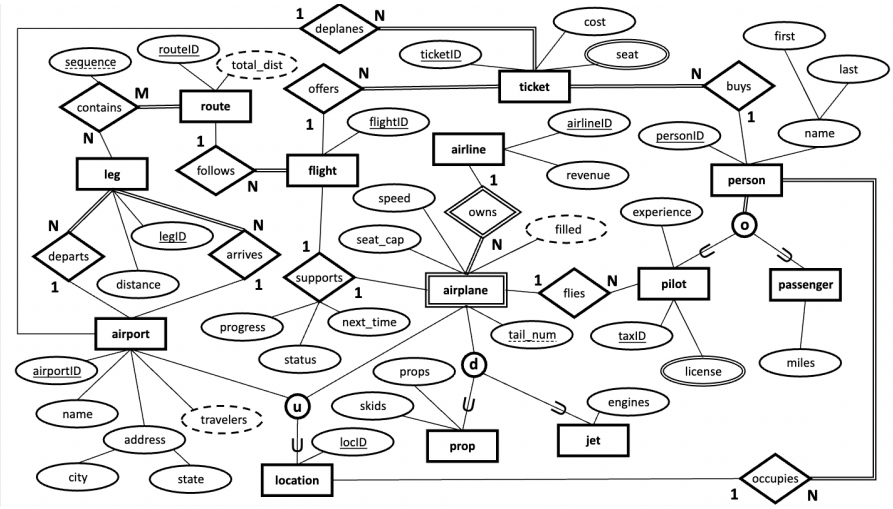
-- Sumana Chilakamarri - SQL Project
-- Flight Management Course Project Mechanics (v1.0)
-- Relational Schema & Unhandled Constraints



Relational Schema

 # Strong/Regular Entities:

location = (locID)

airline = (airlineID, revenue)

person = (personID, first_name, last_name, locID [fk14])
fk14: locID → location.locID, locID is non-null

route = (routeID)

flight = (flightID, routeID [fk11], airlineID, tail_num [fk6], progress, airplane_status, next_time)
fk6 → airlineID, tail_num → airplane.airlineID, airplane.tail_num
fk11: routeID → route.routeID, routeID is non-null

airport = (airportID, airport_name, city, state, locID [fk20])
fk20: locID → location.locID

ticket = (ticketID, cost, personID [fk7], airportID [fk9], flightID [fk10])
fk7: personID → person.personID, personID is non-null
fk9: airportID → airport.airportID, airportID is non-null
fk10: flightID → flight.flightID, flightID is non-null

seat = (ticketID [fk18], seat_number)
fk18: ticketID → ticket.ticketID

license = (personID [fk17], license_type)
fk17: personID → pilot.personID

leg = (legID, distance, depart_airportID[fk13], arrive_airportID[fk12])
fk13: depart_airportID → airport.airportID, depart_airportID is non-null
fk12: arrive_airportID → airport.airportID, arrive_ariportID is non-null

 # Weak Entity:

airplane = (airlineID [fk3], tail_num, speed, seat_cap, locID [fk19])
fk3: airlineID → airline.airlineID, airlineID is non-null
fk19: locID → location.locID

 # Specialization (Subclasses):

pilot = (personID [fk1], taxID, airlineID, tail_num [fk8], experience)
fk1: personID → person.personID
fk8: airlineID, tail_num → airplane.arilineID, airplane.tail_num

passenger = (personID [fk2], miles)
fk2: personID → person.personID

prop = (airlineID, tail_num [fk4], number_of_skids, number_of_propellers)
fk4: airlineID, tail_num → airplane.airlineID, airplane.tail_num

jet = (airlineID, tail_num [fk5], number_of_engines)
fk5: airlineID, tail_num → airplane.airlineID, airplane.tail_num

 # Many-to-Many:

contain = (routeID[fk15], legID[fk16], sequence)
fk15: routeID → route.routeID
fk16: legID → leg.legID, legID is non-null

 # Unhandled Constraints

Ensure that the number of passengers is within the seat capacity of the airplane
Ensure that each pilot has a valid license to fly the type of airplanes

```
# Ensure that the airplanes arrive at the departing airport before departure
time
# Ensure that the arrival and departure times are in accordance with the
status of the plane (e.g. if the plane is delayed, it’s arrival time at the
arrival airport is updated accordingly)
# Ensure that person location can only be at plane or airport based
# Ensure that person should be either pilot or passenger or both
# Ensure that airplane can only support one flight at a time
# Ensure that airplanes can either be jet or prop if it has a type



-- Flight Management Course Project Mechanics (v1.0) STARTING SHELL
-- Views, Functions & Stored Procedures

# Throughout this phase, I used a variety of queries to create a starting
shell,
# stored procedures, views, etc., to get specific information

set global transaction isolation level serializable;
set global SQL_MODE = 'ANSI,TRADITIONAL';
set names utf8mb4;
set SQL_SAFE_UPDATES = 0;
set @thisDatabase = 'flight_management';

use flight_management;
--
-----------------------------------------------------------------------------
-- stored procedures and views
--
-----------------------------------------------------------------------------

-- [1] add_airplane()
--
-----------------------------------------------------------------------------
/* This stored procedure creates a new airplane.  A new airplane must be
sponsored
by an existing airline, and must have a unique tail number for that airline.
username.  An airplane must also have a non-zero seat capacity and speed. An
airplane
might also have other factors depending on it's type, like skids or some
number
of engines.  Finally, an airplane must have a database-wide unique location if
it will be used to carry passengers. */
```

```
--
-------------------------------------------------------------------------------
drop procedure if exists add_airplane;
delimiter //
create procedure add_airplane (in ip_airlineID varchar(50), in ip_tail_num
varchar(50),
      in ip_seat_capacity integer, in ip_speed integer, in ip_locationID
varchar(50),
    in ip_plane_type varchar(100), in ip_skids boolean, in ip_propellers
integer,
    in ip_jet_engines integer)
sp_main: begin
      -- ensure that the airplane is sponsored by an existing airline
    if ip_airlineID not in (select airlineID from airline) then
            leave sp_main;
      end if;

    -- ensure that the airplane have a unique tail number for that airline
    if (ip_airlineID, ip_tail_num) in (select airlineID, tail_num from
airplane) then
            leave sp_main;
      end if;

    -- ensure that the airplane have a non-zero seat capacity and speed
    if ip_seat_capacity=0 or ip_speed=0 then
            leave sp_main;
      end if;

    -- ensure that the airplane an airplane have a database-wide unique
location if it will be used to carry passengers
    if ip_locationID is not null and ip_locationID in (select distinct
locationID from airplane) then
            leave sp_main;
      end if;
    if ip_locationID is not null and ip_locationID in (select * from location
where locationID like 'plane%') then
            leave sp_main;
      end if;

    -- insert the airplane if it pass above guards
      insert into airplane (airlineID, tail_num, seat_capacity, speed,
locationID, plane_type, skids,propellers, jet_engines)
    values (ip_airlineID, ip_tail_num, ip_seat_capacity, ip_speed,
ip_locationID, ip_plane_type, ip_skids, ip_propellers, ip_jet_engines);
```

```
    -- insert the locationID if the plan will be used to carry passengers
    if ip_locationID is not null and ip_locationID not in (select * from
location where locationID like 'plane%') then
            insert into location(locationID)
        values (ip_locationID);
      end if;


end //
delimiter ;

-- [2] add_airport()
--
-------------------------------------------------------------------------------
/* This stored procedure creates a new airport.  A new airport must have a
unique
identifier along with a database-wide unique location if it will be used to
support
airplane takeoffs and landings.  An airport may have a longer, more
descriptive
name.  An airport must also have a city and state designation. */
--
-------------------------------------------------------------------------------
drop procedure if exists add_airport;
delimiter //
create procedure add_airport (in ip_airportID char(3), in ip_airport_name
varchar(200),
    in ip_city varchar(100), in ip_state char(2), in ip_locationID
varchar(50))
sp_main: begin
      -- ensure that the new airport must have a unique id
    if ip_airportID is null or ip_airportID in (select airportID from airport)
then
            leave sp_main;
      end if;

    -- ensure that the new airport has a database-wide unique locationID if it
will be used to support airplane takeoffs and landings
    if ip_locationID is not null and ip_locationID in (select distinct
locationID from airport) then
            leave sp_main;
      end if;
```

```
    if ip_locationID is not null and ip_locationID in (select * from location
where locationID like 'port%') then
            leave sp_main;
      end if;

    -- ensure that the new airport has a city and state designation
    if ip_city is null or ip_state is null then
            leave sp_main;
      end if;

    -- insert the airport if it pass above guards
    insert into airport (airportID, airport_name, city, state, locationID)
    values (ip_airportID, ip_airport_name, ip_city, ip_state, ip_locationID);

    -- insert the locationID if it will be used to support airplane takeoffs
and landings
    if ip_locationID is not null and ip_locationID not in (select * from
location where locationID like 'port%') then
            insert into location(locationID)
        values (ip_locationID);
      end if;

end //
delimiter ;


-- [3] add_person()
--
-------------------------------------------------------------------------------
/* This stored procedure creates a new person.  A new person must reference a
unique
identifier along with a database-wide unique location used to determine where
the
person is currently located: either at an airport, or on an airplane, at any
given
time.  A person may have a first and last name as well.

Also, a person can hold a pilot role, a passenger role, or both roles.  As a
pilot,
a person must have a tax identifier to receive pay, and an experience level.
Also,
a pilot might be assigned to a specific airplane as part of the flight crew.
As a
passenger, a person will have some amount of frequent flyer miles. */
```

```
--
-------------------------------------------------------------------------------
drop procedure if exists add_person;
delimiter //
create procedure add_person (in ip_personID varchar(50), in ip_first_name
varchar(100),
    in ip_last_name varchar(100), in ip_locationID varchar(50), in ip_taxID
varchar(50),
    in ip_experience integer, in ip_flying_airline varchar(50), in
ip_flying_tail varchar(50),
    in ip_miles integer)
sp_main: begin

      -- ensure that the new person has a unique id
    if ip_personID is null or ip_personID in (select personID from person)
then
            leave sp_main;
      end if;

    -- ensuere that the new person has locationID either at the airport or in
the airpolane
    if ip_locationID is null or ip_locationID not in (select locationID from
location) then
            leave sp_main;
      end if;

    -- insert the person if it passes above guards
      insert into person (personID, first_name, last_name, locationID)
    values (ip_personID, ip_first_name, ip_last_name, ip_locationID);


    -- insert the person into pilot if he/she is a pilot
    -- as a pilot, a person must have a tax identifier
    if ip_taxID is not null then
            insert into pilot(personID, taxID, experience, flying_airline,
flying_tail)
        values (ip_personID, ip_taxID, ip_experience, ip_flying_airline,
ip_flying_tail);
      end if;

    -- insert the person into passenger if he/she is a passenger
    -- as a passenger, the person will have some amount of frequent flyer
miles
    if ip_miles is not null then
```

```
            insert into passenger(personID, miles)
        values (ip_personID, ip_miles);
      end if;

end //
delimiter ;


-- [4] grant_pilot_license()
--
-------------------------------------------------------------------------------
/* This stored procedure creates a new pilot license.  The license must
reference
a valid pilot, and must be a new/unique type of license for that pilot. */
--
-------------------------------------------------------------------------------
drop procedure if exists grant_pilot_license;
delimiter //
create procedure grant_pilot_license (in ip_personID varchar(50), in
ip_license varchar(100))
sp_main: begin

      -- ensure that the license refer to a valid pilot
    if ip_personID not in (select personID from pilot) then
            leave sp_main;
      end if;

    -- ensure that it is a new/unique type of license for that pilot
    if ip_license in (select license from pilot_licenses where
personID=ip_personID) then
            leave sp_main;
      end if;

      -- insert the new license for the valid pilot if passes guards above
      insert into pilot_licenses (personID, license)
      values (ip_personID, ip_license);

end //
delimiter ;


-- [5] offer_flight()
--
-------------------------------------------------------------------------------
```

```
/* This stored procedure creates a new flight.  The flight can be defined
before
an airplane has been assigned for support, but it must have a valid route.
Once
an airplane has been assigned, we must also track where the airplane is along
the route, whether it is in flight or on the ground, and when the next action
-
takeoff or landing - will occur. */
--
--------------------------------------------------------------------------------
drop procedure if exists offer_flight;
delimiter //
create procedure offer_flight (in ip_flightID varchar(50), in ip_routeID
varchar(50),
    in ip_support_airline varchar(50), in ip_support_tail varchar(50), in
ip_progress integer,
    in ip_airplane_status varchar(100), in ip_next_time time)
sp_main: begin

      -- ensure that the flight has a valid route
    if ip_routeID not in (select routeID from route) then
            leave sp_main;
      end if;

      insert into flight (flightID, routeID, support_airline, support_tail,
progress, airplane_status, next_time)
    values (ip_flightID, ip_routeID, ip_support_airline, ip_support_tail,
ip_progress, ip_airplane_status, ip_next_time);

end //
delimiter ;


-- [6] purchase_ticket_and_seat()
--
--------------------------------------------------------------------------------
/* This stored procedure creates a new ticket.  The cost of the flight is
optional
since it might have been a gift, purchased with frequent flyer miles, etc.
Each
flight must be tied to a valid person for a valid flight.  Also, we will make
the
(hopefully simplifying) assumption that the departure airport for the ticket
will
```

```
be the airport at which the traveler is currently located.  The ticket must
also
explicitly list the destination airport, which can be an airport before the
final
airport on the route.  Finally, the seat must be unoccupied. */
--
--------------------------------------------------------------------------------
drop procedure if exists purchase_ticket_and_seat;
delimiter //
create procedure purchase_ticket_and_seat (in ip_ticketID varchar(50), in
ip_cost integer,
      in ip_carrier varchar(50), in ip_customer varchar(50), in ip_deplane_at
char(3),
    in ip_seat_number varchar(50))
sp_main: begin

      if(ip_customer in (select personID from person) and ip_carrier in
(select flightID from flight) and ip_deplane_at is not null
            and ip_deplane_at not in (select flightID from flight where
routeID in (select routeID from route_path where legID in (select legID from
leg where ip_deplane_at =arrival )))) then
            INSERT ignore INTO ticket (ticketID, cost, carrier, customer,
deplane_at)
            VALUES (ip_ticketID, ip_cost, ip_carrier, ip_customer,
ip_deplane_at);
      end if;

      if(ip_seat_number not in (select seat_number from ticket_seats)) then
            INSERT ignore INTO ticket_seats (ticketID, seat_number)
            VALUES (ip_ticketID, ip_seat_number);
      end if;
end //
delimiter ;


-- [7] add_update_leg()
--
--------------------------------------------------------------------------------
/* This stored procedure creates a new leg as specified.  However, if a leg
from
the departure airport to the arrival airport already exists, then don't create
a
new leg - instead, update the existence of the current leg while keeping the
existing
```

```
identifier.  Also, all legs must be symmetric.  If a leg in the opposite
direction
exists, then update the distance to ensure that it is equivalent.   */
--
--------------------------------------------------------------------------------
drop procedure if exists add_update_leg;
delimiter //
create procedure add_update_leg (in ip_legID varchar(50), in ip_distance
integer,
    in ip_departure char(3), in ip_arrival char(3))
sp_main: begin
      -- if a leg from the departure airport to the arrival airport exist
    if exists (select * from leg where departure=ip_departure and
arrival=ip_arrival) then
            -- update the existence of the current leg while keeping the
existing id
        update leg set distance=ip_distance where departure=ip_departure and
arrival=ip_arrival;
      -- else the leg from the departure airport to the arrival airport not
exist
      else
            -- insert the new leg as specified
        insert into leg (legID, distance, departure, arrival) values
(ip_legID, ip_distance, ip_departure, ip_arrival);
      end if;

    -- if a leg in the opposite direction exists
    if exists (select * from leg where departure=ip_arrival and
arrival=ip_departure) then
            -- update the distance to ensure that it is equivalent
        update leg set distance=ip_distance where departure=ip_arrival and
arrival=ip_departure;
      end if;

end //
delimiter ;

-- [8] start_route()
--
--------------------------------------------------------------------------------
/* This stored procedure creates the first leg of a new route.  Routes in our
system must be created in the sequential order of the legs.  The first leg of
the route can be any valid leg. */
```

```
--
-------------------------------------------------------------------------------
drop procedure if exists start_route;
delimiter //
create procedure start_route (in ip_routeID varchar(50), in ip_legID
varchar(50))
sp_main: begin
      -- insert the new route into route table
      insert into route (routeID) values (ip_routeID);

    -- insert the first leg of a new route into route_path table
    insert into route_path (routeID, legID, sequence) values (ip_routeID,
ip_legID, 1);

end //
delimiter ;


-- [9] extend_route()
--
-------------------------------------------------------------------------------
/* This stored procedure adds another leg to the end of an existing route.
Routes
in our system must be created in the sequential order of the legs, and the
route
must be contiguous: the departure airport of this leg must be the same as the
arrival airport of the previous leg. */
--
-------------------------------------------------------------------------------
drop procedure if exists extend_route;
delimiter //
create procedure extend_route (in ip_routeID varchar(50), in ip_legID
varchar(50))
sp_main: begin

if ip_legID is null or ip_routeID is null then leave sp_main;
end if;

-- max sequence
set @max_sequence = (select max(sequence) from route_path where ip_routeID =
routeID group by routeID);

-- departure location for new leg
set @new_leg_departure = (select departure from leg where ip_legID = legID);
```

```
-- leg of RouteID
set @leg = (select legID from route_path where ip_routeID = routeID and
@max_sequence = sequence);
if @leg is null then leave sp_main;
end if;

set @route_arrival = (select arrival from leg where @leg = legID);
if @route_arrival != @new_leg_departure then leave sp_main;
end if;

if ip_legID not in (select legID from route_path) then insert into route_path
values (ip_routeID, ip_legID, @max_sequence + 1);
end if;


end //

delimiter ;




-- [10] flight_landing()
--
-------------------------------------------------------------------------------
/* This stored procedure updates the state for a flight landing at the next
airport
along it's route.  The time for the flight should be moved one hour into the
future
to allow for the flight to be checked, refueled, restocked, etc. for the next
leg
of travel.  Also, the pilots of the flight should receive increased
experience, and
the passengers should have their frequent flyer miles updated. */
--
-------------------------------------------------------------------------------
drop procedure if exists flight_landing;
delimiter //
create procedure flight_landing (in ip_flightID varchar(50))
sp_main: begin

      declare distance int default 0;
```

```
      if ip_flightID in (select flightID from flight) then
            UPDATE flight
            SET airplane_status = 'on_ground', next_time = ADDTIME(next_time,
'01:00:00')
        where flightID=ip_flightID;
      end if;

      # adding experience to pilot
      if ip_flightID in (select flightID from flight where support_tail in
(select flying_tail from pilot)) then
            UPDATE pilot
            SET experience = experience + 1
        where personID in (select * from (select p.personID from flight as f
join pilot as p on f.support_airline=p.flying_airline and
f.support_tail=p.flying_tail where flightID = ip_flightID) as temp);
      end if;

      set distance = (select l.distance from flight as f join route_path as r
on f.routeID=r.routeID join leg as l on r.legID=l.legID where
f.flightID=ip_flightID and f.progress=r.sequence);
      UPDATE passenger
      SET miles = miles + distance
      WHERE personID in (select t.customer from ticket as t join flight as f
on t.carrier=f.flightID where t.carrier = ip_flightID);


end //
delimiter ;



-- [11] flight_takeoff()
--
-------------------------------------------------------------------------------
/* This stored procedure updates the state for a flight taking off from its
current
airport towards the next airport along it's route.  The time for the next leg
of
the flight must be calculated based on the distance and the speed of the
airplane.
And we must also ensure that propeller driven planes have at least one pilot
assigned, while jets must have a minimum of two pilots. If the flight cannot
take
```

```
off because of a pilot shortage, then the flight must be delayed for 30
minutes. */
--
-------------------------------------------------------------------------
drop procedure if exists flight_takeoff;
delimiter //
create procedure flight_takeoff (in ip_flightID varchar(50))
sp_main: begin

      declare next_duration int default 0;
    -- check whether the flight exists or not
      if (ip_flightID not in (select flightID from flight)) then
            leave sp_main;
      end if;

      -- the jet airplane has less than 2 pilot
      if (select plane_type from flight as f join airplane as a on
f.support_airline=a.airlineID and f.support_tail=a.tail_num where
f.flightID=ip_flightID) = 'jet'
            and (select COUNT(p.personID) from flight as f join pilot as p on
f.support_airline=p.flying_airline and f.support_tail=p.flying_tail where
f.flightID=ip_flightID)<2 then
        -- update the next_time be delayed for 30 minutes
        update flight set next_time=next_time + interval 30 minute where
flightID=ip_flightID;
        leave sp_main;
      end if;

    -- the propeller driven airplane has less than 1 pilot
    if (select plane_type from flight as f join airplane as a on
f.support_airline=a.airlineID and f.support_tail=a.tail_num where
f.flightID=ip_flightID) = 'prop'
            and (select COUNT(p.personID) from flight as f join pilot as p on
f.support_airline=p.flying_airline and f.support_tail=p.flying_tail where
f.flightID=ip_flightID)<1 then
        -- update the next_time be delayed for 30 minutes
        update flight set next_time=next_time + interval 30 minute where
flightID=ip_flightID;
        leave sp_main;
      end if;

    -- if there is no pilot shortage issue
    -- update the progress
    update flight set progress=progress+1 where flightID=ip_flightID;
```

```
    -- update the airplane_status
    update flight set airplane_status='in_flight' where flightID=ip_flightID;
    -- calculate the duration of next leg and store it as a variable
    set next_duration = (select l.distance/a.speed from flight as f join
route_path as r on f.routeID=r.routeID and f.progress=r.sequence join leg as l
on r.legID=l.legID join airplane as a on f.support_airline=a.airlineID and
f.support_tail=a.tail_num where f.flightID=ip_flightID);
    -- update the next_time of the flight based in the next_duration
    update flight set next_time = next_time + interval next_duration hour
where flight.flightID=ip_flightID;

end //
delimiter ;


-- [12] passengers_board()
--
-------------------------------------------------------------------------------
/* This stored procedure updates the state for passengers getting on a flight
at
its current airport.  The passengers must be at the airport and hold a valid
ticket
for the flight. */
--
-------------------------------------------------------------------------------
drop procedure if exists passengers_board;
delimiter //
create procedure passengers_board (in ip_flightID varchar(50))
sp_main: begin
      declare board_plane_loc varchar(10);

      -- check whether the flight exists or not
      if (ip_flightID not in (select flightID from flight)) then
            leave sp_main;
      end if;

    -- check the passengers are at the airport that the flight  is departing
    -- check the passengers hold the valid ticket or not
    if exists (select * from flight as f
                        join ticket as t on t.carrier=f.flightID
                        join person as p on t.customer=p.personID
                        where flightID=ip_flightID
                and p.locationID = (select a.locationID from flight as f join
route_path as r on r.routeID=f.routeID join leg as l on r.legID=l.legID join
```

```
airport as a on l.departure=a.airportID where flightID=ip_flightID and
f.progress+1=r.sequence)) then
            -- update the board_loc with the departing airplane of the flight
        set board_plane_loc = (select a.locationID from flight as f join
airplane as a on f.support_airline=a.airlineID and f.support_tail=a.tail_num
where flightID=ip_flightID);

        -- update the locationID of person from the departing airpot to the
airplane of the flight
        -- need to create a outer table for update the table in inner select
        update person
        set locationID = board_plane_loc
        where personID in (select * from (select t.customer from flight as f
                                                                    join ticket
as t on t.carrier=f.flightID
                                                                    join person
as p on t.customer=p.personID
                                                                    where
flightID=ip_flightID
                                                and p.locationID = (select
a.locationID from flight as f join route_path as r on r.routeID=f.routeID join
leg as l on r.legID=l.legID join airport as a on l.departure=a.airportID where
flightID=ip_flightID and f.progress+1=r.sequence)) as temp);

            end if;

end //
delimiter ;

-- [13] passengers_disembark()
--
-------------------------------------------------------------------------------
/* This stored procedure updates the state for passengers getting off of a
flight
at its current airport.  The passengers must be on that flight, and the flight
must
be located at the destination airport as referenced by the ticket. */
--
-------------------------------------------------------------------------------
drop procedure if exists passengers_disembark;
delimiter //
create procedure passengers_disembark (in ip_flightID varchar(50))
sp_main: begin
```

```
      declare deplane_loc varchar(10);

      -- check the passengers are ib the flight
    -- check the flight is on the ground
    -- check the flight is located at the destination airport
      if exists (select * from flight as f join ticket as t on t.carrier =
f.flightID join person as p on t.customer=p.personID join route_path as r on
f.routeID = r.routeID and f.progress=r.sequence join leg as l on
r.legID=l.legID
                        where flightID = ip_flightID
                        and airplane_status='on_ground'
                        and p.locationID = (select locationID from flight as f
join airplane as a on f.support_airline=a.airlineID and
f.support_tail=a.tail_num where flightID = ip_flightID)
                        and t.deplane_at=l.arrival) then
            -- update the deplane_loc with the deplane airport
        set deplane_loc = (select distinct a.locationID from flight as f join
ticket as t on t.carrier = f.flightID join route_path as r on f.routeID =
r.routeID and f.progress=r.sequence join leg as l on r.legID=l.legID join
airport as a on a.airportID=l.arrival where flightID =ip_flightID and
t.deplane_at=l.arrival);
        -- update the locationID of person from the airplane to the
destination airport
        -- need to create a outer table for update the table in inner select
        update person
        set locationID = deplane_loc
        where personID in (select * from (select t.customer from flight as f
join ticket as t on t.carrier = f.flightID join person as p on
t.customer=p.personID join route_path as r on f.routeID = r.routeID and
f.progress=r.sequence join leg as l on r.legID=l.legID where flightID
=ip_flightID
                                          and airplane_status='on_ground'
                                          and p.locationID = (select
locationID from flight as f join airplane as a on
f.support_airline=a.airlineID and f.support_tail=a.tail_num where flightID =
ip_flightID)
                                          and t.deplane_at=l.arrival) as
temp);

      end if;

end //
delimiter ;
```

```
-- [14] assign_pilot()
--
-------------------------------------------------------------------------------
/* This stored procedure assigns a pilot as part of the flight crew for a
given
airplane.  The pilot being assigned must have a license for that type of
airplane,
and must be at the same location as the flight.  Also, a pilot can only
support
one flight (i.e. one airplane) at a time.  The pilot must be assigned to the
flight
and have their location updated for the appropriate airplane. */
--
-------------------------------------------------------------------------------
drop procedure if exists assign_pilot;
delimiter //
create procedure assign_pilot (in ip_flightID varchar(50), ip_personID
varchar(50))
sp_main: begin

      declare plane_type varchar(10);
    declare pilot_loc varchar(10);

      -- ensure that the pilot have the license for the airplane of the flight
    set plane_type = (select a.plane_type from flight as f join airplane as a
on f.support_airline=a.airlineID and f.support_tail=a.tail_num where
flightID=ip_flightID);
    if plane_type not in (select license from pilot as p join pilot_licenses
as pl on p.personID=pl.personID where p.personID=ip_personID) then
            leave sp_main;
      end if;

    -- ensure that the pilot and the plane of the flight are at the location
    set pilot_loc = (select locationID from pilot as pil join person as per on
pil.personID=per.personID where pil.personID=ip_personID);
    if pilot_loc != (select a.locationID from flight as f join route_path as r
on r.routeID=f.routeID join leg as l on r.legID=l.legID join airport as a on
l.arrival=a.airportID where flightID=ip_flightID and f.progress=r.sequence)
then
            leave sp_main;
      end if;

    -- ensuer that the pilot is only assigned to one flight
```

```
    if (select flying_airline from pilot where personID=ip_personID) is not
null or (select flying_tail from pilot where personID=ip_personID) is not null
then
            leave sp_main;
      end if;

    -- update the flying_airline & flying_tail in the pilot if the guards all
passed
      update pilot
      set flying_airline=(select support_airline from flight where
flightID=ip_flightID)
      where personID=ip_personID;

      update pilot
      set flying_tail=(select support_tail from flight where
flightID=ip_flightID)
      where personID=ip_personID;

    -- update the locationID of the pilot in person table
    update person
      set locationID = (select a.locationID from flight as f join airplane as
a on f.support_airline=a.airlineID and f.support_tail=a.tail_num where
flightID=ip_flightID)
      where personID=ip_personID;

end //
delimiter ;

-- [15] recycle_crew()
--
-------------------------------------------------------------------------------
/* This stored procedure releases the assignments for a given flight crew.
The
flight must have ended, and all passengers must have disembarked. */
--
-------------------------------------------------------------------------------
drop procedure if exists recycle_crew;
delimiter //
create procedure recycle_crew (in ip_flightID varchar(50))
sp_main: begin

      declare end_leg int default 0;
    declare arrival_loc varchar(10);
    declare total_p int default 0;
```

```
    declare disembarked_p int default 0;

      -- ensure that the flight ended
    -- the progress of the flight equals to the last sequence of leg
    set end_leg = (select max(r.sequence) from flight as f join route_path as
r on f.routeID=r.routeID where flightID=ip_flightID);
      if end_leg != (select progress from flight where flightID=ip_flightID)
then
            leave sp_main;
      end if;

    -- ensure that the flight is on the ground
    if ip_flightID not in (select flightID from flight where
airplane_status='on_ground') then
            leave sp_main;
      end if;

    -- ensure that all the passenger disembark
    -- get the locationID of the arrival airport
    set arrival_loc = (select a.locationID from flight as f
                                    join route_path as r on
f.routeID=r.routeID
                        join leg as l on r.legID=l.legID
                        join airport as a on l.arrival=a.airportID
                        where f.flightID=ip_flightID and
f.progress=r.sequence);
      -- get the total passenger for the flight
    set total_p = (select count(t.customer)
                              from flight as f
                              join ticket as t on f.flightID=t.carrier
                              where f.flightID=ip_flightID);
      -- get the disembarked passenger using the arrival_loc for the flight
    set disembarked_p = (select count(t.customer)
                                          from flight as f
                                          join ticket as t on
f.flightID=t.carrier
                                          join person as p on
t.customer=p.personID
                                          where f.flightID=ip_flightID and
p.locationID=arrival_loc);
      if total_p != disembarked_p then
            leave sp_main;
      end if;
```

```
      -- update the locationID of the pilots for the ending flight to the
arrival airport
    update person
    set locationID=arrival_loc
    where personID in (select personID from pilot as p join flight as f on
f.support_airline=p.flying_airline and f.support_tail=p.flying_tail where
f.flightID=ip_flightID);

    -- update the flying_airline & flying_tail as null if the above guards
pass
    update pilot
    set flying_airline=null, flying_tail=null
    where personID in (select * from (select personID from pilot as p join
flight as f on f.support_airline=p.flying_airline and
f.support_tail=p.flying_tail where f.flightID=ip_flightID) as temp);


end //
delimiter ;

-- [16] retire_flight()
--
-------------------------------------------------------------------------------
/* This stored procedure removes a flight that has ended from the system.  The
flight must be on the ground, and either be at the start its route, or at the
end of its route.  */
--
-------------------------------------------------------------------------------
drop procedure if exists retire_flight;
delimiter //
create procedure retire_flight (in ip_flightID varchar(50))
sp_main: begin
      declare end_leg int default 0;
      -- ensure that the flight is on the ground
    if ip_flightID not in (select flightID from flight where
airplane_status='on_ground') then
            leave sp_main;
      end if;
    -- ensure the flight is either at the start its route or at the end of its
route
    set end_leg = (select max(r.sequence) from flight as f join route_path as
r on f.routeID=r.routeID where f.flightID=ip_flightID);
    if ip_flightID not in (select flightID from flight where progress=0 OR
progress=end_leg) then
```

```
            leave sp_main;
      end if;
    delete from flight where flightID = ip_flightID;
end //
delimiter ;

-- [17] remove_passenger_role()
--
-------------------------------------------------------------------------------
/* This stored procedure removes the passenger role from person.  The
passenger
must be on the ground at the time; and, if they are on a flight, then they
must
disembark the flight at the current airport.  If the person had both a pilot
role
and a passenger role, then the person and pilot role data should not be
affected.
If the person only had a passenger role, then all associated person data must
be
removed as well. */
--
-------------------------------------------------------------------------------
drop procedure if exists remove_passenger_role;
delimiter //
create procedure remove_passenger_role (in ip_personID varchar(50))
sp_main: begin

      -- ensure that the passenger exists
    if (ip_personID not in (select personID from passenger)) then
            leave sp_main;
    end if;

    -- ensure that the passenger is on ground or if on flight then they must
disembark at current airport
    -- according to piazza, if not in this list -> just leave the stored
procedure
    if (ip_personID in (select personID from passenger where personID in
(select personID from person where locationID like '%plane%'))) then
            leave sp_main;
    end if;

    -- remove all remaining information unless the passenger is also a pilot
    if (ip_personID not in (select personID from pilot)) then
        delete from passenger where ip_personID = personID;
```

```
        delete from person where ip_personID = personID;
    end if;
end //
delimiter ;


-- [18] remove_pilot_role()
--
-------------------------------------------------------------------------------
/* This stored procedure removes the pilot role from person.  The pilot must
not
be assigned to a flight; or, if they are assigned to a flight, then that
flight
must either be at the start or end of its route.  If the person had both a
pilot
role and a passenger role, then the person and passenger role data should not
be
affected.  If the person only had a pilot role, then all associated person
data
must be removed as well. */
--
-------------------------------------------------------------------------------
drop procedure if exists remove_pilot_role;
delimiter //
create procedure remove_pilot_role (in ip_personID varchar(50))
sp_main: begin

      -- ensure that the pilot exists
    if (ip_personID not in (select personID from pilot)) then
            leave sp_main;
    end if;

    -- ensure that the pilot is not assigned to a flight or either at start or
end of flight
    if (ip_personID in (select flying_airline from pilot) or ip_personID in
((select flying_airline from pilot where flying_tail in(select support_tail
from flight where routeID in (select routeID from route_path where
sequence=2))))) then
            leave sp_main;
    end if;

    -- remove all remaining information unless the pilot is also a passenger
    if (ip_personID not in (select personID from passenger)) then
            delete from pilot_licenses where ip_personID = personID;
```

```
        delete from pilot where ip_personID = personID;
        delete from person where ip_personID = personID;
    end if;
end //
delimiter ;


-- [19] flights_in_the_air()
--
-------------------------------------------------------------------------------
/* This view describes where flights that are currently airborne are located.
*/
--
-------------------------------------------------------------------------------
create or replace view flights_in_the_air (departing_from, arriving_at,
num_flights,
      flight_list, earliest_arrival, latest_arrival, airplane_list) as

    select l.departure, l.arrival, count(f.flightID),
group_concat(f.flightID), min(f.next_time), max(f.next_time),
group_concat(a.locationID)
      from flight as f
      join route_path as r on f.routeID=r.routeiD
      join leg as l on r.legID=l.legID
      join airplane as a on f.support_airline=a.airlineID and
f.support_tail=a.tail_num
      where airplane_status='in_flight' and f.progress=r.sequence
      group by l.departure, l.arrival;
select null, null, 0, null, null, null, null;

-- [20] flights_on_the_ground()
--
-------------------------------------------------------------------------------
/* This view describes where flights that are currently on the ground are
located. */
--
-------------------------------------------------------------------------------
create or replace view flights_on_the_ground (departing_from, num_flights,
      flight_list, earliest_arrival, latest_arrival, airplane_list) as
    select l.departure as departing_from, count(f.flightID) as num_flights,
group_concat(flightID) as flight_list, min(f.next_time) as earliest_arrival,
max(f.next_time) as latest_arrival, group_concat(a.locationID) as
airplane_list
      from flight as f
```

```
    join route_path as r on f.routeID=r.routeID
      join leg as l on r.legID=l.legID
      join airplane as a on f.support_airline=a.airlineID and
f.support_tail=a.tail_num
      where airplane_status='on_ground' and f.progress+1=r.sequence
      group by l.departure
      order by flight_list;
select null, 0, null, null, null, null;

-- [21] people_in_the_air()
--
-----------------------------------------------------------------------------
/* This view describes where people who are currently airborne are located. */
--
-----------------------------------------------------------------------------
create or replace view people_in_the_air (departing_from, arriving_at,
num_airplanes,
      airplane_list, flight_list, earliest_arrival, latest_arrival,
num_pilots,
      num_passengers, joint_pilots_passengers, person_list) as

    select l.departure as departing_from, l.arrival as arriving_at,
count(distinct a.locationID) as num_airplane, group_concat(distinct
a.locationID) as airplane_list, group_concat(distinct f.flightID) as
flight_list, min(f.next_time) as earliest_arrival, max(f.next_time) as
latest_arrival, count(pilot.personID) as num_pilots, count(passenger.personID)
as num_passengers, count(p.personID) as joint_pilots_passengers,
group_concat(p.personID) as person_list
      from flight as f
      join airplane as a on f.support_airline=a.airlineID and
f.support_tail=a.tail_num
      join person as p on p.locationID=a.locationID
      join route_path as r on f.routeID=r.routeiD
      join leg as l on r.legID=l.legID
      left join pilot on p.personID=pilot.personID
      left join passenger on p.personID=passenger.personID
      where airplane_status='in_flight' and f.progress=r.sequence
      group by l.departure, l.arrival;
select null, null, 0, null, null, null, null, 0, 0, null, null;

-- [22] people_on_the_ground()
--
-----------------------------------------------------------------------------
```

```
/* This view describes where people who are currently on the ground are
located. */
--
-------------------------------------------------------------------------
create or replace view people_on_the_ground (departing_from, airport,
airport_name,
      city, state, num_pilots, num_passengers, joint_pilots_passengers,
person_list) as

      select a.airportID as departing_from, p.locationID as airport,
a.airport_name, a.city, a.state,count(pilot.personID) as num_pilots,
count(passenger.personID) as num_passengers, count(p.personID) as
joint_pilots_passengers, group_concat(p.personID) as person_list
      from person as p
    join airport as a on p.locationID=a.locationID
      left join pilot on p.personID=pilot.personID
      left join passenger on p.personID=passenger.personID
      where p.locationID like 'port%'
      group by a.airportID, p.locationID, a.airport_name, a.city, a.state
      order by a.airportID;
select null, null, null, null, null, 0, 0, null, null;

-- [23] route_summary()
--
-------------------------------------------------------------------------
/* This view describes how the routes are being utilized by different flights.
*/
--
-------------------------------------------------------------------------
create or replace view route_summary (route, num_legs, leg_sequence,
route_length,
      num_flights, flight_list, airport_sequence) as

    select r.routeID as route, count(distinct r.legID) as num_legs,
group_concat(distinct r.legID order by sequence) as leg_sequence,
                  cast((case when count(distinct flightID)>1 then
sum(distance)/count(distinct flightID) else sum(distance) end) as decimal) as
route_length,
                  count(distinct flightID) as num_flights,
group_concat(distinct flightID) as flight_list, group_concat(distinct
concat(departure, '->', arrival) order by sequence) as airport_sequence
      from route_path as r
      join leg as l on r.legID = l.legID
      left join flight as f on r.routeID = f.routeID
```

```
      group by r.routeID, f.routeID;
select null, 0, null, 0, 0, null, null;

-- [24] alternative_airports()
--
-------------------------------------------------------------------------------
/* This view displays airports that share the same city and state. */
--
-------------------------------------------------------------------------------
create or replace view alternative_airports (city, state, num_airports,
      airport_code_list, airport_name_list) as

    select city, state, (case when count(*)>1 then count(*) else count(*) end)
as num_airports,
                   group_concat(airportID order by airportID) as
airport_code_list, group_concat(airport_name order by airportID) as
airport_name_list
      from airport
      group by city, state
      having num_airports>1
      order by city, airport_code_list;

select null, null, 0, null, null;

-- [25] simulation_cycle()
--
-------------------------------------------------------------------------------
/* This stored procedure executes the next step in the simulation cycle.  The
flight
with the smallest next time in chronological order must be identified and
selected.
If multiple flights have the same time, then flights that are landing should
be
preferred over flights that are taking off.  Similarly, flights with the
lowest
identifier in alphabetical order should also be preferred.

If an airplane is in flight and waiting to land, then the flight should be
allowed
to land, passengers allowed to disembark, and the time advanced by one hour
until
the next takeoff to allow for preparations.
```

```
If an airplane is on the ground and waiting to takeoff, then the passengers
should
be allowed to board, and the time should be advanced to represent when the
airplane
will land at its next location based on the leg distance and airplane speed.

If an airplane is on the ground and has reached the end of its route, then the
flight crew should be recycled to allow rest, and the flight itself should be
retired from the system. */
--
-------------------------------------------------------------------------------
drop procedure if exists simulation_cycle;
delimiter //
create procedure simulation_cycle ()
sp_main: begin

-- next flight
set @next_flight = (select flightID
from flight where next_time = (select min(next_time) from flight)
group by flightID
order by airplane_status = 'in_flight', flightID ASC
limit 1);

-- landing
IF EXISTS (select flightID from flight where airplane_status = 'in_flight' and
flightID = @next_flight)
then

update flight
set airplane_status = 'on_ground'
where flightID = @next_flight;

set @new_next_flight = (select flightID
from flight
where next_time > (select next_time from flight where flightID = @next_flight)
group by flightID
order by airplane_status = 'in_flight', flightID ASC
limit 1);

set @new_time = DATE_SUB((select next_time from flight where flightID =
@new_next_flight), interval 1 hour);

UPDATE flight
SET next_time = @new_time
```

```
WHERE flightID = @next_flight;

end if;

-- flight takeoff stuff
IF EXISTS (select flightID from flight where airplane_status = 'on_ground' and
flightID = @next_flight)
then

-- flight progress stuff
update flight
set airplane_status = 'in_flight'
where flightID = @next_flight;

update flight
set progress = progress + 1
where flightID = @next_flight;

set @calculated_dist = (select distance from leg l, airplane a, flight f,
route_path rp
where f.flightID = @next_flight and f.routeID = rp.routeID and f.support_tail
= a.tail_num and rp.legID = l.legID);

set @calculated_speed = (select speed from leg l, airplane a, flight f,
route_path rp
where f.flightID = @next_flight and f.routeID = rp.routeID and f.support_tail
= a.tail_num and rp.legID = l.legID);

set @calculated_time = @calculated_dist/@calculated_speed;
UPDATE flight
SET next_time = DATE_ADD(next_time, interval @calculated_time hour)
WHERE flightID = @next_flight;

end if;

-- flight ended
if exists(select f.flightID from flight f, route_path rp where
f.airplane_status = 'on_ground' and f.routeID = rp.routeID and f.flightID =
@next_flight
and f.progress = rp.sequence) then delete from flight where flightID =
@next_flight;

end if;
```

```
end //
delimiter ;
```