Ticket booking database:

Brief description and detailed analysis of the database, its tables and views, ER-diagram, writing SQL queries with descriptions.

Brief description of the database

Name	Type	Description
aircrafts	table	Aircraft: models and maximum range in km
airports	table	Airports: codes, names, cities, coordinates, time zones
boarding_passes	table	Boarding passes: pass numbers, ticket numbers, flight ID, seat numbers
bookings	table	Bookings: rooms, booking date, total booking amounts
flights	table	Flights: numbers, scheduled departure and arrival times, actual departure and arrival times, departure and arrival airports, flight statuses, aircraft codes
flights_v	view	Flights: columns of the flights table are supplemented with local and actual time of departure and arrival according to schedule, cities and names of airports of departure and arrival, estimated and actual flight duration
routes	materialized view	Routes: flight numbers, codes and names of airports of departure and arrival, cities of departure and arrival, aircraft codes, flight durations, days of the week of flights
seats	table	Seats on board the aircraft: aircraft codes, seat numbers, class of service
ticket_flights	table	Flights: ticket numbers, flight IDs, classes of service, fares
tickets	table	Tickets: ticket and booking numbers, IDs, names and contact information of passengers

Advanced database analysis (DB)

Booking (bookings) is the main entity of the database. Several passengers can be included in one booking, each of which is issued a separate ticket (tickets). Each passenger is unique, that is, it is not a separate entity, so it is impossible to uniquely find all the tickets of the same passenger. The booking indicates the total amount included in the flight booking for all passengers (total_amount).

The ticket includes one or more flights (ticket_flights). Several flights can be included in one ticket in cases where there is no direct flight connecting the points of departure and destination (flight with transfers), or when the ticket is taken "round trip". It is assumed that all tickets in the same booking have the same set of flights.

Each flight follows from one airport to another. Flights with the same number have the same departure and destination points, but will have different departure dates. There is no separate entity for cities, but the name of the city is indicated and can be used to identify the airports of one city.

When checking in for a flight, which is possible one day before the planned departure date, the passenger is issued a boarding pass (boarding_passes), which indicates the seat number on the plane (seat_no). A passenger can only check in for the flight he has on his ticket (UNIQUE CONSTRAINT). The combination of flight and seat on the aircraft is unique to prevent the issuance of two boarding passes per seat (UNIQUE CONSTRAINT). Boarding passes are assigned consecutive numbers (boarding_no) in the order passengers check in for the flight, and these numbers will be unique only within that flight.

The number of seats in the aircraft and their distribution by class of service depends on the model of the aircraft operating the flight. Each aircraft model has only one cabin layout.

Each flight is uniquely identified by the number and date of departure (UNIQUE CONSTRAINT), and connects two points - airports of departure and arrival. The concept of "flight with transfers" does not exist. If there is no direct flight from one airport to another, the ticket simply includes a few required flights. If the flight is delayed, then the actual time of departure and arrival differs from the planned ones.

Aircrafts table:

Includes information for all aircraft. This table is referenced by the flights table and the seats table by foreign keys aircraft_code.

Columns:

- aircraft_code: aircraft code, IATA - primary natural key;
- model: aircraft model (in Russian, jsonb);
- range: maximum flight range in km.

Airports table:

Includes information for all airports. This table is referenced by the flights table with foreign keys arrival_airport and departure_airport.

Columns:

- airport_code: three-letter airport code - primary natural key;
- airport_name: airport name (in Russian, jsonb);
- city: airport city (in Russian, jsonb);
- longitude: airport coordinates - longitude;
- latitude: airport coordinates - latitude;
- timezone: airport time zone.

Таблица boarding_passes:

Includes boarding pass information and references the ticket_flights table with a composite foreign key ticket_no and flight_id.

Columns:

- ticket_no: ticket number - primary key;
- flight_id: flight ID - primary key;
- boarding_no: boarding pass number;
- seat_no: seat number on the plane.

Bookings table:

Includes booking information. This table is referenced by the tickets table with the booking_ref foreign key.

Columns:

- book_ref: booking number - primary key;
- book_date: booking date;
- total_amount: booking total amount.

Flights table:

Includes information on flights and references the airports table with the arrival_airport and departure_airport foreign keys, and the aircrafts table with the aircraft_code foreign key. This table is referenced by the ticket_flights table with the flight_id foreign key.

Columns:

- flight_id: flight id - primary surrogate key;
- flight_no: flight number;
- scheduled_departure: scheduled departure time;
- scheduled_arrival: scheduled arrival time;
- departure_airport: departure airport;

- arrival_airport: arrival airport;
- status: flight status;
- aircraft_code: aircraft code, IATA;
- actual_departure: actual departure time;
- actual_arraval: actual arrival time.

Seats table:

Includes information on seats in the aircraft cabin. This table references the aircrafts_data table with the aircraft_code foreign key.

Columns:

- aircraft_code: aircraft code, IATA – primary key;
- seat_no: seat number - primary key;
- fare_condition: service class.

Ticket_flights table:

Includes information on flights by combining ticket and flight data, references the tickets table with the ticket_no foreign key and the flights table with the flight_id foreign key. This table is referenced by the boarding_passes table with foreign keys ticket_no and flight_id.

Columns:

- ticket_no: ticket number - primary key;
- flight_id: flight ID - primary key;
- fare_condition: service class;
- amount: flight amount.

Tickets table:

Includes ticket information and references the bookings table with the book_ref foreign key. This table is referenced by the ticket_flights table with the ticket_no foreign key.

Columns:

- ticket_no: ticket number - primary key;
- book_ref: booking number;
- passenger_id: passenger ID;
- passenger_name: passenger's name;
- contact_data: passenger contact details, json.

Flight_v view:

Includes additional information above the flights table (flights), namely, decryption of data about the airports of departure and arrival, local time of departure and arrival, and duration of flights.

Columns:

- flight_id: flight ID;
- flight_no: flight number;
- scheduled_departure: scheduled departure time;
- scheduled_departure_local: scheduled departure time, local time at departure point;
- scheduled_arrival: scheduled arrival time;
- scheduled_arrival_local: scheduled arrival time, local time at arrival point;
- scheduled_duration: scheduled flight duration;
- departure_airport: departure airport code;
- departure_airport_name: name of departure airport;
- departure_city: departure city;
- arrival_airport: arrival airport code;
- arrival_airport_name: name of arrival airport;
- arrival_city: arrival city;
- status: flight status;
- aircraft_code: aircraft code, IATA;
- actual_departure: actual departure time;
- actual_departure_local: actual departure time, local time at departure point;
- actual_arrival: actual arrival time;
- actual_arrival_local: actual arrival time, local time at arrival point;
- actual_duration: actual flight duration.

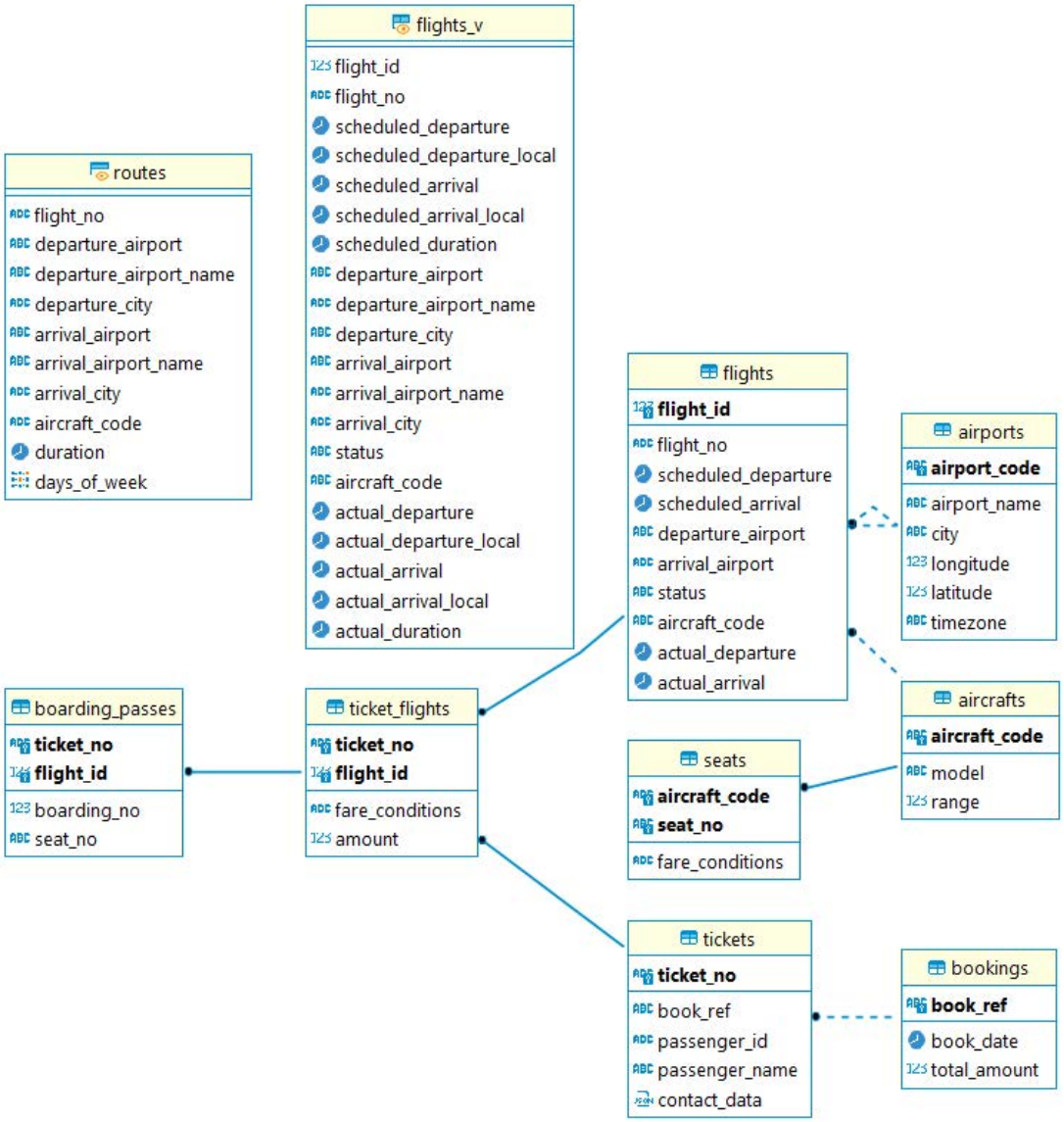
Routes materialized view:

Contains information about flight routes without reference to specific dates.

Columns:

- flight_no: flight number;
- departure_airport:departure airport code;
- departure_airport_name: name of departure airport;
- departure_city: departure city;
- arrival_airport: arrival airport code;
- arrival_airport_name: name of arrival airport;
- arrival_city: arrival city;
- aircraft_code: aircraft code, IATA;
- duration: flight duration;
- days_of_week: days of the week when flights are performed.

ER-diagram



SQL requests and their descriptions

- Display the names of aircraft that have less than 50 seats;
- Display the percentage change in the monthly amount of ticket bookings, rounded to the nearest hundredth;
- Display the names of the aircrafts without Business Class seats. It is necessary to use the array_agg function in the request;
- Display a cumulative total of the number of seats in aircraft for each airport for each day. Take into account only those aircraft that flew empty and only those days when more than one such aircraft took off from the same airport;
- Display the percentage of flights on routes from the total number of flights;
- Display the number of passengers for each code of the mobile operator;
- Classify financial turnover (sum of ticket prices) by routes.

Display the names of aircraft that have less than 50 seats

Code:

```
select model
from(
	select a.model, count(s.seats_no) as seats_number
	from aircrafts a
	join seats s using (aircraft_code)
	group by a.model) t
where seats_number < 50
```

Description:

Join the aircrafts and seats tables by the aircraft_code key, group them by aircraft models, and display only the names of those models that have less than 50 seats.

Display the percentage change in the monthly amount of ticket bookings, rounded to the nearest hundredth

Code:

```
with recursive r as(
	select min(date_trunc('month', book_date)) x from bookings
	union
	select x + interval '1 month' as x
	from r
	where x < (select max(date_trunc('month', book_date)) x from bookings))
select x as "month", sa as sum_amount,
	coalesce(round((coalesce(sa, 0) - lag(coalesce(sa, 0), 1, 0)
over(order by x)) / lag(coalesce(sa, 0), 1, null) over (order by x)*100, 2),0)
as persentage_change
	from r
left join(
	select date_trunc('month', book_date) cm, sum(total_amount) sa
	from bookings
	group by 1) t on t.cm = r.x
order by 1
```

Description:

I create a recursive query in order to sequentially sort through all the months. Then I sequentially display the month, the amount of ticket booking prices grouped by months in the left join block, and the percentage change in the amount of ticket bookings prices for the current month compared to the previous one. Using the round operator, I round the value to the second decimal place. Using the coalesce operator, I print 0 if the value does not exist (for the first month there is no previous one, so for it the percentage change compared to the previous month will be 0). Using the lag operator, I select the previous value that differs from the current one by 1 position.

Display the names of the aircrafts without Business Class seats. It is necessary to use the array_agg function in the request

Code:

```
select model
from(
      select a.model, array_agg(distinct s.fare_conditions)::_text as arrays
      from seats s
      join aircrafts a using (aircraft_code)
      group by a.model) t
where not arrays @> array['Business']
```

Description:

Using the array_agg function, I create an array from the unique values of the fare_conditions column of the seats table (Economy, Business, Comfort), join the aircrafts table by the aircraft_code key. Using the @> operator, I look for the Business value in the arrays column, and using where not operator, I display the names of only those aircraft that do not have a business class. Since the fare_conditions column contains varchar values, I change the data type of the arrays column to _text to use the @> operator.

Display a cumulative total of the number of seats in aircraft for each airport for each day. Take into account only those aircraft that flew empty and only those days when more than one such aircraft took off from the same airport;

Code :

```
select t1.departure_airport, t1.actual_departure::date, t2.seats_count,
sum(t2.seats_count) over (partition by t1.departure_airport,
t1.actual_departure::date order by t1.actual_departure)
from(
      select f.aircraft_code, f.departure_airport, f.actual_departure,
      count(f.flight_id) over (partition by f.departure_airport,
      f.actual_departure::date) as flights_count
      from flights f
      left join boarding_passes bp on f.flight_id = bp.flight_id
      where bp.flight_id is null and f.actual_departure::date is not null) t1
join(
      select a.aircraft_code, count(s.seats_no) as seats_count
      from aircrafts a
      join seats s on a.aircraft_code = s.aircraft_code
      group by a.aircraft_code) t2 on t1.aircraft_code = t2.aircraft_code
where t1.flights_count > 1
```

Description:

1. First, in the from block, I join the boarding_passes table to the flights table using a left join, while using the where statement, I select only those flights that are not in the boarding_passes table, but the flights table has an actual departure date, which indicates that the flight took place. Thus, by using left join, we leave only those flights on which there were no passengers. The key is to check that the flight does not exist in the boarding_passes table, since only the absence of boarding passes for a flight ensures that there are no passengers on it.

Finally, using a window function, I select the number of flights for each airport on each date;

2. In the join block, I add the value of the number of seats for each aircraft model;
3. In the select block, I leave only those values where the number of flights per day for each airport is greater than 1, and using the window function, I display the cumulative sum of seats for each date at each airport.

Display the percentage of flights on routes from the total number of flights

Code:

```
select departure_airport_name, arrival_airport_name,
flights_count/sum(flights_count) over() * 100 as flights_percentage
from(
      select departure_airport_name, arrival_airport_name,
      count(flight_id) as flights_count
      from routes r
      join flights f using (flight_no)
      group by departure_airport_name, arrival_airport_name) t
```

Description:

I join the table with flights to the table with routes, then I count the number of flights grouped by airports of departure and airports of arrival. As a result, I display the airport of departure and the airport of arrival (route) and, using a window function, the percentage of flights on these routes from the total number of flights.

Display the number of passengers for each code of the mobile operator

Code:

```
select substring(contact_data->>'phone' from 3 for 3), count(passenger_id)
from tickets
group by substring(contact_data->>'phone' from 3 for 3)
```

Description:

Working with the json format - using the ->> operator, I access the value of the phone key and using the substring operator, I extract 3 numbers - the operator code. As a result, I display the number of passengers grouped by the code of the mobile operator.

Classify financial turnover (sum of ticket prices) by routes

Code:

```
select
        case
                when sum_routes < 50000000 then 'low'
                when sum_routes >= 150000000 then 'high'
                else 'middle'
        end c,
        count(sum_routes)
from(
        select departure_airport_name, arrival_airport_name, sum(sum_flights) as sum_routes
        from(
                select f.flight_id, sum(tf.amount) as sum_flights
                from flights f
                join ticket_flights tf on f.flight_id = tf.flight_id
                group by f.flight_id) t
        join flights f on f.flight_id = t.flight_id
        join routes r on r.flight_no = f.flight_no
        group by departure_airport_name, arrival_airport_name) t
group by c
```

Description:

1. Inner from block: join tables fligts and ticket_flights and display the sum of the cost of tickets for each flight;
2. Outer from block: join the table obtained from the subquery in the previous block with the flights table, in which we need the flight_no column to join the routes table. Display the sum of the cost of flights grouped by routes;
3. Select block: using a loop, determine 3 classes by the sum of the cost of tickets on routes and display the number of routes distributed among these classes according to the condition.