e-Hotels Report

Group 180: Ahvaanish Kunaratnam, Michael Sandler, James Travers

Submitted March 31, 2025

Part 1: Technologies Used

e-Hotels used a wide array of technologies for its tech stack. For the database, the PostgreSQL database management system was used. The backend was written using the Java programming language. The frontend was developed using HTML. The client-server framework connecting the frontend with the backend was Apache Tomcat.

Part 2: Steps to Install application

1. You have to download (or have) PostgreSQL (Pgadmin4), Intellij IDEA, Apache Tomcat and the zip file with the project in submission.

2. You have to set up the PostgreSQL database with the password admin and according to the ConnectionDB.java file (should be good by default minor changes might need to be made)

3. You must run the e-Hotels sql file then the sample data file in postgresql query tool

4. You must then import the databases folder into intellij

5. Then set up the run configuration for tomcat

6. Then running the tomcat config should produce the local site

For more specific details the method for creating/importing the website was identical to everything done lab7.

Part 3: DDL's

```
CREATE TABLE HotelChain (
   ID serial PRIMARY KEY,
   NumHotels int
);

CREATE TABLE InstCentralOffice (
   HotelChainID int NOT NULL REFERENCES HotelChain(ID),
   Address varchar(20) NOT NULL,
   PRIMARY KEY(HotelChainID, Address)
);

CREATE TABLE ChainPhone (
   HotelChainID int NOT NULL REFERENCES HotelChain(ID),
   PhoneNum bigint NOT NULL CHECK (PhoneNum>=1000000000 AND PhoneNum
<=999999999999999),
```

```
    PRIMARY KEY(HotelChainID, PhoneNum)
);

CREATE TABLE ChainEmail (
    HotelChainID int NOT NULL REFERENCES HotelChain(ID),
    EmailAddr VARCHAR(255) NOT NULL CHECK (EmailAddr SIMILAR TO '%@%.%'),
    PRIMARY KEY (HotelChainID, EmailAddr)
);



CREATE TABLE Hotel (
    HotelAddr varchar(20) PRIMARY KEY,
    NumRooms int,
    City varchar(20),
    StarRating int CHECK (StarRating<= 5 AND StarRating>=0),
    HotelChainID int NOT NULL REFERENCES HotelChain(ID)
);

CREATE TABLE HotelPhone (
    HotelAddr varchar(20) NOT NULL REFERENCES Hotel(HotelAddr),
    PhoneNum bigint NOT NULL CHECK (PhoneNum>=1000000000 AND PhoneNum
<=999999999999999),
    PRIMARY KEY (HotelAddr, PhoneNum)
);

CREATE TABLE HotelEmail (
    HotelAddr varchar(20) NOT NULL REFERENCES Hotel(HotelAddr),
    EmailAddr VARCHAR(255) NOT NULL CHECK (EmailAddr SIMILAR TO '%@%.%'),
    PRIMARY KEY (HotelAddr, EmailAddr)
);

CREATE TABLE Room(
    HotelAddr varchar(20) NOT NULL REFERENCES Hotel(HotelAddr),
    RoomNum int,
    Price decimal(10,2),
    Capacity smallint,
    View varchar(8) CHECK (View in ('Mountain', 'Sea','None')),
    Extendable bool,
    PRIMARY KEY (HotelAddr, RoomNum)
);

CREATE TABLE InstProblem (
    HotelAddr varchar(20) NOT NULL,
```

```
    RoomNum int NOT NULL,
    Type varchar(40),
    PRIMARY KEY (HotelAddr, RoomNum, Type),
        FOREIGN KEY (HotelAddr, RoomNum) REFERENCES Room(HotelAddr, RoomNum)
);

CREATE TABLE InstAmenity (
    HotelAddr varchar(20) NOT NULL,
    RoomNum int NOT NULL,
    Type varchar(40),
    PRIMARY KEY (HotelAddr, RoomNum, Type),
        FOREIGN KEY (HotelAddr, RoomNum) REFERENCES Room(HotelAddr, RoomNum)
);

CREATE TABLE Employee(
    SINOrSSN int PRIMARY KEY CHECK (SINOrSSN >=100000000 AND SINOrSSN<=999999999),
    FirstName varchar(20) NOT NULL,
    MiddleName varchar(20) NOT NULL,
    LastName varchar(20) NOT NULL,
    Address varchar(20) NOT NULL,
    Role varchar(20) NOT NULL,
    HotelAddr varchar(20) NOT NULL REFERENCES Hotel(HotelAddr)
);

CREATE TABLE Manages(
    SINOrSSN int REFERENCES Employee(SINOrSSN),
    HotelAddr varchar(20) REFERENCES Hotel(HotelAddr),
    PRIMARY KEY(SINOrSSN, HotelAddr)
) ;

CREATE TABLE Customer(
    ID varchar(20) PRIMARY KEY,
    FirstName varchar(20) NOT NULL,
    MiddleName varchar(20) NOT NULL,
    LastName varchar(20) NOT NULL,
    Address varchar(20) NOT NULL,
    IDType varchar(20) NOT NULL,
    Date_Reg date NOT NULL
);

CREATE TABLE Booking(
    HotelAddr varchar(20) NOT NULL,
    RoomNum int NOT NULL,
    CheckInDate Date,
```

```
    CustomerID varchar(20) NOT NULL REFERENCES Customer(ID),
    CheckOutDate Date,
    Duration int,
    PRIMARY KEY (HotelAddr, RoomNum, CheckInDate),
        FOREIGN KEY (HotelAddr, RoomNum) REFERENCES Room(HotelAddr, RoomNum)
);


CREATE TABLE Renting(
    HotelAddr varchar(20) NOT NULL,
    RoomNum int NOT NULL,
    CheckInDate Date,
    CustomerID varchar(20) NOT NULL REFERENCES Customer(ID),
    CheckOutDate Date,
    Duration int,
    Balance decimal(10,2),
    EmployeeSINOrSSN int NOT NULL REFERENCES Employee(SINOrSSN),
    PRIMARY KEY (HotelAddr, RoomNum, CheckInDate),
        FOREIGN KEY (HotelAddr, RoomNum) REFERENCES Room(HotelAddr, RoomNum)
);

CREATE TABLE Payment(
    HotelAddr varchar(20) NOT NULL,
    RoomNum int NOT NULL,
    CheckInDate Date NOT NULL,
    CustomerID varchar(20) NOT NULL REFERENCES Customer(ID),
    PaymentAmount decimal(10,2) CHECK (PaymentAmount>0),
    PaymentDate Date NOT NULL,
    PRIMARY KEY(HotelAddr, RoomNum, CheckInDate),
        FOREIGN KEY (HotelAddr, RoomNum, CheckInDate) REFERENCES Renting(HotelAddr,
RoomNum, CheckInDate)
);

CREATE TABLE Archive(
    ArchiveID serial PRIMARY KEY,
    CheckInDate date,
    CheckOutDate date,
    Duration int,
    --Balance decimal(10,2),
    Type char(7) CHECK(Type in ('Renting','Booking'))
);

CREATE FUNCTION add_manager_function()
        RETURNS TRIGGER AS
```

```
$BODY$
BEGIN
        IF NEW.Role='Manager' THEN
   INSERT INTO Manages(SINOrSSN, HotelAddr)
   VALUES (NEW.SINOrSSN, NEW.HotelAddr);
        END IF;
   RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER add_manager
AFTER INSERT ON Employee
FOR EACH ROW
EXECUTE PROCEDURE add_manager_function();

CREATE FUNCTION archive_renting_function()
        RETURNS TRIGGER AS
$BODY$
BEGIN
   INSERT INTO Archive(CheckInDate, CheckOutDate, Duration, Type)
   VALUES (NEW.CheckInDate, NEW.CheckOutDate, NEW.Duration, 'Renting');
   RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER archive_renting
AFTER INSERT ON Renting
FOR EACH ROW
EXECUTE PROCEDURE archive_renting_function();


CREATE FUNCTION archive_booking_function()
        RETURNS TRIGGER AS
$BODY$
BEGIN
   INSERT INTO Archive(CheckInDate, CheckOutDate, Duration, Type)
   VALUES (NEW.CheckInDate, NEW.CheckOutDate, NEW.Duration, 'Booking');
   RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER archive_booking
AFTER INSERT ON Booking
FOR EACH ROW
```

```
EXECUTE PROCEDURE archive_booking_function();


CREATE FUNCTION add_payment_function()
        RETURNS TRIGGER AS
$BODY$
BEGIN
        UPDATE Renting
   SET balance = balance - NEW.PaymentAmount
   WHERE Renting.HotelAddr = NEW.HotelAddr AND Renting.RoomNum = NEW.RoomNum AND
Renting.CheckInDate = NEW.CheckInDate;
        RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER add_payment
AFTER INSERT ON Payment
FOR EACH ROW
EXECUTE FUNCTION add_payment_function();


CREATE FUNCTION add_hotel_function()
        RETURNS TRIGGER AS
$BODY$
BEGIN
        UPDATE HotelChain
   SET NumHotels = NumHotels + 1
   WHERE HotelChain.ID = NEW.HotelChainID;
        RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER add_hotel
AFTER INSERT ON Hotel
FOR EACH ROW
EXECUTE PROCEDURE add_hotel_function();


CREATE FUNCTION remove_hotel_function()
        RETURNS TRIGGER AS
$BODY$
BEGIN
        UPDATE HotelChain
   SET NumHotels = NumHotels - 1
```

```
   WHERE HotelChain.ID = OLD.HotelChainID;
        RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER remove_hotel
AFTER DELETE ON Hotel
FOR EACH ROW
EXECUTE PROCEDURE remove_hotel_function();


CREATE FUNCTION add_room_function()
        RETURNS TRIGGER AS
$BODY$
BEGIN
        UPDATE Hotel
   SET NumRooms = NumRooms + 1
   WHERE Hotel.HotelAddr  = NEW.HotelAddr;
        RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER add_room
AFTER INSERT ON Room
FOR EACH ROW
EXECUTE PROCEDURE add_room_function();


CREATE FUNCTION remove_room_function()
        RETURNS TRIGGER AS
$BODY$
BEGIN
        UPDATE Hotel
   SET NumRooms = NumRooms - 1
   WHERE Hotel.HotelAddr  = OLD.HotelAddr;
        RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER remove_room
AFTER DELETE ON Room
FOR EACH ROW
EXECUTE PROCEDURE remove_room_function();
```

CREATE INDEX idx_room_capacity ON Room(Capacity); -- It is not possible to adjust the capacity of a hotel room. In addition, due to there being a wide range of capacity options (ranging from 1 to 5 guests), indexing can decrease query time by up to 80%.
CREATE INDEX idx_room_view ON Room(View); -- It is not possible to change the view of a hotel room, as it would always face the same direction. In addition, as there are 3 options for views (mountain, sea, none), this index could reduce query time by up to 67%.
CREATE INDEX idx_hotel_city on Hotel(City); -- The hotel cannot change cities without changing addresses (which is also impossible). In addition, there are 4 different cities in the populated database (Ottawa, Kanata, Toronto, Hamliton), this index could reduce query time by up to 75%.

```
CREATE VIEW roomsPerArea AS
SELECT City, SUM(numRooms) as "Number of Rooms Available"
FROM Hotel
GROUP BY City;

CREATE VIEW totalCapacity AS
SELECT HotelAddr, SUM(Capacity) as "Maximum Number of Guests"
FROM Room
GROUP BY HotelAddr;
```