CS166 Final Project - Hotel Management Database

Adrian Monges Rodriguez and Harini Venkatesan
among003 and hvenk001

1. Introduction

In this project, we build a Hotel Management Database with five different entities. These entities include:

a. Hotel
b. Room
c. Staff
d. Customer
e. Maintenance Company

We use the same relational schema given to us with the project. Our contributions to the project are mostly in the java file where we write code to implement 16 different queries. We also used indexes to make our queries efficient.

2. How to run

Start by navigating to the path `CS166/Project/postgresql` starting and creating a database on the terminal by running the following commands:

```
source ./startPostgreSQL.sh
source ./createPostgreDB.sh
```

Then proceed to copy the data files on the path CS166Project/data/ to the database's temporary directory using the following:

```
cp -r /home/csmajs/NETID/CS166Project/data/. /tmp/NETID/myDB/data/
```

Where NETID is your netid (eg: hvenk001)

Once the files are successfully in the database's data folder, navigate to `CS166Project/sql/` and run `create.sql:`

```
psql -h localhost -p $PGPORT $USER"_DB" < create.sql
```

In order to check the performance of our indexes run:

```
psql -h localhost -p $PGPORT $USER"_DB" < createIndexes.sql
```

Finally, run the java file by navigating to `CS166Project/java/`

```
source ./compile.sh
```

3. **Changes and Extensions**

To our DBProject.java file, we added three different functions to provide different outputs to different query execution. The three functions are:

a. executeQueryNoPrint() - this function executes our query without printing the number of rows returned by the query. We used this query to establish new keys such as bookingID or repairID. In our queries IDs for Room, Assignment of staff, repairs, and booking are automatically generated. We implemented this instead of making the user add a new ID in order to avoid conflicts with IDs that already exist.
b. executeQueryReturnData() - this function returns the data returned by the query to assign it to some other local variable within our different query functions.
c. executeQueryReturnDataMultiple() - this function returns data with multiple columns to assign it to some other local variable within our different query functions.

4. **Assumptions**

As a part of implementation of the queries we made the following assumptions:

1. We assumed that in the customer table, each pair of firstname and lastname is unique.

2. For numberOfAvailableRooms() and numberOfBookedRooms(), it returns available and booked rooms for all of time since it does not require a date.
3. We are including a start date and an end date to the top k highest price room in date range.
4. We assumed that the number of available and booked rooms in Booking should not count duplicates.

5. Error checking

a. For all of our INSERT functions such as INSERT into Customer, Booking, Repair etc. we automatically generate the keys or the IDs by adding 1 to the last ID in the table in order to avoid duplicate record creations of the same ID.

b. We made sure to check our date value is entered in the same format as `mm/dd/yyyy` by running a while loop until the correct form of date is entered.

c. To ensure that we do not add duplicate rooms in the `addRoom()` function, we check to make sure that the hotelID added by the user is a valid hotelID in the table Hotel.

d. In the function `addRepair()` we checked if the hotelID and roomID exists within the hotel by checking for it in the Repair table. We also checked if companyID exists in Maintenance Company table.

e. In the function bookRoom() we perform similar error checking methods as addRepair().

f. In the function `assignHouseCleaningToRoom()`, we generated an automatic assignedID and checked for a valid SSN, hotelID and roomID. We also checked if the role of the staff is “HouseCleaning”.

g. In the function `addRepairRequest()`, we check to see if the entered ssn is a manager’s ssn, a valid managerID, repairID in the Repair table and a valid date.

h. For all the get queries, we check to make sure the inputted values by the user exists in the corresponding tables.

We worked on this project together in labs the entire time. We brainstormed on all the queries and error checking together, and we each typed out half the queries. Adrian worked on implementing the indexes. Harini typed out this documentation.