CSCC43 AirBnB Project Report

William Chiu
1006998493

Aug 8, 2023
Prof. Nick Koudas
CSCC43: Introduction to Databases
GitHub Repo Link: https://github.com/williamc99/AirBnB_ProjectC43

Purpose

The purpose of this project was to create an application that is based on the popular property renting website “AirBnB”. Using Java and SQL, I was tasked with designing and implementing an application that can support operations such as creating a user, creating listings, booking listings, and generating reports using the data gathered. By following proper SQL schema practices and making use of the Java JDBC driver, I was able to make use of SQL to effectively and efficiently store all related AirBnB data.

Problems Encountered

Some problems encountered were: “How was I going to efficiently store all the data?”, “Which data types and tables would work best for storing this data?”, “How do I make sure all the tables are in sync?”, and “How do I determine a key for Listings, Bookings, and Reviews?”.

For the first two questions, I found that using five tables: Users, Listings, Bookings, Reviews, and Amenities was the best solution. Users, Listings, and Reviews are pretty self explanatory, as they store what their name suggests. For Amenities, I chose to store these in a separate table because it wouldn’t make sense to store all the Amenity values under each listing, as there were some queries where I had to check whether an Amenity that a user inputted actually existed. This means that having a table of Amenities would be the best solution for this because I could easily check which Amenities exist, and also have the option to add more Amenities in the future if I need to. So this was the most scalable and effective solution.

As for Bookings, I initially decided to use a separate table called “Days” just to record start/end dates and then have them correspond to Bookings or Listings (in other words, start/end dates would be stored in a separate table rather than columns of a table). But I found this approach to be redundant and a lot more unnecessary work/querying. I chose to add start/end dates columns to the Bookings table so that it could record the booking dates, but also I added the “status” and “statusReason” columns to identify whether a booking was “booked”, “canceled”, or “completed”. This also meant that I could set the status as “unavailable” if I wanted to store dates that the host sets as unavailable. So overall, I think that the Bookings table is very versatile and effective at handling dates.

For the third question, to make sure that the tables were in sync, I made use of Foreign Keys and “ON DELETE CASCADE” to make sure that all the tables were using the same listingIDs or bookingIDs, but then also be able to delete any rows automatically if a Foreign Key was deleted in it’s own table.

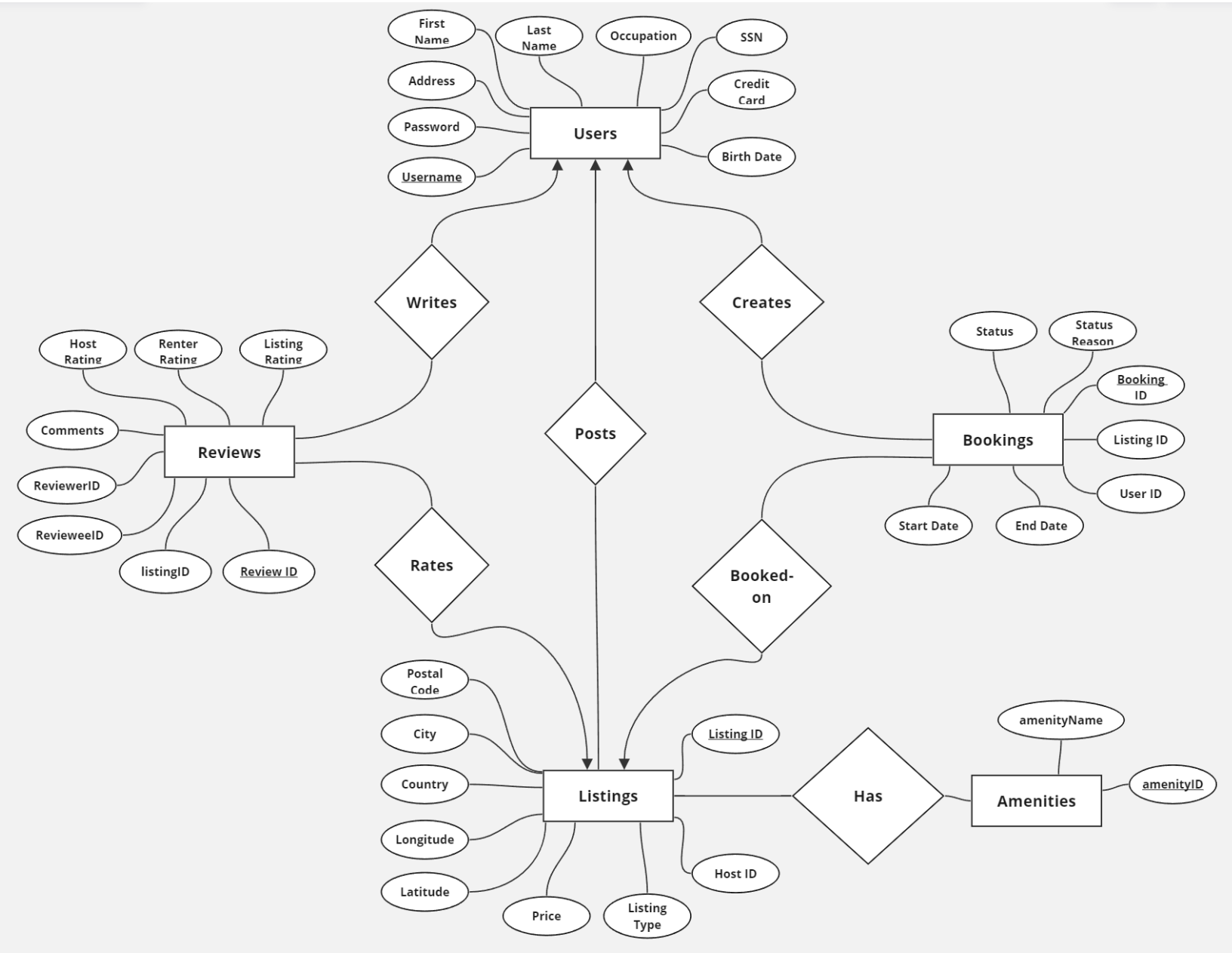
For the last question, I was stumped about how to set unique keys for these tables, especially the Reviews table as there aren’t many attributes that could create a unique key (especially if you consider how reviews can go both ways, i.e renter writes about host, host writes about renter). However I found out about AUTO INCREMENT keys where it will create a unique int key and automatically increment it every time a new row is created. This easily solved my problem and I was able to make use of it for three of my tables.

Assumptions

- Users cannot have more than one account, and that is why I chose not to split up user types to Renter and Host. This means that every user can either host or rent. I did this because on the actual AirBnb website, it is against their policies to have multiple accounts. It would also make sense that hosts are allowed to book other peoples' listings as well
- Assume that a user can only review/comment on a Listing ro Host once. That is, they cannot post multiple reviews for the same listing or host (however, they can write reviews for different listings from the same host)
- Just like the actual AirBnB website, I am assuming that you cannot book only one night. For example, you cannot check in and check out on Jan 15. It has to be check in Jan 15, check out Jan 16. So I am assuming that bookings and listing availability will be in a range rather than singular dates.
- Assume that when a renter rates a listing, they also have to rate the host
- I'm assuming that the users are competent and will not enter the wrong information if the instructions are clearly stated. For example, if the prompt says "Enter a date (YYYY-MM-DD)", they will not enter something like "12/23/2023"
- Assuming that if a user were to ever be deleted, their related bookings, listings and reviews are also deleted, since it wouldn't make sense to have reviews for a user that doesn't exist, it wouldn't make sense to show a listing that doesn't exist anymore
- Assuming that a user cannot change their username after account creation and that username is unique
- Assuming that a listing price can be free
- Assuming that there can't be a home with zero amenities
- Assuming that amenities search filter is Amenity1 OR Amenity2 (like the actual AirBnB website), not Amenity1 AND Amenity2

ER Diagram

Note: There is an online, higher resolution picture in my project repository
In AirBnB_Project/resources/CSCC43 Project ER Diagram.jpg



Relation Schema

Users (username, password, address, first name, last name, occupation, ssn, credit card, birth date**)**
Listings (listingID, hostID, listing type, price, postal code, city, country, longitude, latitude**)**
Bookings (bookingID, listingID, userID, status, status reason, start date, end date**)**
Reviews (reviewID, listingID, reviewerID, revieweeID, comments, host rating, renter rating, listing rating**)**
Amenities (amenityID, amenityName**)**

Creates (username, bookingID**)**
Posts (username, listingID**)**
BookedOn (listingID, bookingID**)**
Has (listingID, amenityID**)**
Rates (reviewID, listingID**)**
Writes(reviewID, username**)**

DDL / Data Files

Note: These files can be found in my repository folder
In: AirBnB_Project/resources/SQL Files

```
DROP TABLE IF EXISTS Users, Listings, Bookings, Reviews, Amenities;

CREATE TABLE Users(
   username varchar(16) PRIMARY KEY NOT NULL,
   password varchar(16) NOT NULL,
   firstName varchar(15) NOT NULL,
   lastName varchar(15) NOT NULL,
   address varchar(25) NOT NULL,
   occupation varchar(20) NOT NULL,
   ssn varchar(9) NOT NULL,
   creditCard varchar(16),
   birthDate date NOT NULL
);

CREATE TABLE Listings(
   listingID int NOT NULL AUTO_INCREMENT,
```

```
    hostID varchar(16) NOT NULL,
    listingType varchar(20) NOT NULL,
    address varchar(25) NOT NULL,
    country varchar(16) NOT NULL,
    city varchar(24) NOT NULL,
    postalCode varchar(10) NOT NULL,
    price decimal(6, 2) NOT NULL,
    longitude decimal(9, 6) NOT NULL,
    latitude decimal(8, 6) NOT NULL,
    amenities varchar(300),
    PRIMARY KEY (listingID),
    FOREIGN KEY (hostID) REFERENCES Users(username) ON DELETE CASCADE,
    CHECK (listingType IN ('House', 'Apartment', 'Guesthouse', 'Hotel')),
    CHECK (price >= 0)
);

CREATE TABLE Bookings(
    bookingID int NOT NULL AUTO_INCREMENT,
    listingID int NOT NULL,
    userID varchar(16) NOT NULL,
    status varchar(16) NOT NULL,
    statusReason varchar(10),
    price float,
    startDate DATE NOT NULL,
    endDate DATE NOT NULL,
    PRIMARY KEY (bookingID),
    FOREIGN KEY (listingID) REFERENCES Listings(listingID) ON DELETE CASCADE,
    FOREIGN KEY (userID) REFERENCES Users(username) ON DELETE CASCADE,
    CHECK (status IN ('completed', 'cancelled', 'unavailable', 'booked')),
    CHECK (statusReason IN ('host', 'renter')),
    CHECK (price >= 0)
);

CREATE TABLE Reviews(
    reviewID int NOT NULL AUTO_INCREMENT,
    reviewerID varchar(16) NOT NULL,
    revieweeID varchar(16) NOT NULL,
    listingID int,
    comment varchar(400),
    hostRating int,
    listingRating int,
    renterRating int,
    PRIMARY KEY (reviewID),
    FOREIGN KEY (reviewerID) REFERENCES Users(username) ON DELETE CASCADE,
    FOREIGN KEY (revieweeID) REFERENCES Users(username) ON DELETE CASCADE,
    FOREIGN KEY (listingID) REFERENCES Listings(listingID) ON DELETE CASCADE,
    CHECK (hostRating >= 1 AND hostRating <= 5),
    CHECK (listingRating >= 1 AND listingRating <= 5),
    CHECK (renterRating >= 1 AND renterRating <= 5)
```

```
);

CREATE TABLE Amenities(
   amenityID int PRIMARY KEY NOT NULL,
   amenityName varchar(26) NOT NULL
);
```

PLEASE NOTE: The Data Script File is in the repository mentioned above, I chose not to include it here as it would take up too many pages.

For information about the application itself, including methods, price algorithms, etc, please refer to the user manual.