MyBnB

CSCC43 Project

# Description

MyBnB is a re-imagining of the massively popular online sharing economy AirBnB, that aims to provide a quick and efficient way for hosts of locations to connect with potential renters of places of residence. Through ease-of-use for both hosts and renters, MyBnB promotes active sharing and provides relevant avenues to connect these two groups over a common, transparent platform.

Throughout our planning and implementation of MyBnB, we encountered various conceptual problems when designing a database schema. We were able to solve these, using methods in SQL.

One major problem was redundancy. We had to eliminate redundancy in our database relation by using normal forms, but there was more redundancy to solve. For example, addresses are stored for both users and listings of locations, and would be redundant if they were stored separately. We solved this problem by introducing primary keys (such as latitude and longitude) to uniquely identify a tuple, and reduce redundancy by a large amount in our database. We also created our own arbitrary identifiers (using serial values in SQL) to create our own keys for when keys were not obvious. Another example of redundancy is keeping separate log tables when listings or bookings are deleted as it would mean two tables would effectively have the same columns. We solved this problems by introducing our own custom types in the form of enumerations. With enumerations, we simulated restricted strings, keeping constant “status” values.

Another problem was efficiency. Not only would we have to store information that was not excessive, but we would also have to store the correct amounts of information that let us perform all the operations, queries and reports efficiently. For example, we split up information into numerous tables to reduce redundancy, but this posed a problem with efficiency as it meant having to constantly JOIN tables every time we wanted to get information. To solve this, we created appropriate indexes to allow for quick access to the fields that we would select on. We also used views to simplify our database schema and increase efficiency of queries, as our views prepared all the joining of tables without us having to design separate queries to do so.

# Assumptions

We made some key assumptions when designing our database for MyBnB.

1. **A user may be both a host of one listing and a renter of another listing simultaneously.** Essentially, this means that user accounts are generic, and any user can put up a listing for rent, and any user can rent out others’ listings. There is no separate registration or user accounts for “only hosts” or “only renters”. We felt that this was a reasonable assumption to make as many sharing economy platforms, including AirBnB itself, employ this model. In a realistic perspective, the situation of when a person may want to live somewhere else while renting out their own place of stay is not so uncommon, either. This model saves people in situations like these from having to make separate accounts. We made this assumption because we found little reason to separate hosts and renters (i.e. there are no attributes that are not shared by them).
2. **Credit cards are unique to one person each.** This means that two user accounts cannot have the same credit card number and credit card type. This is a reasonable assumption as users sharing credit cards would likely live in the same place together, such as families, and thus would only have one account to rent out places together. We made this assumption for our key of the credit cards relation, which is the credit card number and type.
3. **“Within a year” refers to within last year, and “recently” refers to within the last three months.** This means that when creating the report of total cancellations “within a year,” we only consider the cancellations within the past year. This is likely the most useful information to us, as cancellations before last year are irrelevant. Furthermore, when restricting comments and ratings to users who have rented “recently” from a host, we consider only users who have rented within the last three months. This allows them time to fully rent out a stay for some time (of which the duration may be up to a month), while restricting it so that not a significant amount of time has passed since their stay.
4. **“Distance” refers to the “great circle” distance between two latitudes and longitudes.** This means that we convert latitudes and longitudes to points on a sphere and take the shortest path between them, a straight line. This follows the “Harversine” formula. The Harversine formula assumes the Earth to be a perfect sphere (instead of its true shape, a spheroid). However, distances generated by the formula are still comparable to each other (i.e. two Harversine distances still maintain any inequalities with each other, compared to the true distances). Furthermore, distances only have a margin of error of 0.5%. Therefore, this assumption is still reasonable as we are mostly ordering by distance and calculating vicinities between points, and these operations are mostly accurate under this formula.
5. **Latitudes and longitudes point to one exact address and postal code each.** (The exact address referred to here is omitting unit numbers, such as in condominiums, where all units share the same geographical co-ordinates. Unit numbers can be specified separately.) This is reasonable, especially if latitudes and longitudes go to many decimal points, as geographical co-ordinates will accurately identify a particular exact address and postal code. This assumption allows us to uniquely identify addresses and perform operations.

# Entity-Relationship Model

# Schema

Address(*latitude, longitude, postal\_code, city, province, country, street\_address*)

Users(*SIN, first\_name, last\_name, birth\_date, occupation, email, register\_date\_time, login\_date\_time*)  
UserAddress(*SIN, latitude, longitude*)  
ProfileRatings(*userID, raterID, rating, rating\_date\_time*)  
ProfileComments(*userID, commenterID, content, comment\_date\_time*)

CreditCards(*card\_number, card\_type, expirydate*)  
RenterPayments(*card\_number, card\_type, renterID*)

Listings(*listingID, type, title, description, bedrooms, beds, bathrooms, max\_guests, unit\_number, is\_available, posted\_time*)  
ListingHosts(*listingID*, *hostID*)  
ListingAddress(*listingID, latitude, longitude*)  
ListingRatings(*listingID, raterID, rating, rating\_date\_time*)   
ListingComments(*listingID, commenterID, content, comment\_date\_time*)  
ListingAvailability(*listingID, availabilityID*)  
Amenities(*listingID, type*)

Availability(*availabilityID, type, starts\_on, ends\_on, daily\_price, num\_guests, is\_available*)

Bookings(*bookingID, starts\_on, ends\_on, status, num\_guests, updated\_on*)  
Rentals(*bookingID, renterID*)  
BookedAvailabilities(*bookingID, availabilityID*)  
BookingPayments(*bookingID, card\_number*)

# DDL Statements

The SQL schema was optimized from the relation schema listed above. Many tables were joined where there were “one to many” relationships, and normal form was kept.

# User Manual

# System Limitations