Name: Satinder Sikand

Question 1:

Hotels: Has a hotel id which is a char 36 UUID. Any char 36 seen is meant to serve as a UUID. The UUID acts as the primary key. A hotel has a name, number of floors, and other relevant information.

Staff: Has the staff id which is a UUID. Staff belong to Hotels so it has a foreign key. Also has its own relevant info.

Room_types: Meant to be a table to hold the definitions of the different kind of rooms. e.g. Luxury, regular, penthouse, etc.

Rooms: The rooms in each Hotels. A room has its own UUID. It belongs to a hotel and is of a certain room type (e.g. bachelors, etc.), so it has a foreign key for the two tables as well. Has its own attributes. A hotel id, room#, and floor is also a unique key which cannot be repeated.

Amenities: The different kind of services available for a room. Think wifi, minibar, etc.

Room_amenities: The amenities a room has. A relationship table for amenities and rooms. Has a foreign key to amenities and rooms.

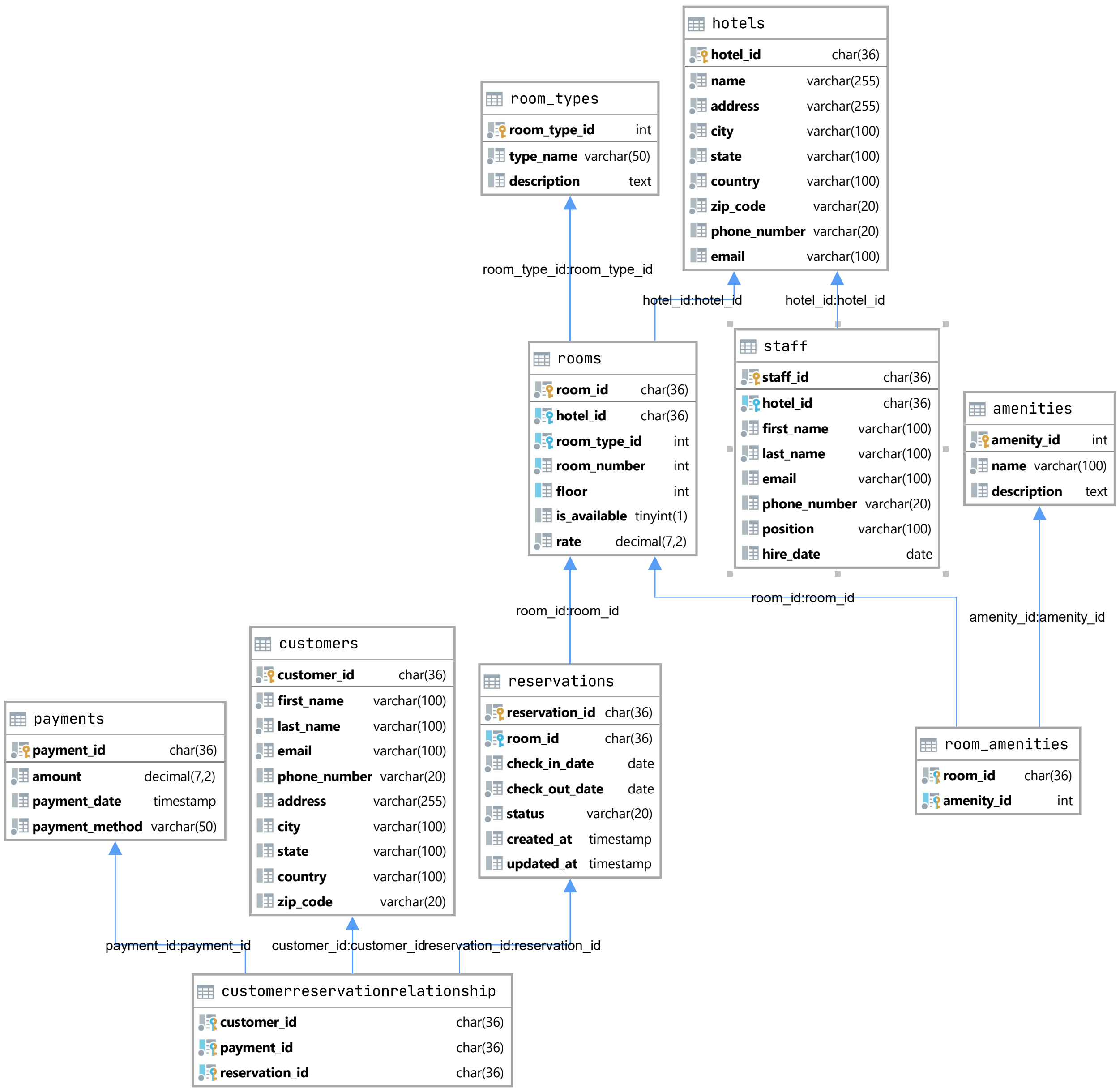
Reservations: Reservations made for a room. Has its own UUID to identify different reservations. A room is reserved so you need a foreign key to room to reference it. Check in and check out attributes are there for when the room is reserved and other attributes that seemed relevant were included.

Payments: Payments made by customers. Has its own UUID. Relevant attributes included as well.

Customers: A table for customers of the hotels. Each customer has their own UUID. Relevant attributes such as names and other information were included.

CustomerReservationRelationship: A relationship table for the payments a customer makes to reserve a room. References the customer id, the payment id, and the reservation id, so it needs references to all three. The combination of the three also makes up the primary key for the table. Relevant information can be gleaned from foreign key tables.

The schema printout can be found on the next page. The code used for the creation of the schema can be found in housing_cloud.sql.


hotels
hotel_id char(36)
name varchar(255)
address varchar(255)
city varchar(100)
state varchar(100)
country varchar(100)
zip_code varchar(20)
phone_number varchar(20)
email varchar(100)
room_types
room_type_id int
type_name varchar(50)
description text
room_type_id:room_type_id
hotel_id:hotel_id
hotel_id:hotel_id
rooms
room_id char(36)
hotel_id char(36)
room_type_id int
room_number int
floor int
is_available tinyint(1)
rate decimal(7,2)
staff
staff_id char(36)
hotel_id char(36)
first_name varchar(100)
last_name varchar(100)
email varchar(100)
phone_number varchar(20)
position varchar(100)
hire_date date
amenities
amenity_id int
name varchar(100)
description text
room_id:room_id
room_id:room_id
amenity_id:amenity_id
customers
customer_id char(36)
first_name varchar(100)
last_name varchar(100)
email varchar(100)
phone_number varchar(20)
address varchar(255)
city varchar(100)
state varchar(100)
country varchar(100)
zip_code varchar(20)
reservations
reservation_id char(36)
room_id char(36)
check_in_date date
check_out_date date
status varchar(20)
created_at timestamp
updated_at timestamp
room_amenities
room_id char(36)
amenity_id int
payments
payment_id char(36)
amount decimal(7,2)
payment_date timestamp
payment_method varchar(50)
payment_id:payment_id
customer_id:customer_id
reservation_id:reservation_id
customerreservationrelationship
customer_id char(36)
payment_id char(36)
reservation_id char(36)

Name: Satinder Sikand

Question 2:

There are two csv files. One is called included_answer.csv and the other removed_answer.csv. The first one contains all the rows which matched the requirements of the header information provided. The second one contains all the rows that did not match the header information/requirements.

Approach:

I first removed duplicate majors in the majors.csv file. Every id in majors was unique, however the name of those majors often repeated themselves and had the same description. No other information could be gleaned from it. I decided to simply use the first occurrence of the major name and ignore the name duplicates that come after for this reason since it felt safe to ignore. I decided not to use the displayIds as the unique major identifier as the given headers in the question said majorIds, which simply the "id" header in majors.csv implied more to me.

As the headers given stated that a majorId is mandatory (no option written as was the case for address), I ignored any persons in the persons.csv that did not have a major. The same reasoning was used to ignore any persons without a bedId. Also, not every personId mapped to a bed in use (bedId), so I only used the ones that did. I did all this by joining the occupancy and inventory csv/tables, and then joining them to the persons table.

Then I threw out repeating emails and kept only the first occurrence of them. This was done as once again, the given headers stated that emails had to be unique. So even though the personIds were unique, they didn't match the requirements and were thrown out. This resulted in 100 rows. An interesting thing to note is that the resulting emails were exactly the number of unique emails there were.

Rows that were missing a bedId and majorId were put into the removed_answer.csv. There were also many rows (nearly half) that did have a bedId and majorId, but they did not have a unique email address and thus did not meet the requirements, so I sent them to the removed_answer.csv as well.

The code used to do this can be found in housing.py.