

***Target Application 1***

***Team27:***

***Arno Dunstatter***

***Gabriel Cruz***

***Melika Nassizadeh***

***Kevin Hsieh Dec 01, 2021***

table of contents………………………………………………………….…..……...2

overview………………………………………………………………………………3

entity relationship diagram……………………………………………………….....4

create table statements…………………………………………………..……..…..5

***aircraft*** table……………………………………………………………………5

***airport\_cities*** table…………………………………………………………….6

***baggage\_info*** table…………………………………………………………..…7

***boarding***\_passes table………………………………………………………..8

***bookings*** table………………………………………………………………….9

***business\_waitlist*** table………………………………………………………10

***discounts*** table…………………………………….………………………….11

***economy\_waitlist*** table…………………………………….…………….....12

***flights*** table…………………………………….…………………………......13

***passengers*** table…………………………………….………………………...14

***passengers\_bookings***\_table…………………………………….……………15

***tickets*** table…………………………………….……………………………..16

transactions………………………………………………………………………..17

implementation process…………………………………………………………..22

video link/ending statements…………………………………………………..…27

**Target application 1:**

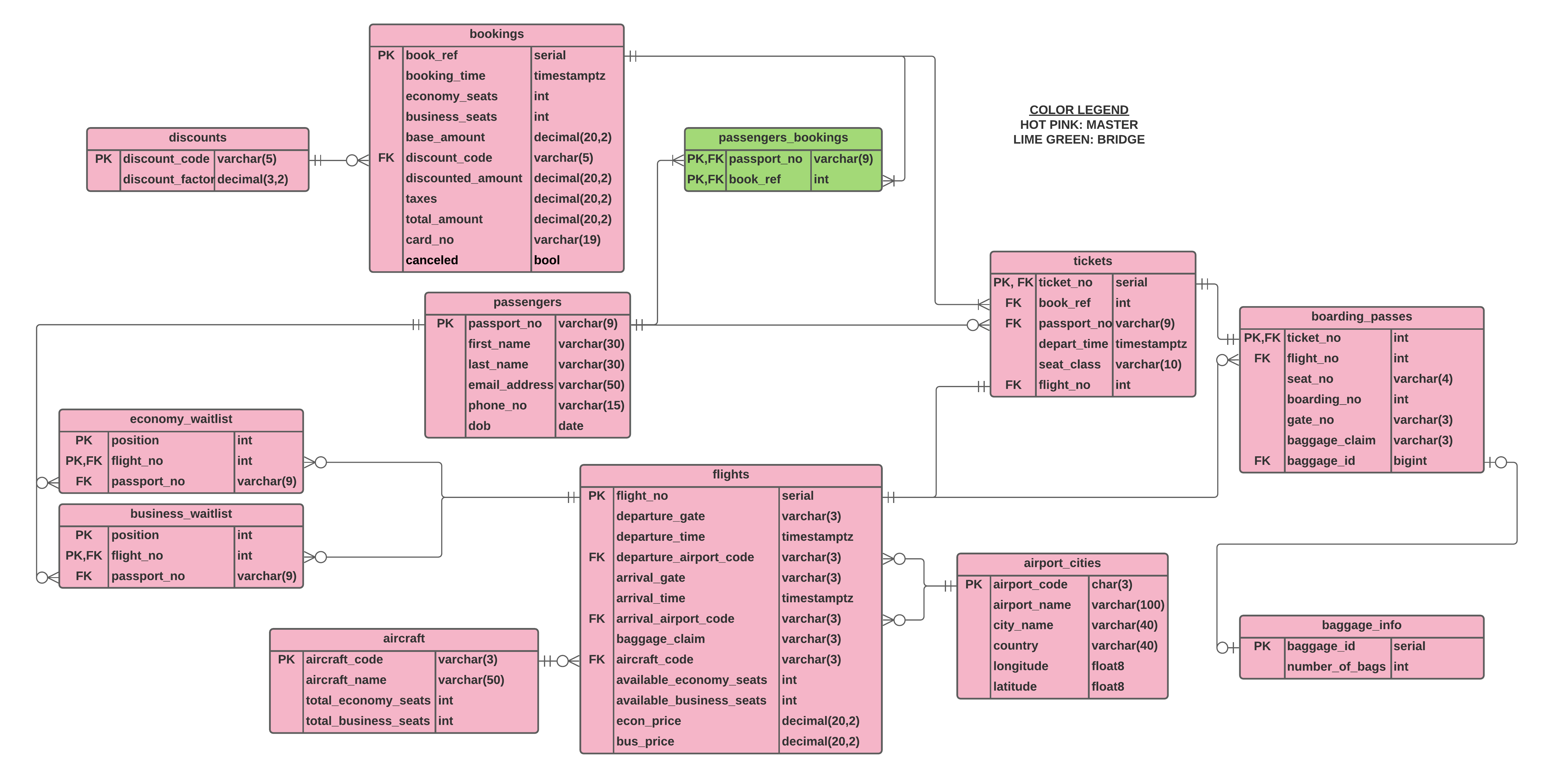
The design focuses on the tables that are necessary to book flight reservations, ticketing, and boarding.

This data includes: Direct or connecting flights up to 1 connection from 2520 flights to choose from, seat class (economy/business) standard personal data, baggage info, payment using credit card number, taxes calculated at 8.25%, discounts can be applied using discount code inputs, base and total amount in dollars, boarding passes with boarding time, departure gate, arrival gate, number of checked bags, actual arrival time, arrival gate, baggage claim carousel number. Cancellations and waitlists have also been implemented. Travel distances between airports have also been calculated based on longitude and latitude.

The design assumes that CheapFlights operates only under one airline. An overview of the database will be presented, followed by the details of how each of the database tables will be created.

Each table will be followed with a table of sample data. To assist in the mission of referential integrity, foreign key constraints were used. More details about the implementation are provided towards the end of the proposal under notes sections. A video link will be provided on the last page with a demo.

This design was targeted for and tested on PostgreSQL using React for the frontend, NodeJS, JS for the backend.



### This table contains aircrafts with identifiers based on IATA(International Air Transport Association) Type codes along with aircraft names with a total of 20 seats per aircraft due to this being a toy airline.

# CREATE TABLE "aircraft"

# (

# "aircraft\_code" VARCHAR(3),

# "aircraft\_name" VARCHAR(50),

# "total\_economy\_seats" INT,

# "total\_business\_seats" INT,

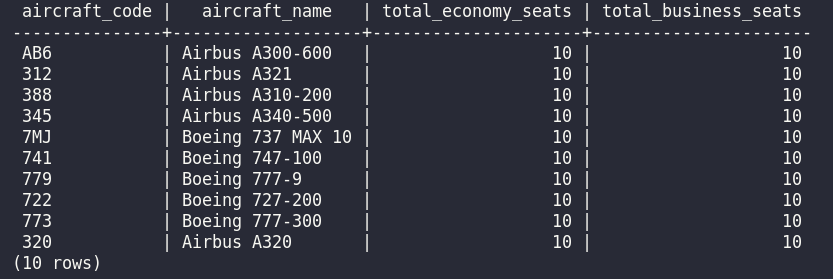
# PRIMARY KEY ("aircraft\_code")

# );

# functional dependencies

### aircraft\_code 🡪 aircraft\_name, total\_economy\_seats, total\_business\_seats

**sample data**



This table contains IATA airport codes and their corresponding airport names along with city,country, longitude, and latitude coordinates that was used to calculate distances between airport to airport using the earthdistances extension.

CREATE TABLE "airport\_cities"

(

"airport\_code" CHAR(3),

"airport\_name" VARCHAR(100),

"city\_name" VARCHAR(40),

"country" VARCHAR(40),

"longitude" FLOAT8,

"latitude" FLOAT8,

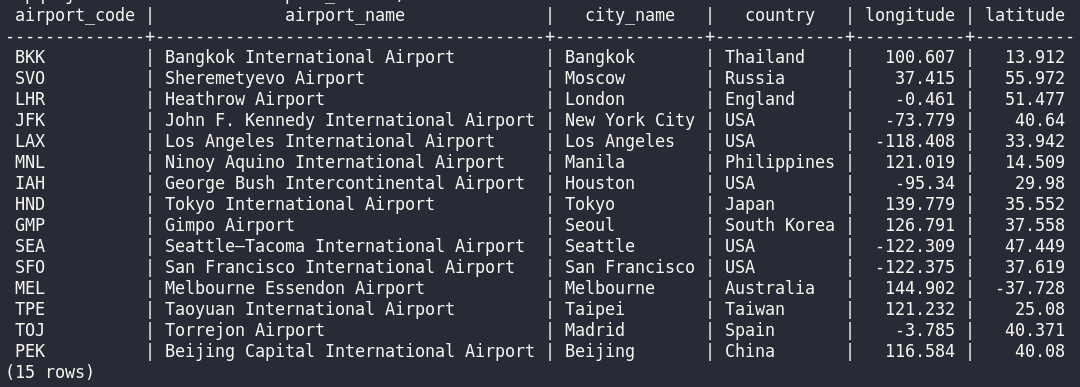
PRIMARY KEY ("airport\_code")

);

# functional dependencies

### airport\_code 🡪 airport\_name, city\_name, country, longitude, latitude

**sample data**



This table contains baggage info that utilizes an autoincremented serial primary key as the baggage\_id and gets number of checked bags from customer when checking-in

CREATE TABLE "baggage\_info"

(

"baggage\_id" SERIAL,

"number\_of\_bags" INT,

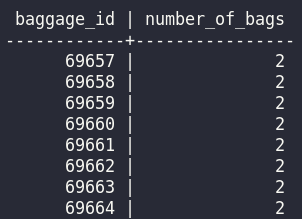
PRIMARY KEY ("baggage\_id")

);

# functional dependencies

### baggage\_id 🡪 number\_of\_bags

**sample data**



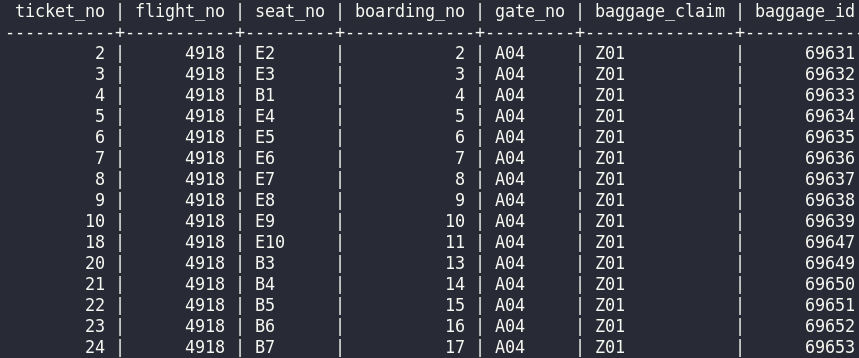
This table contains information that could have been included as part of the tickets table, but we chose to make it it's own table so that it could be populated as passengers check-in for their flights, at which time they have their boarding\_no and seat\_no assigned on a first-come-first-serve basis and also receive their baggage\_id.

# CREATE TABLE "boarding\_passes"   (      "ticket\_no"     *INT*,      "flight\_no"     *INT*,      "seat\_no"       *VARCHAR*(4),      "boarding\_no"   *INT*,      "gate\_no"       *VARCHAR*(3),      "baggage\_claim" *VARCHAR*(3),      "baggage\_id"    *BIGINT*,      PRIMARY KEY ("ticket\_no"),      CONSTRAINT "FK\_boarding\_passes.flight\_no" FOREIGN KEY ("flight\_no")      REFERENCES "flights"("flight\_no")   );

# functional dependencies

### ticket\_no 🡪 flight\_no, seat\_no, boarding\_no, gate\_no, baggage\_claim, baggage\_id

**sample data**



The book\_ref is an artificial pk which is a serial data-type which increments by 1 each time a new booking is added. This table contains information relevant to each particular booking - the time of the booking, the number of seats of seat class, the costs before discounts, after discounts, taxes, and the total costs, as well as the customer's card number and the cancel status which is set by default to false and only updated to true if the whole booking is canceled.

# CREATE TABLE "bookings"

# (

# "book\_ref" SERIAL,

# "booking\_time" TIMESTAMPTZ,

# "economy\_seats" INT,

# "business\_seats" INT,

# "base\_amount" DECIMAL(20, 2),

# "discount\_code" VARCHAR(5),

# "discounted\_amount" DECIMAL(20, 2),

# "taxes" DECIMAL(20, 2),

# "total\_amount" DECIMAL(20, 2),

# "card\_no" VARCHAR(19),

# "canceled" BOOL,

# PRIMARY KEY ("book\_ref"),

# CONSTRAINT "FK\_bookings.discount\_code" FOREIGN KEY ("discount\_code")

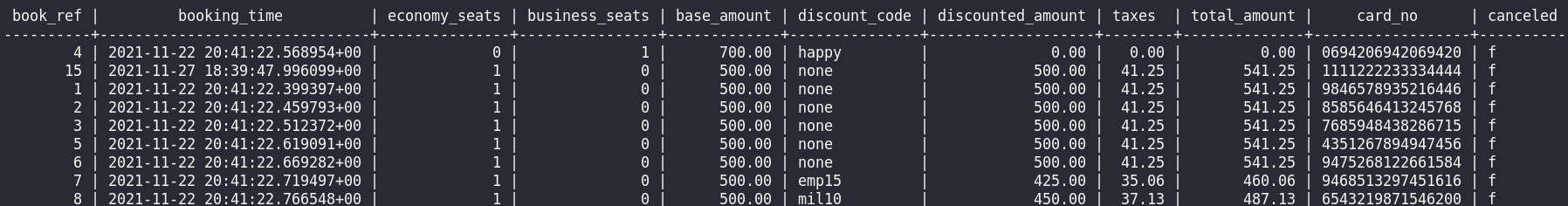
# REFERENCES "discounts"("discount\_code")

# );

# functional dependencies

### book\_ref 🡪 booking\_time, economy\_seats, business\_seats, base\_amount, discount\_code, discounted\_amount, taxes, total\_amount, card\_no, canceled

**sample data**



# All flights have customers' passport numbers who are waiting to change to business class in this table, along with their flight numbers and their position in the waitlist.

# CREATE TABLE "business\_waitlist"

# (

# "position" INT,

# "flight\_no" INT,

# "passport\_no" VARCHAR(9),

# PRIMARY KEY ("position", "flight\_no"),

# CONSTRAINT "FK\_business\_waitlist.passport\_no" FOREIGN KEY ("passport\_no")

# REFERENCES "passengers"("passport\_no"),

# CONSTRAINT "FK\_business\_waitlist.flight\_no" FOREIGN KEY ("flight\_no")

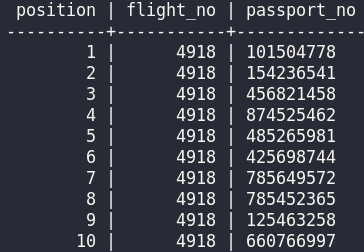
# REFERENCES "flights"("flight\_no")

# );

# functional dependencies

### position 🡪 flight\_no, passport\_no

**sample data**



Contains discount codes and discount factors

CREATE TABLE "discounts"

(

"discount\_code" VARCHAR(5),

"discount\_factor" DECIMAL(3, 2),

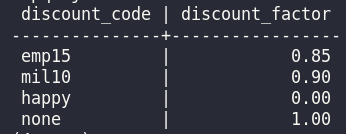
PRIMARY KEY ("discount\_code")

);

# functional dependencies

### discount\_code 🡪 discount\_factor

**sample data**

****

All flights have customers' passport numbers who are waiting to change to economy class in this table, along with their flight numbers and their position in the waitlist.

CREATE TABLE "economy\_waitlist"

(

"position" INT,

"flight\_no" INT,

"passport\_no" VARCHAR(9),

PRIMARY KEY ("position", "flight\_no"),

CONSTRAINT "FK\_economy\_waitlist.passport\_no" FOREIGN KEY ("passport\_no")

REFERENCES "passengers"("passport\_no"),

CONSTRAINT "FK\_economy\_waitlist.flight\_no" FOREIGN KEY ("flight\_no")

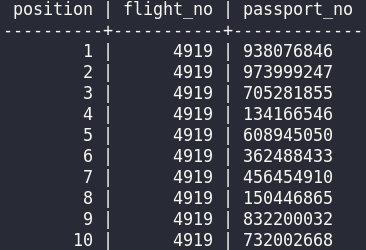
REFERENCES "flights"("flight\_no")

);

# functional dependencies

### position 🡪 flight\_no, passport\_no

**sample data**



### This table contains a complete list of routes that the airline flies. A route is defined as a path with an origin airport and a destination airport. An auto-incremented primary key is used for flights numbers.

# CREATE TABLE "flights"

# (

# "flight\_no" SERIAL,

# "departure\_gate" VARCHAR(3),

# "departure\_time" TIMESTAMPTZ,

# "departure\_airport\_code" VARCHAR(3),

# "arrival\_gate" VARCHAR(3),

# "arrival\_time" TIMESTAMPTZ,

# "arrival\_airport\_code" VARCHAR(3),

# "baggage\_claim" VARCHAR(3),

# "aircraft\_code" VARCHAR(3),

# "available\_economy\_seats" INT,

# "available\_business\_seats" INT,

# "econ\_price" DECIMAL(20, 2),

# "bus\_price" DECIMAL(20, 2),

# PRIMARY KEY ("flight\_no"),

# CONSTRAINT "FK\_flights.arrival\_airport\_code" FOREIGN KEY (

# "arrival\_airport\_code") REFERENCES "airport\_cities"("airport\_code"),

# CONSTRAINT "FK\_flights.aircraft\_code" FOREIGN KEY ("aircraft\_code")

# REFERENCES "aircraft"("aircraft\_code"),

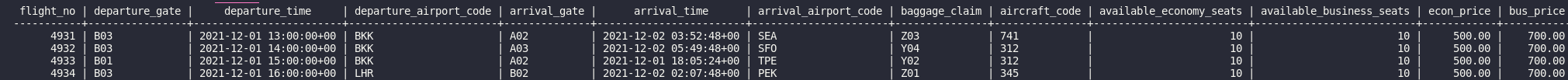
# CONSTRAINT "FK\_flights.departure\_airport\_code" FOREIGN KEY (

# "departure\_airport\_code") REFERENCES "airport\_cities"("airport\_code")

# );

# functional dependencies

### flight\_no 🡪 departure\_gate, departure\_time, departure\_airport\_code, arrival\_gate, arrival\_time, arrival\_airport\_code, baggage\_claim, aircraft\_code, available\_economy\_seats, available\_business\_seats, econ\_price, bus\_price

**sample data** 

This table contains personal information relevant to each passenger.

CREATE TABLE "passengers"

(

"passport\_no" VARCHAR(9),

"first\_name" VARCHAR(30),

"last\_name" VARCHAR(30),

"email\_address" VARCHAR(50),

"phone\_no" VARCHAR(15),

"dob" DATE,

PRIMARY KEY ("passport\_no")

);

# functional dependencies

### passport\_no 🡪 first\_name, last\_name, email\_address, phone\_no, dob

**sample data**



# Since one passenger can have more than one booking, and one booking can have more than one passengers, we used a bridge/join table to avoid having a many-to-many relationship. The many-to-many relationship between the two attributes in this table make it so that this table has no functional dependencies.

# CREATE TABLE "passengers\_bookings" (

# "passport\_no" varchar(9),

# "book\_ref" int,

# PRIMARY KEY ("passport\_no", "book\_ref"),

# CONSTRAINT "FK\_passengers\_bookings.book\_ref"

# FOREIGN KEY ("book\_ref")

# REFERENCES "bookings"("book\_ref"),

# CONSTRAINT "FK\_passengers\_bookings.passport\_no"

# FOREIGN KEY ("passport\_no")

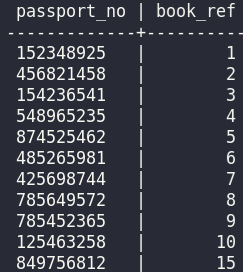
# REFERENCES "passengers"("passport\_no")

# );

# functional dependencies

### None

**sample data**

****

### Contains information about which flight a passenger is on using their ticket number. Displays departure time, seat class, book ref, passport number, and flight number. Ticket\_no is an auto-incremented and generated serial data type whenever a reservation is made.

# CREATE TABLE "tickets" (

# "ticket\_no" serial,

# "book\_ref" int,

# "passport\_no" varchar(9),

# "depart\_time" timestamptz,

# "seat\_class" varchar(10),

# "flight\_no" int,

# PRIMARY KEY ("ticket\_no"),

# CONSTRAINT "FK\_tickets.passport\_no"

# FOREIGN KEY ("passport\_no")

# REFERENCES "passengers"("passport\_no"),

# CONSTRAINT "FK\_tickets.book\_ref"

# FOREIGN KEY ("book\_ref")

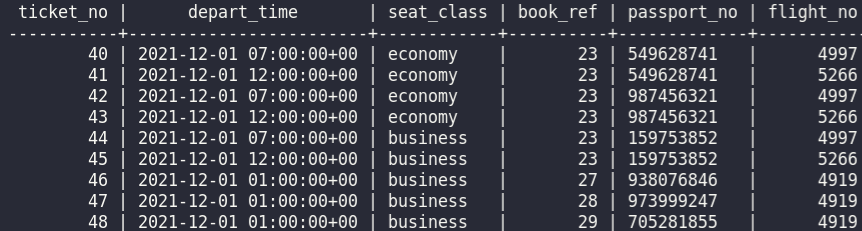
# REFERENCES "bookings"("book\_ref")

# );

# functional dependencies

### ticket\_no 🡪 depart\_time, seat\_class, book\_ref, passport\_no, flight\_no

**sample data**



The following sql statements are part of the transaction for booking a flight where we first narrow the search for available seats in a flight, in this case, customer requests an economy seat so we decrement available economy seats for given flight number. They can also input a discount code, which will be applied if other than default, where default discount is set to ‘none’. It then calculates total amount based on taxes, seat class, and discount. Finally it updates the bookings table with standard customer information and inserts into the passengers table with customer info.

example

BEGIN;

SELECT available\_economy\_seats, available\_business\_seats

FROM flights

WHERE flight\_no = 7437;

UPDATE flights

SET available\_economy\_seats = available\_economy\_seats - 1

WHERE flight\_no = 7437;

INSERT INTO bookings (economy\_seats, business\_seats, discount\_code, card\_no)

VALUES (1,0,'none',1594789622331515);

SELECT MAX(book\_ref) AS mbr

FROM bookings;

SELECT economy\_cost, business\_cost

FROM seat\_class\_costs;

SELECT discount\_factor

FROM discounts

WHERE discount\_code = 'none';

UPDATE bookings

SET base\_amount=500,

discounted\_amount=500,

taxes=41.25,

total\_amount=541.25

WHERE book\_ref = 227;

INSERT INTO passengers

VALUES ('588970267','Shepherd','Aggio','saggio0@edublogs.org','552-967-7436', CAST('1993-04-26' AS DATE))

ON CONFLICT (passport\_no) DO UPDATE

SET first\_name='Shepherd', last\_name='Aggio',email\_address='saggio0@edublogs.org',phone\_no='552-967-7436',dob=CAST('1993- 04-26' AS DATE)

WHERE passengers.passport\_no = '588970267';

INSERT INTO passengers\_bookings

VALUES ('588970267',227);

INSERT INTO tickets (depart\_time, seat\_class, book\_ref, passport\_no, flight\_no)

SELECT departure\_time, 'economy', 227, '588970267', 7437

FROM flights

WHERE flight\_no = 7437;

COMMIT;

### The following sql statements are part of the transaction for cancelling a flight. We first get user input on the flight they wish to cancel from the web app, then we update the booking and set canceled status to true, then the seats from flight number is updated to reflect the cancelled flight. Which in this case, 1 economy seat is added back from flight 7437 where book\_ref = 227.

### example

BEGIN;

UPDATE bookings

SET canceled = 't'

WHERE book\_ref = 227;

SELECT flight\_no

FROM tickets

WHERE book\_ref = 227;

SELECT economy\_seats

FROM bookings

WHERE book\_ref = 227;

SELECT business\_seats

FROM bookings

WHERE book\_ref = 227;

UPDATE flights

SET available\_economy\_seats = available\_economy\_seats + 1

WHERE flight\_no = 7437;

COMMIT;

### The following sql statements are part of the transaction for check-in process to generate boarding passes. User clicks on check-in button on web app, it prompts for number of checked bags user wishes to check-in, then it inserts into the baggage\_info table with an auto-incremented primary key called baggage\_info. It then assigns a seat to the customer according to their seat class, for example, customer booked economy so it will assign E1 if customer is first to book an economy seat for the flight, likewise, it will assign E2 for the next customer to book an economy seat for this flight (B1-B10 for business class). For simplicity, each aircraft only has 10 economy and 10 business seats. It also assigns a baggage\_id for baggage claims with the carousel number (Z02 in this case). They’re also given a departure gate number (A03 in this case).

### example

BEGIN;

INSERT INTO baggage\_info(number\_of\_bags)

VALUES(2);

SELECT baggage\_id

FROM baggage\_info

ORDER BY baggage\_id DESC

LIMIT 1;

SELECT flight\_no

FROM tickets

WHERE ticket\_no = 244;

SELECT seat\_class

FROM tickets

WHERE ticket\_no = 244;

SELECT COUNT(\*)+1

FROM boarding\_passes

LEFT JOIN tickets USING(ticket\_no)

WHERE tickets.flight\_no = 4929

AND seat\_class = 'economy';

SELECT CONCAT('E', 1);

SELECT COUNT(\*)+1

FROM boarding\_passes

WHERE flight\_no = 4929;

INSERT INTO boarding\_passes(ticket\_no, flight\_no, seat\_no, boarding\_no, gate\_no, baggage\_claim, baggage\_id)

VALUES(244, 4929, 'E1',1,'A03','Z02',69681);

COMMIT;

### The following sql statements are part of the transactions for changing the seat class, from economy to business or vice versa, if the seat class is full, it will put customer on a waitlist for the respective seat class they’re looking to switch to. In this example, customer swaps seats from economy to business. For simplicity’s sake, we will be offering free seat upgrades and no refunds for downgrades.

### example

BEGIN;

SELECT seat\_class

FROM tickets

WHERE ticket\_no = 244;

SELECT passport\_no, flight\_no

FROM tickets

WHERE ticket\_no = 244;

SELECT available\_business\_seats

FROM flights

WHERE flight\_no = 4929;

UPDATE flights

SET available\_business\_seats = available\_business\_seats - 1

WHERE flight\_no = 4929;

UPDATE flights

SET available\_economy\_seats = available\_economy\_seats + 1

WHERE flight\_no = 4929;

UPDATE tickets

SET seat\_class = 'business'

WHERE ticket\_no = 244;

SELECT count(\*)

FROM boarding\_passes

WHERE flight\_no = 4929 AND

seat\_no ILIKE 'b%';

UPDATE boarding\_passes

SET seat\_no = 'B1'

WHERE ticket\_no = 244;

COMMIT;

### The following sql transaction puts a customer on to a waitlist for the seat class they wish to change into. If that seat class is full, it adds them to position 1 if no one else is on the waitlist, and increments position by one based on the last waitlist position for that seat class. In this example, customer wants to downgrade and is placed into the economy waitlist for flight 4919 due to all the economy seats being filled with other passengers. It then assigns them position 1.

### example

BEGIN;

SELECT seat\_class

FROM tickets

WHERE ticket\_no = 46;

SELECT passport\_no, flight\_no

FROM tickets

WHERE ticket\_no = 46;

SELECT available\_economy\_seats

FROM flights

WHERE flight\_no = 4919;

SELECT passport\_no

FROM business\_waitlist

WHERE position = 1 AND flight\_no = 4919;

SELECT position

FROM economy\_waitlist

WHERE flight\_no = 4919 AND passport\_no = '938076846';

SELECT MAX(position)

FROM economy\_waitlist

WHERE flight\_no = 4919;

INSERT INTO economy\_waitlist

VALUES (11, 4919, '938076846');

COMMI

* Process:
  + User selects [Book a flight] button
  + Customer inputs their departing city, their arrival city, their date of departure, number of passengers for this booking
  + Based on departing city, arrival city, and flight table we have to determine if their desired start and end location is possible and we have to generate possible options. To do this we need a maximum number of connections. See [here](https://stackoverflow.com/questions/9695016/homework-trying-to-calculate-multiple-flight-stops-between-two-cities). Every connection is a join of flights on itself. Example:
    - First we look for direct flights:
      * Query1: --get user\_departure\_airport\_code
      * SELECT departure\_airport\_code
        + FROM airports
        + WHERE airport\_city = departure\_city\_given;
      * Query2: --get user\_arrival\_airport\_code
        + SELECT arrival\_airport\_code

FROM airports

WHERE airport\_city = arrival\_city\_given;

* + - * Query3: --look for direct flights
      * Query4: if query 3’s results.rows.length == 0 then we need to look for a connecting flight. Join flights on itself using arrival\_airport\_code = departure\_airport\_code
        + need to select both flight\_nos
      * Query5: if query4’s results.rows.length == 0 then we need to look for 2 connections - basically just do another join on query4

***continued***

* + If there are possible options the website displays flight options (flight\_no) and related information (departure\_time, departure\_airport)
  + Customer makes a selection (flight\_no)
  + for each passenger customer selects a seat\_class (associated with cost per ticket) and they’re then assigned a seat\_no
  + For each passenger customer inputs their customer information: passport\_no, first\_name, last\_name, emai\_address, phone\_no, DOB
  + Total pre-discount, pre-tax cost is then displayed
  + “Do you have a discount code?” [yes] [no]
    - if [yes]: customer inputs discount code
    - QUERY: if discount code is found in discount table it’s associated discount is applied to the total order then taxes are applied to give final cost
    - else output “invalid code. Do you have a discount code?” [yes] [no] (recurse)
  + Final cost is displayed for this booking\_no
  + TRANSACTION: To book the flights the customer must input their card\_no and click [book] - this is when the singular transaction is sent to the server to update all relevant tables (passengers, bookings, tickets)
* Checking boarding info:
  + Process:
    - User selects [View a booked flight] button
    - User inputs their email\_address

Website uses their email address to query the server and retrieve all bookings associated with that email (email in passengers is associated with bookings via passengers\_bookings) which are on or after the present timestamptz, these are ordered by departure date for displaying to customer on website

* Boarding/Check-in:
  + Process:
    - User Selects [Check In] button
    - User inputs their email address and the weight of their checked bag
    - TRANSACTION: email\_address in passengers table is linked to tickets via tickets’ passport\_no FK. Select for email\_address given. Select for tickets departing on present timestamptz. Join with flights on flight\_no. Assign boarding\_no for all boarding passes based on first- come-first-serve basis. Assign baggage\_id based on autoincrementation of baggage\_info. A single email\_address can generate multiple entries in boarding\_passes if that passenger has more than one flight that day
* Waitlist for each flights’ seat classes
  + Process:
    - For each seat class we will have a waitlist table with columns position, flight\_no, passport\_no. Position and flight\_no together will be a composite pk.
    - When we have a customer with passport\_no = PPN, on flight\_no = FN, that wants a seat in class = C and the available\_seats for that class on that flight is 0 then we add them to that seat class waitlist
    - When someone on flight\_no = FN cancels their seat in class = C or changes to the other class then we select the passport\_no of the person waiting at position 1 for flight\_no = FN, so they can be awarded a seat in their desired.
    - We then decrement the positions of all entries with flight\_no = FN in the table
* Cancellations
  + Process:
    - User selects [Cancel a booked flight] button
    - User inputs their email\_address
    - Website uses their email address to query the server and retrieve all bookings associated with that email (email in passengers is associated with bookings via passengers\_bookings) which are on or after the present timestamptz, these are ordered by departure date for displaying to customer on website. Each booking will have a [Cancel Booking] button next to it. If the user clicks the button a confirmation window will be displayed “Are you sure you want to cancel your booking for your flight from `departure\_city` to `arrival\_city` on `date`? [Yes, cancel it] [No, never mind]
    - If the user presses [Yes, cancel it] then a transaction is dispatched to the database that changes the bool value of `canceled` in the bookings table to True (it is False by default)

***VIDEO LINK HERE***

***Responsibilities:***

***Arno:***

***Kevin:***

***Gabriel:***

***Melika:***