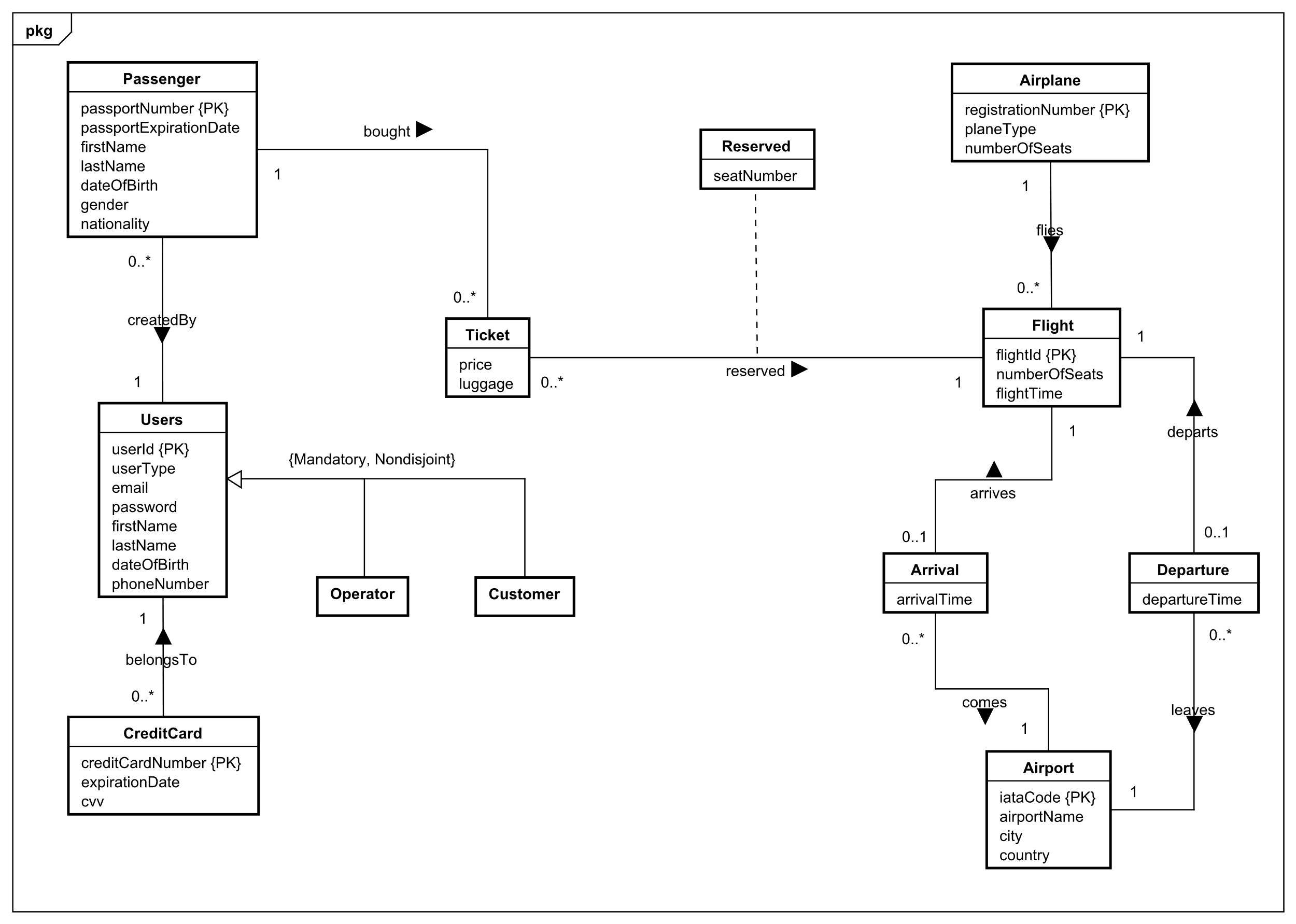
**Database**

**Conceptual Model**

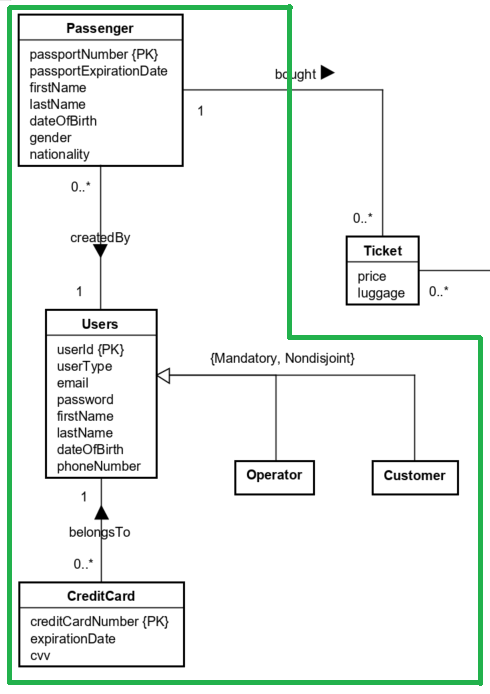
The conceptual model is heavily influenced by the domain model.



Entity/Relationship modelling models a business situation by describing the relevant entities and their relationships. For example, *Passenger* and *Ticket* are entities, and *boughtBy* describes the relationship between the entities.

This conceptual model is in third normal form. Normalization is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise (Connolly and Begg, 2015). It is in third normal form since it satisfies the first and second normal form, and no non-primary key attribute are transitively dependent on the primary key.

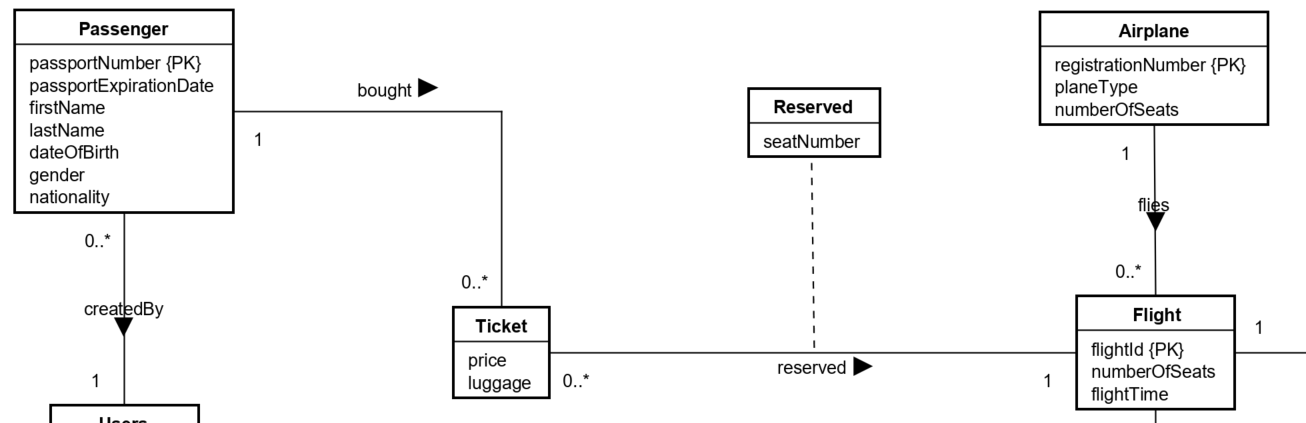
To better explain the conceptual model, it will be separated into three sections, Section 1, Section 2 Section 3.



In Section 1, the strong entities present are the *CreditCard, Users* and *Passenger* entities. The *Users* entity holds the account information of users that use the system, while the *CreditCard* entity holds the credit card information of the users. The relationship between these two strong entities is called *belongsTo* and is a one-to-many relationship. The reasoning behind the one-to-many relationship is that one user can store multiple credit cards in the system and choose which card they prefer to use. The extended entity/relationship modelling is used between *Users*, *Customer* and *Operator*. *Users* acts as the superclass to the subclasses *Customer* and *Operator.* The participation constraint is mandatory and nondisjoint, which means it has a single relation with one or more discriminators to distinguish the type of each tuple (Connolly and Begg, 2015).

The other strong entity in this section is the *Passenger* entity. The *Passengers* entity has the information of passengers, with the *passportNumber* as the primary key The *Passenger* entity is created for the handling of creating multiple tickets by one customer. For example, if one customer decides to buy tickets for 3 people for one flight, each of the 3 people will be regarded as individual passengers, with their own *passportNumber,* names and so on. The relationship, *createdBy*, describes the one-to-many relationship, since multiple passengers can be created by a user, but a passenger can only be created by one user.

The Section 2 primarily deals with the handling of passengers who buy tickets.

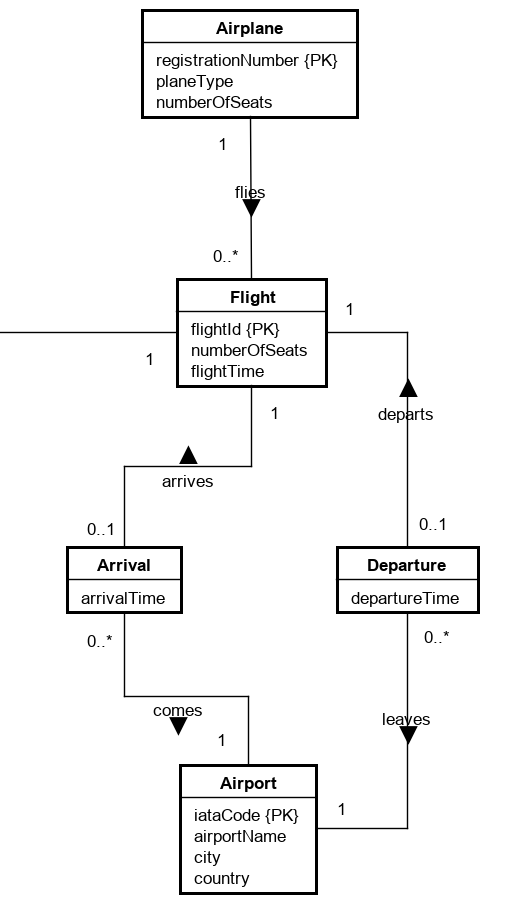


The new strong entities in this are *Airplane* and *Flight.* The *Airplane* entity describes the airplane, *Flight* describes the flight information. The relationship between *Airplane* and *Flight*, flies, is a one-to-many relationship because a flight can only have one airplane flying and an airplane can fly zero to many flights.

The *Ticket* is a weak entity. *Ticket* is a weak entity becausepassengers’ *passportNumber* can be stored alongside *flightId* from the *Flight* entity. The *passportNumber* and *flightId* will act as the primary keys, which then prevents passengers with the same *passportNumber* to buy more than one ticket for the same flight. *Ticket* and *Passenger* are related by the relationship *boughtBy*, which is a one-to-many relationship. It is a one-to-many relationship since a passenger can buy multiple tickets, but a ticket must belong to only one passenger.

How the reservation of seats is handled can be seen in this section as well. Each airplane has a fixed number of seats. The attribute, *numberOfSeats,* in the *Flight* entity shows how many seats are left that are unreserved in the airplane. The relationship, *reserved,* describes the reservation of seats. The attribute, *seatNumber,* indicates the seat number belonging to a particular flight reserved by a passenger. As seats are reserved, the attribute *numberOfSeats* in *Flight* should decrease.

In section 3 are 3 strong entities and 2 weak entities. Strong entities present are *Airplane*, *Flight* and *Airport* and weak entities are *Departure* and *Arrival*.



The *Flight* entity holds the information of flights that use the system, while *Airplane* holds information about airplanes. The relationship between these strong entities is called *flies*, and it is a one-to-many relationship. The reason, why it is a one-to-many relationship, is because *Flight* can store only one airplane (as the one flight will be done with one airplane) and *Airplane* can store multiple flights (as the airplane can be used for many flights). One-to-many relationship is also used between *Airport*-*Arrival* and *Airport*-*Departure.*

*Arrival* and *Departure* are weak entities, because there is no need for primary keys. There is a one-to-one relationship between strong entity *Flight* and weak entities *Arrival* and *Departure*. These one-to-one relationships are called *arrives* and *departs*. This relationship is used, because each flight can have only one departure and one arrival, and that Arrival/Departure can belong to only one *Flight*.

**Logical model**

The logical model will be used as the basis for the creation of the physical database. Deriving relations for logical data model was done by following the following steps (Connolly and Begg, 2015):

1. strong entity types
2. weak entity types
3. one-to-many (1:\*) binary relationship types
4. one-to-one (1:1) binary relationship types
5. one-to-one (1:1) recursive relationship types
6. superclass/subclass relationship types
7. many-to-many (\*:\*) binary relationship types
8. complex relationship types
9. multi-valued attributes

An example for each of the steps will be showcased, but since there were no one-to-one (1:1) recursive relationship types, many-to-many (\*:\*) binary relationship types, complex relationship types, and multi-valued attributes in the conceptual model, steps 5, 7, 8, and 9 are skipped.

Only one example for each step will be shown.

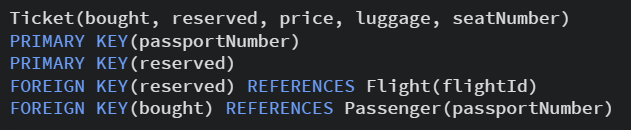
**Step 1:** Strong entity types.



An entity *Users* has attributes *userId, userType, email, password, firstName, lastName, dateOfBirth, phoneNumber,* with the *userId* being the primary key.

**Step 2:** Weak entity types AND **Step 3:** One-to-many (1:\*) binary relationship types

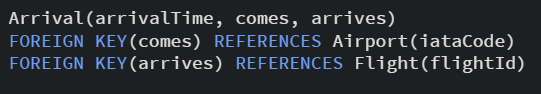
An interesting weak entity is the *Ticket.*



The *Ticket* entity only had the price. But since its relationships with the *Passenger* and *Flight* entities are one-to-many, the *Passenger* and *Flight* entities are designated as the parent entities (which is Step 3). This makes the *Ticket* the child entity. This means that the *fligthId* and *passportNumber* will be foreign keys in the *Ticket* entity. The relationship attribute, *seatNumber,* is also added as an attribute.

**Step 4:** One-to-one (1:1) binary relationship types

The relationship between *Flight* and *Arrival* is a one-to-one type.



The *arrivalTime* will still be an attribute, but the *flightId* will be added as a foreign key. The other foreign showed in the code above comes from the one-to-many relationship with *Airport.*

**Step 6:** Superclass/subclass relationship types



Since the constraint is mandatory and nondisjoint, just one table, *Users,* will be needed.