | [{"bookings": [{"end": "2024-06-20T00:00:00+07:00", "start": "2024-06-15 |
| vip room for 1 guest | 10 | 300 | [{"bookings": [{"end": "2024-06-16T00:00:00+07:00", "start": "2024-06-11 |
| normal room for 2 guests | 10 | 300 | [{"bookings": [{"end": "2024-06-28T00:00:00+07:00", "start": "2024-06-26 |
| vip room for 2 guests | 10 | 300 | [{"bookings": [{"end": "2024-06-12T00:00:00+07:00", "start": "2024-06-09 |
| normal room for 3 guests | 10 | 300 | [{"bookings": [{"end": "2024-07-01T00:00:00+07:00", "start": "2024-06-26 |
| vip room for 3 guests | 10 | 300 | [{"bookings": [{"end": "2024-06-07T00:00:00+07:00", "start": "2024-06-06 |
| normal room for 4 guests | 10 | 300 | [{"bookings": [{"end": "2024-06-21T00:00:00+07:00", "start": "2024-06-17 |
| vip room for 4 guests | 10 | 300 | [{"bookings": [{"end": "2024-06-18T00:00:00+07:00", "start": "2024-06-17 |

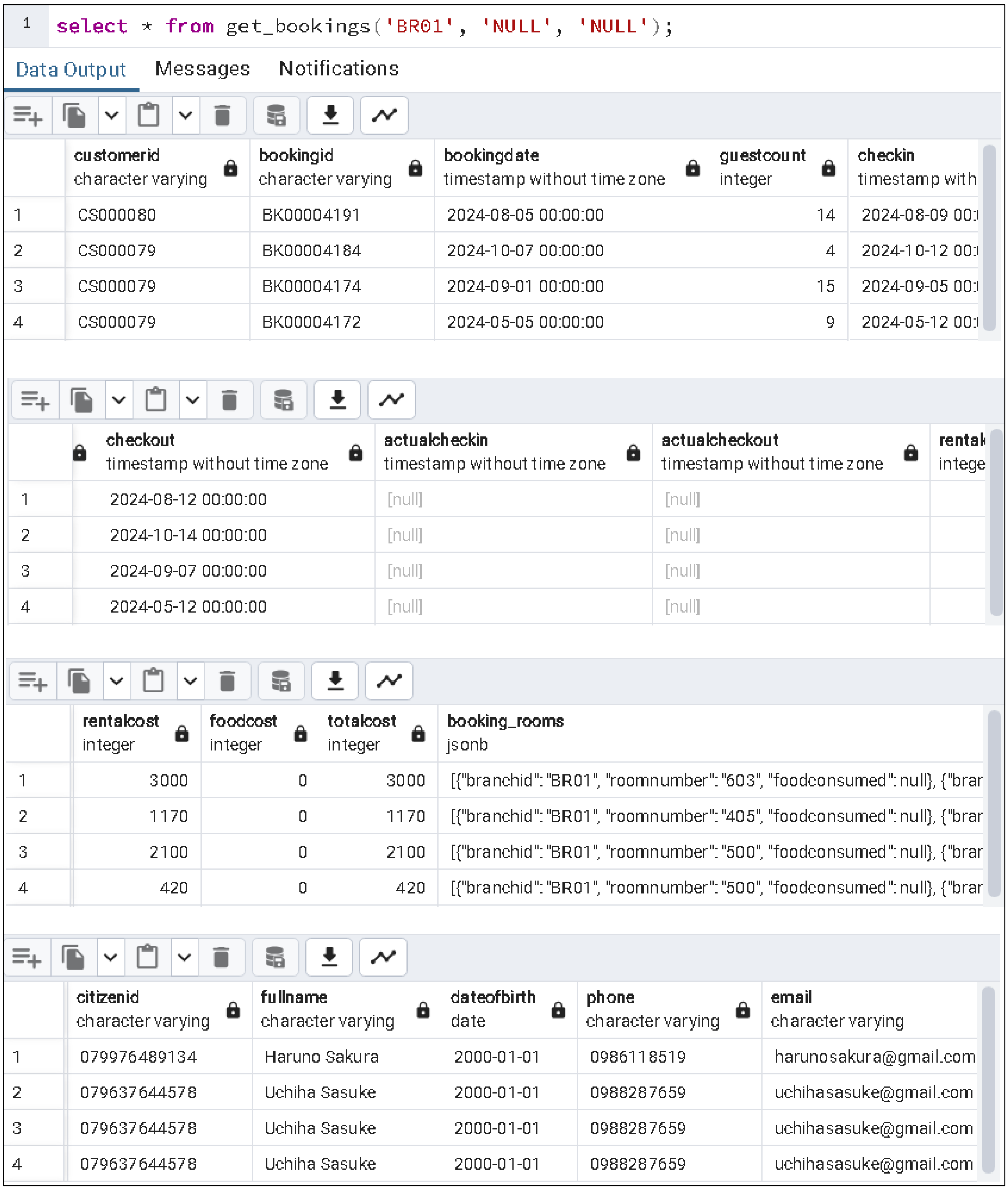
| | count_slot double precision | occupancy_rate double precision |
|---|---|---|
| | 92 | 0.30666666666666664 |
| | 40 | 0.13333333333333333 |
| | 112 | 0.37333333333333335 |
| | 120 | 0.4 |
| | 94 | 0.31333333333333335 |
| | 78 | 0.26 |
| | 138 | 0.46 |
| | 133 | 0.44333333333333336 |

6. Function 6: `get_bookings`

   a. Create function

```
CREATE OR REPLACE FUNCTION get_bookings(
        inputbranchid varchar,
        checkinnull varchar,
        checkoutnull varchar
    ) RETURNS TABLE (
        CustomerID VARCHAR,
        BookingID VARCHAR,
        BookingDate TIMESTAMP,
        GuestCount INTEGER,
        CheckIn TIMESTAMP,
        CheckOut TIMESTAMP,
        ActualCheckIn TIMESTAMP,
        ActualCheckOut TIMESTAMP,
        RentalCost INTEGER,
        FoodCost INTEGER,
        TotalCost INTEGER,
        booking_rooms jsonb,
        CitizenID VARCHAR,
        FullName VARCHAR,
        DateOfBirth DATE,
        Phone VARCHAR,
        Email VARCHAR,
        Username VARCHAR,
        Password VARCHAR
    ) AS
        $body$
        BEGIN
            RETURN QUERY
            EXECUTE format('SELECT * from (
                SELECT booking.*, booking.rentalcost + booking.foodcost as totalcost,
                    jsonb_agg(jsonb_build_object(
                        ''branchid'', booking_room.branchid,
                        ''roomnumber'', booking_room.roomnumber,
                        ''foodconsumed'', (SELECT jsonb_agg(jsonb_build_object(
                            ''foodtypeid'', food_join.foodtypeid,
                            ''foodname'', food_join.foodname,
                            ''foodprice'', food_join.foodprice,
                            ''amount'', food_join.amount
                        )) FROM (foodconsumed natural join foodtype) as food_join
                        WHERE food_join.branchid = booking_room.branchid
                        AND food_join.roomnumber = booking_room.roomnumber
                        AND food_join.bookingid = booking_room.bookingid
                        GROUP BY booking_room.bookingid, booking_room.branchid,
                        booking_room.roomnumber)
                ))
            FROM booking NATURAL JOIN booking_room WHERE booking_room.branchid = %L GROUP
            BY booking.bookingid) as br NATURAL JOIN customer
            WHERE br.ActualCheckIn IS %s AND br.ActualCheckOut IS %s
            ORDER BY br.bookingid DESC;', inputbranchid, checkinnull, checkoutnull);
        END;
        $body$
        LANGUAGE 'plpgsql';
```

b. Call function



```
select * from get_bookings('BR01', 'NULL', 'NULL');
```

Data Output | Messages | Notifications

| | customerid character varying | bookingid character varying | bookingdate timestamp without time zone | guestcount integer | checkin timestamp with |
|---|---|---|---|---|---|
| 1 | CS000080 | BK00004191 | 2024-08-05 00:00:00 | 14 | 2024-08-09 00: |
| 2 | CS000079 | BK00004184 | 2024-10-07 00:00:00 | 4 | 2024-10-12 00: |
| 3 | CS000079 | BK00004174 | 2024-09-01 00:00:00 | 15 | 2024-09-05 00: |
| 4 | CS000079 | BK00004172 | 2024-05-05 00:00:00 | 9 | 2024-05-12 00: |

| | checkout timestamp without time zone | actualcheckin timestamp without time zone | actualcheckout timestamp without time zone | rentak intege |
|---|---|---|---|---|
| 1 | 2024-08-12 00:00:00 | [null] | [null] | |
| 2 | 2024-10-14 00:00:00 | [null] | [null] | |
| 3 | 2024-09-07 00:00:00 | [null] | [null] | |
| 4 | 2024-05-12 00:00:00 | [null] | [null] | |

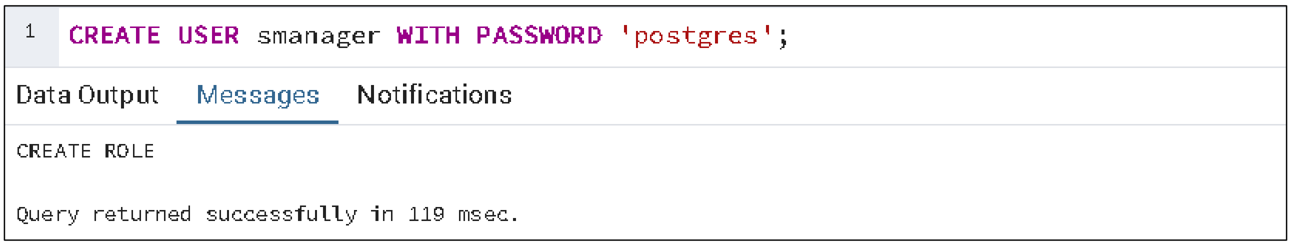
| | rentalcost integer | foodcost integer | totalcost integer | booking_rooms jsonb |
|---|---|---|---|---|
| 1 | 3000 | 0 | 3000 | [{"branchid": "BR01", "roomnumber": "603", "foodconsumed": null}, {"brar |
| 2 | 1170 | 0 | 1170 | [{"branchid": "BR01", "roomnumber": "405", "foodconsumed": null}, {"brar |
| 3 | 2100 | 0 | 2100 | [{"branchid": "BR01", "roomnumber": "500", "foodconsumed": null}, {"brar |
| 4 | 420 | 0 | 420 | [{"branchid": "BR01", "roomnumber": "500", "foodconsumed": null}, {"brar |

| | citizenid character varying | fullname character varying | dateofbirth date | phone character varying | email character varying |
|---|---|---|---|---|---|
| 1 | 079976489134 | Haruno Sakura | 2000-01-01 | 0986118519 | harunosakura@gmail.com |
| 2 | 079637644578 | Uchiha Sasuke | 2000-01-01 | 0988287659 | uchihasasuke@gmail.com |
| 3 | 079637644578 | Uchiha Sasuke | 2000-01-01 | 0988287659 | uchihasasuke@gmail.com |
| 4 | 079637644578 | Uchiha Sasuke | 2000-01-01 | 0988287659 | uchihasasuke@gmail.com |

## PART 3: BUILDING APPLICATIONS (3 points)

### I. Create user

1. Create user

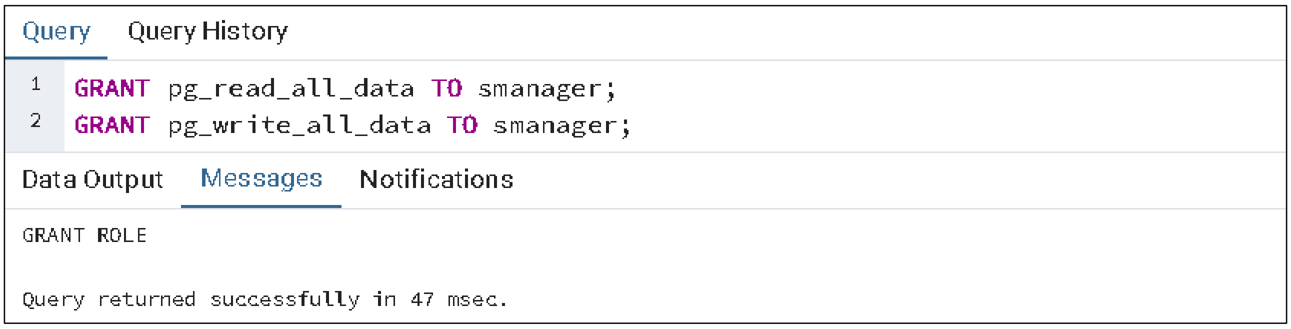


```
CREATE USER smanager WITH PASSWORD 'postgres';
```

Data Output Messages Notifications

```
CREATE ROLE

Query returned successfully in 119 msec.
```

2. Grant privileges



Query Query History

```
GRANT pg_read_all_data TO smanager;
GRANT pg_write_all_data TO smanager;
```

Data Output Messages Notifications

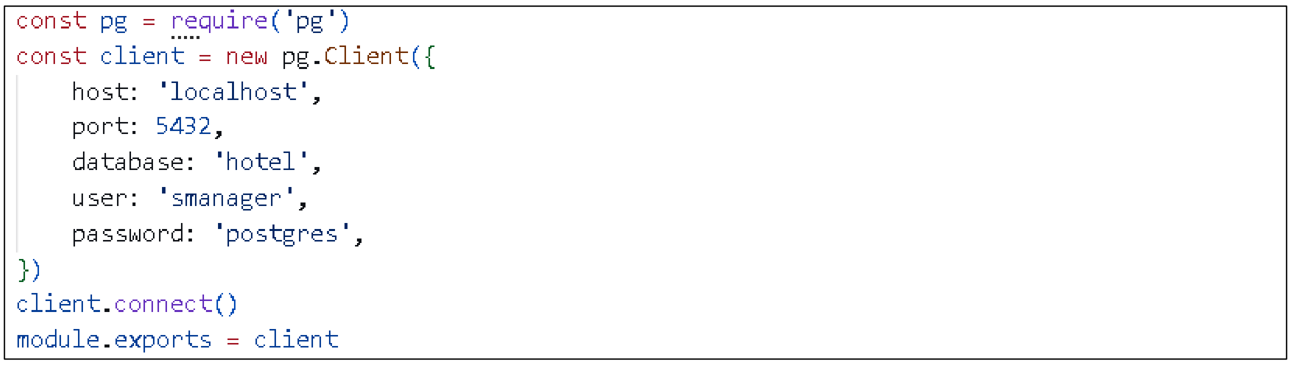
```
GRANT ROLE

Query returned successfully in 47 msec.
```

### II. Programming environment

- For backend, we use `Express.js` – a minimal and flexible Node.js web application framework to build the server
- For frontend, we use `React` along with Ant.Design for styling

### III. How to connect application to PostgreSQL database

To connect to the database, we use `pg` – a database driver that allows our `Express.js` app to interact with PostgreSQL database.



```
const pg = require('pg')
const client = new pg.Client({
    host: 'localhost',
    port: 5432,
    database: 'hotel',
    user: 'smanager',
    password: 'postgres',
})
client.connect()
module.exports = client
```

## IV. Implement features

1. Code for creating new user:

```
hotelRouter.post("/signup", async (request, response) => {
    try {
        let result = await client.query(`
        INSERT INTO customer (CitizenID, FullName, Phone, Email, Username, Password,
        DateOfBirth) VALUES
        ('${request.body.citizenid}','${request.body.fullname}','${request.body.phone}','$
        {request.body.email}','${request.body.username}','${request.body.password}','$
        {request.body.dob}');
        `)
        response.send(result)
    }
    catch (e) {
        console.log("error is:", e)
        response.status(401).json(e.message)
    }
})
```

2. Code for calling a store procedure/ function of Part 2

2.1 Procedure `create_booking`

```
hotelRouter.post("/createbooking", async (request, response) => {
    try {
        let bookingDate = request.body.bookingdate ? `'${request.body.bookingdate}'` : null
        let newQuery = `CALL create_booking('${request.body.guestcount}',${bookingDate},'$
        {request.body.checkin}','${request.body.checkout}', '${request.body.customerid}',
        '${JSON.stringify(request.body.booking_rooms)}');`
        console.log(newQuery)
        let result = await client.query(newQuery)
        response.send({ result: "success", message: result })
    }
    catch (e) {
        console.log("error is:", e)
        response.status(401).json(e.message)
    }
})
```

### 2.2 Procedure `checkout`

```
hotelRouter.post("/checkout", async (request, response) => {
    try {
        let newQuery = `CALL checkout('${request.body.bookingid}','${JSON.stringify
        (request.body.roomlist)}');`
        console.log(newQuery)
        let result = await client.query(newQuery)
        response.send(result)
    }
    catch (e) {
        response.status(401).json(e.message)
    }
})
```

### 2.3 Function `get_vacant_roomtypes`

```
hotelRouter.post("/getvacant", async (request, response) => {
    try {
        let newQuery = `
        SELECT * from get_vacant_roomtypes('${request.body.branchID}','${request.body.
        guestCount}','${request.body.roomCount}','${request.body.checkInDate}','${request.
        body.checkOutDate}');`
        console.log(newQuery)
        let result = await client.query(newQuery)
        response.send(result)
    }
    catch (e) {
        console.log("error is:", e)
        response.status(401).json(e.message)
    }
})
```

### 2.4 Function `get_branch_statistics`

```
hotelRouter.post("/statistics", async (request, response) => {
    try {
        let newQuery = `SELECT * FROM get_branch_statistics('${request.body.branchID}','$
        {request.body.year}');`
        console.log(newQuery)
        let result = await client.query(newQuery)
        response.send(result)
    }
    catch (e) {
        response.status(401).json(e.message)
    }
})
```

3. Additional procedure/ function in DBMS (if any)

3.1 Function `get_rooms_calendar`

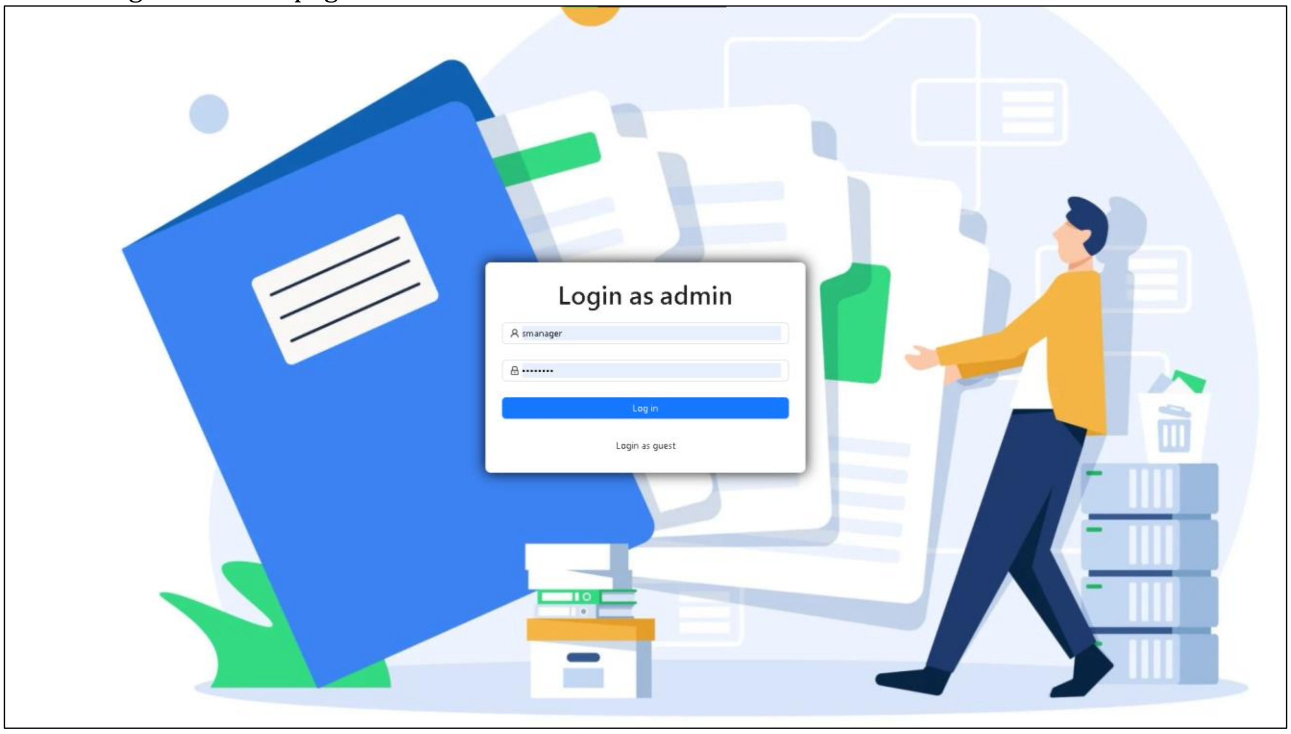
```
hotelRouter.post("/getroomcalendar", async (request, response) => {
    try {
        let newQuery = `select * from get_rooms_calendar('${request.body.branchid}','$
        {request.body.inputyear}','${request.body.inputmonth}');`
        console.log(newQuery)
        let result = await client.query(newQuery)
        response.send(result)
    }
    catch (e) {
        response.status(401).json(e.message)
    }
})
```

3.2 Function `get_bookings`

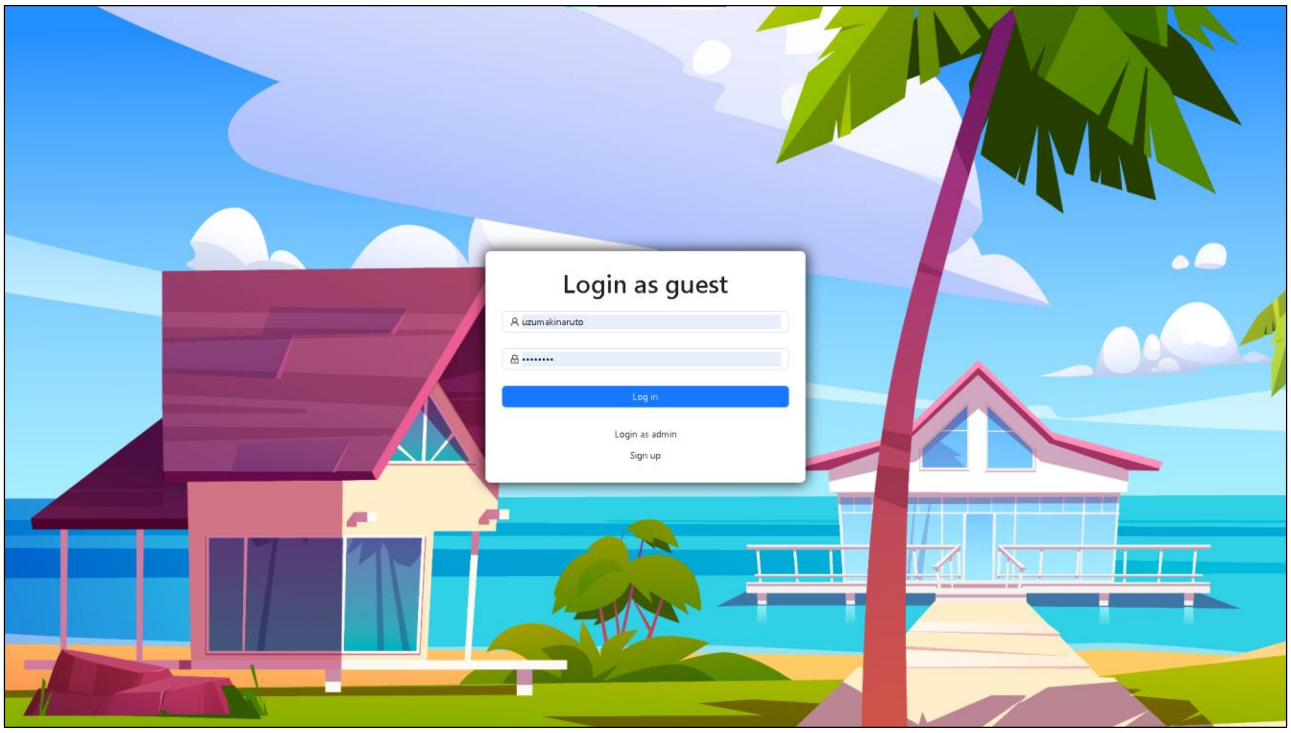
```
hotelRouter.post("/getbookings", async (request, response) => {
    try {
        let newQuery = `select * from get_bookings('${request.body.branchid}', '${request.
        body.checkinnull}', '${request.body.checkoutnull}');`
        console.log(newQuery)
        let result = await client.query(newQuery)
        response.send(result)
    }
    catch (e) {
        console.log("error is:", e)
        response.status(401).json(e.message)
    }
})
```

## V. Application screenshot

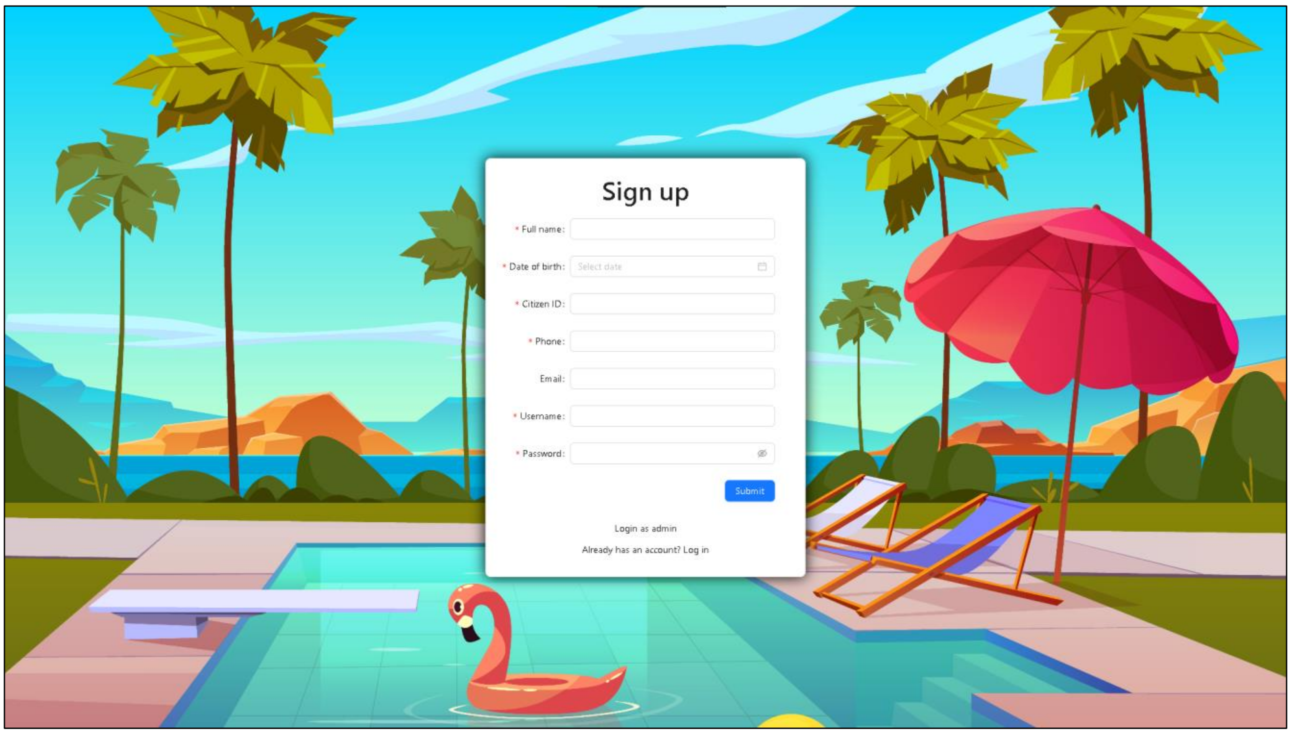
1. Login as admin page



2. Login as guest page



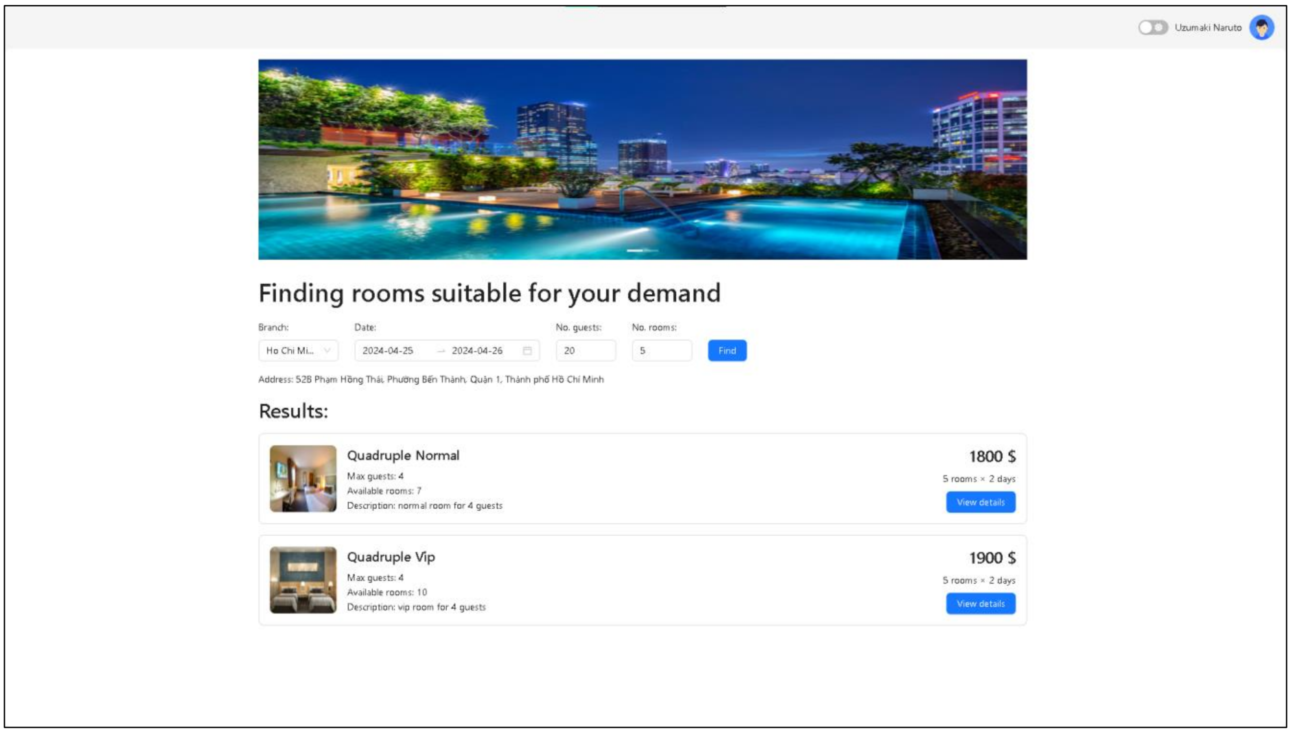
3. Sign up page



## Sign up

| | |
|---|---|
| * Full name: | |
| * Date of birth: | Select date |
| * Citizen ID: | |
| * Phone: | |
| Email: | |
| * Username: | |
| * Password: | |

4. Page for finding vacant rooms:



## Finding rooms suitable for your demand

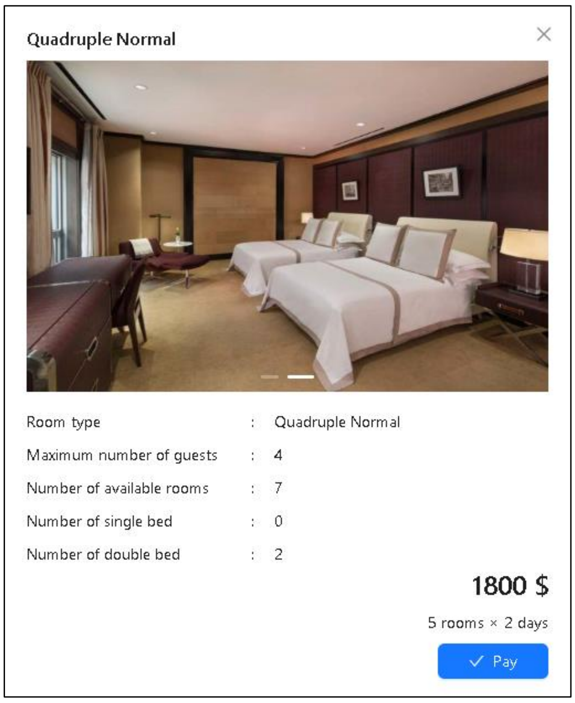
| Branch: | Date: | No. guests: | No. rooms: |
|---|---|---|---|
| Ho Chi Mi... | 2024-04-25 → 2024-04-26 | 20 | 5 |

Address: 52B Phạm Hồng Thái, Phường Bến Thành, Quận 1, Thành phố Hồ Chí Minh

Results:

Quadruple Normal
Max guests: 4
Available rooms: 7
Description: normal room for 4 guests
1800 $
5 rooms × 2 days

Quadruple Vip
Max guests: 4
Available rooms: 10
Description: vip room for 4 guests
1900 $
5 rooms × 2 days



Quadruple Normal

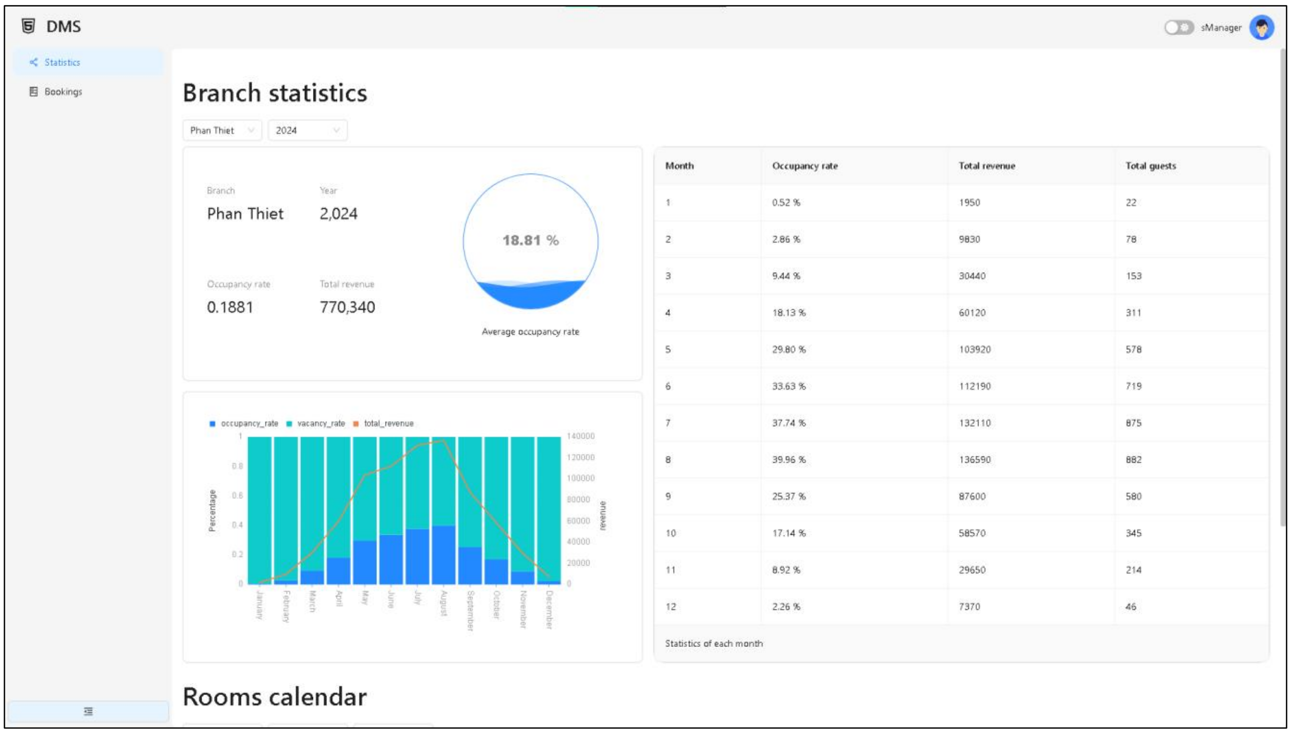
| | | |
|---|---|---|
| Room type | : | Quadruple Normal |
| Maximum number of guests | : | 4 |
| Number of available rooms | : | 7 |
| Number of single bed | : | 0 |
| Number of double bed | : | 2 |

1800 $

5 rooms × 2 days

Modal for booking

5. Statistics page



Branch statistics

Phan Thiet 2024

| Branch | Year |
|---|---|
| Phan Thiet | 2,024 |

| Occupancy rate | Total revenue |
|---|---|
| 0.1881 | 770,340 |

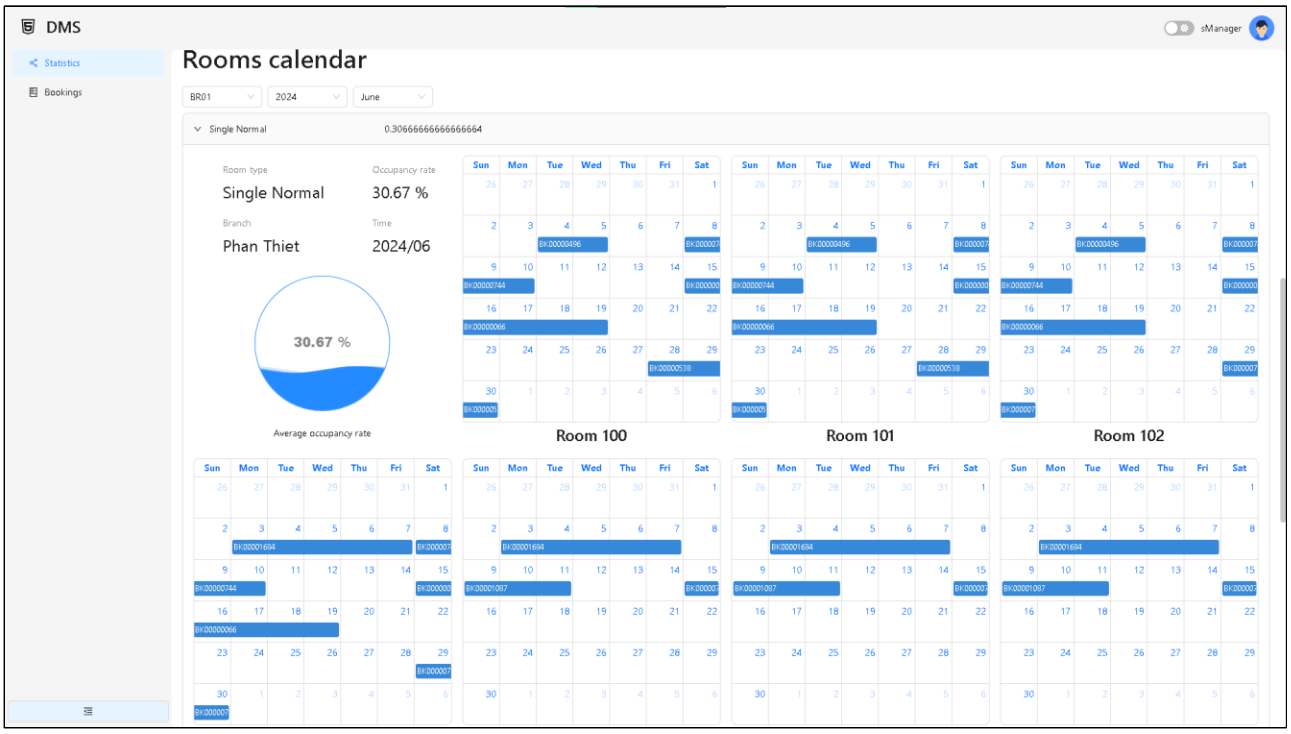
18.81 %

| Month | Occupancy rate | Total revenue | Total guests |
|---|---|---|---|
| 1 | 0.52 % | 1950 | 22 |
| 2 | 2.86 % | 9830 | 78 |
| 3 | 9.44 % | 30440 | 153 |
| 4 | 18.13 % | 60120 | 311 |
| 5 | 29.80 % | 103920 | 578 |
| 6 | 33.63 % | 112190 | 719 |
| 7 | 37.74 % | 132110 | 875 |
| 8 | 39.96 % | 136590 | 882 |
| 9 | 25.37 % | 87600 | 580 |
| 10 | 17.14 % | 58570 | 345 |
| 11 | 8.92 % | 29650 | 214 |
| 12 | 2.26 % | 7370 | 46 |

Statistics of each month

Rooms calendar

Branch statistics



DMS

Rooms calendar

BR01 | 2024 | June

Single Normal 0.30666666666666664

Room type: Single Normal

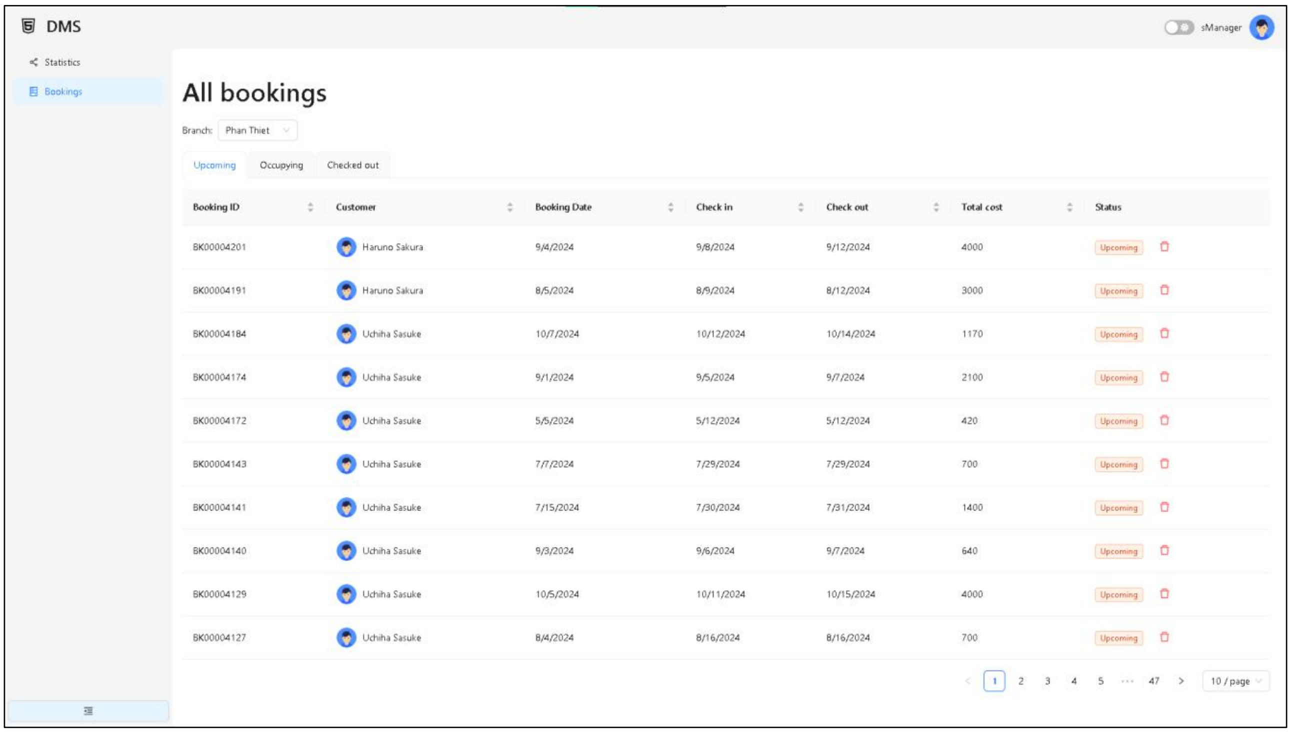
Occupancy rate: 30.67 %

Branch: Phan Thiet

Time: 2024/06

Rooms calendar

6. Bookings management page



DMS

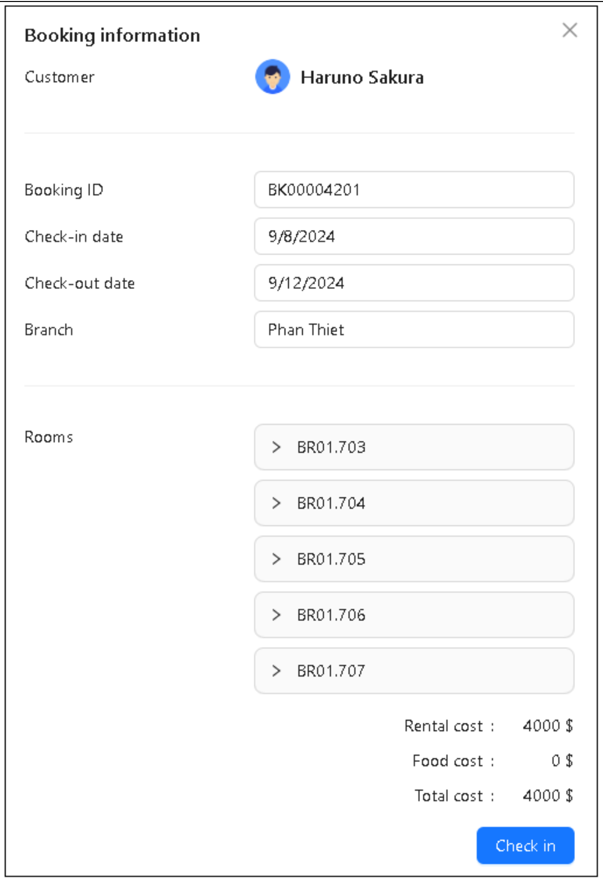
All bookings

Branch: Phan Thiet

Upcoming | Occupying | Checked out

| Booking ID | Customer | Booking Date | Check in | Check out | Total cost | Status |
|---|---|---|---|---|---|---|
| BK00004201 | Haruno Sakura | 9/4/2024 | 9/8/2024 | 9/12/2024 | 4000 | Upcoming |
| BK00004191 | Haruno Sakura | 8/5/2024 | 8/9/2024 | 8/12/2024 | 3000 | Upcoming |
| BK00004184 | Uchiha Sasuke | 10/7/2024 | 10/12/2024 | 10/14/2024 | 1170 | Upcoming |
| BK00004174 | Uchiha Sasuke | 9/1/2024 | 9/5/2024 | 9/7/2024 | 2100 | Upcoming |
| BK00004172 | Uchiha Sasuke | 5/5/2024 | 5/12/2024 | 5/12/2024 | 420 | Upcoming |
| BK00004143 | Uchiha Sasuke | 7/7/2024 | 7/29/2024 | 7/29/2024 | 700 | Upcoming |
| BK00004141 | Uchiha Sasuke | 7/15/2024 | 7/30/2024 | 7/31/2024 | 1400 | Upcoming |
| BK00004140 | Uchiha Sasuke | 9/3/2024 | 9/6/2024 | 9/7/2024 | 640 | Upcoming |
| BK00004129 | Uchiha Sasuke | 10/5/2024 | 10/11/2024 | 10/15/2024 | 4000 | Upcoming |
| BK00004127 | Uchiha Sasuke | 8/4/2024 | 8/16/2024 | 8/16/2024 | 700 | Upcoming |

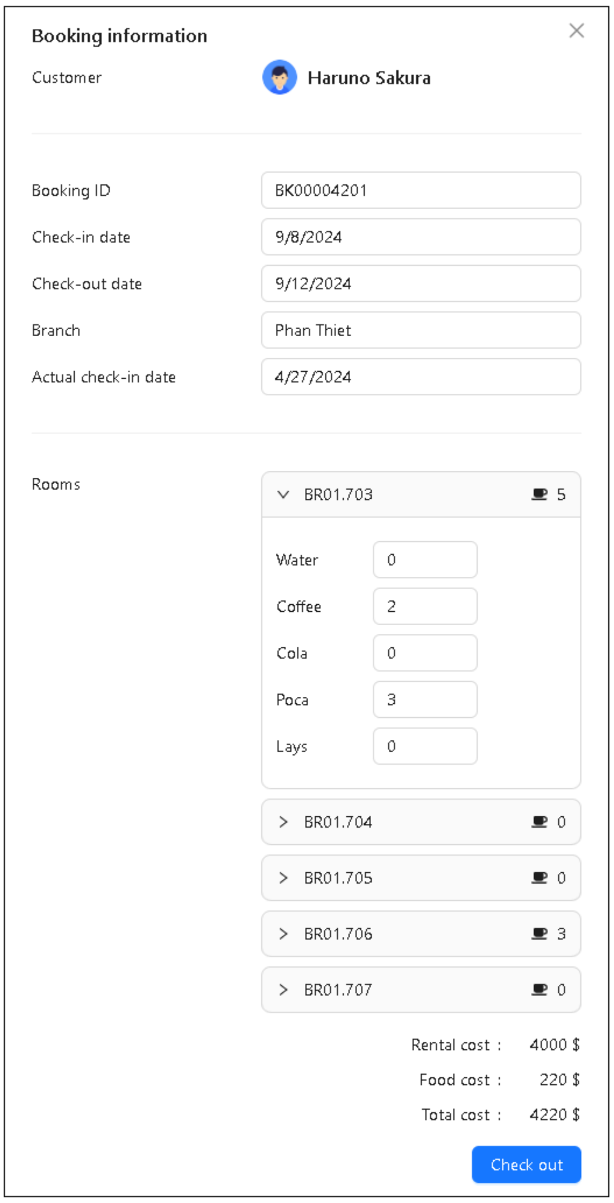
List of bookings



Booking information

| | |
|---|---|
| Customer | Haruno Sakura |
| Booking ID | BK00004201 |
| Check-in date | 9/8/2024 |
| Check-out date | 9/12/2024 |
| Branch | Phan Thiet |
| Rooms | BR01.703 |
| | BR01.704 |
| | BR01.705 |
| | BR01.706 |
| | BR01.707 |

| | |
|---|---|
| Rental cost : | 4000 $ |
| Food cost : | 0 $ |
| Total cost : | 4000 $ |



Booking information

| | |
|---|---|
| Customer | Haruno Sakura |
| Booking ID | BK00004201 |
| Check-in date | 9/8/2024 |
| Check-out date | 9/12/2024 |
| Branch | Phan Thiet |
| Actual check-in date | 4/27/2024 |

Rooms

| | |
|---|---|
| BR01.703 | 5 |
| Water | 0 |
| Coffee | 2 |
| Cola | 0 |
| Poca | 3 |
| Lays | 0 |
| BR01.704 | 0 |
| BR01.705 | 0 |
| BR01.706 | 3 |
| BR01.707 | 0 |

| | |
|---|---|
| Rental cost : | 4000 $ |
| Food cost : | 220 $ |
| Total cost : | 4220 $ |

Modal for Check-in and Check-out

## Submission & Presentation

You should submit your report which preferably in 2-sided black and white on Week 18 as previous announcement.

You don’t have to prepare the slides for presentation, but need to prepare data, scripts to create database and demo the application.