Danny Satterthwaite and Siddhartha Kumar Senthil Kumar

CS 5040 Final Project

Title: Airline Flight and Ticket Scheduling Management System

Purpose: Design a database with the goal of mapping a flight/customer all the way from purchase of the ticket to the moment each plane lands by tracking airline and airport capacity and scheduling constraints.

Description: Millions of people fly each day and a traveler's journey from purchasing their plane ticket to landing at their desired destination is complicated with many moving interconnected and dependent parts. Through the lens of the travelers/passengers, the airlines, and finally the airports, our database is designed to track these processes. Note, we have omitted several non-essential elements of the traveling experience for simplicity such as: buying food/products at the airport, checking a bag, going to a lounge, etc.

User Groups:

1) Booking Agencies
2) Passengers
3) Airline Scheduling Managers
4) Airline Executives
5) Airport Crew
6) Airline Staff
7) Potential travelers

Usage:

1) Booking Agencies
    a) Connects the customer to the flights.
    b) This database could feed the booking agencies showing all of the necessary information (discussed below) to the customers.
    c) The booking agencies need to maintain this connection.
    d) With the information they can keep their platform up to date and potentially offer new features/ services to their users.
2) Passengers
    a) Who buys (or could be refunded for) the tickets for a flight.
    b) This purchasing is fueled by this database system. It tells the passenger (possibly through a booking agency) what flights exist (location and time of departure and arrival) and the flight availability.
    c) Passengers can also book connecting flights.
    d) Passengers can also use the system to get updates on their flight and any potential delays (which our system determines and can show the user).

    e) If a passenger (for whatever valid reason) needs a refund, they can request a refund.
3) Airline Scheduling Managers
    a) Use the system to check how their companies day to day operations are going by looking at the following: departure/arrival delays, gate delays, crew delays, etc.
    b) Using this information they can then take action to fix delays, make customers happy, manage flight attendants and crew.
    c) Understand and monitor booking traffic.
    d) Understand and monitor passenger traffic through their fleet.
4) Airline Executives
    a) In consultation with the airline schedule managers, use this information to make informed decision on the following:
        i) Purchasing new planes
        ii) Buying more gate slots at airports
        iii) Hiring more pilots and/or flight attendants
        iv) Examine potential new routes based on the booking data
5) Airport Crew
    a) Monitor this system to track and update information on the following:
        i) Which runway should the plane take off/land
        ii) Which gate should an arriving plane go to
        iii) Outbound delays (eg: is there a large queue for a runway/planes)
        iv) Inbounds delays (are planes having to delay landing due to disturbances on the ground)
        v) Gate departure delays (did the plane leave their gate later than expected, eg. late crew/ cleaning / passengers)
        vi) Gate arrival delays (eg. did a plane get to its gate on time)
6) Airline Staff (pilots and flight attendants)
    a) Pilots and crew members can check their scheduling software to look at when they might expect to be scheduled to work. All of this information is fed via this database.
    b) They can also access it to look at their own flights and see if everything is on time.
7) Potential Travelers
    a) Look online at either a booking agency or airline website to look at potential flights for upcoming trips. All of this information is tracked and fed via our database.

Constraints and Business Rules:

- Every passenger must book through a booking channel. That is, they must book at the airport (in person via buying a ticket from the counter), through the airline's official website, or through an aggregator booking site like Expedia or a credit card company.
- Every passenger must fly through an airport but not every airport must have a passenger.
- Every passenger must make a booking and every booking must be tied to a passenger

- Every booking is for at least one ticket (by definition of booking) and every ticket must be tied to a booking.
- Every booking has an accompanying transaction but there could be other transactions (eg. buying food) that aren't explicitly tied to a booking, which we are not presenting in our database.
- All transactions must be tied to a passenger and every passenger must have a transaction (the act of going on a flight requires a transaction to occur).
- A ticket allows you to try to go through security (but not every person with a ticket must go through security).
- Airline employees also must go through security to get on a flight.
- All security stations require the passenger to have a ticket.
- Only the passengers that pass through security and are not on a no fly list are permitted to board.
- A passenger that has successfully made it through security could still miss their flight. Thus there are ultimately three groups; Those that make their flight with no issues, those that make it through security but miss their flight due to personal issues, and finally those that miss their flight through schedule conflicts.
- Those that miss their flight can be helped by airline support agents to rebook to another flight.
- Not every flight must have passengers on board. For example, the airline could be relocating the plane to an airline specific flight hub (real world scenario, not necessarily modeled in our project)
- Every flight is either a domestic or an international flight. International flights must go through Customs/Immigration.
- Airlines contain equipment and infrastructure (their trucks, podiums etc) and the employees that work for them.
- There are three categories of airline employees. Support staff (like corporate staff or the gate agents), flight attendants, and pilots.
- Every flight must have flight attendants and pilots, but obviously every single flight attendant and pilot can't work each flight.
- Airports comprise two main sections. The airport employees and the airport infrastructure. The airport infrastructure is divided into three groups. "Other" which could be trucks or the terminals themselves, "gates" which are where airplanes dock so passengers can embark/ disembark, and "runways" which are where the planes land and take off from.
- Every gate doesn't need a flight to land or depart from it.
- Every flight doesn't need to take off (it could be canceled due to weather etc).
- If a flight does take off it must get in the air and every flight that takes off must land at a runway. But, not every runway is required to have a plane land on it.

Conceptual Schema:

- Description:

    - In the broadest terms, our database (and thus schema) comprises three core subschemas: the passengers (blue), the airlines (teal), and the airports (yellow). These colored subschema denote each user group's view. To go on a commercial flight, one must book a ticket. The left portion of the schema focuses on this. It describes how (via the entities and relationships) a passenger can go from booking a flight to boarding it. The middle portion of the schema is the airline subschema which describes how airlines and their workers interact with the airport, flights, and passengers. The right portion of the schema is dedicated to tracking the airport and its workers and infrastructure (gates runways etc). All three subschema come together (intersection) to form a flight that takes off full of customers and lands.
- Additional Notes
    - Due to the complexity and large number of entities and attributes, in talking with Professor Lee, we have omitted the attributes from our schema for clarity and instead have them organized in a list below.
    - *Also note the modification of the Airport entity to incorporate the "has" relationship you suggested in your feedback to step 2.*
- Attribute list
    - → Format**: entities,** *subclasses,* - attributes → attributes are bullets under the **entities** and/or *subclasses*
    - **Booking Channel (*Directly, Booking Site, On Location)***
        - Booking ID
        - Source {One of the subclasses}
    - **Passenger**
        - Passenger ID
        - First name
        - Last name
        - DOB
        - Address
            - door#, city, state, zip, country
        - Phone number
        - Booking ID (another entity connected via a relationship)
    - **Transaction (*Purchase, Refund)***
        - Transaction ID
        - Booking ID
        - Date
        - Time
        - Payment form (miles or credit card)
        - Credit card number if applicable
        - *Refund*
            - Reason
            - Date Issued
    - **Flight Booking**
        - Booking ID

    - Airline
    - Flight number (multi-valued for connecting travelers)
    - Origin (of the whole trip)
    - Destination (of the whole trip)
    - Number of seats (multivalued as many seats (tickets) per booking ex a family)
    - Time
- **Ticket**
    - Ticket #
    - Passenger name
    - Seat number (multi-valued → 1a, 2f, etc)
    - Class (first, business, etc)
    - Number of bags permitted per person
    - Boarding time
    - Gate number
    - Boarding group
    - Airportcode
- **Security (*Free to Fly, On No Fly List)***
    - A weak entity to denote the process of a passenger going through security. Identified by the Passenger ID and Ticket #. Has subclasses for those that make it through or not.
    - TimeThough
    - *On No Fly List*
        - Reason
    - *Free to Fly (Miss Flight, Set to Make Flight)*
        - *Miss Flight*
            - Reason
            - Want to rebook (yes or no)
        - *Set to Make Flight*
            - Represents the subset of passengers that actually make it on their desired plane via boarding.
            - Time
- **Flight (*Domestic, International)***
    - Flight #
    - Flight Date
    - Route # (is it a standard route between two cities we can track)
    - Airline
    - Origin
    - Destination
        - MealProvided
    - Is international (yes no to denote if it's an international flight. Forms the splitting of the subclasses)
        - RequiredExtraSecurity
    - Departure time

- Departure runway
- Arrival time
- Landing runway
- Departure gate
- Arrival gate
- Number of seats (how many could the flight seat if it was full)
- Booked (true or false, can derive from bookings and number of potential seats)
- Employees (list of employee ids for those working the flight)
- Status {On time, delayed, canceled}
- *International (Passed Customs, Not Allowed Entry)*
    - Two subclasses representing passengers that successfully passed through customs vs those that did not. Both have an attribute to hold the name of the agent talked to. Those not allowed entry have the additional attribute of reason why

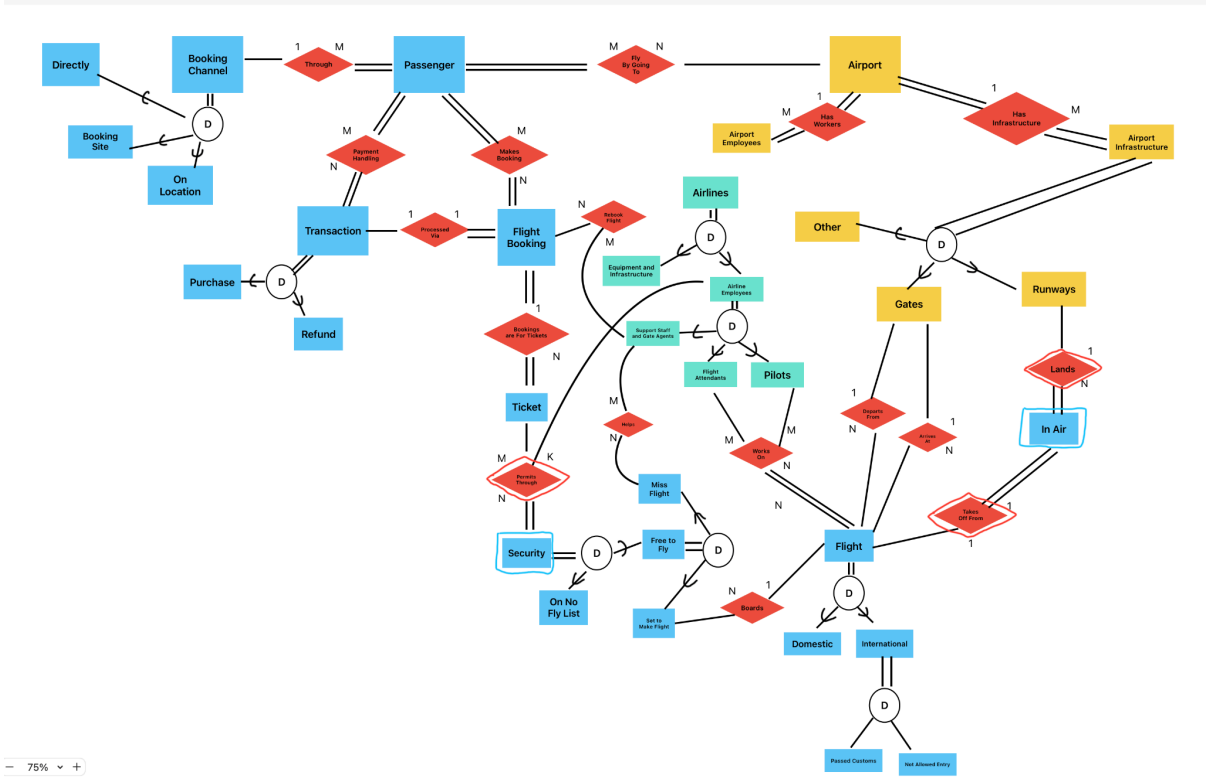
- **Airlines (*Equipment and Infrastructure, Airline Employees)***
    - Company ID
    - Company headquarters location
    - *Equipment and Infrastructure*
        - Item Number
    - *Airline Employees (Support Staff and Gate Agents, Flight Attendants, Pilots)*
        - Employee ID
        - Name
        - *Support Staff and Gate Agents*
            - Title
        - *Flight Attendants*
            - Position
            - Seniority (years on job)
        - *Pilot*
            - Captain (yes or no)
            - Years experience

- **Airport**
    - *Per your feedback we split this to use the relations "has workers" and "has infrastcutrue"*
    - Airport code (ex BTV)
    - Address

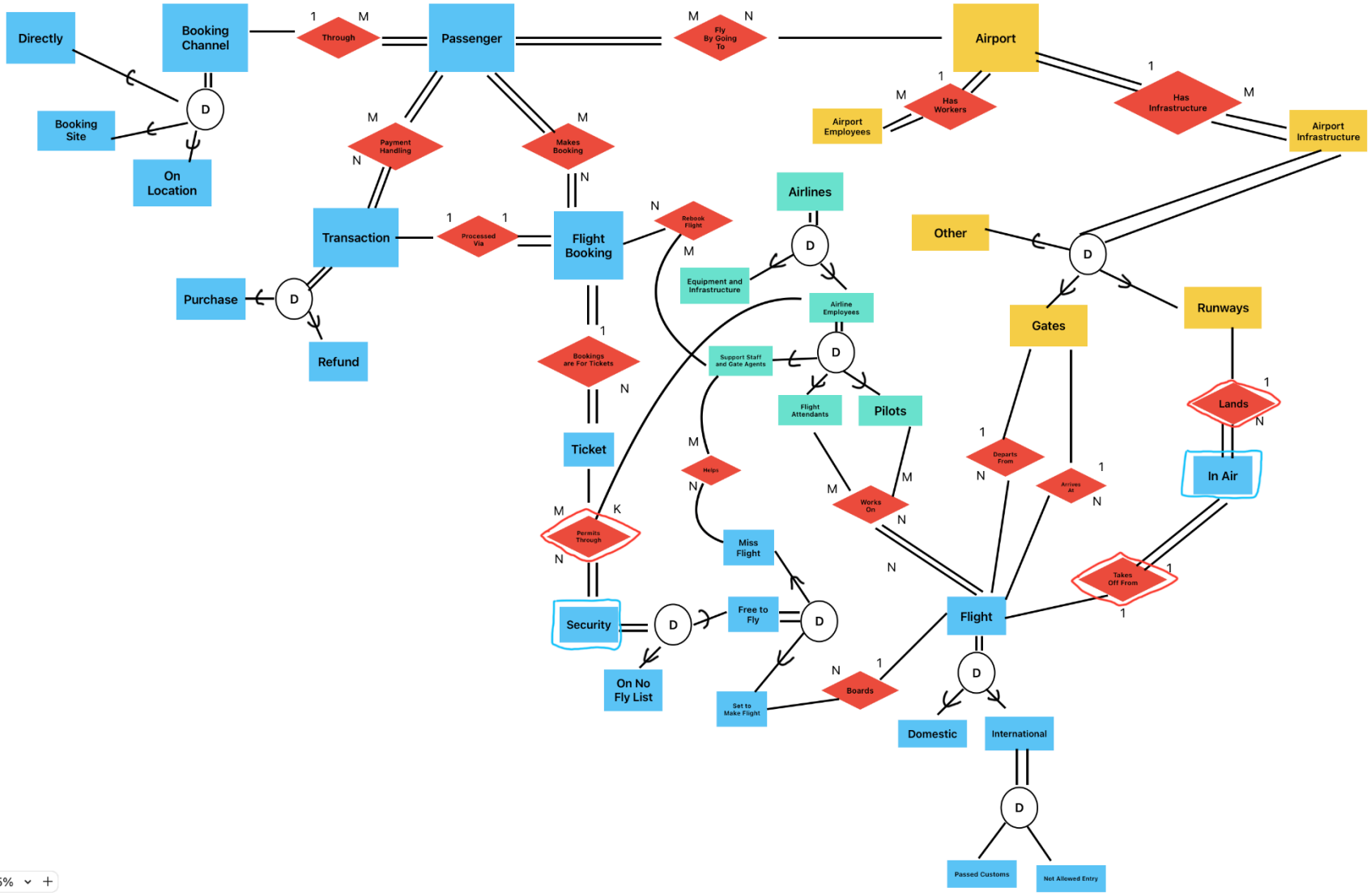
- **Airport Employees**
        - Employee ID
        - Name
- ***Airport Infrastructure (Other, Gates, Runways)***
        - Type (what subclass is it)

        - DatePurchased
        - *Other*
            - Inventory ID number
            - Description
        - *Gates*
            - Gate #
            - Occupied (yes or no)
            - Time
        - *Runways*
            - Runway ID
            - Time
            - In use (yes or no)
    - **In Air**
        - A weak entity is identified by the flight number and runway ID. Represents if a plane actually took off.
        - Time
        - AirportCode

- Schema Visual (See Below)



- Schema Visual Sidways (Same as previous picture just rotated and enlarged)


Directly
Booking Channel
Booking Site
On Location
D
1
M
Through
Passenger
M
N
Fly By Going To
Airport
Airport Employees
M
1
Has Workers
1
Has Infrastructure
M
Airport Infrastructure
M
Payment Handling
N
Makes Booking
N
Transaction
1
1
Processed Via
Flight Booking
N
Rebook Flight
M
Purchase
D
Refund
1
Bookings are For Tickets
N
Ticket
M
K
Permits Through
N
Security
D
On No Fly List
Free to Fly
D
Set to Make Flight
Airlines
D
Equipment and Infrastructure
Airline Employees
Support Staff and Gate Agents
D
Flight Attendants
Pilots
M
Helps
N
Miss Flight
M
Works On
N
N
Other
D
Gates
Runways
1
Lands
N
In Air
1
Departs From
N
1
Arrives At
N
Takes Off From
1
Flight
N
1
Boards
D
Domestic
International
D
Passed Customs
Not Allowed Entry

75%

Part 2: Relational Schema and Additional Constraints

Additional Constraints:

1. BookingChannel (BookingID, Source)
   - Source: Must be one of {‘Directly’, ‘Booking Site’, ‘On Location’}.

2. Source_Directly (BookingID, AirlineName)
   - AirlineName: Must be a valid airline name from the Airlines table.

3. Source_Booking_Site (BookingID, SiteName)
   - SiteName: Must be a valid booking site name.

4. Source_On_Location (BookingID, AirportName)
   - AirportName: Must be a valid airport name from the Airport table.

5. Passenger (PassengerID, Firstname, Lastname, DOB, StreetName, StreetNumber, City, State, Zip, Country, PhoneNumber, BookingID)
   - DOB: Must be a valid date
   - Zip: Must conform to the country's postal code format.
   - PhoneNumber: Must be a valid phone number with appropriate format based on the country.
   - BookingID: Must reference an existing booking.

6. Transaction (TransactionID, BookingID, DateTime, PaymentForm, CreditCardNumber, Refund)
   - DateTime: Must be a valid timestamp, not a future date.
   - PaymentForm: Must be one of {‘Miles’, ‘Credit Card’}.
   - CreditCardNumber: If PaymentForm = 'Credit Card', must be a valid credit card number format.
   - Refund: Can be either ‘Yes’ or ‘No’.

7. Purchase (TransactionID)
   - TransactionID: Must correspond to a valid transaction where Refund = ‘No’.

8. Refund (TransactionID, Reason)

- Reason: Must be a valid predefined reason for refund (e.g., ‘flight cancellation’, ‘change of plans’).

9. FlightBooking (BookingID, Airline, FlightNumber, Origin, Destination, NumberOfSeats, TransactionID)
- Origin: Must reference a valid airport code from the Airport table.
- Destination: Must reference a valid airport code from the Airport table.
- NumberOfSeats: Must be greater than zero and less than or equal to the maximum seats on the flight.

10. Ticket (Ticket#, PassengerName, SeatNumber, Class, NumberOfBags, BoardingTime, GateNumber, BoardingGroup, BookingID)
- Class: Must be one of {‘First’, ‘Business’, ‘Economy’}.
- SeatNumber: Must follow valid seat number format (e.g., '1A', '2F', etc.).
- NumberOfBags: Must be a non-negative integer.
- BoardingTime: Must be a valid timestamp before the flight departure time.
- GateNumber: Must reference a valid gate number in the airport.

11. Security (PassengerID, TicketNumber)
- PassengerID: Must reference an existing passenger.
- TicketNumber: Must reference an existing ticket.

12. On_No_Fly_List (PassengerID, TicketNumber, Reason)
- Reason: Must be a valid reason for the passenger being on the no-fly list.

13. Free_To_Fly (PassengerID, TicketNumber, MakeFlight?)
- MakeFlight?: Must be a boolean (true or false).

14. Misses_Flight (PassengerID, TicketNumber, Reason, WantToRebook)
- Reason: Must be a valid reason for missing the flight (e.g., ‘late arrival’, ‘personal issues’).
- WantToRebook: Must be a boolean (yes or no).

15. Set_To_Make_Flight (PassengerID, TicketNumber, Flight#)
- Flight#: Must reference an existing flight.

16. Flight (Flight#, FlightDate, RouteNumber, Airline, Origin, Destination, IsInternational, DepartureTime, DepartureRunway, ArrivalTime, LandingRunway, DepartureGate, ArrivalGate, NumberOfSeats, Booked, Employees, Status)
- FlightDate: Must be a valid date in the future.
- DepartureTime, ArrivalTime: Must be valid timestamps, and ArrivalTime must be after DepartureTime.
- IsInternational: Must be a boolean (true for international, false for domestic).

- NumberOfSeats: Must be greater than zero and match the available seats for the aircraft.
- Status: Must be one of {‘On Time’, ‘Delayed’, ‘Cancelled’}.
- Employees: Must reference valid employees (pilot, flight attendants, support staff).

17. Domestic (Flight#)
    - Flight#: Must reference a domestic flight.

18. International (Flight#)
    - Flight#: Must reference an international flight.

19. Passed_Customs (Flight#)
    - Flight#: Must reference an international flight.

20. Not_Allowed_Entry (Flight#)
    - Flight#: Must reference an international flight.

21. Airlines (CompanyID, HeadquartersLocation)
    - HeadquartersLocation: Must be a valid location (city, state, country).

22. Equipment_Infrastructure (CompanyID, ItemNumber)
    - ItemNumber: Must be unique per company.

23. AirlineEmployee (CompanyID, EmployeeID, Name)
    - EmployeeID: Must be unique per airline.
    - Name: Must be non-empty.

24. Support_Staff_Gate_Agents (CompanyID, EmployeeID, Title)
    - Title: Must be one of the predefined titles for gate agents (e.g., 'Gate Agent', 'Support Staff').

25. Flight_Attendants (CompanyID, EmployeeID, Position, Seniority)
    - Position: Must be a valid flight attendant position (e.g., ‘Senior’, ‘Junior’).
    - Seniority: Must be a positive integer representing years of service.

26. Pilot (CompanyID, EmployeeID, Captain, YearsExperience)
    - Captain: Must be a boolean (true or false).
    - YearsExperience: Must be a non-negative integer.

27. Airport (AirportCode, Address)
    - AirportCode: Must be a valid IATA Airport code.
    - Address: Must be a non-empty address string.

28. AirportEmployee (AirportCode, EmployeeID, Name)
    - EmployeeID: Must be unique per airport.

- Name: Must be non-empty.

29. AirportInfrastructure (AirportCode, Type)
    - Type: Must be one of {‘Other’, ‘Gates’, ‘Runways’}.

30. Other (AirportCode, InventoryID#, Description)
    - InventoryID#: Must be unique per airport.
    - Description: Must be non-empty.

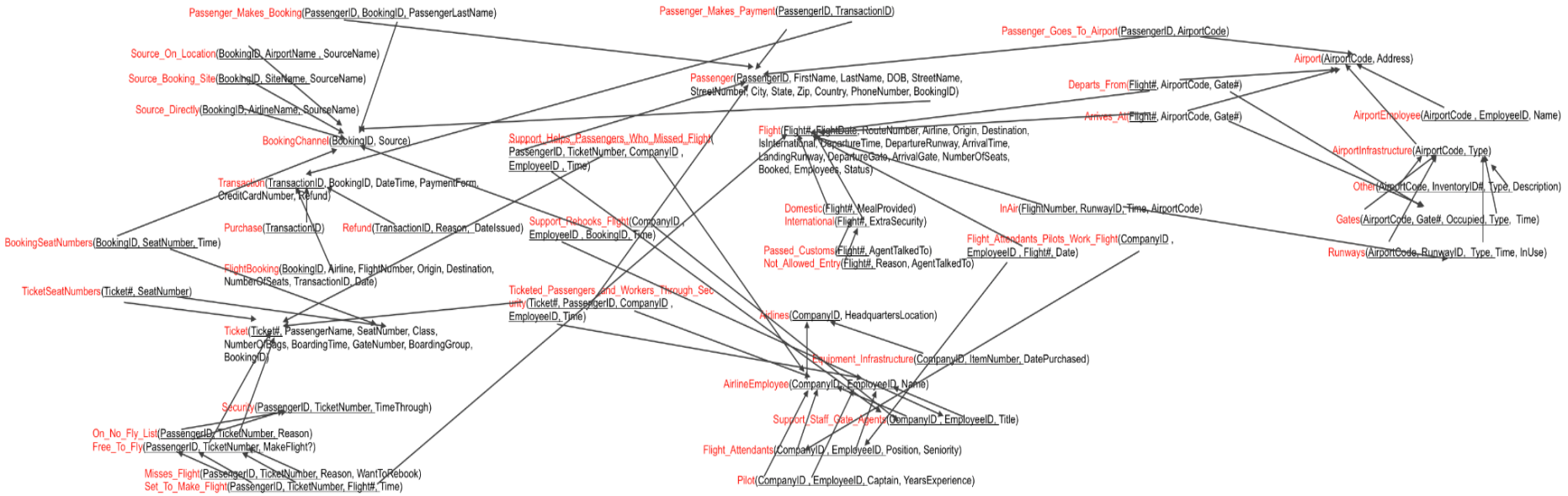
31. Gates (AirportCode, Gate#, Occupied, Time)
    - Occupied: Must be a boolean (true or false).
    - Time: Must be a valid timestamp.

32. Runways (AirportCode, RunwayID, Time, InUse)
    - RunwayID: Must be unique per airport.
    - InUse: Must be a boolean (true or false).
    - Time: Must be a valid timestamp.

33. InAir (FlightNumber, RunwayID)
    - FlightNumber: Must reference an existing flight.
    - RunwayID: Must reference a valid runway from the airport.

Relational Schema:

- *Note, taking your feedback into consideration, we added several more non-key attributes to the relations/ tables that only had key attributes. These are reflected in the relational schema (see below… and of course the code) as well as the EER attribute list (above).*
- See Below


Passenger_Makes_Booking(PassengerID, BookingID, PassengerLastName)
Passenger_Makes_Payment(PassengerID, TransactionID)
Passenger_Goes_To_Airport(PassengerID, AirportCode)
Source_On_Location(BookingID, AirportName , SourceName)
Source_Booking_Site(BookingID, SiteName, SourceName)
Source_Directly(BookingID, AirlineName, SourceName)
BookingChannel(BookingID, Source)
Passenger(PassengerID, FirstName, LastName, DOB, StreetName, StreetNumber, City, State, Zip, Country, PhoneNumber, BookingID)
Airport(AirportCode, Address)
Departs_From(Flight#, AirportCode, Gate#)
Arrives_At(Flight#, AirportCode, Gate#)
AirportEmployee(AirportCode , EmployeeID, Name)
AirportInfrastructure(AirportCode, Type)
Other(AirportCode, InventoryID#, Type, Description)
Gates(AirportCode, Gate#, Occupied, Type, Time)
Runways(AirportCode, RunwayID, Type, Time, InUse)
Support_Helps_Passengers_Who_Missed_Flight(PassengerID, TicketNumber, CompanyID , EmployeeID , Time)
Flight(Flight#, FlightDate, RouteNumber, Airline, Origin, Destination, IsInternational, DepartureTime, DepartureRunway, ArrivalTime, LandingRunway, DepartureGate, ArrivalGate, NumberOfSeats, Booked, Employees, Status)
Transaction(TransactionID, BookingID, DateTime, PaymentForm, CreditCardNumber, Refund)
Domestic(Flight#, MealProvided)
International(Flight#, ExtraSecurity)
InAir(FlightNumber, RunwayID, Time, AirportCode)
Purchase(TransactionID)
Refund(TransactionID, Reason, DateIssued)
Support_Rebooks_Flight(CompanyID , EmployeeID , BookingID, Time)
Flight_Attendants_Pilots_Work_Flight(CompanyID , EmployeeID , Flight#, Date)
BookingSeatNumbers(BookingID, SeatNumber, Time)
Passed_Customs(Flight#, AgentTalkedTo)
Not_Allowed_Entry(Flight#, Reason, AgentTalkedTo)
FlightBooking(BookingID, Airline, FlightNumber, Origin, Destination, NumberOfSeats, TransactionID, Date)
TicketSeatNumbers(Ticket#, SeatNumber)
Ticketed_Passengers_and_Workers_Through_Security(Ticket#, PassengerID, CompanyID , EmployeeID, Time)
Airlines(CompanyID, HeadquartersLocation)
Ticket(Ticket#, PassengerName, SeatNumber, Class, NumberOfBags, BoardingTime, GateNumber, BoardingGroup, BookingID)
Equipment_Infrastructure(CompanyID, ItemNumber, DatePurchased)
AirlineEmployee(CompanyID, EmployeeID, Name)
Security(PassengerID, TicketNumber, TimeThrough)
Support_Staff_Gate_Agents(CompanyID , EmployeeID, Title)
On_No_Fly_List(PassengerID, TicketNumber, Reason)
Free_To_Fly(PassengerID, TicketNumber, MakeFlight?)
Flight_Attendants(CompanyID , EmployeeID, Position, Seniority)
Misses_Flight(PassengerID, TicketNumber, Reason, WantToRebook)
Set_To_Make_Flight(PassengerID, TicketNumber, Flight#, Time)
Pilot(CompanyID , EmployeeID, Captain, YearsExperience)

Step 3: Implementation

Triggers:

- Trigger adjustAvailableSeats (After Booking or Cancellation)
    - To ensure accurate tracking of seat availability on flights. The trigger adjusts the number of available seats each time a booking is made or canceled.
    - After an update on the Ticket table, this trigger checks if a ticket has been booked or canceled and updates the number of available seats for the related flight accordingly. It decreases the seat count upon booking and increases it upon cancellation. This trigger uses the flightNumberOfSeatsLeft function by keeping the seat inventory accurate and up-to-date, facilitating better management of bookings and customer expectations.

- Trigger enforceNoFlyList (Before Ticket Booking)
    - To prevent passengers on the no-fly list from booking flights, enhancing security measures and regulatory compliance. Before inserting a new record into the Booking table, this trigger checks if the passenger is listed on the no-fly list. If the passenger is found on the list, the booking transaction is blocked and an alert is generated. Works in conjunction with the bookFlight function to ensure security protocols are adhered to, reducing the risk of allowing flagged individuals from traveling.

- Trigger auditTrailLog (After any Data Modification)
    - To maintain a historical log of all changes made to critical data within the database for auditing and compliance purposes. After any update, insert, or delete operation across key tables (e.g., Flight, Booking, Passenger), this trigger records the old and new values along with the timestamp and the identity of the user making the change. Provides a reliable audit trail that can be used for tracking changes, troubleshooting issues, and responding to audit requests.

- Trigger: automaticRefundInitiation (After Flight Cancellation)
    - This trigger automatically processes refunds for passengers when a flight is canceled, ensuring timely compensation without requiring any action from passengers.
    - Upon detecting a flight cancellation in the database, the trigger initiates refund processes for all booked passengers, adjusting their accounts and logging each transaction for accountability.This automation enhances customer service by providing immediate and hassle-free financial resolution, thereby improving passenger satisfaction and trust in the airline's operations.

- Trigger: checkOverbooking (before Flight departure)
    - This trigger is designed to automatically verify whether a flight has available seats by invoking the flightNumberOfSeats function during the booking process.

Regardless of the booking approach used (online, in-person, or via an agent), this trigger actively monitors the number of reservations in real-time. If the function detects that all seats are booked, the trigger intervenes to prevent additional bookings, ensuring that the flight does not become overbooked and maintaining accurate seat availability for other passengers.

- Trigger sendNotification (if Flight status is updated)
    - This trigger is designed to proactively send real-time updates regarding a flight's status as notifications to all passengers booked on that specific flight. By delivering timely alerts, this trigger aims to assist passengers in adjusting their schedules promptly, especially those with connecting flights that might be affected by delays or cancellations. The notification system relies on several key procedures and functions, including seeAvailableFlights and getAvailableFlights, to provide accurate and current information on flight status. This ensures passengers are informed as soon as any changes occur, helping to minimize disruptions to their travel plans.

Procedures:

- Procedure seeAvailableFlights(origin, destination, FlightDate)
    - Whether you are a potential traveler, a gate agent, a machine at a check-in counter, or the CEO of a company, knowing what flights have availability is of utmost importance.
    - This procedure takes in the desired origin location, desired destination location and the date you wish to fly and displays a list of all flights with at least one seat.
    - It shows all appropriate information for the ticket for the flights such as flight number, departure time, boarding time, airline, seat number, seat class, and boarding group etc so the potential passenger can make an informed decision.
    - The results are sorted by the number of seats followed by the airline.
    - This procedure works hand in hand with the function bookFlight as a passenger could use this procedure to check availability and then use the bookFlight function to book the desired flight.

- Procedure seeCrewWhoCanWork(FlightDate, AirlineName)
    - For airlines, managing crew schedules is incredibly important.
    - This procedure is designed for an airline scheduling manager or executive to see which crew has availability to work.
    - For example, a crew member could call in sick so the manager could use this procedure to find a suitable replacement.
    - Or the scheduling manager could simply create next month's schedule and use the procedure to build the schedule from scratch.
    - Or, if the scheduling is all managed by an app, this procedure could be automatically called as part of a bigger algorithm.

    - It works by looking at a given day and looping over all flights and identifying all crew members working flights and thus we can see the crew members not working flights and thus available to work.
    - It displays a list of aforementioned crew members free to work.

- Procedure handleMissedFlights(PassengerID, FlightDate, Flight#)
    - Passengers miss flights all the time, and in a perfect world, the airlines can handle these missed flights automatically. To make this work we make a few key assumptions. First, if a passenger misses their flight they must be booked onto another flight by the same airline. And, this procedure allocates the passenger to the first available flight.
    - It works by looping over all passengers who have missed their flight, looking at the original flight's origin and destination, and then comparing that to a list of all flights with availability with the same origin and destination (by calling the getAvaibleFlights function). If a flight exists, the bookFlight function is called and the passenger is automatically booked. If not, the refundFlight function is called to refund the passenger.

- Procedure executiveFlightSummary(start_date, end_date)
    - Knowing which routes are popular is incredibly important for the airline leadership team as it can help them make informed decisions on new routes or cancel existing routes.
    - This procedure could do a plethora of calculations but for our implementation, we will call two functions (see below) to calculate the number of flights in a given period and the average occupancy rate both grouping by origin and destination.
    - It then displays the number of flights and average occupancy rate for each origin/destination pair.

Functions:

- Function getNumberOfFlights(origin, destination, start_date, end_date) returns numberOfFlights
    - This function gets the number of flights from one location to another location in a given period of time.
    - It counts the number of instances of flights for a given origin/destination pair and returns this.
    - Note depending on the implementation of the executiveFlightSummary procedure we may group by airline as well.
    - This is used in the executiveFlightSummary procedure.

- Function getOccupancyRate(origin, destination, start_date, end_date) returns averageOccupancyRate
    - For a given origin/destination pair, for each flight, find the percent of seats that are actually booked and return the average of this (averageOccupancyRate) for a given time.

    - For example, we could see that 75% of seats on flights from Boston to Edinburgh are actually booked on average.
    - This is key for the executiveFlightSummary procedure.

- Function flightNumberOfSeatsLeft (Flight#, FlightDate) returns numberOfSeatsLeft
    - For a given flight on a given day, how many seats are left (free to book)
    - This is similar to occupancy rate but instead looks at a single flight and is looking at the actual number of seats.
    - This is used in many of the above procedures and a key fact to be able to calculate.

- Function refundFlight(Flight#, Reason) returns TransactionID
    - This function creates a refund transaction.
    - This is used in a plethora of scenarios whenever we need to quickly refund a passenger for a flight.

- Function getAvailableFlights(origin, destination, FlightDate) returns availableFlights
    - Exactly the same as the procedure just a function that returns a list of available flights instead. Used in handling missed flight procedures.

- Function bookFlight(PassengerID, Flight#, SeatNumber, FlightDate, AirlineName) returns BookingID, ticket number
    - The process of a passenger going for a trip starts with them buying a ticket.
    - A seat on a specific flight can be uniquely identified by the flight number of the overall flight, the seat number, the date, and the airline. For example, seat 1a on UA 123456789 on 11/7/24 corresponds to only one seat ever (due to the date). This makes sure no two identical tickets are booked.
    - We also must know who is booking the flight/ticket so that we can backtrack to determine their purchasing means, demographic, etc.
    - There are many ways that booking a flight could go wrong such as the flight, passenger, seat, airline, date either not existing or not being available so this function will handle these cases.
    - It also catches the passengers that are on no-fly lists before they book the ticket so we can minimize problems at the security/check-in counter
    - It returns the booking ID and ticket number so that the passenger knows and feels confident that the booking worked.

Indexes:

- First consider the flight table that stores all the flights. As constructed, flight number and flight date are the composite primary key and thus, after doing some research (and trying and getting an error) oracle automatically creates an index. However, we can look at creating just an index for flight date. We first consider file organization. As we can see

both looking at the functions and thinking of likely scenarios, knowing which flights are coming up is key. Said another way, once a flight has happened, it's of less importance to us. For example, think of an arrival board, we care about the new flights that are upcoming. Thus, sorting the records so we can insert new flights at the end (and effectively ignore the flights that have already happened) is a worthy trade-off (given the slightly higher cost of the sequential file). Thus we would want this table stored in a sequential file ordering on flight date (for this new index on date we are creating and not the default index created on the primary key). Then, as we are often looking for flights between two dates, or upcoming flights (date>givenDate), we do range searches, and thus a B+ (oracles default) makes the most sense.

- Next, consider the passenger table. Unlike the flights, ordering is far less important to use given our most frequent uses. Said another way, we care far more than a passenger exists in the first place, rather than when they say, created their account, their age, etc. Thus, we don't get a performance boost by investing early (in the table creation) in a sequential file. Thus we would save this expense (as there is no benefit) and would use a heap file for storing the passenger table. Furthermore, we see that we are most often making sure that passenger IDs match and thus we have selections of equality and thus we prefer to use hash indexes. This strikes a balance of using the hashing to improve efficiency while not going overkill with a sequential file. So, we create a heap file with a hash index on the passenger ID via the code. However, in doing some more research, it doesn't seem Oracle directly supports hash indexing directly so instead we create a cluster file structure with passengerID as the cluster key to group the passengers by their ID (group the records) and then Oracle creates a B+ tree on the clustering key. While not ideal, as we would prefer to just implement a hash index directly, this seems like a nice compromise given Oracle's limitations.

Packages:

- We lastly consider the packages. In doing so we will create two packages as a vehicle for use and/or implementation of the procedures (almost all of them) we have already created. Thus, the logic is very similar to code we already have written but we modify the function and procedure definitions to fit into package form.
- The first package we chose to create is called "quickNumbersandFact" and holds our functions and procedures that get quick metrics/ numbers about a flight, airport, or airline. It includes the following functions and procedures (as described more in-depth above) "getNumberOfFlights," "getOccupancyRate," "flightNumberOfSeatsLeft," and "executiveFlightSummary." This package is ideal as it is a wrapper/ holder for these quick metrics and if our database changes or we move over to a new system of sorts, these functions flock together so to speak, and thus packaging them together makes logical sense.
- The second package we chose to create is called "flightInfoandHandling." This stores a majority of the other procedures/functions we already created (see the code). While the quick summary statistics provided by "quickNumbersandFact" are helpful, we often need more and to do more. That's where "flightInfoandHandling" it combines the functions and procedures that handle refunding a flight ("refundFlight"), booking a

flight(“bookFlight”), handling missed flights(“handleMissedFlights”), and seeing and accessing available flights (“seeAvailableFlights”, “getAvailableFlights”). These are the go-to utility functions/ procedures for holding up the core utilities of the airlines, the airports, or travelers. So, combining them into one easy-to-use place makes sense.

Step 4: Programming

We created all the tables described in our schema and relational schema and then wrote insert statements for over 16 of these tables. In doing so consider a majority of the constraints we described (see previous section). During these table creation statements we also created the indexes. From there, In looking at the triggers, functions, and procedures, you will see code for all of the aforementioned and described triggers, functions, and procedures. From there we explicitly called and validated the following functions: getNumberOfFlights, getOccupancyRate, flightNumberOfSeatsLeft, getAvailableFlights. We also validated the following triggers: adjustAvailableSeats, enforceNoFlyList, auditTrailLog. We lastly evaluated the following procedures: seeAvailableFlights, seeCrewWhoCanWork, handleMissedFlights, executiveFlightSummary. These calls occur after the creation statements. Note, while we did not explicitly call all of them, we call a significant majority (far more than the requirement). Lastly, we created the two packages we previously described. Within the package we see that we had no errors AND we decided to more concretely test the executive flight summary function within the package as it calls other functions already in the package, i.e. get number of flights and get occupancy rate. For the second package we also tested one of the functions to see available flights. As all functions in both packages are identical the packages we’ve already created testing any more would be redundant. See attached code files. Note that we combined everything in one file for ease of use. Also note there are a few lines of code commented out that are remnants of our drafts.