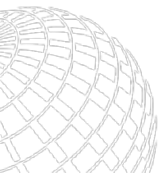
CheapFlights

Target Application 1

Team27:
Arno Dunstatter
Gabriel Cruz
Melika Nassizadeh
Kevin Hsieh

Dec 01, 2021

Table of Contents



Target application 1:

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

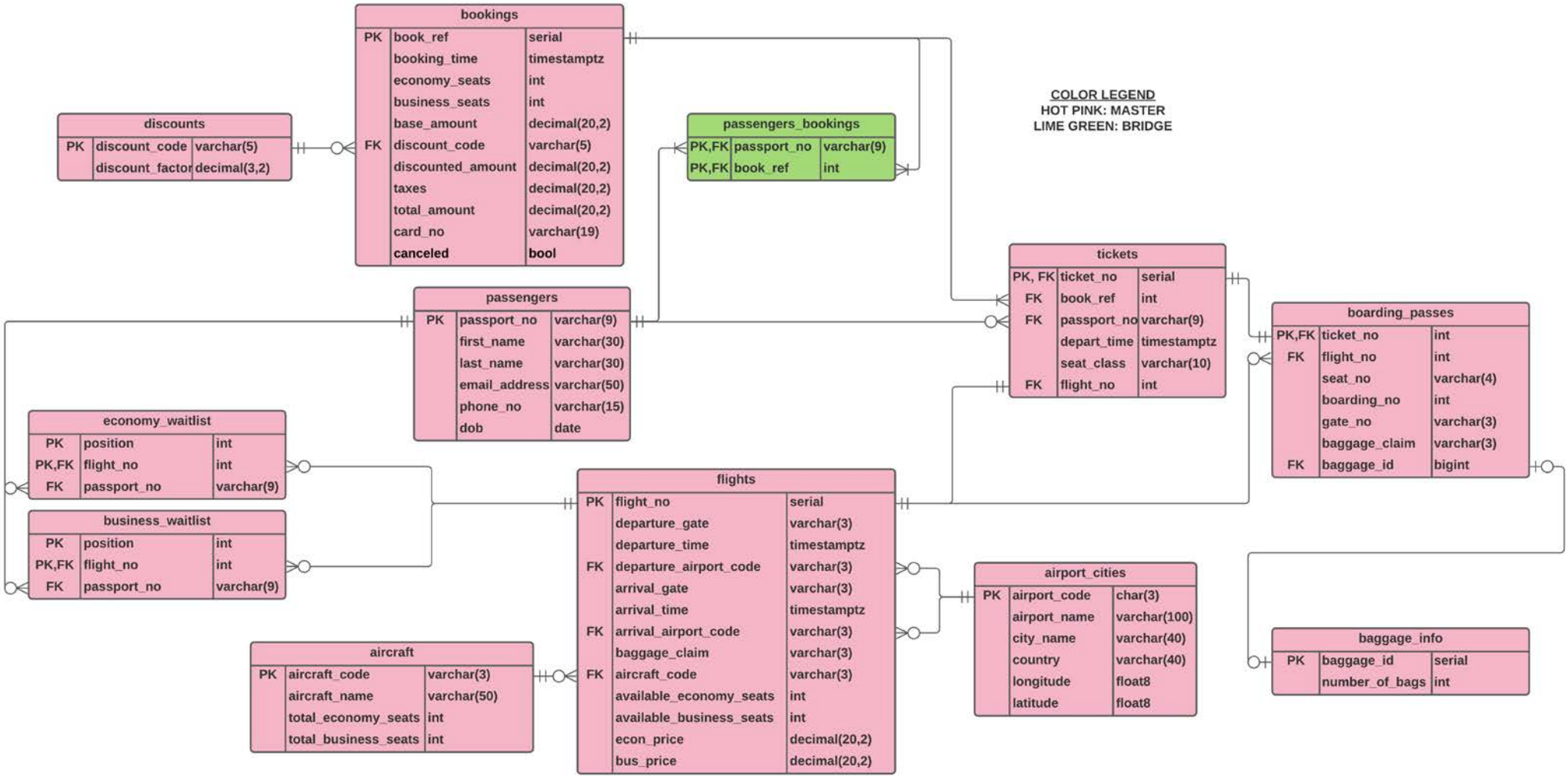
This data includes: Direct or connecting flights up to 1 connection from 2520 flights to choose from(dates only from 2021-12-08 to 2021-12-16), seat class (economy/business) standard personal data, baggage info, payment using credit card number, taxes calculated at 8.25%, discounts can be applied using 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.

Overview



Entity Relationship Diagram

aircraft table

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

aircraft_code	aircraft_name	total_economy_seats	total_business_seats
AB6	Airbus A300-600	10	10
312	Airbus A321	10	10
388	Airbus A310-200	10	10
345	Airbus A340-500	10	10
7MJ	Boeing 737 MAX 10	10	10
741	Boeing 747-100	10	10
779	Boeing 777-9	10	10
722	Boeing 727-200	10	10
773	Boeing 777-300	10	10
320	Airbus A320	10	10

(10 rows)

Create Table Statements

airport_cities table

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

airport_code	airport_name	city_name	country	longitude	latitude
BKK	Bangkok International Airport	Bangkok	Thailand	100.607	13.912
SVO	Sheremetyevo Airport	Moscow	Russia	37.415	55.972
LHR	Heathrow Airport	London	England	-0.461	51.477
JFK	John F. Kennedy International Airport	New York City	USA	-73.779	40.64
LAX	Los Angeles International Airport	Los Angeles	USA	-118.408	33.942
MNL	Ninoy Aquino International Airport	Manila	Philippines	121.019	14.509
IAH	George Bush Intercontinental Airport	Houston	USA	-95.34	29.98
HND	Tokyo International Airport	Tokyo	Japan	139.779	35.552
GMP	Gimpo Airport	Seoul	South Korea	126.791	37.558
SEA	Seattle–Tacoma International Airport	Seattle	USA	-122.309	47.449
SFO	San Francisco International Airport	San Francisco	USA	-122.375	37.619
MEL	Melbourne Essendon Airport	Melbourne	Australia	144.902	-37.728
TPE	Taoyuan International Airport	Taipei	Taiwan	121.232	25.08
TOJ	Torrejon Airport	Madrid	Spain	-3.785	40.371
PEK	Beijing Capital International Airport	Beijing	China	116.584	40.08

(15 rows)

Create Table Statements

baggage_info table

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

baggage_id	number_of_bags
69657	2
69658	2
69659	2
69660	2
69661	2
69662	2
69663	2
69664	2

Create Table Statements

boarding_passes table

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

ticket_no	flight_no	seat_no	boarding_no	gate_no	baggage_claim	baggage_id
2	4918	E2	2	A04	Z01	69631
3	4918	E3	3	A04	Z01	69632
4	4918	B1	4	A04	Z01	69633
5	4918	E4	5	A04	Z01	69634
6	4918	E5	6	A04	Z01	69635
7	4918	E6	7	A04	Z01	69636
8	4918	E7	8	A04	Z01	69637
9	4918	E8	9	A04	Z01	69638
10	4918	E9	10	A04	Z01	69639
18	4918	E10	11	A04	Z01	69647
20	4918	B3	13	A04	Z01	69649
21	4918	B4	14	A04	Z01	69650
22	4918	B5	15	A04	Z01	69651
23	4918	B6	16	A04	Z01	69652
24	4918	B7	17	A04	Z01	69653

Create Table Statements

bookings table

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

book_ref	booking_time	economy_seats	business_seats	base_amount	discount_code	discounted_amount	taxes	total_amount	card_no	canceled
4	2021-11-22 20:41:22.568954+00	0	1	700.00	happy	0.00	0.00	0.00	0694206942069420	f
15	2021-11-27 18:39:47.996099+00	1	0	500.00	none	500.00	41.25	541.25	1111222233334444	f
1	2021-11-22 20:41:22.399397+00	1	0	500.00	none	500.00	41.25	541.25	9846578935216446	f
2	2021-11-22 20:41:22.459793+00	1	0	500.00	none	500.00	41.25	541.25	8585646413245768	f
3	2021-11-22 20:41:22.512372+00	1	0	500.00	none	500.00	41.25	541.25	7685948438286715	f
5	2021-11-22 20:41:22.619091+00	1	0	500.00	none	500.00	41.25	541.25	4351267894947456	f
6	2021-11-22 20:41:22.669282+00	1	0	500.00	none	500.00	41.25	541.25	9475268122661584	f
7	2021-11-22 20:41:22.719497+00	1	0	500.00	emp15	425.00	35.06	460.06	9468513297451616	f
8	2021-11-22 20:41:22.766548+00	1	0	500.00	mil10	450.00	37.13	487.13	6543219871546200	f

Create Table Statements

business_waitlist table

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

position	flight_no	passport_no
1	4918	101504778
2	4918	154236541
3	4918	456821458
4	4918	874525462
5	4918	485265981
6	4918	425698744
7	4918	785649572
8	4918	785452365
9	4918	125463258
10	4918	660766997

Create Table Statements

discounts table

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

discount_code	discount_factor
emp15	0.85
mil10	0.90
happy	0.00
none	1.00

Create Table Statements

economy_waitlist table

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

position	flight_no	passport_no
1	4919	938076846
2	4919	973999247
3	4919	705281855
4	4919	134166546
5	4919	608945050
6	4919	362488433
7	4919	456454910
8	4919	150446865
9	4919	832200032
10	4919	732002668

Create Table Statements

flights table

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

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
4931	B03	2021-12-01 13:00:00+00	BKK	A02	2021-12-02 03:52:48+00	SEA	Z03	741	10	10	500.00	700.00
4932	B03	2021-12-01 14:00:00+00	BKK	A03	2021-12-02 05:49:48+00	SFO	Y04	312	10	10	500.00	700.00
4933	B01	2021-12-01 15:00:00+00	BKK	A02	2021-12-01 18:05:24+00	TPE	Y02	312	10	10	500.00	700.00
4934	B03	2021-12-01 16:00:00+00	LHR	B02	2021-12-02 02:07:48+00	PEK	Z01	345	10	10	500.00	700.00

Create Table Statements

passengers table

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

passport_no	first_name	last_name	email_address	phone_no	dob
152348925	Madeleine	Delgado	dogfighter@uioct.com	530-387-9894	1995-06-23
456821458	Aryaan	Grimes	magic211289@halumail.com	216-695-8282	1991-03-25
154236541	Ibrar	Povey	cadelo@enwsueicn.com	641-636-1290	2001-01-20
548965235	Carlos	Ordonez	la69guy@suttal.com	417-598-2028	1987-01-13
874525462	Rakesh	Verma	kguai@tyonyihi.com	607-377-6090	1996-01-14
485265981	Marsha	Bean	aleandrus@hungclone.xyz	203-506-5852	1988-06-20
425698744	Ernst	Leiss	ki86@omilk.site	262-348-5000	1991-05-30
785649572	Marsha	Vu	guigoloki@77q8m.com	503-499-1868	2002-11-04
785452365	Carlos	Rincon	bsintel@ecallen.com	603-769-3408	1993-02-15
125463258	Garfield	Vu	claudiah@enwsueicn.com	424-500-3443	1992-11-19
849756812	Dave	Davidson	dave_davidson@hotmail.org	1234567890	1986-12-22

Create Table Statements

passengers_bookings table

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

passport_no	book_ref
152348925	1
456821458	2
154236541	3
548965235	4
874525462	5
485265981	6
425698744	7
785649572	8
785452365	9
125463258	10
849756812	15

tickets table

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

ticket_no	depart_time	seat_class	book_ref	passport_no	flight_no
40	2021-12-01 07:00:00+00	economy	23	549628741	4997
41	2021-12-01 12:00:00+00	economy	23	549628741	5266
42	2021-12-01 07:00:00+00	economy	23	987456321	4997
43	2021-12-01 12:00:00+00	economy	23	987456321	5266
44	2021-12-01 07:00:00+00	business	23	159753852	4997
45	2021-12-01 12:00:00+00	business	23	159753852	5266
46	2021-12-01 01:00:00+00	business	27	938076846	4919
47	2021-12-01 01:00:00+00	business	28	973999247	4919
48	2021-12-01 01:00:00+00	business	29	705281855	4919

Create Table Statements

Transaction for making a booking

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;
```

Transaction for cancelling a flight

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;
```

Transaction for check-in process

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;
```

Transaction for changing a seat class or swapping with another passenger

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;
```

Transaction for waitlist if seat class is full

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
```

Implementation process: how customers book their flights

- 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. 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

Implementation Process

Implementation process: how customers book their flights *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)

Implementation process: checking boarding info

- 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

Implementation process: boarding/check-in

- 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

Implementation process: waitlist

- 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

Implementation process: cancellations

- 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)

https://youtu.be/wBvcwuKgGq4

https://github.com/gabrielzurc10/database-proj

Responsibilities:

Arno:

He worked on the ER Diagram, exporting the tables to the database, populating the tables using https://www.mockaroo.com/, writing functions that implemented transactions and queries, and worked generally on maintaining and developing the backend of the project.

Kevin:

Worked on ER diagram, populating the tables, figuring out cardinality between relations, used javascript to create the backend scripts for transactions

Gabriel:

Set up the react front end server and the node server with express.js library. This allows the front end to make API calls from the node server that fetches the SQL queries from the database.

Melika:

Worked on created the query to search for direct, connecting, one-way, and round-trip destination flights based on the user input.

Using react, she implemented a collapsible table to show the result of the query and made unique check boxes that identify the users choice.

While she also used redux, she mainly used <navigate> to transfer important information between the webpages.

After connecting various JavaScript files to fetch information from the database, she was able to pass the information to the transaction query for purchasing and booking a flight.

Video Link & Ending Statements