Group 20: Entity-Relationship Model

This document outlines our group’s entity-relationship model which represents the attached application description. A visual representation of this model is in the attached diagram file.

Overall, this model is designed to help an airline develop a better understanding of their passengers for management and marketing purposes.

**Part 1: SQL Domain DDL**

This section contains SQL DDL for all the domain attributes organized by the primary object (entity or relationship) that they belong to.

**Airport**

CREATE DOMAIN D\_AirportId INTEGER

CHECK (VALUE > 10000 AND VALUE < 99999)

CREATE DOMAIN D\_IATA CHAR(3)

**Passengers**

CREATE DOMAIN D\_PassengerId INTEGER

CHECK ( VALUE > 100000000 AND VALUE < 999999999)

CREATE DOMAIN D\_PassportNumber INTEGER

CHECK (VALUE < 999999999)

CREATE DOMAIN D\_Name VARCHAR(75)

CREATE DOMAIN D\_Age INTEGER

CHECK (VALUE > 0 AND VALUE < 130)

CREATE DOMAIN D\_Email VARCHAR(100)

**Members**

CREATE DOMAIN D\_Points INTEGER

CHECK (VALUE > =0)

CREATE DOMAIN D\_StartDate DATE

CREATE DOMAIN D\_MembershipId INTEGER

CHECK (VALUE > 10000 AND VALUE < 99999)

**NonMembers**

CREATE DOMAIN D\_EmailSubscription BOOLEAN

CREATE DOMAIN D\_WasMember BOOLEAN

**PremiumMember**

CREATE DOMAIN D\_PremiumPoints INTEGER

CHECK (VALUE >= 0)

CREATE DOMAIN D\_RedeemedPremRewards VARCHAR(200)

**Flight**

CREATE DOMAIN D\_FlightId INTEGER

CHECK (VALUE > 10000 AND VALUE < 99999)

CREATE DOMAIN D\_FlightDate DATE

**Airplane**

CREATE DOMAIN D\_AirplaneId INTEGER

CHECK (VALUE > 100000000 AND VALUE < 999999999)

CREATE DOMAIN D\_Model VARCHAR(50)

**CommercialPlane**

CREATE DOMAIN D\_Brand VARCHAR(20)

**CargoPlane**

CREATE DOMAIN D\_Capacity DOUBLE

CHECK (VALUE > 0)

**PassengerPlane**

CREATE DOMAIN D\_SafetyPackage VARCHAR(100)

**Seat**

CREATE DOMAIN D\_SeatCode CHAR(4)

CREATE DOMAIN D\_Class CHAR(14)

CHECK (VALUE IN (“First Class”, “Business Class”, “Economy Class”, “Other”))

CREATE DOMAIN D\_Type CHAR(6)

CHECK (VALUE IN (“Window”, “Aisle”, “Middle”, “Other”))

**Fleet**

CREATE DOMAIN D\_FleetId INTEGER

CHECK (VALUE > 0 AND VALUE < 100)

CREATE DOMAIN D\_Manager VARCHAR(100)

**Booking**

CREATE DOMAIN D\_BookingDate DATE

CREATE DOMAIN D\_CardType VARCHAR(10)

CREATE DOMAIN D\_TicketNumber INTEGER

CHECK (VALUE > 1000 AND VALUE <9999)

**Movement**

CREATE DOMAIN D\_DepartureTime TIME

CREATE DOMAIN D\_ArrivalTime TIME

CREATE DOMAIN D\_ArrivalTimeZone CHAR(5)

CREATE DOMAIN D\_FirstMealTime TIME

**ServicedAt**

CREATE DOMAIN D\_LeadTechnician VARCHAR(100)

CREATE DOMAIN D\_AssistantTechnician VARCHAR(100)

CREATE DOMAIN D\_AvailableServices VARCHAR(100)

CREATE DOMAIN D\_Cost DOUBLE

**Home**

CREATE DOMAIN D\_PostalCode CHAR(6)

CREATE DOMAIN D\_Country VARCHAR(100)

CREATE DOMAIN D\_City VARCHAR(100)

CREATE DOMAIN D\_Region VARCHAR(50)

CREATE DOMAIN D\_Languages VARCHAR(20)

**BelongsTo**

CREATE DOMAIN D\_EngineType VARCHAR(20)

CREATE DOMAIN D\_FuelConsumption DOUBLE

CREATE DOMAIN D\_PlaneFleetNumber INTEGER

CHECK (VALUE > 0)

CREATE DOMAIN D\_StartDate DATE

**Part 2: Relationship Keys**

Our model has a total of 7 relationships. The keys of each relationship are outlined in the table below.

|  |  |  |
| --- | --- | --- |
| Relationship | Roles [Entities] | Keys |
| Booking | Passengers  Flights  Seats | *PassengerId* (from Passengers)  *FlightId* (from Flights)  *Seat Number* (from Seats) |
| Movement | Flights  Arrival [Airport]  Departure [Airport] | *FlightId* (from Flights)  *ArrivesAt* (from Arrival i.e. Airport)  *DepartsFrom* (from Departure i.e. Airport) |
| ServicedAt | Fleet  Airport | *FleetId* (from Fleet)  *AirportId* (from Airport) |
| Belongs To | Airplane  Fleet | *AirplaneId* (from Airplane)  *FleetNumber* (from Fleet) |
| SeatOn | Seat  Airplane | *SeatCode* (from Seat)  *AirplaneId* (from Airplane) |
| FlightBy | Flight  Airplane | *FlightId* (from Flights)  *AirplaneId* (from Airplane) |
| Home | Passengers  Airport | *PassengerId* (from Passengers)  *AirportId* (from Airport) |

**Part 3: Justification Ternary and Example**

One of our ternary relationships is the Movement relationship. The three roles that contribute to this relationship are *Flight*, *Arrival*, and *Departure*. *Flight* comes from the flight entity and *Arrival* and *Departure* both come from the Airport Entity. This relationship needed to be ternary because the Movement construct would be incomplete without each of the three inputs, as is explained below.

Without the *Flight* entity, this relationship would only capture arbitrary arrival and departure destinations, without any link to passengers or the date. This would not be a useful construct for our purposes, which involves modeling the preferences and movements of passengers.

Secondly, omitting either the *Departure* or the *Arrival* airports would not capture the full details of flight movement; to include just the departure destination or just the arrival destination would only capture half of the movement. Consequently, all three of these roles were necessary.

**Part 4: Justification Choices**

**4.1 Attribute vs. Entity**

Deciding on how to represent the membership program was a challenge. Initially, we considered adding attributes to the *Passenger* entity to represent membership. However, this would lead to a lot of additional columns that would only hold null values for many non-member passengers. This was amplified by the fact that we also wanted to track information about premium memberships distinctly from regular memberships; since premium members are a small minority of the customer base, this would lead to even more null values. Instead, we decided to make sub-entities to represent members and premium members, using an IsA hierarchy. This eliminated a lot of redundancy that would have occurred by using attributes.

**4.2. Entity vs. Relationship**

In one of our earlier drafts of our model, we considered having *Flight* as a relationship, with the *Passengers*, *Airplane*, and *Airport* entities connected. However, we decided to make flight an entity so that we could also capture information about booking details in addition to movement details. For example, we wanted to capture information about what credit cards passengers use to book, what date they booked on, and their ticket number. This information does not pertain directly to the flight movement, and so we determined that it would make more sense to model it separately.

**Part 5: Justification IsA Constraints**

There are three cases of IsA constraints contained within our model. They are listed as follows.

1. Disjoint and Covering

2. Disjoint and Not Covering

3. Not Disjoint and Not Covering

The remainder of this section will discuss the different situations where these constraints apply and justify why they were necessary.

**5.1 Disjoint and Covering**

This combination of constraints was necessary to describe the membership hierarchy. All passengers can be classified in this binary ofmembers or non-members (i.e. they’re either a member or they’re not). As a result, the two sub-entities cover the *Passenger* entity.

Since the two groups are mutually exclusive (i.e. a single passenger cannot be both a member and a non-member), the entities of *Members* and *NonMembers* are disjoint.

**5.2 Disjoint and Non-Covering**

This combination of constraints was used for the sub-entity *PremiumMemberships*. This entity does not cover the super-entity *Members*, as not all members will have the premium package. However, it was necessary to have an additional entity to represent premium members so the airline can track premium points and redeemed rewards separate from the points and rewards for regular members. Since there are no other sub-entities to *Members,* the *PremiumMembers* entity is also disjoint.

**5.2 Non-Disjoint and Non-Covering**

This combination of constraints was used for the sub-entities of *CommercialPlanes*. As was discussed in the application description, commercial planes can have singular or dual purposes. For example, planes that primarily carry passengers may also be able to transport some cargo for extra profit. This may be helpful if the airline ever considers an expansion of dual-purpose flights and wants to track their frequency and success.

Since carrying passengers and carrying cargo can occur at the same time, these entities are not disjoint. However, these two types of planes are the only types of commercial planes owned by our airline, so they do cover the whole set of commercial planes.

**Part 6: Justification PartOf**

In our model, the weak entity is called *Seat.* This entity was necessary because, as was stated in the application description, passengers make a seat selection when they book a flight (see Figure 1). This is critical as the price may vary depending on the seat class (ex. first class) and type (ex. aisle seat). However, the seat itself is not identifiable without knowing the specific airplane that will be used for the flight. Consequently, the weak entity *Seat* is connected to the strong entity *Airplane* by a PartOf relationship called *SeatOn*.