Solutions:

1. We create three tables.

```
CREATE TABLE Flights (
      date              DATE,
      flight_no         CHAR(5),
      from              CHAR(20),
      to                CHAR(20),
      plane_id          INTEGER NOT NULL,
      PRIMARY KEY (date, flight_no),
      FOREIGN KEY (plane_id) REFERENCES
            Planes (transponder_no) ON DELETE NO ACTION
);
```

plane_id may not be NULL because each flight must be assigned one plane (participation constraint). A Flight instance may be assigned to only one Plane instance (one to many relationship), we can incorporate the Uses relation in the Flights table itself. The "`ON DELETE NO ACTION`" clause prevents a Plane instance from being deleted before the Flights that use the Plane get assigned a different one.

```
CREATE TABLE Planes (
      transponder_no  INTEGER,
      type            CHAR(20),
      PRIMARY KEY (transponder_no)
);
```

transponder_no uniquely identifies a plane.

```
CREATE TABLE Seat_assignment (
      seat_no         CHAR(3),
      leg_room        CHAR(20),
      plane_id        INTEGER,
      PRIMARY KEY (plane_id, seat_no),
      FOREIGN KEY  (plane_id) REFERENCES
            Planes (transponder_no) ON DELETE CASCADE
);
```

Since Seats is a weak entity associated with a plane, we can merge the entity Seats and the relationship has_seats into a single table Seat_assignment. We allow cascaded deletes because Seats is a weak entity of the plane, and if the plane row is removed from the database the seats of that plane should be removed as well.

2. We denote the schema for Flights (*date, flight_no, from, to, plane_id*) as DFROT, Planes (*transponder_no, type*) as TY and Seat_assignment (*seat_no, leg_room, transponder_no*) as SLT.

   a) From the key constraints we get the following functional dependencies:

      DF → DFROT
      T → TY
      TS → TSL

      Also because a flight always flies between a given pair of airports, we get

      F → RO

   b) We can obtain additional functional dependencies using Armstrong's axioms. Using decomposition and transitivity (DF → DFROT, T → TY) we can obtain:

      DF → Y

      Using decomposition, augmentation and transitivity (DF → DFROT, TS → TSL) we can obtain:

      DFS → L

      Reflexivity can give some more trivial FDs.

   c) The two additional conditions lead to more functional dependencies. The additional attribute *has_first_class* is denoted *as* H and the *seat_color* as C. The presence of first class in a flight is a function of the type of the plane, so in other words H is functionally determined by Y. Thus we get the condition Y → H, we know that T → Y so by transitivity

      T→ H.

      Since the color of the seat matches the exterior of the plane

      T → C.

      The last two functional dependencies can be used to modify the relational representation; since C is functionally determined by T, we make *seat_color* an attribute of Plane. We now have an attribute that depends on type, which is not a key of Planes. Therefore, we should create a new table Plane_type (type, has_first_class) to prevent redundancies (and therefore avoid anomalies).

3. We need to create three additional tables to represent the two entities and the relations travels_on.

```
CREATE TABLE Passengers (
        lastname        CHAR(20),
        firstname       CHAR(20),
        credit_card           INTEGER,
        PRIMARY KEY (lastname, firstname)
);

CREATE TABLE Frequent_flyer (
        lastname        CHAR(20),
        firstname       CHAR(20),
        acct_no         CHAR(16),
        miles           INTEGER,
        PRIMARY KEY (lastname, firstname),
        FOREIGN KEY (lastname, firstname) REFERENCES
                   Passengers ON DELETE CASCADE
);
```

Here the entity frequent flyer is in an ISA relationship with Passenger, so it is a subclass of passenger. When the row corresponding to a passenger is deleted from the Passengers table, then all the rows that referenced to it from the Frequent_flyer table should be deleted.

```
CREATE TABLE Reservation (
        reserved_on     DATE,
        date            DATE,
        flight_no       CHAR(5),
        plane           INTEGER,
        seat_no         char(3),
        lastname        CHAR(20),
        firstname       CHAR(20),
        PRIMARY KEY (date, flight_no, plane, seat_no),
        FOREIGN KEY (date, flight_no)
                REFERENCES Flights
                ON DELETE CASCADE,
        FOREIGN KEY (plane, seat_no)
                REFERENCES Seat_assignment (plane_id, seat_no)
                ON DELETE CASCADE,
        FOREIGN KEY (lastname, firstname)
                REFERENCES Passengers ON DELETE CASCADE
    );
```

This table basically links the relation Seat_assignment with flights (in the form of an aggregate) to the entity Passenger. So it references the primary keys of the tables corresponding to the same. Moreover because a given seat on a flight may be assigned to one passenger only, we make (*date, flight_no, plane, seat_no*) the primary key. On removal of any of the referenced records from the database the reservation does not hold, so we allow cascaded deletes.