Airplane Tickets System

Name: Andrés Felipe Sánchez Sánchez

Overview and Context:

In this case, the goal is to design a system which lets an airline manage their flights, resources and sales. We can describe the business operation as this: An airline has many flights which have an origin city and a destination city and also a time of departure and a duration of the flight. Also, every flight has assigned an aircraft which could be of many types (for instance AIRBUS A320 etc) and depending on this type the aircraft have a capacity of seats which also can be categorized in different classes like tourist class, VIP class etc and the price depends of this class. Indeed, a flight has assigned a set of tickets and every ticket has a seat associated in the airplane, also the flight has assigned a crew which has pilots and airhostesses.

Also, we have to manage the sales of tickets, then we have to keep the information of customers, but first we have to make a definition of customer: customer is everyone which pay for a set of tickets (a customer is different to a passenger but a customer may be a passenger also), this separation of concepts will let the airline to query for its customers and later offer them promotions for example.

In the business model of an airline there is also the possibility to offer different destinations which can be accomplished by several consecutive flights also called 'flight with stepovers', but this functionality can be implemented with graphs which take individual flights (stored in the db), this brings more flexibility to the system because if the logic of code (and not the database) builds a graph in which there are many edges (individual flights which are stored in the database) then this graph could calculate several paths with any number of stepovers between an origin and destination, Then when a user search flights between different cities the graph is built and calculate the different routes which the user could take. But what would happen if one of the flights in the stepovers is canceled? Then when a new user searches for a flight between the same origin and destination the route which contains the canceled flight will not be shown because the graph is built again and the canceled flights will not be retrieved from the database to build the graph.

With this overview we are going to define some uses cases:

Uses Cases:

Title	**Register an AirCraft**
Primary Actor	**System Administrator**
Success Scenario	System Administrator presses a button to register a new aircraft then, It fills the data related to the machine like model and type.

Title	**Register a Crew**

Primary Actor	**System Administrator**
Success Scenario	The System Administrator presses a button to register a new group of crew which is going to serve the airline. Then he/she fills the data of the pilots and the airhostesses which belong to the crew group and stores it in the database.

Title	**Register a flight**
Primary Actor	**System Administrator**
Success Scenario	System Administrator press a button to register a new flight, then he/she has to associate a crew and an aircraft to the flight and after that fill data related to the duration of the flight, the destination, the origin and a state of the flight which could be scheduled, on boarding, canceled, going or canceled. Then the data of the flight is stored in the database.

Title	**Sell tickets of a flight**
Primary Actor	**System Administrator**
Success Scenario	System Administrator presses a button to create the ticket of a flight, which also depends on the aircraft and its seats. Then the tickets are stored in the database without passenger associated

Title	**Make a purchase of tickets**
Primary Actor	**Customer**
Success Scenario	The customer enters to the system and makes a query to get a flight sending the destination and origin as parameters, then he/she selects the flight(s) of interest and then the system shows them the available tickets in that flight, after that the customer selects the tickets (which can be of several

	classes like VIP,tourist etc) which want to buy, then the customer registers in the system all her/his data and finally pay. The tickets are associated with a “purchase” which associates a customer with a set of tickets. At the moment of the purchase, the client must fill the data related to the passengers who take every ticket.

Title	**Cancel a Flight**
Primary Actor	**System Administrator**
Success Scenario	System Administrator presses a button to cancel a flight. It means that the state of the flight must be changed to canceled.

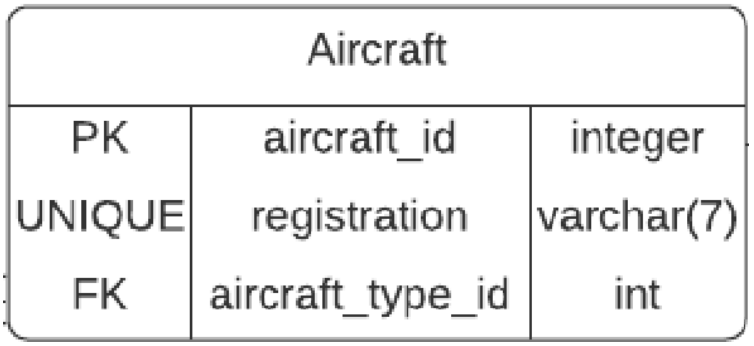
Title	**Change the price of a seat class**
Primary Actor	**System Administrator**
Success Scenario	System Administrator presses a button to edit the price of a class seat, it means that he/she can change the price for instance of the VIP class, but the sold tickets of that class must preserve the price when they were sold.

Description of Entities and Attributes:

- **Aircraft Entity:**

This entity let store information related to the aircraft owned by the airline, every aircraft has the following attributes:

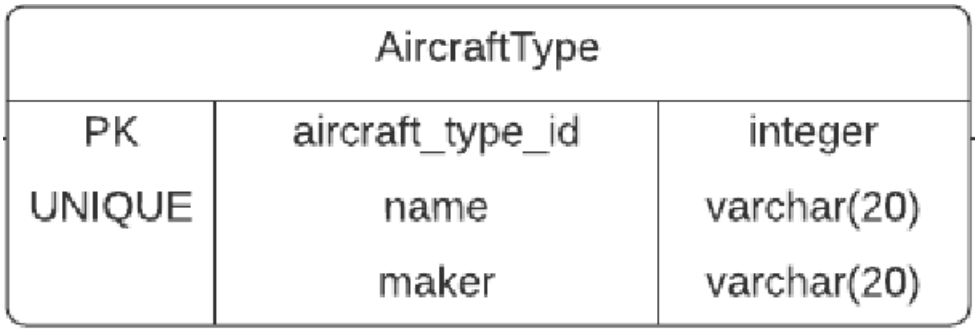
- **aircraft_id:** is the primary key which identifies the aircraft and its type is integer.
- **registration:** is the identification of the aircraft in the context of all airlines. its type is varchar
- **aircraft_type_id:** is a foreign key which references the table aircraft_type_id. It let to know the type of the aircraft, for instance if the aircraft is an Airbus A320. Its type is integer.



- **AircraftType Entity and Attributes:**

This entity let store information related to the different types of aircraft acquired by the airline, it means that everyone can query for the features of the plane based on this information. The attributes of this entity are the following:

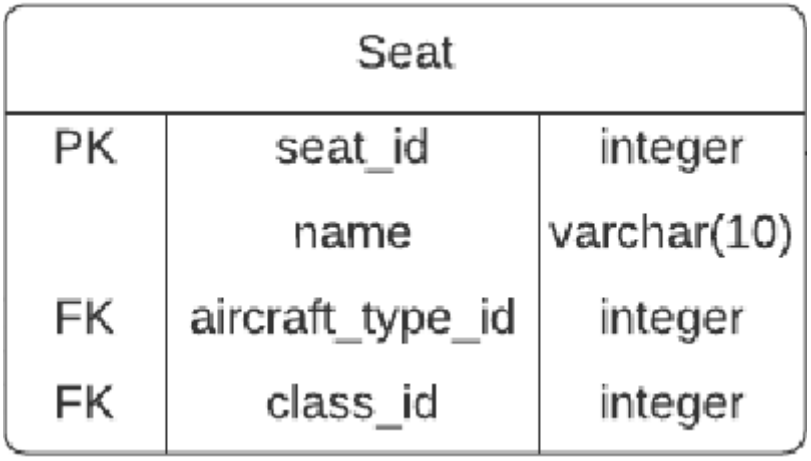
- **aircraft_type_id:** is the primary key which identifies the type of aircraft and its type is integer.
- **name:** is the name of the type of aircraft (for instance AIRBUS A320) which must be unique because it is not allowed to have two different instances of aircraft type with the same name. its type is varchar
- **maker:** it's the name of the vendor of the aircraft. Its type is varchar.



- **Seat Entity and Attributes:**

This entity stores information related to the seats in the aircrafts. It has the following attributes:

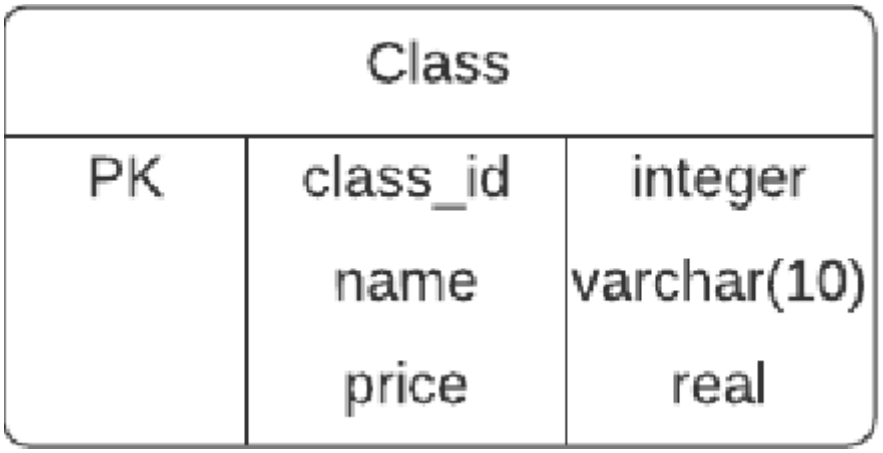
- **seat_id:** is the primary key of the table and its type is integer.
- **name:** is the name that lets the customer identify which is the seat assigned in the purchase and usually is the same enumeration that is depicted in the aircraft. its type is varchar
- **aircraft_type_id:** is a foreign key which references the AircraftType entity because the seat belongs to a type of aircraft.
- **class_id:** is a foreign key which references the Class entity and let to know which class is the seat (for instance is a seat of VIP class or tourist class). Its type is integer.



- **Class Entity and Attributes**

This entity lets store information related to the class of a seat. It has the following attributes:

- **class_id:** is the primary key of the class entity and its type is integer
- **name:** is the name of the class of seat for instance (executive, VIP, tourist class). Its type is varchar.
- **price:** is the price of a seat in the class. its type is real.

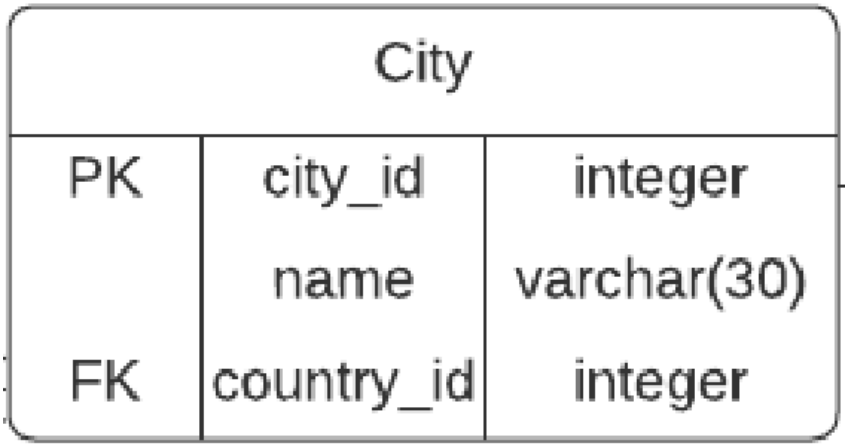


- **City Entity and Attributes**

This entity stores information about a city which is part of the origin or destination set of cities offered by the airline. This entity has the following attributes:

- **city_id:** is the primary key of the entity and its type is integer

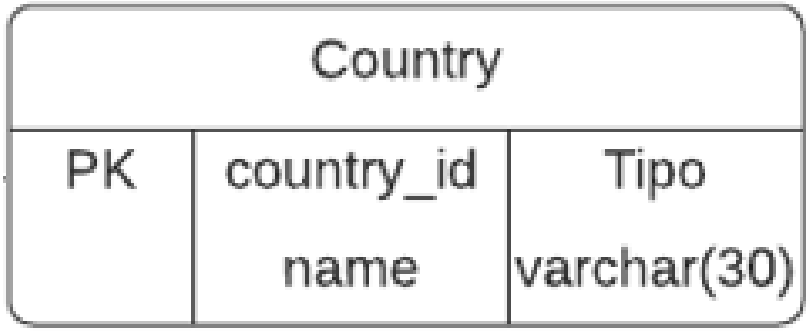
- **name:** is the name of the city which the airline has as origin or destination of a flight.
- **country_id:** is a foreign key which references the country entity, and let know what country which the city belongs to.



- **Country Entity and Attributes**

This entity stores information about the countries where are placed the cities. This entity has the following attributes:

- **country_id:** is the primary key of the entity and its type is integer
- **name:** is the name of the country which the airline has destinations or origins cities.



- **Pilot Entity and Attributes**

This entity stores information about the pilots hired by the airline. This entity has the following attributes:

- **pilot_id:** is the primary key of the entity and its type is integer
- **name:** is the name of the pilot which belongs to the airline. its type is varchar
- **license:** is a string which stands for the license of the pilot acquired after passing the course to operate planes. Its type is varchar
- **DNI:** is the identification document of the pilot. Its type is varchar.
- **crew_id:** is a foreign key which references the crew entity and let know which crew the pilot belongs to. Its type is integer.

Pilot		
PK	pilot_id	integer
	name	varchar(50)
UNIQUE	license	varchar(50)
UNIQUE	DNI	varchar(50)
FK	crew_id	integer

- **Airhostess Entity and Attributes**

This entity stores information about the airhostesses who work for the airline. This entity has the following attributes:

- **airhostess_id:** is the primary key of the entity and its type is integer
- **name:** is the name of the airhostness who works in the airline.Its type is varchar.
- **DNI:** is the identification document of the airhostness. Its type is varchar
- **crew_id:** is a foreign key which references the crew entity and lets know which crew the airhostness belongs to. Its type is integer.

Airhostess		
PK	airhostess_id	integer
	name	varchar(50)
UNIQUE	DNI	varchar(50)
FK	crew_id	integer

- **Crew Entity and Attributes**

This entity stores information about the crews (groups of workers that are needed to create a flight and includes pilots and airhostnesses). This entity has the following attributes:

- **crew_id:** is the primary key of the entity and its type is integer.

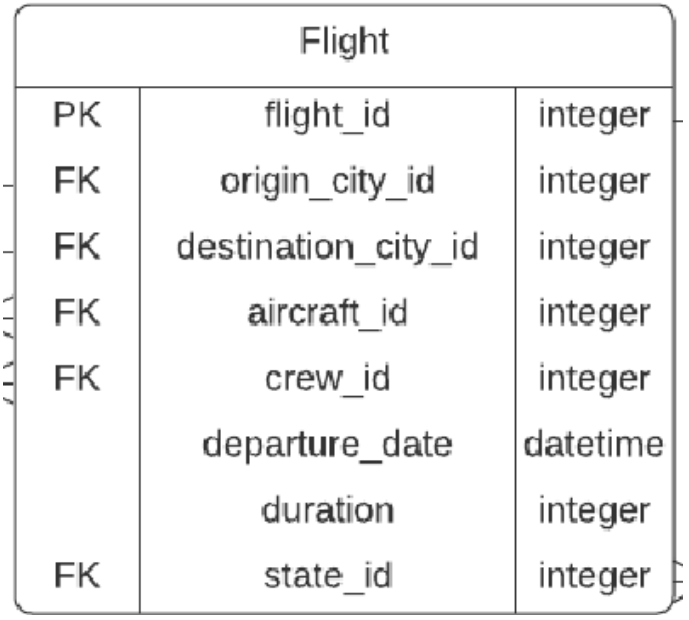
Crew		
PK	crew_id	Tipo

- **Flight Entity and Attributes**

This entity stores information about the flights scheduled by the airline. This entity has the following attributes:

- **flight_id:** is the primary key of the entity and its type is integer
- **origin_city_id:** is a foreign key which references the entity city and represents the id of the origin city of the flight.
- **destination_city_id:** is a foreign key which references the entity city and represents the id of the destination city of the flight.

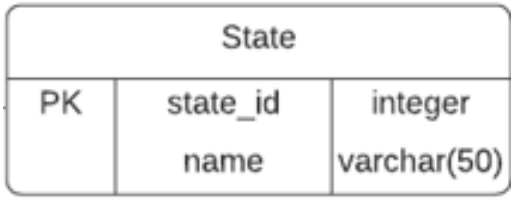
- **aircraft_id:** is a foreign key which references the Aircraft entity and represents the aircraft that is used to carry out the flight.
- **crew_id:** is a foreign key which references the crew entity and let know which will be the crew of the flight.
- **departure_date:** is a field that indicates the date and time when the flight will be carried out. Its type is datetime
- **duration:** is a field that represents the duration of the flight in minutes.
- **state_id:** is a foreign key which references the state entity and let know the state of the flight (which could be for example scheduled, onboarding, ongoing, finished or canceled).



- **State Entity and Attributes**

This entity stores information about the different states that a flight can have. This entity has the following attributes:

- **state_id:** is the primary key of the entity and its type is integer
- **name:** is the name of the state of a flight.Its type is varchar.

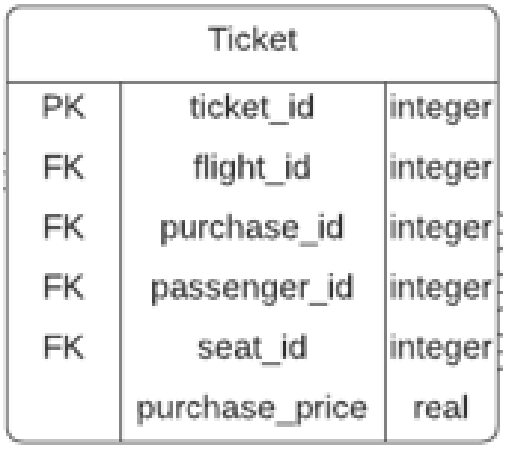


- **Ticket Entity and Attributes**

This entity stores information about the tickets which are sold in a flight. This entity has the following attributes:

- **ticket_id:** is the primary key of the entity and its type is integer
- **flight_id:** is a foreign key which references the Flight entity and represents the flight to which the ticket belongs.
- **purchase_id:** is a foreign key which references the entity Purchase and represents the purchase which contains that ticket (if this value is null it means that the ticket is not sold)

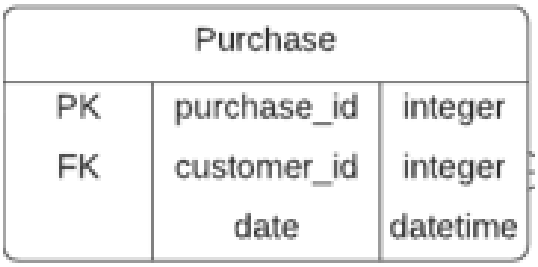
- **passenger_id:** is a foreign key which references the Passenger entity and represents the passenger which is the owner of the ticket.
- **seat_id:** is a foreign key which references the Seat entity and let know which seat of the aircraft corresponds to the ticket.
- **departure_date:** is a field that indicates the date and time when the flight will be carried out. Its type is datetime
- **purchase_price:** is a field that represents the price of the seat at the moment of the purchase (this is a denormalization because if the price of the seat changes then the price of the ticket will not be consistent with the purchase price).



- **Purchase Entity and Attributes**

This entity stores information about the purchases carried out by customers. This entity has the following attributes:

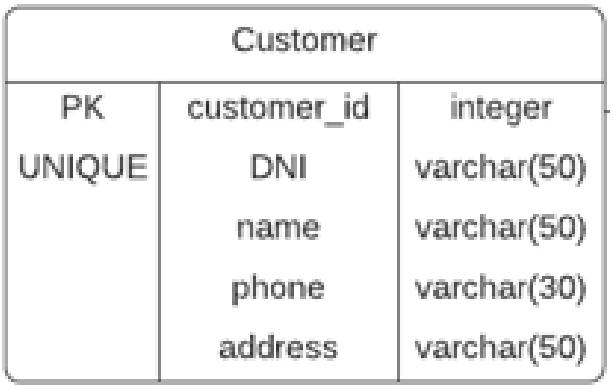
- **purchase_id:** is the primary key of the entity and its type is integer
- **customer_id:** is a foreign key which references the Customer entity and let know who is the customer that carried out the purchase.
- **date:** this field represents the date of the purchase.



- **Customer Entity and Attributes**

This entity stores information about the customers of the airline. This entity has the following attributes:

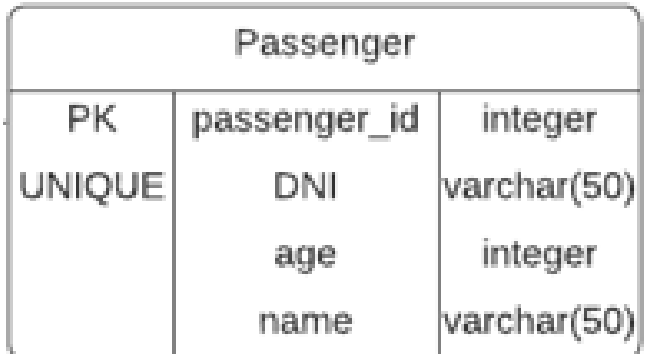
- **customer_id:** is the primary key of the entity and its type is integer
- **DNI:** . is the identification document of the customer. Its type is varchar
- **name:** this attribute represents the name of the customer.
- **phone:** this attribute represents the phone of the customer.
- **address:** this attribute represents the address of the customer.



- **Passenger Entity and Attributes**

This entity stores information about the passengers of the airline. This entity has the following attributes:

- **passenger_id:** is the primary key of the entity and its type is integer
- **DNI:** . is the identification document of the passenger. Its type is varchar
- **name:** this attribute represents the name of the passenger.
- **age:** this attribute represents the age of the passenger



Description of relationship and functions:

For every one of the use cases we are going to define some functions that must be carried out to achieve the accomplishment of the use case. These functions use the relationships in the ER model to be carried out. Notice that the relationships are referenced by numbers that can be viewed in the ER diagram specified later.

- **Register a flight Use Case:**
  To carry out this use case is important to execute this functions:
  - **Associate the flight with an aircraft:** this function is possible thanks to the many to one relationship referenced as 4 in the diagram. This relationship allows a flight to have only one aircraft, and on the other hand an aircraft can be assigned to multiple flights (but not at the same time).
  - **Associate the flight with a crew:** this function is possible thanks to the many to one relationship referenced as 5 in the diagram. This relationship allows a flight to have only one crew, and on the other hand a crew can be assigned to multiple flights (but not at the same time). Also, the crew can reference the pilots and airhostesses with the relationships referenced by the numbers 6 and 7.
  - **Associate the flight with a destination and origin city:** this function is possible thanks to the many to one relationships referenced as 12 and 13 in

the diagram. This relationship allows a flight to have only one origin city and only one destination city, and on the other hand these cities can be referenced by other flights.
    - **Associate the flight with the scheduled state:** this function is possible thanks to the many to one relationship referenced as 9 in the diagram. This relationship allows a flight to have only one state, and on the other hand allows a state to be referenced in many flights.
- **Register an Aircraft Use Case:**
  To carry out this use case is important to execute this functions:
    - **Associate the aircraft with an aircraft type:** this function is possible thanks to the many to one relationship referenced as 1 in the diagram. This relationship allows an aircraft to have only one aircraft type, and on the other hand an aircraft type can be assigned to multiple aircraft.

- **Register a Crew (Pilots and airhostesses) Use Case:**
  To carry out this use case is important to execute this functions:
    - **Associate the pilot with a crew:** this function is possible thanks to the many to one relationship referenced as 6 in the diagram. This relationship allows a pilot to belong to only a crew, and on the other hand a crew can have several pilots.
    - **Associate the airhostess with a crew:** this function is possible thanks to the many to one relationship referenced as 7 in the diagram. This relationship allows a airhostess to belong to only a crew, and on the other hand a crew can have several airhostesses.

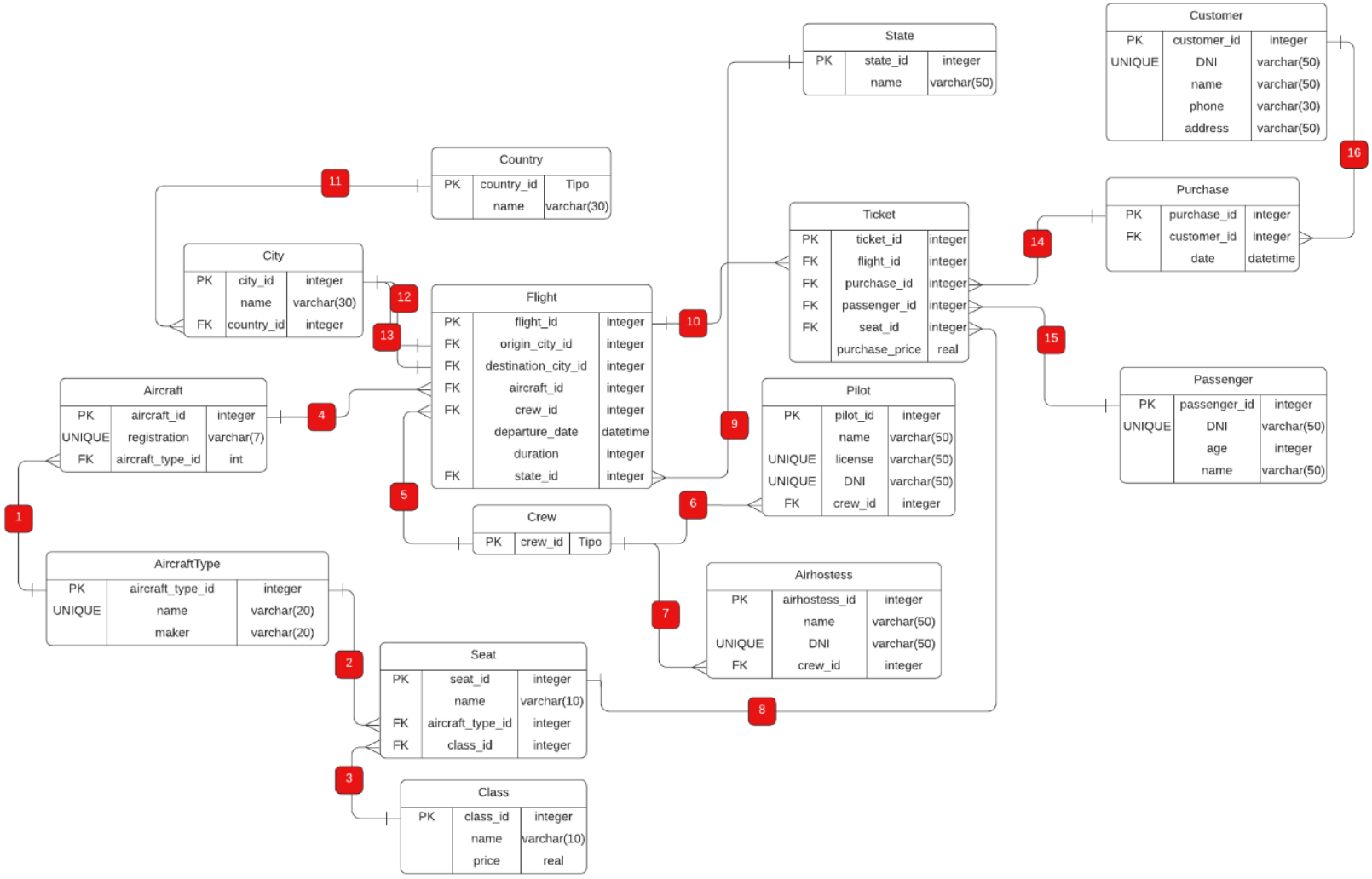
- **Sell Tickets of a Flight and Make a Purchase of tickets Use Case**
  To carry out this use case is important to execute this functions
    - **Create a ticket of a flight without passenger assigned:** this function is possible thanks to the fact that the passenger_id attribute of the ticket entity can be defined as null when the ticket is created.
    - **Associate the ticket with flight:** this function is possible thanks to the many to one relationship referenced as 10 in the diagram. This relationship allows a ticket to belong to only flight, and on the other hand a flight can have several tickets.
    - **Associate the ticket with a seat:** this function is possible thanks to the many to one relationship referenced as 8 in the diagram. This relationship allows that a seat can be referenced in many tickets (but we must define as a unique identifier the pair seat_id and flight_id because in one flight the seat only can be assigned to one ticket). Also the purchase price can be defined querying with the relationship 3 the current price of the seat.
    - **Associate the customer with a purchase and the purchase with many tickets:**this function is possible thanks to the many to one relationships referenced as 16 and 14 in the diagram. These relationships allow the customer to make a purchase of many tickets.
    - **Associate the ticket with a passenger:**this function is possible thanks to the many to one relationship referenced as 15 in the diagram. This relationship allow to associate a ticket with a passenger, and also allow a passenger to be associated with many tickets (but in different flights).

- **Cancel a Flight Use Case:**

    - **Associate the flight with the canceled state:** this function is possible thanks to the many to one relationship referenced as 9 in the diagram. This relationship allows a flight to have only one state, and on the other hand allows a state to be referenced in many flights.

- **Change the price of a seat class Use Case:**
    - **Change the field price in the class entity:** It's allowed by the nature of the database to allow updating this field. And to preserve the original price of purchases its present the attribute purchase_price in the ticket entity

ER Diagram:


State
PK state_id integer
name varchar(50)
Customer
PK customer_id integer
UNIQUE DNI varchar(50)
name varchar(50)
phone varchar(30)
address varchar(50)
Country
PK country_id Tipo
name varchar(30)
Purchase
PK purchase_id integer
FK customer_id integer
date datetime
Ticket
PK ticket_id integer
FK flight_id integer
FK purchase_id integer
FK passenger_id integer
FK seat_id integer
purchase_price real
City
PK city_id integer
name varchar(30)
FK country_id integer
Flight
PK flight_id integer
FK origin_city_id integer
FK destination_city_id integer
FK aircraft_id integer
FK crew_id integer
departure_date datetime
duration integer
FK state_id integer
Passenger
PK passenger_id integer
UNIQUE DNI varchar(50)
age integer
name varchar(50)
Aircraft
PK aircraft_id integer
UNIQUE registration varchar(7)
FK aircraft_type_id int
Pilot
PK pilot_id integer
name varchar(50)
UNIQUE license varchar(50)
UNIQUE DNI varchar(50)
FK crew_id integer
Crew
PK crew_id Tipo
AircraftType
PK aircraft_type_id integer
UNIQUE name varchar(20)
maker varchar(20)
Airhostess
PK airhostess_id integer
name varchar(50)
UNIQUE DNI varchar(50)
FK crew_id integer
Seat
PK seat_id integer
name varchar(10)
FK aircraft_type_id integer
FK class_id integer
Class
PK class_id integer
name varchar(10)
price real
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16