Account

The Account entity contains a person's information, and the account id is what distinguishes each account.n. There are two types of account: user and hotel_owner.

User and Hotel_Owner

Note an account can be a user, hotel_owner, or both. More specifically, the user role and hotel_owner role for an account determine a set of permissions that we granted to the account user of the FlyNext app. An account with a HOME-OWNER role is an account **with both** user and homeowner permission. Hence, this account will have access to all user-authorized and hotel-owner-authorized endpoints, but not vice versa.

The user entity has an "ISA" relationship with the account entity, containing information about his/her hotel, booking records, and Itinerary info.

The homeowner entity has an "ISA" relationship with the account entity, containing a list of hotels that he/she owns.

Note: Once the user creates a hotel, the user automatically becomes an owner when the user creates a hotel.

To update the user's status, we will create a **new home_owner tuple** that has the same account_id reference as the user object.

We will also **update the role of the account** (id reference both home_owner and user) from "USER" to "HOME-OWNER".

Itinerary

The Itinerary entity contains information about a user's trip details. It has a partial many-to-many relationship with the hotel. An Itinerary can contain multiple hotel bookings (HotelBookingRecord Entity) as well as multiple flight bookings(FlightInfo entity). Hence, it has a many-to-one relationship with the HotelBookingRecord entity and the FlightInfo) entity. It has a total participation, one-to-many relationship with the User.

Hotel

Each hotel has a unique **hotel_id** identifier and has a **foreign key city and country** referencing Location entity. It has a partial, one-to-many relationship with the Hotel_Image entity.

The hotel also has a partial, one-to-one relationship with the owner (HotelOwner), a one-to-many relationship with the HotelRoomType entity a hotel can contain multiple different room types and a partial, one-to-many relationship with the user entity(user books hotel). Each hotel has a location; hence the hotel entity has a one-to-many relationship with the location entity.

Location

We designed Location as an entity since there could be multiple hotels in the same city/location (location is on the many sides of the many-to-one relationship with the hotel entity and the airport entity). The Location will have a city and country attribute as its composite primary key.

Airport

The airport entity has attributes id and code, and the id is the primary key of the Airport entity. The Airport has foreign keys **city** and **country**, which reference the Location entity since each Airport entity has a location.

FlightInfo

The FlightInfo entity contains the last_name and booking_reference attributes as its composite primary key. It also contains the **itinerary id** that associates the flight bookings with the Itinerary entity.

HotelRoomType

We designed HotelRoomType as a weak entity, as different hotels can have the same room type. Hence, we make the hotelId (foreign key referencing Hotel.id) and roomType a composite primary key for uniquely identifying each hotel's room type.

HotelBookingRecord

HotelBookingRecord entity contains a unique identifier hotelBookingId, foreign key userId that references the one user entity, itineraryId that references the specific Itinerary, and hotelRoomType which references one HotelRoomType entity.

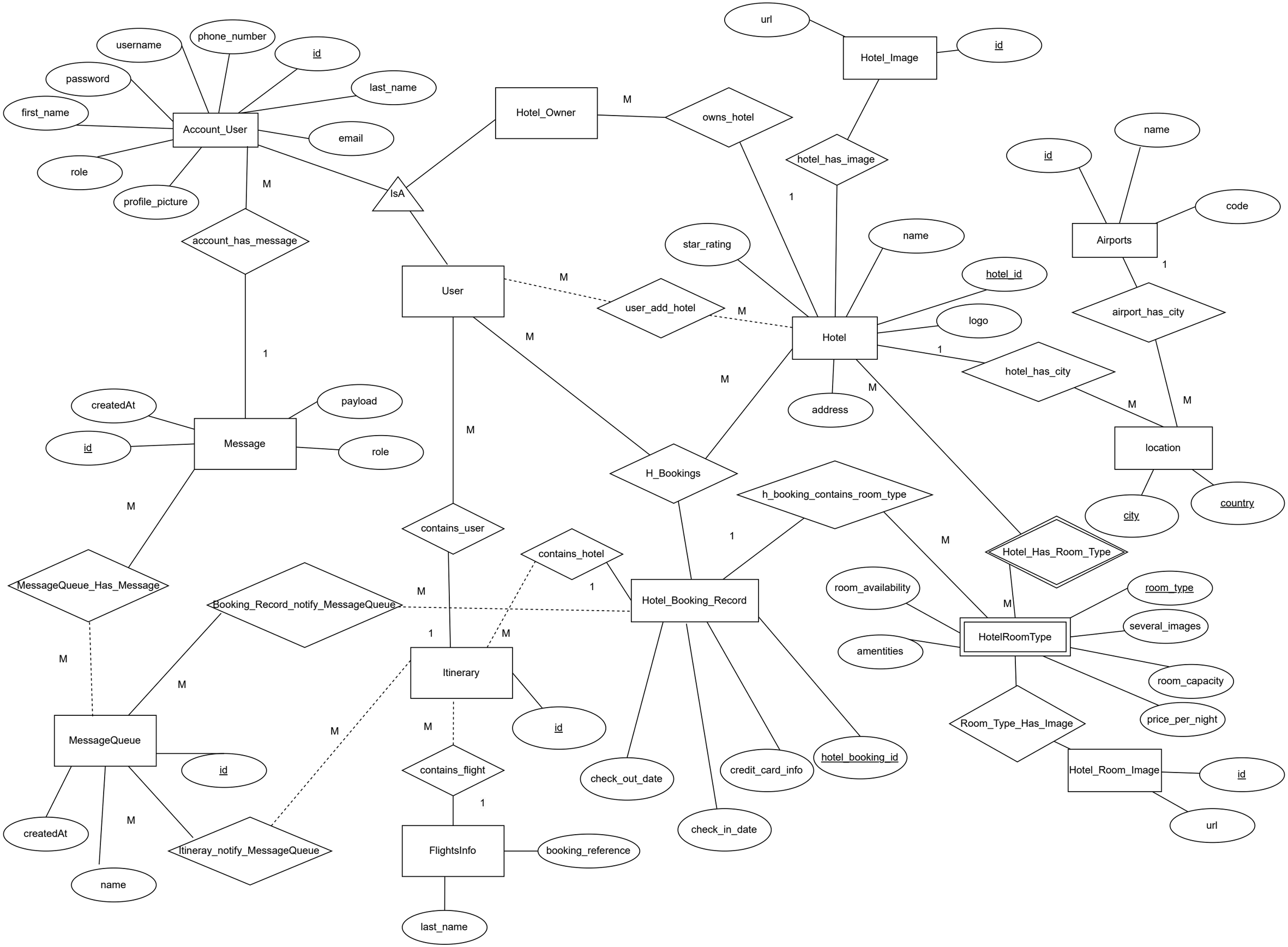
Users have a many-to-one relationship with the HotelBookingRecord entity since users can create many booking records. Note all of these foreign key relationships have one-to-one relationships with HotelBookingRecord.

Flight Info, Airports, and Location

All Flight Info, Airports, and Location entities are created by retrieving information from the AFS system.

MessageQueue

The MessageQueue entity stores the notification for the corresponding user and hotel owner. Each MessageQueue entity object has a unique identifier id, createdAt, a unique attribute name (defined by the type of the messages stored), and a list of Message Entities. There could be many messages related to many Itineraries, and many messages related to many booking records. We define **"HotelBookingRecord"** and **"Itinerary"** as the two types of MessageQueue that we use to store user and hotel-owner notifications.


username
phone_number
id
password
last_name
first_name
Account_User
email
role
profile_picture
M
account_has_message
1
createdAt
payload
id
Message
role
M
MessageQueue_Has_Message
M
MessageQueue
id
createdAt
M
name
Itineray_notify_MessageQueue
Booking_Record_notify_MessageQueue
M
M
IsA
Hotel_Owner
M
owns_hotel
User
M
user_add_hotel
M
M
contains_user
1
Itinerary
id
M
contains_flight
1
FlightsInfo
booking_reference
last_name
contains_hotel
1
M
Hotel_Booking_Record
check_out_date
check_in_date
credit_card_info
hotel_booking_id
H_Bookings
h_booking_contains_room_type
1
M
url
Hotel_Image
id
hotel_has_image
1
star_rating
name
hotel_id
logo
Hotel
1
address
M
M
hotel_has_city
M
M
id
name
Airports
code
1
airport_has_city
location
city
country
Hotel_Has_Room_Type
M
room_availability
amentities
HotelRoomType
room_type
several_images
room_capacity
price_per_night
Room_Type_Has_Image
Hotel_Room_Image
id
url