CSI2132[B] Deliverable 2

Group Number: 108
Name: Hao Liu
Student Number: 300381848

Technologies Used

- DBMS: PostgreSQL
- Programming Language: Java
- Tools: Maven, Tomcat, pgAdmin

Installation Guide

1. Clone the Repository

```
git clone git@github.com:hliu315/ehotels.git
cd ehotels
```

2. Set Up the Database:

```
psql -U postgres -d ehotels -f schema.sql
psql -U postgres -d ehotels -f population.sql
```

3. Build and Run the Application:

```
mvn clean package
mvn tomcat7:run
```

4. Access the UI:

- Customer Interface: http://localhost:8080/ehotels/customer/search.jsp

DDLs for database Creation

- Tables

```
CREATE TABLE HotelChain (
    chain_id SERIAL PRIMARY KEY,
    central_address VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20) NOT NULL
);

CREATE TABLE Hotel (
    hotel_id SERIAL PRIMARY KEY,
    chain_id INT NOT NULL REFERENCES HotelChain(chain_id) ON DELETE CASCADE,
    category INT CHECK (category BETWEEN 1 AND 5),
    address VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20) NOT NULL
);

CREATE TABLE Room (
    room_id SERIAL PRIMARY KEY,
    hotel_id INT NOT NULL REFERENCES Hotel(hotel_id) ON DELETE CASCADE,
    price DECIMAL(10,2) CHECK (price > 0),
    amenities TEXT[],
    capacity INT CHECK (capacity > 0),
    sea_view BOOLEAN DEFAULT FALSE,
    mountain_view BOOLEAN DEFAULT FALSE,
    extendable BOOLEAN DEFAULT FALSE,
```

```
    problems TEXT,
    available BOOLEAN DEFAULT TRUE
);

CREATE TABLE Customer (
    customer_id SERIAL PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    id_type VARCHAR(50) NOT NULL,
    id_number VARCHAR(100) UNIQUE NOT NULL,
    registration_date DATE DEFAULT CURRENT_DATE
);

CREATE TABLE Employee (
    employee_id SERIAL PRIMARY KEY,
    hotel_id INT NOT NULL REFERENCES Hotel(hotel_id) ON DELETE CASCADE,
    full_name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    sin VARCHAR(30) UNIQUE NOT NULL,
    role VARCHAR(100) NOT NULL
);

CREATE TABLE Booking (
    booking_id SERIAL PRIMARY KEY,
    customer_id INT NOT NULL REFERENCES Customer(customer_id) ON DELETE CASCADE,
    room_id INT NOT NULL REFERENCES Room(room_id) ON DELETE CASCADE,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    CHECK (start_date < end_date)
);

CREATE TABLE Renting (
    renting_id SERIAL PRIMARY KEY,
    booking_id INT REFERENCES Booking(booking_id) ON DELETE SET NULL,
    customer_id INT NOT NULL REFERENCES Customer(customer_id) ON DELETE CASCADE,
    room_id INT NOT NULL REFERENCES Room(room_id) ON DELETE CASCADE,
    employee_id INT NOT NULL REFERENCES Employee(employee_id) ON DELETE CASCADE,
    check_in_date DATE NOT NULL,
    check_out_date DATE,
    payment_status BOOLEAN DEFAULT FALSE
);
```

- Triggers

```
-- Trigger 1: Prevent double booking
CREATE OR REPLACE FUNCTION check_room_available()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM Booking
        WHERE room_id = NEW.room_id
        AND NOT (NEW.end_date <= start_date OR NEW.start_date >= end_date))
    THEN
        RAISE EXCEPTION 'Room % is already booked for the selected dates.',
NEW.room_id;
    END IF;
    RETURN NEW;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_double_booking
BEFORE INSERT ON Booking
FOR EACH ROW EXECUTE FUNCTION check_room_available();

-- Trigger 2: Ensure each hotel has exactly one manager
CREATE OR REPLACE FUNCTION check_hotel_manager()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.role = 'Manager' AND EXISTS (
        SELECT 1 FROM Employee
        WHERE hotel_id = NEW.hotel_id AND role = 'Manager')
    THEN
        RAISE EXCEPTION 'Hotel % already has a manager.', NEW.hotel_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_single_manager
BEFORE INSERT OR UPDATE ON Employee
FOR EACH ROW EXECUTE FUNCTION check_hotel_manager();
```

- Indexes

```
CREATE INDEX idx_hotel_chain ON Hotel(chain_id);
CREATE INDEX idx_room_price ON Room(price);
CREATE INDEX idx_booking_dates ON Booking(start_date, end_date);
```

- Views

```
-- View 1: Available rooms per area
CREATE VIEW available_rooms_per_area AS
SELECT h.address AS area, COUNT(r.room_id) AS available_rooms
FROM Hotel h
JOIN Room r ON h.hotel_id = r.hotel_id
WHERE r.available = TRUE
GROUP BY h.address;

-- View 2: Aggregated capacity per hotel
CREATE VIEW hotel_capacity AS
SELECT h.hotel_id, SUM(r.capacity) AS total_capacity
FROM Hotel h
JOIN Room r ON h.hotel_id = r.hotel_id
GROUP BY h.hotel_id;
```