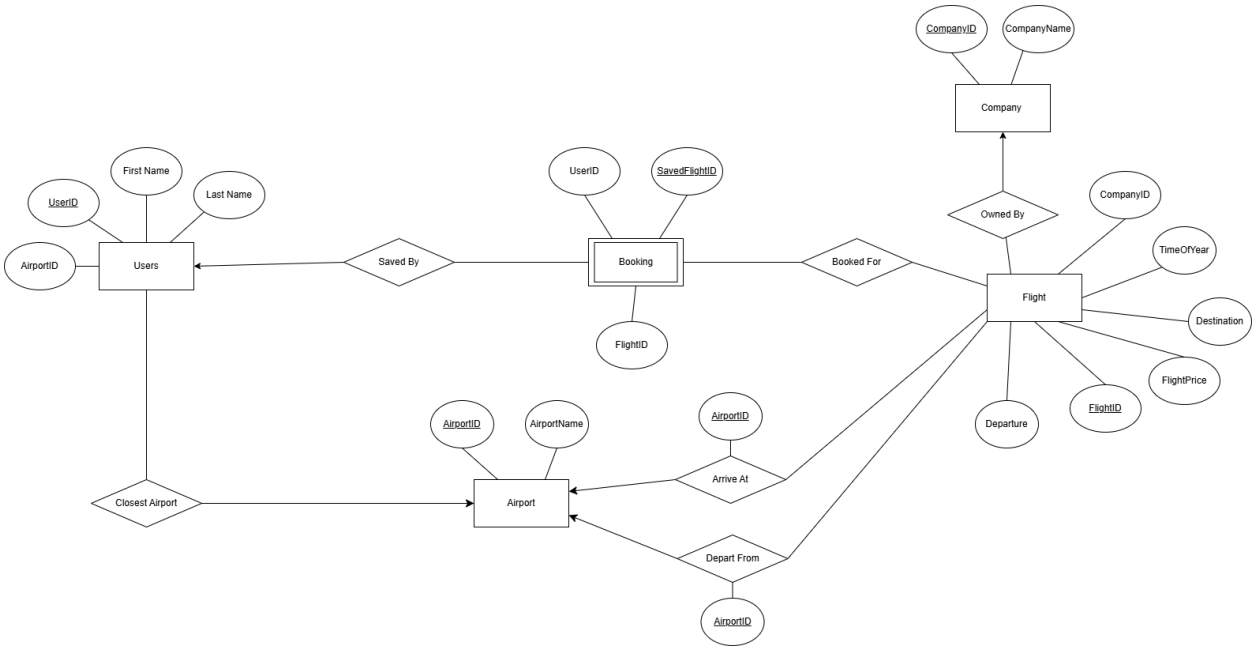
CS 411 - Stage 2: Conceptual and Logical Database Design

ER Model Diagram



Explaining our Model

Users Table:
In this table, UserID is stored as the Primary Key since it is a unique way to represent an individual who wants to have an account in our application. We also save each individual's first and last name since it is needed for their tickets. The AirportID is a foreign key that relates the individual user to their closest airports from which they prefer to fly from. This table is related to the Booking table by the user that saved each booking and to the Airport table by the closest airport to each user. The users table is a distinct entity, rather than an attribute off of the booking since we want user personalization, such as being able to add a favorite airports. Also, it is much easier to organize users based off of a specific key (to prevent mishaps with people of the same name).

Airport Table:
The Airport table uses AirportID as its primary key to uniquely identify the airport by its abbreviated name (AL, BHM, ORD). The other attribute in the table is the airport name which will be used for displaying user-friendly information instead of using airport codes. As mentioned above, this table is related to the Users table though the nearest airport to each user. It is also connected to the Flights table through the specified AirportID, collecting all flights whose departures or destinations go through the specific airport. The airport table is a distinct entity such that we can track ID connections between outgoing and incoming flights later on. This will help find stopover deals for our advanced creative functionality.

Flight Table:
This table will have FlightID as its primary key to uniquely identify a certain flight departing from and arriving at specific AirportIDs (which is a foreign key). We have also added FlightPrice and TimeOfYear so that we are able to identify the times of the year that are the cheapest to fly to a certain destination. Departure will store the airport that users will fly out of, and the desired destination will be stored in Destination. Finally, so customers can find an airline that is the cheapest or preferred, we have added companyID as a foreign key so that this information can be accessed from the company table which the Flights table is linked to. The flight table is a separate entity such that we can easily keep track of each individual flight information to which we can connect to specific users after booking a flight.

Company Table:
This table stores the companyID as a primary key to uniquely identify each aviation company. It also stores CompanyName so that it can be used on the saved flight information in a user-friendly, readable way. The company table is a unique entity such that we can filter by company IDs if a user wants a specific airline. This unique companyIDs are better fitted for filtering rather than a single string attribute since company names can be similar such as AirAsia and AirAsiaX.

Bookings Table:
This is a weak entity since it relies on all other tables' information to store the optimal fight. This means that it does not exist independently. All attributes in this table including UserID, SavedFlightID, and FlightID will build a personalized list of saved, optimal flights for a user so that they can have quick access to their preferred flights. Finally, we added a primary key of SavedFlightID so that each unique saved flight can be identified. The bookings table is a unique entity so that we can uniquely identify flights over the course of time.

Database Relationships
In our database design, we have five database relationships. First, we have the relationship between the User and the Bookings databases. This is a one-to-many relationship since each user has the ability to save multiple flights/bookings (and no specific booking can be shared by more than 1 user). This is represented by the Saved By relation in the design above. Next, there is a one-to-many relationship between the Users and Airport databases. This is because each user is attached to one main airport (via their “base” location and the location of the airport), meaning the airport can be the preferred airport for multiple users. We also have a one-to-many relationship between the Airport and Flight database for both since flights will depart from or arrive at only one airport, but airports can oversee many flights. AirportID is the primary key for both relationships linking to Flight.Departure and Flight.Destination (for example Airport.AirportID=Flight.Departure).The Flight database then links to the Company database in a one-to-many relationship since one aviation company can be linked to multiple flights. Finally, there is a one-to-one relationship between the Booking database and the Flights database since each booking corresponds to one specific flight booked.

Relational Schema:

Users(UserId: INT (PK), AirportID: INT (FK to Airport.AirportID), FirstName: VARCHAR(250), LastName: VARCHAR(250))

Airport(AirportID: INT (PK), AirportName VARCHAR(250))

Flight(FlightID: INT (PK), Departure: INT (FK to Airport.AirportID), Destination: INT (FK to Airport.AirportID), CompanyID: INT (FK to Company.CompanyID), FlightPrice: REAL, TimeOfYear: DATE)

Company(CompanyID: INT (PK), CompanyName: VARCHAR(250))

Booking(SavedFlightID: INT (PK), UserID: INT (FK to Users.UserId), FlightID: INT (FK to Flight.FlightID))

Normalization:

Our Relation Schema is normalized without having to make any changes. Our Users table is normalized since the primary key UserID is a superkey meaning that all non-key attributes are functionally dependent only on the primary key. The same can be said for the Airport, Flights, and Company tables. Since these tables were already normalized there were no changes that needed to be made.

One change we made was to the Booking table. Initially, our schema for Booking was Booking(SavedFlightID: INT (PK), UserID: INT (FK to Users.UserId), FlightID: INT (FK to Flight.FlightID, CompanyID: INT (FK to Company.CompanyID), AirportID: INT (FK to Airport.AirportID). However, we realized that AirportID and CompanyID can be accessed indirectly through Flight's Foreign Keys, so it made more sense to remove these attributes since they can still easily be accessed through FlightID. Thus in all cases every attribute depends on a single key in each entity.