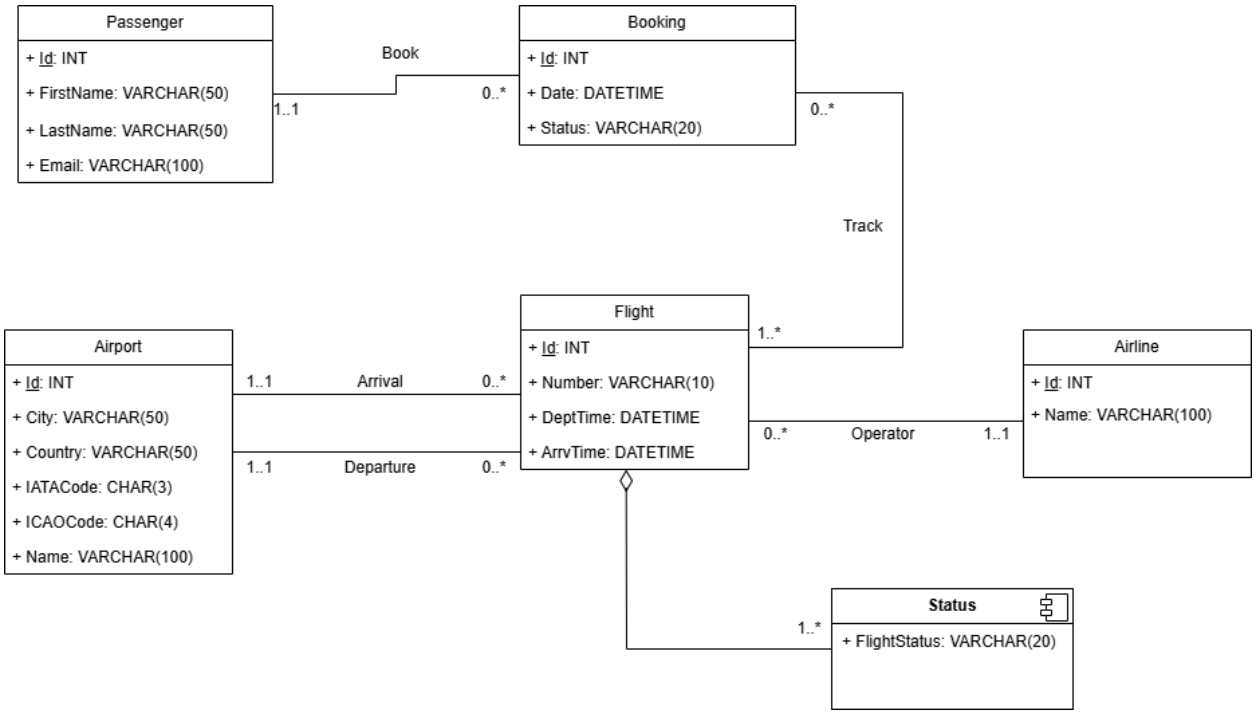
CS 411 Stage2

Project Title: SkyTrack: never miss a flight!

ER/UML diagram



Assumptions:

Flight Assumptions:

1. Each flight has a unique identifier (e.g., flight number).
2. Flight data includes attributes like departure times, taxi times, arrival times, and delays.
3. Every flight can only be operated by a specific airline and flies on a specific route (For example UA xxx can only fly from LAX to SFO in every trip it makes)
4. A flight may be a part of a longer flight by being one of the connecting flights. This ensures the flight still takes the same route, however, can either be the last flight after a connecting flight or be an earlier flight before another connecting flight.
5. A flight belongs to only one airline but can have multiple passengers.

Justification for why we modeled flight as Entity and not attribute:

1. A flight contains multiple independent attributes like departure_time, arrival_time, and status.

2. Airlines operate multiple flights, so Flight should be a separate entity rather than an attribute of Airline.

Airline Assumptions:

1. Each airline has a unique identifier (e.g., airline id).
2. An airline will operate zero or more flights.
3. Performance metrics like on-time departure and arrival percentages, average delay times, and cancellation rates are associated with each airline, which will be needed for the heat map.

Justification for why we modeled airline as Entity and not attribute:

1. Separating it allows us to track airline performance metrics, such as on-time arrival rates, delay trends, and cancellation rates.
2. If an airline changes its name or ceases operations, updating it in a separate table avoids data inconsistencies across multiple flight records.

Airport Assumptions:

1. Each airport has a unique identifier (e.g., airport code).
2. Multiple flights can depart from a specific airport.
3. Multiple flights can arrive at a specific airport.
4. Airports are located in specific cities.

Justification for why we modeled airport as Entity and not attribute:

1. Airports are reusable across multiple flights, meaning that the same airport appears in multiple flights as a departure and arrival location.
2. If an airport updates its name or country (e.g., Istanbul Atatürk Airport changed to Istanbul Airport), updating it in one place ensures data consistency.
3. Separating it allows route-based analysis, such as determining the busiest airports and commonly used flight paths.

Passenger Assumptions:

1. Each passenger has a unique (e.g., passenger_id).
2. A passenger can book multiple flights.

Justification for why we modeled passenger as Entity and not attribute:

1. Passengers can have multiple bookings. If passenger details were embedded in Booking, their details would be duplicated multiple times, leading to redundancy.

2. Passengers are **independent entities**, meaning they may book multiple flights over time, which should be tracked separately.

Booking Assumption:

1. Each booking has a unique booking_id.
2. A booking links one passenger to one or more flights.
3. A passenger can make multiple bookings, and each booking may refer to multiple flights.

Justification for why we modeled booking as Entity and not attribute:

1. Booking is a **bridge entity** that resolves the **many-to-many (M:M) relationship** between Passenger and Flight.
2. It ensures that **each booking is unique** and keeps track of booking_status, booking_date, and seat_numbers separately.

Status Assumption:

1. Each flight will have one and only status

Justification for why we modeled booking as Entity and not attribute:

1. Having status in the flight table violates 3NF since status is also dependent on arrival/departure time, which creates transitive dependency.

<u>Relational Schema:</u>

AIRLINE (Stores airline company information)

Airline(airline_id: INT [PK], airline_name: VARCHAR(100))
{airline_id} -> {airline_name}
Schema is in 3NF because airline_name is dependent on the primary key airline_id.

AIRPORT (Contains airport details)

Airport(airport_id: INT [PK], name: VARCHAR(100), city: VARCHAR(50), country: VARCHAR(50), iata_code: CHAR(3), icao_code: CHAR(4))

{airport_id} -> {name, city, country, iata_code, icao_code}

{city} -> {country}

{iata_code} -> {name, city, country, icao_code. airport_id}

{icao_code} -> {name, city, country, iata_code, airport_id}

Airport id implies city, country, iata_code, name, and icao_code.

City and country implies each other.

Iata_code implies name, city, country icao_code, and airport id

Icao_code implies name, city, country, iata_code, and airport id

FLIGHT (Stores flight information)

Flight(flight_id: INT [PK], airline_id: INT [FK to Airline.airline_id], departure_airport_id: INT [FK to Airport.airport_id], arrival_airport_id: INT [FK to Airport.airport_id], flight_number: VARCHAR(10), departure_time: DATETIME, arrival_time: DATETIME)

{flight_id} ->{airline_id, departure_airport_id, arrival_airport_id, flight_number, departure_time, arrival_time}

Schema is in 3NF because airline_id, departure_airport_id, arrival_airport_id, flight_number, departure_time, arrival_time are dependent on flight_id and there are no transitive dependencies.

Status (Stores flight status)

Status(flight_id: INT [PK], status: VARCHAR(20))

{flight_id} -> {status}

Schema is in 3NF because status is dependent on flight_id and there are no transitive dependencies.

PASSENGER (Stores passenger information)

Passenger(passenger_id: INT [PK], first_name: VARCHAR(50), last_name: VARCHAR(50), email: VARCHAR(100))

{passenger_id} -> {first_name, last_name, email}

Schema is in 3NF because first_name, last_name, email are dependent on passenger_id and there are no transitive dependencies.

BOOKING (Stores booking information)

Booking(booking_id: INT [PK], flight_id: INT [FK to Flight.flight_id], passenger_id: INT [FK to Passenger.passenger_id], booking_date: DATETIME, booking_status: VARCHAR(20))

{booking_id} -> {flight_id, passenger_id, booking_date, booking_status}

Schema is in 3NF because flight_id, passenger_id, booking_date, booking_status are dependent on booking_id and there are no transitive dependencies.

Cardinality:

Entity	**Related Entity**	**Cardinality**	**Explanation**
Airline	Flight	one-to-many	One airline may operate many flights, but each flight is operated by only one airline.
Airport	Flight	one-to-many	An airport may have multiple departing/arrival flights, but each flight departs from/arrives at only one airport.
Flight	Booking	many-to-many	A flight may be tracked by many bookings and each booking may track many flights.
Passenger	Booking	one-to-many	A passenger can have multiple bookings, but each booking belongs to one passenger.
Flight	Status	one-to-one	Each flight can only be associated with one status at a time.