MyBnB: Database Design Project

Catherine Sun, Christine Zhao

August 8, 2023

Contents

1 Introduction

1.1 Purpose

AirBnB has had a negative effect on the renting market for residents worldwide. Since hosts would rather receive larger amounts of money by hosting multiple people per year than hosting a single person for a longer duration, there needs to be a way to eat from the monopoly that AirBnB has on the industry, in order to pressure the company to make needed changes to their business model. MyBnB's goal is to improve these issues by acting as a competitor to AirBnB. MyBnB regularly runs reports that aim to weed out the possible commercial hosts, which has negatively impacted the housing market. The goal of this project is to design a database that efficiently and effectively manages listings and bookings of locations worldwide, while also supporting various features such as rating and commenting, user reports and searching.

MyBnB was an opportunity to learn apply years of learning to develop a complete application connected to a relational dataabase, and also learn new java libraries. The ER diagram and schema should avoid redundancy whenever possible, but also be logical enough that SQL queries can be formulated easily. The lessons learned and the successes of this project will aid us in creating future projects along our computer science journey.

1.2 Conceptual Problems Encountered

1.2.1 Differentiate Ratings By Column or Table

Since a renter could provide separate ratings for both the host and the listing, we ran into the problem of creating an ER diagram that allowed differentiating between the three possible ratings for the same booking. Since host ratings, renter ratings and listing ratings have identical attributes, we did not want to create 3 separate entities/tables as it would be redundant. Our solution was to create an extra attribute titled "object", which would theoretical take "host", "renter" and "listing" as possible values. Doing this would make differentiating the ratings the responsibility of the user working with the data instead of being built-in within the schema. This solution prevents redundancy and also allows for easier extension, should we wish to introduce a 4th type of rating.

1.2.2 Foreign Keys And History

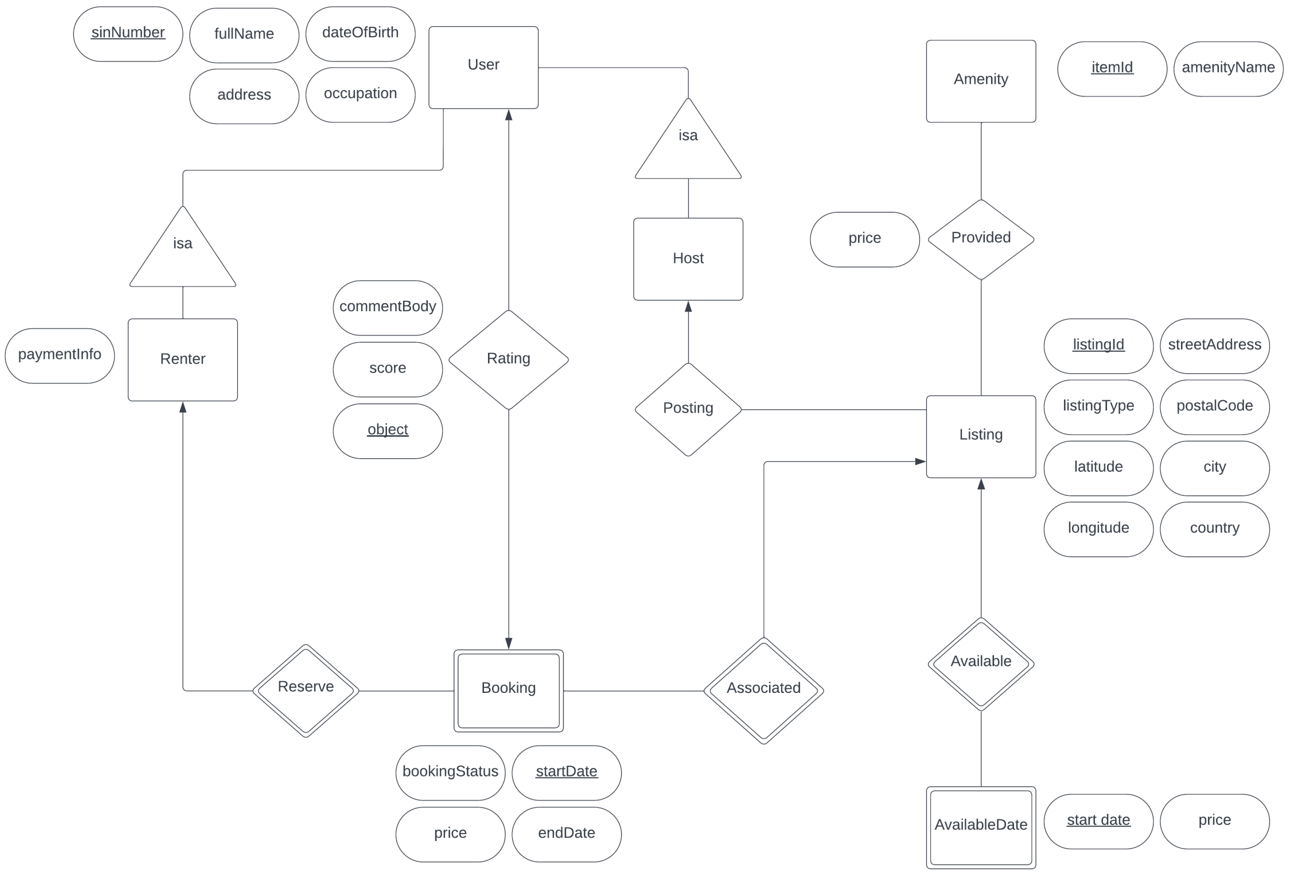
When creating relationships between our different entities, there was a commonly encountered problem regarding how closely the entities should be linked. We wanted to use foreign keys whenever possible into order to reduce redundancy and for a safer schema design. However, in order to support element deletion while preserving much of the history of our database for features such as renter history and various reports, we decided to keep certain conceptually related tables without relationships between them. For example, while our Posting table contains information on host id's and the listing_ids belonging to them, the identical listing_id field in the Listing table is not actually foreign key, as we want to preserve listing history even if the posting is removed. Similar logic is used by not defining a foreign key constraint for listing_id in the Booking table. By keeping host and listing_id information in the Posting table, the removal of both the host account and the listing will have no effect on renter history.

2 Design

2.1 Design Assumptions

- **Booking a range of dates considers the “end date” as date of checkout.** This is consistent with booking AirBnBs and hotels in real life.
- **A host cannot book their own listing.**
- **A user can only have one occupation on their profile.** Having the database store multiple occupations for a user’s profile is unnecessary for the functions of MyBnB.
- **The price of amenities is the daily cost**. Bookings of a longer duration should have a higher cost of amenities than shorter bookings, as the renter will be using them for longer.
- **Setting the price of a listing on a specific date does not include price of amenities**. The total price of a booking, which includes the base price of the dates booked along with the cost of amenities, is calculated upon booking.
- **Using distance to calculate for adjacent postal codes.** If two listings have similar longitudes and latitudes, then their postal codes are adjacent.
- **Amenities are added when creating a listing.** Cannot modify the amenities provided after the listing is up.

2.2 ER Diagram



2.3 Relation Schema

- User(<u>sinNumber</u>, fullName, occupation, address, dateOfBirth)
- Renter(<u>sinNumber</u>, paymentInfo)
- Host(<u>sinNumber</u>)
- Listing(<u>listingId</u>, listingType, latitude, longitude, streetAddress, postalCode, city, country)
- Posting(<u>hostSin, listingId</u>)
- Booking(<u>listingId, renterSin, startDate</u>, endDate, bookingStatus, price)
- Rating(<u>authorSin, renterSin, listingId, startDate, object</u>, commentBody, score)
- Amenity(<u>itemId</u>, amenityName)
- ProvidedAmenity(<u>itemId, listingId</u>, price)
- AvailableDate(<u>listingId, startDate</u>, price)

2.4 Data Definition Language (DDL) Statements

```
CREATE TABLE User (
        sinNumber CHAR(9) PRIMARY KEY,
        fullName VARCHAR(30) NOT NULL,
        occupation VARCHAR(50),
        address VARCHAR(50),
        dateOfBirth DATE
);

CREATE TABLE Renter (
        sinNumber CHAR(9) PRIMARY KEY,
        paymentInfo VARCHAR(30),
        FOREIGN KEY (sinNumber) REFERENCES User(sinNumber) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Host (
        sinNumber CHAR(9) PRIMARY KEY,
        FOREIGN KEY (sinNumber) REFERENCES User(sinNumber) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Listing (
        listingId INTEGER AUTO_INCREMENT PRIMARY KEY,
        listingType VARCHAR(30),
        latitude REAL CHECK (latitude >= -90 AND latitude <= 90),
        longitude REAL CHECK (longitude >= -180 AND longitude <= 180),
        streetAddress VARCHAR(30) NOT NULL,
        postalCode CHAR(6) NOT NULL,
        city VARCHAR(30) NOT NULL,
        country VARCHAR(30) NOT NULL
);

CREATE TABLE Posting (
        hostSin CHAR(9),
        listingId INTEGER,
        PRIMARY KEY(hostSin, listingId),
        FOREIGN KEY (hostSin) REFERENCES User(sinNumber) ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (listingId) REFERENCES Listing(listingId) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Booking (
        listingId INTEGER,
        renterSin CHAR(9),
        startDate DATE,
        endDate DATE NOT NULL,
        bookingStatus VARCHAR(20) NOT NULL,
        price REAL CHECK (price >= 0),
        PRIMARY KEY(listingId, renterSin, startDate),
        FOREIGN KEY (listingId) REFERENCES Listing(listingId) ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (renterSin) REFERENCES User(sinNumber) ON DELETE CASCADE ON UPDATE CASCADE,
        CHECK (startDate <= endDate)
);

CREATE TABLE Rating (
        authorSin CHAR(9),
        renterSin CHAR(9),
        listingId INTEGER,
        startDate DATE,
        commentBody TEXT,
        score INTEGER CHECK (score >= 1 AND score <= 5) NOT NULL,
        object VARCHAR(30),
        PRIMARY KEY (authorSin, renterSin, listingId, startDate, object),
        FOREIGN KEY (listingId) REFERENCES Listing(listingId) ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (renterSin, listingId, startDate)
    REFERENCES Booking(renterSin, listingId, startDate) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Amenity (
        itemId INTEGER AUTO_INCREMENT PRIMARY KEY,
        amenityName VARCHAR(30) UNIQUE
);

CREATE TABLE ProvidedAmenity (
        itemId INTEGER,
        listingId INTEGER,
        price REAL CHECK (price >= 0),
        PRIMARY KEY(itemId, listingId),
        FOREIGN KEY (itemId) REFERENCES Amenity(itemId) ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (listingId) REFERENCES Listing(listingId) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE AvailableDate (
        listingId INTEGER,
        startDate DATE,
        price REAL CHECK (price >= 0),
        PRIMARY KEY(listingId, startDate),
        FOREIGN KEY (listingId) REFERENCES Listing(listingId) ON DELETE CASCADE ON UPDATE CASCADE
```

```
);
```

3 Implementation

3.1 Source Code

GitHub link: https://github.com/catherine-sun/MyBnB

4 User Manual

See separate document for user manual.
Also found at doc/userManual.pdf on the GitHub repo above.

5 Limitations and Future Directions

5.0.1 User interface

The current user interface is not ideal for showing large amounts of data due to difficulty showing columned lists and is lacking emphasizing by bold or italicized text. When displaying listing cards as part of features such as browsing, viewing renting history, or viewing user profiles, each card takes up a large section of the terminal interface, which means long lists require a lot of scrolling. A potential way to improve this is by integrating a GUI module where the user can select actions and perform requests through clicking buttons instead of typing numbers into the command line. This would bring MyBnB even closer to its biggest competitor, AirBnB.

5.0.2 Logging in with SIN

The current system requires the user to enter their SIN to log in, which can be frustrating to remember for some people. An improved system would include creating usernames and passwords to login instead.

5.0.3 Displaying full addresses in searches

This can raise privacy concerns. A way to address this is by hiding street addresses from users until they book the listing.

5.0.4 Searching for nearby listings requires postal code

Requiring postal codes to search for nearby listings may be a drawback as users might only have information related to the city or region they wish to visit and book a listing. To improve this, MyBnB can integrate with geography libraries in order to find adjacent postal codes and nearby towns in a smarter way.

5.0.5 Deleting user accounts removes its bookings

Since the Booking table has a foreign key constraint for the sinNumber attribute in the User table, and it is also part of its primary key, if a user deletes their account, their entire booking history will be deleted from the database. This is not ideal as it removes all of the user's ratings for a host or listing, even though a deleted user does not equate to an invalid rating when looking at it logically. A possible way to improve this is by changing the primary key for the booking table so that it does not use sinNumber, in this way bookings can remain in the table even if its associated sin numbers no longer exist. Perhaps using only 'listingId, startDate' as a primary key may work, if we wish to keep booking history as part of MyBnB for ghost users. We did not use this primary key in our design due to making the assumption that bookings do not need to be saved, since no report requires deleted users.