Deliverable 2

CSI 2132

Professor: Verena Kantere

Quinn Slavish - 300300282

Suhas Chimmapudi - 300304550

DBMS Used: PostgreSQL (pgAdmin)

Languages used for backend implementation: SQL (pgAdmin), Java (IntelliJ + Tomcat servlet container), Jakarta Server Pages (IntelliJ + Tomcat servlet container) which uses HTML and Javascript.

Steps to guide through the installation process for all these applications:

PGADMIN:https://www.pgadmin.org/download/
Download from the link, select default values for setup, 'port' and 'locale'. Make a password for your pgAdmin. It's important to protect it, but also to use the password as code for your project files to automatically access the databases while coding.

GITHUB: https://gitforwindows.org/
Create an account in github and install git bash for smoother team workflow throughout a project.

INTELLIJ: https://www.jetbrains.com/idea/?var=1
Download for the IDE for an easy work environment between all the jsp, java and the webpage files and links. Along with that, download the Apache Tomcat extension, https://tomcat.apache.org/download-10.cgi

Add it to the run configurations for IntelliJ.

List of DDL statements (this is copied from DeliverableDatabaseDDL.sql):

NOTE: The following DDL lines are copied directly from DeliverableDatabaseDDL.sql.

```
create domain non_neg INT
constraint non_neg_test
check(value >= 0);

create domain non_neg_double NUMERIC(8, 2)
constraint non_neg_payment
check(value >= 0);

CREATE TABLE Person (
   personID VARCHAR(20), PRIMARY KEY(personID),
   personIDType VARCHAR(20) NOT NULL,
```

```
    personFullName VARCHAR(40) NOT NULL,
    personAddress VARCHAR(50)
);

CREATE TABLE HotelChain (
        hotelChainID VARCHAR(50), PRIMARY KEY(hotelChainID),
    addressOfCentralOffices VARCHAR(50)
);

CREATE TABLE Hotel (
    hotelID SERIAL, PRIMARY KEY(hotelID),
    hotelChainID VARCHAR(50), FOREIGN KEY(hotelChainID) REFERENCES
HotelChain(hotelChainID) ON UPDATE CASCADE,
    rating INT check((1 <= rating) and (rating <= 5)),
    hotelAddress VARCHAR(50),
    managerID VARCHAR(20), FOREIGN KEY(managerID) REFERENCES Person(personID)
);

CREATE TABLE HotelRoom (
        roomID SERIAL UNIQUE,
    hotelID non_neg, PRIMARY KEY(hotelID, roomID), FOREIGN KEY(hotelID) REFERENCES
Hotel(hotelID) ON UPDATE CASCADE,
    price non_neg_double NOT NULL,
    amenities VARCHAR(200),
    capacityOfRoom non_neg NOT NULL,
    viewFromRoom VARCHAR(50),
    isExtendable boolean,
    problemsOrDamages VARCHAR(200)
);

CREATE TABLE Booking (
    bookingID SERIAL, PRIMARY KEY(bookingID),
    roomID non_neg, FOREIGN KEY(roomID) REFERENCES HotelRoom(roomID) ON DELETE
SET NULL,
    customerID VARCHAR(20), FOREIGN KEY(customerID) REFERENCES Person(personID)
ON UPDATE CASCADE ON DELETE SET NULL,
    startDate DATE check(startDate < endDate),
    endDate DATE check(endDate > startDate),
    bookingCost non_neg_double NOT NULL,
    bookingStatus VARCHAR(20) check (bookingStatus IN ('BOOKING', 'RENTING',
'ARCHIVED')),
    paymentMethod VARCHAR(20) check (paymentMethod IN ('Credit Card', 'Debit Card',
'In-Person')),
    isPaid BOOLEAN
```

```
);

CREATE TABLE Customer (
   customerID VARCHAR(20), PRIMARY KEY(customerID), FOREIGN KEY(customerID)
REFERENCES Person(personID) ON UPDATE CASCADE ON DELETE CASCADE,
   registrationDate DATE NOT NULL
);

CREATE TABLE Employee (
   employeeID VARCHAR(20), PRIMARY KEY(employeeID), FOREIGN KEY(employeeID)
REFERENCES Person(personID) ON UPDATE CASCADE ON DELETE CASCADE,
   hotelID non_neg, FOREIGN KEY(hotelID) REFERENCES Hotel(hotelID) ON DELETE SET
NULL,
   employeeRole VARCHAR(20)
);

CREATE TABLE HotelEmailAddress (
   emailAddressID non_neg, FOREIGN KEY(emailAddressID) REFERENCES Hotel(hotelID)
ON UPDATE CASCADE ON DELETE CASCADE,
   emailAddressString VARCHAR(50),
   PRIMARY KEY(emailAddressID, emailAddressString)
);
CREATE TABLE HotelChainEmailAddress (
   emailAddressID VARCHAR(50), FOREIGN KEY(emailAddressID) REFERENCES
HotelChain(hotelChainID) ON UPDATE CASCADE ON DELETE CASCADE,
   emailAddressString VARCHAR(50),
   PRIMARY KEY(emailAddressID, emailAddressString)
);

CREATE TABLE HotelPhoneNumber (
   phoneNumberID non_neg, FOREIGN KEY(phoneNumberID) REFERENCES Hotel(HotelID)
ON UPDATE CASCADE ON DELETE CASCADE,
   phoneNumberString VARCHAR(20),
   PRIMARY KEY(phoneNumberID, phoneNumberString)
);
CREATE TABLE HotelChainPhoneNumber (
        phoneNumberID VARCHAR(50), FOREIGN KEY(phoneNumberID) REFERENCES
HotelChain(hotelChainID) ON UPDATE CASCADE ON DELETE CASCADE,
        phoneNumberString VARCHAR(50),
        PRIMARY KEY(phoneNumberID, phoneNumberString)
);

--TRIGGERS
--TRIGGER 1:
```

```
--Function for trigger to check if attempting to delete hotel chains with existing hotels
CREATE FUNCTION check_hotelChain_has_hotels ()
	RETURNS TRIGGER AS
$BODY$
BEGIN
	--Check if there exist hotels of the hotelChain
	IF (SELECT COUNT (hotelID) FROM Hotel WHERE Hotel.hotelChainID =
OLD.hotelChainID) > 0 THEN
		RAISE EXCEPTION 'Cannot delete HotelChain with existing Hotels.';
	END IF;

RETURN OLD;
END
$BODY$ LANGUAGE plpgsql;
--Trigger for checking HotelChain before delete
CREATE TRIGGER check_hotelChain_before_delete
BEFORE DELETE ON HotelChain
FOR ROW
EXECUTE PROCEDURE check_hotelChain_has_hotels();

--TRIGGER 2:
--Function for trigger to check if attempting to delete hotels with existing hotel rooms
CREATE FUNCTION check_hotel_has_rooms ()
	RETURNS trigger AS
	$BODY$
	BEGIN
	--Check if there exist hotels of the hotelChain
	IF (SELECT COUNT (roomID) FROM HotelRoom WHERE HotelRoom.hotelID =
OLD.hotelID) > 0 THEN
		RAISE EXCEPTION 'Cannot delete Hotel with existing Rooms.';
	END IF;

RETURN OLD;
END
$BODY$ LANGUAGE plpgsql;
--Trigger for checking Hotel before delete
CREATE TRIGGER check_hotel_before_delete
BEFORE DELETE ON Hotel
FOR EACH ROW
EXECUTE PROCEDURE check_hotel_has_rooms();

--TRIGGER 3
--Function for trigger to check if a booking is attempting to book an currently used room
CREATE FUNCTION check_booking_creates_conflict ()
```

```
	RETURNS trigger AS
	$BODY$
	BEGIN
	--Check if the new Booking would overlap in time with another Booking
	IF (SELECT COUNT (bookingID) FROM Booking b WHERE b.roomID = NEW.roomID
AND '2024-04-11' BETWEEN '1970-01-01' AND b.endDate) > 0 THEN
		RAISE EXCEPTION 'Booking attempting to be inserted would book the same
room at the same time as another Booking.';
	END IF;

RETURN NEW;
END
$BODY$ LANGUAGE plpgsql;
--Trigger for checking Booked hotel rooms before delete
CREATE TRIGGER check_booking_creates_conflict
BEFORE INSERT ON Booking
FOR EACH ROW
EXECUTE PROCEDURE check_booking_creates_conflict();


--INDEXES

--INDEX 1
--INDEX is over Booking.roomID, Booking.customerID, Booking.bookingStatus
--This index speeds up the checks on Booking of bookingStatus and roomID
CREATE INDEX ON Booking (roomID, customerID, bookingStatus);

--INDEX 2
--INDEX is over Person.personID
--The Person relations is queried (as the parent of Customer and Employee)
--far more often than it is updated by Customer or Employee creation or deletion
CREATE INDEX ON Person (personID);

--INDEX 3
--INDEX is over Customer.customerID
--The Customer relation is queried (primarily during login and sign-up)
--far more often than it is updated by Customer creation or deletion
CREATE INDEX ON Customer (customerID);

--INDEX 4
--INDEX is over Employee.employeeID
--The Employee relation is queried (primarily during login and sign-up)
--far more often than it is updated by Employee creation or deletion
CREATE INDEX ON Employee (employeeID);
```

```
--VIEWS

--VIEW 1
--VIEW provides the number of available rooms per area/street
CREATE VIEW numberOfAvailableHotelRoomsOnSameStreet AS
SELECT REGEXP_SUBSTR(h.hotelAddress, '[A-z]+\s*[A-z]*') AS area, SUM(numberOfRooms)
AS numberOfAvailableRooms
        FROM Hotel h,
   LATERAL(
        SELECT COUNT(*) AS numberOfRooms
                FROM (
                        SELECT *
                        FROM HotelRoom
                        WHERE roomID NOT IN(
                                SELECT roomID
                                FROM Booking
                                WHERE Booking.bookingStatus != 'Archived'
                        )
                ) hR
                WHERE h.hotelID = hR.hotelID
        )
GROUP BY area;

--VIEW 2
--View provides the total capacity of a hotel's rooms alongside the hotel's ID, hotel chain, and
hotel address
CREATE VIEW totalCapacityOfHotels AS
SELECT hotelID, hotelChainID, hotelAddress, SUM(capacityOfRoom) AS hotelCapacity
FROM Hotel NATURAL JOIN HotelRoom
GROUP BY hotelID
ORDER BY hotelCapacity DESC;

--VIEW 3
--View provides a list of the hotels with available rooms
--This query can be further filtered with a WHERE clause to search by area/street
CREATE VIEW hotelsWithAvailableRooms AS
SELECT h.hotelID, h.hotelChainID, h.rating, h.hotelAddress, numberOfAvailableRooms,
h.managerID
FROM Hotel h,
LATERAL(
        SELECT COUNT(*) AS numberOfAvailableRooms
        FROM (
```

```
                SELECT *
                FROM HotelRoom
                WHERE roomID NOT IN(
                        SELECT roomID
                        FROM Booking
                        WHERE Booking.bookingStatus != 'Archived'
                )
        ) hR
        WHERE h.hotelID = hR.hotelID
)
WHERE numberOfAvailableRooms > 0;
```