Q2: Create your tables and foreign keys in the database using the CREATE TABLE and if necessary the ALTER TABLE queries. Once you are done the database should have the same structure as shown in your relational model.

```
CREATE TABLE ba_person (
    Customer_id VARCHAR(255) NOT NULL,
    person_name VARCHAR(30),
    Is_passenger BOOLEAN NOT NULL,
    Passport_no INT,
    PRIMARY KEY (Customer_id)
)  ENGINE=INNODB;

CREATE TABLE ba_airport (
    Airport_code VARCHAR(3),
    Airport_name  VARCHAR(30),
    Country VARCHAR(30),
            PRIMARY KEY (Airport_code)
) ENGINE=INNODB;

CREATE TABLE ba_flight_route (
  Airport_departs VARCHAR(3),
  Airport_arrives  VARCHAR(3),
  PRIMARY KEY (Airport_departs,Airport_arrives),
  FOREIGN KEY (Airport_departs) REFERENCES ba_airport (Airport_code),
  FOREIGN KEY (Airport_arrives) REFERENCES ba_airport (Airport_code)
) ENGINE=INNODB;

CREATE TABLE ba_weekly_schedule (
  Schedule_code VARCHAR(255),
  Flight_airport_departs VARCHAR(3),
  Flight_airport_arrives VARCHAR(3),
  Departure_time TIME,
  Weekday_day VARCHAR(10),
  Weekday_year INT,
  PRIMARY KEY (Schedule_code),
  FOREIGN KEY (Flight_airport_departs) REFERENCES ba_flight_route (Airport_departs),
  FOREIGN KEY (Flight_airport_arrives) REFERENCES ba_flight_route (Airport_arrives)
)  ENGINE=INNODB;

CREATE TABLE ba_year (
    Y_year INT,
    Year_factor DOUBLE,
          PRIMARY KEY (Y_year)
    )  ENGINE = INNODB;
```

```
CREATE TABLE ba_week_day (
   Year_year INT,
   Day_week VARCHAR(10),
   Weekday_factor DOUBLE,
   PRIMARY KEY (Year_year,Day_week),
   FOREIGN KEY (Year_year) REFERENCES ba_year (Y_year)
   )  ENGINE=INNODB;

CREATE TABLE ba_price_table(
   Airport_departs VARCHAR(3),
   Airport_arrives VARCHAR(3),
   Year_year INT,
   Base_price DOUBLE,
   FOREIGN KEY (Airport_departs) REFERENCES ba_flight_route (Airport_departs),
         FOREIGN KEY (Airport_arrives) REFERENCES ba_flight_route (Airport_arrives),
         FOREIGN KEY (Year_year) REFERENCES ba_year (Y_year)
         )   ENGINE = INNODB;

 CREATE TABLE ba_flight(
   Flight_number VARCHAR(255),
   Weekly_schedule_code VARCHAR(255),
   Week_number INT,
   PRIMARY KEY (Flight_number),
   FOREIGN KEY (Weekly_schedule_code) REFERENCES ba_weekly_schedule
(Schedule_code)
   )   ENGINE = INNODB;

 CREATE TABLE ba_reservation(
   Reservation_number VARCHAR(255),
   Is_booking BOOLEAN,
   Flight_number VARCHAR(255),
   PRIMARY KEY (Reservation_number),
   FOREIGN KEY (Flight_number) REFERENCES ba_flight (Flight_number)
   )   ENGINE = INNODB;


CREATE TABLE ba_travels_in(
    Reservation_number VARCHAR(255),
    Passenger_customerID VARCHAR(255),
    PRIMARY KEY (Reservation_number, Passenger_customerID),
    FOREIGN KEY (Reservation_number) REFERENCES ba_reservation
(Reservation_number),
    FOREIGN KEY (Passenger_customerID) REFERENCES ba_person (Customer_id)
    )   ENGINE = INNODB;


CREATE TABLE ba_main_contact(
   Reservation_number VARCHAR(255),
```

```
    Passenger_customerID VARCHAR(255),
    Email VARCHAR(30),
    Phone_number BIGINT,
    PRIMARY KEY (Reservation_number, Passenger_customerID),
    FOREIGN KEY (Reservation_number) REFERENCES ba_reservation
(Reservation_number),
    FOREIGN KEY (Passenger_customerID) REFERENCES ba_person (Customer_id)
    )   ENGINE = INNODB;

CREATE TABLE ba_provides(
    Reservation_number VARCHAR(255),
    CustomerID VARCHAR(255),
    Ticket_number VARCHAR(255),
    PRIMARY KEY (Reservation_number, CustomerID),
    FOREIGN KEY (Reservation_number) REFERENCES ba_reservation
(Reservation_number),
    FOREIGN KEY (CustomerID) REFERENCES ba_person (Customer_id)
    )   ENGINE = INNODB;

CREATE TABLE ba_is_paid_by(
    Reservation_number VARCHAR(255),
    CustomerID VARCHAR(255),
    Final_price DOUBLE,
    Credit_cardNO BIGINT,
    PRIMARY KEY (Reservation_number, CustomerID),
    FOREIGN KEY (Reservation_number) REFERENCES ba_reservation
(Reservation_number),
    FOREIGN KEY (CustomerID) REFERENCES ba_person (Customer_id)
    )   ENGINE = INNODB;
```

Add a unique Key constraint.

```
ALTER TABLE ba_provides ADD CONSTRAINT uk_ba_provides UNIQUE (Ticket_number);
```

Q3. Write procedures for filling the database with flights, etc. These procedures will work as an interface with the front-end.

A.)

```
CREATE PROCEDURE addYear (IN year INT, IN factor DOUBLE)
BEGIN
  INSERT INTO ba_year (Y_year, Year_factor)
  VALUES (year, factor)
  ON DUPLICATE KEY UPDATE Year_factor = factor;
END
```

B.)

```
CREATE PROCEDURE addDay (IN year INT, IN day VARCHAR(10), IN factor DOUBLE)
BEGIN
```

```
  INSERT INTO ba_week_day (Year_year, Day_week, Weekday_factor)
  VALUES (year, day, factor)
  ON DUPLICATE KEY UPDATE Weekday_factor = factor;
END

C.)CREATE PROCEDURE addDestination (IN airport_code VARCHAR(3), IN name
VARCHAR(30), IN country VARCHAR(30))
BEGIN
  INSERT INTO ba_airport (Airport_code, Airport_name, Country)
  VALUES (airport_code, name, country)
  ON DUPLICATE KEY UPDATE Airport_name = name, Country = country;
END

D.)CREATE PROCEDURE addRoute (IN departure_airport_code VARCHAR(3),
                      IN arrival_airport_code VARCHAR(3),
                      IN year INT,
                      IN routeprice DOUBLE
                      )
BEGIN
  INSERT IGNORE INTO ba_flight_route (Airport_departs, Airport_arrives)
  VALUES (departure_airport_code, arrival_airport_code);

  INSERT INTO ba_price_table (Airport_departs, Airport_arrives, Year_year, Base_price)
  VALUES (departure_airport_code, arrival_airport_code, year, routeprice)
  ON DUPLICATE KEY UPDATE Base_price = routeprice;
END

E.) CREATE PROCEDURE addFlight (IN departure_airport_code VARCHAR(3),
                       IN arrival_airport_code VARCHAR(3),
                       IN year INT,
                       IN day VARCHAR(10),
                       IN departure_time TIME
                      )
BEGIN
  DECLARE schedule_code VARCHAR(255);
  DECLARE flight_number VARCHAR(255);
  SET schedule_code = UUID();
  SET flight_number = UUID();
  INSERT INTO ba_weekly_schedule (Schedule_code, Flight_airport_departs,
Flight_airport_arrives,  Departure_time, Weekday_day, Weekday_year)
  VALUES (schedule_code, departure_airport_code, arrival_airport_code,
DATE_FORMAT(departure_time, '%H:%i:%s') , day, year);

  INSERT INTO ba_flight(Flight_number, Weekly_schedule_code, Week_number)
  VALUES (flight_number, schedule_code, DATE_FORMAT(departure_time, '%v'));
  END
```

Q4. Write two help-functions that do some of the calculations necessary for the booking procedure

```
SET GLOBAL log_bin_trust_function_creators = 1;
DROP FUNCTION if exists calculateFreeSeats;
DROP FUNCTION if exists calculatePrice;
DELIMITER //

CREATE FUNCTION calculateFreeSeats(flightnumber varchar(255))
     RETURNS int
BEGIN
      DECLARE paid_seats INT;
      DECLARE airplane_capacity INT;
      set airplane_capacity = 40;
      SELECT COUNT(*) into paid_seats FROM ba_provides
      WHERE Reservation_number in (
      SELECT Reservation_number FROM ba_reservation
      WHERE Flight_number=flightnumber
      );
      RETURN(airplane_capacity-paid_seats);
END //

CREATE FUNCTION calculatePrice(flightnumber varchar(255))
     RETURNS int
BEGIN
      DECLARE route_price DOUBLE;
      DECLARE weekdayfactor DOUBLE;
      DECLARE occupied_seats INT;
      DECLARE profitfactor DOUBLE;

      # Base route
      SELECT p.Base_price into route_price FROM ba_price_table p
                  JOIN ba_weekly_schedule w
                        ON  w.Flight_airport_departs = p.Airport_departs
                        AND w.Flight_airport_arrives = p.Airport_Arrives
                  INNER JOIN ba_flight f
                        ON w.Schedule_code = f.Weekly_schedule_code
                  WHERE f.Flight_number = '17b7129c-7573-11ec-8e13-0c9d927ef5fb'
                  AND w.Weekday_year = p.Year_year;

      -- weekdayfactor --
      SELECT ba_week_day.Weekday_factor into weekdayfactor FROM ba_week_day
      INNER JOIN  ba_weekly_schedule
                        ON ba_week_day.Year_year = ba_weekly_schedule.Weekday_year
                              AND ba_week_day.Day_week =
ba_weekly_schedule.Weekday_day
      INNER JOIN ba_flight
                        ON ba_weekly_schedule.Schedule_code =
ba_flight.Weekly_schedule_code
```

```
                WHERE ba_flight.Flight_number = flightnumber;

        -- free seats --
        SELECT 40 - calculateFreeSeats(flightnumber) INTO occupied_seats;

        -- profit factor --
                SELECT ba_year.Year_factor INTO profitfactor FROM ba_year
        INNER JOIN  ba_weekly_schedule
                        ON ba_year.Y_year = ba_weekly_schedule.Weekday_year
        INNER JOIN ba_flight
                        ON ba_weekly_schedule.Schedule_code =
ba_flight.Weekly_schedule_code
                WHERE ba_flight.Flight_number = flightnumber;

        RETURN(route_price*weekdayfactor * ((occupied_seats + 1)/40) * profitfactor);
END //
```

Q5. Create a trigger that issues unique unguessable ticket-numbers (of type integer) for each passenger on a reservation once it is paid. An appropriate MySQL function to find unguessable numbers is rand().

```
CREATE TRIGGER issue_tickets AFTER UPDATE ON ba_reservation
FOR EACH ROW
BEGIN
        IF (NEW.Is_booking = TRUE) AND (OLD.Is_booking = FALSE) THEN
        INSERT INTO ba_provides (Reservation_number, CustomerID, Ticket_number)
                SELECT ba_travels_in.Reservation_number,
                        ba_travels_in.Passenger_customerID,
            FLOOR(RAND()*(1000000+1)) FROM ba_travels_in
        WHERE ba_travels_in.Reservation_number = NEW.Reservation_number;
    END IF;
END;
```

Q6. It is now time to write the stored procedures necessary for creating and handling a reservation from the front-end. In addition to the input and output detailed below, see the test-files for appropriate error-messages to return in case of unsuccessful payments etc.

```
CREATE PROCEDURE addReservation (IN dep_airport_code VARCHAR(3), IN
arr_airport_code VARCHAR(3), IN year INT, IN week INT, IN day VARCHAR(10), IN
dep_time TIME, IN number_of_passengers INT, OUT output_reservation_nr INT)
BEGIN
        SET @weekschedulecode = NULL;
    SET @flightno = NULL;
    SET @resno_exists = 1;
```

```
   SELECT Schedule_code INTO @weekschedulecode FROM ba_weekly_schedule
WHERE Flight_airport_departs = dep_airport_code
                                    AND Flight_airport_arrives = arr_airport_code
                                    AND Departure_time = dep_time
                                    AND Weekday_day = day
                                    AND Weekday_year = year;
        SELECT Flight_number INTO @flightno FROM ba_flight WHERE
Weekly_schedule_code = @weekschedulecode;

   -- START TRANSACTION;

   -- Raise error if flight does not exist
   IF @flightno IS NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'There exist no flight
for the given route, date and time';
   END IF;

   -- Raise error if there are no seats on the flight
        IF number_of_passengers > calculateFreeSeats(@flightno) THEN
                SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'There are not enough
seats available on the chosen flight';
        END IF;

        -- Generate a unique reservation number and update
        WHILE @resno_exists IS NOT NULL DO
                SELECT FLOOR(100+RAND()*999) * FLOOR(10+RAND()*99) *
FLOOR(1+RAND()*9) INTO output_reservation_nr;
                SELECT 1 INTO @resno_exists FROM ba_reservation WHERE
Reservation_number = output_reservation_nr;

        SET @resno_exists = NULL;
                        INSERT INTO ba_reservation VALUES (output_reservation_nr,
FALSE, @flightno);
        END WHILE;
END;
//
DELIMITER ;


DELIMITER //
CREATE PROCEDURE addPassenger (IN reservation_nr INT, IN passport_number INT, IN
name VARCHAR(30))
BEGIN
   #SET @custid = NULL;
   SET @resno = NULL;
   SET @bookstatus = NULL;
   SET @passengerID = NULL;
```

```
        -- Raise error if reservation number does not exist
    SELECT Reservation_number, Is_booking  INTO @resno, @bookstatus FROM
ba_reservation WHERE Reservation_number = reservation_nr;
    IF @resno IS NULL THEN
                SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'The given reservation
number does not exist';
    END IF;

        -- Raise error if reservation status cannot be changed
    IF @bookstatus THEN
                SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'The booking has
already been payed and no futher passengers can be added';
    END IF;

    -- Update passenger,reservation details and ticket number
    SELECT UUID() INTO @passengerID;
        INSERT INTO ba_person
    VALUES (@passengerID, name, TRUE, passport_number)
    ON DUPLICATE KEY UPDATE Is_passenger = TRUE, Passport_no = passport_number;

    INSERT INTO ba_travels_in
    VALUES(reservation_nr, @passengerID);
    #SELECT Customer_id INTO @custid FROM ba_person WHERE Passport_no =
passport_number;
        #INSERT INTO ba_travels_in VALUES (reservation_nr, @custid);
END;
//
DELIMITER ;


DELIMITER //
CREATE PROCEDURE addContact (IN reservation_nr INT, IN passport_number INT,
                                                    IN email VARCHAR(30), IN phone
BIGINT)
BEGIN
    SET @resno_exists = NULL;
    SET @pass_exists = NULL;
        SET @custid = NULL;

        -- Raise error if reservation number does not exist
    SELECT 1 INTO @resno_exists FROM ba_reservation WHERE Reservation_number =
reservation_nr;
    IF @resno_exists IS NULL THEN
                SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'The given reservation
number does not exist';
    END IF;
        -- Raise error if contact is not a passenger of the reservation
```

```
   SELECT Customer_id INTO @custid FROM ba_person WHERE Passport_no =
passport_number LIMIT 1;
   #SELECT 1 INTO @pass_exists FROM ba_travels_in WHERE  Passenger_customerID =
@custid ;
   IF @custid IS NULL THEN
                  SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'The person is not a
passenger of the reservation';
   END IF;

   -- Update passenger contact information
   INSERT INTO ba_main_contact (Reservation_number, Passenger_customerID, Email,
Phone_number)
         VALUES (reservation_nr, @custid, email, phone)
   ON duplicate key update Email=email, Phone_number=phone;
         #UPDATE ba_main_contact
   #SET Email = email,
         #        Phone_number = phone
         #WHERE  Passenger_customerID = @custid  ;
END;
//

DELIMITER ;


DELIMITER //

CREATE PROCEDURE addPayment (IN reservation_nr INT, IN cardholder_name
VARCHAR(30), IN credit_card_number BIGINT)
BEGIN
   SET @has_contact = NULL;
         SET @free_seats = NULL;
   SET @required_seats = NULL;
   SET @payer = NULL;
   SET @flight_number = NULL;
   SET @already_paid = NULL;
   SET @reservation_exists = NULL;

   SELECT COUNT(*) INTO @reservation_exists FROM ba_reservation  WHERE
Reservation_number = reservation_nr;
   IF @reservation_exists < 1 THEN
                  SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'The given reservation
number does not exist';
   END IF;

   # Raise error if reservation is already a booking
   SELECT Is_booking INTO @already_paid FROM ba_reservation WHERE
Reservation_number = reservation_nr LIMIT 1;
   IF @already_paid THEN
```

```
                signal SQLSTATE '45005' SET MESSAGE_TEXT = "Reservation is already
paid for";
   END IF;

        -- Raise error if there's no contact
   SELECT 1 INTO @has_contact FROM ba_main_contact WHERE Reservation_number =
reservation_nr LIMIT 1;
   IF @has_contact IS NULL THEN
                SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'The given reservation
does not have a main contact';
   END IF;

        -- Raise error if there aren´t sufficient seats
   SELECT count(*) INTO @required_seats from ba_travels_in WHERE
Reservation_number = reservation_nr;
   SELECT ba_reservation.Flight_number INTO @flight_number from ba_reservation
WHERE Reservation_number = reservation_nr LIMIT 1;
   SELECT calculateFreeSeats(@flight_number)INTO @free_seats LIMIT 1;

   IF (@free_seats - @required_seats) < 0 THEN
                DELETE FROM ba_main_contact WHERE Reservation_number =
reservation_nr;
                DELETE FROM ba_travels_in WHERE Reservation_number =
reservation_nr;
                DELETE FROM ba_reservation WHERE Reservation_number =
reservation_nr;
                SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Not enough seats
available. Removing reservation.';
   END IF;

   # Get payer customer ID from name
   SELECT Customer_id into @payer from ba_person WHERE person_name =
cardholder_name LIMIT 1;
   IF @payer IS NULL THEN
                SELECT uuid() into @payer;
                INSERT INTO ba_person
                VALUES (@payer, cardholder_name, FALSE, NULL);
   END IF;

   # Add payment
   INSERT INTO ba_is_paid_by (Reservation_number, CustomerID, Final_price,
Credit_cardNO)
   VALUES (reservation_nr, @payer, (@required_seats * calculatePrice(@flight_number)),
credit_card_number);

   # Update reservation into booking
        UPDATE ba_reservation
   SET Is_booking = TRUE
```

```
    WHERE  Reservation_number = reservation_nr;

END;
```

Q7. Create a view allFlights containing all flights in your database with the following information: departure_city_name, destination_city_name, departure_time,departure_day, departure_week, departure_year, nr_of_free_seats, current_price_per_seat. See the test code for an example of how it can look like

```
CREATE VIEW allFlights
AS SELECT w.Flight_airport_departs, w.Flight_airport_arrives, w.Weekday_day,
f.Week_number, w.Weekday_year, calculateFreeSeats(f.Flight_number)
FROM ba_flight f LEFT JOIN ba_weekly_schedule w
ON w.Schedule_code = f.Weekly_schedule_code;
```

Q8: Answer the following theoretical questions:

a) How can you protect the credit card information in the database from hackers?

ANSWER: To protect sensitive data, mySQL provides encryption, key generation, digital signatures and other cryptographic features. Credit card information can thus be stored in an encrypted format. When the card number is entered, it should convert the number into a unique alphanumeric encoded id along with generating a key which can revert back this encoded id into the credit card number. Access to the key should be restricted to an individual or a team which closely works on database security. To enable encryption, for a new file-per-table tablespace, we can specify the ENCRYPTION option in a CREATE TABLE statement.

b) Give three advantages of using stored procedures in the database (and thereby execute them on the server) instead of writing the same functions in the front-end of the system (in for example java-script on a web-page)?

ANSWER: It reduces the network traffic between application and mysql server . It reduce data transfer and communication cost(assuming a client-server setting). If there is any computation that requires different columns of data, instead of extracting those columns and performing computations on application level, stored procedures will allow users to extract only the after computed data.

- Stored procedures reduce duplication of effort if a database program is needed by several applications which implies, if various applications require similar computation on data, triggers and stored procedures helps in avoiding different and duplicate methods to perform similar computations and unify them with a single program.

Q9: Open two MySQL sessions. We call one of them A and the other one B. Write START TRANSACTION; in both terminals

a) In session A, add a new reservation.

Opening 2 terminals at same time and after "Start Transaction". I add a new reservation in first terminal which added a reservation successfully but on the other terminal no reservation was added when check using "select * from Reservation;".This is because the reservation is not added in the database but just stored in that terminal and not committed.

b) Is this reservation visible in session B? Why? Why not?

A: The reservation added in session A was visible in session B. This is because START TRANSACTION is implemented outside the stored procedure to add reservation. The statements in the stored procedure are implicitly committed to the database. If START TRANSACTION is implemented inside the stored procedure, then the reservation added in session A is not visible in session B. This is because the changes of adding the reservation in session A are not committed to the database and hence not visible in session B.

c) What happens if you try to modify the reservation from A in B?
Explain what happens and why this happens and how this relates to the concept of isolation of transactions.

A: The reservation added in session A was not visible in session B before COMMIT. Modifying the reservation resulted in "0 row(s) affected" as there was no record found in session B for the reservation created in session A. This relates to the concept that a transaction is isolated until it is committed to the database.The effects of an incomplete/active transaction in one session cannot be seen in a concurrent session.

Q10: Is your BryanAir implementation safe when handling multiple concurrent transactions?

a) Did overbooking occur when the scripts were executed? If so, why? If not, why not?

ANSWER: No, overbookings did not occur in our database when the scripts were executed concurrently. This is probably because one transaction updates the tables fractionally faster than the other one and hence only the first reservation gets confirmed.

b) Can an overbooking theoretically occur? If an overbooking is possible, in what order must the lines of code in your procedures/functions be executed.

ANSWER: Yes, theoretically overbooking can occur if during the payment of one reservation, another reservation on The same flight also accesses the related tables simultaneously. In such a case, the updates of the first reservation confirmation will not be reflected in time for the free seat-availability check during the second reservation confirmation and hence both reservations will get confirmed while in fact there are not enough
seats available for all confirmed passengers.

c) Try to make the theoretical case occur in reality by simulating that multiple sessions call the procedure at the same time.

ANSWER:It was not possible to make the theoretical case occur due to the fact that it was not possible to start or perform the same operations at the exact same time in both the terminals.

d) Modify the test scripts so that overbookings are no longer possible using (some of) the commands START TRANSACTION, COMMIT, LOCK TABLES, UNLOCK TABLES, ROLLBACK, SAVEPOINT, and SELECT...FOR UPDATE.

ANSWER:So it will first make payment and then the second payment procedure can be called after unlocking.This can be done by using LOCK TABLES and UNLOCK TABLES queries in the beginning and end.