Brian Air

TDDD37 Database Technology - theoretical questions

Simon Jakobsson (simja649) and Gustav Hanstorp (gusha433)

23 december 2020

1 Introduction

This is the hand-in for the project in TDDD37 made by Simon Jakobsosn and Gustav Hanstorp. The first sections will be answering the questions, followed by the EER-diagram and schema, finally this will be followed by the code itself.

2 Question 8

a) We put all values through a hash. This way the credit-card number is encoded, and not even people with access to the db can read it.

b)
i. It's more secure, because hackers would have an easier time finding vulnerabilities in the front-end. It is harder to get access to the servers structure.

ii. You're not affected by the different languages in front- and back-end, which in turn makes easier to change, test and apply new code.

iii.It's faster, instead of sending a lot of data back and forth to handle different statements you only do it once.

3 Question 9

a) In session A, add a new reservation.
Done

b) Is this reservation visible in session B? Why? Why not?
No it is not, this is because of the isolation between different transactions principle

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 databases.

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
We guess the reason behind this is that A has a lock. Our other guess would have been that B would have updated A but A couldn't see it. After everything would have been committed both A and B would have seen it.

This relates to the concept of isolation because a transaction should not interfere with another, thus be invisible to each other.

4 Question 10

a) No we did not get an overbooking. This is the cause of our payment method that has 3 if statements to be able too book, if the reservation exists, if there is a contact and if there enough seats. This will not approve two bookings resulting in a overbooking and delete the other reservation(s).

b)

Yes it is possible, if there are no locks on the payment method and if they were to execute at nearly the exact time, bypassing our if statements there could be an overbooking.
c)
We put a sleep argument in the code, before the critical part, in the addPayment function and we came to a conclusion that a overbooking did occur.
d)
We add the following to the testscript:
LOCK TABLES RESERVED WRITE, RESERVATION WRITE, CONTACT WRITE, CREDENTIALS WRITE, BOOKING WRITE,
YEAR READ, WEEKLYSCHEDULE READ, WEEKDAY READ, ROUTE READ;
CALL addPayment(...)
UNLOCK ALL;

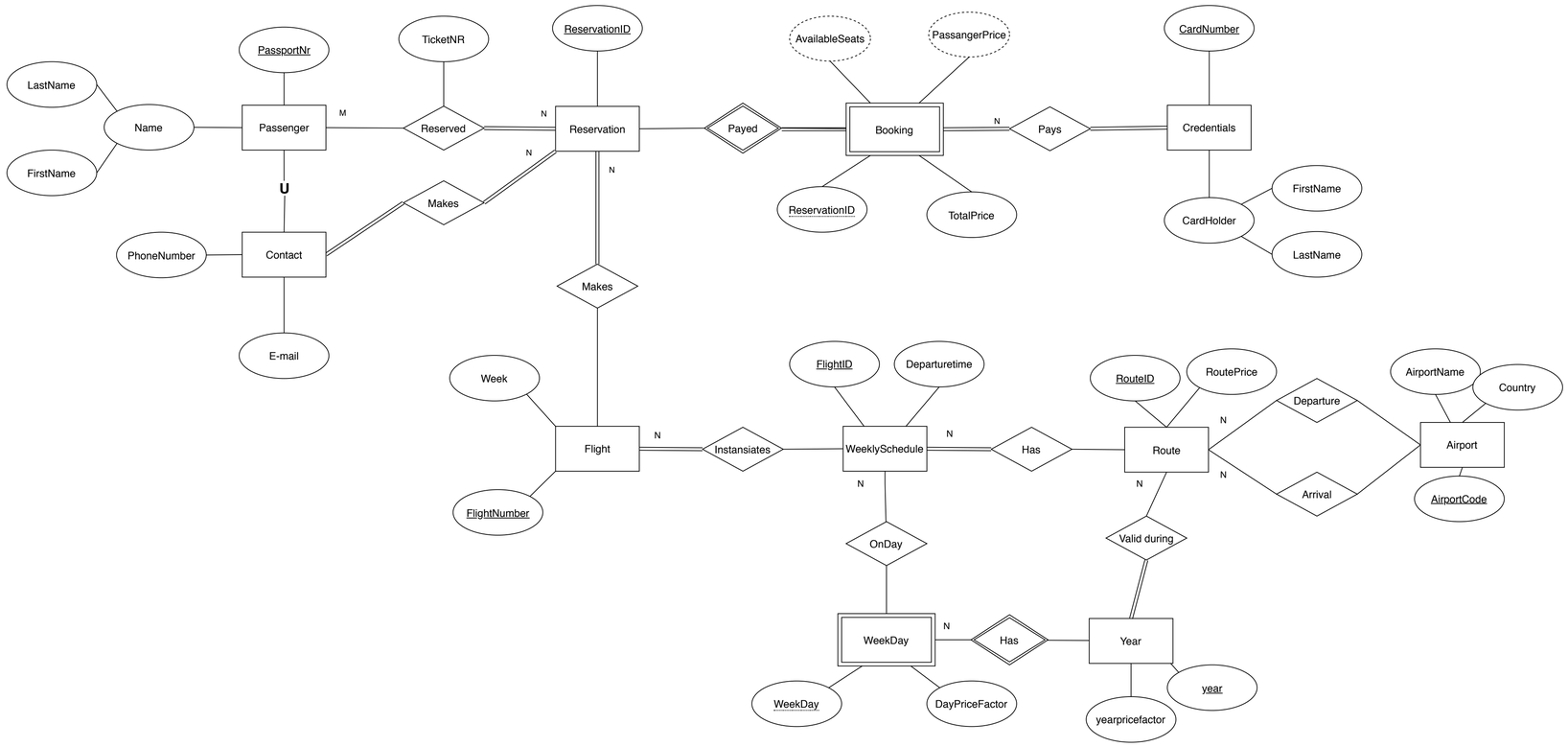
This solves the overbooking issue because we create locks on the relevant tables; write if we need to change the table and read if we are just acquiring data from it. This makes it so that only one session at the time can make changes in the relevant tables for payment, and thus, even if it's a sleep, they won't execute at the same time and no overbooking can occur. In our case it will say that there are not enough seats for one of the reservations and instead delete it.

5 Secondary Index

For a secondary index to be effective it would need to be a case where it uses a binary search, while the normal case would use a linear search.
For instance this could be finding which contact is responsible for a which reservation(s) (secondary index based on contact), or finding which ticket number belongs to which passenger(secondary index based on ticket number), etc.

An example:
Looking at the RESERVATION table we have 4 columns, each an INT on 4 bytes, which means 16 bytes in total for each row. Let's say that (the convenient number of) 192 reservations are made every day over the span of 7 years (a year is 365 days).
This mean that we will have a total of 490560 reservations. Assuming that the block size is 512 we get a blocking factor of 32 (512/16).
The amount of blocks needed to store our files are, assuming that there is no wasted space, 15330 (490560/32).
This mean that if we were to use a linear search we would get 7665 (15330/2) cases on average.
Having a secondary index based on contact, we would get the same amount of records, namely 490560 (as we have a non-ordered non-key dense index). Although we would only have 8 bytes of data in each row of the index, 4 for the contact ID and 4 for the pointer.
This means that we would get a blocking factor of 64 (512/8), and assuming there is no wasted space we can see that 7665 (490560/64) blocks are needed to store our data.
Using a binary search we would get 13 (LOG2(7665), rounded up) cases as a worst case.
As we can see, this is a lot faster and easier than not having a secondary index, especially when the amount of data stored increases.
To implement mentioned design we would simply add "KEY 'search_by_contact' ('CONTACT')" after the primary key when we create the CONTACT table.

Assignment #:	2
Description:	EER Model
Students:	Simon Jakobsson Gustav Hanstorp
Gitlab URL:	https://gitlab.liu.se/simja649/db



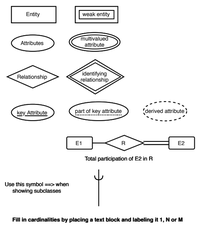
Legend

1. All relations with no indication of number is assumed to be 0 or 1

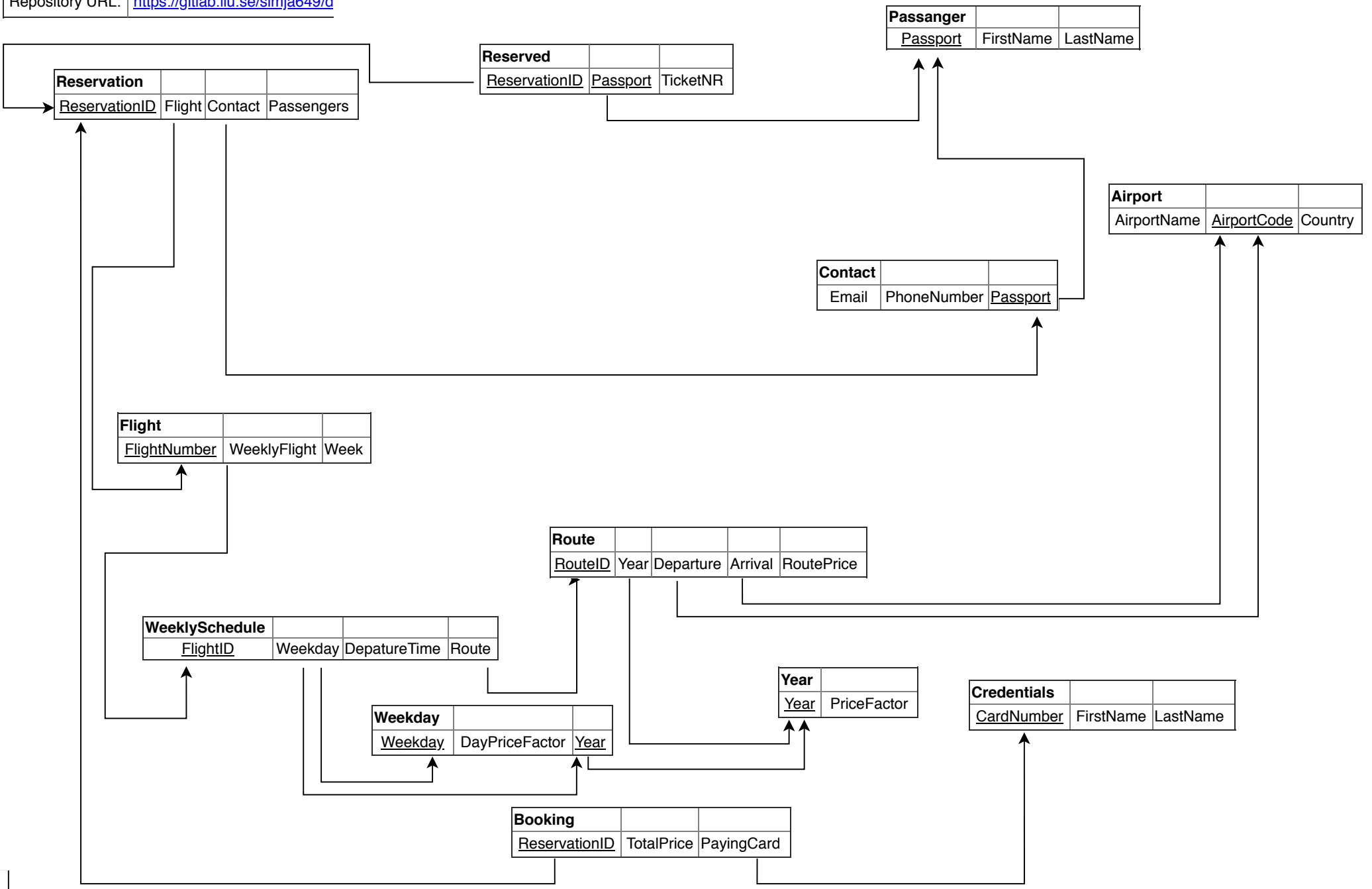
Assumptions:
1. A passanger may not be booked for more than one seat for each flight.
2. A reservation must have strictly one contact and a contact must have at least one reservation.
3. A cardholder may pay any number of bookings.
4. A Flight must have a route, while a route can have any number of flights.
5. A flight must have one airplane, while a airplane can make any number of flights.
6. An airport can have any number of routes through it, and a route can have any number of airports (stopovers).

Note:

1. This template contains two pages (EER Model, Relational Model, see the bottom of the editor window).
2. Fit each diagram on one page, preserve the page size.
3. Fill out the header at the top left corner on both pages.
4. Keep the diagrams clean, i.e. shapes and lines should not overlap, maintain optimal distance between the objects, etc.
5. Save a working copy as an XML .drawio file to be able to correct the diagrams in the future.
6. To submit the diagrams, they must be exported and uploaded to Gitlab as a PDF file (File->Export as->PDF->All pages).



Assignment #:	2
Description:	Relational Model
Students:	Simon Jakobsson Gustav Hanstorp
Repository URL:	https://gitlab.liu.se/simja649/d



```
################################################################
#Drop all tables, functions, unlock all tables if crash etc.
################################################################
SET FOREIGN_KEY_CHECKS = 0;
UNLOCK TABLES;
DROP TABLE IF EXISTS PASSENGER;
DROP TABLE IF EXISTS CONTACT;
DROP TABLE IF EXISTS RESERVATION;
DROP TABLE IF EXISTS RESERVED;
DROP TABLE IF EXISTS FLIGHT;
DROP TABLE IF EXISTS WEEKLYSCHEDULE;
DROP TABLE IF EXISTS WEEKDAY;
DROP TABLE IF EXISTS BOOKING;
DROP TABLE IF EXISTS YEAR;
DROP TABLE IF EXISTS CREDENTIALS;
DROP TABLE IF EXISTS ROUTE;
DROP TABLE IF EXISTS AIRPORT;

DROP PROCEDURE IF EXISTS addYear;
DROP PROCEDURE IF EXISTS addDay;
DROP PROCEDURE IF EXISTS addDestination;
DROP PROCEDURE IF EXISTS addRoute;
DROP PROCEDURE IF EXISTS addFlight;
DROP PROCEDURE IF EXISTS addReservation;
DROP PROCEDURE IF EXISTS addPassenger;
DROP PROCEDURE IF EXISTS addContact;
DROP PROCEDURE IF EXISTS addPayment;

DROP FUNCTION IF EXISTS calculateFreeSeats;
DROP FUNCTION IF EXISTS calculatePrice;

DROP VIEW IF EXISTS allFlights;
SET FOREIGN_KEY_CHECKS = 1;

##################
#Create all tables
##################

CREATE TABLE PASSENGER(
      PASSPORTNR INTEGER NOT NULL,
    NAME VARCHAR(30),
    CONSTRAINT PK_PASSENGER PRIMARY KEY(PASSPORTNR)
    );

CREATE TABLE CONTACT(
      PASSPORTNR INTEGER NOT NULL,
      EMAIL VARCHAR(30),
    PHONENUMBER BIGINT,
    CONSTRAINT PK_CONTACT PRIMARY KEY(PASSPORTNR)
    );

CREATE TABLE RESERVATION(
      RESERVATIONID INTEGER NOT NULL AUTO_INCREMENT,
    FLIGHT INTEGER,
    CONTACT INTEGER,
    PASSENGERS INTEGER,
    CONSTRAINT PK_RESERVATION PRIMARY KEY(RESERVATIONID)
    );
```

```
CREATE TABLE RESERVED(
      PASSPORTNR INTEGER NOT NULL,
    RESERVATIONID INTEGER NOT NULL,
    TICKETNR INTEGER,
    PRIMARY KEY(PASSPORTNR, RESERVATIONID)
      );

CREATE TABLE FLIGHT (
      FLIGHTNUMBER INTEGER NOT NULL AUTO_INCREMENT,
    WEEKLYFLIGHT INTEGER NOT NULL,
    WEEK INTEGER,
    CONSTRAINT PK_FLIGHT PRIMARY KEY(FLIGHTNUMBER)
    );

CREATE TABLE WEEKLYSCHEDULE(
      FLIGHTID INTEGER NOT NULL AUTO_INCREMENT,
    WEEKDAY VARCHAR(10) NOT NULL,
    YEAR INTEGER NOT NULL,
    DEPARTURETIME TIME,
    ROUTE INTEGER NOT NULL,
    CONSTRAINT PK_WEEKLYSCHEDULE PRIMARY KEY(FLIGHTID)
    );

CREATE TABLE WEEKDAY(
      WEEKDAY VARCHAR(10) NOT NULL,
    DAYPRICEFACTOR DOUBLE NOT NULL,
    YEAR INTEGER NOT NULL,
    CONSTRAINT PK_WEEKDAY PRIMARY KEY(WEEKDAY,YEAR)
    );

CREATE TABLE BOOKING(
      RESERVATIONID INTEGER NOT NULL,
    TOTALPRICE DOUBLE,
    PAYINGCARD BIGINT NOT NULL,
    CONSTRAINT PK_BOOKING PRIMARY KEY(RESERVATIONID)
    );

CREATE TABLE YEAR(
      YEAR INTEGER NOT NULL,
    PRICEFACTOR DOUBLE,
    CONSTRAINT PK_YEAR PRIMARY KEY(YEAR));

CREATE TABLE CREDENTIALS(
      CARDNUMBER BIGINT NOT NULL,
    NAME VARCHAR(30),
    CONSTRAINT PK_CREDENTIALS PRIMARY KEY(CARDNUMBER));

CREATE TABLE ROUTE(
      ROUTEID INTEGER NOT NULL AUTO_INCREMENT,
    DEPARTURE VARCHAR(3) NOT NULL,
    ARRIVAL VARCHAR(3) NOT NULL,
    ROUTEPRICE DOUBLE,
    YEAR INTEGER NOT NULL,
    CONSTRAINT PK_ROUTE PRIMARY KEY(ROUTEID)
    );

CREATE TABLE AIRPORT(
      AIRPORTCODE VARCHAR(3) NOT NULL,
    AIRPORTNAME VARCHAR(30) NOT NULL,
```

```
    COUNTRY VARCHAR(30),
    CONSTRAINT PK_AIRPORT PRIMARY KEY(AIRPORTCODE));

##############
#FOREIGN KEYS
##############
ALTER TABLE ROUTE ADD CONSTRAINT FK_ROUTE_DEPARTURE FOREIGN KEY(DEPARTURE)
REFERENCES AIRPORT(AIRPORTCODE);
ALTER TABLE ROUTE ADD CONSTRAINT FK_ROUTE_ARRIVAL FOREIGN KEY(ARRIVAL) REFERENCES
AIRPORT(AIRPORTCODE);
ALTER TABLE ROUTE ADD CONSTRAINT FK_ROUTE_YEAR FOREIGN KEY(YEAR) REFERENCES
YEAR(YEAR);

ALTER TABLE CONTACT ADD CONSTRAINT FK_CONTACT FOREIGN KEY(PASSPORTNR) REFERENCES
PASSENGER(PASSPORTNR);

ALTER TABLE RESERVATION ADD CONSTRAINT FK_RESERVATION_FLIGHT FOREIGN KEY(FLIGHT)
REFERENCES FLIGHT(FLIGHTNUMBER);
ALTER TABLE RESERVATION ADD CONSTRAINT FK_RESERVATION_CONTACT FOREIGN KEY(CONTACT)
REFERENCES CONTACT(PASSPORTNR);

ALTER TABLE RESERVED ADD CONSTRAINT FK_RESERVED_PASSPORT FOREIGN KEY (PASSPORTNR)
REFERENCES PASSENGER(PASSPORTNR);
ALTER TABLE RESERVED ADD CONSTRAINT FK_RESERVED_RESERVATION FOREIGN KEY
(RESERVATIONID) REFERENCES RESERVATION(RESERVATIONID);

ALTER TABLE FLIGHT ADD CONSTRAINT FK_FLIGHT FOREIGN KEY(WEEKLYFLIGHT) REFERENCES
WEEKLYSCHEDULE(FLIGHTID);

ALTER TABLE WEEKLYSCHEDULE ADD CONSTRAINT FK_WEEKLYSCHEDULE_WEEKDAY FOREIGN
KEY(WEEKDAY) REFERENCES WEEKDAY(WEEKDAY);
ALTER TABLE WEEKLYSCHEDULE ADD CONSTRAINT FK_WEEKLYSCHEDULE_YEAR FOREIGN KEY(YEAR)
REFERENCES YEAR(YEAR);
ALTER TABLE WEEKLYSCHEDULE ADD CONSTRAINT FK_WEEKLYSCHEDULE_ROUTE FOREIGN
KEY(ROUTE) REFERENCES ROUTE(ROUTEID);

ALTER TABLE WEEKDAY ADD CONSTRAINT FK_WEEKDAY FOREIGN KEY(YEAR) REFERENCES
YEAR(YEAR);

ALTER TABLE BOOKING ADD CONSTRAINT FK_BOOKING_RESERVATION FOREIGN
KEY(RESERVATIONID) REFERENCES RESERVATION(RESERVATIONID);
ALTER TABLE BOOKING ADD CONSTRAINT FK_BOOKING_PAYINGCARD FOREIGN KEY(PAYINGCARD)
REFERENCES CREDENTIALS(CARDNUMBER);

#############
# Procedures
#############
DELIMITER //

CREATE PROCEDURE addYear(year INTEGER, pricefactor DOUBLE)
                     BEGIN
                                  INSERT INTO YEAR VALUES(year,pricefactor);

                     END //

 CREATE PROCEDURE addDay(year INTEGER, weekday VARCHAR(10), pricefactor DOUBLE)
                     BEGIN
                                  INSERT INTO WEEKDAY
VALUES(weekday,pricefactor,year);
```

```
                    END //

CREATE PROCEDURE addDestination(airportcode VARCHAR(3), name_id VARCHAR(30),
country_id VARCHAR(30))
                    BEGIN
                                INSERT INTO AIRPORT
VALUES(airportcode,name_id,country_id);
                    END //

CREATE PROCEDURE addRoute(departureAirportCode VARCHAR(3), arrivalAirportCode
VARCHAR(3), year INTEGER, routePrice DOUBLE)
                    BEGIN
                                INSERT INTO ROUTE(DEPARTURE, ARRIVAL,
ROUTEPRICE, YEAR) VALUES(departureAirportCode, arrivalAirportCode, routePrice,
year);
                    END //

CREATE PROCEDURE addFlight(IN departureAirportCode VARCHAR(3),IN arrivalAirportCode
VARCHAR(3),IN year INTEGER,IN weekday VARCHAR(10),IN departureTime TIME)
BEGIN
      DECLARE cnt INT DEFAULT 1;
      DECLARE found_id INTEGER;
      SELECT ROUTEID INTO found_id FROM ROUTE AS R WHERE R.DEPARTURE =
departureAirportCode AND R.ARRIVAL = arrivalAirportCode AND R.YEAR = year;
      INSERT INTO WEEKLYSCHEDULE (WEEKDAY, YEAR, DEPARTURETIME, ROUTE) VALUES
(weekday, year, departureTime, found_id);
            WHILE cnt < 53 DO
                  INSERT INTO FLIGHT(WEEK, WEEKLYFLIGHT) VALUES
                        (cnt, (SELECT W.FLIGHTID FROM WEEKLYSCHEDULE AS W WHERE
W.YEAR = year and W.DEPARTURETIME = departureTime AND W.ROUTE IN
                                          (SELECT ROUTEID FROM ROUTE AS R WHERE
R.ARRIVAL = arrivalAirportCode AND R.DEPARTURE = departureAirportCode and R.YEAR =
year)
                                                      AND W.WEEKDAY IN (SELECT
WEEKDAY FROM WEEKDAY WD WHERE WD.WEEKDAY = weekday AND WD.YEAR = year)));

                  SET cnt = cnt + 1;
            END WHILE;
END; //

CREATE PROCEDURE addReservation(IN departure_airportcode VARCHAR(3),IN
arrival_airportcode VARCHAR(3), IN year INTEGER, week INTEGER, IN day VARCHAR(10),

                                                      time TIME, IN
number_passengers INTEGER , OUT output_reservation_nr INTEGER)
BEGIN
      DECLARE flight_number INT DEFAULT NULL;
    DECLARE res_number INT;

    SELECT FLIGHTNUMBER INTO flight_number FROM FLIGHT F WHERE F.WEEK = week AND
F.WEEKLYFLIGHT IN
                                (SELECT FLIGHTID FROM WEEKLYSCHEDULE W WHERE
W.DEPARTURETIME = time AND W.YEAR = year AND W.WEEKDAY = day AND ROUTE IN(
                                                      (SELECT ROUTEID FROM ROUTE R
WHERE R.YEAR = year and R.DEPARTURE = departure_airportcode AND R.ARRIVAL =
arrival_airportcode)));

    #Does the flight exists?
      IF flight_number IS NULL
```

```
      THEN
            SELECT "There exist no flight for the given route, date and time" as
"Message";
      ELSE
            IF number_passengers > calculateFreeSeats(flight_number)
            THEN
                  SELECT "There are not enough seats available on the chosen
flight" as "Message";
            ELSE
                  SET res_number = rand() * 10000;
                  INSERT INTO RESERVATION(RESERVATIONID, FLIGHT, PASSENGERS) VALUES
(res_number, flight_number, number_passengers);
                  SET output_reservation_nr = res_number;
            END IF;
      END IF;
END;
//
CREATE PROCEDURE addPassenger(IN reservation_nr INTEGER, IN passport_number
INTEGER, IN name VARCHAR(30))

BEGIN
      DECLARE not_valid INT default NULL;
    DECLARE already_passenger INT DEFAULT NULL;
    DECLARE already_booked INT DEFAULT NULL;

    SELECT RESERVATIONID INTO not_valid  FROM RESERVATION WHERE RESERVATIONID =
reservation_nr;
    SELECT PASSPORTNR INTO already_passenger FROM PASSENGER WHERE PASSPORTNR =
passport_number;
    SELECT TICKETNR INTO already_booked FROM RESERVED WHERE RESERVATIONID =
reservation_nr LIMIT 1;

    IF not_valid IS NULL
    THEN
            SELECT "The given reservation number does not exist" AS 'MESSAGE';
    ELSE
            IF already_booked IS NULL
        THEN
                  IF already_passenger IS NULL
                  THEN
                        INSERT INTO PASSENGER(PASSPORTNR, NAME) VALUES
(passport_number, name);
                        INSERT INTO RESERVED(PASSPORTNR, RESERVATIONID) VALUES
(passport_number, reservation_nr);
                        SELECT 'Passenger added' AS 'Message';
                  ELSE
                        INSERT INTO RESERVED(PASSPORTNR, RESERVATIONID) VALUES
(passport_number, reservation_nr);
                        SELECT 'Passenger added' AS 'Message';
                  END IF;
            ELSE
                  SELECT "The booking has already been payed and no futher
passengers can be added" AS 'Message';
            END IF;
    END IF;
END;
//
CREATE PROCEDURE addContact(IN reservation_nr INTEGER, IN passport_number INTEGER,
IN email VARCHAR(30), IN phone BIGINT)
```

```
BEGIN
      DECLARE not_valid_res INT DEFAULT NULL;
    DECLARE not_valid_ticket_res INT DEFAULT NULL;
    DECLARE already_contact INT DEFAULT NULL;

    SELECT RESERVATIONID INTO not_valid_res FROM RESERVATION WHERE RESERVATIONID =
reservation_nr;
    SELECT PASSPORTNR INTO not_valid_ticket_res FROM RESERVED WHERE PASSPORTNR =
passport_number AND RESERVATIONID = reservation_nr;
      SELECT PASSPORTNR INTO already_contact FROM CONTACT WHERE PASSPORTNR =
passport_number;

      IF not_valid_res IS NULL
            THEN
                  SELECT "The given reservation number does not exist" AS
'MESSAGE';
            ELSE
                  IF not_valid_ticket_res IS NULL
                        THEN
                              SELECT "The person is not a passenger of the
reservation" AS 'MESSAGE';
                        ELSE
                IF already_contact IS NULL
                              THEN
                                    INSERT INTO
CONTACT(PASSPORTNR,EMAIL,PHONENUMBER) VALUES(passport_number, email, phone);
                                    UPDATE RESERVATION
                                    SET CONTACT = passport_number
                                    WHERE RESERVATIONID = reservation_nr;
                                    SELECT 'Contact added' AS 'MESSAGE';
                              ELSE
                                    UPDATE RESERVATION
                                    SET CONTACT = passport_number
                                    WHERE RESERVATIONID = reservation_nr;
                                    SELECT 'Contact added' AS 'MESSAGE';
                        END IF;
                  END IF;
    END IF;

END;
//

CREATE PROCEDURE addPayment(IN reservation_nr INTEGER, IN cardholder_name
VARCHAR(30), IN credit_card_number BIGINT)

BEGIN
      DECLARE reservation_contact INTEGER DEFAULT NULL;
      DECLARE flight_number INT DEFAULT NULL;
    DECLARE price_total DOUBLE DEFAULT NULL;
    DECLARE number_of_passengers INTEGER;
    DECLARE existing_reservation INTEGER DEFAULT NULL;
    DECLARE already_booked INTEGER DEFAULT NULL;
    DECLARE already_credential DOUBLE DEFAULT NULL;

      SELECT CONTACT INTO reservation_contact FROM RESERVATION WHERE RESERVATIONID
= reservation_nr;
    SELECT FLIGHT INTO flight_number FROM RESERVATION WHERE RESERVATIONID =
reservation_nr;
```

```
    SELECT RESERVATIONID INTO existing_reservation FROM RESERVATION WHERE
RESERVATIONID = reservation_nr;
      SET price_total = calculatePrice(flight_number);
    SELECT COUNT(*) INTO number_of_passengers FROM RESERVED WHERE RESERVATIONID =
reservation_nr;
    SELECT RESERVATIONID INTO already_booked FROM BOOKING WHERE RESERVATIONID =
reservation_nr;
    SELECT CARDNUMBER INTO already_credential FROM CREDENTIALS WHERE NAME =
cardholder_name;

   IF existing_reservation IS NULL
            THEN
        SELECT 'The given reservation number does not exist' AS 'Message';
        ELSE
            IF reservation_contact IS NULL
            THEN
                  SELECT "The reservation has no contact yet" AS 'MESSAGE';
            ELSE
                  #Are there enough seats?
                  IF number_of_passengers > calculateFreeSeats(flight_number)
                  THEN
                        SELECT 'There are not enough seats available on the flight
anymore, deleting reservation' AS 'MESSAGE';
                        DELETE FROM RESERVED WHERE RESERVATIONID = reservation_nr;
                DELETE FROM RESERVATION WHERE RESERVATIONID = reservation_nr;
                  ELSE
                        IF already_booked IS NOT NULL
                THEN
                              SELECT 'The reservation has already been paid' AS
'Message';
                ELSE
                              IF already_credential IS NULL
                              THEN
                                    INSERT INTO CREDENTIALS
VALUES(credit_card_number, cardholder_name);
                         SELECT SLEEP(5);
                                    INSERT INTO BOOKING VALUES (reservation_nr,
price_total, credit_card_number);
                                    UPDATE RESERVED
                                    SET TICKETNR = rand() * 100000
                                    WHERE RESERVATIONID = reservation_nr;

                                    SELECT 'Payment complete' AS 'MESSAGE';
                              ELSE
                                    INSERT INTO BOOKING VALUES (reservation_nr,
price_total, credit_card_number);
                                    UPDATE RESERVED
                                    SET TICKETNR = rand() * 100000
                                    WHERE RESERVATIONID = reservation_nr;

                                    SELECT 'Payment complete' AS 'MESSAGE';
                              END IF;
                        END IF;
                  END IF;
        END IF;
    END IF;
END;
//
```

```
################
# FUNCTIONS
################


CREATE FUNCTION calculateFreeSeats(flightnumber integer)
RETURNS INT
BEGIN
      DECLARE payed_seats INTEGER;
      SELECT COUNT(*) INTO payed_seats FROM RESERVED WHERE TICKETNR IS NOT NULL AND
RESERVATIONID IN (SELECT RESERVATIONID FROM RESERVATION WHERE FLIGHT =
flightnumber);
    RETURN 40 - payed_seats;
END;
//

CREATE FUNCTION calculatePrice(flightnumber integer)
RETURNS DOUBLE
BEGIN
    DECLARE route_price DOUBLE;
    DECLARE weekday_factor DOUBLE;
    DECLARE booked_passengers integer;
    DECLARE profit_factor DOUBLE;
    DECLARE flight_year INTEGER;
    DECLARE flight_day VARCHAR(10);

    SELECT YEAR INTO flight_year FROM WEEKLYSCHEDULE WHERE FLIGHTID = flightnumber;
    SELECT WEEKDAY INTO flight_day FROM WEEKLYSCHEDULE WHERE FLIGHTID =
flightnumber;
      SELECT ROUTEPRICE INTO route_price FROM ROUTE WHERE ROUTEID IN
                        (SELECT ROUTE FROM WEEKLYSCHEDULE WHERE FLIGHTID =
flightnumber);
    SELECT DAYPRICEFACTOR INTO weekday_factor FROM WEEKDAY WHERE WEEKDAY =
flight_day AND YEAR = flight_year;

    SET booked_passengers = 40 - calculateFreeSeats(flightnumber);

    SELECT PRICEFACTOR INTO profit_factor FROM YEAR WHERE YEAR = flight_year;

      #Totalprice = routeprice to/from * weekdayfactorday * (bookedPASSENGERsflight
+1)/40 *profitfactor

      RETURN route_price * weekday_factor * ((booked_passengers  + 1) /40) *
profit_factor;
END;
//

############
# TRIGGER
############

CREATE TRIGGER randomTicketNr before UPDATE ON RESERVED
FOR EACH ROW
BEGIN
      SET new.TICKETNR = FLOOR((rand() * 100000 ));
END;
//
delimiter ;
############
```

```
# VIEWS
############

CREATE VIEW allFlights AS
      SELECT route.DEPARTURE AS 'departure_city_name',
      route.ARRIVAL AS 'destination_city_name',
      ws.DEPARTURETIME AS 'departure_time',
      ws.WEEKDAY AS 'departure_day',
      flight.WEEK AS 'departure_week',
      ws.YEAR AS 'departure_year',
      calculateFreeSeats(flight.FLIGHTNUMBER) AS 'nr_of_free_seats',
      calculatePrice(flight.FLIGHTNUMBER) AS 'current_price_per_seat'
    FROM ROUTE route, WEEKLYSCHEDULE ws, FLIGHT flight
    WHERE route.ROUTEID = ws.ROUTE AND flight.FLIGHTNUMBER = ws.FLIGHTID;
```