Group 20: Relational Model

**Part 1: Relational Model**

*Obtain an initial Relational Model from your Entity-Relationship model in Assignment 3.1, using the simple ER-to-Relational mapping techniques covered in the lectures and the textbook. Give the SQL CREATE TABLE statements for the relations in your model, using the SQL Domains that you already defined in ERNotes.*

**SQL Create Table Statements**

CREATE TABLE Passengers (

passengerId D\_PassengerId,

name D\_Name,

age D\_Age,

passportNumber D\_PassportNumber NOT NULL UNIQUE,

email D\_Email,

emailSubscription D\_EmailSubscription,-- non-member attribute

wasMember D\_WasMember, -- non-member attribute

membershipId D\_MembershipId, --member attribute

points D\_Points, -- member attribute

startDate D\_StartDate, -- member attribute

passengerType CHAR(1) NOT NULL, -- new attribute

PRIMARY KEY(passengerId)

);

CREATE TABLE Flight(

flightId D\_FlightId,

flightDate D\_FlightDate,

PRIMARY KEY(flightId)

);

CREATE TABLE Airport (

airportId D\_AirportId,

IATA D\_IATA,

PRIMARY KEY(airportId)

);

CREATE TABLE Fleet (

fleetId D\_FleetId,

manager D\_Manager,

PRIMARY KEY(fleetId)

);

CREATE TABLE PremiumMember (

passengerId D\_PassengerId,

premiumPoints D\_PremiumPoints,

redeemedPremRewards D\_RedeemedPremRewards

PRIMARY KEY(passengerId, redeemedPremRewards),

FOREIGN KEY (passengerId) REFERENCES Passenger

);

CREATE TABLE Seat (

seatCode D\_SeatCode,

airplaneId D\_AirplaneId,

class D\_Class,

type D\_Type,

PRIMARY KEY(seatCode, airplaneId),

FOREIGN KEY (airplaneId) REFERENCES Airplane ON DELETE CASCADE

);

CREATE TABLE Airplane (

airplaneId D\_AirplaneId,

model D\_Model,

PRIMARY KEY(airplaneId)

);

CREATE TABLE CargoPlane (

airplaneId D\_AirplaneId,

capacity D\_Capacity,

brand D\_Brand,

PRIMARY KEY(airplaneId),

FOREIGN KEY (airplaneId) REFERENCES Airplane

);

CREATE TABLE PassengerPlane (

airplaneId D\_AirplaneId,

safetyPackage D\_SafetyPackage,

brand D\_Brand,

PRIMARY KEY(airplaneId, safetyPackage),

FOREIGN KEY(airplaneId) REFERENCES Airplane

);

CREATE TABLE Home (

passengerId D\_PassengerId,

airportId D\_AirportId,

postalCode D\_PostalCode,

languages D\_Languages, -- languages is a set-valued attribute

city D\_City,

country D\_Country,

region D\_Region,

PRIMARY KEY(passengerId, airportId, languages),

FOREIGN KEY (passengerId) REFERENCES Passengers,

FOREIGN KEY (airportId) REFERENCES Airport

);

CREATE TABLE FlightBy (

flightId D\_FlightId,

airplaneId D\_AirplaneId,

PRIMARY KEY(flightId,airplaneId),

FOREIGN KEY (airplaneId) REFERENCES Airplane,

FOREIGN KEY (flightId) REFERENCES Flight

);

CREATE TABLE Booking (

passengerId D\_PassengerId,

flightId D\_FlightId,

seatCode D\_SeatCode,

bookingDate D\_BookingDate NOT NULL,

cardType D\_CardType,

ticketNumber D\_TicketNumber NOT NULL,

PRIMARY KEY(passengerId, flightId, seatCode, Promotions),

FOREIGN KEY (passengerId) REFERENCES Passengers,

FOREIGN KEY (flightId) REFERENCES Flight,

FOREIGN KEY (seatCode) REFERENCES Seat

);

CREATE TABLE BelongsTo (

airplaneId D\_AirplaneId,

fleetId D\_FleetId,

engineType D\_Engine,

planeFleetNumber D\_PlaneFleetNumber

startDate D\_StartDate,

fuelConsumption D\_FuelConsumption,

PRIMARY KEY(airplaneId, fleetId),

FOREIGN KEY (airplaneId) REFERENCES Airplane,

FOREIGN KEY (fleetId) REFERENCES Fleet

);

CREATE TABLE Movement (

flightId D\_FlightId,

arrivesAt D\_AirportId,

departsFrom D\_AirportId,

departureTime D\_DepartureTime,

arrivalTime D\_ArrivalTime,

arrivalTimeZone D\_ArrivalTimeZone,

firstMealTime D\_FirstMealTime,

PRIMARY KEY(flightId, arrivesAt, departsFrom),

FOREIGN KEY (arrivesAt) REFERENCES Airport(airportId),

FOREIGN KEY (departsFrom) REFERENCES Airport(airportId),

FOREIGN KEY (flightId) REFERENCES Flight

);

CREATE TABLE ServicedAt (

airportId D\_AirportId,

fleetId D\_FleetId,

leadTechnician D\_LeadTechnician,

assistantTechnician D\_AssistantTechnician,

availableServices D\_AvailableServices, --set-valued attribute

cost D\_Cost, --set-valued attribute

PRIMARY KEY (airportId, fleetId, availableServices, cost)

FOREIGN KEY (airportId) REFERENCES Airport,

FOREIGN KEY (fleetId) REFERENCES Fleet

);

**Part 2: IsA Translation #1**

*Describe for one of the IsA relationships in your ER model with a given combination of disjointness and covering constraints, how you will translate the entities in the IsA relationship to relations, NOT using the default translation of the IsA, but one of the choices available when disjointness or covering constraints are present. Justify in writing your choice, making sure that you write again the disjointness and covering constraints that apply to your translation. Include the SQL statements that define the correct assertions to enforce the combination of disjointness and covering constraints that apply to your relational model translation (that is a correct CREATE ASSERTION statement including a CHECK NOT EXISTS ( … ) and the correct SQL query inside the parenthesis).*

We enforced the covering IsA constraint by not including a relation for commercial planes. Instead, we just added the attributes of the CommercialPlane entity (i.e. brand) to the relations CargoPlane and PassengerPlane. This will ensure that all commercial planes are categorized as either a CargoPlane, a PassengerPlane or both. In other words, there is no other place to store information about a commercial plane, so it is covering. To get a full list of Commercial Planes, we can take the union of the airplane ids from both these tables. The CREATE TABLE statements used are below.

CREATE TABLE CargoPlane (

airplaneId D\_AirplaneId,

capacity D\_Capacity,

brand D\_Brand, -- CommercialPlane Attribute

PRIMARY KEY(airplaneId),

FOREIGN KEY (airplaneId) REFERENCES Airplane

);

CREATE TABLE PassengerPlane (

airplaneId D\_AirplaneId,

safetyPackage D\_SafetyPackage,

brand D\_Brand, -- CommercialPlane Attribute

PRIMARY KEY(airplaneId, safetyPackage),

FOREIGN KEY(airplaneId) REFERENCES Airplane

);

**An alternative translation that we considered is as follows.** We could have included a relation CommercialPlanes and used assertions to enforce this IsA translation. This constraint required one assertion to enforce the covering constraint; in simple terms, the assertion checks that there are no commercial planes that don’t classify as either a cargo plane or a passenger plane. The CREATE TABLE statement and assertion are below.

CREATE TABLE CommercialPlane (

airplaneId D\_AirplaneId,

brand D\_Brand,

PRIMARY KEY(airplaneId),

FOREIGN KEY (airplaneId) REFERENCES Airplane

);

CREATE ASSERTION CommercialPlane\_Covering

CHECK (NOT EXISTS (

SELECT \* FROM CommercialPlane CP

WHERE NOT EXISTS (

SELECT \* FROM CargoPlane C, PassengerPlane P

WHERE CP.airplaneId = C.airplaneId

OR CP.airplaneId= P.airplaneId )

)

)

**The reason we decided to use the first translation** described was that it was a lot less costly to enforce; with the assertion, each transaction (i.e. deleting, updating, inserting, etc.) would require checks that used significant processing power; it is much easier to just embed these constraints within the relations themselves.

**Part 3: IsA Translation #2**

*Describe for a second the IsA relationships in your ER model with a different combination of disjointness and covering constraints from the one selected for IsA Translation 1 above, how you will translate the entities in the IsA relationship to relations, NOT using the default translation of the IsA, and also using a different choice from the one selected for IsA Translation 1 above. Justify in writing your choice, making sure that you write again the disjointness and covering constraints that apply to your translation. Include the SQL statements that define the correct assertions to enforce the combination of disjointness and covering constraints that apply to your relational model translation (that is a correct CREATE ASSERTION statement including a CHECK NOT EXISTS ( … ) and the correct SQL query insIde the parenthesis).*

We enforced the covering and disjoint IsA constraint by absorbing the Member and NonMember entities into the Passengers relation. We did this by adding the attributes from the Member (membershipId, points, startDate, redeemedRewards) and NonMember (emailSubscription and wasMember) subtypes to the Passenger relation. In addition, we introduced a new attribute called passengerType which encodes which subtype the passenger belongs to (i.e. will be M for member and N for non-member). This enforces the disjoint constraint; because passengerId is unique, a given passenger can only appear once, and will have a single passengerType value. Therefore they can never be both a member and a non-member.

The covering constraint is enforced by adding the NOT NULL clause on the passengerType attribute. This will ensure that every passenger will either be a member or a non-member; without this clause, a passenger could have a NULL value for this attribute indicating that they are neither. The CREATE TABLE statement used is below.

CREATE TABLE Passengers (

passengerId D\_PassengerId,

name D\_Name,

age D\_Age,

passportNumber D\_PassportNumber NOT NULL UNIQUE,

email D\_Email,

emailSubscription D\_EmailSubscription,-- non-member attribute

wasMember D\_WasMember, -- non-member attribute

membershipId D\_MembershipId, --member attribute

points D\_Points, -- member attribute

startDate D\_StartDate, -- member attribute

passengerType CHAR(1) NOT NULL, -- new attribute

PRIMARY KEY(passengerId)

);

**An alternative translation that we considered is as follows.** We could have included two additional relations: Member and NonMember and used assertions to enforce this IsA translation. These constraints would require two assertions to enforce. The first assertion would enforce covering. In simple terms, it would check that there are no passengers that cannot be found in either the member or non-member relations. The second assertion would enforce disjointedness. In simple terms, it would check that there are no passengers in both the member and non-member relations. The CREATE TABLE statements and assertion DDL can be found below.

CREATE TABLE NonMember (

passengerId D\_PassengerId,

emailSubscription D\_EmailSubscription,

wasMember D\_WasMember,

PRIMARY KEY(passengerId),

FOREIGN KEY (passengerId) REFERENCES Passengers

);

CREATE TABLE Member (

passengerId D\_PassengerId,

membershipId D\_MembershipId NOT NULL ,

points D\_Points,

startDate D\_StartDate,

redeemedRewards D\_RedeemedRewards,

PRIMARY KEY(passengerId, redeemedRewards),

FOREIGN KEY (passengerId) REFERENCES Passengers

);

CREATE ASSERTION Passenger\_Covering

CHECK ( NOT EXISTS (

SELECT \* FROM Passengers P

WHERE NOT EXISTS (

SELECT \* FROM NonMember NM, Member M

WHERE P.passengerId = NM.passengerId

OR P.passengerId= M.passengerId)

)

)

CREATE ASSERTION Membership\_Disjoint

CHECK ( NOT EXISTS (

SELECT \* FROM NonMember NM, Member M

WHERE M.passengerId = NM.passengerId

)

)

Like in part 2 of this document, we decided to use the initial IsA translation where the constraints are embedded in the tables themselves. **The reason we decided to use this translation is because** it was less costly from a transactional standpoint; instead of repeatedly needing to perform complex assertion checks, the constraints can be easily contained in the structure of the tables, requiring minimal processing power to enforce.